ELTE | FACULTY OF INFORMATICS

3D Point Cloud processing and analysis
Course_02
Filtering

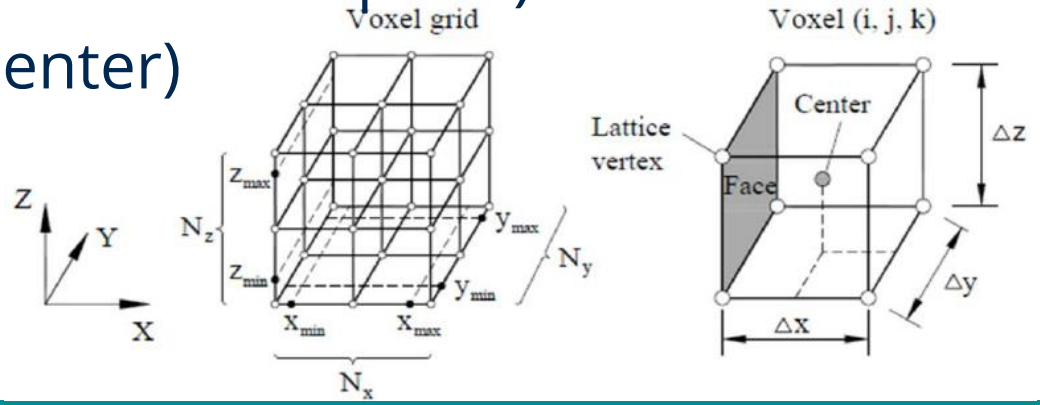
Massinissa Aouragh:
Faculty of Informatics, Department of Artificial Intelligence
Robert Bosch Kft
m2j7au@inf.elte.hu

Filtering

- Voxel grid down-sampling
- Farthest point sampling
- Normal space sampling
- Surface sampling

Voxel Grid Down sampling

- Builds 3D voxel grid on top of the point cloud
- One point is taken from each cell to approximate all the points in the cell
- Determination of the point can be:
  - Random selection
  - Center point (barycenter of the cell may be a not real point)
  - Centroid point (nearest point to the barycenter)



Voxel Grid Down sampling

Given point cloud $p_1, p_2, \ldots, p_n$

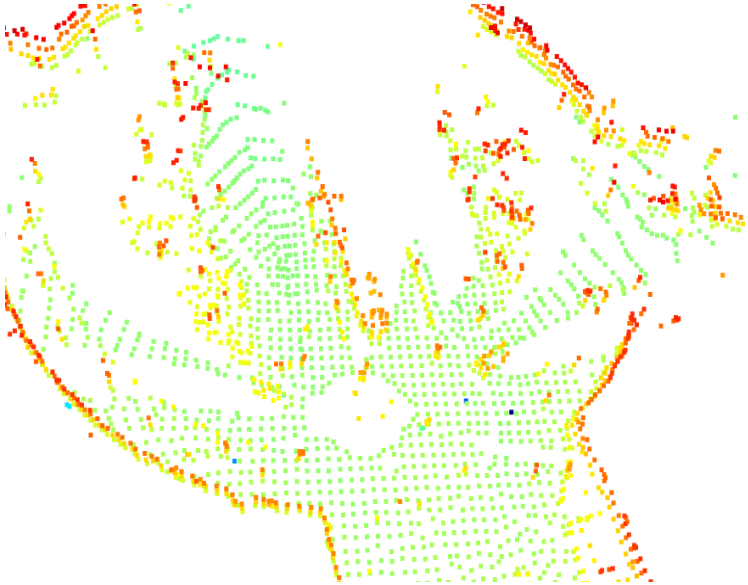
$$x_{min} = min(x_1, x_2, \ldots, x_n)$$
$$x_{max} = max(x_1, x_2, \ldots, x_n)$$
$$y_{min} = min(y_1, y_2, \ldots, x_n)$$
$$y_{max} = max(y_1, y_2, \ldots, x_n)$$
$$z_{min} = min(z_1, z_2, \ldots, x_n)$$
$$z_{max} = max(z_1, z_2, \ldots, x_n)$$

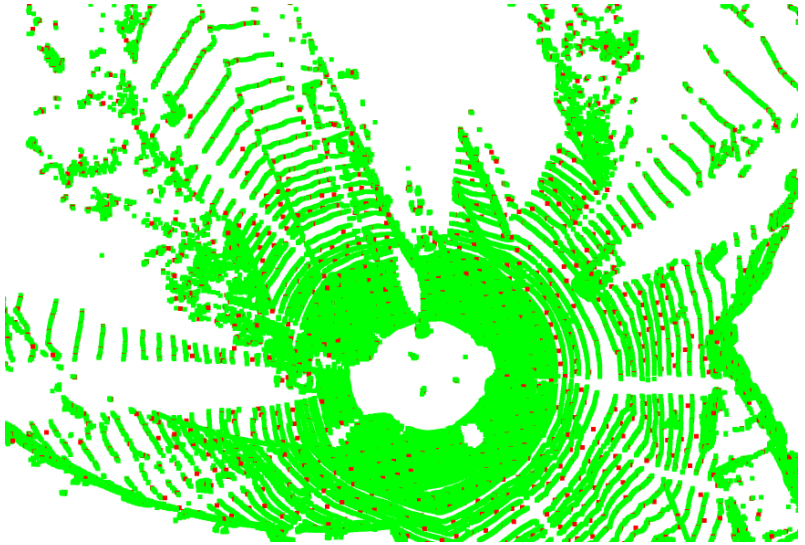
$For\ voxel\ of\ size\ r\ we\ have:$

$$N_x = \lfloor \frac{(x_{max} - x_{min})}{r} \rfloor$$
$$N_y = \lfloor \frac{(y_{max} - y_{min})}{r} \rfloor$$
$$N_z = \lfloor \frac{(z_{max} - z_{min})}{r} \rfloor$$

$Voxel\ Index:$

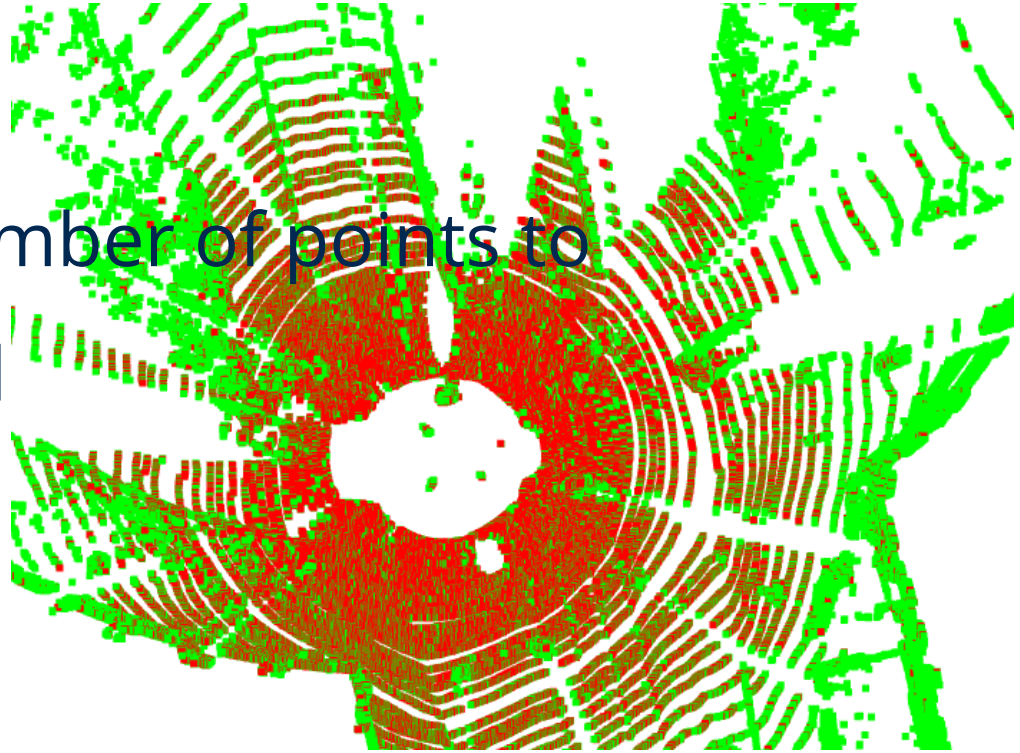
$$i_x = \lfloor \frac{(x - x_{min})}{r} \rfloor$$
$$i_y = \lfloor \frac{(y - y_{min})}{r} \rfloor$$
$$i_z = \lfloor \frac{(z - z_{min})}{r} \rfloor$$
$$i = i_x + i_y * N_x + i_z * N_x * N_y$$





Farthest Point Sampling FPS

- Iterative algorithm that assigns high possibilities to points with longer distances
- Number of iterations defines the number of points to sample from the original point cloud



Farthest Point Sampling FPS

Input: Point cloud P of shape (N, D) denoted by number of point N and point cloud dimension D (D=3), and number of points to be sampled n_point.

Distance[N] = {inf}

Sampled[n_point] = {-1}

sample_idx = randint(0, N)

For j = 0 to n_point - 1 do

    for i = 0 to N - 1 do

        dist = geo_dist(P[i,:], P[sample_idx, :])

        if dist ≤ **Distance**[i] then

            Distance[i] = dist

        end

    end

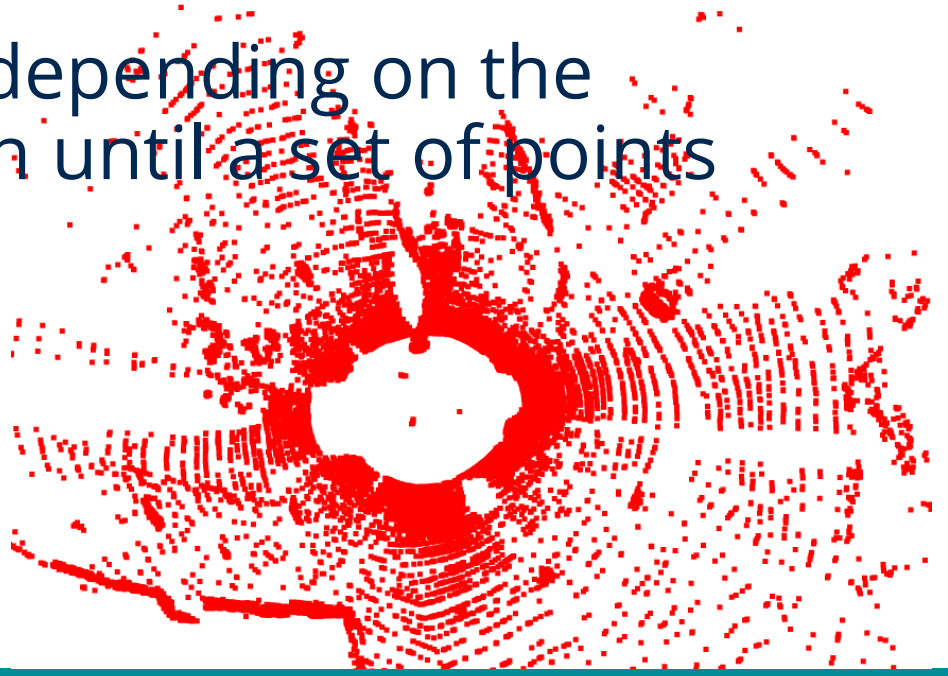
    sample_idx = argmax**(Distance)**

    Sampled[j] = sample_idx

end

Normal Space Sampling NSS

- Point selection based on the surface normal
- First needs a bucket construction of points in the normal space
- Points are selected from all the points depending on the selection criteria (e.g., uniform selection until a set of points is reached)

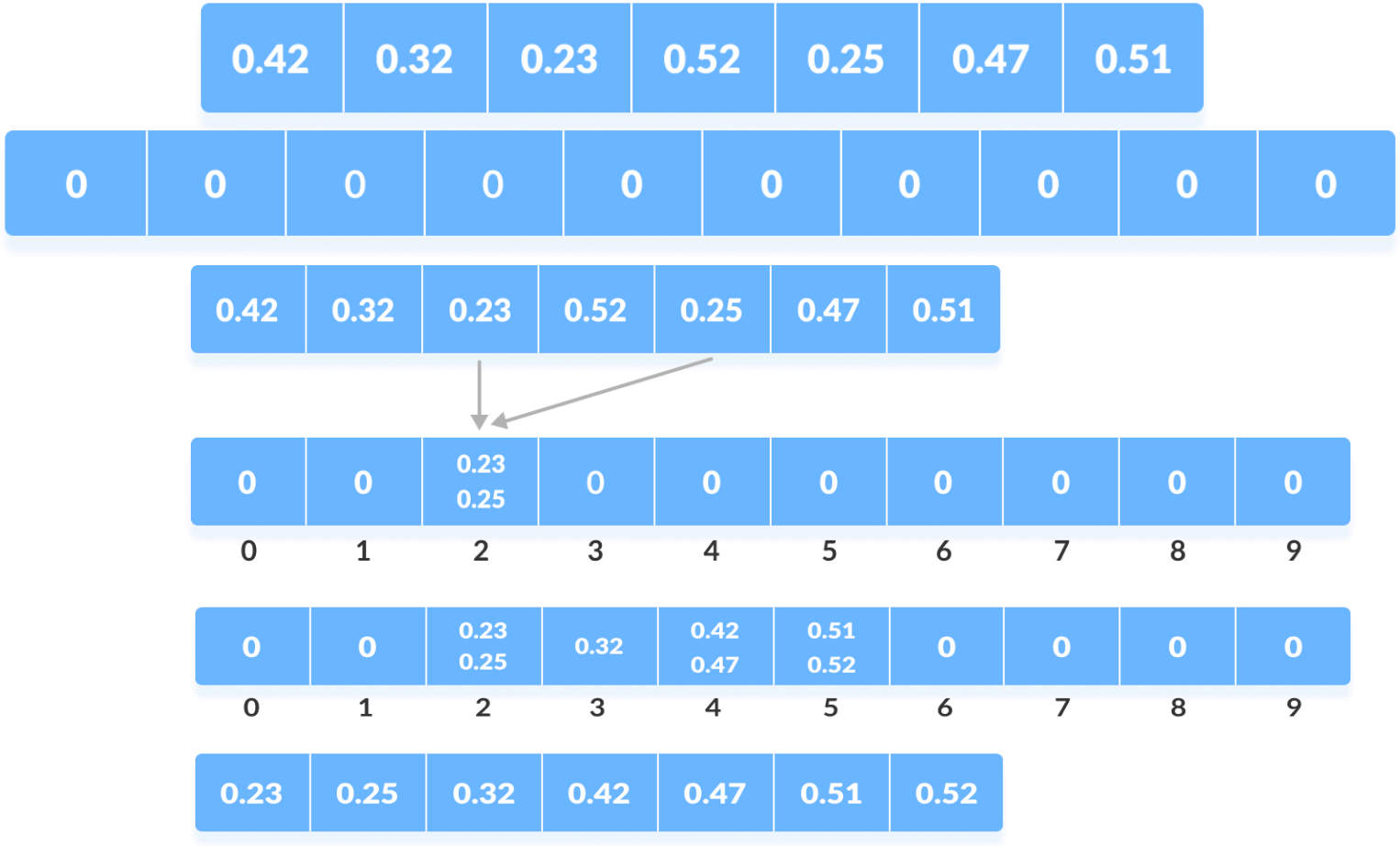


Normal Space Sampling NSS

- Normal Estimation:
  Vector perpendicular to the tangent plane of the surface at a given point. We can apply PCA on a neighborhood points to obtain eigenvalues $\lambda_1, \lambda_2 and\ \lambda_3 where\ \lambda_1 \geq \lambda_2 \geq \lambda_3$ and the corresponding eigenvectors $v_1, v_2\ and\ v_3$. The last significant component $v_3$is the surface normal n.
- Bucket Sorting (Scatter-Gather):
  Sorting algorithm that divides the unsorted array elements into several groups called buckets. Each bucket is then sorted by any suitable sorting algo.

Normal Space Sampling NSS (Bucket sorting)

Normal Space Sampling NSS (Bucket sorting)



Bucket sorting with Normals

Input: array of point normals (N, D=3)
Bucket: list
Number_of_normals = len(normals)
For i = 0 to Number_of_normals do
    Bucket.pushback(list())
For i = to Number_of_normals do
    px = abs(normals[i][0]), py = abs(normals[i][1]), pz = abs(normals[i][2])
    r = $\sqrt{px^2 + py^2 + pz^2}$
    px /= px / r, py /= py / r, pz /= pz / r
    j = int(Number_of_normals * $\frac{px+py+pz}{3}$)
    bucket[j].pushback(i)

Point Sampling from bucket

Input: array of point normals (N, D=3)

Bucket_normal: sort_into_buckets(normals)

Valid_buckets: list

For i = 0 to Bucket_normal.size do

    if Bucket_normal[i] is not empty then

        Valid_buckets[i] = Bucket_normal[i]

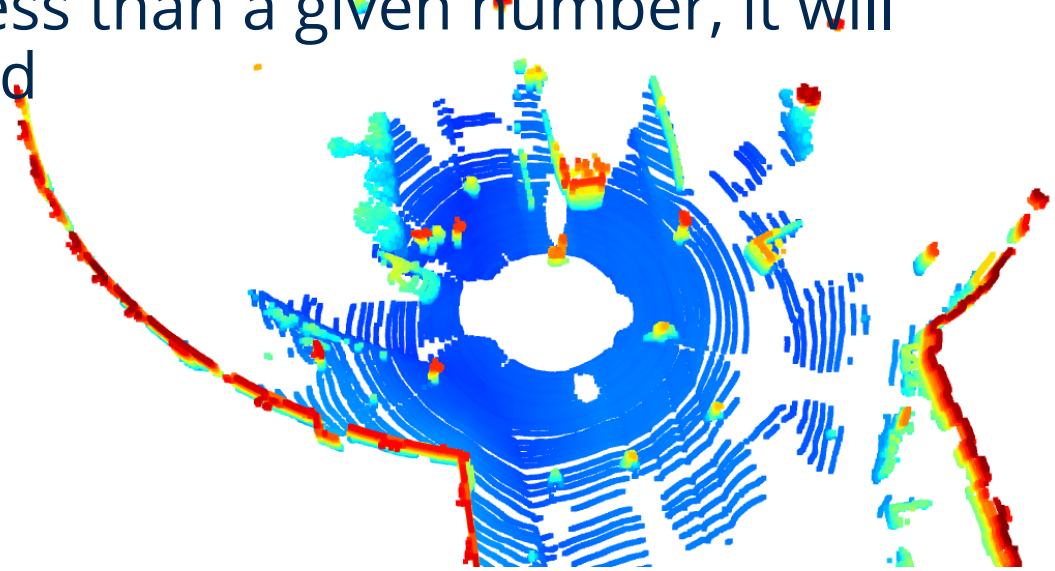
Sample from Valid_buckets (e.g., take from each bucket one random value)

Noise Removal

- Radius Outlier Removal:
  - Filters points based on the number of neighbors in a certain radius.
  - For each point we first find it neighbors with a certain radius
  - If the number of the neighbors k is less than a given number, it will be considered as outlier and removed

r = 1m

k_min = 100



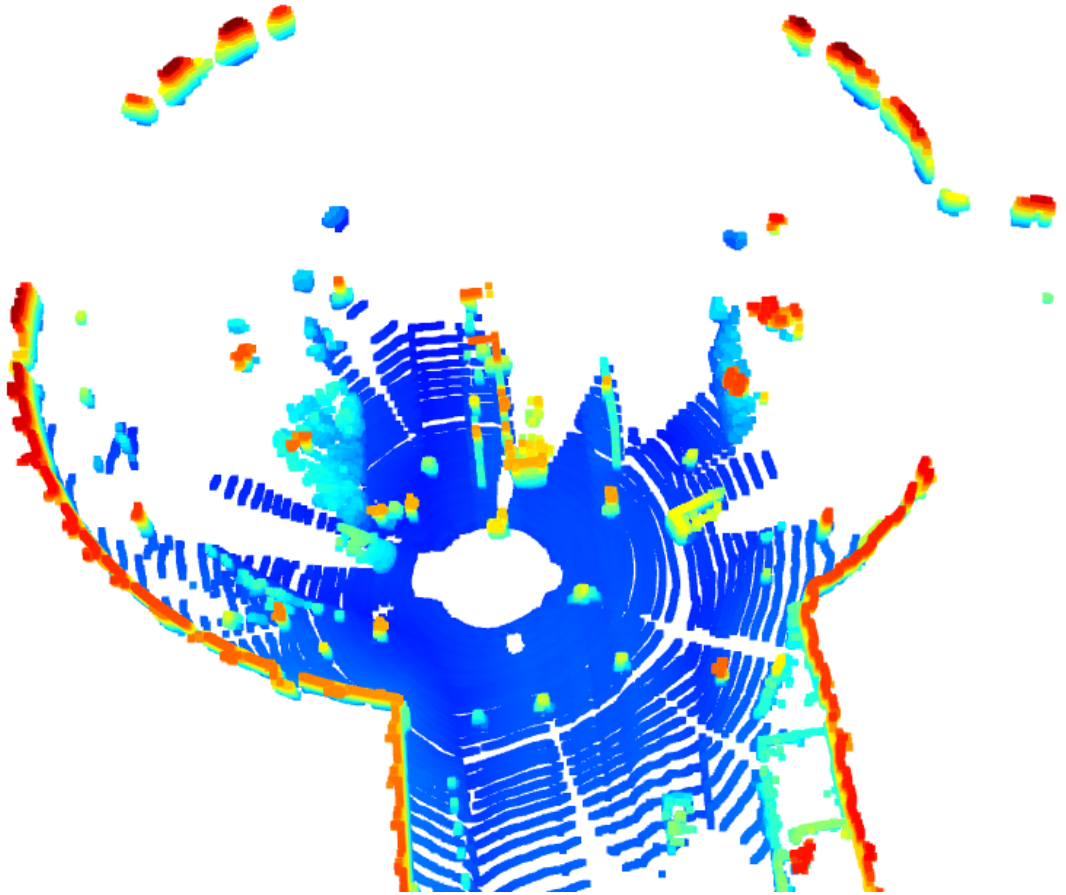
Noise Removal

- Statistical Outlier Removal:
  - Removes points that are further away from their neighbors
  - For each point we find a neighborhood and calculate its distance to the neighbors d_{ij}
  - The distance are then modeled by Gaussian distribution d ~ N(μ, σ)
  - If the mean distance to the neighboring points is outside a defined interval the point will be removed

Statistical Outlier Removal

- Set K the number of closest points around Pi
- Set standard deviation multiplier α
- For every point P_i in the 3D point cloud
  - Find the location of the k nearest neighbors to P_i
  - Compute the average distance d_i from P_i to its k nearest neighbors
- Compute the mean μ_d of the distances d_i
- Compute the standard deviation σ_d of the distance d_i
- Compute the threshold $T = \mu_d + \alpha \,.\, \sigma_d$
- Eliminate the points in the point cloud for which the average distance to its k neighbors is at distance $d > \mathrm{T}$

Statistical Outlier Removal



Mesh down sampling

- Sampling points from mesh using the vertices and faces
- Randomly select vertices based on triangle areas using Heron's formula: $A = \sqrt{s(s-a)(s-b)(s-c)}$ where s is the semi perimeter and a, b, c are the measures of the sides
- S = 0.5 * (a + b + c)
- a(p1, p2) = $||p1 - p2||$
- b(p2, p3) = $||p2 - p3||$
- c(p3, p1) = $||p3 - p1||$

Mesh down sampling

- Get vertices and faces from mesh
- Calculate triangle areas for all the faces
- Sample faces using the areas as weights
- From the sampled faces take the barycentre for each face to sample a point