



# 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 Analysis



3、Downsampling



4、Filtering



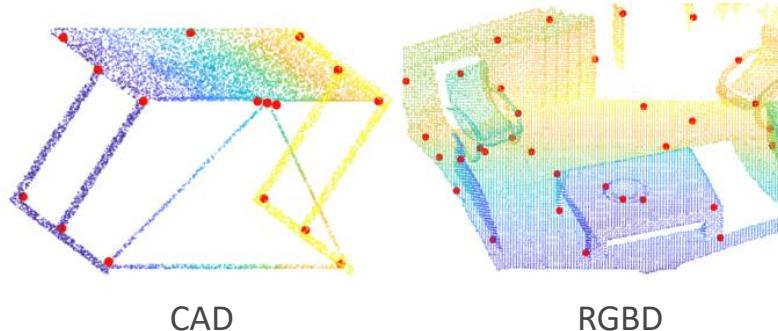
# 3D Point Cloud



## Representation

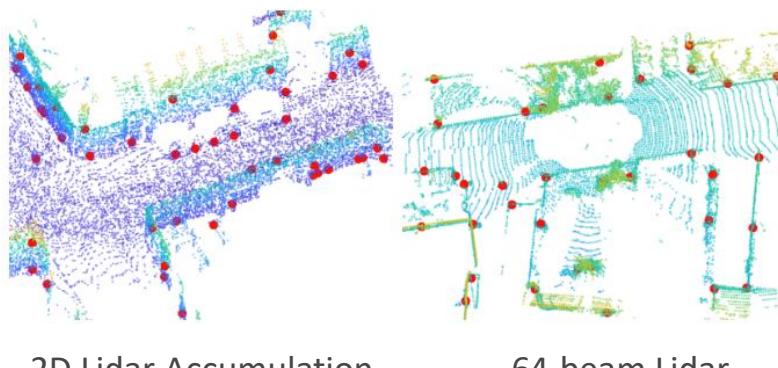
- $N \times 3$  matrix or  $N \times (3 + D)$  matrix

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



## Data sources:

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





# Applications of Point Clouds



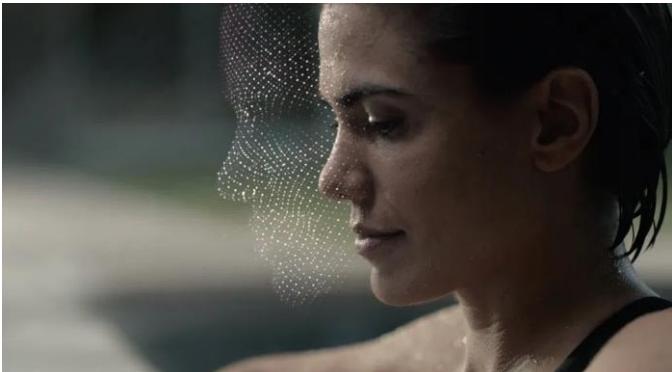
## Robotics, Autonomous driving

- 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=27OuOCeZmwI>

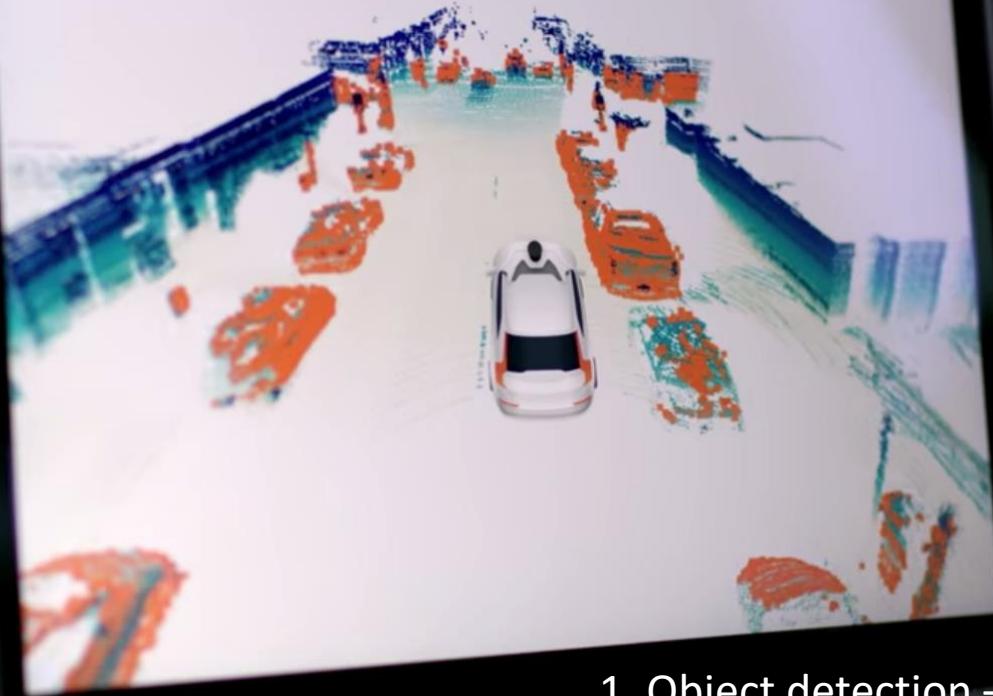
Our Road to Self-Driving Vehicles | Uber ATG



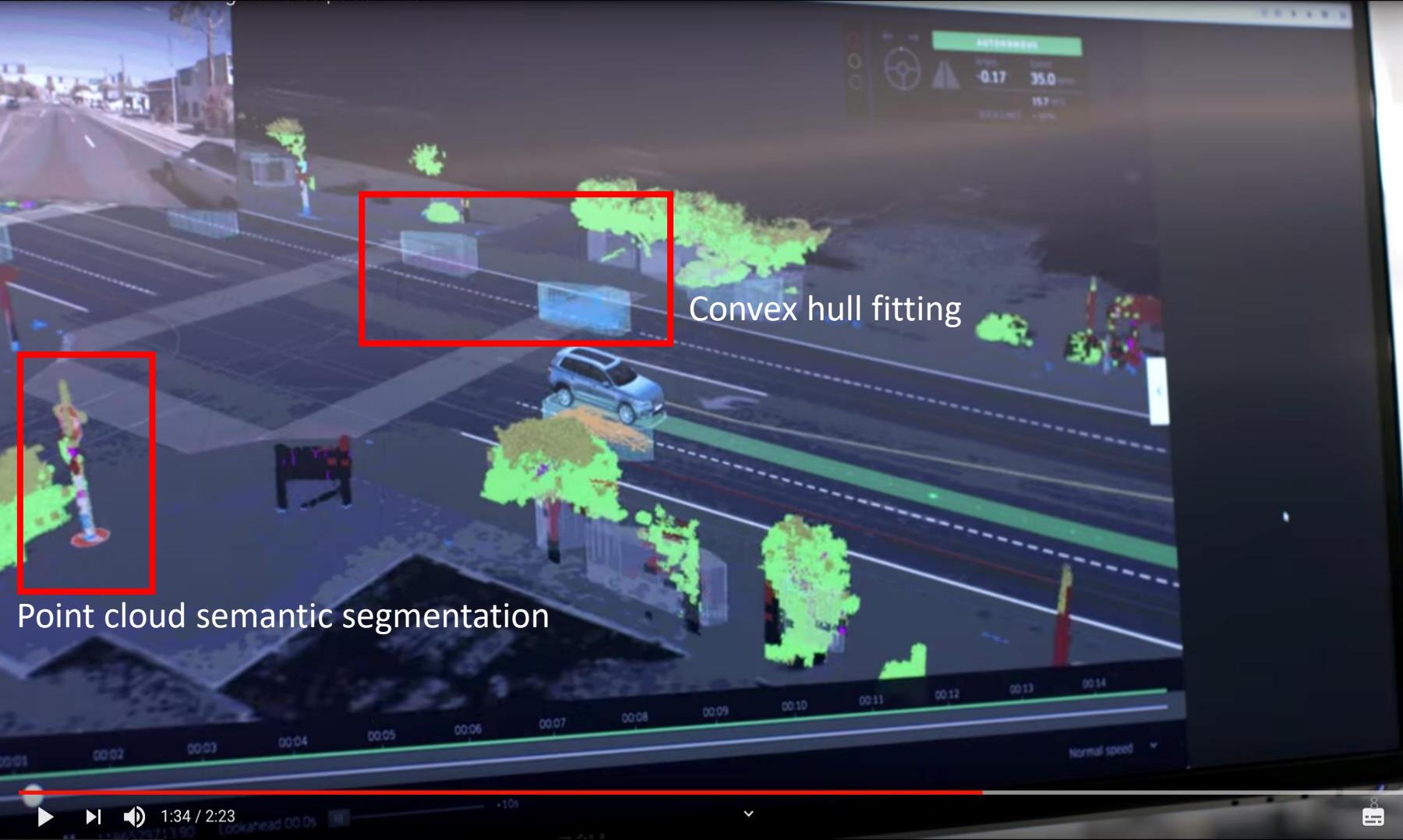


Velodyne 64





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



```
heistlab
heistlab run script
Starts up the stub server at http://localhost:1699

Installation:
1. 'cd heistlab && virtualenv venv'
2. 'source venv/bin/activate'
3. 'pip install -r requirements.txt'

Usage:
1. 'python heistlab.py'

No virtualenv? 'sudo pip install virtualenv' or 'sudo easy_install virtualenv'

Import argparse
import sys
import multiprocessing
import os
import sys
sys.dont_write_bytecode = True
from stub import stub
from stub import port_from_request
from stub import port_with_it_would_be
from stub import port_with_it_could_be
from stub import port
from stub import request
from flask import request, abort

__author__ = "devitas"
app = apt.create_app()

app.after_request
def validate_and_restrict(response):
    port_from_request.headers.get('Heistlab-Restrict-Port', 16999)
    restricted_port = int(response.headers.get('Heistlab-Restrict-Port', 16999))
    if port != restricted_port:
        return app.response_class("Invalid Port", 404)
    else:
        return response

app.after_request
def validate_and_restrict(response):
    response.headers.add("Access-Control-Allow-Origin", "*")
    response.headers.add("Access-Control-Allow-Methods", "GET, POST, PUT, DELETE")
    response.headers.add("Access-Control-Allow-Headers", "Content-Type, Authorization")

    return response

def startServer(port1):
    app.run("0.0.0.0",
           port=port1,
           threaded=True,
           debug=True)

-- INSERT --
```

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



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

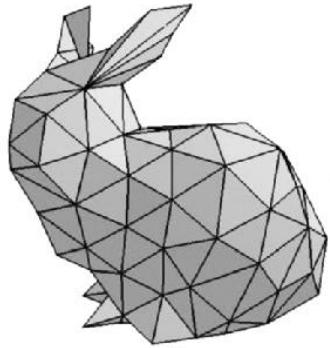


## Advantages & Difficulties

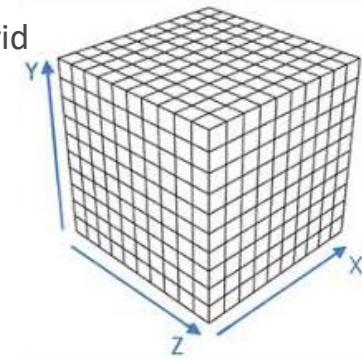


### What are the options for 3D?

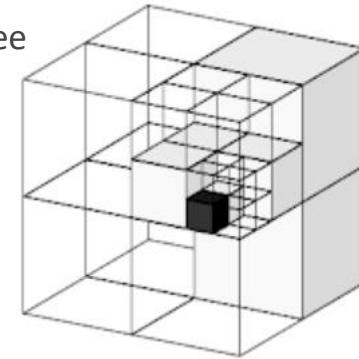
Mesh



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
  - 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/kPCA, downsampling and filtering
2	Nearest neighbor algorithms
3	Clustering algorithms
4	Model fitting
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 50% theory and 50% practice
  - Theory – proof, derivation
  - Practice
    - Algorithms
    - Engineering practices
    - Assignments are mostly coding



# Prerequisite



## Basic Linear Algebra

- Eigen decomposition, SVD, PCA



## Basic Probability Theory

- Bayes rule
- Marginalization
- Graph theory



## Computer Vision

- Camera models



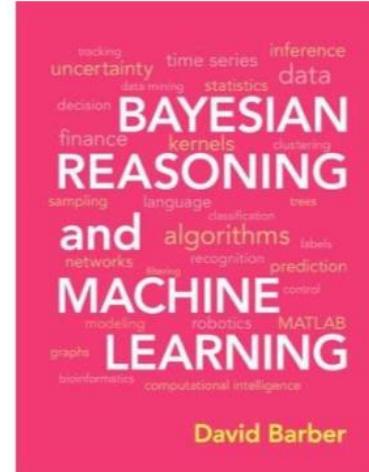
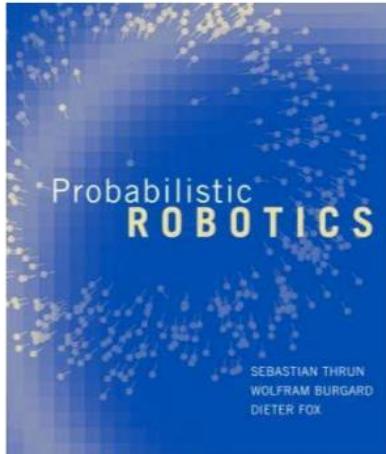
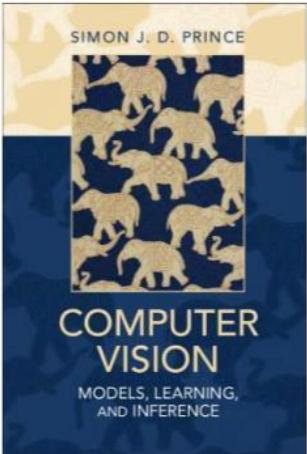
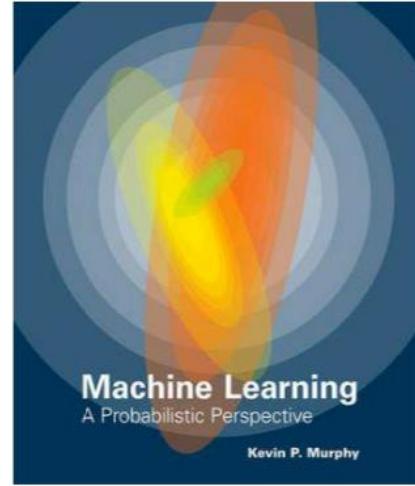
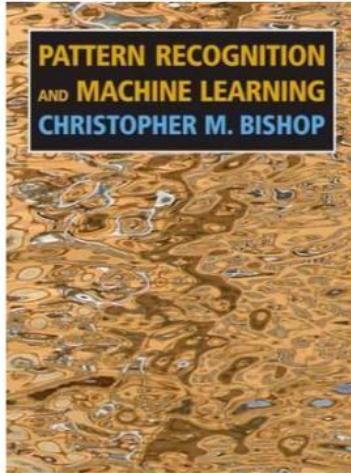
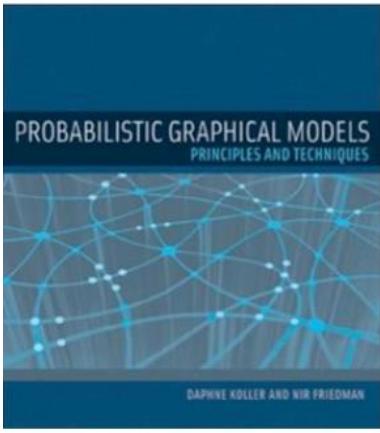
## Simple Optimization

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



## 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



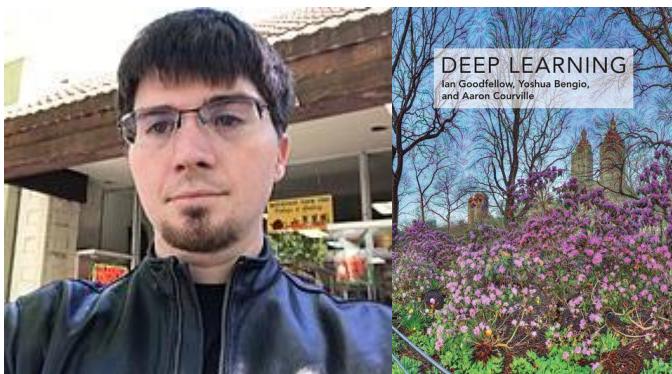


# Course Outline - Theory



## 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

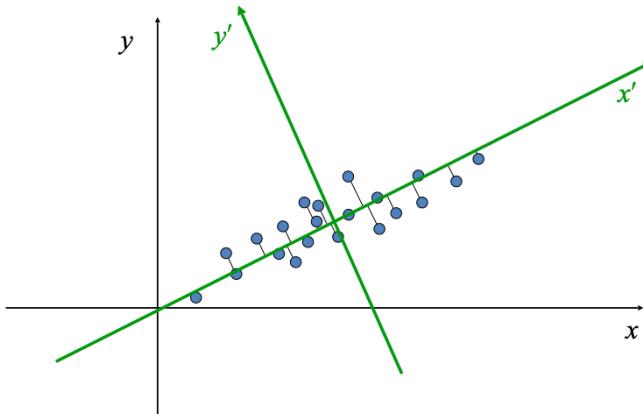
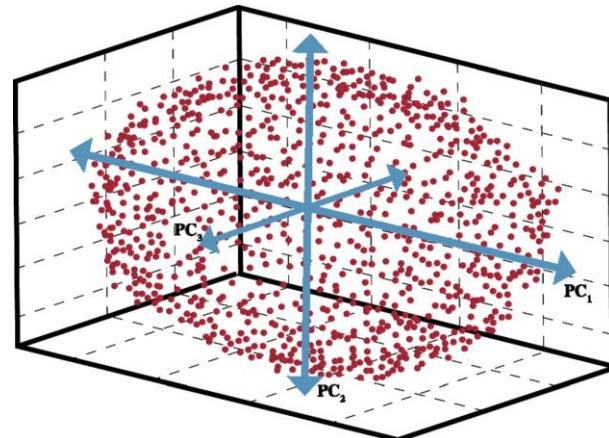


# Principle Component Analysis

PCA is to find the dominant directions of the point cloud

## Applications:

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



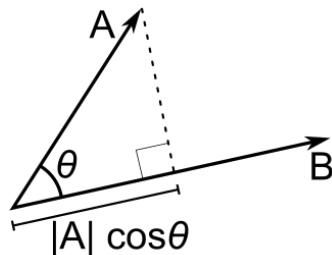


# Principle Component Analysis



Let's start with some physical intuitions.

Vector Dot Product

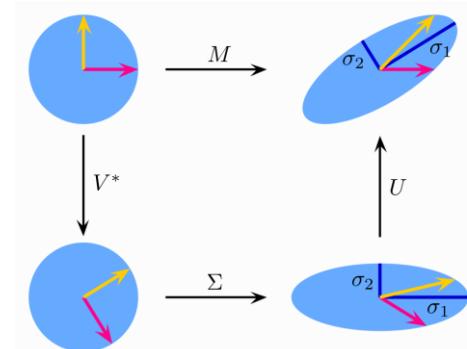


Matrix-Vector Multiplication

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

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,

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

是拉伸矩阵，手算易验，拉伸矩阵的特征值就是各个拉伸元素



## 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:

- Apply the **spectral theorem**,  $U$  is orthogonal,  $\Lambda$  is diagonal

$$x^T Ax = x^T U \Lambda U^T x = \bar{x}^T \Lambda \bar{x} = \sum_{i=1}^n \lambda_i \bar{x}_i^2$$

- Obviously,

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

- Also, orthogonal matrix  $U$  doesn't change the norm of any vector

$$\sum_{i=1}^n x_i^2 = x^T x = x^T U U^T x = (U^T x)^T (U^T x) = \bar{x}^T \bar{x} = \sum_{i=1}^n \bar{x}_i^2$$

- Combining the above 3 equations,

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



## Principle Component Analysis

- **Input:**  $x_i \in \mathbb{R}^n, i = 1, 2, \dots, m$
- **Output:** principle vectors  $z_1, z_2, \dots, z_k \in \mathbb{R}^n, k \leq n$



**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 3<sup>rd</sup> one?**

A: Repeat the above steps.



## Principle Component Analysis - Proof

tilde X是去掉平均值后的点的集合

- Normalize the data to be zero mean

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

- PCA is to get largest variance when projected to a direction  $z \in \mathbb{R}^n, \|z\|_2 = 1$

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

$z$ 是所选成分  
是投影到主成分后保留的信息

- The mean variance of the projections is

因为 是平均值投影后的结果， 所以平方后就是方差

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

- So, maximize it,

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

使方差最大的 $z$ 就是主成分  
使方差最小的 $z$ 就是法向量



## Principle Component Analysis - Proof

- Now, maximize this

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

- Recall the Rayleigh Quotients  $\lambda_{\min}(A) \leq \frac{x^T A x}{x^T x} \leq \lambda_{\max}(A), \forall x \neq 0$
- Recall our Spectral Theorem  $A = U \Lambda U^T = \sum_{i=1}^n \lambda_i u_i u_i^T, \Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$
- Apply to PCA

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

- First principle vector  $\textcolor{red}{z_1} = \textcolor{red}{u_1}$ ,  $u_1$  is the first column of  $U_r$



## Principle Component Analysis - Proof

- Let's take a look at  $H = \tilde{X}\tilde{X}^T = U_r\Sigma^2U_r^T$
- Perform SVD on  $\tilde{X}$ :  $\tilde{X} = U_r\Sigma V_r^T = \sum_{i=1}^r \sigma_i u_i v_i^T$

Spectral Theorem and  
SVD are closely related

- Find  $z_2$  by deflation

$$\tilde{x}_i^{(1)} = \tilde{x}_i - u_1(u_1^T \tilde{x}_i), i = 1, \dots, m$$

$$\tilde{X}^{(1)} = [\tilde{x}_1^{(1)}, \dots, \tilde{x}_m^{(1)}] = (I_n - u_1 u_1^T) \tilde{X}$$

- Combine the above equations:

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



## Principle Component Analysis - Proof

- We have removed the first components, finding  $z_2$  is by

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

$$\tilde{X}^{(1)} = \sum_{i=2}^r \sigma_i u_i v_i^T$$

- The result is simply  $z_2 = u_2$ ,  $u_2$  is the 2<sup>nd</sup> column of  $U_r$
- $z_3, \dots, z_m$  can be found by similar deflation.



## PCA - Summary

Given  $x_i \in \mathbb{R}^n, i = 1, 2, \dots, m$ , 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 vectors are the columns of  $U_r$   
(Eigenvector of  $X$  = Eigenvector of  $H$ )



## PCA – Dimensionality Reduction

Given  $x_i \in \mathbb{R}^n, i = 1, 2, \dots, m$ , perform PCA to get  $l$  principle components  $\{z_1, z_2, \dots, z_l\}, z_j \in \mathbb{R}^n$

- Compress  $x_i$  from  $n$  dimension to  $l$  dimension, with  $l \ll n$

Encoder

$$\begin{bmatrix} a_{i1} \\ \vdots \\ a_{il} \end{bmatrix} = \begin{bmatrix} z_1^T \\ \vdots \\ z_l^T \end{bmatrix} x_i$$

- Reconstruct  $x_i$  from the principle components

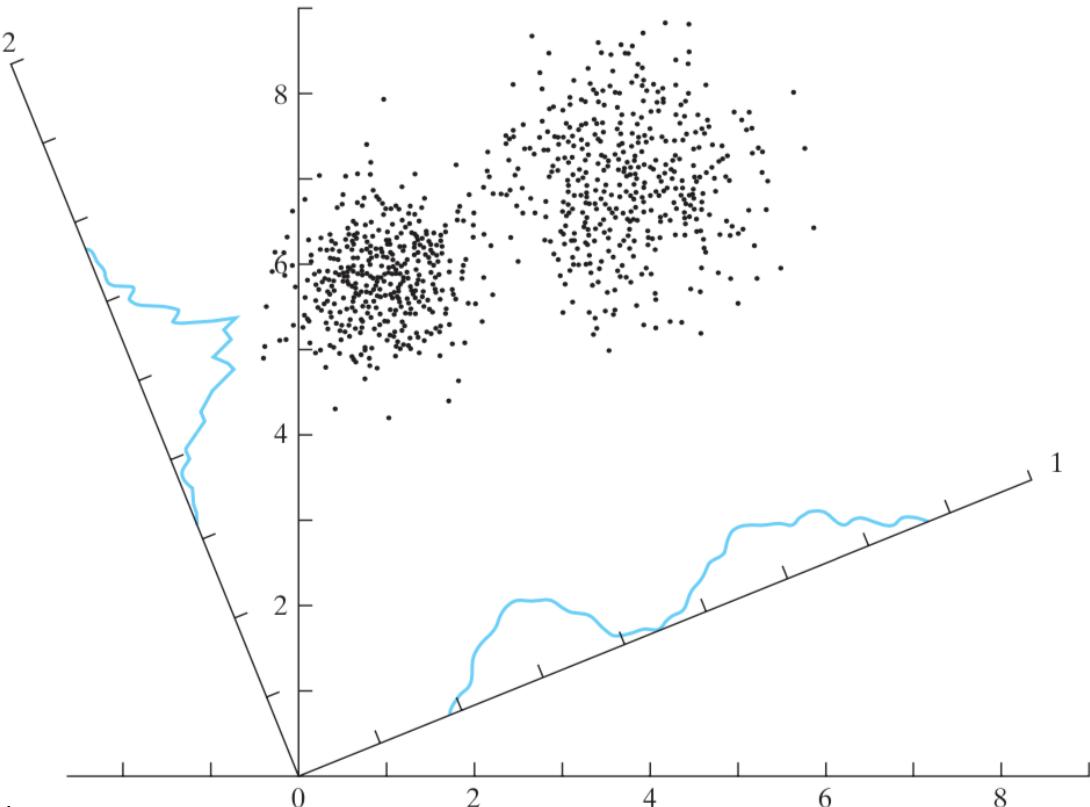
Decoder

$$\hat{x}_i = \sum_{j=1}^l a_j z_j = [z_1, \dots, z_l] \begin{bmatrix} a_{i1} \\ \vdots \\ a_{il} \end{bmatrix}$$



## PCA – Dimensionality Reduction

Point cloud is projected  
into two principle axis {1, 2}





## PCA – Dimensionality Reduction

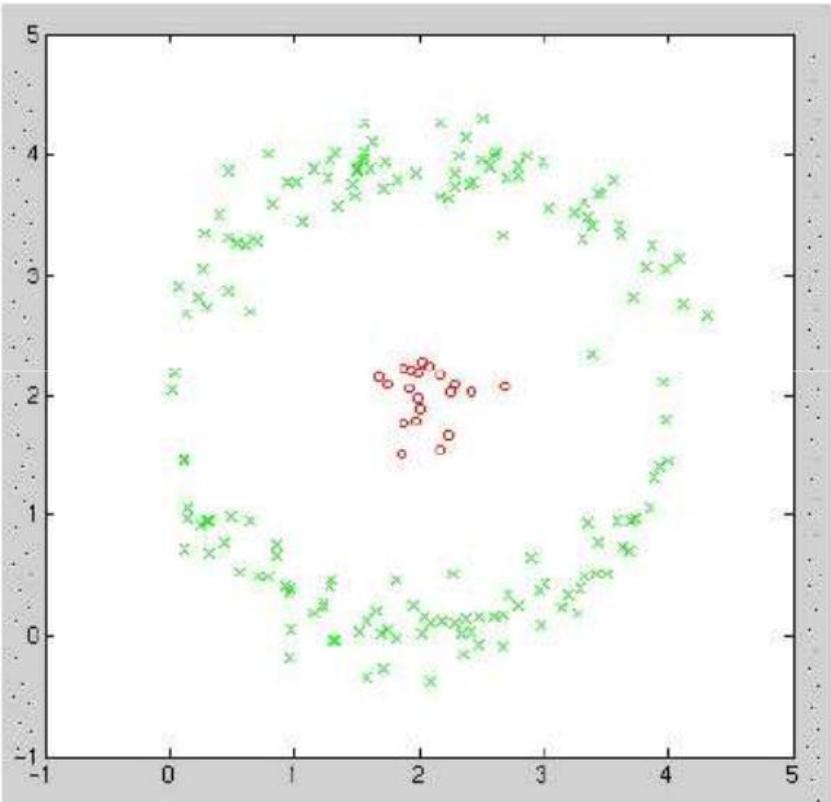
- Represent a  $H \times W$  binary/gray-scale image by a vector  $x_i \in \mathbb{R}^n, n = HW$
- Get the principle vectors  $\{z_1, \dots, z_l\}, z_j \in \mathbb{R}^n$
- Digit recognition by clustering over the principle components  $a_i = [a_1, \dots, a_l]^T \in \mathbb{R}^l$
- Similarly, face recognition by **Eigenfaces**





## Kernel PCA

- PCA is linear
- How to handle data not linearly separable?
- Lift it to **high dimension!**





## Kernel PCA

- Original data

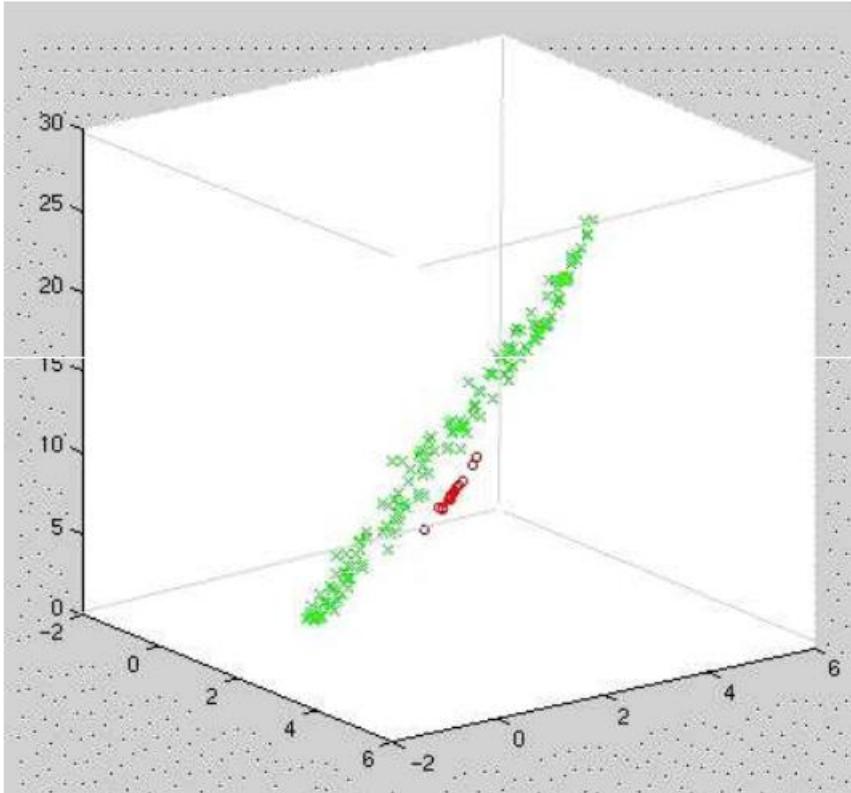
$$x_i = [x_{i1}, x_{i2}] \in \mathbb{R}^2$$

- Lifted data

$$\phi(x_i) = [x_{i1}, x_{i2}, x_{i1}^2 + x_{i2}^2] \in \mathbb{R}^3$$

- They are separable now.

- E.g., some principle component of  $\phi(x_i)$  is able to tell the difference between the red and green





## Kernel PCA

- Input data  $x_i \in \mathbb{R}^{n_0}$ , non-linear mapping  $\phi: \mathbb{R}^{n_0} \rightarrow \mathbb{R}^{n_1}$
- Follow the standard Linear PCA on the lifted space  $\mathbb{R}^{n_1}$

1. Assume  $\phi(x_i)$  is already zero-center

$$\frac{1}{N} \sum_{i=1}^N \phi(x_i) = 0$$

2. Compute correlation matrix

$$\tilde{H} = \frac{1}{N} \sum_{i=1}^N \phi(x_i) \phi^T(x_i)$$

3. Solve the eigenvectors/eigenvalues by  $\tilde{H}\tilde{z} = \tilde{\lambda}\tilde{z}$

- Problem solved? No fully.
  - How to define  $\phi$ ?
  - Can we avoid working with the high dimension data?



## Kernel PCA

- Note that eigenvectors can be expressed as linear combination of features

$$\tilde{z} = \sum_{j=1}^N \alpha_j \phi(x_j)$$

- Proof:

$$\tilde{H}\tilde{z} = \tilde{\lambda}\tilde{z}$$

$$\frac{1}{N} \sum_{i=1}^N \phi(x_i) \phi^T(x_i) \tilde{z} = \tilde{\lambda}\tilde{z}$$

  
scalar

- Find the eigenvector  $\tilde{z}$  = find the coefficient  $\alpha_j$



## Kernel PCA

- Put that linear combination into  $\tilde{H}\tilde{z} = \tilde{\lambda}\tilde{z}$

$$\frac{1}{N} \sum_{i=1}^N \phi(x_i) \phi^T(x_i) \left( \sum_{j=1}^N \alpha_j \phi(x_j) \right) = \tilde{\lambda} \sum_{j=1}^N \alpha_j \phi(x_j)$$

$$\frac{1}{N} \sum_{i=1}^N \phi(x_i) \left( \sum_{j=1}^N \alpha_j \phi^T(x_i) \phi(x_j) \right) = \tilde{\lambda} \sum_{j=1}^N \alpha_j \phi(x_j)$$

- Let's define kernel function  $k(x_i, x_j) = \phi^T(x_i) \phi(x_j)$

$$\frac{1}{N} \sum_{i=1}^N \phi(x_i) \left( \sum_{j=1}^N \alpha_j k(x_i, x_j) \right) = \tilde{\lambda} \sum_{j=1}^N \alpha_j \phi(x_j)$$

- Multiply both sides by  $\phi(x_k), k = 1, \dots, N,$

$$\sum_{i=1}^N \sum_{j=1}^N \alpha_j k(x_k, x_i) k(x_i, x_j) = N \tilde{\lambda} \sum_{j=1}^N \alpha_j k(x_k, x_j), k = 1, \dots, N$$



## Kernel PCA

- Now define the **Gram matrix**  $K \in \mathbb{R}^{N \times N}, K(i, j) = k(x_i, x_j)$ 
  - $K$  is symmetric because  $k(x_i, x_j) = k(x_j, x_i)$
- The above equation can be written as

$$K^2\alpha = N\tilde{\lambda}K\alpha$$

- Remove  $K$  on both sides

$$\begin{aligned} K\alpha &= N\tilde{\lambda}\alpha \\ K\alpha &= \lambda\alpha \end{aligned}$$

- Again, get the eigenvectors  $\alpha_r$  and eigenvalues  $\lambda_r, r = 1, \dots, l$
- However, we have to ensure that  $\tilde{z}$  is unit vector

Note that we are solving the **linear PCA in the feature space**

$$\tilde{H}\tilde{z} = \tilde{\lambda}\tilde{z} \quad \tilde{z} = \sum_{j=1}^N \alpha_j \phi(x_j)$$



## Kernel PCA

- The normalization of  $\tilde{z}$  leads to

$$1 = \tilde{z}_r^T \tilde{z}_r$$

$$1 = \sum_{i=1}^N \sum_{j=1}^N \alpha_{ri} \alpha_{rj} \phi^T(x_i) \phi(x_j)$$

$$1 = \alpha_r^T K \alpha_r$$

- Note that  $K\alpha = \lambda\alpha$ , we have  $\alpha_r^T \lambda_r \alpha_r = 1, \forall r$
- That is, normalize  $\alpha_r$  to be norm  $1/\lambda_r$
- Now, the  $r^{th}$  principle vector in the lifted space is given below, which is **unknown**

$$\tilde{z}_r = \sum_{j=1}^N \alpha_{rj} \phi(x_j)$$



## Kernel PCA

- Now, the  $r^{th}$  principle vector in the lifted space is given below

$$\tilde{z}_r = \sum_{j=1}^N \alpha_{rj} \phi(x_j)$$

- But we know the projection of data point  $x$  projected into principle component  $z_r$

$$\phi^T(x) \tilde{z}_r = \sum_{j=1}^N \alpha_{rj} \phi^T(x) \phi(x_j) = \sum_{j=1}^N \alpha_{rj} k(x, x_j)$$

- One more thing, we **assume**  $\phi(x_i)$  is of zero mean.



## Kernel PCA

- Normalize  $\phi(x_i)$  to be zero mean

$$\tilde{\phi}(x_i) = \phi(x_i) - \frac{1}{N} \sum_{j=1}^N \phi(x_j)$$

- The normalized kernel  $\tilde{k}(x_i, x_j)$  is given by

$$\begin{aligned}\tilde{k}(x_i, x_j) &= \tilde{\phi}^T(x_i) \tilde{\phi}(x_j) \\ &= \left( \phi(x_i) - \frac{1}{N} \sum_{k=1}^N \phi(x_k) \right)^T \left( \phi(x_j) - \frac{1}{N} \sum_{l=1}^N \phi(x_l) \right) \\ &= k(x_i, x_j) - \frac{1}{N} \sum_{k=1}^N k(x_i, x_k) - \frac{1}{N} \sum_{k=1}^N k(x_j, x_k) + \frac{1}{N^2} \sum_{k=1}^N \sum_{l=1}^N k(x_k, x_l)\end{aligned}$$

- In the matrix form  $\widetilde{K} = K - 2\frac{1}{N}K + \frac{1}{N}K\frac{1}{N}$ , where  $\frac{1}{N}(i, j) = \frac{1}{N}, \forall i, j$



- Kernel choices
  - Linear  $k(x_i, x_j) = x_i^T x_j$
  - Polynomial  $k(x_i, x_j) = (1 + x_i^T x_j)^p$
  - Gaussian  $k(x_i, x_j) = e^{-\beta \|x_i - x_j\|_2}$
  - Laplacian  $k(x_i, x_j) = e^{-\beta \|x_i - x_j\|_1}$
- Usually choose by experiments if there is no explicit knowledge what kernels best separate the data points.



## Kernel PCA - Summary

- Select a kernel  $k(x_i, x_j)$ , compute the Gram matrix  $K(i, j) = k(x_i, x_j)$
- Normalize  $K$

$$\tilde{K} = K - \frac{2\mathbb{I}_1}{N}K + \frac{\mathbb{I}_1}{N}K\frac{\mathbb{I}_1}{N}$$

- Solve the eigenvector/eigenvalues of  $\tilde{K}$

$$\tilde{K}\alpha_r = \lambda_r\alpha_r$$

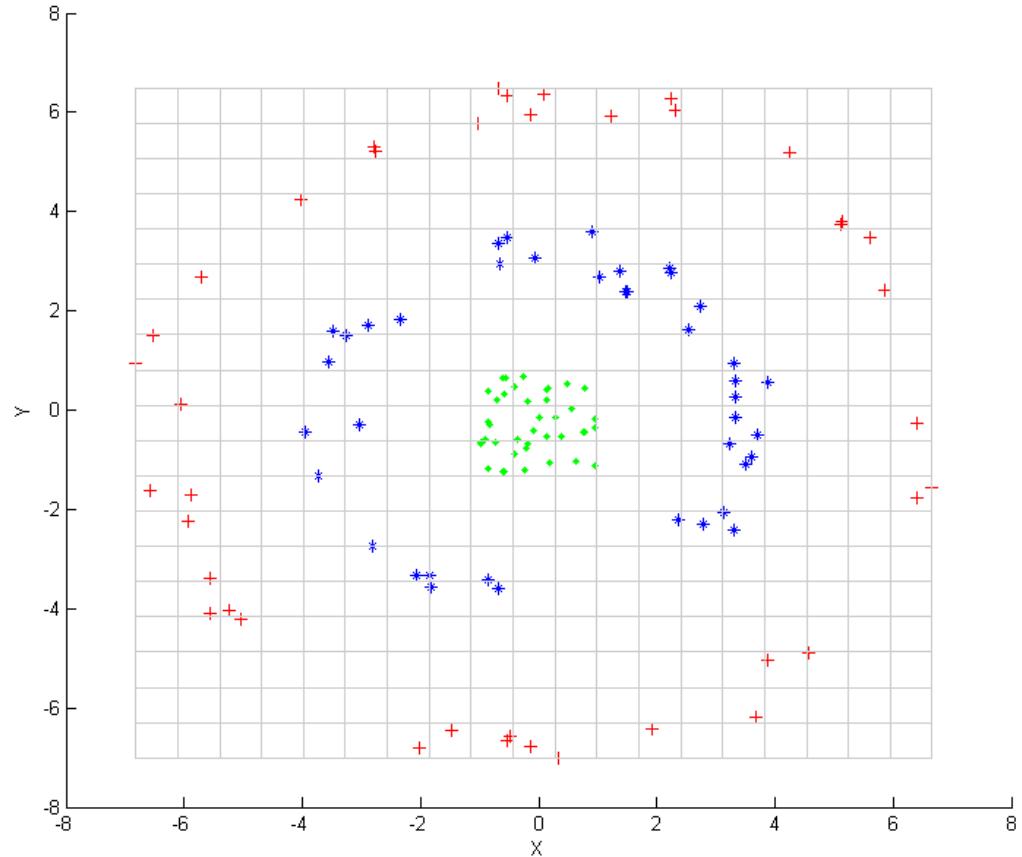
- Normalize  $\alpha_r$  to be  $\alpha_r^T \alpha_r = \frac{1}{\lambda_r}$
- For any data point  $x \in \mathbb{R}^n$ , compute its projection onto  $r^{th}$  principle component  $y_r \in \mathbb{R}$

$$y_r = \phi^T(x)\tilde{z}_r = \sum_{j=1}^N \alpha_{rj} k(x, x_j)$$



## Kernel PCA - Example

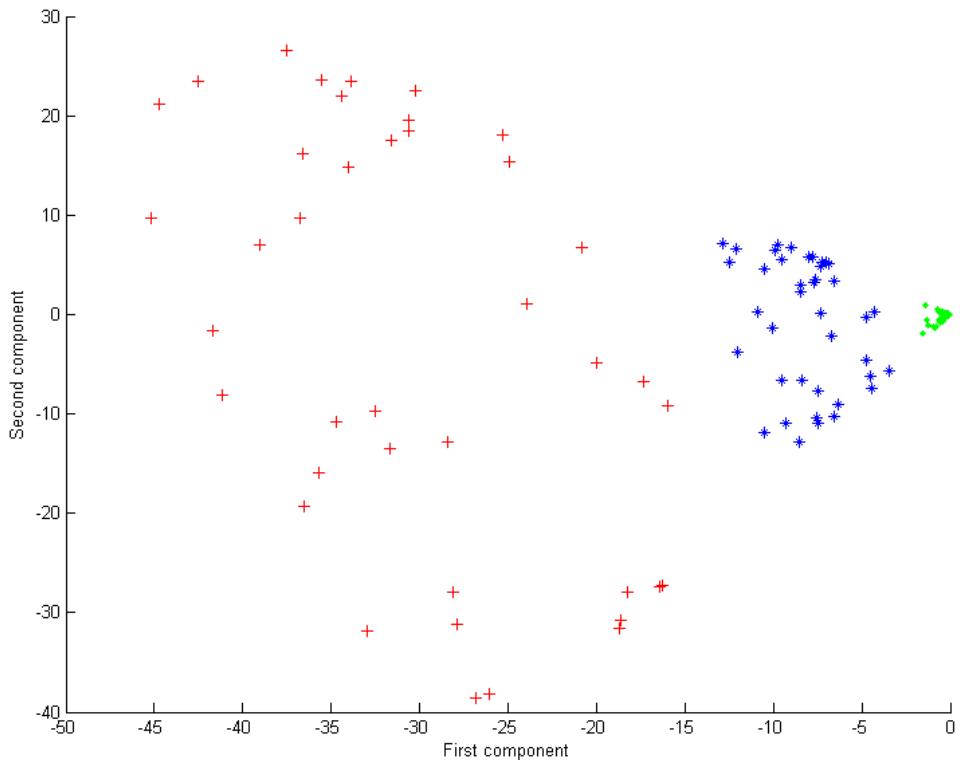
Input data is not separable by linear PCA





## Kernel PCA - Example

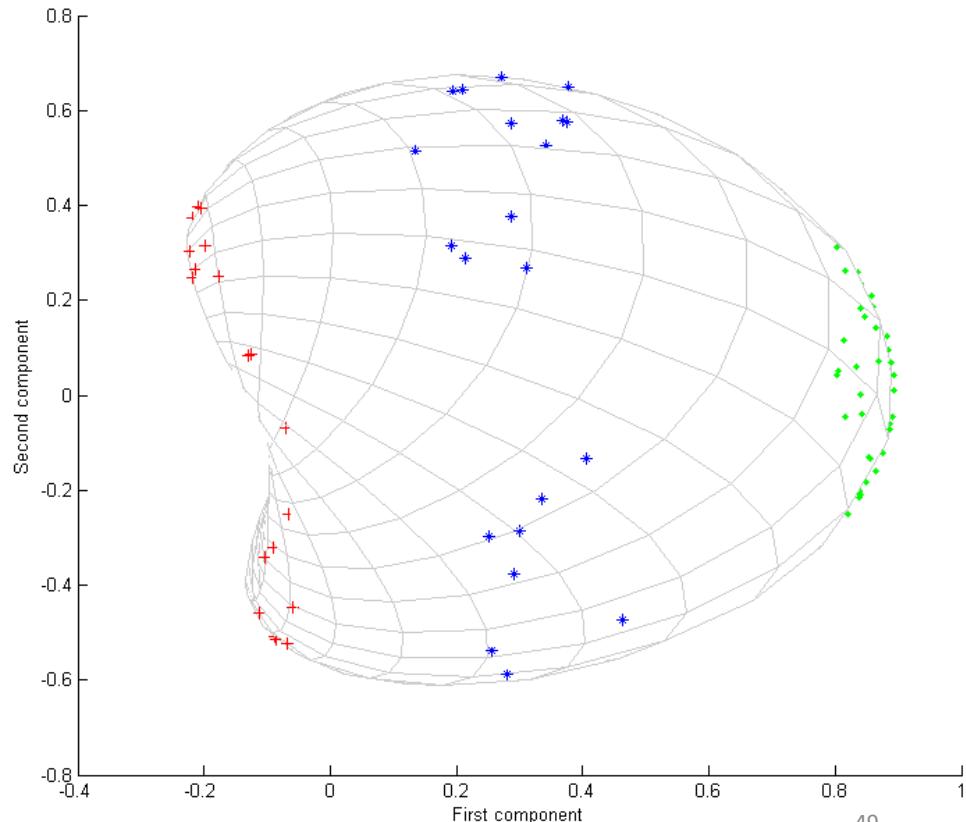
- Projection into 1<sup>st</sup> and 2<sup>nd</sup> principle components
- kPCA polynomial kernel  
$$k(x_i, x_j) = (1 + x_i^T x_j)^2$$
- Points can be separated by the first projection  $y_0$





## Kernel PCA - Example

- Projection into 1<sup>st</sup> and 2<sup>nd</sup> principle components
- kPCA Gaussian kernel  
$$k(x_i, x_j) = e^{-\beta \|x_i - x_j\|_2}$$
- Points can be separated by the first projection  $y_0$





# Surface Normal



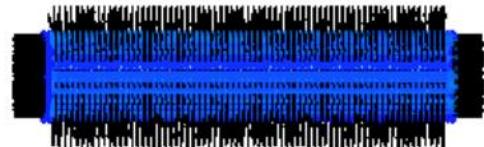
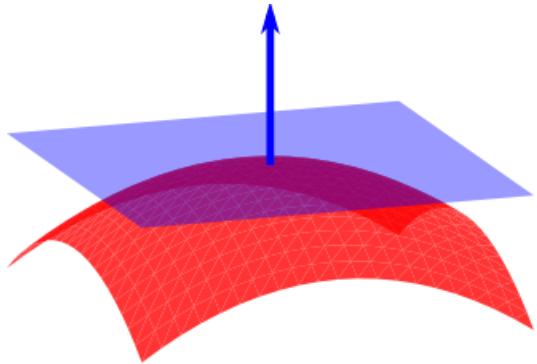
## Surface normal on surface

- 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?

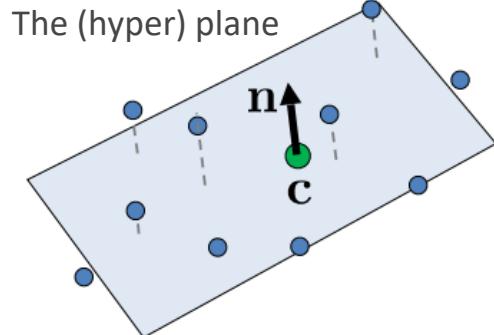


# Surface Normal Estimation – Definition

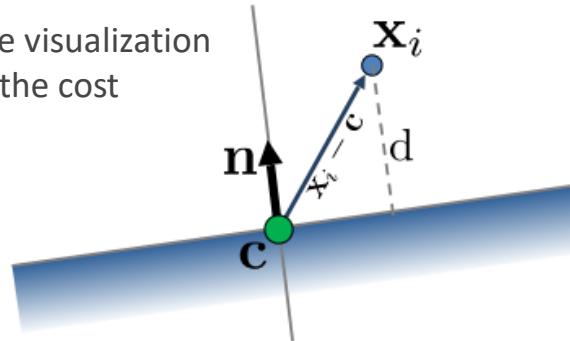
## Problem Definition

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.

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



The visualization  
of the cost



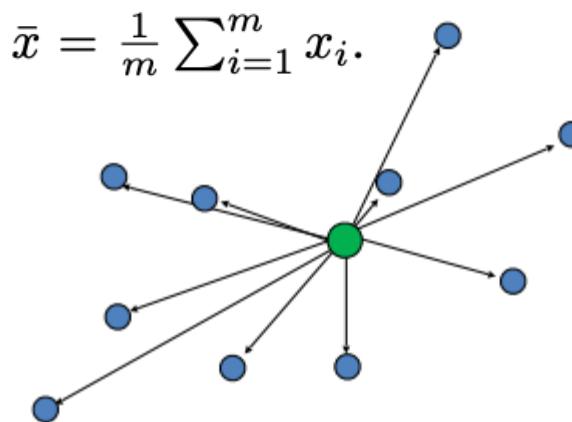


## Surface Normal Estimation – Proof

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

$$\begin{aligned} \mathbf{c}^* &= \arg \min_{\mathbf{c}} \sum_{i=1}^m ((\mathbf{x}_i - \mathbf{c})^T \mathbf{n})^2 \\ &= \arg \min_{\{c_j\}} \sum_{i=1}^m \sum_{j=1}^n (x_{ij} - c_j)^2 n_j^2 \\ &= \arg \min_{\{c_j\}} \sum_{i=1}^m \sum_{j=1}^n (x_{ij} - c_j)^2 \\ &= \arg \min_{\mathbf{c}} \sum_{i=1}^m \|\mathbf{x}_i - \mathbf{c}\|^2 \end{aligned}$$

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





## 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,

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

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

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

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

PCA:

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



# 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. RANSAC

- a. Lecture 4

4. Deep Learning!



## Surface Normal Estimation



### Weighted normal estimation

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

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

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

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

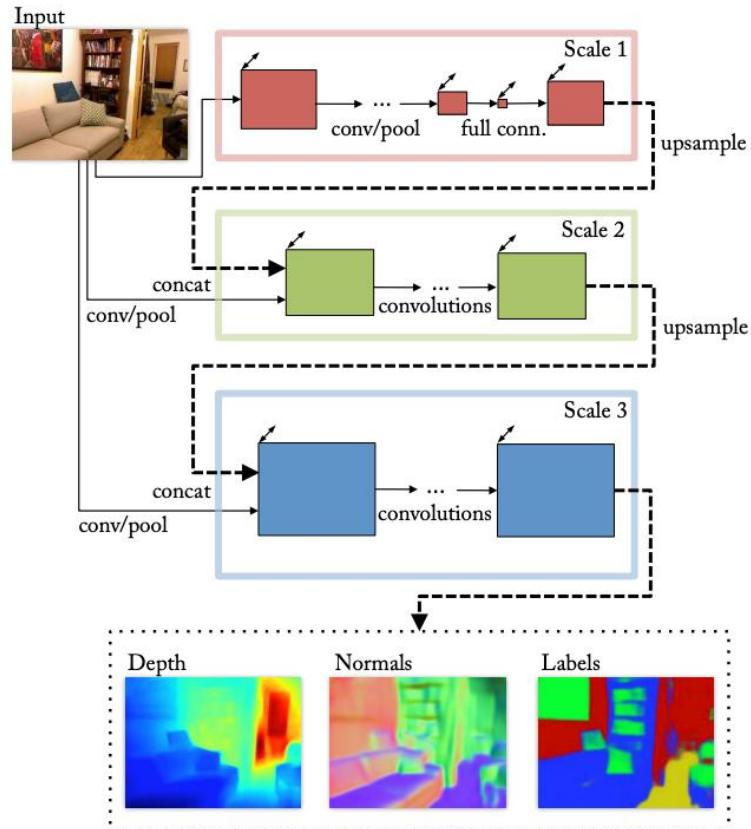
$$W = \begin{bmatrix} w_1 & 0 & \cdots & 0 \\ 0 & w_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & w_m \end{bmatrix}$$



# Deep Learning about Surface Normal

## 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

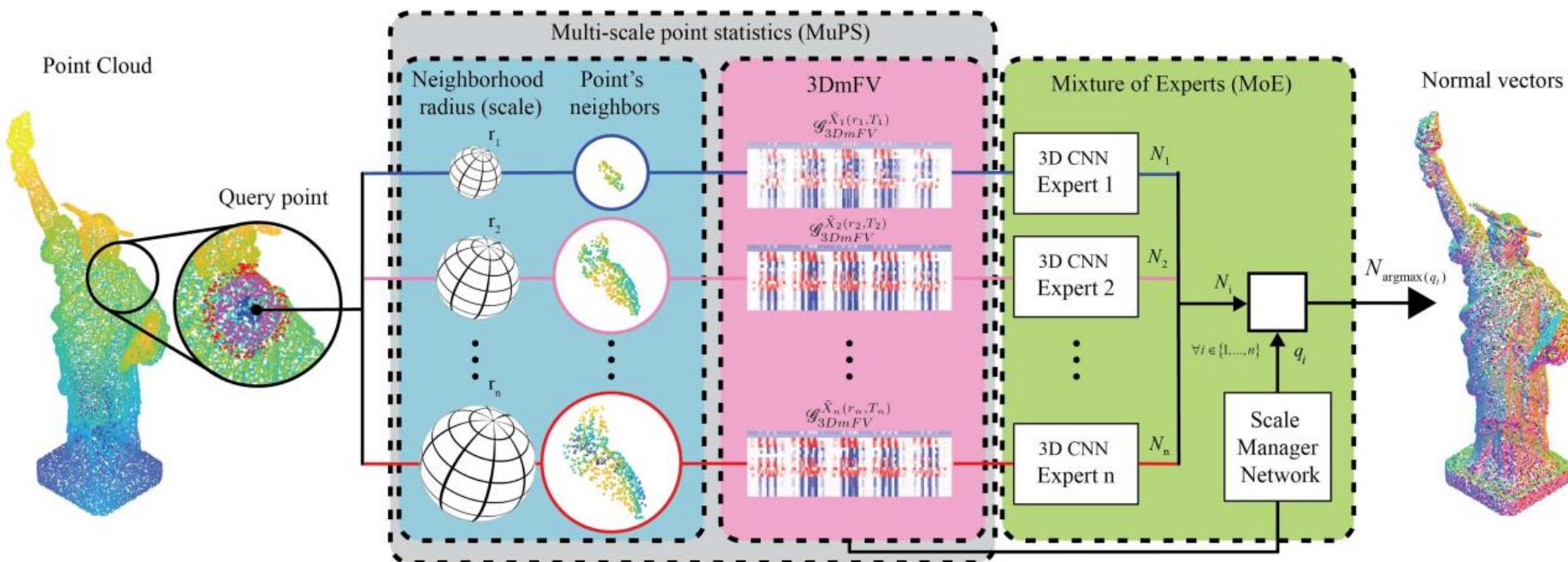




# Deep Learning about Surface Normal

## Nesti-Net: Normal Estimation for Unstructured 3D Point Clouds using Convolutional Neural Networks

- CVPR 2019





# Filters



## Noise removal

- Radius Outlier Removal
- Statistical Outlier Removal



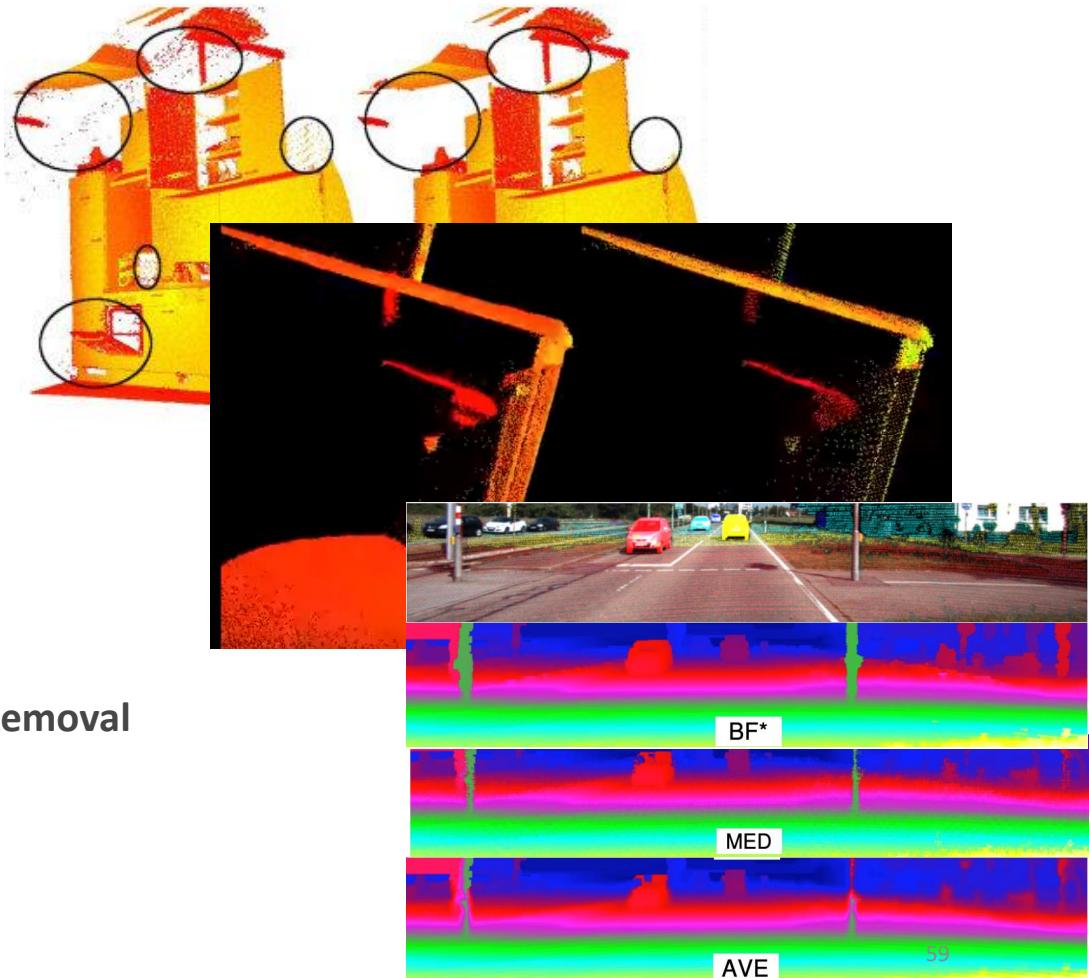
## Downsampling

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



## Upsampling / Smoothing / Noise Removal

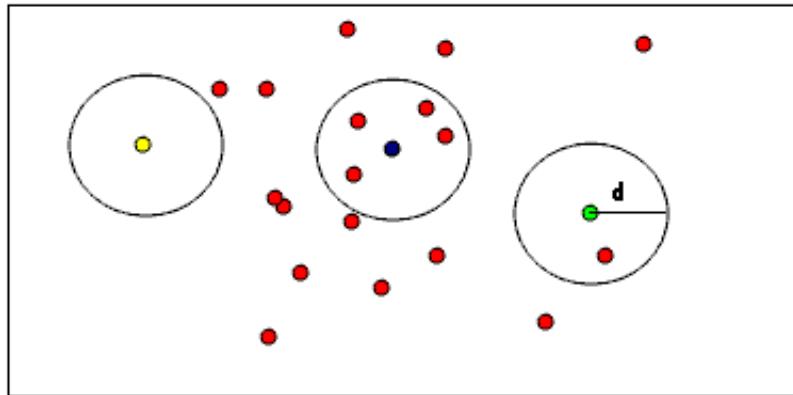
- Bilateral Filter





## 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  $d \sim N(\mu, \sigma)$

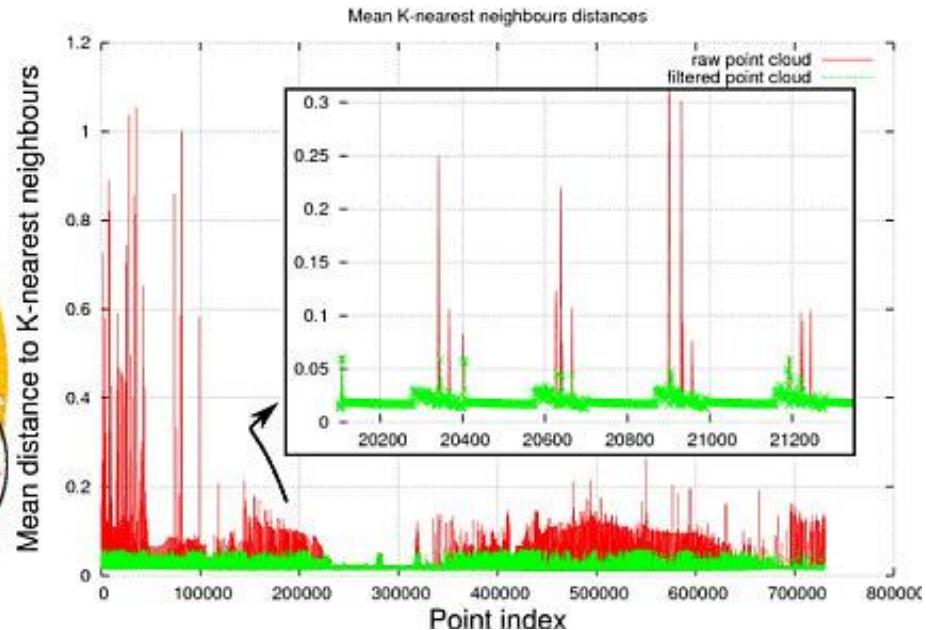
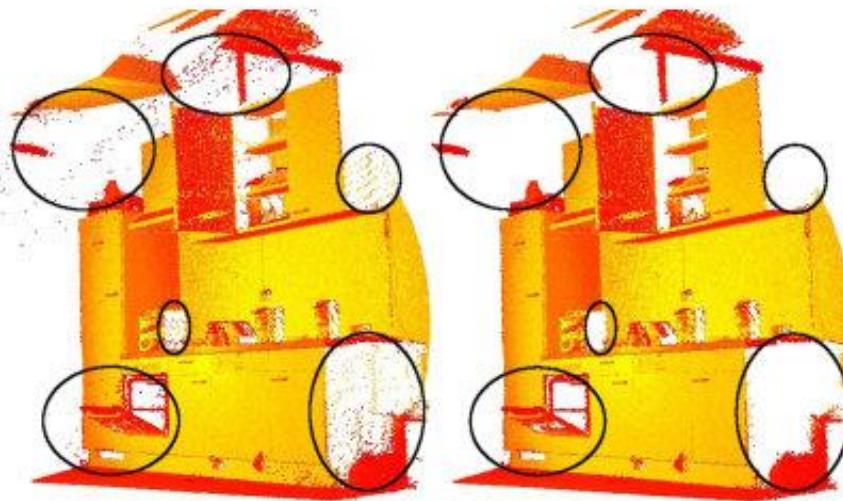
$$\mu = \frac{1}{nk} \sum_{i=1}^m \sum_{j=1}^k d_{ij}, \quad \sigma = \sqrt{\frac{1}{nk} \sum_{i=1}^m \sum_{j=1}^k (d_{ij} - \mu)^2}$$

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

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



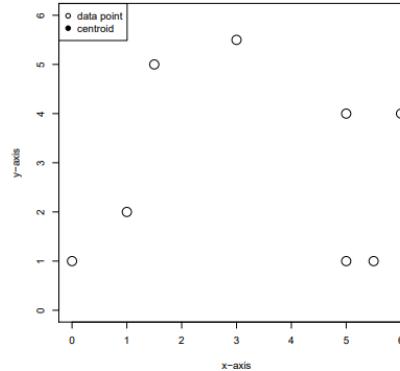
## Statistical Outlier Removal





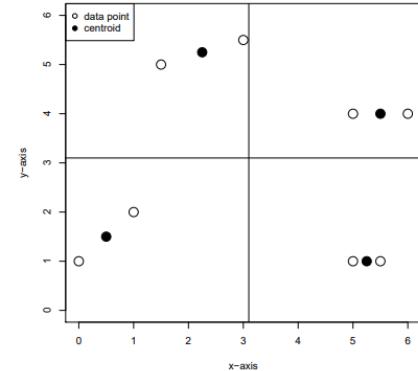
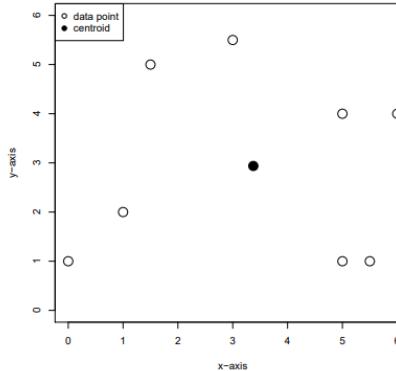
# Voxel Grid Downsampling

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
2. Random select
  - a. Randomly select a point in the cell
  - b. Less accurate but faster



## 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, 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"><li>• For all a, <code>comp(a,a)==false</code></li><li>• If <code>comp(a,b)==true</code> then <code>comp(b,a)==false</code></li><li>• if <code>comp(a,b)==true</code> and <code>comp(b,c)==true</code> then <code>comp(a,c)==true</code></li></ul>

- 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**
- ❖ Imagine we have  $N=10000$  points, we know after downSampling 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

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

E.g.,  $\text{hash}(h_x, h_y, h_z) = (\textcolor{red}{h_x} + h_y * D_x + h_z * D_x * D_y) \% \text{container\_size}$



## 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!

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

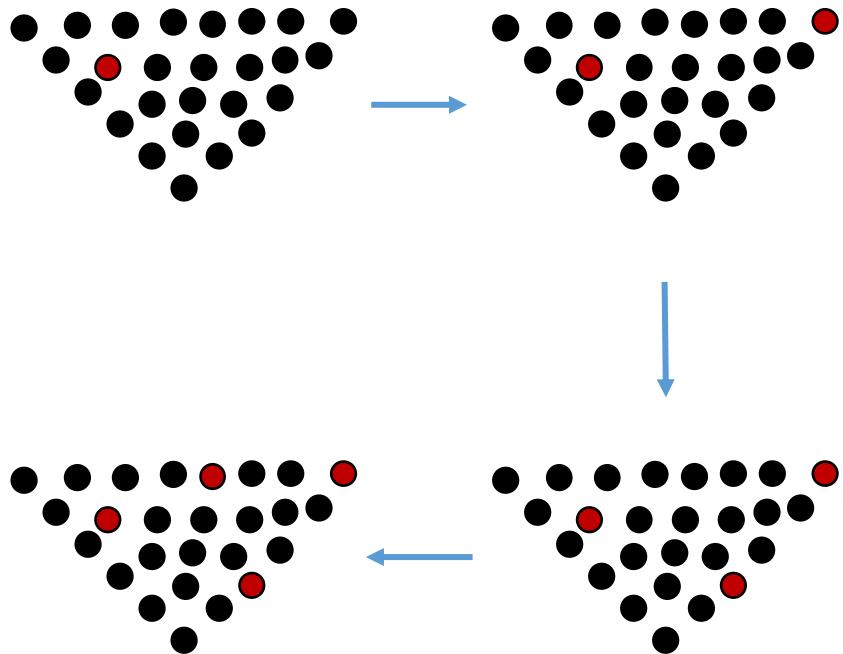
Hexagon icon: If you find a *conflict*, what do you do?

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



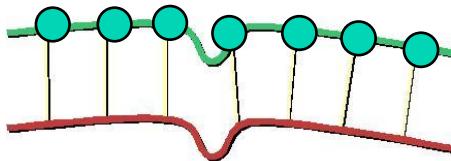


# Normal Space Sampling (NSS)

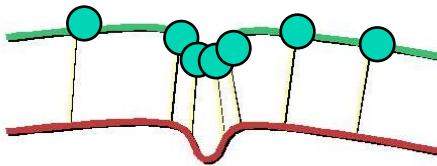


## Used in Iterative Closest Point

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



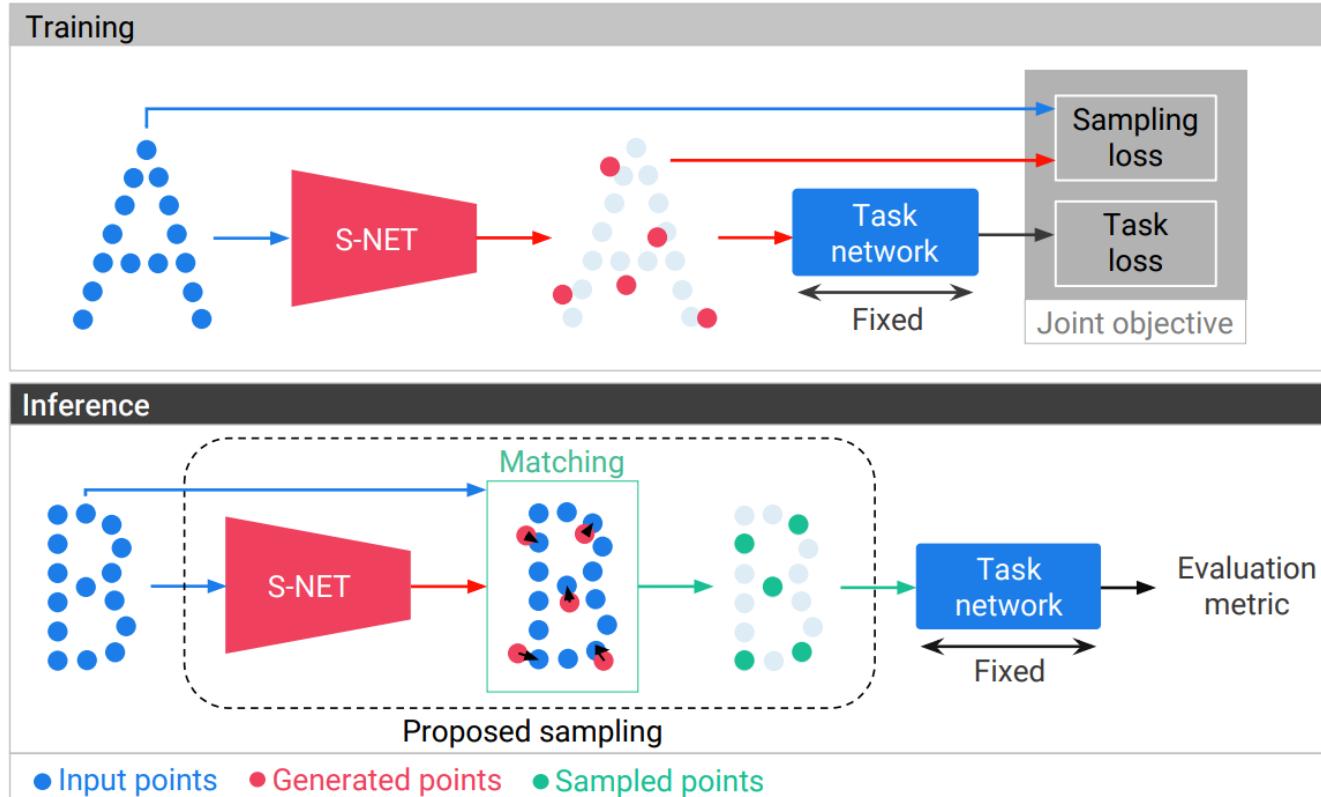
Uniform Sampling



Normal Space Sampling



# Learning to Sample



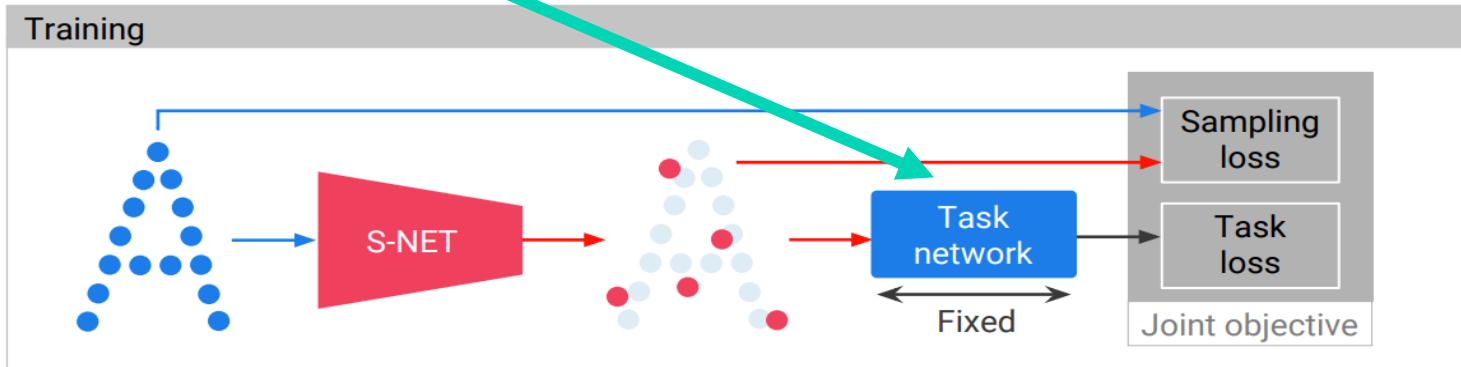


# Learning to Sample

**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

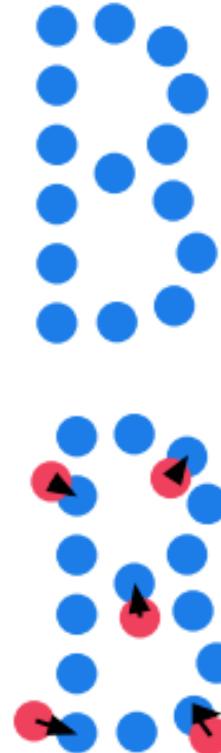
Let us denote the generated point set as  $G$  and the input point set as  $P$ . We construct a sampling regularization loss, composed out of three terms:

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

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

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

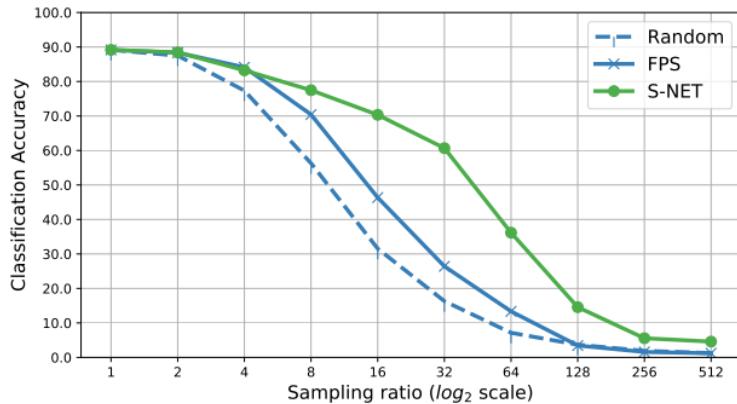
(2) + (4) = Chamfer loss





## 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	<b>88.3</b>	87.8
256	86.6	88.1	<b>88.3</b>
128	86.2	87.9	<b>88.6</b>
64	81.5	86.1	<b>87.7</b>
32	77.0	82.2	<b>87.3</b>
16	65.8	76.7	<b>85.6</b>
8	45.8	61.6	<b>83.6</b>
4	26.9	35.2	<b>73.4</b>
2	16.6	18.3	<b>53.0</b>



## Learning to Sample – for reconstruction



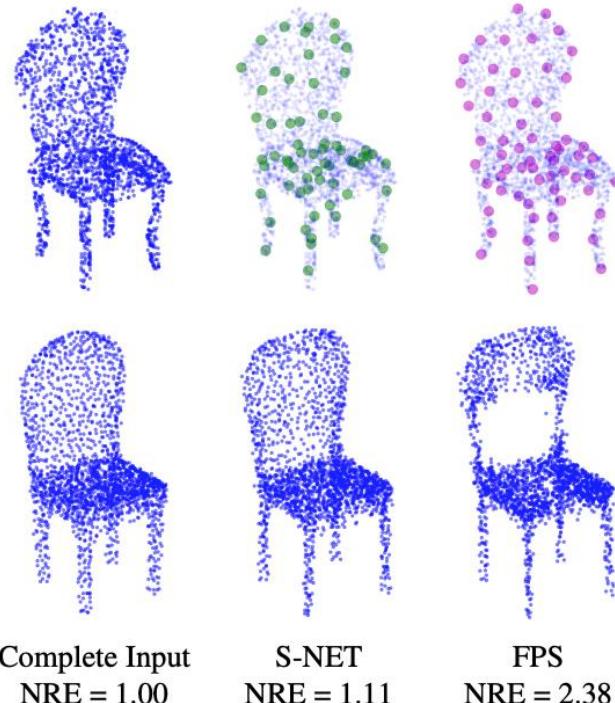
**NRE:**

Normalized Reconstructed Error



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

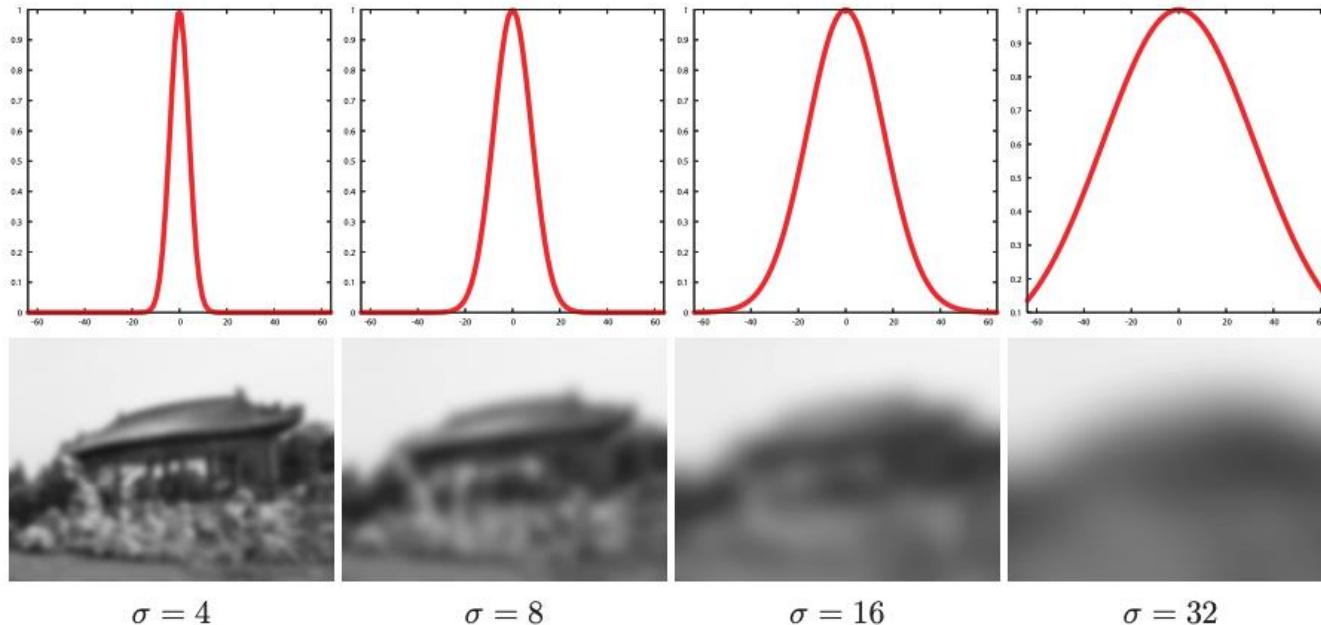
- This is expected because of Chamfer loss





## 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  $(p, q)$

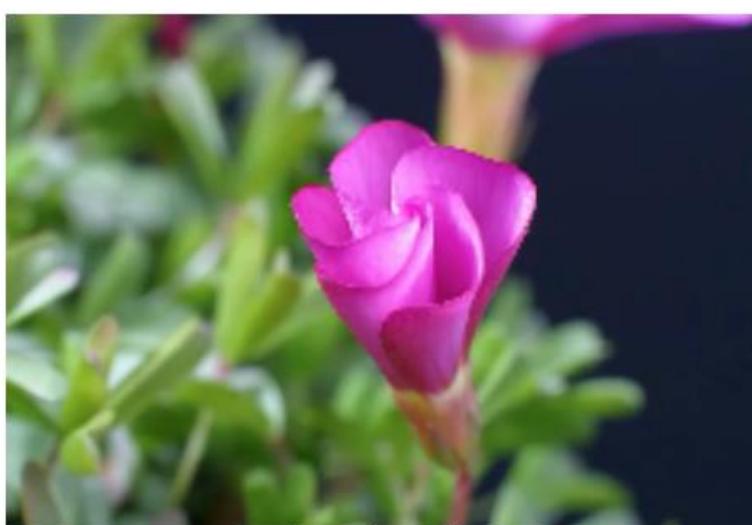
- Compute distance weight  $G_{\sigma_s}$  intensity weight  $G_{\sigma_r}$

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

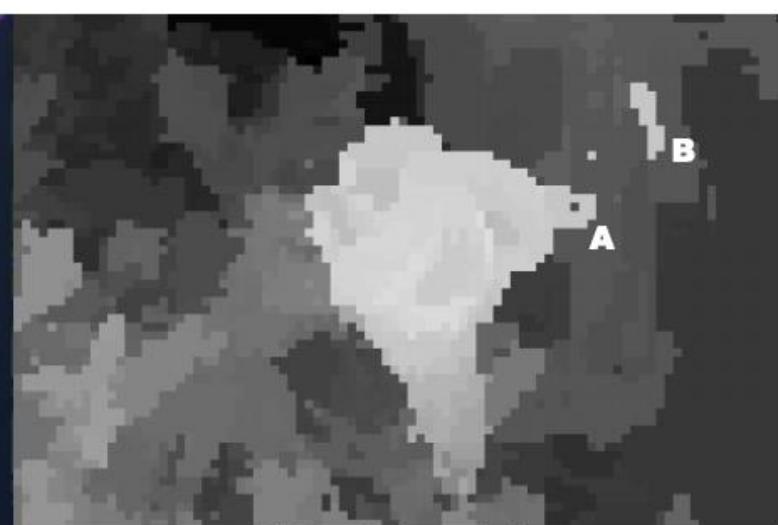
- Apply Bilateral Filter to get intensity of pixel  $p$

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

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

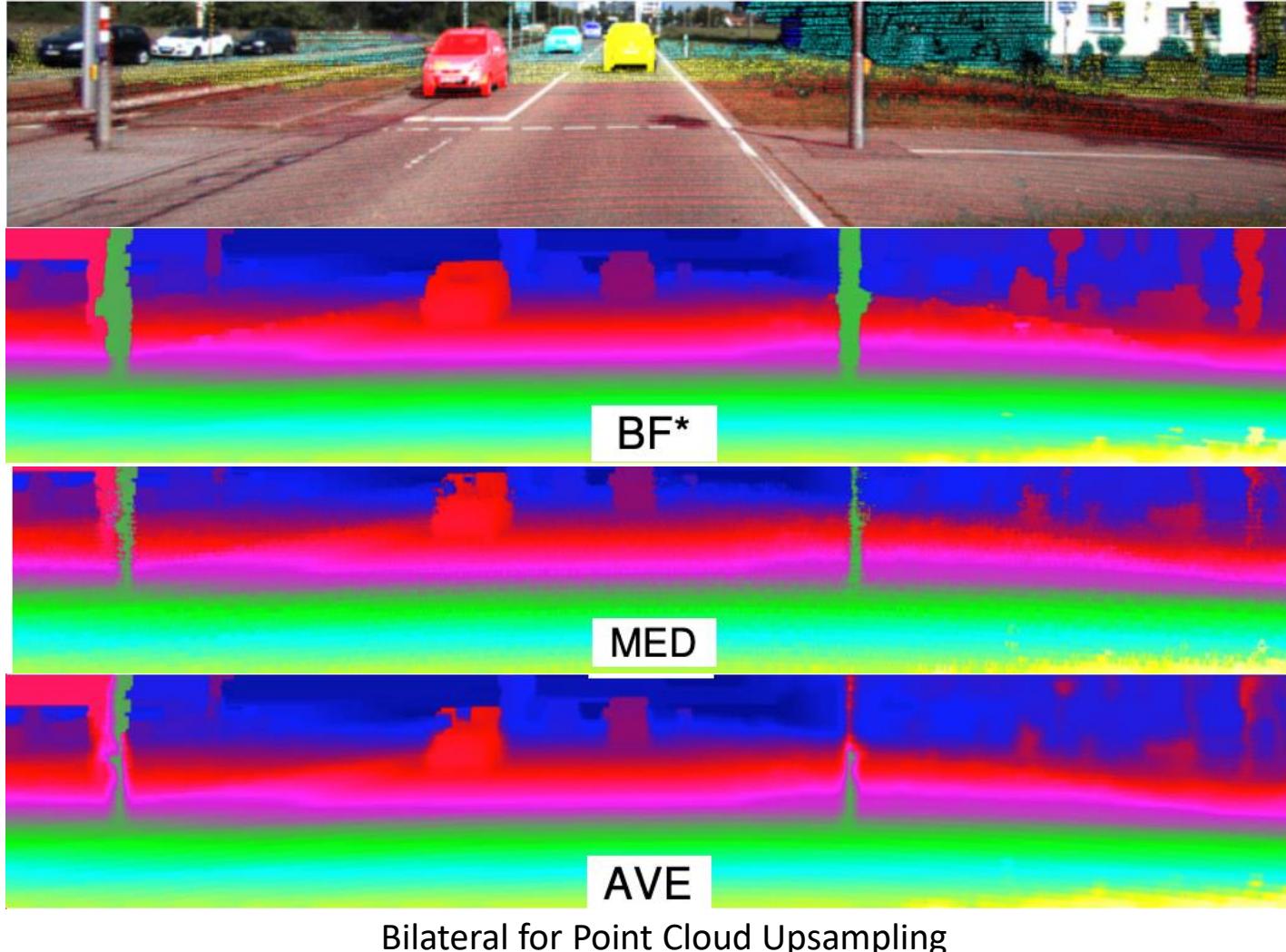


a) Input image Flower



b) Input depth map ( $8 \times 8$ )







## 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](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.

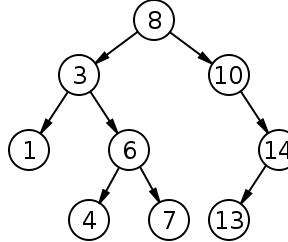


# Lecture Outline



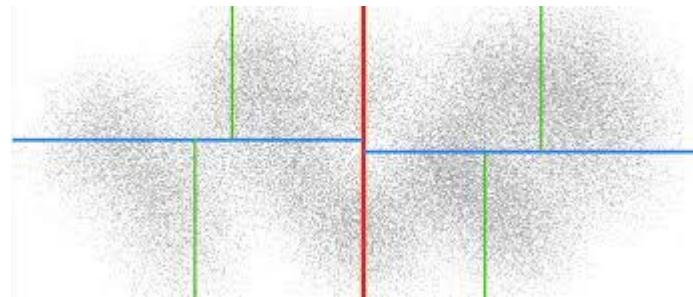
## Binary Search Tree

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



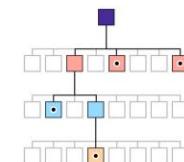
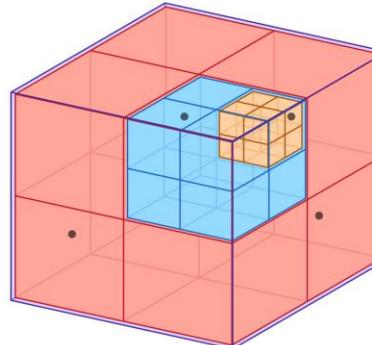
## Kd-tree

- 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!

