# Point Cloud Techniques and Applications

Tuan Dang
Ph.D. Candidate
Email: tuan.dang@uta.edu
Homepage: www.tuandang.info
University of Texas at Arlington

Phuc Pham
Teaching Assistant
Email: phuc.phamhuythien@hcmut.edu.vn

Ho Chi Minh City University of Technology

# Agenda

- Section 1 (24/4): Introduction to Point Cloud and basic techniques to process
- Section 2 (26/4): Machine Learning with Point Cloud

# Requirement

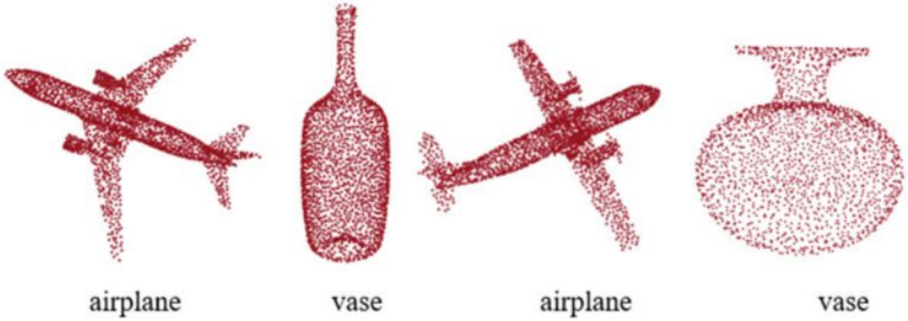
- Linear Algebra
- Basic (numpy)
- Advanced (pytorch)

# Hints to effectively learn this course

- Understanding is the key
- Concepts are connected in a chain: each concept is a link in a chain
- If you don’t understand some concepts at certain links, please stop or go back to review them
- Encourage ask questions even sometimes interrupt the instructor

# Section 1
# Introduction to Point Cloud and basic techniques to process

# What is Point Cloud?

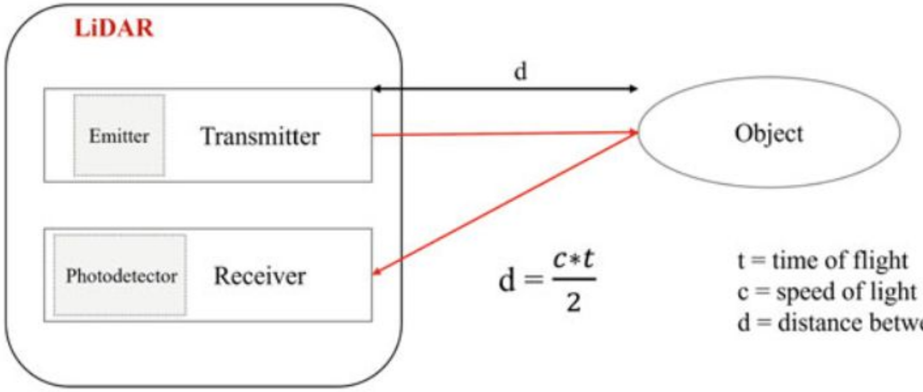



- A list of point 3D points:  [p1, p2
- Each point can be represented
  - (X,Y,Z[,feature1,feature2...])
  - X,Y,Z
  - X,Y,Z,I (intensity)
  - X,Y,Z,R,G,B
  - ...
- The number of points may vary from one point cloud to another

# How to obtain pointcloud?

- LiDAR
- Depth images

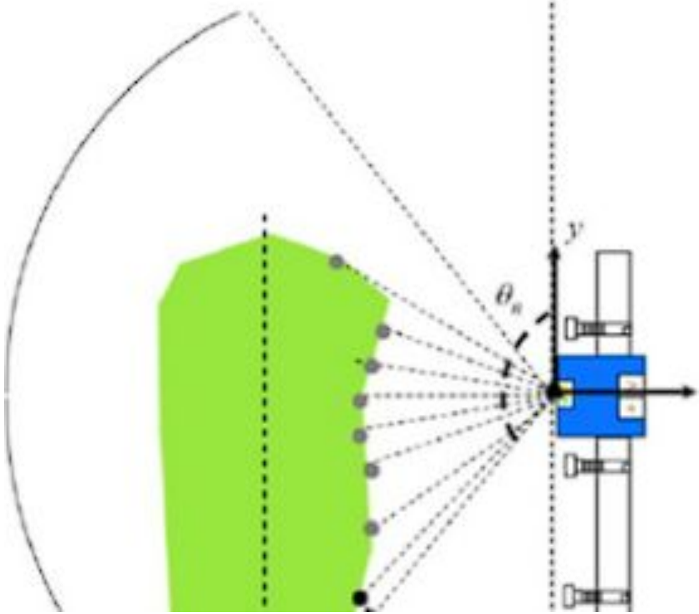
# LiDAR (outdoor)




$$d = \frac{c*t}{2}$$

t = time of flight
c = speed of light in air
d = distance between LiDAR and object

**Sing Beam**



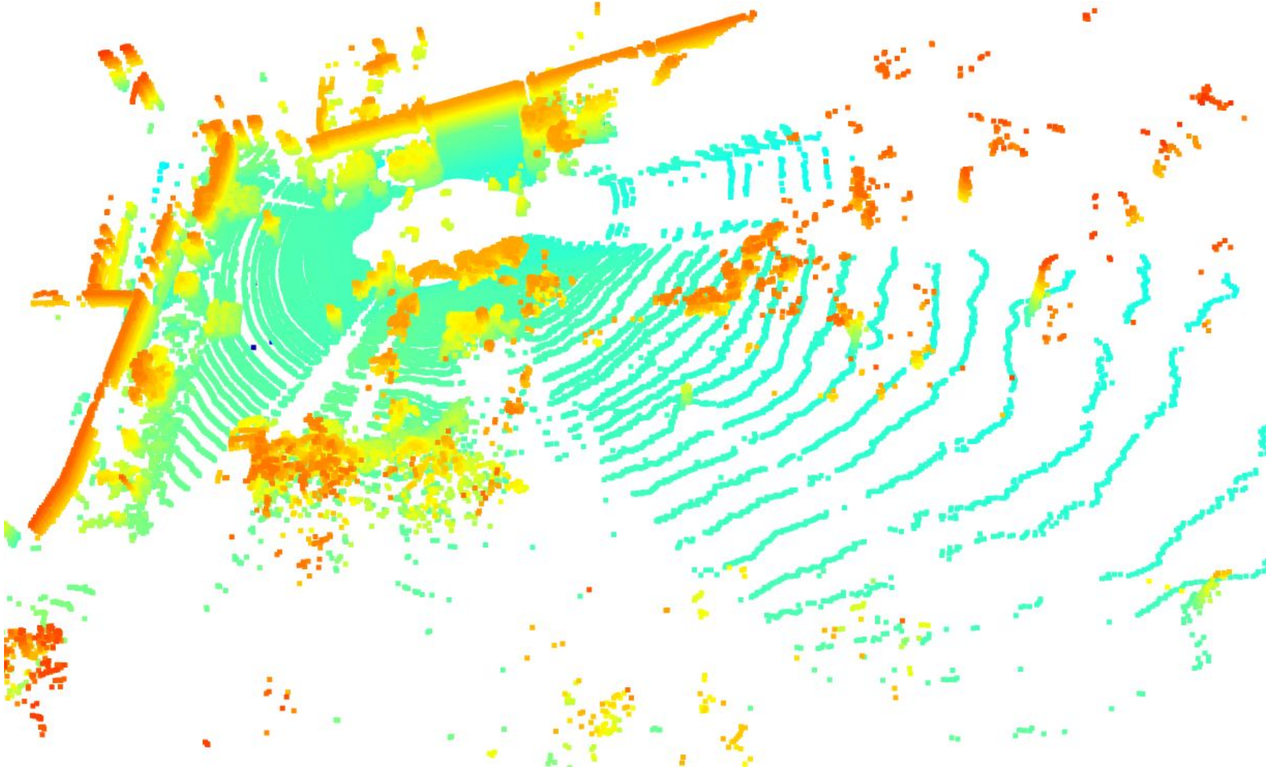

For each point: (X,Y,Z, I)

**Multiple beams**

# Industry LiDAR

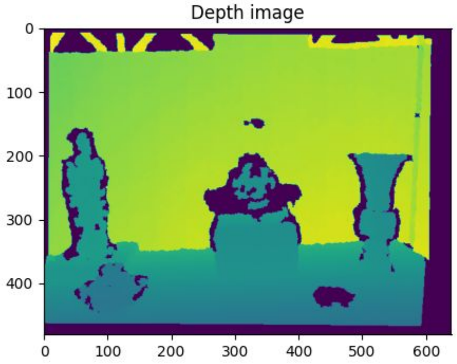


# LiDAR Data



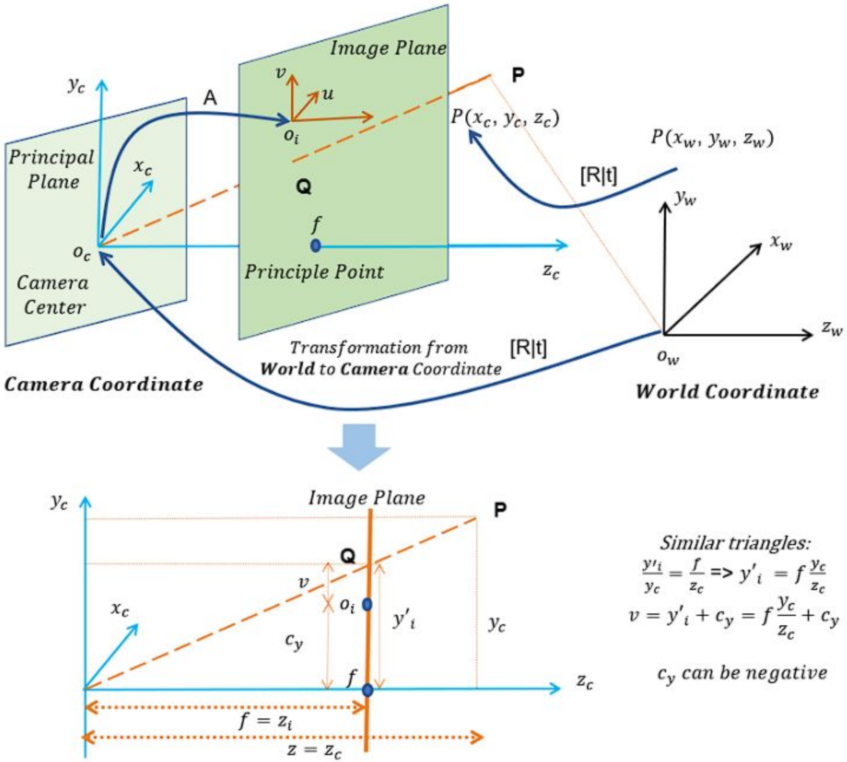
*Generated from: https://github.com/tuantdang/pointcloud_lessons/blob/main/session1/load_point_clouds.py*

# RGB-D camera (most popular, indoor)


*Generated from: https://github.com/tuantdang/pointcloud_lessons/blob/main/session1/registration.py*

# Camera model (Pinhole)




## From world to camera

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \frac{1}{z} \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ 1 \end{bmatrix}$$

## From Camera to World

$$\begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = R \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} + t = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

or can be written as:

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

# Commercial RGB-D cameras

Microsoft
Kinect

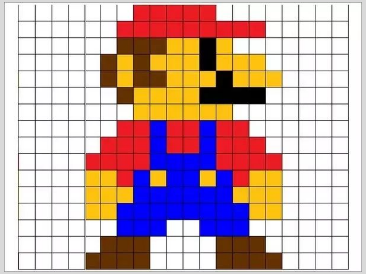
Intel
Realsense D435

Intel
Realsense L515

# Depth Measurement

- Stereo Vision
- Structure Light
- LiDAR with MEMS

# Point Cloud vs Image

| | Unordered- irregular structure | Well-defined grid structure |
|---|---|---|
| Memory Space |  [p1, p2, …, ] = [p2, p1, …, ] |  WxHxC |
| Storage Space | txt, ply, bin | JPEG, PNG, BITMAP |

# Point cloud processing

- Filter noises
  - Downsampling
  - Noise Removal
- Search
- Registration

# Voxel Grid Sampling

- Build voxel grid
- Select point strategies
  - Random
  - Center point
  - Centroid point

# Build Voxel Grid Coordinate

$$\begin{aligned} x_{\min} &= \min(x_1, x_2, \cdots, x_n), \\ x_{\max} &= \max(x_1, x_2, \cdots, x_n), \\ y_{\min} &= \min(y_1, y_2, \cdots, y_n), \\ &\vdots \\ z_{\max} &= \max(z_1, z_2, \cdots, z_n). \end{aligned}$$

Point Cloud Range

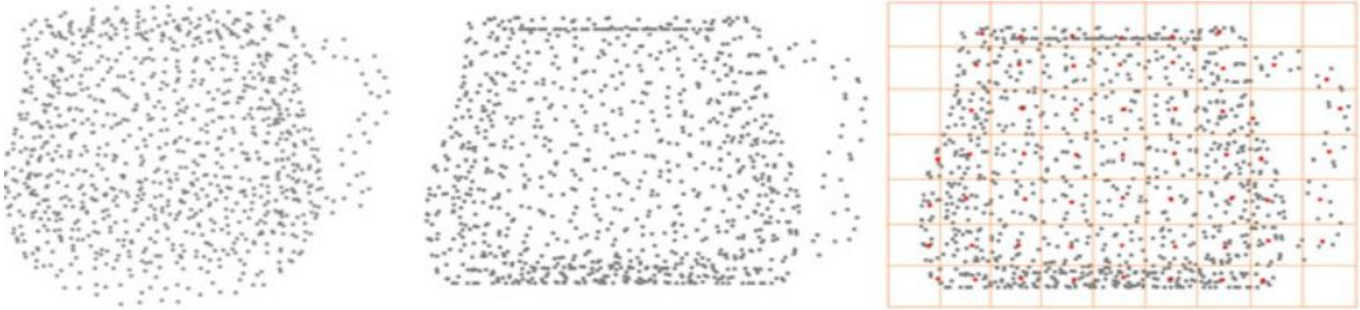
$$\begin{aligned} N_x &= (x_{\max} - x_{\min})/r, \\ N_y &= (y_{\max} - y_{\min})/r, \\ N_z &= (z_{\max} - z_{\min})/r, \end{aligned}$$

Grid Dimensions

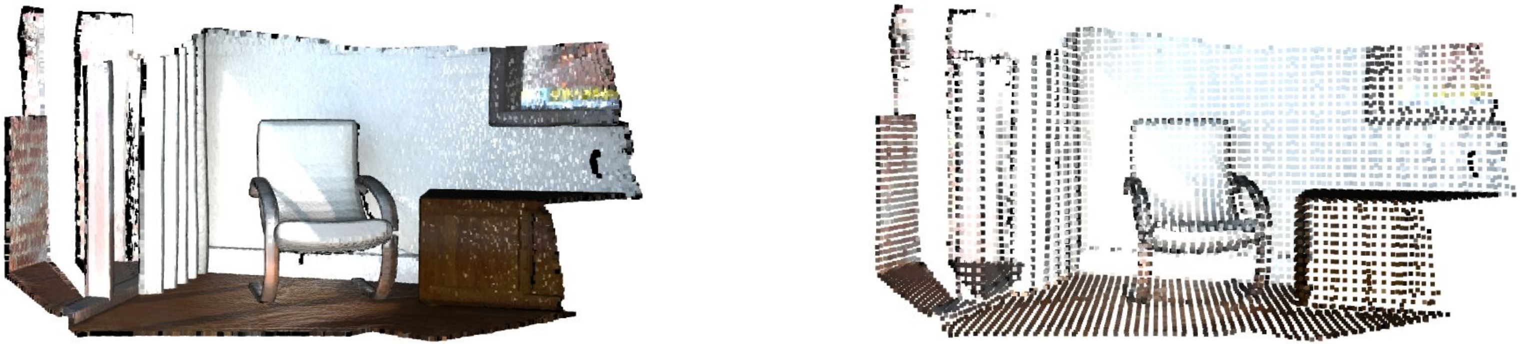
r : voxel size

$$\begin{aligned} i_x &= \lfloor (x - x_{\min})/r \rfloor, \\ i_y &= \lfloor (y - y_{\min})/r \rfloor, \\ i_z &= \lfloor (z - z_{\min})/r \rfloor, \\ i &= i_x + i_y * N_x + i_z * N_x * N_z. \end{aligned}$$

Voxel indexing
0,1,..., NxNyNz-1

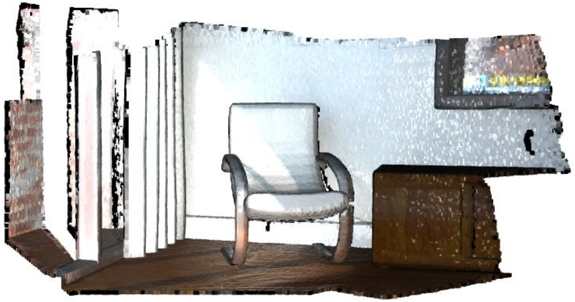


Center Point Selection



# Furthest Point Sampling (FPS)

- This heuristic algorithm
- Algorithm
  - Step 1: Random select a point
  - Step 2: Calculate distances between current selected point to all remaining points
  - Step 3: Select a point which is the furthest to the previous point
  - Step 4: Loop step 2 & 3 until m points are chosen
- Advantages
  - Evenly across original point cloud
  - Capture Esen



Voxel Sampling

FPS

*Generated from: https://github.com/tuantdang/pointcloud_lessons/blob/main/session1/down_sampling.py*

# Surface Normal

- A vector that is **perpendicular** to the plane
- A plane can be determined by
  - Perpendicular vector (a,b,c)
  - A point in the plane (x,y,z)
- Plane equation from perpendicular vector and inner-point: **a**x+**b**y+**c**z+d = 0

$$\mathbf{n} = [n_x, n_y, n_z]^T = \frac{[a, b, c]^T}{\|[a, b, c]^T\|}.$$

# Normal vector in Practise

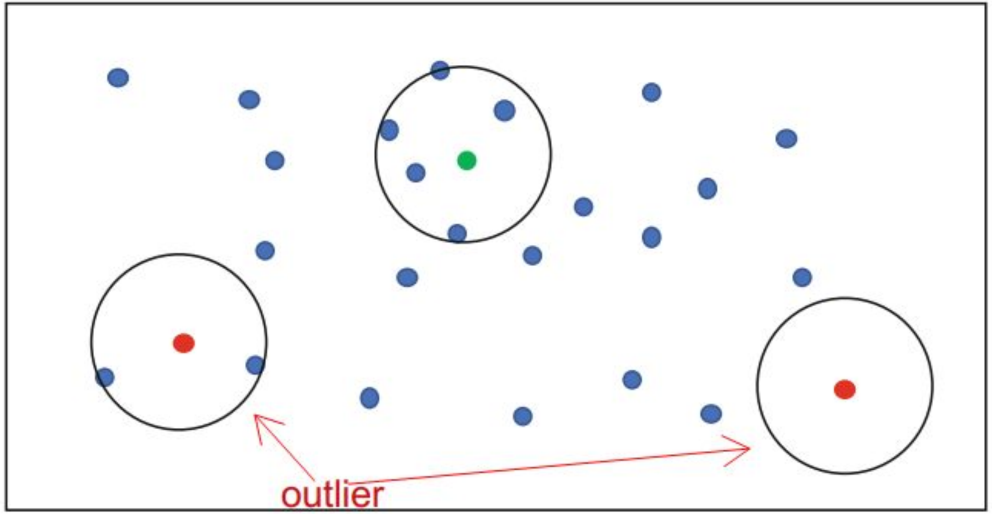
- Not points are perfect in a plane => eigenvector
- Collect a set of point
- Calculate eigenvectors and eigenvalues ($\lambda 1$, $\lambda 2$, $\lambda 3$)
- The least driven-value is normal vector: **n = $\boldsymbol{\lambda 3}$**

# Surface variance = Curvature

$$c = \frac{\lambda_3}{\lambda_1 + \lambda_2 + \lambda_3}$$

# Noise Removal: Radius Outlier Removal

- Search around query point **p** with radius r, call **ball(p, r)**
- If number of points in the ball < $k_{min}$, **p** is outlier.




Radius outlier removal: $k_{min} = 4, r = 1$

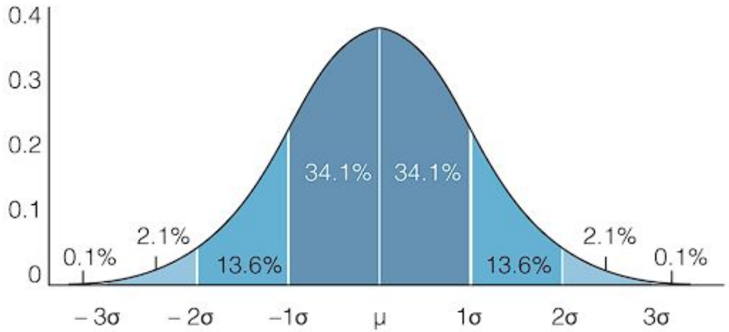
# Statistical Outlier Removal

Model inlier as Gaussian **N(μ, σ) distribution**

- n : number of points, k neighbors, **c : variance steps**
- dij : distance from point i to point j

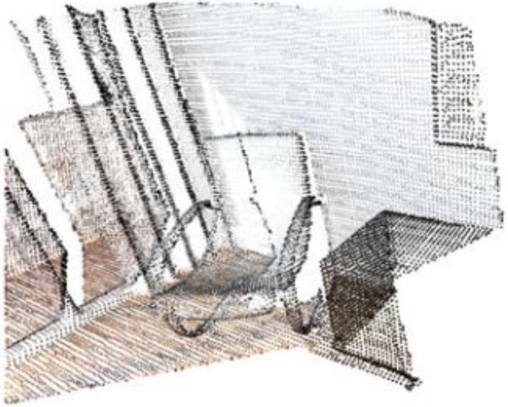
$$\mu = \frac{1}{nk}\sum_{i=1}^{n}\sum_{j=1}^{k} d_{ij}, \quad \sigma = \sqrt{\frac{1}{nk}\sum_{i=1}^{n}\sum_{j=1}^{k}(d_{ij}-\mu)^2}.$$

$$\frac{1}{k}\sum_{j=1}^{k} d_{ij} > \mu + c\sigma \quad \text{or} \quad \frac{1}{k}\sum_{j=1}^{k} d_{ij} < \mu - c\sigma, \; c \in \mathbb{R}^+.$$

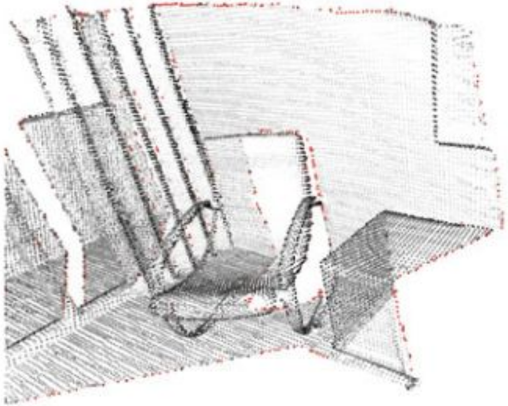


## Examples

Original (a)

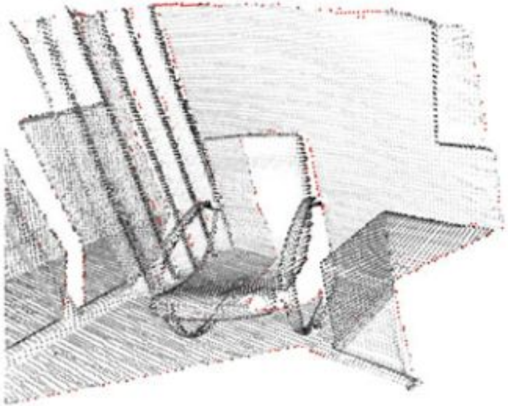


(b) Voxel size = 0.02



(c)

kmin=16,r=0.05



(d)

Statistical Outlier removal k=20,c=2

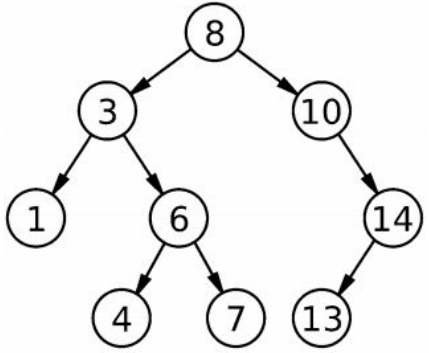
# Search  Nearest Neighbors (-NN)

- Why searching neighbors?
- K-NN : O(NlogN) or O(NlogK) since using sort algorithm
- Radius-NN : O(N) compute and compare


Comparison of $K$-NN and Radius-NN. (**a**) $K$-NN. (**b**) Radius-NN

# 1D-BST = 1D Binary Search Tree

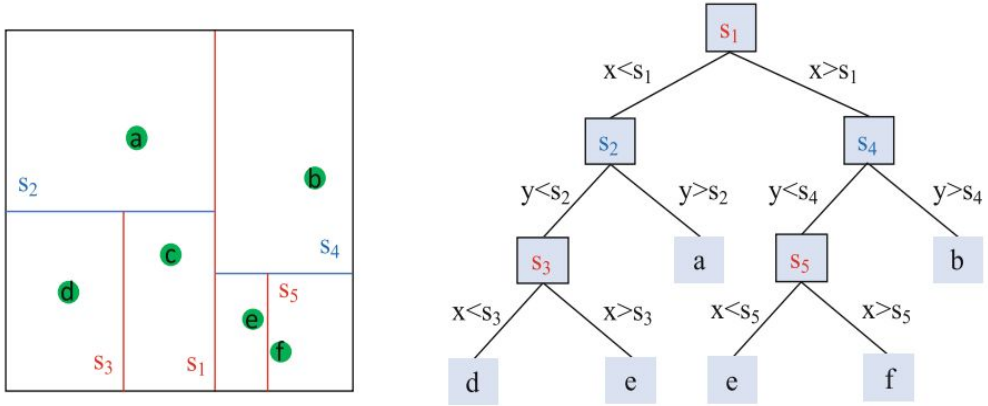
8,3,10,1,6,14,4,7,14




5

# k-dimensional tree = k-d tree

- K-d tree is a BST where each node is k-dimensional point
- Example:
  - Input 2D points: a,b,c,d,e,f
  - Draw a line: perpendicular x-axis to separate two subsets
  - For each subset, draw  lines:
    - perpendicular y-axis
    - Separate two subsets
  - Repeat until no more data
- Note: Using round-robin
  - 2D: perpendicular lines: x>y>x>y>...
  - 3D: perpendicular planes: x>y>z>x>y>z>...

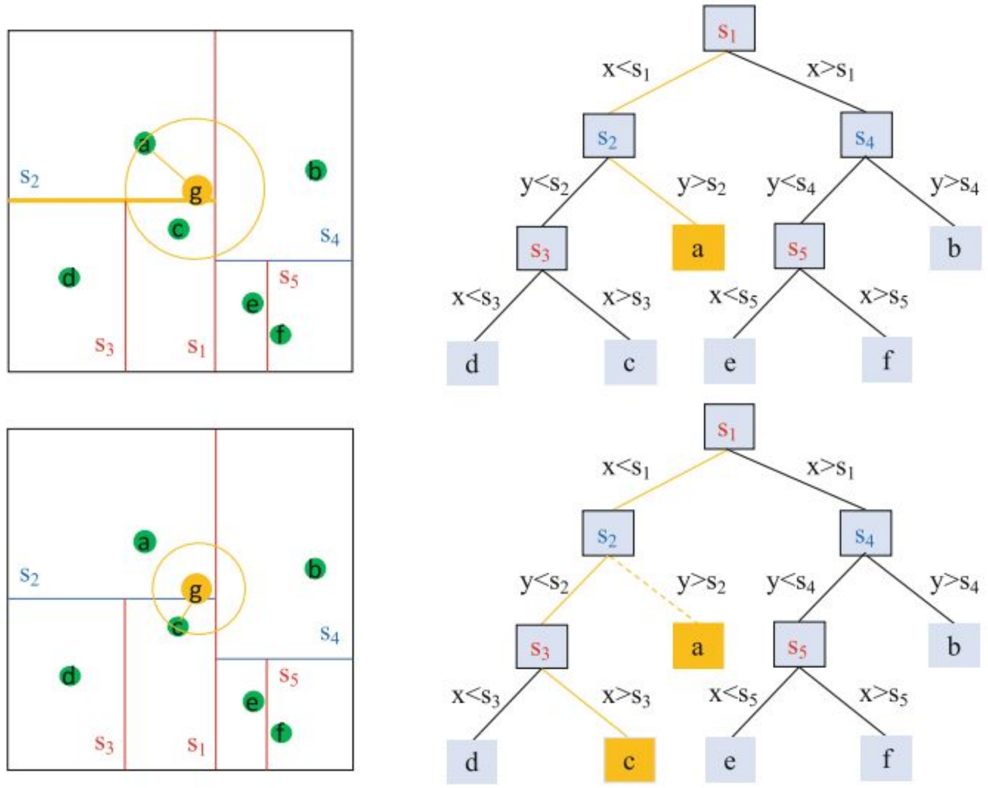



**Construct 2-d tree from 2D data**

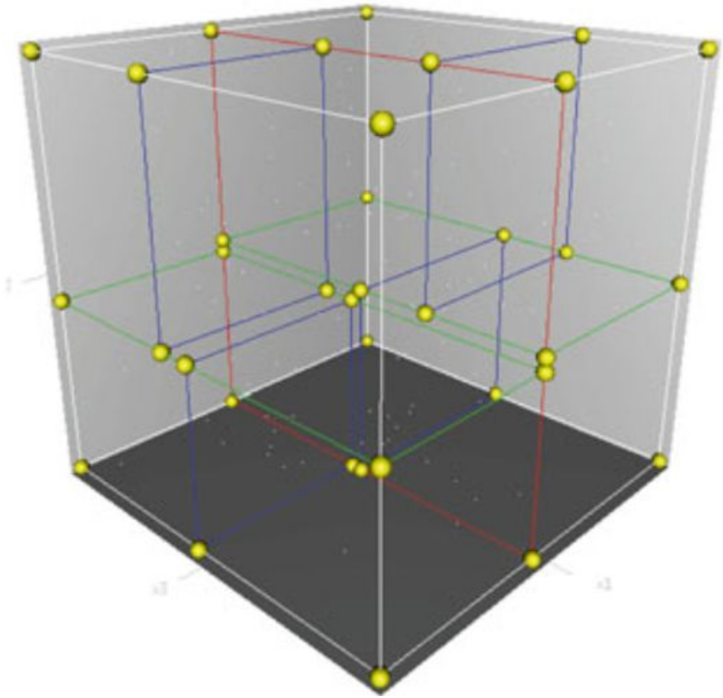
# K-NN search on k-d tree

Example with K=2 and query **g, with worst distance d = infinity**

- Search from root s1
    - gx < s1 and gy >s2 where ||g-a|| < d store **a**.
    - Update d = ||g - a||
    - Since |gx-s1|<d and |gy-s2| < d, **c** is stored
    - Update d = ||g - c||
    - ....
- Complexity O(logN): best case and O(N) worst case

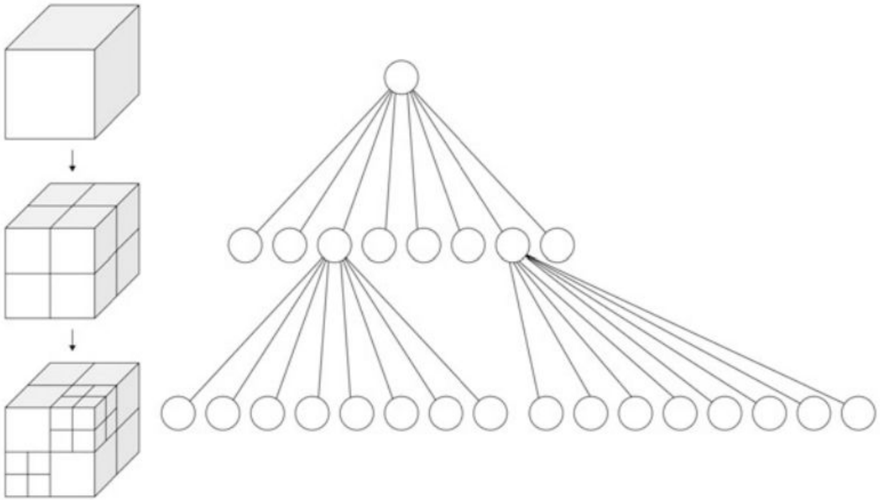


# k-d tree with k=3



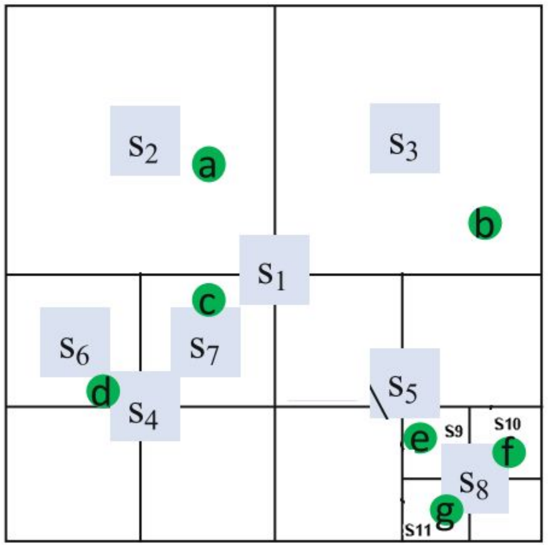
# Octree

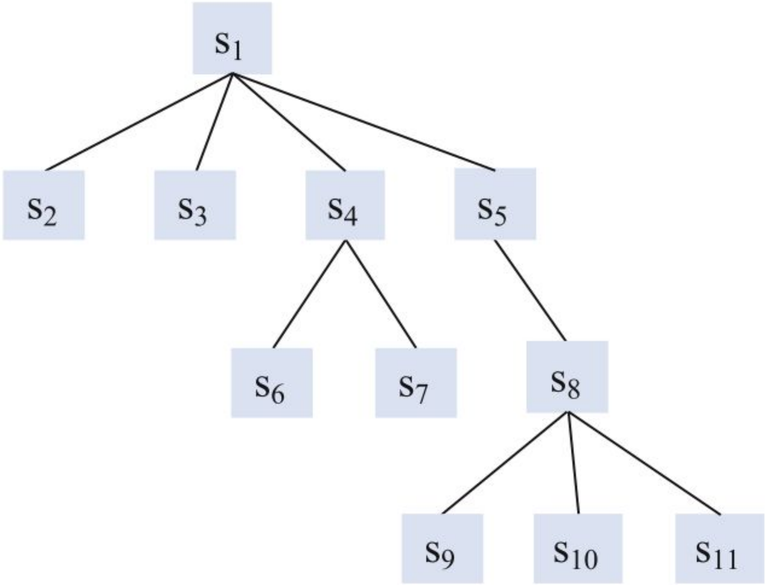
- k-d tree split along **a dimension** while octree split along **a point**.
- The center represents for the regions and has 8 children (3D) or 4 children (2D)



# Octree construction

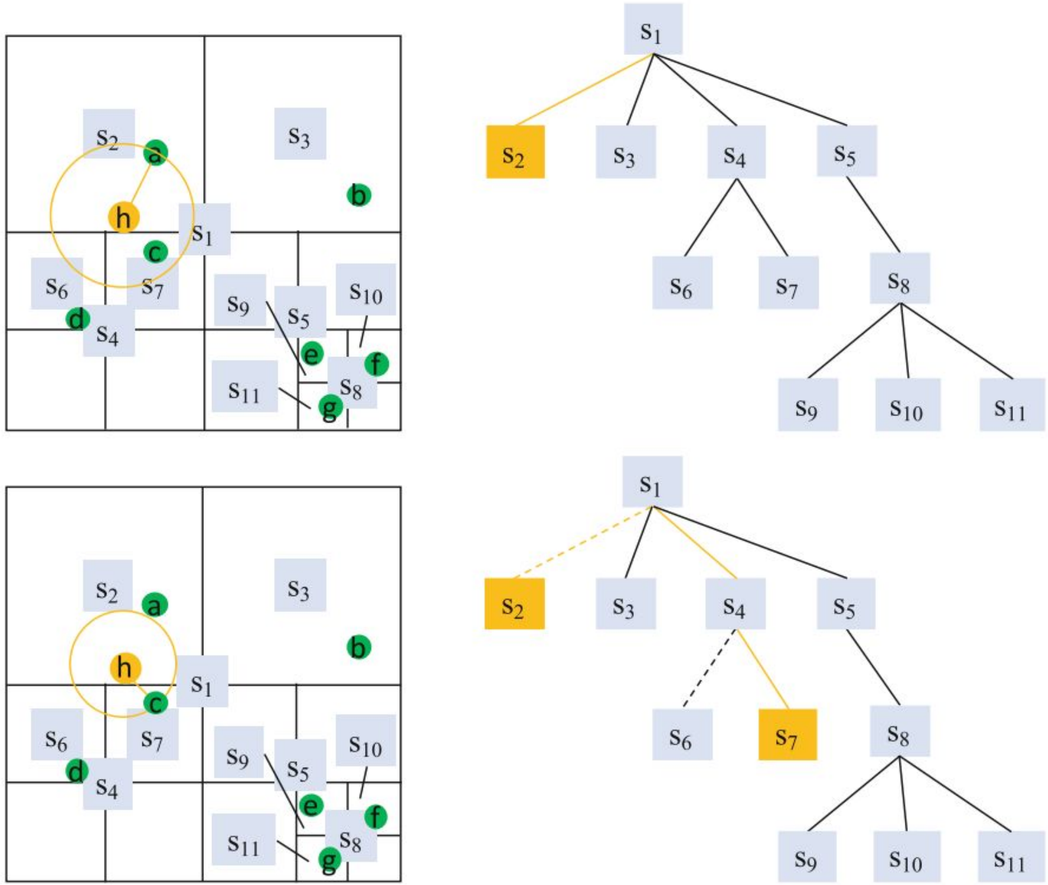
2D example a(4, 6), b(7,5), c(4,4), d(2,2), e(6, 3), f(8,2), g(7,1)



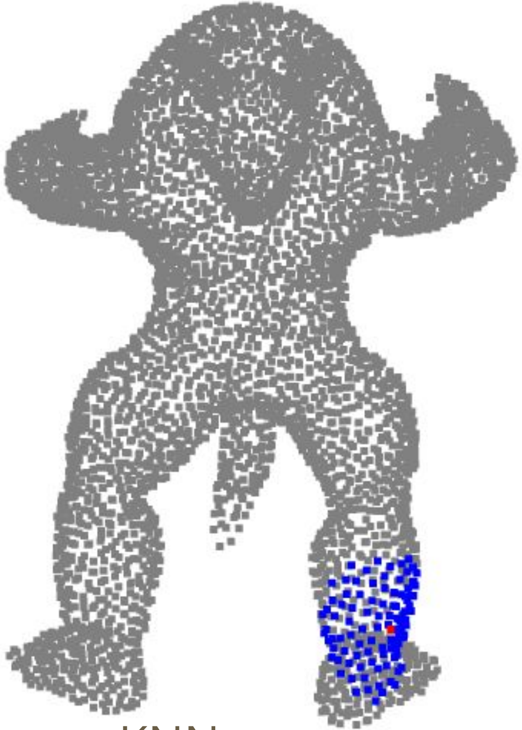



# Octree search with K-NN

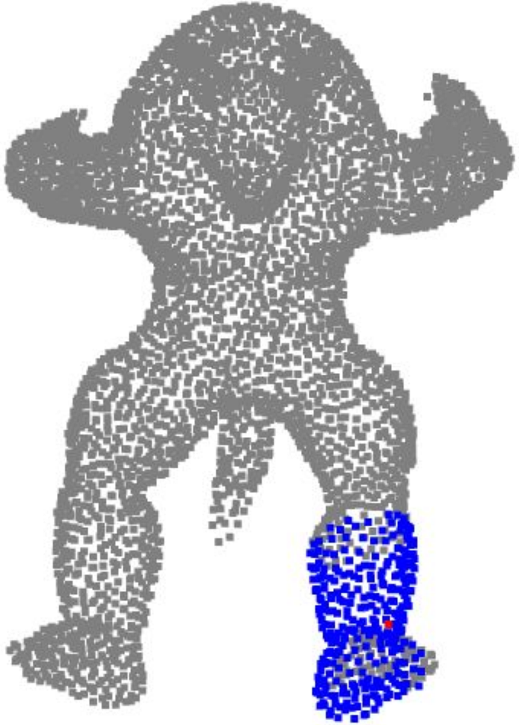
- Use Depth-First Search (DFS)
- Example :
  - Query **h**
  - k=2
  - Initialize distance **d = ∞**
  - Start from s1 find h in s2 among children > store **a**
  - Update d = ||h-a||
  - Ball circle(h, d) intersect with **s4**
  - Search children s4: s6,s7
  - Store **c** where d < |h-c|



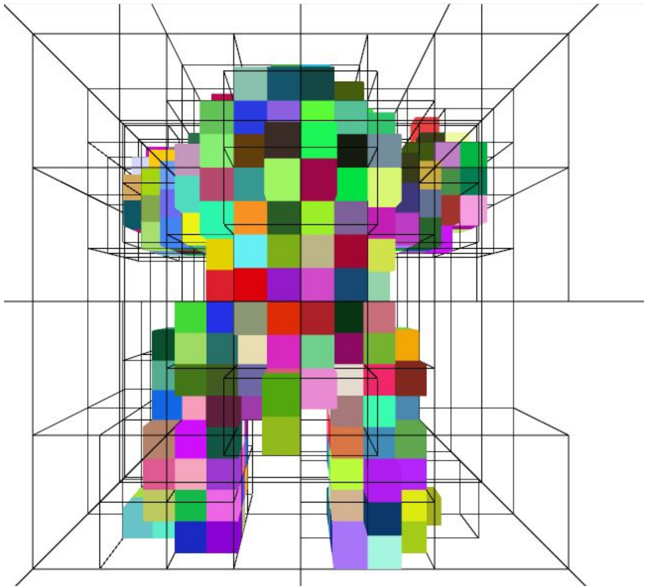
# Example



KNN



Radius



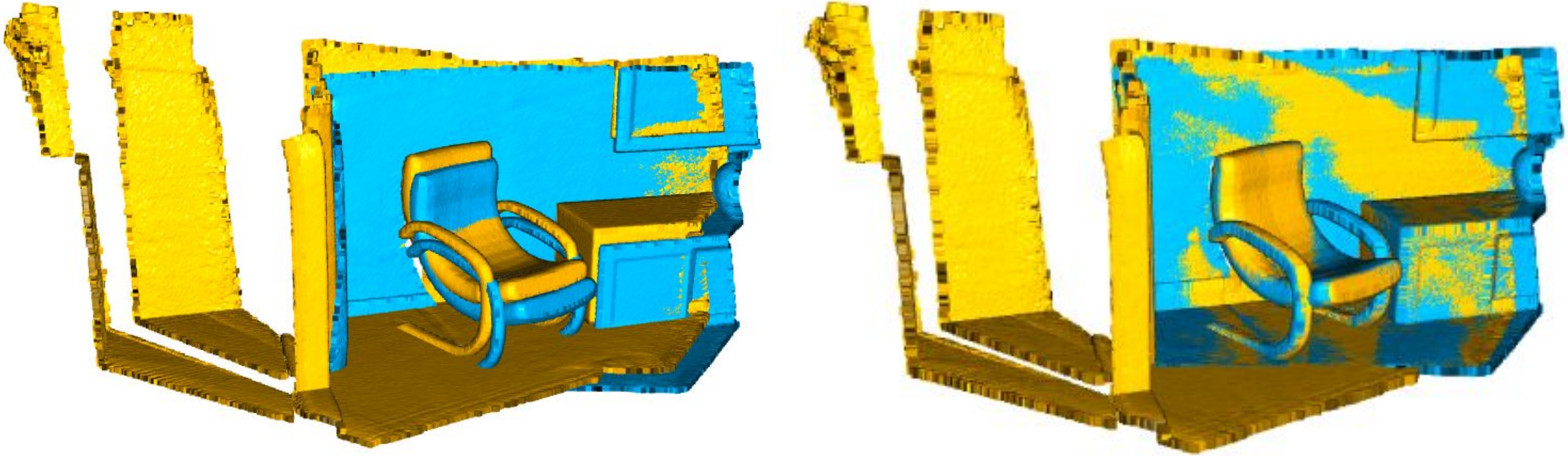
Octree

# Registration

- Align two or more point clouds into **a common coordinate**
- Iterative Closest Point (ICP) is the most common method
  - Find the correspondence between two point sets
  - Find the transformation that minimizes the Euclidean distance between matching points
- Mathematic formation:
  - Let A = $\{a_i\}$, B = $\{b_j\}$ are two point sets
  - Let T is transformation
  - Point $\mathbf{m_i}$ in A which corresponding with $\mathbf{Tb_j}$
  - Find

$$T = \arg\min_{T} \left\{ \sum_i \|T \cdot b_i - m_i\|^2 \right\}$$

# Example

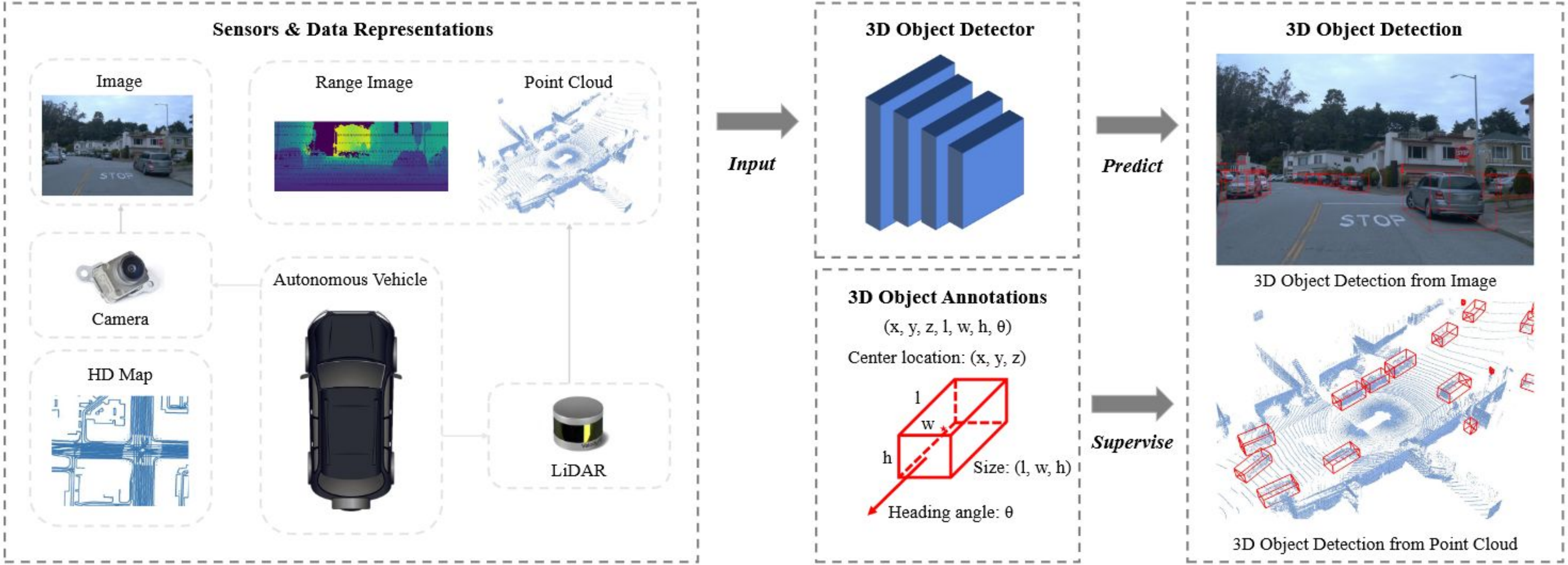


# Coding Examples

- Introduction to **Open3D installation (v0.1)**
- Create folder to store all files below:
- Load a point cloud
  - Bin (KITTI): load_bin.py
  - Ply (Shapenet): load_ply.py
  - Txt (ModelNet): load_txt.py (convert from numpy PCL)
- Downsampling : **downsampling.py**
- Noise Removal : **noise_removal.py**
- Search neighbors: k-d-tree / octoc.tree (color search results)
- Registration: registration.py
- Reconstruct point cloud from RGB images and depth images:

# Section 2
# Introduction to Point Cloud and basic techniques to process

# Overview

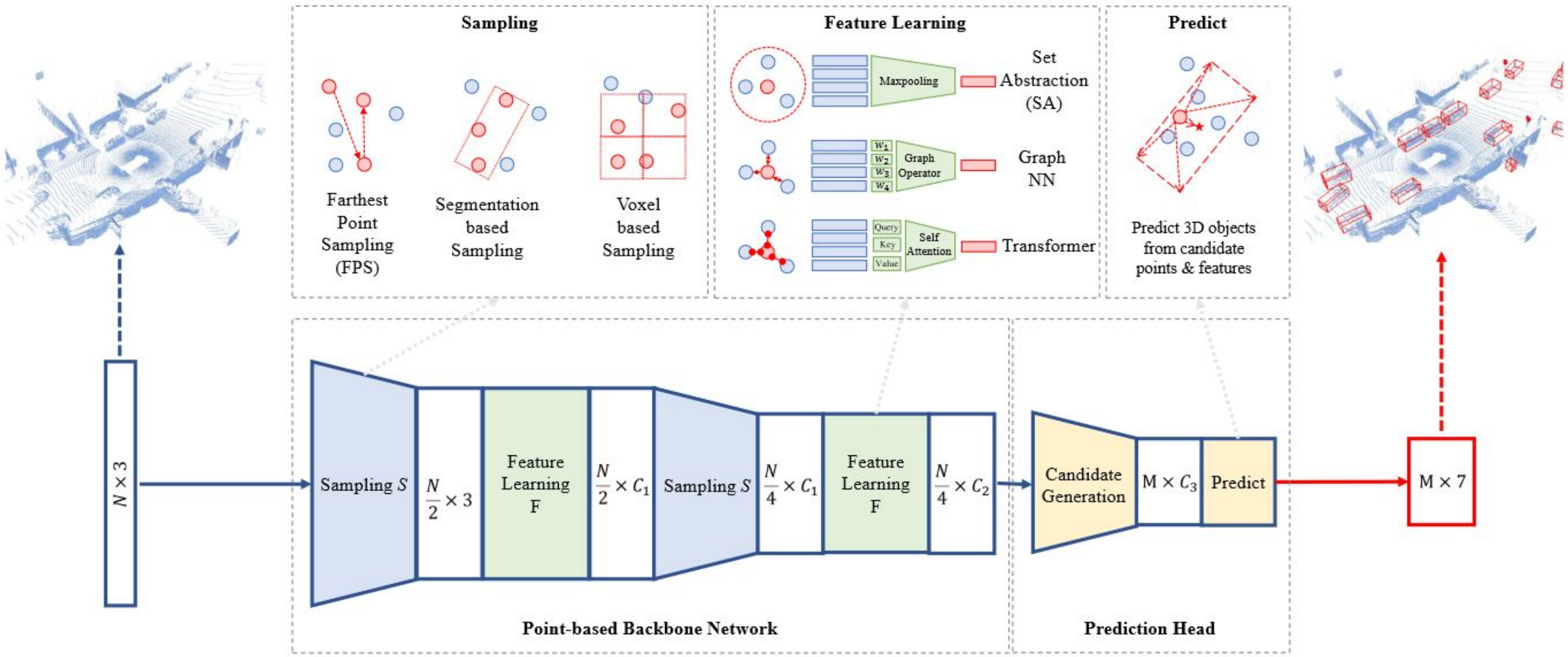


Sensors & Data Representations
Image
Range Image
Point Cloud
Camera
Autonomous Vehicle
HD Map
LiDAR
Input
3D Object Detector
Predict
3D Object Annotations
(x, y, z, l, w, h, θ)
Center location: (x, y, z)
l
w
h
Size: (l, w, h)
Heading angle: θ
Supervise
3D Object Detection
3D Object Detection from Image
3D Object Detection from Point Cloud

# Datasets

| Dataset | Year | Size (hr.) | Real-world | LiDAR scans | Images | 3D annotations | Classes | night/rain | Locations | Other data |
|---|---|---|---|---|---|---|---|---|---|---|
| KITTI [80, 81] | 2012 | 1.5 | Yes | 15k | 15k | 200k | 8 | No/No | Germany | - |
| KAIST [50] | 2018 | - | Yes | 8.9k | 8.9k | Yes | 3 | Yes/No | Korea | thermal images |
| ApolloScape [104, 166] | 2019 | 100 | Yes | 20k | 144k | 475k | 6 | -/- | China | - |
| H3D [198] | 2019 | 0.77 | Yes | 27k | 83k | 1.1M | 8 | No/No | USA | - |
| Lyft L5 [107] | 2019 | 2.5 | Yes | 46k | 323k | 1.3M | 9 | No/No | USA | maps |
| Argoverse [29] | 2019 | 0.6 | Yes | 44k | 490k | 993k | 15 | Yes/Yes | USA | maps |
| AIODrive [293] | 2020 | 6.9 | No | 250k | 250k | 26M | - | Yes/Yes | - | long-range data |
| A*3D [202] | 2020 | 55 | Yes | 39k | 39k | 230k | 7 | Yes/Yes | SG | - |
| A2D2 [82] | 2020 | - | Yes | 12.5k | 41.3k | - | 14 | -/- | Germany | - |
| Cityscapes 3D [77] | 2020 | - | Yes | 0 | 5k | - | 8 | No/No | Germany | - |
| nuScenes [15] | 2020 | 5.5 | Yes | 400k | 1.4M | 1.4M | 23 | Yes/Yes | SG, USA | maps, radar data |
| Waymo Open [250] | 2020 | 6.4 | Yes | 230k | 1M | 12M | 4 | Yes/Yes | USA | maps |
| Cirrus [288] | 2021 | - | Yes | 6.2k | 6.2k | - | 8 | -/- | USA | long-range data |
| PandaSet [301] | 2021 | 0.22 | Yes | 8.2k | 49k | 1.3M | 28 | Yes/Yes | USA | - |
| KITTI-360 [142] | 2021 | - | Yes | 80k | 300k | 68k | 37 | -/- | Germany | - |
| Argoverse v2 [295] | 2021 | - | Yes | - | - | - | 30 | Yes/Yes | USA | maps |
| ONCE [172] | 2021 | 144 | Yes | 1M | 7M | 417K | 5 | Yes/Yes | China | - |

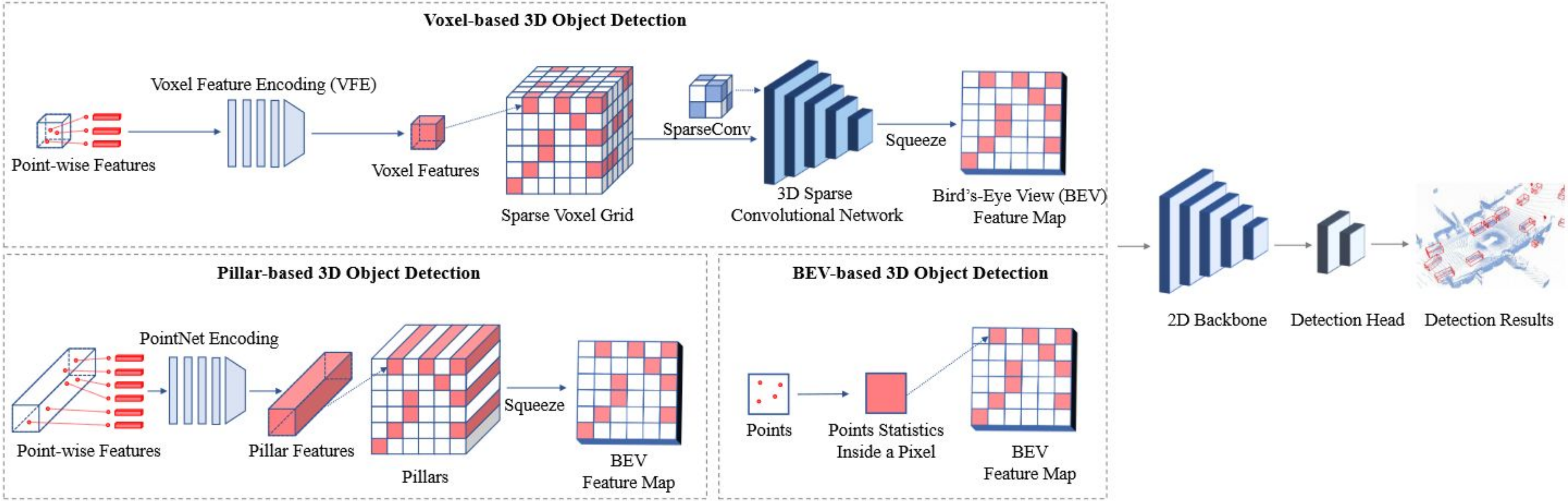
# Point-based 3D methods



# Examples

| Method | Context $\Omega$ | Sampling $S$ | Feature $F$ |
|---|---|---|---|
| PointRCNN [234] | Ball Query | FPS | Set Abstraction |
| IPOD [318] | Ball Query | Seg. | Set Abstraction |
| STD [319] | Ball Query | FPS | Set Abstraction |
| 3DSSD [321] | Ball Query | Fusion-FPS | Set Abstraction |
| Point-GNN [238] | Ball Query | Voxel | Graph |
| StarNet [189] | Ball Query | Targeted-FPS | Graph |
| Pointformer [193] | Ball Query | FPS + Refine | Transformer |

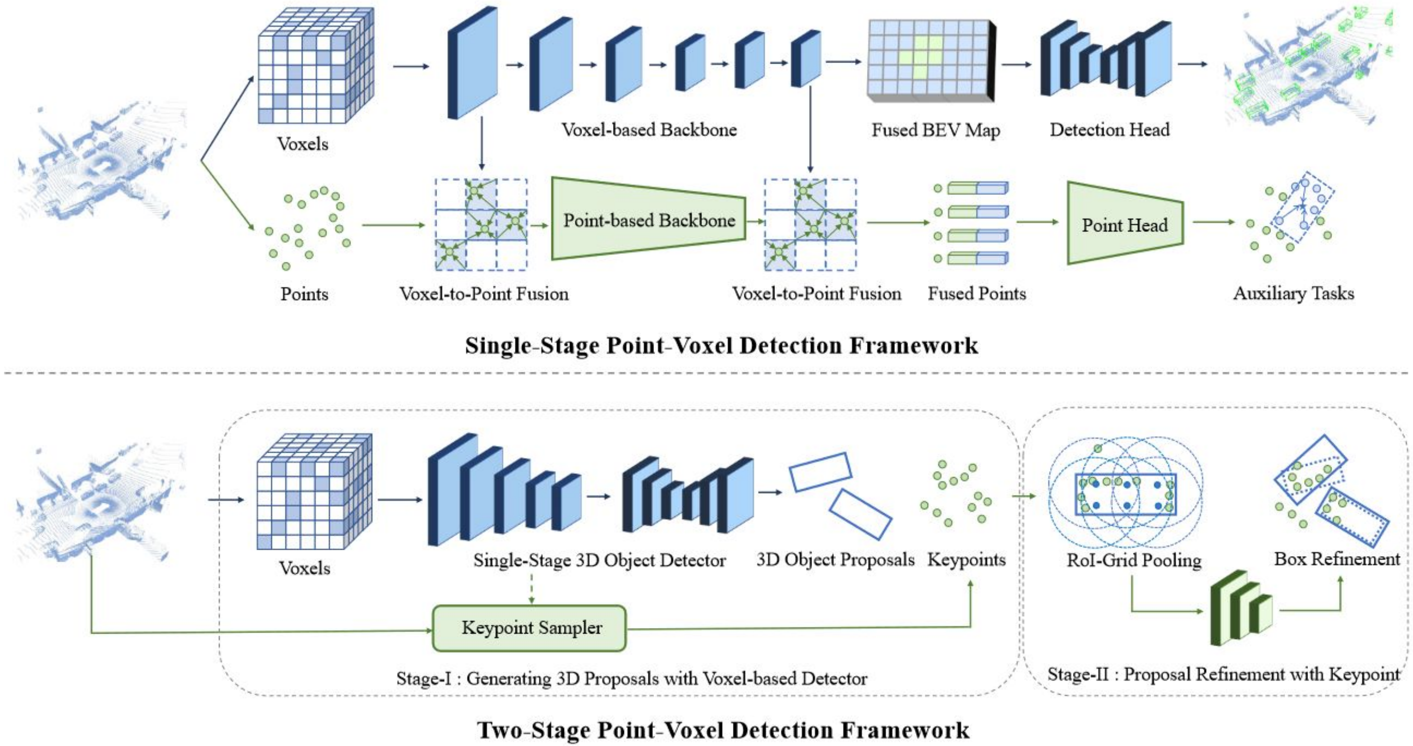
# Voxel (3D grid)-based methods



# Example

| Method | Representation | Feature Encoding | Backbone Model | Refinement Head |
|---|---|---|---|---|
| Vote3D [264] | voxel | voxelization | 3D sparse CNN w/ voting | - |
| Vote3Deep [66] | voxel | voxelization | 3D sparse CNN w/ voting | - |
| 3D-FCN [118] | voxel | voxelization | 3D FCN | - |
| VeloFCN [119] | BEV map | projection | 2D FCN | - |
| BirdNet [10] | BEV map | projection | 2D RPN | - |
| PIXOR [314] | BEV map | projection | 2D CNN | - |
| HDNet [313] | BEV map, HD map | projection | 2D CNN | - |
| VoxelNet [359] | voxel, BEV map | voxelization, VFE | 3D sparse CNN, 2D RPN | - |
| SECOND [312] | voxel, BEV map | voxelization, VFE | 3D sparse CNN, 2D RPN | - |
| MVF [360] | voxel, BEV+PV map | voxelization, VFE | 2D CNN | - |
| PointPillars [117] | pillar, BEV map | PointNet encoding | 2D CNN | - |
| Pillar-OD [283] | pillar, BEV+CYV map | PointNet encoding | 2D CNN | - |
| Part-$A^2$ Net [236] | voxel, BEV map | voxelization, VFE | 3D sparse CNN, 2D RPN | voxel head |
| Voxel R-CNN [57] | voxel, BEV map | voxelization, VFE | 3D sparse CNN, 2D RPN | voxel head |
| CenterPoint [329] | voxel, BEV map | voxelization, VFE | 3D sparse CNN, 2D RPN | - |
| Voxel Transformer [173] | voxel, BEV map | voxelization, VFE | 3D Transformer, 2D RPN | - |

# Hybrid: Point-Voxel based methods

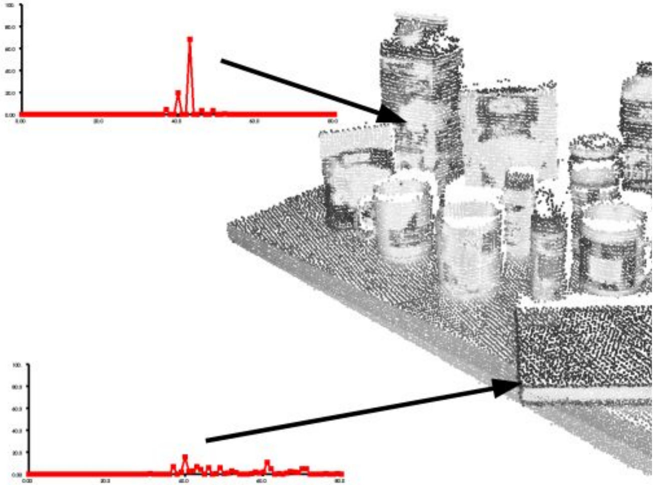


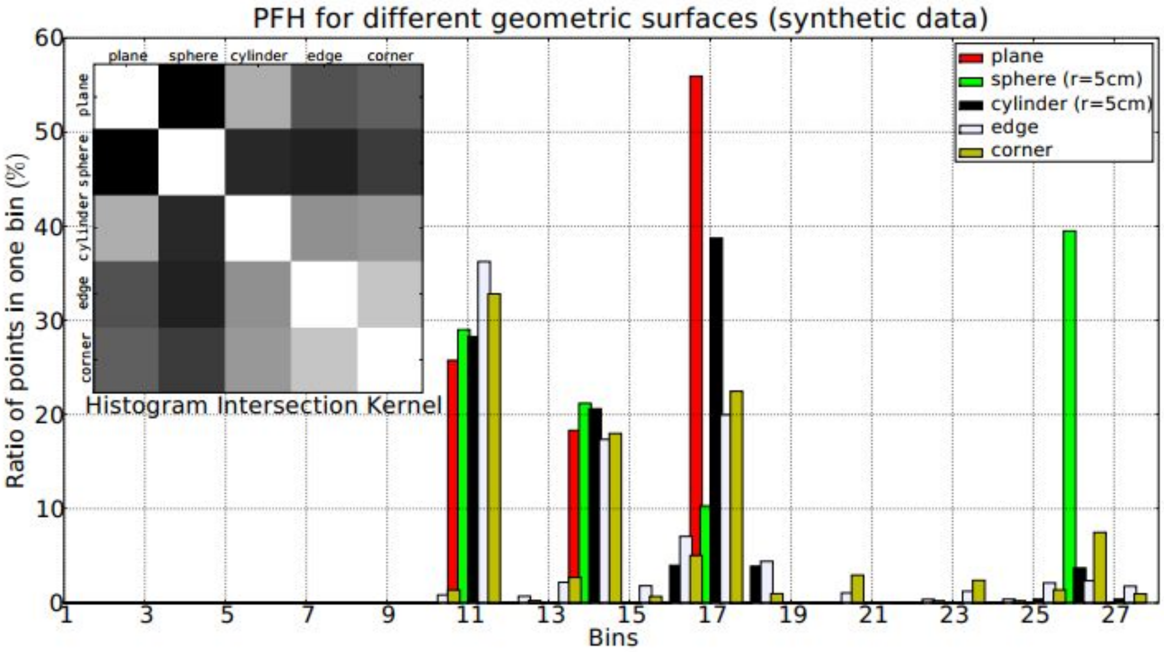
# Examples

| Method | Contribution |
|---|---|
| **Single-Stage Detection Framework** | |
| PVCNN [152] | Point-Voxel Convolution |
| SPVNAS [254] | Sparse Point-Voxel Convolution |
| SA-SSD [92] | Auxiliary Point Network |
| PVGNet [181] | Point-Voxel-Grid Fusion |
| **Two-Stage Detection Framework** | |
| Fast Point R-CNN [43] | RefinerNet |
| PV-RCNN [235] | RoI-grid Pooling |
| PV-RCNN++ [237] | VectorPool |
| Pyramid R-CNN [171] | RoI-grid Attention |
| LiDAR R-CNN [134] | Scale-aware Pooling |
| CT3D [233] | Channel-wise Transformer |

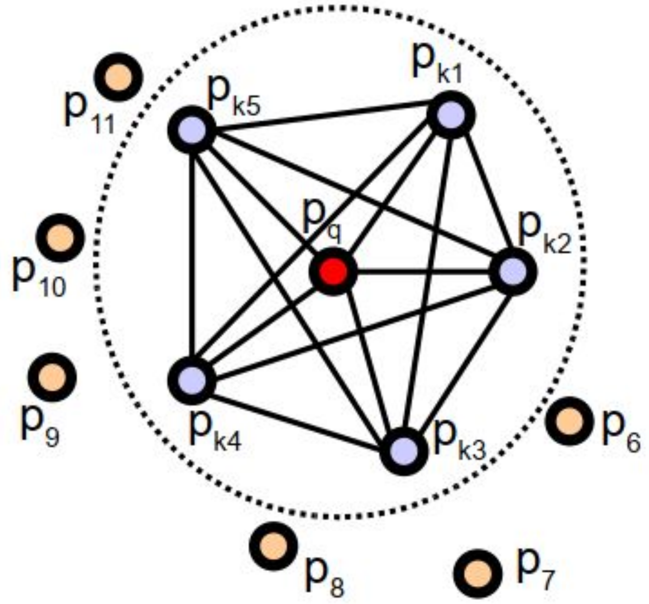
# Intrinsic Geometry Features (Hand-crafted): Point Feature Histogram (PFH)

- Surface normal and curvature are too simple to represent point features





# Algorithms

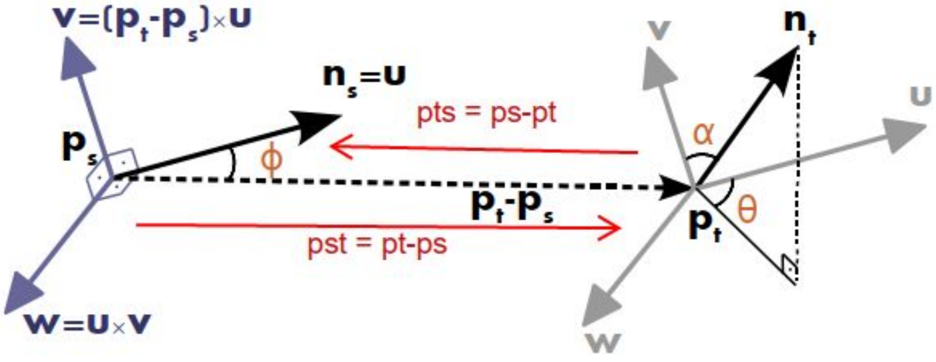



$$(\exists) r_1, r_2 \in \mathbb{R},\ r_1 < r_2,\ \text{such that} \begin{cases} r_1 \Rightarrow \mathcal{P}^{k_1} \\ r_2 \Rightarrow \mathcal{P}^{k_2} \end{cases}, \text{with } 0 < k_1 < k_2$$

**Dual rings**:

- First ring: estimate surface normal (see section 1)
- Second ring: estimate PFH

# Darboux frame




$$\begin{cases} \mathrm{u} = \boldsymbol{n}_s \\ \mathrm{v} = \mathrm{u} \times \dfrac{(\boldsymbol{p}_t - \boldsymbol{p}_s)}{\|\boldsymbol{p}_t - \boldsymbol{p}_s\|_2} \\ \mathrm{w} = \mathrm{u} \times \mathrm{v} \end{cases}$$

$$\alpha = \mathrm{v} \cdot \boldsymbol{n}_t$$

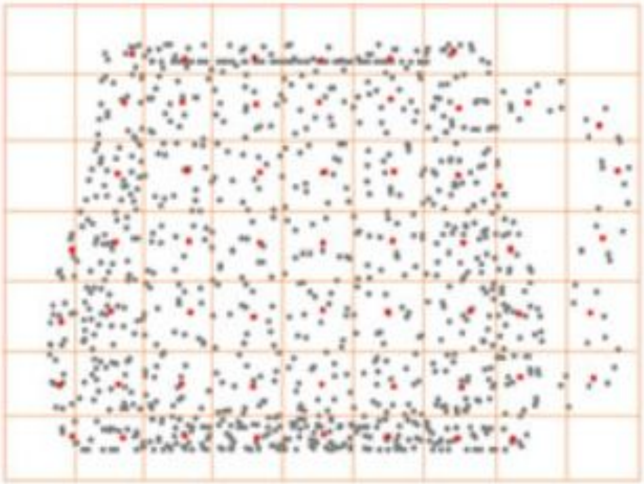
$$\phi = \mathrm{u} \cdot \frac{(\boldsymbol{p}_t - \boldsymbol{p}_s)}{d}$$

$$\theta = \arctan(\mathrm{w} \cdot \boldsymbol{n}_t, \mathrm{u} \cdot \boldsymbol{n}_t)$$

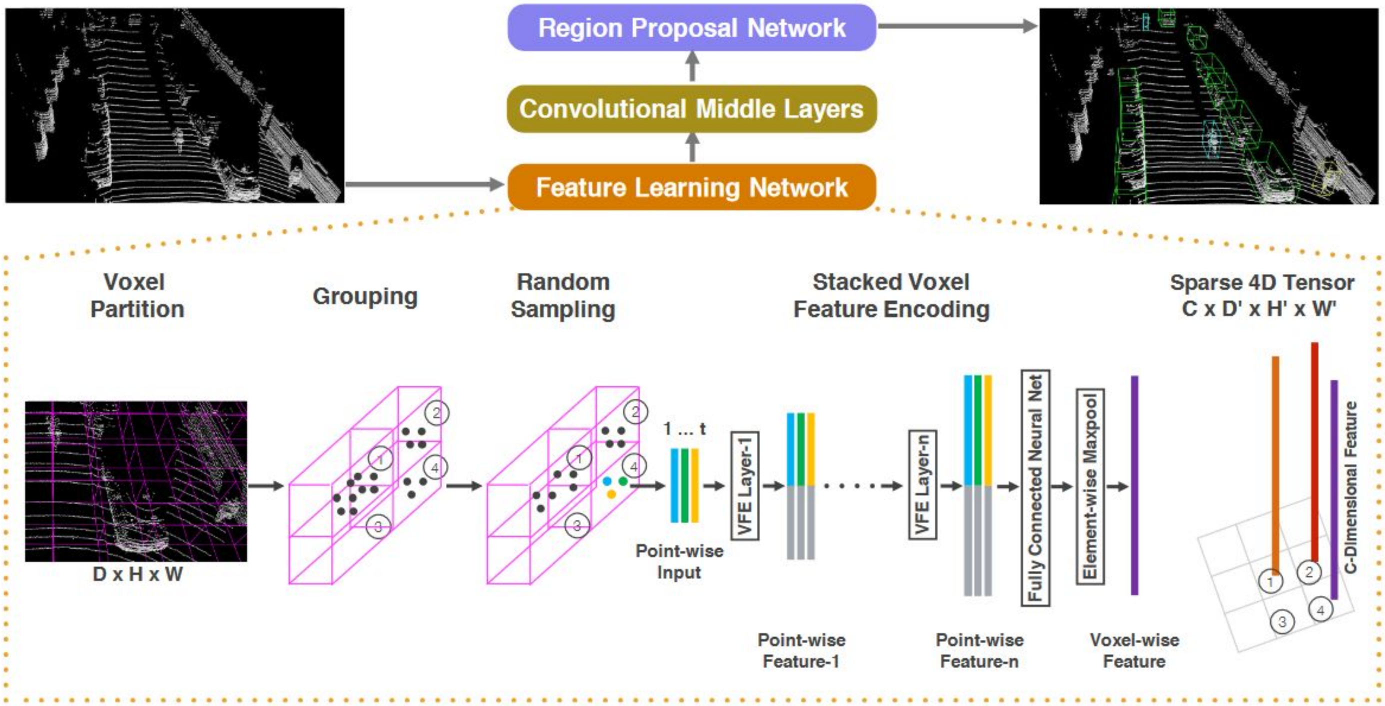
- Binning quadruplets (α,ϕ,θ,d) and indexing to 1-dimension

# Convolutional Operation on Point Cloud

- Cannot be applied directly as on images
  - Voxelize point cloud into grid coordinate
  - Define a fixed grid coordinate for all poincloud
  - 3D convolutional operation > memory & computing cost

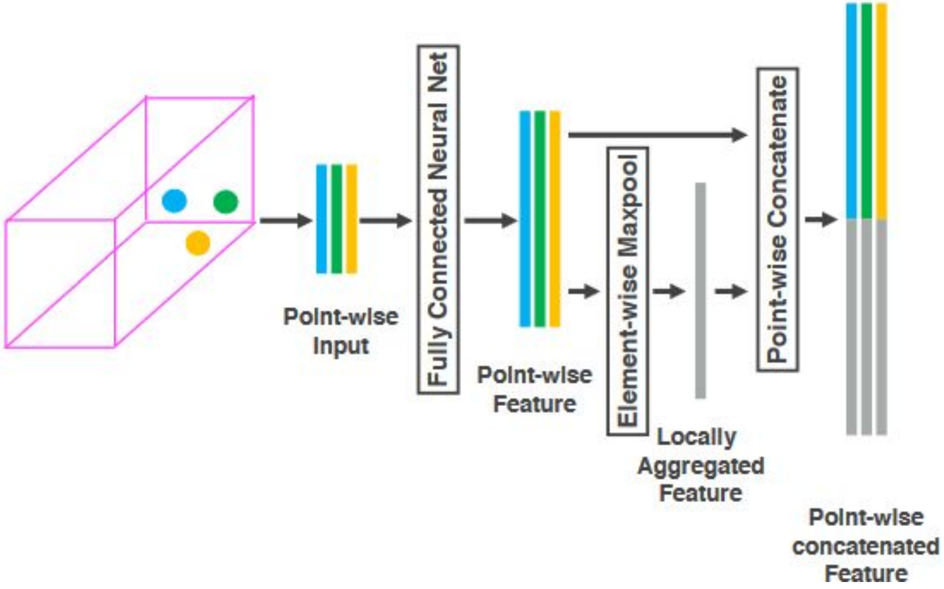


# 3D convolutional operator on 3D voxel grid




*VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection, (Yin Zhou et al)*

# Voxel Feature Encoding (VFE)



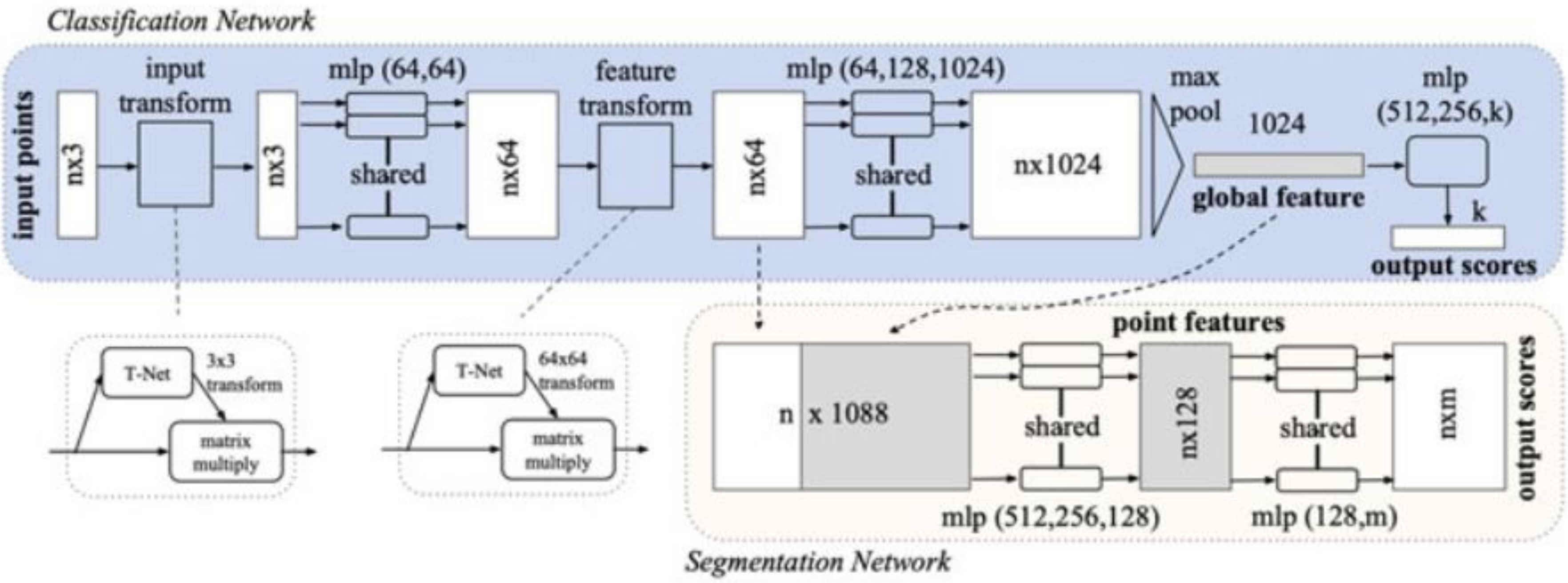
# PointNet

- Consume direct point cloud with **n** points
- Use symmetric function like max-pooling

$$f(x_1, x_2, ..., x_n) = \gamma \left( \underset{i=1,...,n}{\mathrm{MAX}} \{h(x_i)\} \right)$$

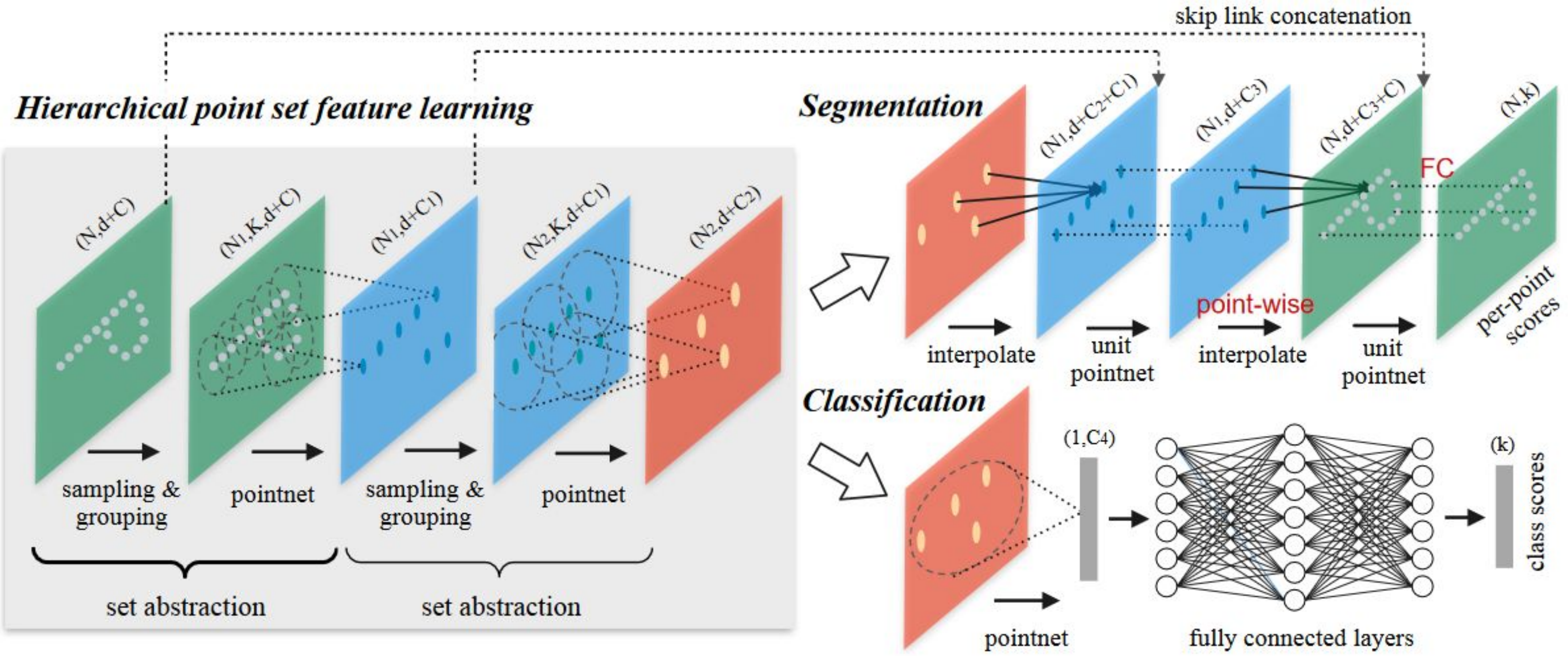
where $\gamma$ and $h$ are usually multi-layer perceptron (MLP) networks

# Architecture

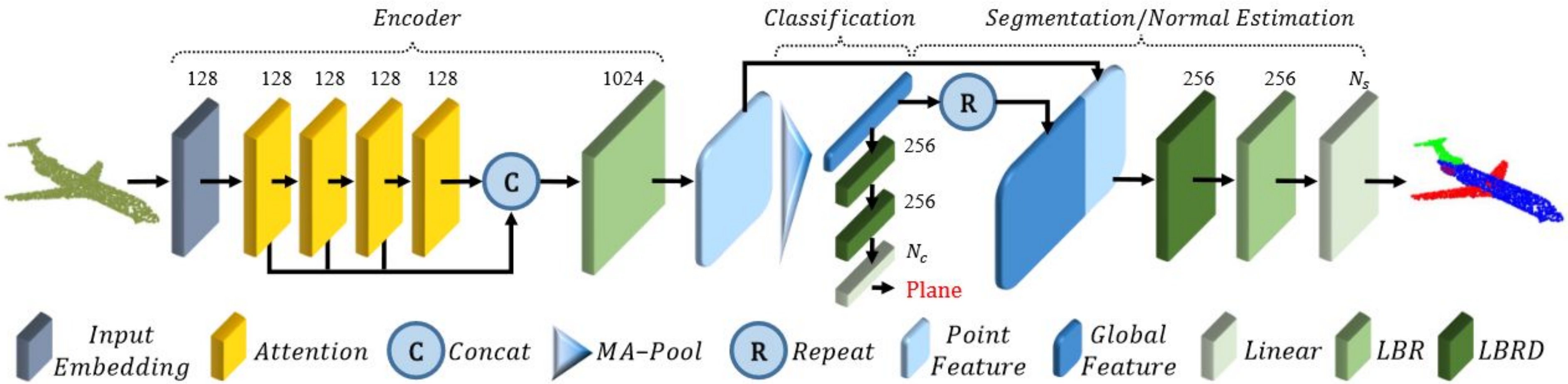


# PointNet++

- Hierarchical Point Set Feature Learning
  - Multi-scale grouping (MSG)
  - Multi-resolution grouping (MRG)



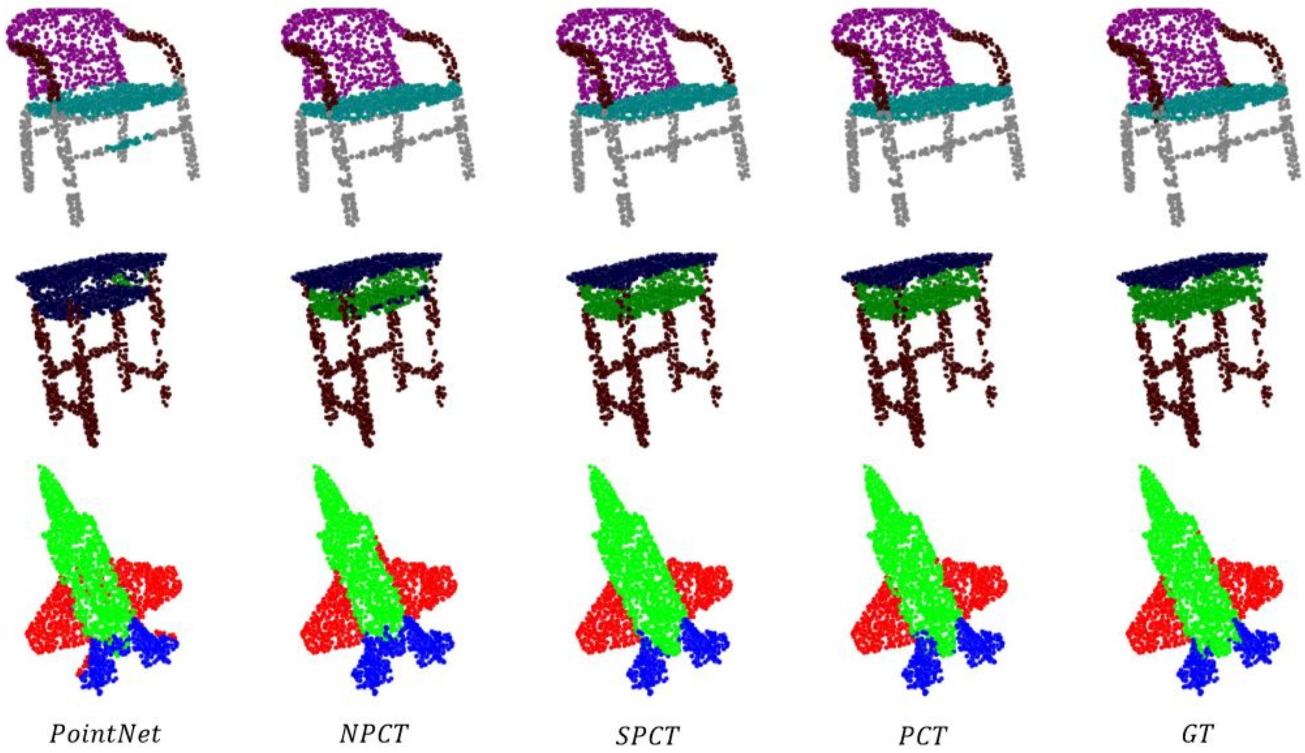
# Point Transformer



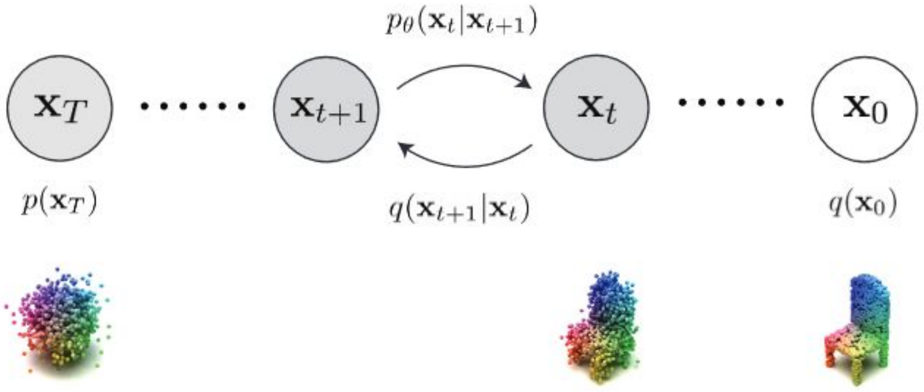

LBR: Linear, Batch, ReLu
LBRD: Linear, Batch, ReLu, DropOut

*PCN: Point Cloud Transformer, Guo et al, 2021*

# Some results



# Generative Point Cloud (Diffusion Approach)




$$q(\mathbf{x}_{0:T}) = q(\mathbf{x}_0)\prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}),$$ **forward**

$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T)\prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t),$$ **backward**

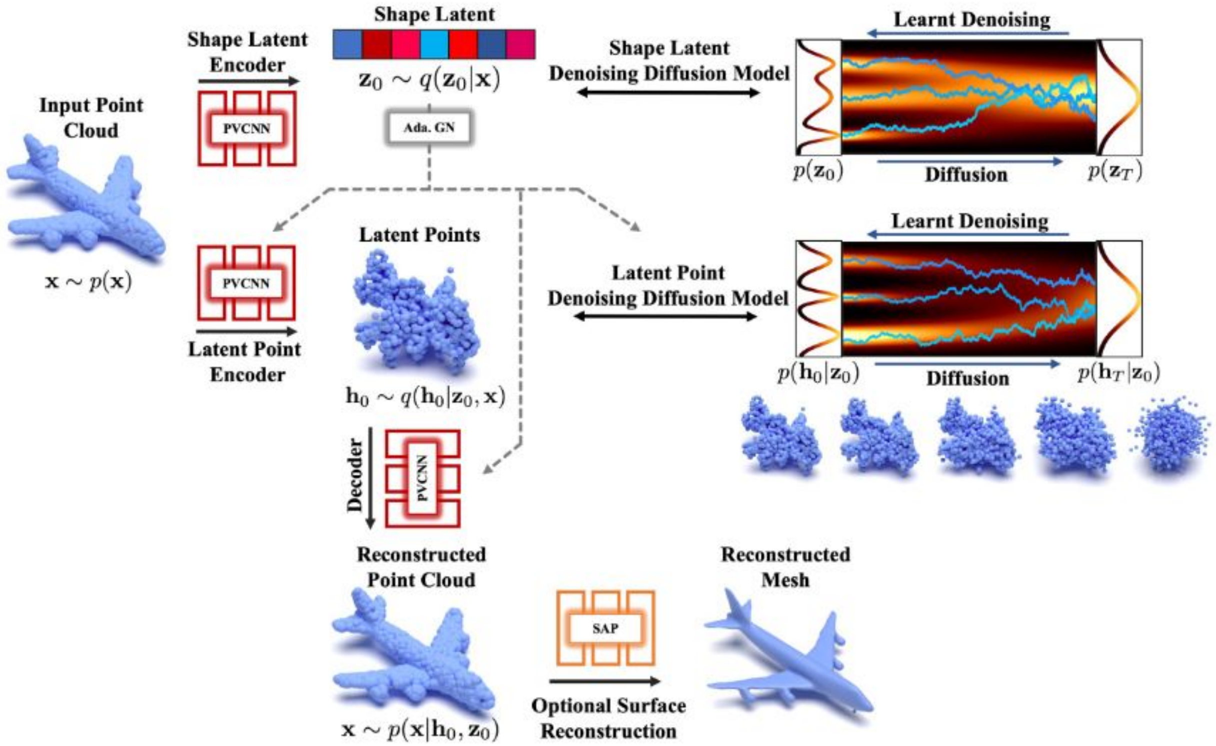
$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) := \mathcal{N}(\sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}),$$

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) := \mathcal{N}(\mu_\theta(\mathbf{x}_t, t), \sigma_t^2\mathbf{I}).$$

$$\mathrm{E}_{q(\mathbf{x}_0)}[\log p_\theta(\mathbf{x}_0)] \geq \mathrm{E}_{q(\mathbf{x}_{0:T})}\left[\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right]$$

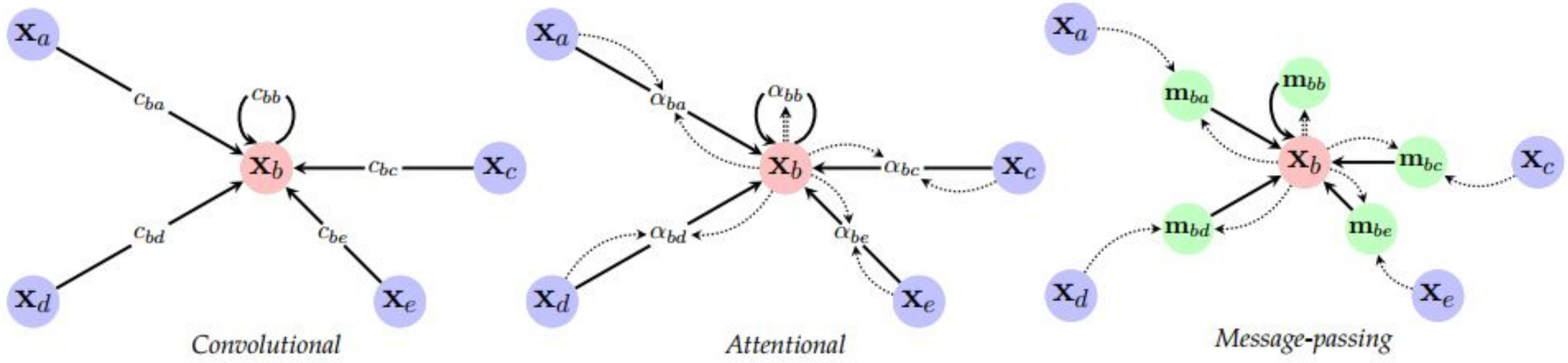
$$\max_\theta \mathrm{E}_{\mathbf{x}_0\sim q(\mathbf{x}_0), \mathbf{x}_{1:T}\sim q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\sum_{t=1}^{T}\log p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)\right]$$

# Latent Point Diffusion




*LION: Latent Point Diffusion Models for 3D Shape Generation, Zeng et al, 2022*

# Advanced: Graph Neural Network on Point Cloud



# Formulation

Graph Convolutional

$$\mathbf{h}_u = \phi\left(\mathbf{x}_u, \bigoplus_{v \in \mathcal{N}_u} c_{uv}\psi(\mathbf{x}_v)\right)$$

Attention

$$\mathbf{h}_u = \phi\left(\mathbf{x}_u, \bigoplus_{v \in \mathcal{N}_u} a(\mathbf{x}_u, \mathbf{x}_v)\psi(\mathbf{x}_v)\right)$$

Message-passing

$$\mathbf{h}_u = \phi\left(\mathbf{x}_u, \bigoplus_{v \in \mathcal{N}_u} \boxed{\psi(\mathbf{x}_u, \mathbf{x}_v)}\right)$$

⊕: Aggregation, permutation-invariant function
Ψ: Point Feature Transform
ϕ : Point Feature Propagation (Diffusion)

$$\psi(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b};$$
$$\phi(\mathbf{x}, \mathbf{z}) = \sigma\left(\mathbf{W}\mathbf{x} + \mathbf{U}\mathbf{z} + \mathbf{b}\right),$$

# Geometric Pytorch

latest

Search docs

INSTALL PYG

Installation

GET STARTED

Introduction by Example

Colab Notebooks and Video Tutorials

TUTORIALS

Design of Graph Neural Networks

Working with Graph Datasets

Use-Cases & Applications

Distributed Training

ADVANCED CONCEPTS

Advanced Mini-Batching

Memory-Efficient Aggregations

Hierarchical Neighborhood Sampling

Compiled Graph Neural Networks

TorchScript Support

PyG Documentation

Note

This documentation is for an **unreleased development version**. Click here to access the documentation of the current stable release.

## PyG Documentation

**PyG** *(PyTorch Geometric)* is a library built upon PyTorch to easily write and train Graph Neural Networks (GNNs) for a wide range of applications related to structured data.

It consists of various methods for deep learning on graphs and other irregular structures, also known as geometric deep learning, from a variety of published papers. In addition, it consists of easy-to-use mini-batch loaders for operating on many small and single giant graphs, multi GPU-support, torch.compile support, DataPipe support, a large number of common benchmark datasets (based on simple interfaces to create your own), the GraphGym experiment manager, and helpful transforms, both for learning on arbitrary graphs as well as on 3D meshes or point clouds.

Join our Slack community!

### Install PyG

- Installation

### Get Started

- Introduction by Example
- Colab Notebooks and Video Tutorials

## Point Cloud Processing

This tutorial explains how to leverage Graph Neural Networks (GNNs) for operating and training on point cloud data. Although point clouds do not come with a graph structure by default, we can utilize PyG transformations to make them applicable for the full suite of GNNs available in PyG. The key idea is to create a synthetic graph from point clouds, from which we can learn meaningful local geometric structures via a GNN's message passing scheme. These point representations can then be used to, *e.g.*, perform point cloud classification or segmentation.

### 3D Point Cloud Datasets

PyG provides several point cloud datasets, such as the `PCPNetDataset`, `S3DIS` and `ShapeNet` datasets. To get started, we also provide the `GeometricShapes` dataset, which is a toy dataset that contains various geometric shapes such cubes, spheres or pyramids. Notably, the `GeometricShapes` dataset contains meshes instead of point clouds by default, represented via `pos` and `face` attributes, which hold the information of vertices and their triangular connectivity, respectively:

```
from torch_geometric.datasets import GeometricShapes

dataset = GeometricShapes(root='data/GeometricShapes')
print(dataset)
>>> GeometricShapes(40)

data = dataset[0]
print(data)
>>> Data(pos=[32, 3], face=[3, 30], y=[1])
```

# Hand-on project

- PointNet
    - Classification
- Visualization:
    - Inference classification