3D Data Processing

Point Clouds Clustering

Hyoseok Hwang

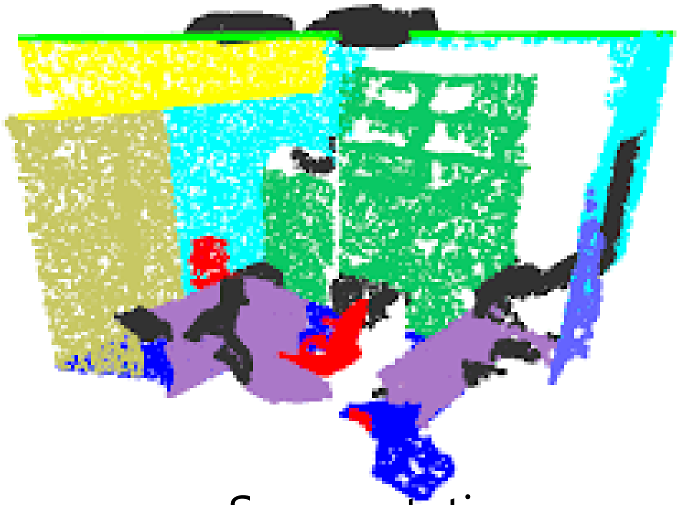
Lectures are based on Open3D functions

Today

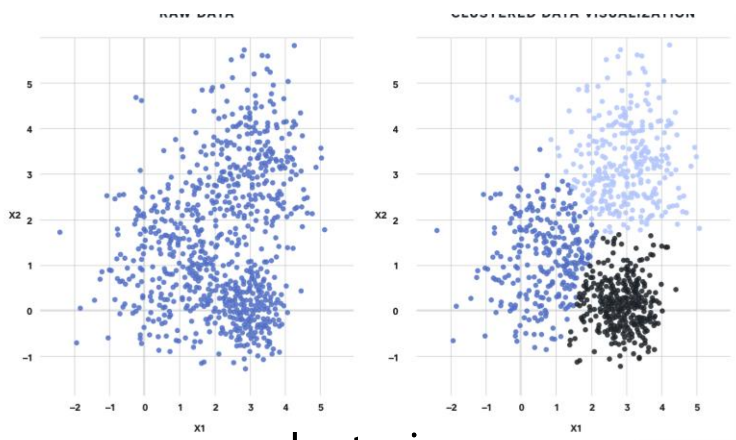
- Clustering (Segmentation)
  - K-means
  - DBSCAN
  - Plane segmentation
  - Planar patch detection
- Transform
  - Translate
  - Rotation
  - Scale

Segmentation & Clustering

- Segmentation
  - Dividing a large group or population into smaller.
  - Make data more homogeneous subgroups based on specific criteria
  - Pixel(Point)-wise classification
- Clustering
  - group together similar objects or data points based on their characteristics or features
  - identify hidden patterns or structures in the data and group
  - Unsupervised method



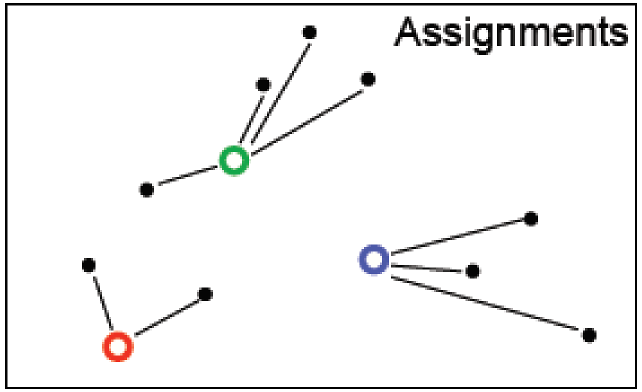
Segmentation

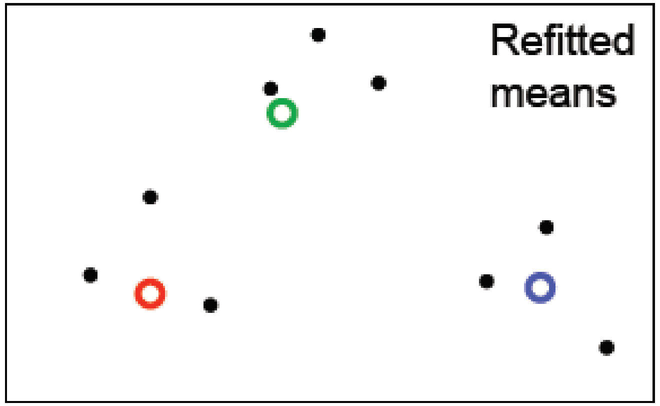


clustering

K-means clustering

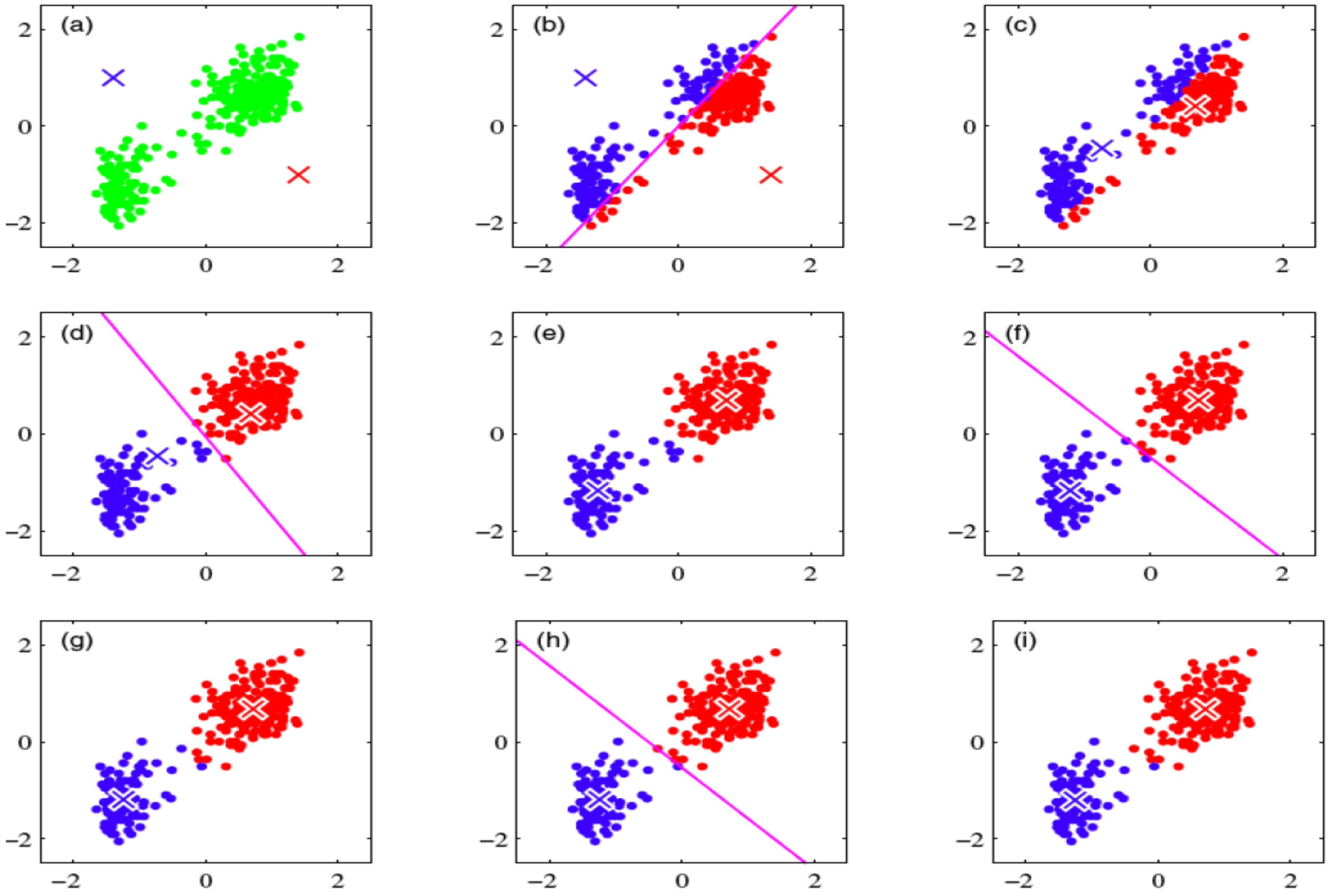
- A distance-based clustering algorithm
- Implementation
  - Initialization: randomly initialize cluster centers
  - The algorithm iteratively alternates between two steps:
    - Assignment step: Assign each data point to the closest cluster
    - Refit step: Move each cluster center to the <u>center of gravity</u> of the data assigned to it





K-means clustering

- An example



http://syskall.com/kmeans.js/

K-means clustering

- What is actually being optimized?

K-means Objective:
Find cluster centers $\mathbf{m}$ and assignments $\mathbf{r}$ to minimize the sum of squared distances of data points $\{\mathbf{x}^{(n)}\}$ to their assigned cluster centers

$$\min_{\{\mathbf{m}\},\{\mathbf{r}\}} J(\{\mathbf{m}\},\{\mathbf{r}\}) = \min_{\{\mathbf{m}\},\{\mathbf{r}\}} \sum_{n=1}^{N}\sum_{k=1}^{K} r_k^{(n)} ||\mathbf{m}_k - \mathbf{x}^{(n)}||^2$$

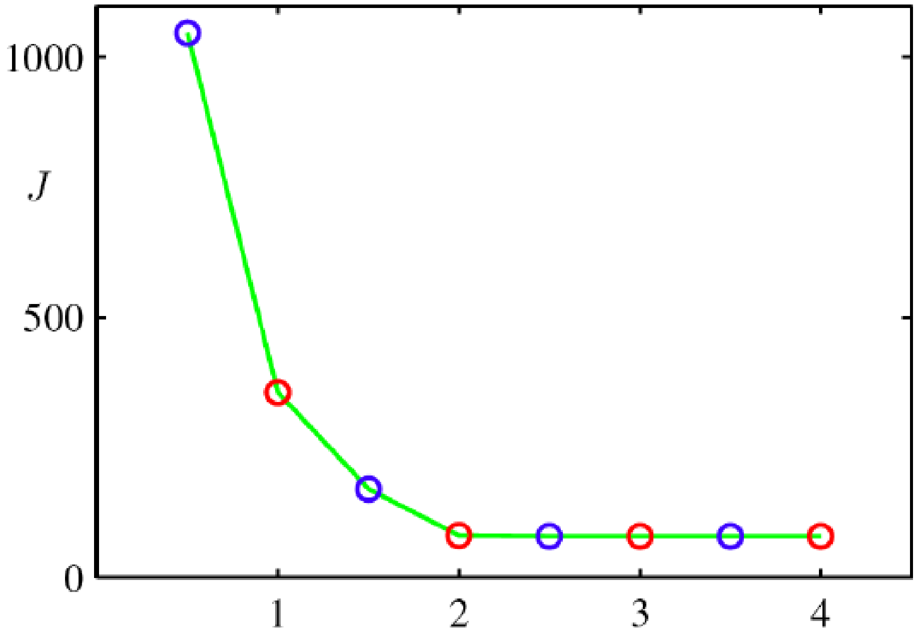
$$\text{s.t.} \sum_k r_k^{(n)} = 1, \forall n, \quad \text{where} \quad r_k^{(n)} \in \{0,1\}, \forall k, n$$

where $r_k^{(n)} = 1$ means that $\mathbf{x}^{(n)}$ is assigned to cluster k (with center $\mathbf{m}_k$)

- Optimization method is a form of coordinate descent ("block coordinate descent")
  - Fix centers, optimize assignments (choose cluster whose mean is closest)
  - Fix assignments, optimize means (average of assigned datapoints)

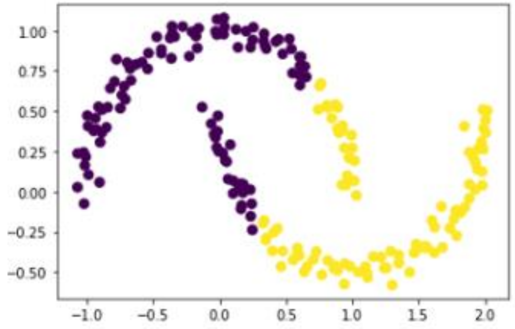
K-means Convergence

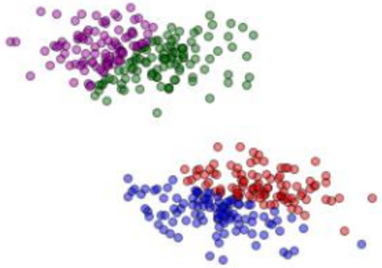
- Whenever an assignment is changed, the sum squared distances J of data points from their assigned cluster centers is reduced
- Whenever a cluster center is moved, J is reduced.
- Test for convergence: If the assignments do not change in the assignment step, we have converged (to at least a local minimum).

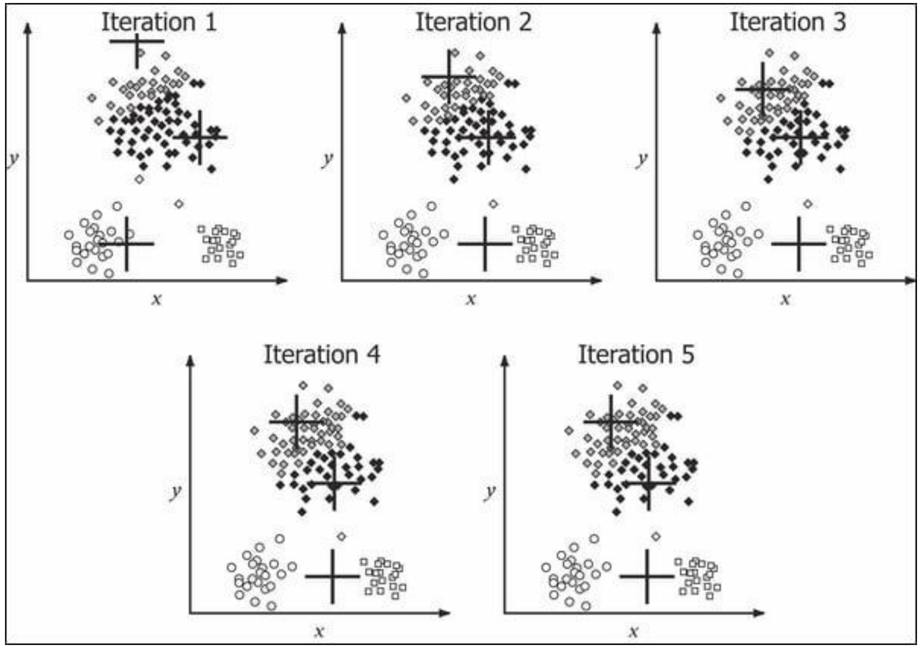


K-means clustering

- Pros
  - Simplicity: K-means clustering is easy to understand and implement.
  - Scalability: K-means clustering is efficient and scalable, making it suitable for large datasets with many variables and observations.
  - Fast convergence: The algorithm usually converges quickly,
- Cons
  - Sensitivity to initial values (Hard to estimate K)
  - Sensitive to outliers
  - Local minima: Only works with convex shapes



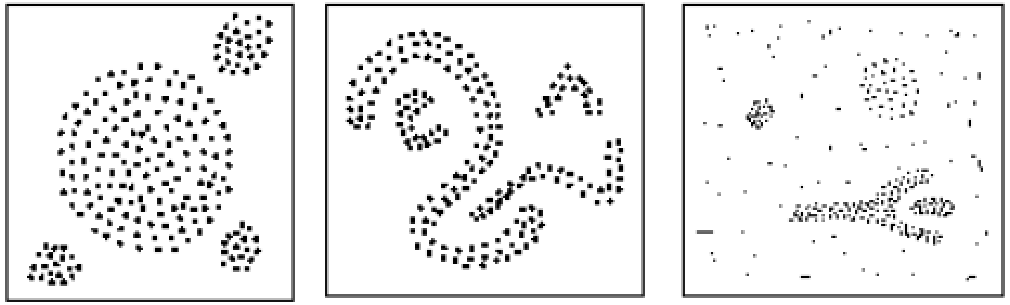




DBSCAN

- Distance-based clustering and its limitations
  - Hard to find clusters with irregular shapes
  - Hard to specify the number of clusters
  - Some points are 'in between' clusters (outliers or background noise)

- Density-based clustering
  - Clustering based on density (local cluster criterion), such as density-connected points
  - Each cluster has a considerable higher density of points than outside of the cluster

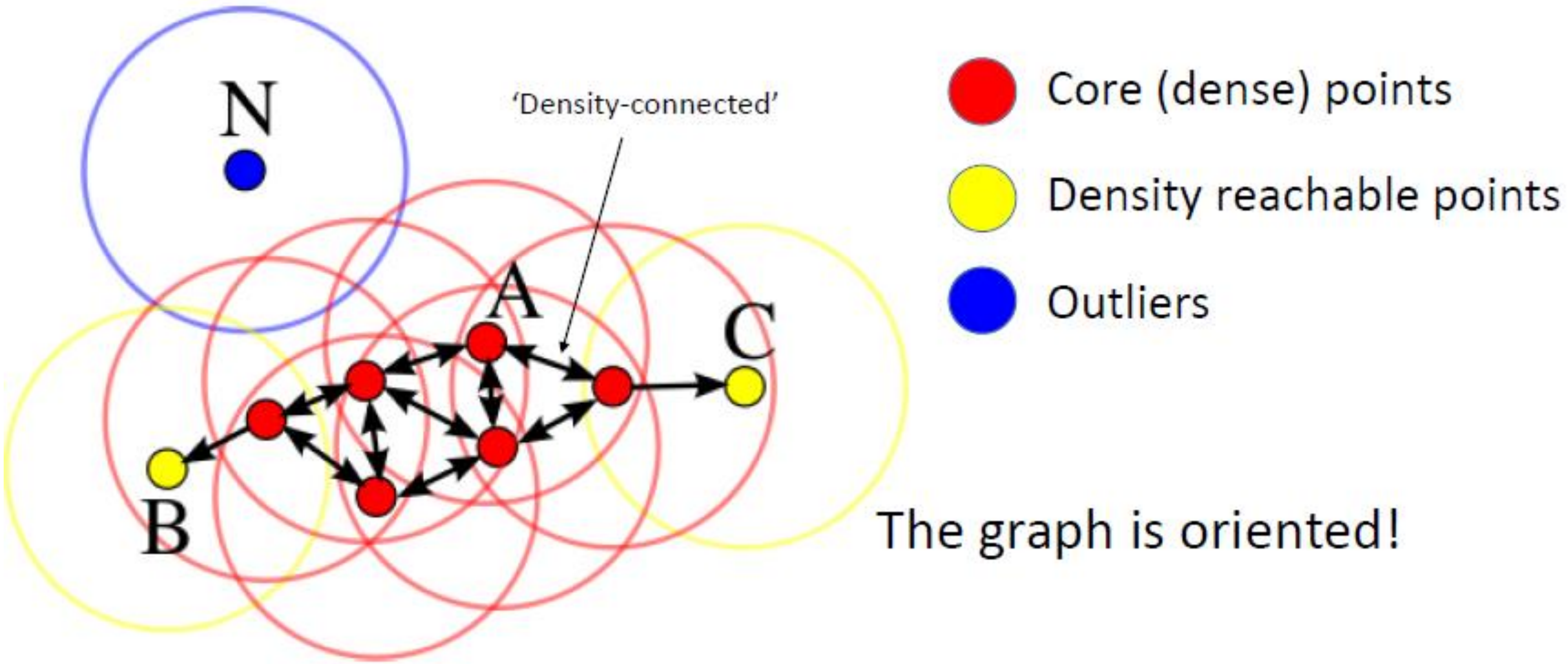


DBSCAN

- **D**ensity-**b**ased **s**patial **c**lustering of **a**pplications with **n**oise
- DBSCAN is a density-based algorithm.
  - Density = number of points within a specified radius r (Eps)
  - A point is a core point if it has more than a specified number of points (MinPts) within Eps
- These are points that are at the interior of a cluster
  - A border point has fewer than MinPts within Eps, but is in the neighborhood of a core point
  - A noise point is any point that is not a core point or a border point.

DBSCAN

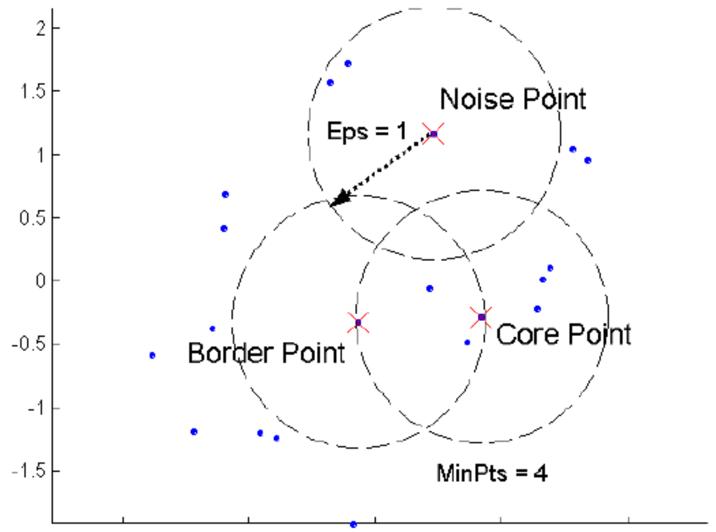
- Core, Border, and Noise points

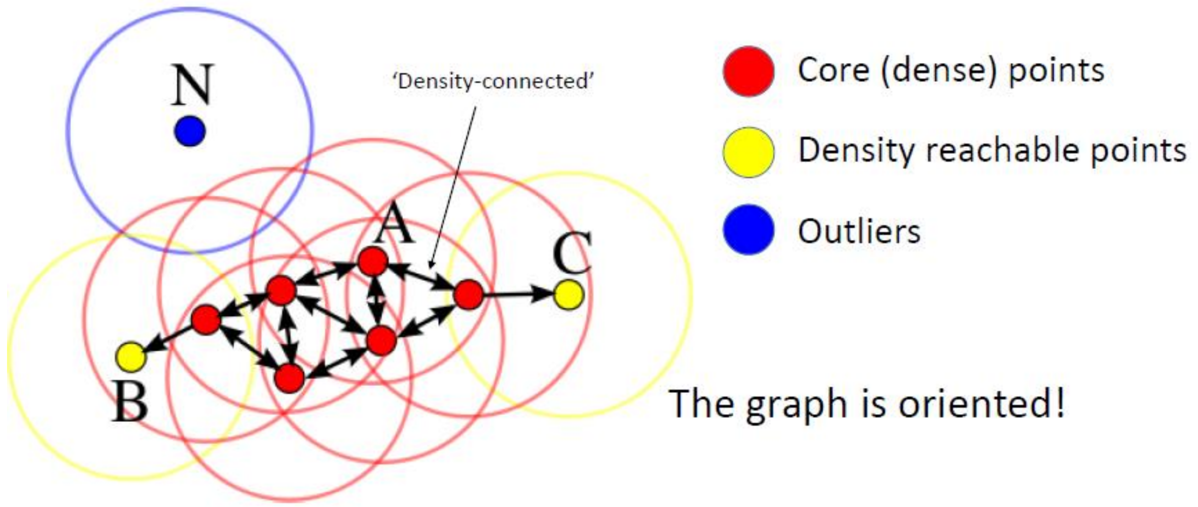


The graph is oriented!

DBSCAN

- A cluster satisfies two properties:
  - All points within the cluster are mutually <u>density-connected</u>
  - If a point is <u>density-reachable</u> from some point of cluster, it is part of the cluster as well

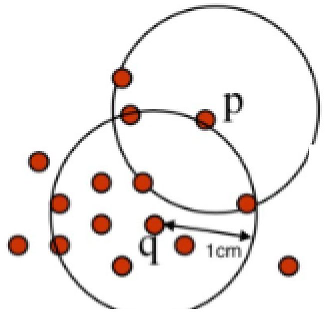




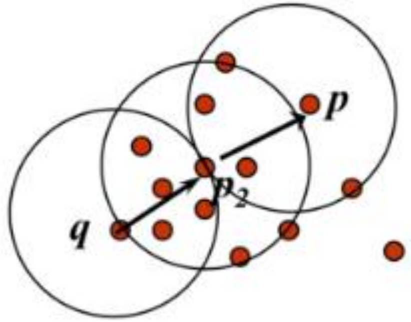
DBSCAN

- Density-Reachable and Density-Connected

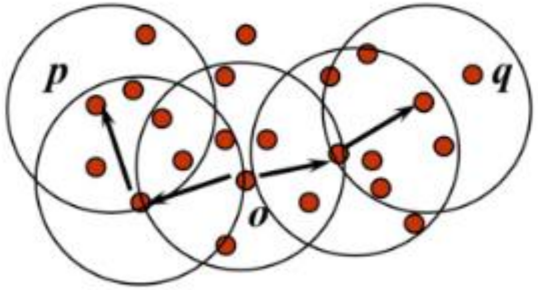
  - Let p be a core point, then every point in its Eps neighborhood is said to be directly density-reachable from p.



  - A point p is density-reachable from a point core point q if there is a chain of points $p_1, \ldots, p_n, p_1 = q, p_n = p$

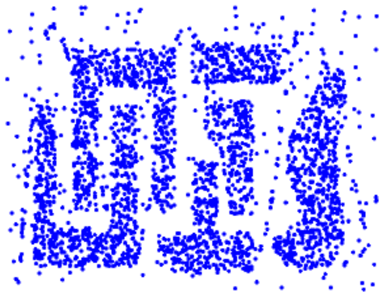


  - A point p is density-connected to a point q if there is a point o such that both, p and q are density-reachable from o

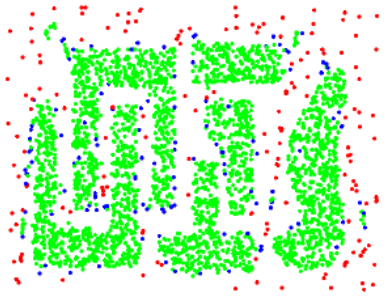


DBSCAN

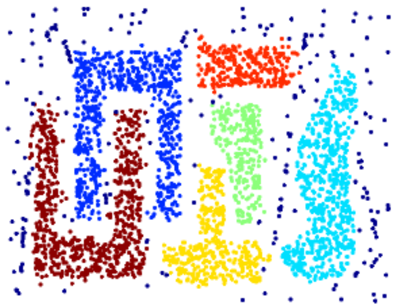
- More information
  - Complexity is O(nlogn)
  - Unlike k-means clustering, deal with the notion of noise
  - Different clusters may have very different densities
  - Very sensitive to the choice of ϵ
  - Concentration of measures will spoil everything in high intrinsic dimensionalities



Original Points



Point types: core, border and noise



Clusters

DBSCAN

- Exercise in Open3D

cluster_dbscan(self, eps, min_points, print_progress=False)

Cluster PointCloud using the DBSCAN algorithm Ester et al., ‘A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise’, 1996. Returns a list of point labels, -1 indicates noise according to the algorithm.

Parameters

- eps (float) – Density parameter that is used to find neighbouring points.
- min_points (int) – Minimum number of points to form a cluster.
- print_progress (bool, optional, default=False) – If true the progress is visualized in the console.

Returns

open3d.utility.IntVector

DBSCAN

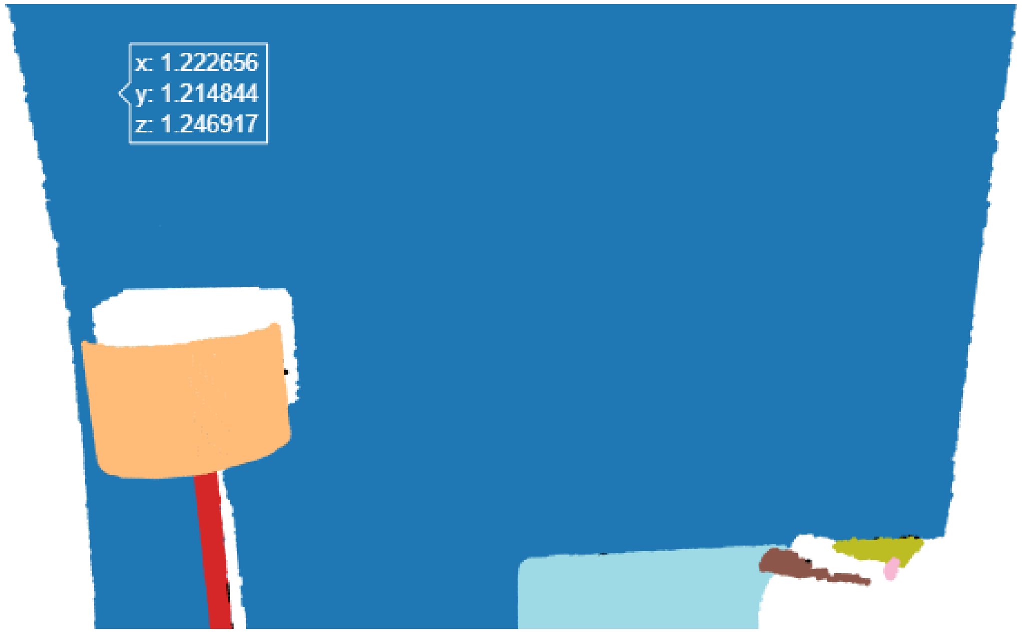
- Exercise in Open3D

```
ply_point_cloud = o3d.data.PCDPointCloud()
pcd = o3d.io.read_point_cloud(ply_point_cloud.path)
o3d.visualization.draw_geometries([pcd])

with o3d.utility.VerbosityContextManager(
        o3d.utility.VerbosityLevel.Debug) as cm:
    labels = np.array(
        pcd.cluster_dbscan(eps=0.02, min_points=10, print_progress=True))

max_label = labels.max()
print(f"point cloud has {max_label + 1} clusters")
colors = plt.get_cmap("tab20")(labels / (max_label if max_label > 0 else 1))
colors[labels < 0] = 0
pcd.colors = o3d.utility.Vector3dVector(colors[:, :3])
o3d.visualization.draw_geometries([pcd])
```

TODO: Change eps and N to 0.3 and 20

DBSCAN

- Exercise in Open3D



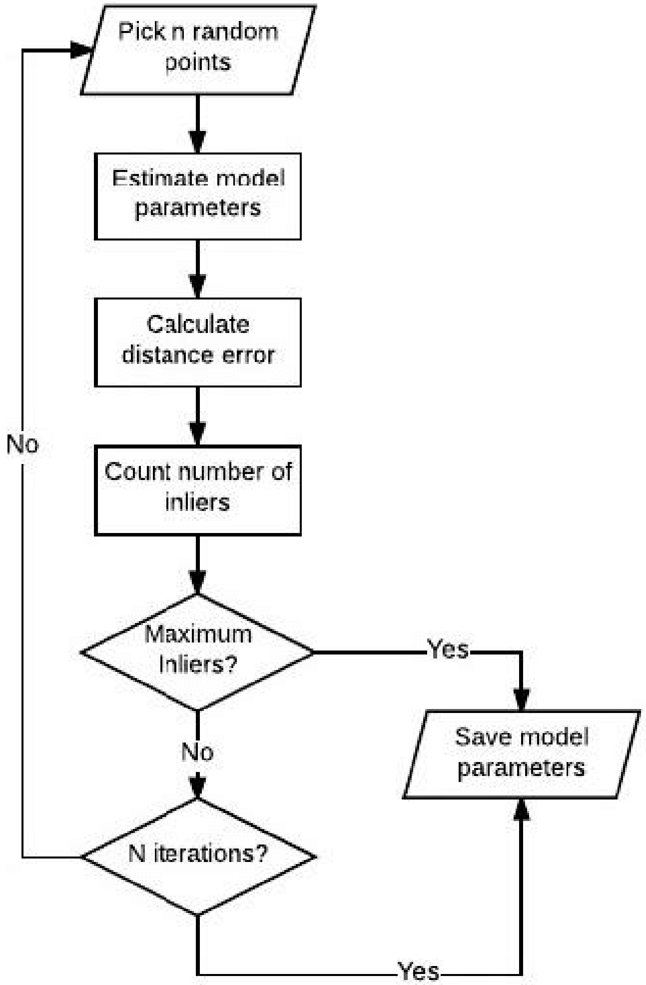
Plane segmentation

- Applying clustering algorithms to common data
- Problems: Most points are connected to the floor
- Solution: For object clustering, find and remove the floor (plane).

DBSCAN result for captured point clouds

Plane segmentation

- Segments a plane in the point cloud using the RANSAC algorithm.
  - Assumption: There is only one floor, and it has the largest area of all the planes.
  - Iterative processing (RANSAC)
    - Select random points
    - Find a plane
    - Find points close to the found plane
  - The plane to find is the one with the most close points.

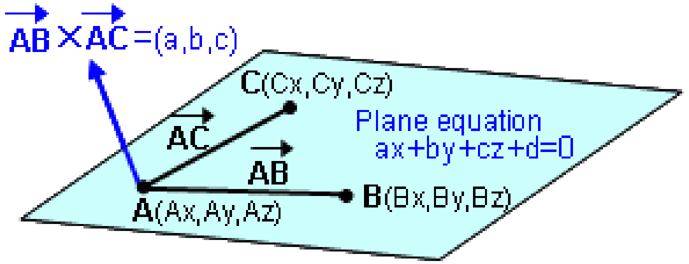


Plane segmentation

- How to get plane parameters from points?

$$ax + by + cz + d = 0$$

- How many points should be selected to estimate a plane?
  - 3 points



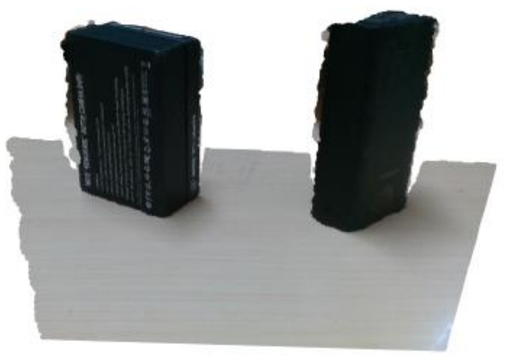
- Distance Between a point and a plane

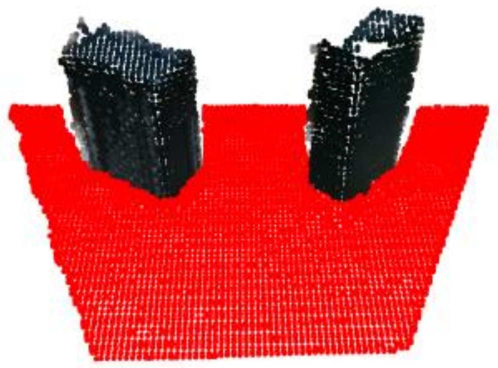
$$d = \frac{|ax + by + cz + d|}{\sqrt{a^2 + b^2 + c^2}}$$

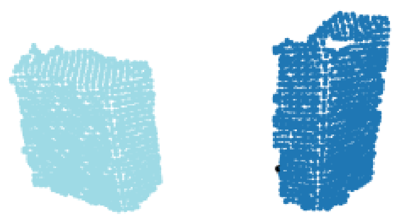
Plane segmentation

- Plane segmentation and clustring

Input point clouds → Plane segmentation → Remove outliers → clustering







Plane segmentation

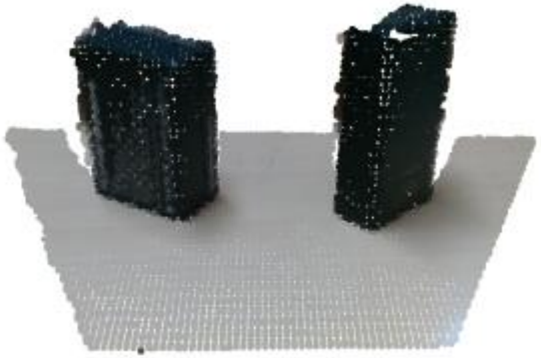
- Practice in Open3D

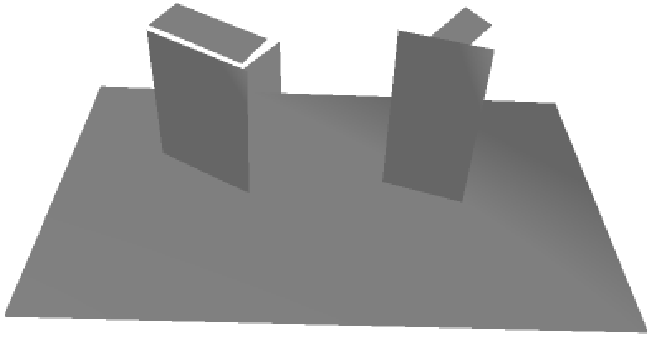
```
pcd = o3d.io.read_point_cloud("./onthedesk.pcd")
pcd_down = pcd.voxel_down_sample(voxel_size=0.005)
o3d.visualization.draw_geometries([pcd_down])

plane_model, inliers = pcd_down.segment_plane(distance_threshold=0.02,
                                              ransac_n=3,
                                              num_iterations=1000)
[a, b, c, d] = plane_model
print(f"Plane equation: {a:.2f}x + {b:.2f}y + {c:.2f}z + {d:.2f} = 0")

inlier_cloud = pcd_down.select_by_index(inliers)
inlier_cloud.paint_uniform_color([1.0, 0, 0])
outlier_cloud = pcd_down.select_by_index(inliers, invert=True)
o3d.visualization.draw_geometries([inlier_cloud, outlier_cloud])

o3d.visualization.draw_geometries([outlier_cloud])
```

Planar patch detection

- Estimate multiple planar patches

```
# using all defaults
oboxes = pcd.detect_planar_patches(
    normal_variance_threshold_deg=60,
    coplanarity_deg=75,
    outlier_ratio=0.75,
    min_plane_edge_length=0,
    min_num_points=0,
    search_param=o3d.geometry.KDTreeSearchParamKNN(knn=30))

print("Detected {} patches".format(len(oboxes)))

geometries = []
for obox in oboxes:
    mesh = o3d.geometry.TriangleMesh.create_from_oriented_bounding_box(obox,
scale=[1, 1, 0.0001])
    mesh.paint_uniform_color(obox.color)
    geometries.append(mesh)
    geometries.append(obox)
# geometries.append(pcd)


o3d.visualization.draw_geometries(geometries)
```





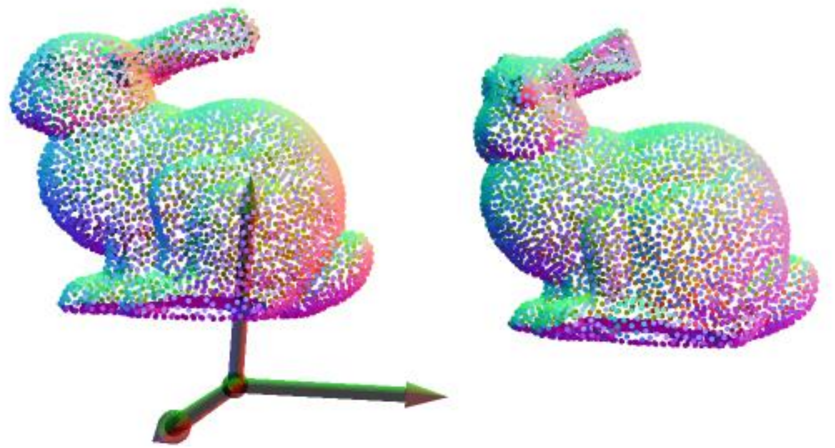
Transform

- Point clouds (geometric) transform
  - Translation
  - Rotation
  - Scaling
  - Linear transform

Transform

- Translate

```
axis = o3d.geometry.TriangleMesh.create_coordinate_frame(size=0.1)
bunny = o3d.data.BunnyMesh()
mesh = o3d.io.read_triangle_mesh(bunny.path)
mesh.compute_vertex_normals()
pcd = mesh.sample_points_poisson_disk(number_of_points=4000)
pcd_translate = copy.deepcopy(pcd).translate((0.2, 0, 0))
o3d.visualization.draw_geometries([axis, pcd, pcd_translate])
```



Transform

- Rotation

```
axis =
o3d.geometry.TriangleMesh.create_coordinate_frame(size=0.1)
bunny = o3d.data.BunnyMesh()
mesh = o3d.io.read_triangle_mesh(bunny.path)
mesh.compute_vertex_normals()
pcd =
mesh.sample_points_poisson_disk(number_of_points=4000)
pcd_rotate = copy.deepcopy(pcd)
pcd_rotate.rotate(mesh.get_rotation_matrix_from_xyz((np.pi /
2, 0, np.pi / 4)), center=(0, 0, 0))
o3d.visualization.draw_geometries([axis, pcd, pcd_rotate])
```

Transform

- Translate and Rotate

```
pcd = mesh.sample_points_poisson_disk(number_of_points=4000)
pcd_rotate = copy.deepcopy(pcd)
pcd_rotate.rotate(mesh.get_rotation_matrix_from_xyz((np.pi / 2, 0, np.pi / 4)),
                center=(0, 0, 0))
o3d.visualization.draw_geometries([axis, pcd, pcd_rotate])
```

- Translate and Rotate by ego-center

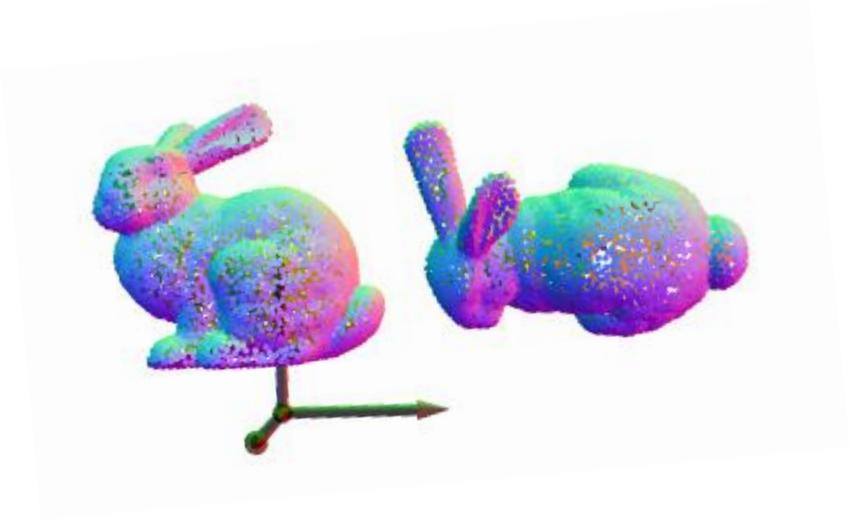
```
pcd = mesh.sample_points_poisson_disk(number_of_points=4000)
pcd_rotate = copy.deepcopy(pcd)
pcd_rotate.rotate(mesh.get_rotation_matrix_from_xyz((np.pi / 2, 0, np.pi / 4)),
                center= pcd_rotate.get_center()))
o3d.visualization.draw_geometries([axis, pcd, pcd_rotate])
```

Transform

- Translate and Rotate

translate

Rotate (0,0,0)



Rotate (object center)

Transform

- Scale

```
axis = o3d.geometry.TriangleMesh.create_coordinate_frame(size=0.1)
bunny = o3d.data.BunnyMesh()
mesh = o3d.io.read_triangle_mesh(bunny.path)
mesh.compute_vertex_normals()
pcd = mesh.sample_points_poisson_disk(number_of_points=4000)
pcd_scale = copy.deepcopy(pcd).translate((0.5, 0, 0))
pcd_scale.scale(2, center=pcd_scale.get_center())
o3d.visualization.draw_geometries([axis, pcd, pcd_scale])
```

Transform

- General Transform

```
axis =
o3d.geometry.TriangleMesh.create_coordinate_frame(size=0.1)
bunny = o3d.data.BunnyMesh()
mesh = o3d.io.read_triangle_mesh(bunny.path)
mesh.compute_vertex_normals()
pcd = mesh.sample_points_poisson_disk(number_of_points=4000)

T = np.eye(4)
T[:3, :3] = mesh.get_rotation_matrix_from_xyz((0, np.pi / 3,
np.pi / 2))
T[0, 3] = 0.2
T[1, 3] = 0.3
print(T)
pcd_T = copy.deepcopy(pcd).transform(T)

o3d.visualization.draw_geometries([axis, pcd, pcd_T])
```

Thank you