

3D Point Clouds

Lecture 1 –

Introduction

主讲人 黎嘉信

Aptiv 自动驾驶
新加坡国立大学 博士
清华大学 本科





1、Course Introduction



2、Principle Component



3、Downsampling



4、Filtering



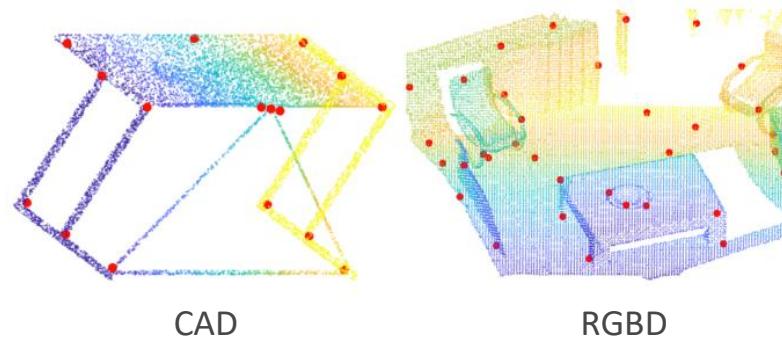
3D Point Cloud



Representation

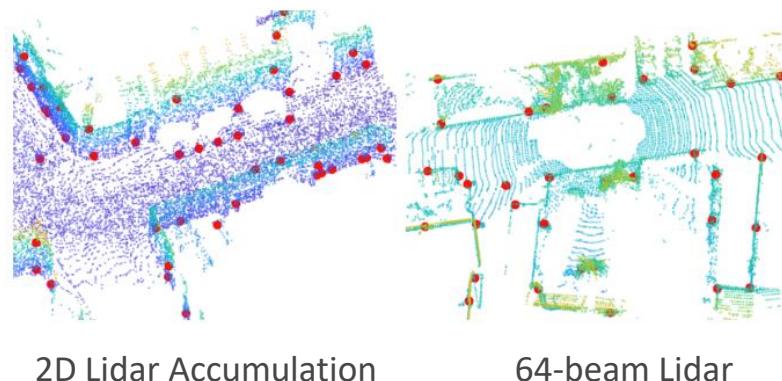
- $N \times 3$ matrix or $N \times (3 + D)$ matrix

$$\begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots \\ x_N & y_N & z_N \end{bmatrix} \quad \begin{bmatrix} x_1 & y_1 & z_1 & d_{11} & \cdots & d_{1D} \\ x_2 & y_2 & z_2 & d_{21} & \cdots & d_{2D} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ x_N & y_N & z_N & d_{N1} & \cdots & d_{ND} \end{bmatrix}$$



Data sources:

- Lidar
- RGB-D
- CAD Models
- Structure-from-Motion (SfM)
- Depth from Images
-





Applications of Point Clouds



Robotics, Autonomous driving

- Localization – SLAM, loop closure, registration
- Perception – object detection, classification
- Reconstruction – SfM, registration



Consumer Electronics

- Face detection / reconstruction – FaceID
- Hand pose – Hololens
- Human pose – Kinect





Applications

<https://www.youtube.com/watch?v=27OuOCeZmwl>

Our Road to Self-Driving Vehicles | Uber ATG



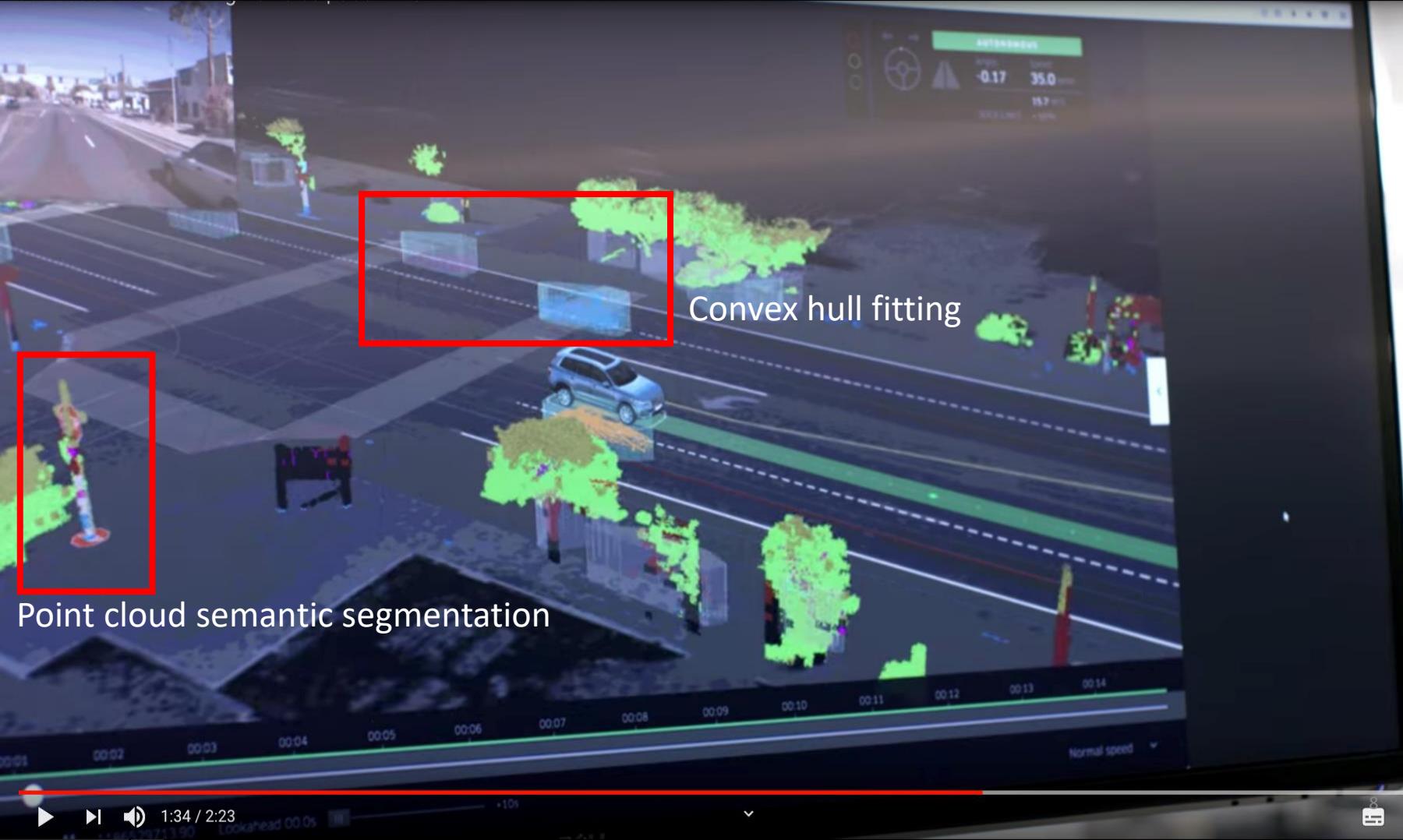


Velodyne 64

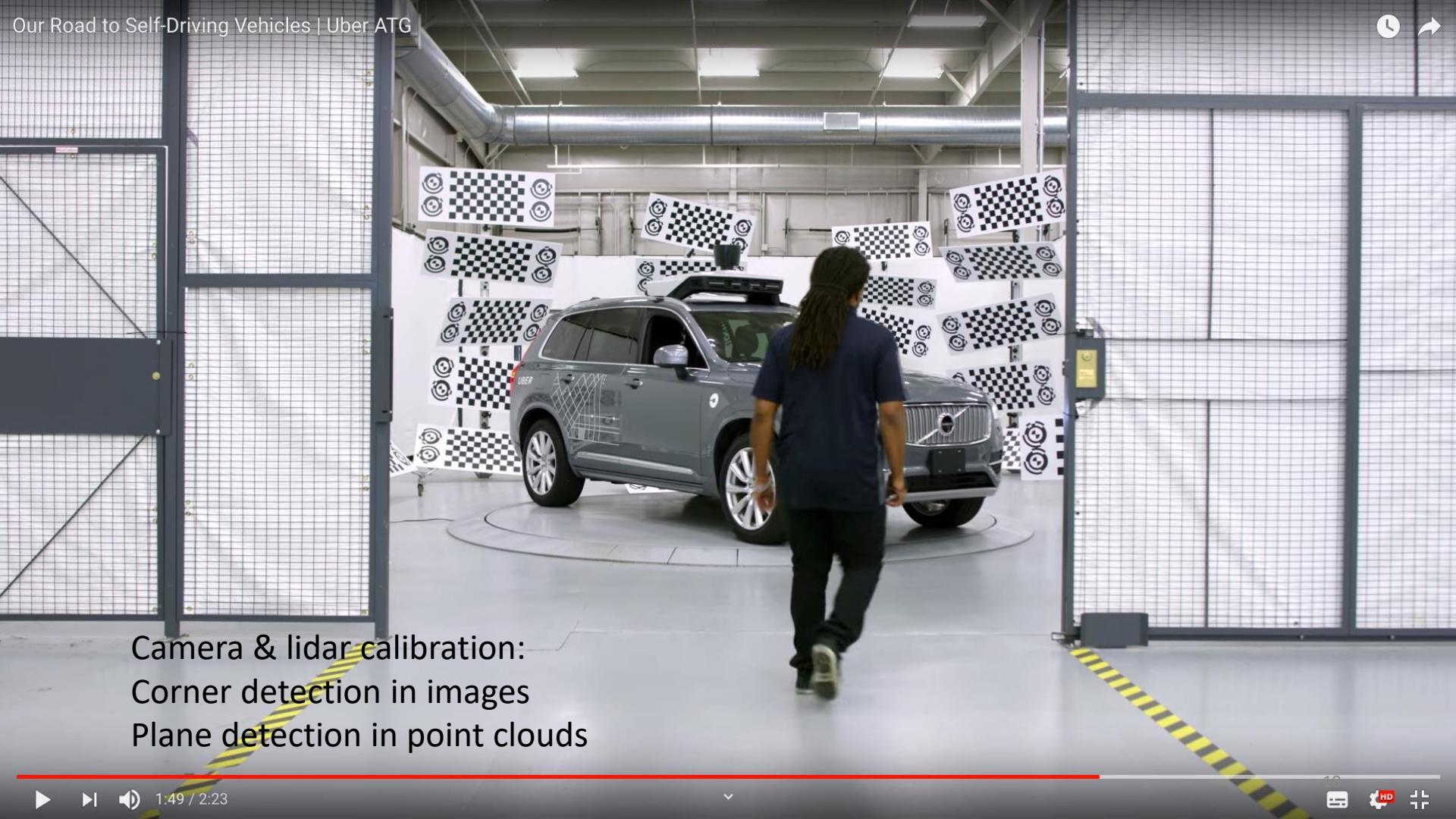




1. Object detection – orange points
2. Clustering
3. Filtering based on map



```
from flann import *****  
Nearest Neighbor Search
```



Camera & lidar calibration:
Corner detection in images
Plane detection in point clouds

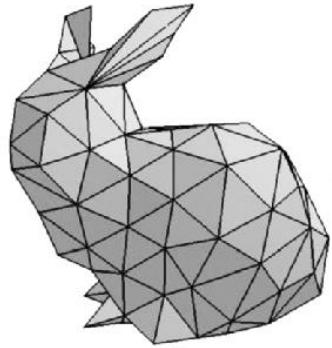


Advantages & Difficulties

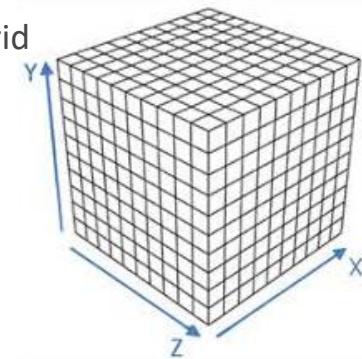


What are the options for 3D?

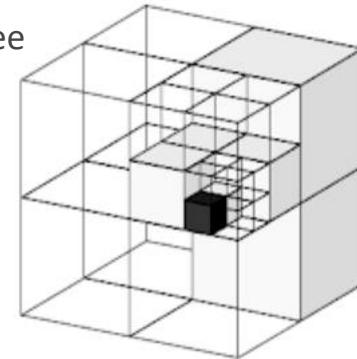
Mesh



Voxel Grid



Octree



Strength of Point Cloud

- 3D information
- Mathematically simple and concise

$$\begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots \\ x_N & y_N & z_N \end{bmatrix}$$



Advantages & Difficulties



Difficulties of Point Cloud Processing

- Sparsity
- Irregular – difficulty in neighbor searching
- Lack of texture information
- Un-ordered – difficulty in deep learning
- Rotation equivariance / invariance

$$\begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots \\ x_N & y_N & z_N \end{bmatrix} = \begin{bmatrix} x_2 & y_2 & z_2 \\ x_1 & y_1 & z_1 \\ \vdots & \vdots & \vdots \\ x_k & y_k & z_k \end{bmatrix}$$





Course Outline



Classical Methods – 60%

- Pros
 - Explainable – It follows physics and we know why it works/doesn't work
 - Controllable – We know how to debug
- Cons
 - Hard to model semantics
 - User-unfriendly



Deep Learning Methods – 40%

- Pros
 - Simple!
 - High performance
 - Data driven
- Cons
 - Un-explainable – No one knows why / how.
 - Un-controllable – Black box
 - Requires special hardware – GPU / FPGA, etc.
 - Simple – The barrier is lower and lower means it will be more and more difficult to find a job.



Classical v.s. Deep Learning



Object Classification

- Classical
 - Keypoint detection
 - Keypoint description
 - Support Vector Machine
- Deep learning
 - Data collection
 - Data labeling
 - Train a network



Object Registration

- Classical
 - Nearest Neighbor Search
 - SVD
 - Iterative Closest Point
- Deep learning
 - Data collection
 - Data labeling
 - Train a network



Object Detection

- Classical
 - Background removal
 - Clustering
 - Classification
- Deep learning
 - Data collection
 - Data labeling
 - Train a network



Course Schedule

Week	Topic
1	Introduction to point clouds, PCA, downsampling and filtering
2	Nearest neighbor algorithms – kd-tree & octree
3	Clustering algorithms
4	Model fitting – Hough Transform, Sample Consensus
5	Deep Learning with point clouds
6	Object detection
7	Keypoints and descriptors
8	Registration
9	2D and 3D SLAM with point clouds



Course Outline

- Theory = you know everything but nothing works.
- Practice = everything works but no one knows why.
- Theory + practice = nothing works and no one knows why.

- This module is around 30% theory and 70% practice
 - Theory – proof, derivation
 - Practice
 - Algorithms
 - Engineering practices
 - Assignments are mostly coding



Prerequisite



Basic Linear Algebra

- Eigen decomposition, SVD, PCA



Basic Probability Theory



Computer Vision

- Camera models



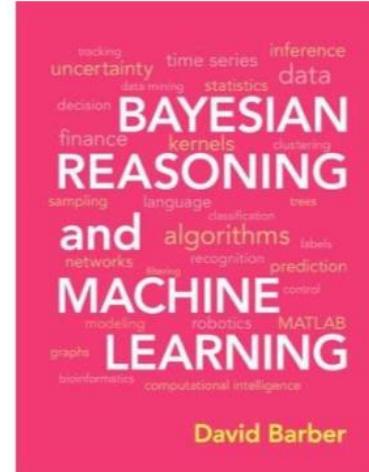
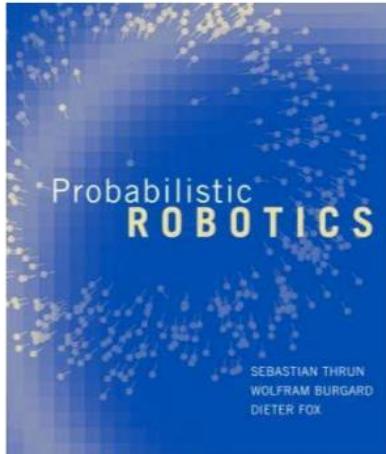
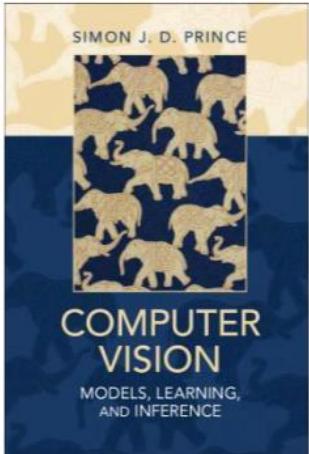
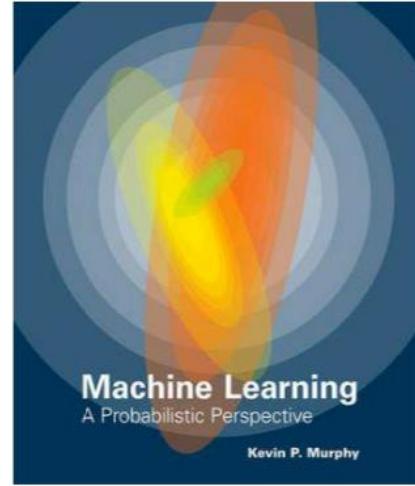
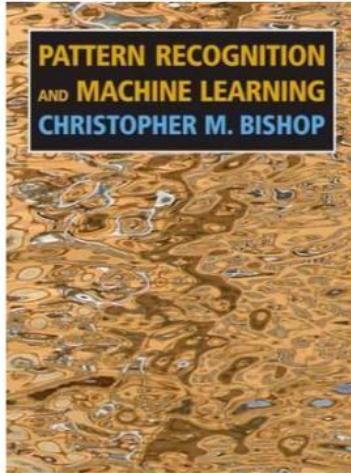
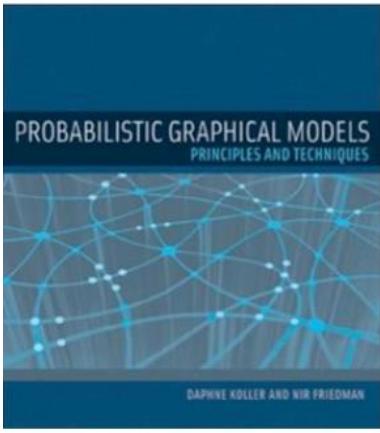
Simple Optimization

- Gradient Descent
- Least Square
 - Gauss-Newton
 - Levenberg Marquardt



Deep learning

- Gradient descent optimizer
 - SGD, Adam, etc.
- Multi-Layer Perceptron (MLP)
- 2D / 3D Convolution
- Activation Function
 - The idea of non-linearity
- Normalization
 - Batch normalization



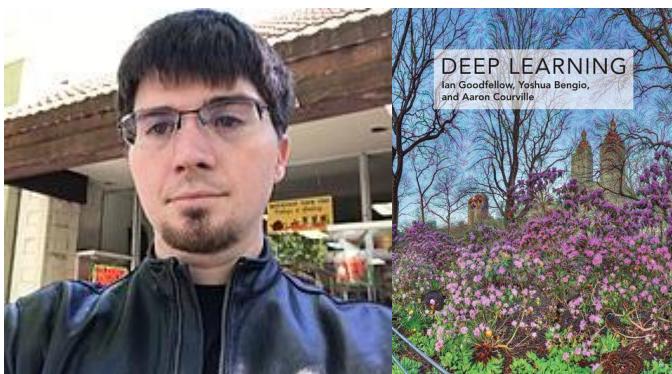


Course Outline - Theory



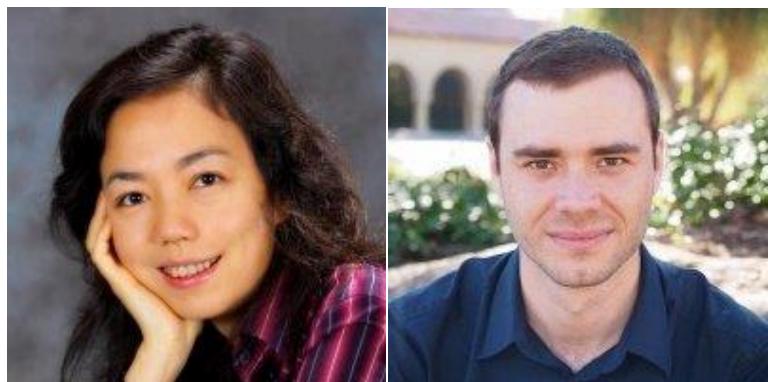
Deep Learning

- Ian Goodfellow, Yoshua Bengio,
Aaron Courville
- <https://www.deeplearningbook.org/>



Stanford CS231n

- Li Fei-fei, Justin Johnson,
Andrej Karpathy, etc
- <http://cs231n.stanford.edu/>





Common Tools for Practice



C++

- Point Cloud Library (PCL)
- Python Binding – pybind11
- Optimization Solver – g2o, Ceres
- Eigen



pybind11



Python

- numpy
- scipy
- Open3D
- Pytorch
- Tensorflow



NumPy

PYTORCH



TensorFlow





Assignments



Weekly Assignments

- Some assignments are compulsory
- Some are optional, for those who are interested.



Big Projects

- Mid-term project
 - Ground estimation on KITTI dataset
- Final project
 - Object detection by Deep Learning and classical methods
 1. Use deep learning based object detector
 2. Use classical method (classification can be deep learning based)
 3. Try to combine both

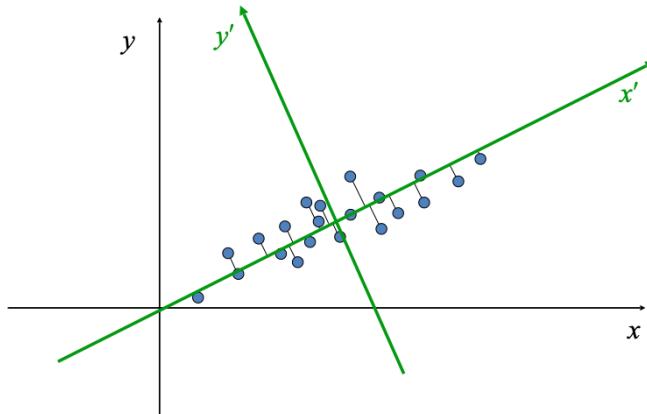
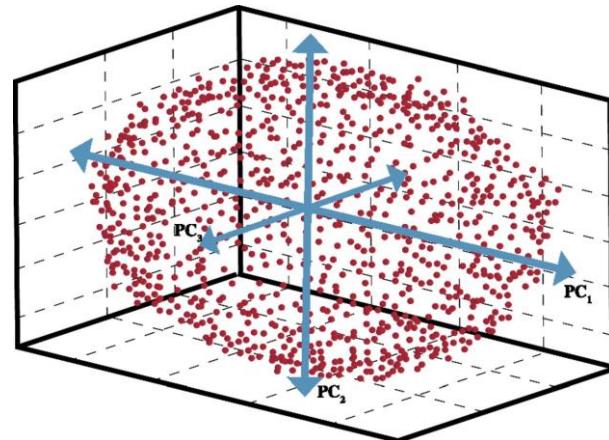


Principle Component Analysis

PCA is to find the dominant directions of the point cloud

Applications:

- Surface normal estimation
- Canonical orientation
- Keypoint detection
- Feature description



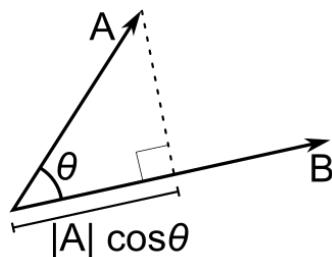


Principle Component Analysis



Let's start with some physical intuitions.

Vector Dot Product

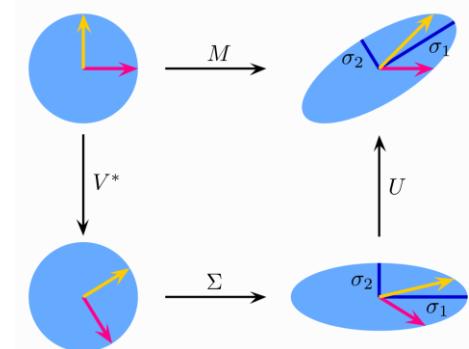


Matrix-Vector Multiplication

$$\begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{32} \\ \vdots & \vdots & \vdots \\ x_{1N} & a_{2N} & a_{3N} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = [\mathbf{a}_1 \quad \mathbf{a}_2 \quad \mathbf{a}_3] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$
$$= x_1 \mathbf{a}_1 + x_2 \mathbf{a}_2 + x_3 \mathbf{a}_3$$

Linear combination

Singular Value Decomposition (SVD)



$$M = U \cdot \Sigma \cdot V^*$$



Spectral Theorem

Let $A \in R^{n,n}$ be symmetric, and $\lambda_i \in R, i = 1, 2, \dots, n$ be the eigenvalues of A. There exists a set of orthonormal vectors $u_i \in R_n, i = 1, 2, \dots, n$, such that $Au_i = \lambda_i u_i$. Equivalently, there exists an orthogonal matrix $U = [u_1, \dots, u_n]$ (i.e., $UU^T = U^T U = I_n$), such that,

$$A = U\Lambda U^T = \sum_{i=1}^n \lambda_i u_i u_i^T, \Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$$



Rayleigh Quotients

Physical meaning of SVD!

Given a symmetric matrix $A \in S^n$,

$$\lambda_{\min}(A) \leq \frac{x^T A x}{x^T x} \leq \lambda_{\max}(A), \forall x \neq 0$$

$$\lambda_{\max}(A) = \max_{x: \|x\|_2=1} x^T A x$$

$$\lambda_{\min}(A) = \min_{x: \|x\|_2=1} x^T A x$$

The maximum and minimum are attained for $x = u_1$ and for $x = u_n$, respectively, where u_1 and u_n are the largest and smallest eigenvector of A , respectively.



Rayleigh Quotients – Proof:

Denote $A = U\Lambda U^T$ as the spectral factorization of A , where Λ contains the ordered eigenvalues on the diagonal, i.e., $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$. U is orthogonal, i.e., $UU^T = U^T U = I_n$. Define $\bar{x} = U^T x$, we have

$$x^T A x = x^T U \Lambda U^T x = \bar{x}^T \Lambda \bar{x} = \sum_{i=1}^n \lambda_i \bar{x}_i^2 \quad (1)$$

It is obvious that

$$\lambda_{\min} \sum_{i=1}^n \bar{x}_i^2 \leq \sum_{i=1}^n \lambda_i \bar{x}_i^2 \leq \lambda_{\max} \sum_{i=1}^n \bar{x}_i^2 \quad (2)$$

Note that,

$$\sum_{i=0}^n x_i^2 = x^T x = x^T U U^T x = (U^T x)^T (U^T x) = \bar{x}^T \bar{x} = \sum_{i=0}^n \bar{x}_i^2 \quad (3)$$

From Eq. 1, 2 and 3, we have

$$\lambda_{\min} x^T x \leq x^T A x \leq \lambda_{\max} x^T x$$

The upper and lower bound can be achieved by $x = u_1$ for $\lambda_{\max} = \lambda_1$ and $x = u_n$ for $\lambda_{\min} = \lambda_n$.



Principle Component Analysis



Q: What is the most significant principle component?

A: A direction such that the variance of the projected data points on that direction is maximal.



Q: How to get the second significant one?

A: Deflation. Remove the most significant component from the data points, i.e., data point minus the projection. Find the most significant component for the deflated data.



Q: How to get the 3rd one?

A: Repeat the above steps.



Principle Component Analysis - Definition

Denote data points as $x_i \in R^n, i = 1, 2, \dots, m$. The mean is $\bar{x} = \frac{1}{m} \sum_{i=1}^m x_i$. Now we represent the centered data points by a $n \times m$ matrix.

$$\tilde{X} = [\tilde{x}_1, \dots, \tilde{x}_m], \tilde{x}_i = x_i - \bar{x}, i = 1, \dots, m$$

The target is to look for a direction $z \in R^n, \|z\|_2 = 1$, so that the variance of \tilde{X} projected onto z is maximized. The projection of \tilde{x}_i onto z is given by dot product,

$$\alpha_i = \tilde{x}_i^T z, i = 1, \dots, m$$

Note that α_i is the projection of \tilde{x}_i along z . The mean square variance of such projections is given by,

$$\frac{1}{m} \sum_{i=1}^m \alpha_i^2 = \frac{1}{m} \sum_{i=1}^m z^T \tilde{x}_i \tilde{x}_i^T z = z^T \tilde{X} \tilde{X}^T z$$



Principle Component Analysis - Proof

Now the problem is turned into an optimization problem:

$$\max_{z \in R^n} z^T (\tilde{X} \tilde{X}^T) z, \text{ s.t.: } \|z\|_2 = 1$$

This problem can be solved via SVD of \tilde{X} . Let's denote the compact SVD of \tilde{X} , where $r = \text{rank}(\tilde{X})$,

$$\tilde{X} = U_r \Sigma V_r^T = \sum_{i=1}^r \sigma_i u_i v_i^T$$

Hence,

$$H = \tilde{X} \tilde{X}^T = U_r \Sigma^2 U_r^T$$

Note that Σ is a diagonal matrix that contains the singular values of \tilde{X} . In addition, the above equation is actually the spectral factorization of H , which means that Σ^2 contains the eigenvalues of H in the diagonal. According to the **Rayleigh Quotients**, the solution to the above optimization is,

$$z = u_1, \text{ mean square variance along } z \text{ is } \sigma_1^2$$



Principle Component Analysis - Deflation

Now we proceed to find the second-largest variation direction, by *deflate* the data points, i.e., removing the first-largest components,

$$\begin{aligned}\tilde{x}_i^{(1)} &= \tilde{x}_i - u_1(u_1^T \tilde{x}_i), i = 1, \dots, m \\ \tilde{X}^{(1)} &= [\tilde{x}_1^{(1)}, \dots, \tilde{x}_m^{(1)}] = (I_n - u_1 u_1^T) \tilde{X}\end{aligned}$$

Applying compact SVD of $\tilde{X} = \sum_{i=1}^r \sigma_i u_i v_i^T$, we have:

$$\begin{aligned}\tilde{X}^{(1)} &= \sum_{i=1}^r \sigma_i u_i v_i^T - (u_1 u_1^T) \sum_{i=1}^r \sigma_i u_i v_i^T \\ &= \sum_{i=1}^r \sigma_i u_i v_i^T - \sum_{i=1}^r \sigma_i u_1 u_1^T u_i v_i^T \\ &= \sum_{i=1}^r \sigma_i u_i v_i^T - \sigma_1 u_1 v_1^T \quad // U \text{ is orthogonal} \\ &= \sum_{i=2}^r \sigma_i u_i v_i^T\end{aligned}$$



Principle Component Analysis - Deflation

Now let's recall that the finding second-largest direction of variation z equals to solving the optimization problem,

$$\max_{z \in R^n} z^T (\tilde{X}^{(1)} \tilde{X}^{(1)T}) z, \text{ s.t.: } \|z\|_2 = 1$$

According to Rayleigh quotients, we know that the maximum is achieved when $z = u_2$, and the second-largest variation is σ_2 .

- Now we know that by deflation we can get direction of second-largest variation
- Similarly, we can get the rest of the directions in a decreasing order



PCA - Summary



Given $x_i \in R^n, i = 1, 2, \dots, m.$, perform PCA by:

1. Normalized by the center

$$\tilde{X} = [\tilde{x}_1, \dots, \tilde{x}_m], \tilde{x}_i = x_i - \bar{x}, i = 1, \dots, m \quad \bar{x} = \frac{1}{m} \sum_{i=1}^m x_i.$$

2. Compute SVD $H = \tilde{X}\tilde{X}^T = U_r\Sigma^2U_r^T$
3. The principle components are the columns of U_r



Surface Normal



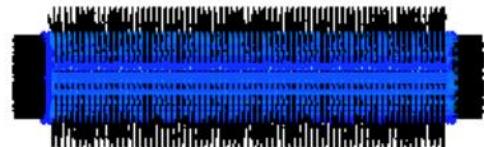
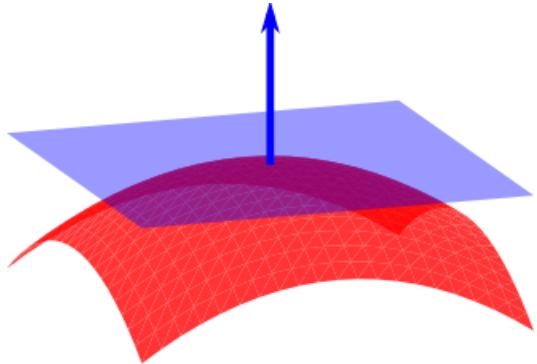
Surface normal on surface

- The vector perpendicular to the tangent plane of the surface at a point P



Applications

- Segmentation / Clustering
- Plane detection
- Point cloud feature for applications like Deep Learning





Surface Normal – How to compute



Surface normal on 3D point cloud

1. Select a point P
2. Find the neighborhood that defines the surface
3. PCA
4. Normal -> the least significant vector
5. Curvature -> ratio between eigen values $\lambda_3/(\lambda_1 + \lambda_2 + \lambda_3)$



Intuitively it is obvious, can we prove it formally?

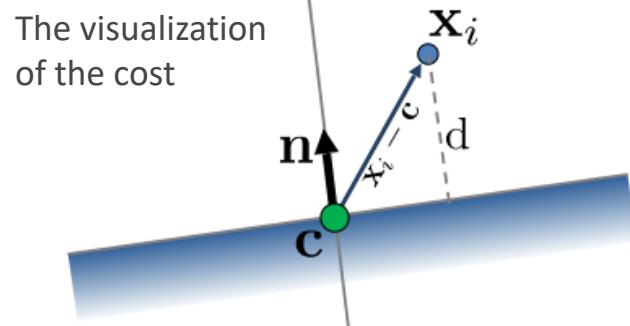
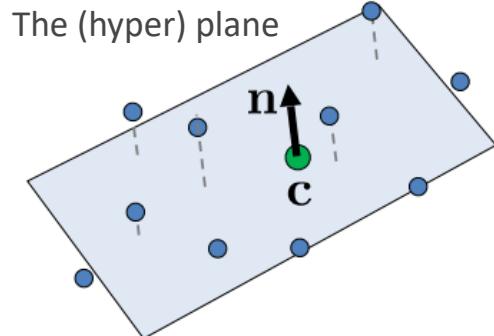


Surface Normal Estimation – Definition

Problem Definition

Denote data points as $x_i \in R^n, i = 1, 2, \dots, m$, find a (hyper) plane, that passes through a **point c** with **normal vector n**, s.t.

$$\min_{\mathbf{c}, \mathbf{n}, \|\mathbf{n}\|=1} \sum_{i=1}^m ((\mathbf{x}_i - \mathbf{c})^T \mathbf{n})^2$$





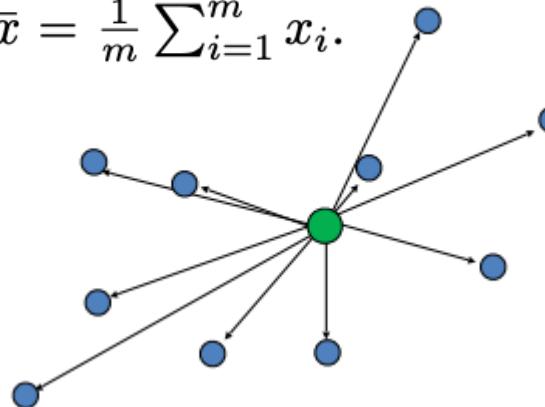
Surface Normal Estimation – Proof

Since c and n are independent variables, let's look at c first

$$\mathbf{c}^* = \arg \min_{\mathbf{c}} \sum_{i=1}^m ((\mathbf{x}_i - \mathbf{c})^T \mathbf{n})^2$$

That means \mathbf{c}^* is the center of the data points

$$\bar{\mathbf{x}} = \frac{1}{m} \sum_{i=1}^m \mathbf{x}_i.$$





Surface Normal Estimation – Proof

So we normalize the data points by its center, similar to what we did in PCA proof.

$$\tilde{X} = [\tilde{x}_1, \dots, \tilde{x}_m], \tilde{x}_i = x_i - \bar{x}, i = 1, \dots, m$$

Now the problem becomes,

$$\min_{n \in R^n} \sum_{i=1}^m (\tilde{x}_i^T n)^2, \text{ s.t.: } \|n\|_2 = 1$$



Surface Normal Estimation



What shall we do when there are noise?

1. Select neighbors according to problem

E.g. Radius based neighbors

- a. Radius larger -> normal estimation is smoother, but affected by irrelevant objects
- b. Radius smaller -> normal estimation is sharper, but noisy

2. Weighted based on other features

- a. Lidar intensity
- b. RGB values

3. Deep Learning!



Surface Normal Estimation



Weighted normal estimation

$$\min_{n \in R^n} \sum_{i=1}^m (w_i \tilde{x}_i^T n)^2, \text{ s.t.: } \|n\|_2 = 1$$

$$\min_{n \in R^n} \sum_{i=1}^m w_i n^T \tilde{x}_i \tilde{x}_i^T n, \text{ s.t.: } \|n\|_2 = 1$$

$$\min_{n \in R^n} n^T \left(\sum_{i=1}^m w_i \tilde{x}_i \tilde{x}_i^T \right) n, \text{ s.t.: } \|n\|_2 = 1$$

$$\min_{n \in R^n} n^T \tilde{X} W \tilde{X}^T n, \text{ s.t.: } \|n\|_2 = 1$$

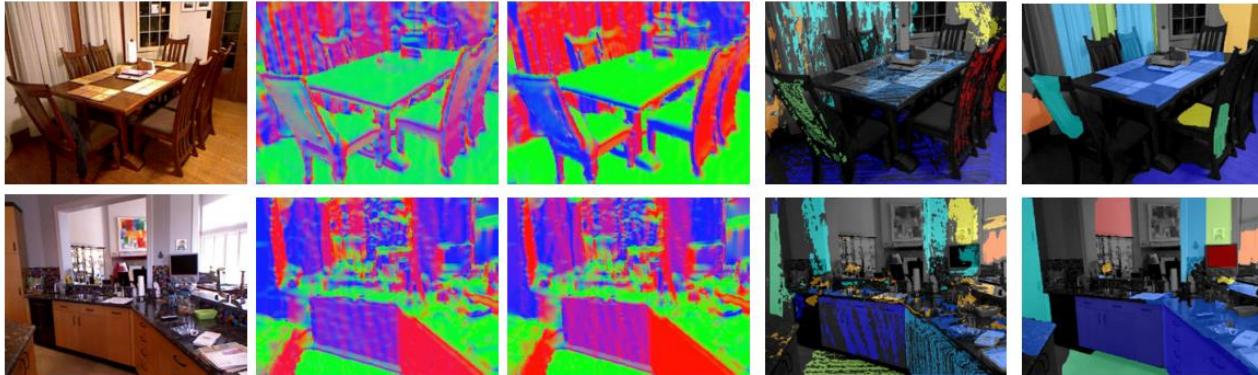
$$W = \begin{bmatrix} w_1 & 0 & \cdots & 0 \\ 0 & w_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & w_m \end{bmatrix}$$



Deep Learning about Surface Normal



NYU Depth V2 dataset



(a) Input RGB

(b) Normals

(c) Aligned

(d) RANSAC Fit

(e) Segmented



ModelNet40





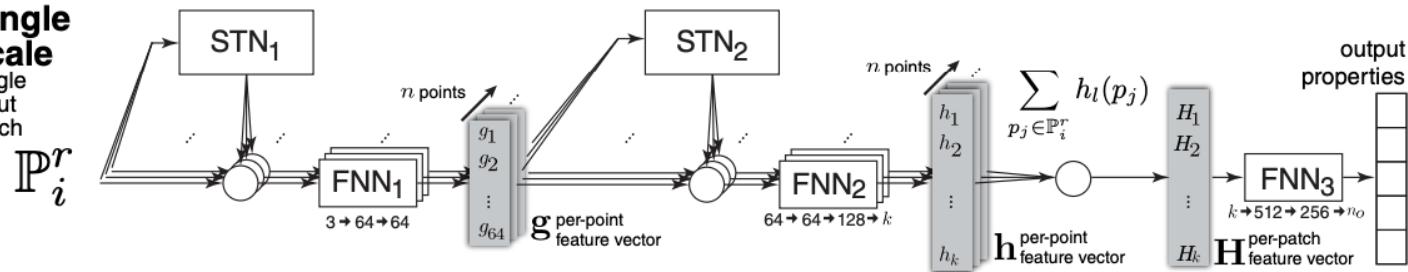
Deep Learning about Surface Normal



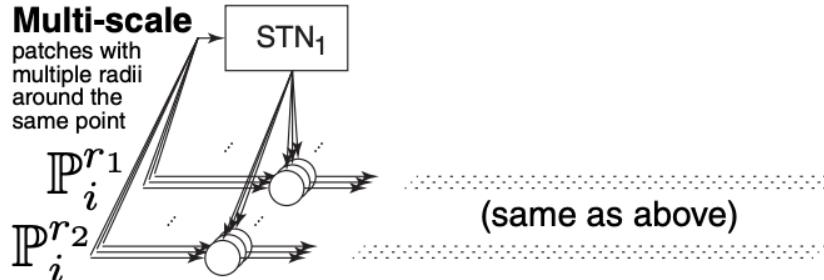
PCPNET Learning Local Shape Properties from Raw Point Clouds

- Eurographics 2018

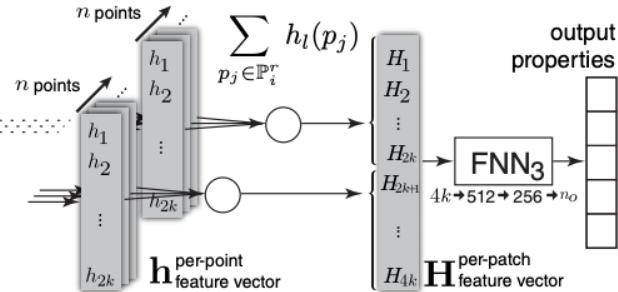
Single scale
single input patch



Multi-scale
patches with multiple radii around the same point



(same as above)

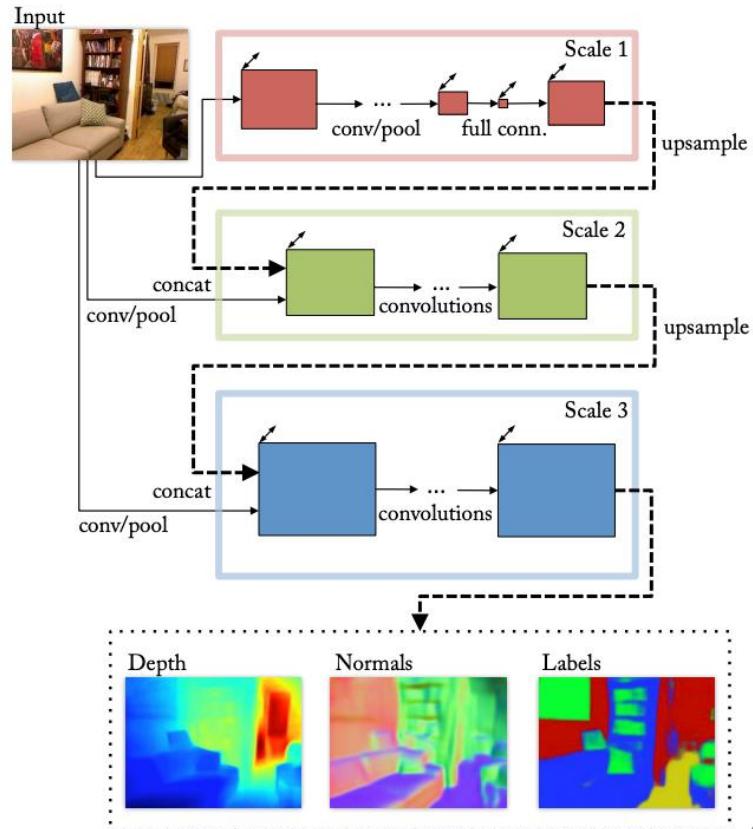




Deep Learning about Surface Normal

Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-Scale Convolutional Architecture

- ICCV 2015
- Joint estimation of depth and surface normal improves the depth result



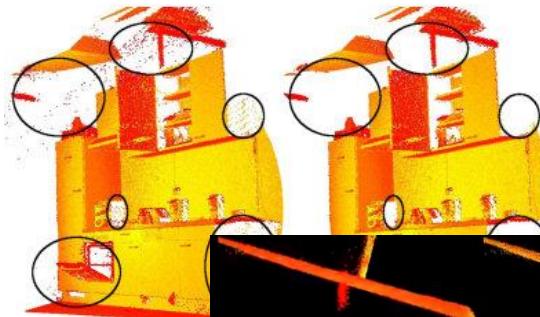


Filters



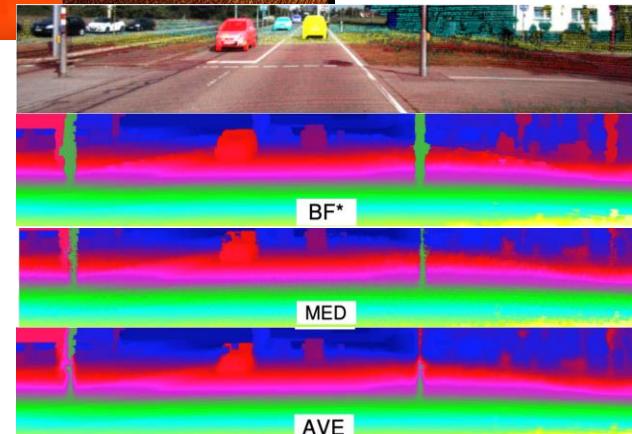
Noise removal

- Radius Outlier Removal
- Statistical Outlier Removal



Downsampling

- Voxel Grid Downsampling
 - Exact / Approximated
 - Centroid / Random Selection
- Farthest Point Sampling
- Normal Space Sampling



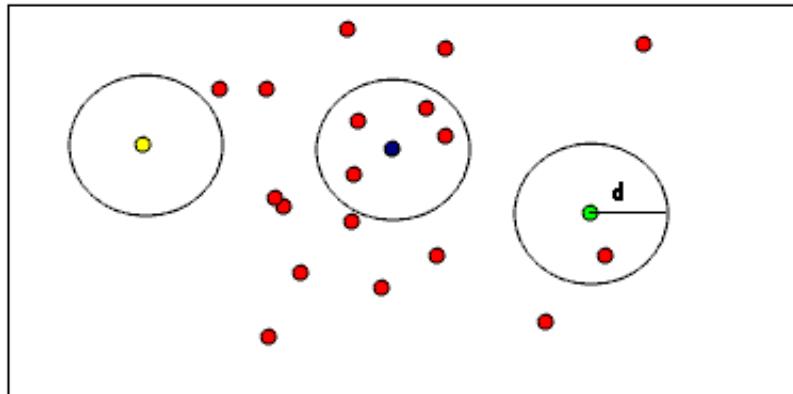
Upsampling / Smoothing / Noise Removal

- Bilateral Filter



Radius Outlier Removal

1. For each point, find a radius = r neighborhood
2. If number of neighbor $k < k^*$, remove the point





Statistical Outlier Removal

1. For each point, find a neighborhood
2. Compute its distance to its neighbors $d_{ij}, i = [1, \dots, m], j = [1, \dots, k]$
3. Model the distances by Gaussian distribution $d \sim N(\mu, \sigma)$

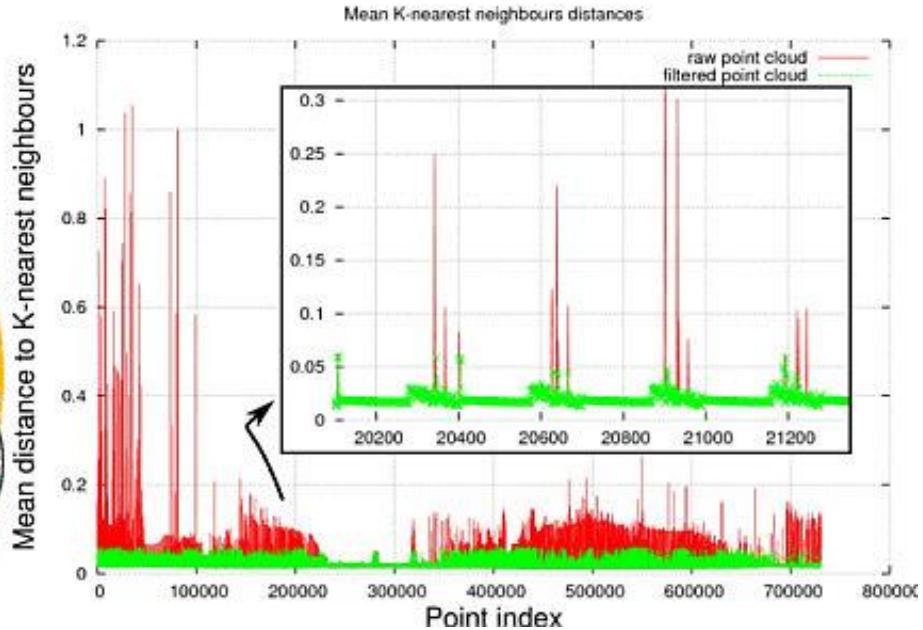
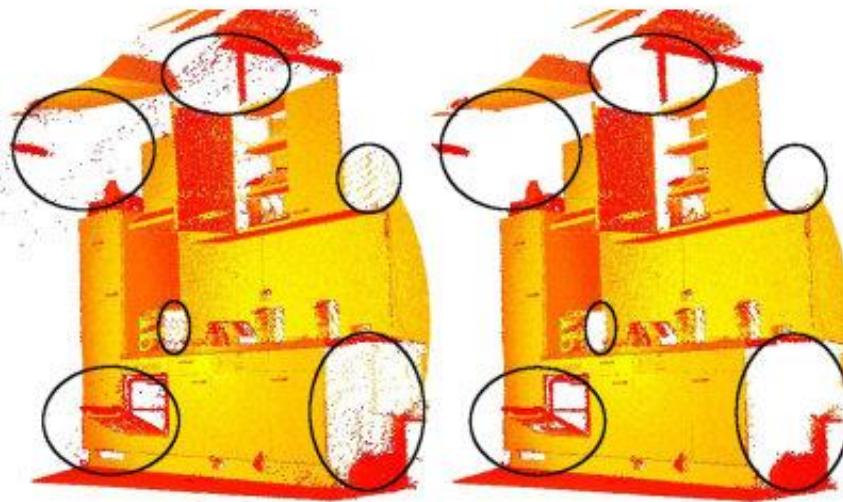
$$\mu = \frac{1}{nk} \sum_{i=1}^m \sum_{j=1}^k d_{ij}, \quad \sigma = \sqrt{\frac{1}{nk} \sum_{i=1}^m \sum_{j=1}^k (d_{ij} - \mu)^2}$$

4. For each point, compute its mean distance to its neighbors
5. Remove the point, if the mean distance is outside some confidence according to the Gaussian distribution
E.g. Remove if

$$\sum_{j=1}^k d_{ij} > \mu + 3\sigma \text{ or } \sum_{j=1}^k d_{ij} < \mu - 3\sigma$$



Statistical Outlier Removal



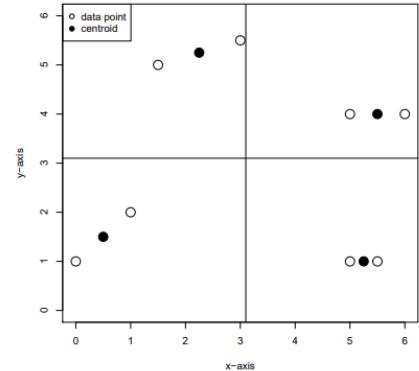
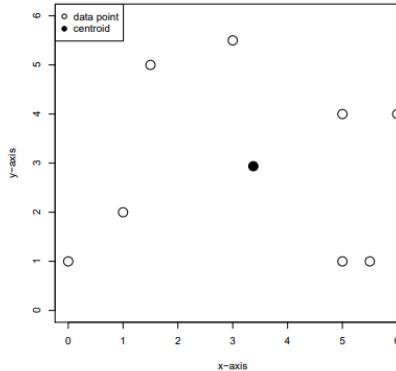
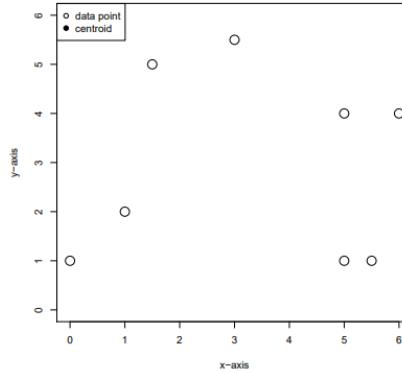


Voxel Grid Downsampling

1. Build a voxel grid that contains the point cloud
2. Take one point in each cell

Q1, how to “take one point”?

Q2, how to make it efficient?





Voxel Grid Downsampling



How to “take one point” from a cell in the grid?

1. Centroid
 - a. For coordinates, compute the average in the cell
 - b. For other attributes, voting / average
 - c. More accurate but slower – Error up to $\text{cell_size} / 2$
2. Random select
 - a. Randomly select a point in the cell
 - b. Less accurate but faster – Error up to cell_size



Voxel Grid Downsampling - Exact

1. Compute the min or max of the point set $\{p_1, p_2, \dots, p_N\}$
 $x_{max} = \max(x_1, x_2, \dots, x_N), x_{min} = \min(x_1, x_2, \dots, x_N), y_{max} = \dots \dots$
2. Determine the voxel grid size r
3. Compute the dimension of the voxel grid
$$D_x = (x_{max} - x_{min})/r$$
$$D_y = (y_{max} - y_{min})/r$$
$$D_z = (z_{max} - z_{min})/r$$
4. Compute voxel index for each point
$$h_x = \lfloor (x - x_{min})/r \rfloor$$
$$h_y = \lfloor (y - y_{min})/r \rfloor$$
$$h_z = \lfloor (z - z_{min})/r \rfloor$$
$$h = h_x + h_y * D_x + h_z * D_x * D_y$$
5. Sort the points according to the index in Step 4
6. Iterate the sorted points, select points according to Centroid / Random method
0, 0, 0, 0, 3, 3, 3, 8, 8, 8, 8, 8, 8, 8, 8,



Voxel Grid Downsampling - Exact



Int32 overflow!

- Example, 3D lidar in autonomous driving. Detection range 200m, voxel grid resolution r=0.05m, assume we crop z to be [-10, 10]
- Dimension of the voxel grid: $(20/0.05) * (400/0.05) * (400/0.05) = 2.56 \times 10^{10}$
- $2^{32} = 4.3 \times 10^9 < 2.56 \times 10^{10}$



Voxel Grid Downsampling - Exact



Strict Weak Ordering!

- In cpp, the sort function in <algorithm> supports customized comparator
- However, the comparator should follow the strict weak ordering:

Expression	Return type	Requirements
<code>comp(a, b)</code>	implicitly convertible to <code>bool</code>	Establishes strict weak ordering relation with the following properties <ul style="list-style-type: none">• For all a, <code>comp(a,a)==false</code>• If <code>comp(a,b)==true</code> then <code>comp(b,a)==false</code>• if <code>comp(a,b)==true</code> and <code>comp(b,c)==true</code> then <code>comp(a,c)==true</code>

- In the voxel grid downsampling setting, the sorting comparator should be `a.index < b.index`, instead of `a.index <= b.index`
- Otherwise this is undefined behavior that may lead to segmentation fault



Voxel Grid Downsampling – Approximated

- ❖ Exact voxel grid downsampling requires sorting $O(N * \log(N))$
- ❖ However, in most cases, the voxel is **SPARSE**
- ❖ Image we have $N=10000$ points, we know after downsample the number $M < 100$. (E.g, 95)
- ❖ Can we have a magic function, that maps the 10000 points into the 100 containers?
- ❖ Finally we just extract one point from the 100 containers. Ideally there will be 95 non-empty containers, and 5 empty.

Hash Table!



Voxel Grid Downsampling – Approximated

1. Compute the min / max of each coordinate
2. Determine the voxel grid size r
3. Compute the dimension of the voxel grid
4. Compute voxel index for each point
5. Use a *hash* function to map each point to a container G_i in $\{G_1, G_2, \dots, G_M\}$
6. Iterate $\{G_1, G_2, \dots, G_M\}$ and get M point!
 - That *hash* function is

$$\text{hash}(h_x, h_y, h_z) : \mathbb{R}^3 \rightarrow \mathbb{R}$$



Voxel Grid Downsampling – Approximated



The **hash** function is not magic, not perfect!

- Different voxel will map into the same value

$$\text{hash}(h_x, h_y, h_z) = \text{hash}(h'_x, h'_y, h'_z), h_x \neq h'_x \text{ or } h_y \neq h'_y \text{ or } h_z \neq h'_z$$

- Consequence: The 10000 points should fill in 95 containers, but in fact fill only 80. You are missing 15 points!



This is called **conflict** in hash table



Voxel Grid Downsampling – Approximated

Hexagon icon: How to solve *conflict* in hash table?

Hexagon icon: Detect it!

$$\text{hash}(h_x, h_y, h_z) = \text{hash}(h'_x, h'_y, h'_z), h_x \neq h'_x \text{ or } h_y \neq h'_y \text{ or } h_z \neq h'_z$$

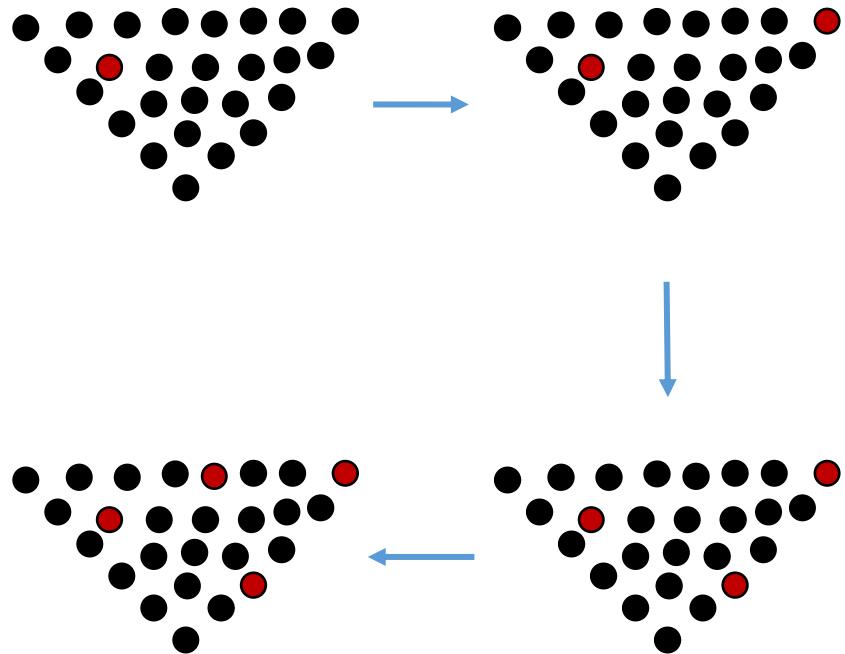
Hexagon icon: If you find a *conflict*, what do you do?

1. Select a point from the container
2. Empty that container



Farthest Point Sampling (FPS)

1. Randomly choose a point to be the first FPS point
2. Iterate until we get the desired number of points
 - a. For each point in the original point cloud, compute its distance to the nearest FPS point
 - b. Choose the point with the largest value, add to FPS set



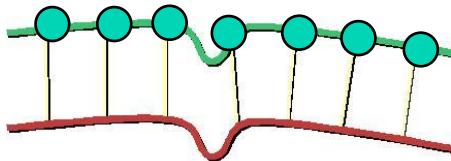


Normal Space Sampling (NSS)

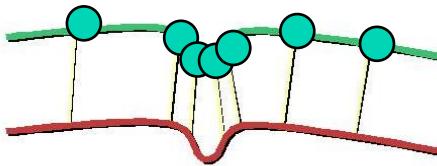


Used in Iterative Closest Point

1. Construct a set of buckets in the normal space
2. Put all points into bucket according to the surface normals
3. Uniformly pick points from all buckets until we have the desired number of points



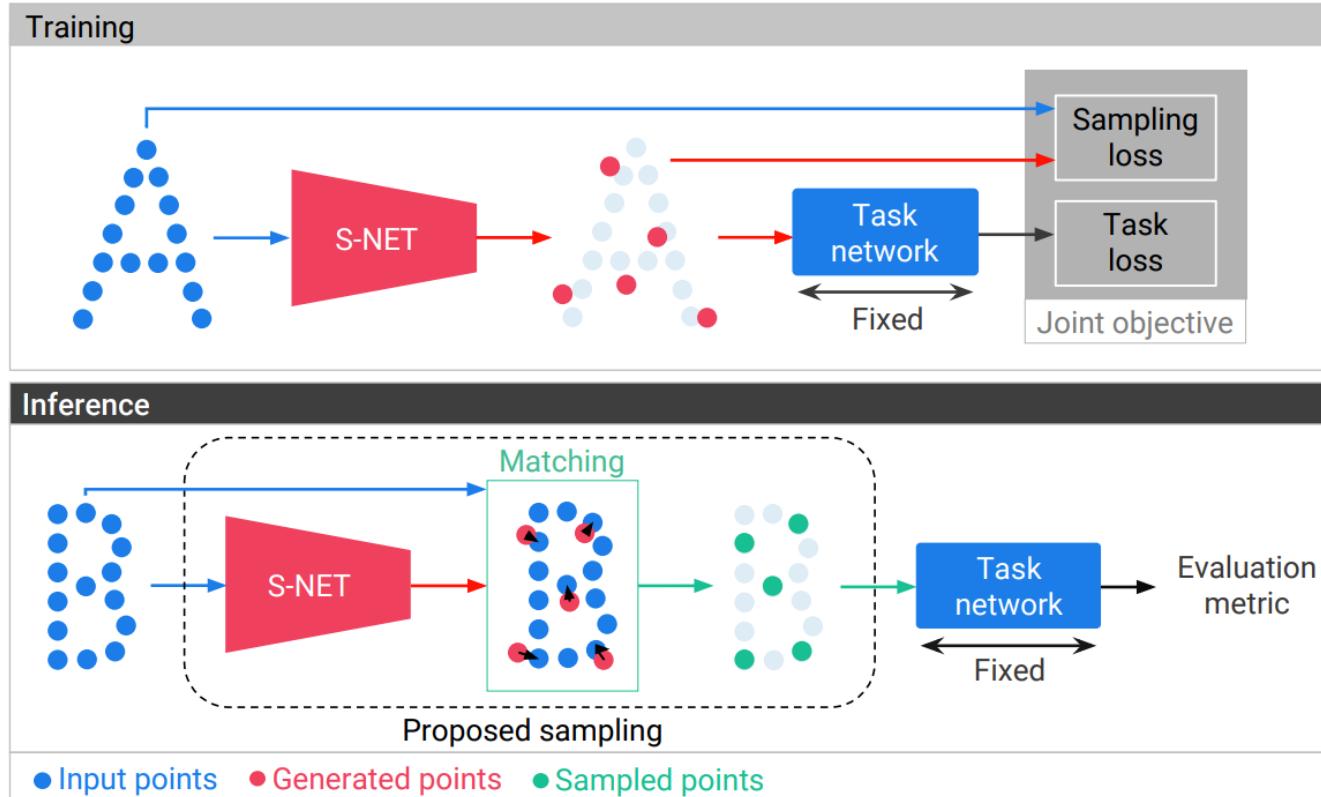
Uniform Sampling



Normal Space Sampling



Learning to Sample



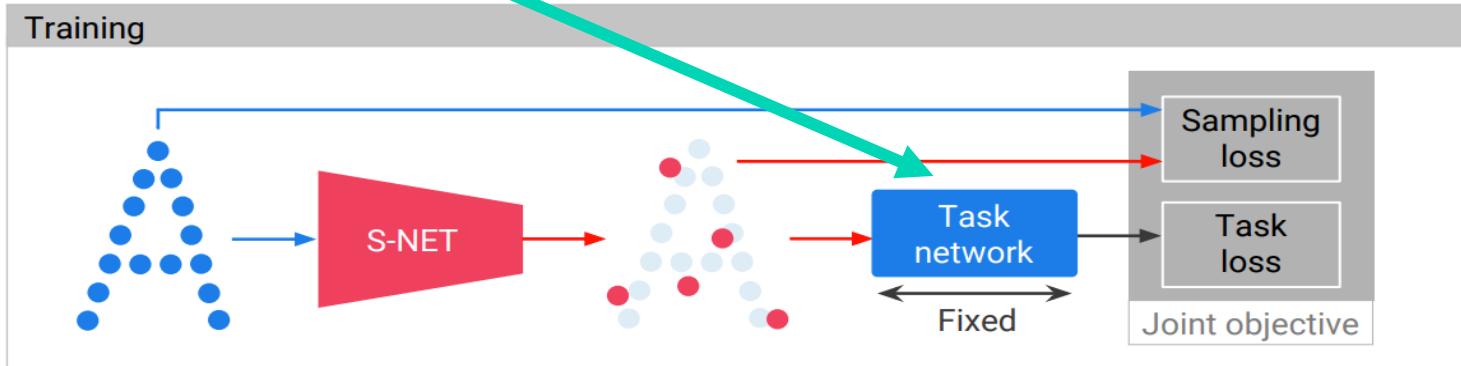


Learning to Sample

Problem statement Given a point set $P = \{p_i \in \mathbb{R}^3, i = 1, \dots, n\}$, a sample size $k \leq n$ and a task network T , find a subset S^* of k points that minimizes the task network's objective function f :

$$S^* = \operatorname{argmin}_S f(T(S)), \quad S \subset P, \quad |S| = k \leq n. \quad (1)$$

Sampling
based on
Semantics





Learning to Sample

Sampling with Geometric Constraints

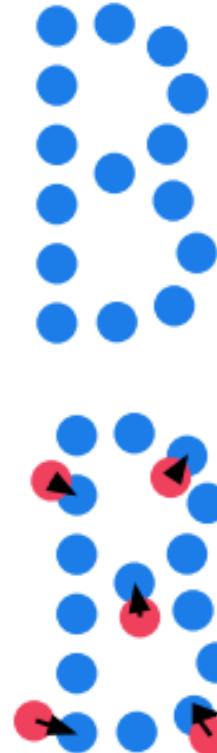
Let us denote the generated point set as G and the input point set as P . We construct a sampling regularization loss, composed out of three terms:

$$L_f(G, P) = \frac{1}{|G|} \sum_{g \in G} \min_{p \in P} \|g - p\|_2^2 \quad (2)$$

$$L_m(G, P) = \max_{g \in G} \min_{p \in P} \|g - p\|_2^2 \quad (3)$$

$$L_b(G, P) = \frac{1}{|P|} \sum_{p \in P} \min_{g \in G} \|p - g\|_2^2. \quad (4)$$

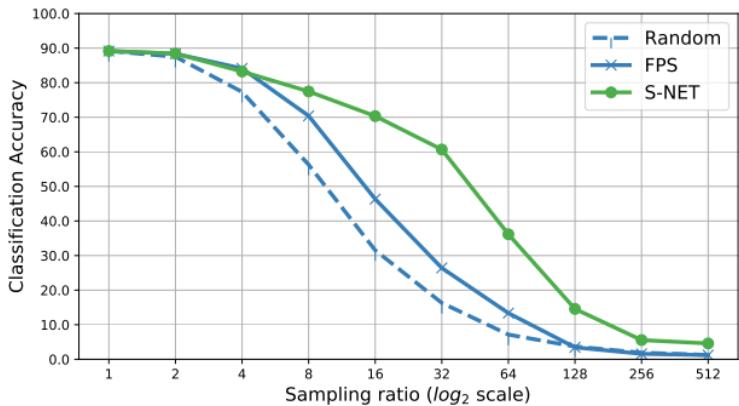
(2) + (4) = Chamfer loss





Learning to Sample – for classification

- The Learning to Sample is targeted to some specific task, e.g., Classification
- Semantics based downsampling instead of pure geometric based.



#Sampled points	Random	FPS	S-NET
1024	89.2	89.2	89.2
512	88.2	88.3	87.8
256	86.6	88.1	88.3
128	86.2	87.9	88.6
64	81.5	86.1	87.7
32	77.0	82.2	87.3
16	65.8	76.7	85.6
8	45.8	61.6	83.6
4	26.9	35.2	73.4
2	16.6	18.3	53.0



Learning to Sample – for reconstruction



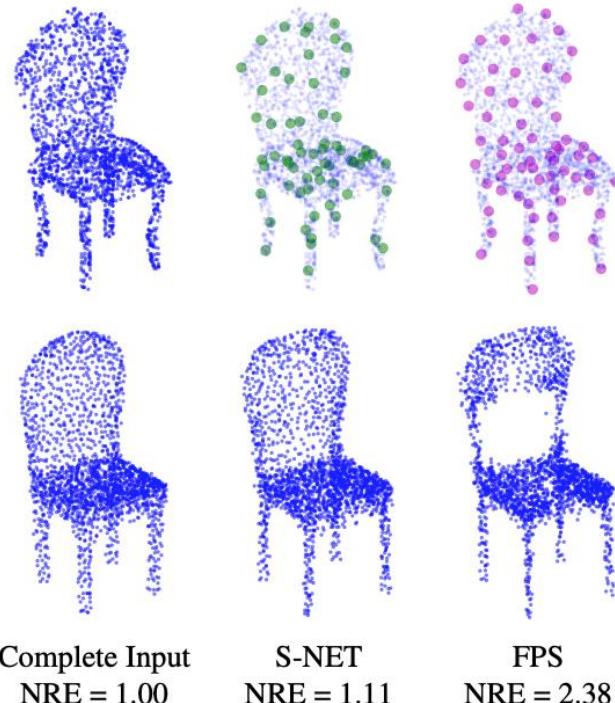
NRE:

Normalized Reconstructed Error



The output of S-Net is visually similar to that of FPS

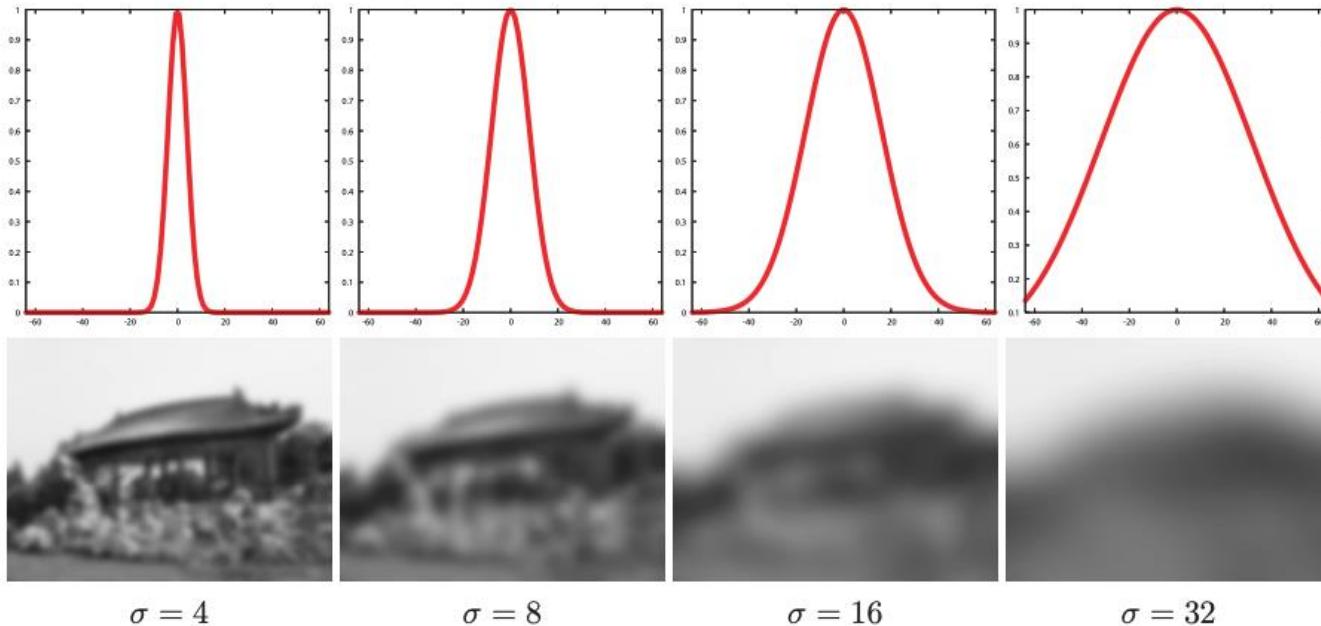
- This is expected because of Chamfer loss





Bilateral Filter – Gaussian Filter

$$GB[I]_{\mathbf{p}} = \sum_{\mathbf{q} \in S} G_\sigma(\|\mathbf{p} - \mathbf{q}\|) I_{\mathbf{q}}, \quad G_\sigma(x) = \frac{1}{2\pi \sigma^2} \exp\left(-\frac{x^2}{2\sigma^2}\right)$$





Edge Preserving Blurring



1 iteration



2 iteration



4 iteration



Bilateral Filter

- Given image I , for each pixel p , find its neighbor S .
- each pair (p, q)

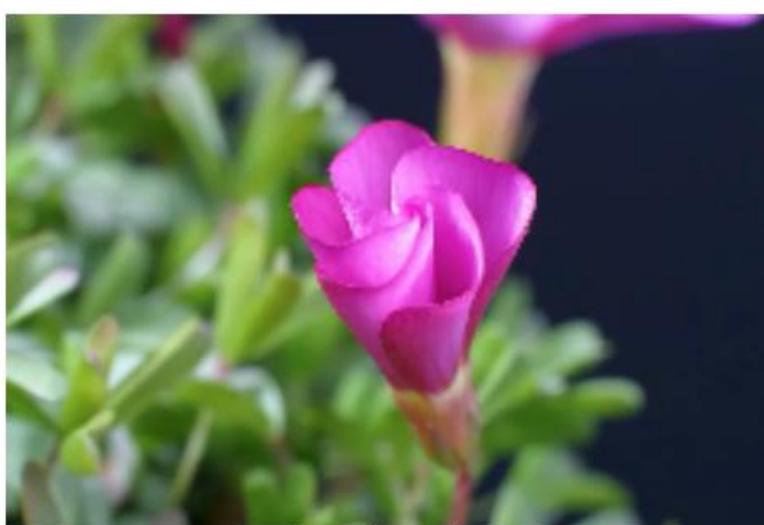
- Compute distance weight G_{σ_s} intensity weight G_{σ_r}

$$G_\sigma(x) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2}{2\sigma^2}\right).$$

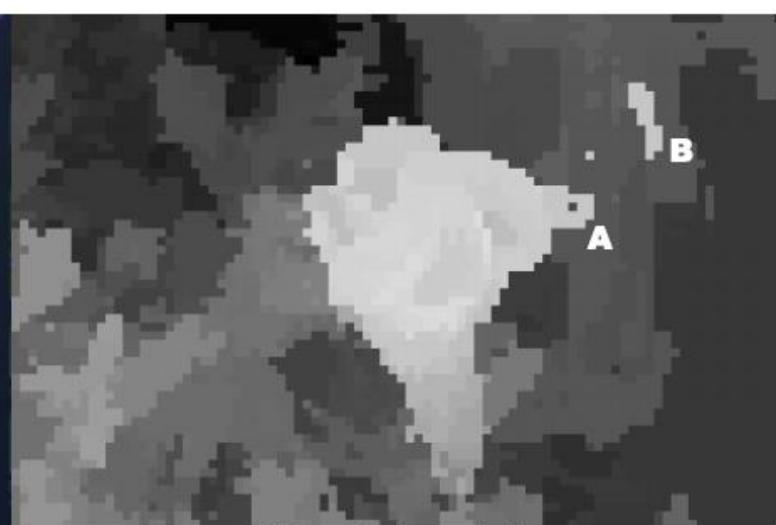
- Apply Bilateral Filter to get intensity of pixel p

$$BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(I_p - I_q) I_q$$

$$W_p = \sum_{q \in S} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(I_p - I_q)$$



a) Input image Flower



b) Input depth map (8×8)





Homework - Compulsory

1. Build dataset for Lecture 1
 - a. Download ModelNet40 dataset
 - b. Select one point cloud from each category
2. Perform PCA for the 40 objects, visualize it.
3. Perform surface normal estimation for each point of each object, visualize it.
4. Downsample each object using voxel grid downsampling (exact, both centroid & random). Visualize the results.
5. Write your own code, DO NOT call apis (PCL, open3d, etc.) except for visualization.



Homework – Optional



KITTI depth dataset

- http://www.cvlibs.net/datasets/kitti/eval_depth_all.php
- Download and get familiar KITTI dataset



Perform depth upsampling / completion for the validation dataset

- Use whatever method you want, **except** Deep Learning method.



Evaluate the result using the evaluation code provided in the kitti-depth development kit.

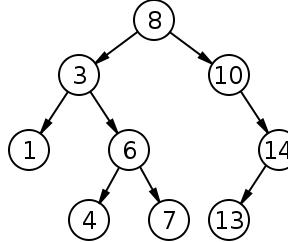


Lecture Outline



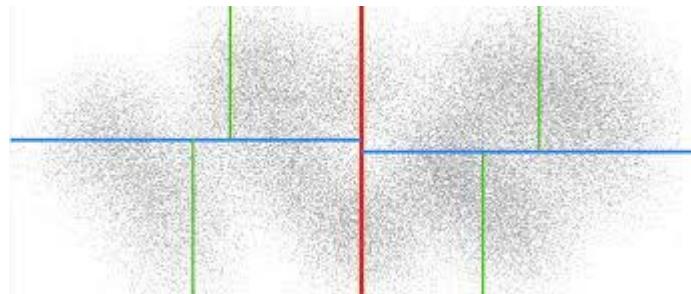
Binary Search Tree

- Basic knowledge about trees
- 1D NN problem
- With Python codes



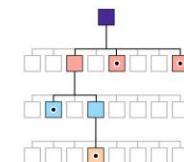
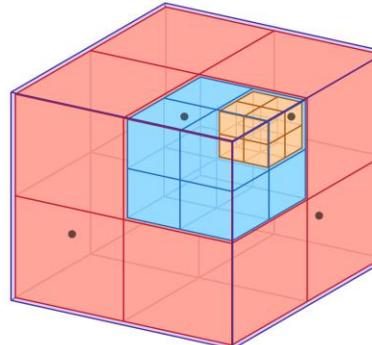
Kd-tree

- Works for data of any dimension
- Illustrated in 2D
- With Python codes



Octree

- Specifically designed for 3D data
- Illustrated in 2D/3D
- With Python codes



Thanks for Listening!

