CMPUT 307

Point Clouds

Subhayan Mukherjee, Shupei Zhang & Anup Basu

Overview

-Intro, Motivation and Context

-Point Cloud Processing

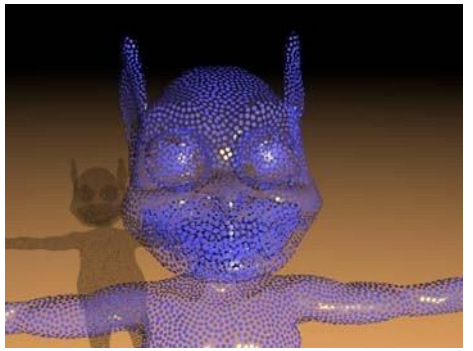
-Splatting

-Examples of some Tools, Videos etc.

Why use points ?

Advantages

- ✓ Natural representation for many 3D acquisition systems
- ✓ No separation of geometry and appearance/attributes
- ✓ No separation of surfaces and volumes

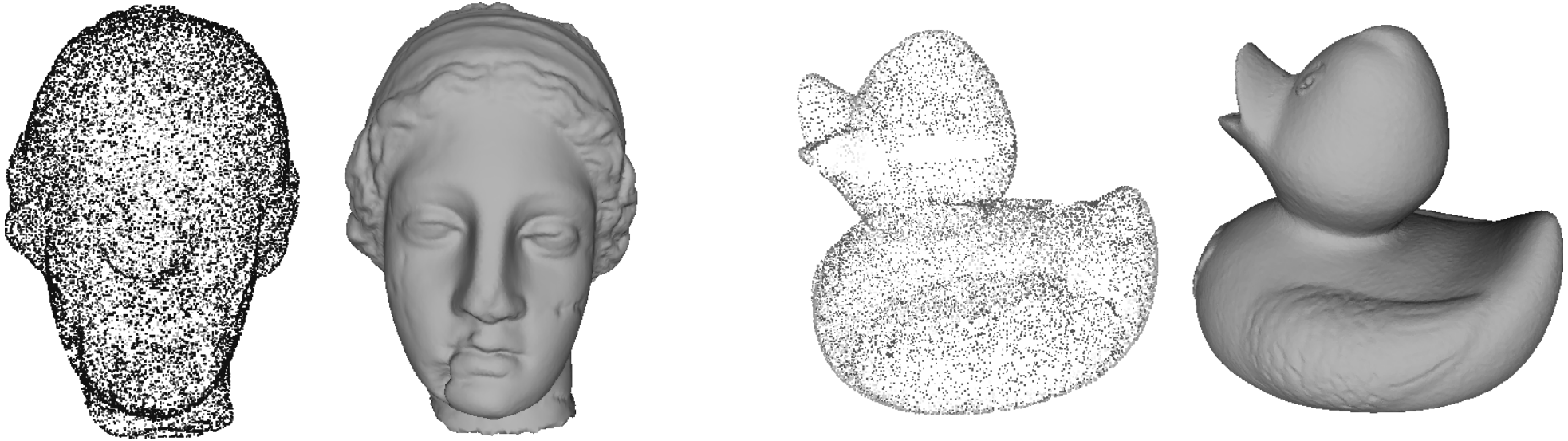


Disadvantages

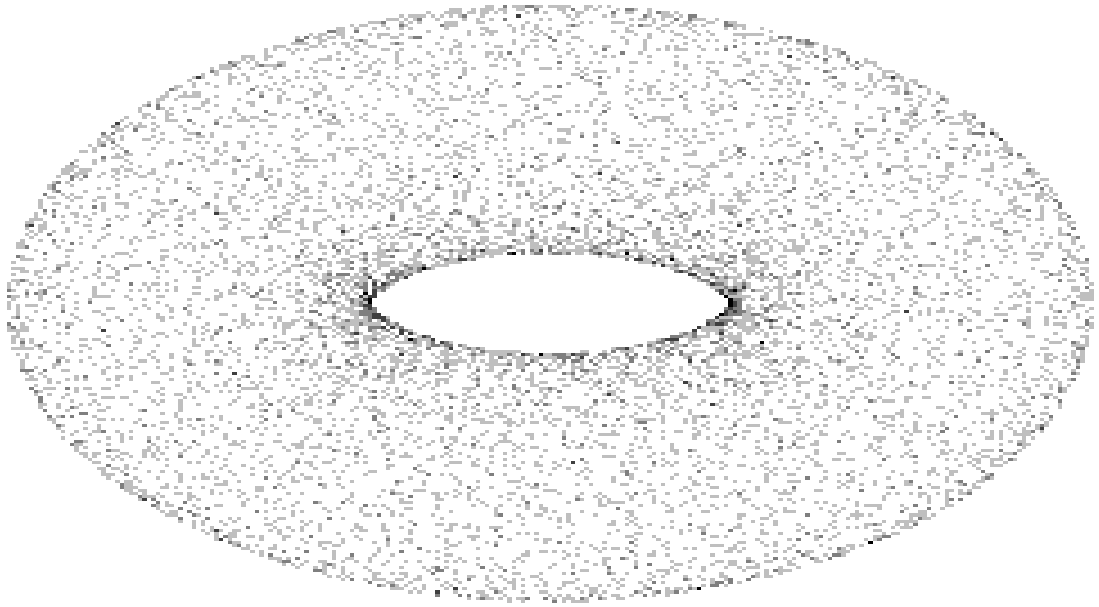
- x No connectivity or topology

Some Examples of Point Clouds

OBJECTS

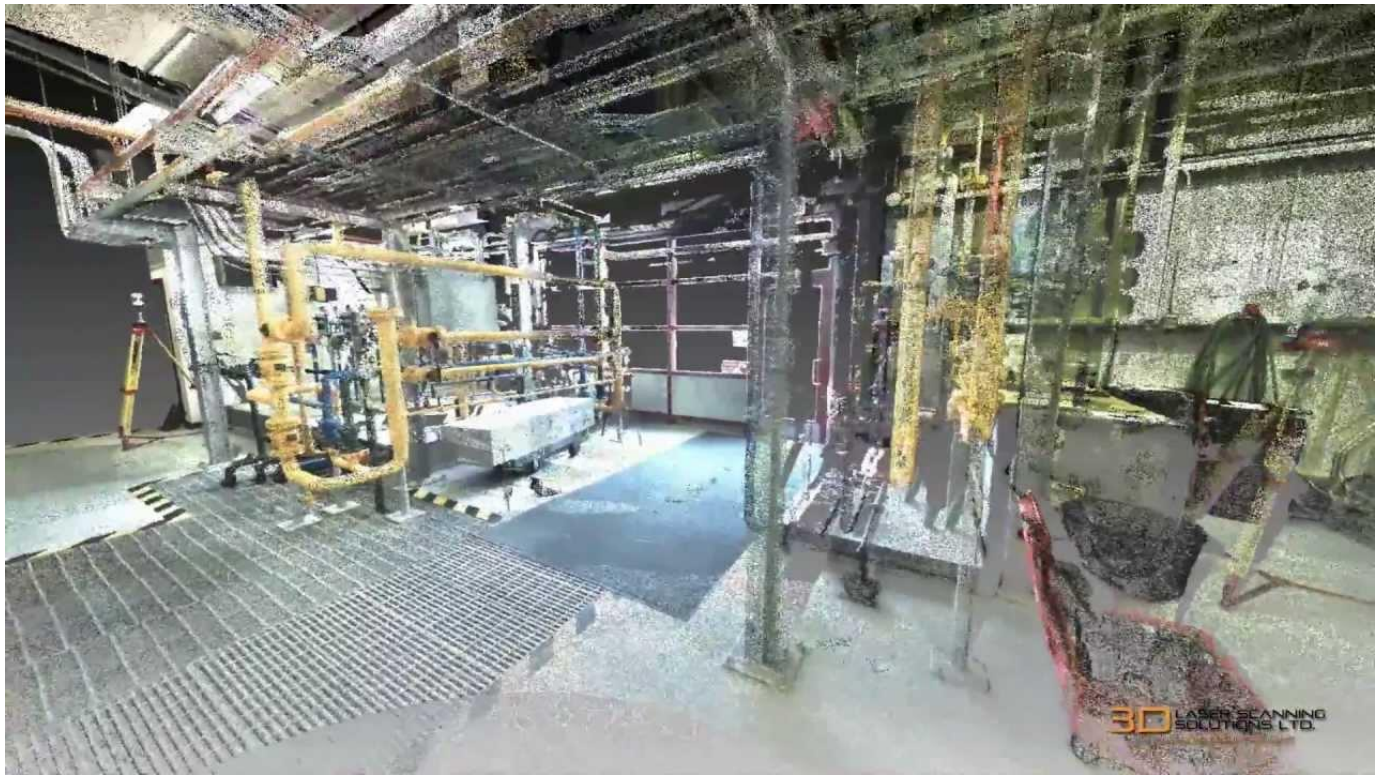


Some Examples of Point Clouds (A Torus)



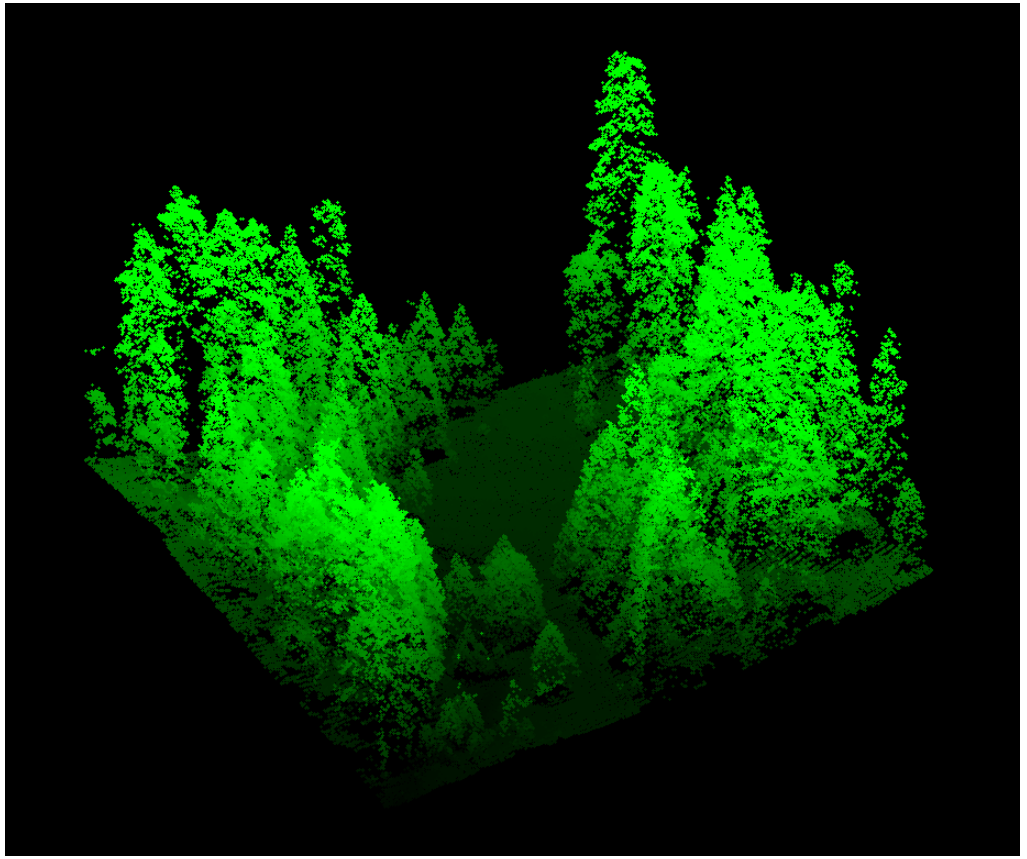
Some Examples of Point Clouds

Factories



Some Examples of Point Clouds

Trees

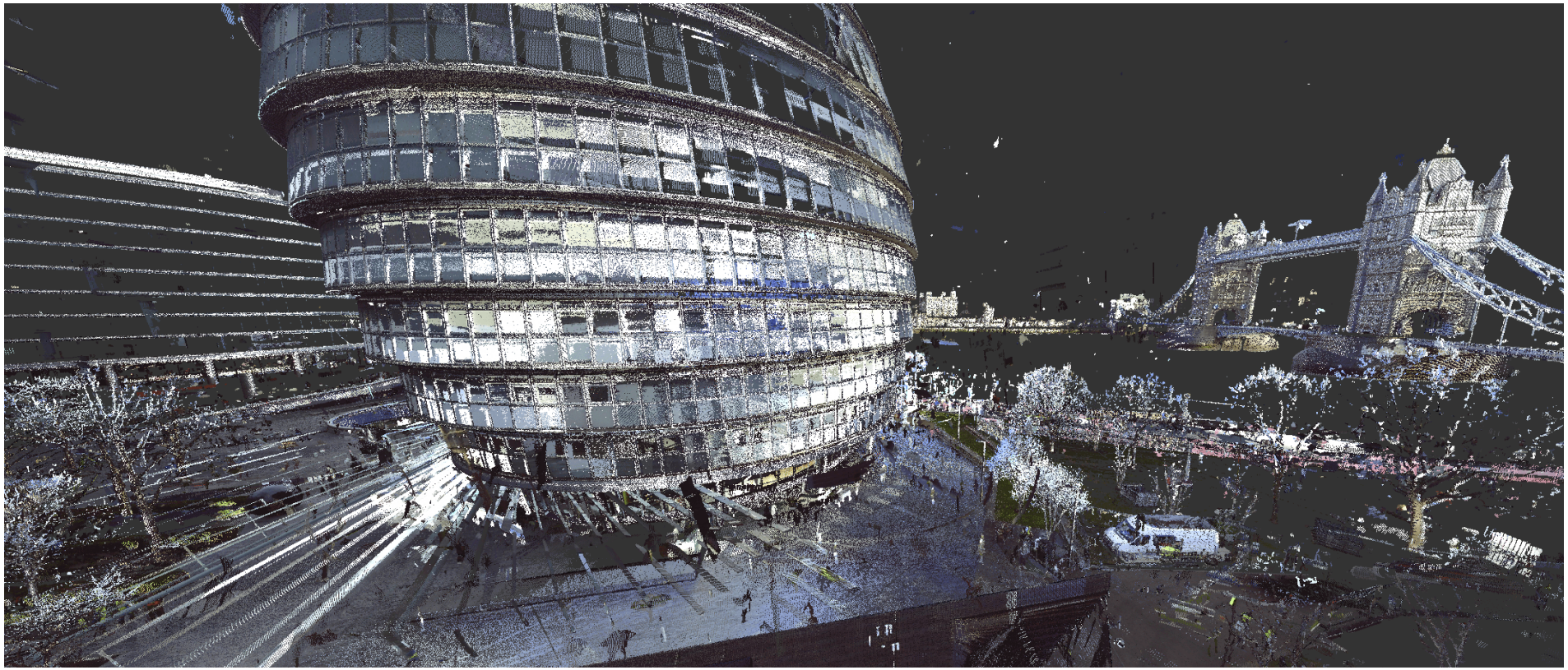


Some Examples of Point Clouds

3D Topology of Land Surfaces

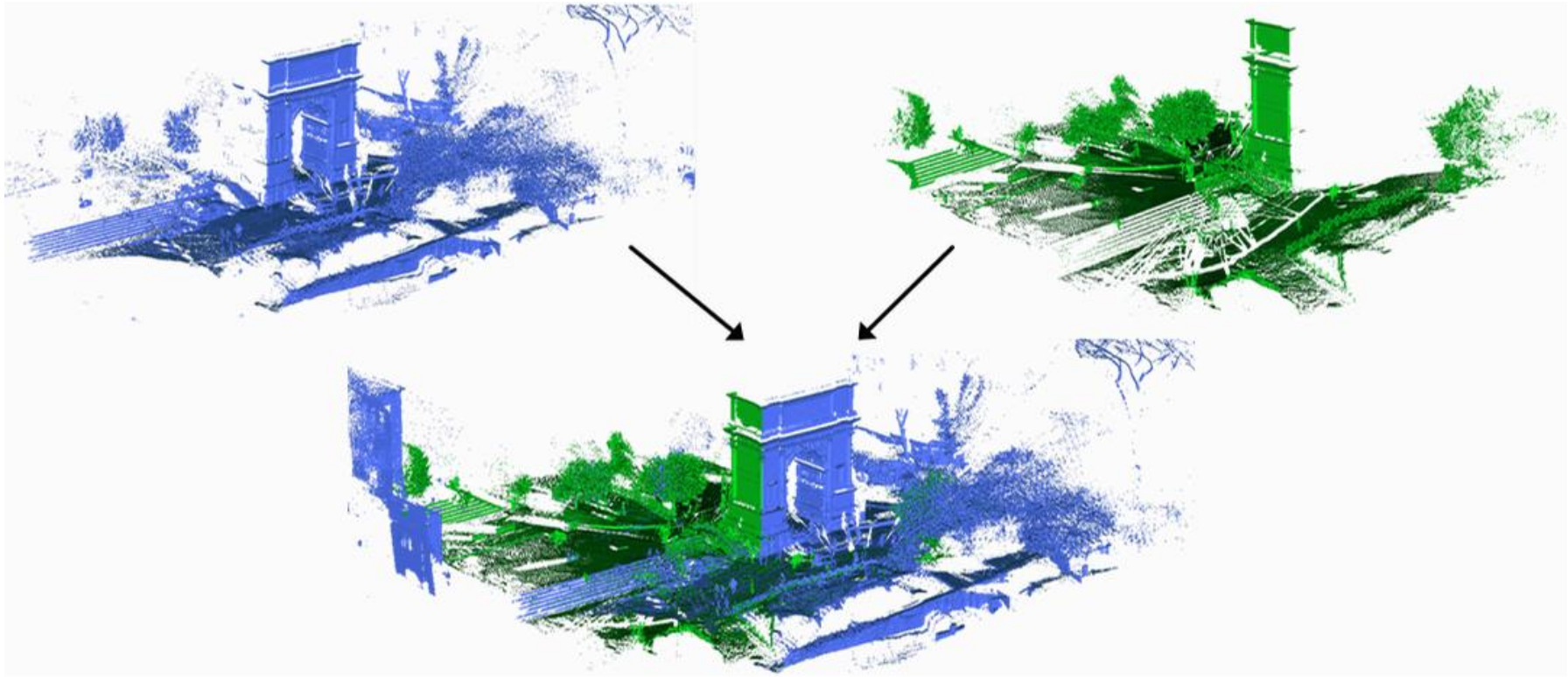
Some Examples of Point Clouds

3D Mapping of Cities

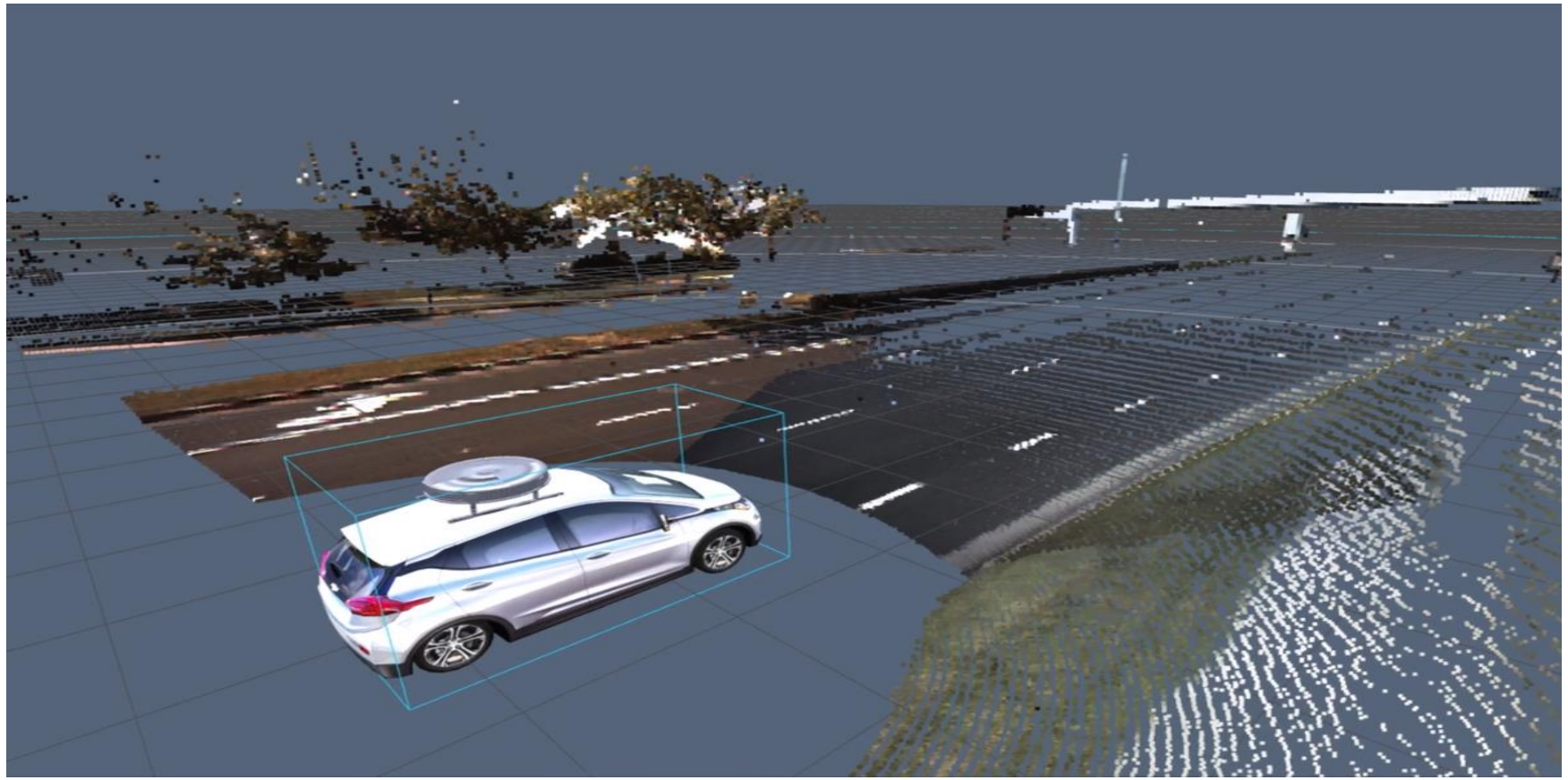


Registration of Point Clouds

Registering 2 Views of Arc du Triumph in Paris



Self Driving Cars using Point Clouds



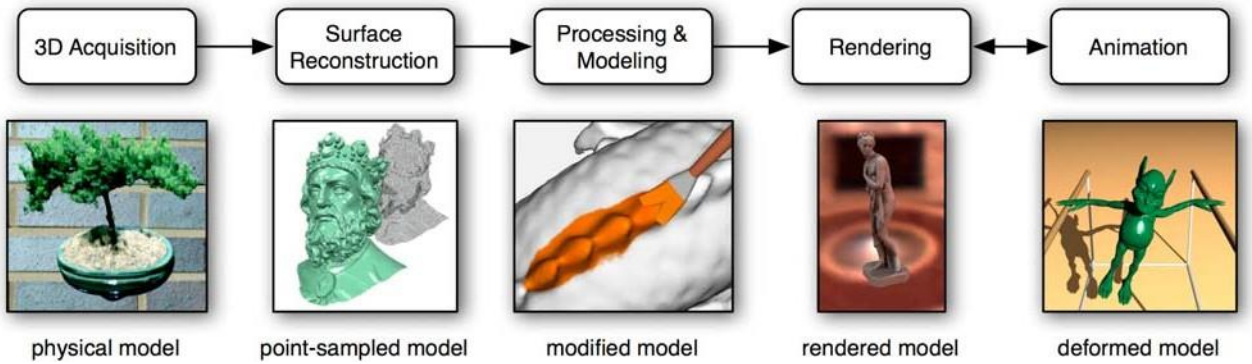
History of Points in Graphics

Images, videos, links for items in ***bold*** in later slides.

- Particle systems [Reeves 1983]
- Points as a display primitive [Whitted, Levoy 1985]
- Oriented particles [Szeliski, Tonnesen 1992]
- Particles and implicit surfaces [Witkin, Heckbert 1994]
- Rendering Architectures [Grossmann, Dally 1998]
- Surfels [Pfister et al. 2000]
- ***QSplat*** [Rusinkiewicz, Levoy 2000] https://www.youtube.com/watch?v=uYHq5QzX-U4
- Point Clouds [Linsen, Prautzsch 2001]
- Point set surfaces [Alexa et al. 2001]
- Radial basis functions [Carr et al. 2001]
- Surface splatting [Zwicker et al. 2001]
- Randomized z-buffer [Wand et al. 2001]
- Sampling [Stamminger, Drettakis 2001]
- ***Pointshop3D*** [Zwicker, Pauly, Knoll, Gross 2002]
- ***Boolean Operations*** [Adams et al. 2003]
- Modeling [Pauly et al. 2003]
- ***blue-c*** [Gross et al. 2003] https://www.youtube.com/watch?v=LOhE5h45qIM
- Meshless Physics [Muller et al. 2004]
- Spherical MLS [Guennebaud, Gross 2007]
- Multiview PointClouds [Azari, Cheng, Basu, 2013]
- Dynamic PointClouds [Azari, Basu, 2014]

Points – A Motivation

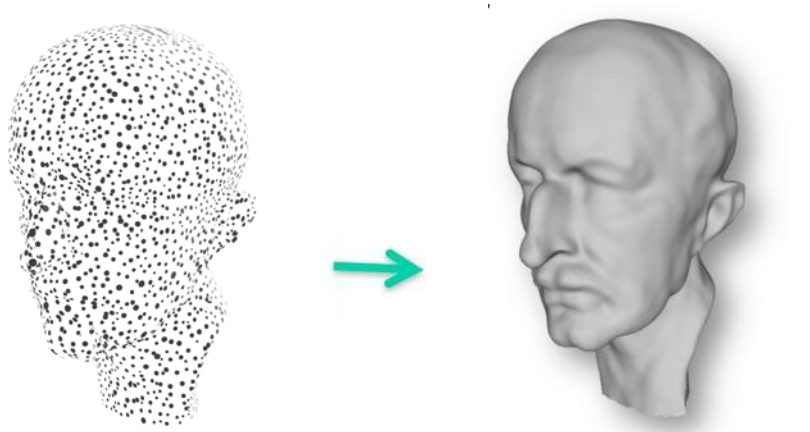
3D content creation pipeline *1997-2009*



Points generalize Pixels !

Surface Model

Compute continuous surfaces from a set of discrete point samples



discrete set of point samples

continuous surface interpolating or approximating the set of points

Example of fitting polygon surface to point clouds:
https://www.youtube.com/watch?v=xmNCulbd4c0

Point-Cloud Processing

- ✓Data representation: PCA and its variants, KD-trees, KNN-graphs.
- ✓Data segmentation: K-means, Gaussian mixture model, spectral clustering.

Data Representation

- Principal component analysis (PCA): Data is represented using an intrinsic coordinate system and projected onto a lower dimensional space (2D or 1D).
- There are many interesting variants of PCA: probabilistic PCA (PPCA), mixture of PPCA, kernel PCA, etc.
- KD-trees (K-dimensional trees): The 3D point cloud is represented as a binary 3D-tree by recursively splitting the point cloud into two subsets. Provides an efficient way to manipulate the point cloud.
- KNN-graph (K nearest neighbor graph): The 3D point cloud is represented as a sparse *undirected weighted graph*.

Data Segmentation

Few basic concepts:

- K-means clustering: Data is grouped into *K* spherical clusters. The number of clusters is provided in advance. This algorithm is often use to initialize other clustering methods.
- The Gaussian mixture model (GMM): A more sophisticated clustering method is based on a mixture of Gaussian distributions. This is generally solved using the expectation-maximization algorithm (EM).
- K-means and GMM work well on spherical or ellipsoidal groups of points. Spectral clustering operates in the *spectral space* spanned by the eigenvectors of the symmetric matrix associated with a KNN-graph.
- Clustering methods need KD-trees for efficiently accessing the data points.

Some Notations and Definitions

Let's start with a few more notations:

The input (observation) space: $\mathbf{X} = [x_1 \ldots x_i \ldots x_n]$, $x_i \in \mathrm{R}3$

The output (latent) space: $\mathbf{Y} = [y_1 \ldots y_i \ldots y_n]$,
$y_i \in \mathrm{R}^d, 1 \leq d \leq 3$

Projection: $\mathbf{Y} = \mathbf{Q}^{\mathrm{T}}\mathbf{X}$ with $\mathbf{Q}^{\mathrm{T}}$ a $d \times 3$ matrix. (Projection from 3 dimension to a lower dimension)

$\mathbf{Q}^{\mathrm{T}}\mathbf{Q} = \mathbf{I}_d$

Computing the Spread of the Data

We start with *n* scalars $x_1 \ldots x_n$; the mean and the variance are given by:

$$\overline{x} = \frac{1}{n}\sum_i x_i$$

$$\sigma_x = \frac{1}{n}\sum_i (x_i - \overline{x})^2$$

$$= \frac{1}{n}\sum_i x_i^2 - \overline{x}^2$$

Computing the Spread of the Data

More generally, for the data set **X**:

The mean: $\overline{\boldsymbol{x}} = \frac{1}{n}\sum_i \boldsymbol{x}_i$

The covariance matrix is *semi-definite positive symmetric* of dimension 3 × 3:

$$\begin{aligned}\mathbf{C}_X &= \frac{1}{n}\sum_i (\boldsymbol{x}_i - \overline{\boldsymbol{x}})(\boldsymbol{x}_i - \overline{\boldsymbol{x}})^\top \\ &= \frac{1}{n}\mathbf{X}\mathbf{X}^\top - \overline{\boldsymbol{x}}\,\overline{\boldsymbol{x}}^\top\end{aligned}$$

Note: **Boldface indicates vectors**.

Maximum-Variance Formulation of PCA

Let's center and project the data **X** onto a line along a unit vector *u*. The variance along this line can be written as:

$$\begin{aligned}
\sigma_u &= \frac{1}{n}\sum_i (\boldsymbol{u}^\top(\boldsymbol{x}_i - \overline{\boldsymbol{x}}))^2 \\
&= \boldsymbol{u}^\top \left(\frac{1}{n}\sum_i (\boldsymbol{x}_i - \overline{\boldsymbol{x}})(\boldsymbol{x}_i - \overline{\boldsymbol{x}})^\top \right) \boldsymbol{u} \\
&= \boldsymbol{u}^\top \mathbf{C}_X \boldsymbol{u}
\end{aligned}$$

Maximum-Variance Formulation of PCA

Find *u* maximizing the variance under the constraint that *u* is a unit vector:

$$\boldsymbol{u}^{\star} = \underset{\boldsymbol{u}}{\operatorname{argmax}} \left\{ \boldsymbol{u}^{\top} \mathbf{C}_X \boldsymbol{u} + \lambda(1 - \boldsymbol{u}^{\top}\boldsymbol{u}) \right\}$$

Maximum-Variance Solution

First note that the 3 × 3 covariance matrix is a symmetric semi-definite positive matrix. (The associated quadratic form above is non-negative).

Taking the derivative with respect to u and setting the derivatives equal to 0, yields: $\mathbf{C}_X u = \lambda u$

Making use of the fact that u is a unit vector we obtain: $\sigma_u = \lambda$

Solution: The *principal* or largest eigenvector–eigenvalue pair (u_{max}, λ_{max}) of the covariance matrix.

Eigen-decomposition of the Covariance Matrix

Assume that the data is centered: $n\mathbf{C}_X = \mathbf{X}\mathbf{X}^\top = n\mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$

Where **U** is a 3 × 3 orthogonal matrix and **Λ** is the diagonal matrix of eigenvalues: **Λ** = [$\lambda_1\,\lambda_2\,\lambda_3$]

If the point-cloud lies on a lower *d*-dimensional space (collinear or planar points):

d = rank(**X**) < 3

and

Λ$_d$ = [$\lambda_1\,\lambda_d$] $\mathbf{C}_X = \widetilde{\mathbf{U}}\mathbf{\Lambda}_d\widetilde{\mathbf{U}}^\top$

$\widetilde{\mathbf{U}}$ = **U**$\mathbf{I}_{3\times d}$ is a 3 × *d* column-orthogonal matrix

$\widetilde{\mathbf{U}}^\top = \mathbf{I}_{d\times 3}^\top\mathbf{U}^\top$ is a *d* × 3 row-orthogonal matrix

Data Representation in the Eigen (Sub)space

Coordinate change: **Y** = **QX**; We have

$$\mathbf{Y}\mathbf{Y}^\top = \mathbf{Q}\mathbf{X}\mathbf{X}^\top\mathbf{Q}^\top = n\mathbf{Q}\widetilde{\mathbf{U}}\mathbf{\Lambda}_d\widetilde{\mathbf{U}}^\top\mathbf{Q}^\top$$

The projected data have a diagonal covariance matrix: $\frac{1}{n}\mathbf{Y}\mathbf{Y}^\top = \mathbf{\Lambda}_d$

by identification we obtain $\mathbf{Q} = \widetilde{\mathbf{U}}^\top$

The projected data have an identity covariance matrix, this is called *whitening the data*:

$$\frac{1}{n}\mathbf{Y}\mathbf{Y}^\top = \mathbf{I}_d \qquad \mathbf{Q} = \mathbf{\Lambda}_d^{-\frac{1}{2}}\widetilde{\mathbf{U}}^\top$$

Projection of the data points onto principal direction u_i:

$$(y_{1i}\dots y_{ni}) = \underbrace{\lambda_i^{-1/2}}_{\text{whitening}}\boldsymbol{u}_i^\top(\boldsymbol{x}_1\dots\boldsymbol{x}_n)$$

Summary of PCA

The eigenvector-eigenvalue pairs of the covariance matrix correspond to a *spectral* representation of the point cloud, or a *within representation*.

This eigen-decomposition allows us to reduce the dimensionality of the point cloud to one plane or one line and then to project the cloud onto such a linear subspace.

The largest eigenvalue-eigenvector pair defines the direction of maximum variance. By projecting the data onto this line one can order the data (useful for data organization, i.e., KD-trees).

The eigenvalue-eigenvector pairs can be efficiently computed using the power method: get a random unit vector $x^{(0)}$ and iterate $x^{(k+1)} = \mathbf{C}x^{(k)}$, normalize $x^{(k+1)}$, etc., until

$| x^{(k+1)} - x^{(k)} |$ k $< \varepsilon$. Then $u_{max} = x^{(k+1)}$.

Simplified Steps to Compute Eigenvalues and Eigenvectors

1. Compute the Covariance Matrix A

2. Find Determinant of (A - λI); Equate this to 0 & Solve for different Roots for λ

3. Arrange the Roots in Decreasing Order; The Largest Gives the Eigen value for the First Principal Component; the second largest is the Eigen Value for the Second Principal Component & so on.

4. We still need to compute the Normalized Eigen Vectors, which are the Principal Components

Computing the Eigenvectors in 2D

Considering an arbitrary set of 2D points, let the first eigenvalue be λ_1 and the first eigenvector be X_1.
Let's denote the covariance matrix obtained for the set of points, A = $\begin{bmatrix} v_1 & c_{12} \\ c_{12} & v_2 \end{bmatrix}$ where v_1 and v_2 are the variance terms and c_{12} is the covariance term.
We get two equations from $(A - \lambda_1 I) * X_1 = 0$ where X_1 = $\begin{bmatrix} a_1 \\ b_1 \end{bmatrix}$ and $\mathrm{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
We calculate the Eigenvector for the 1st Eigenvalue from the two equations obtained from the below:

$$\begin{bmatrix} (v_1 - \lambda_1) & c_{12} \\ c_{12} & (v_2 - \lambda_1) \end{bmatrix} * \begin{bmatrix} a_1 \\ b_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$(v_1 - \lambda_1)\, a_1 + c_{12} b_1 = 0$ (Eq. **1**)
$c_{12} a_1 + (v_2 - \lambda_1)\, b_1 = 0$ (Eq. **2**)

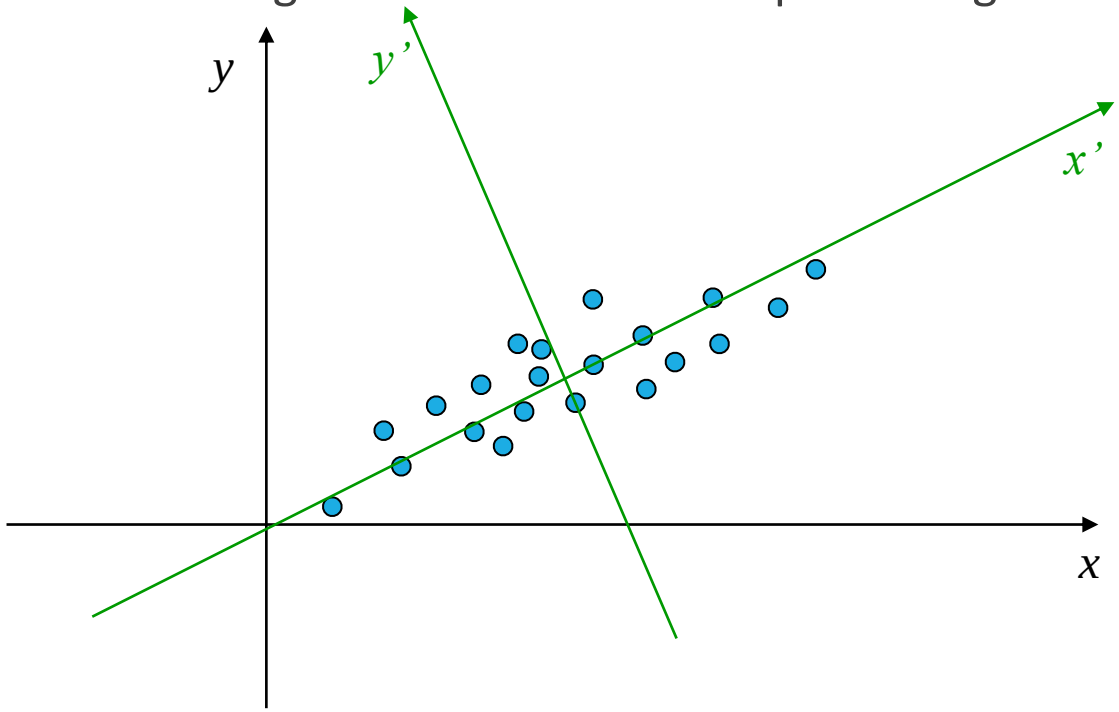
Along with the above two equations, we use the following third equation to get the eigenvector direction:

$a_1^2 + b_1^2 = 1$ (Eq. **3**)

Solving above three equations, we can get the 1st eigenvector (similar steps to get the 2nd eigenvector).

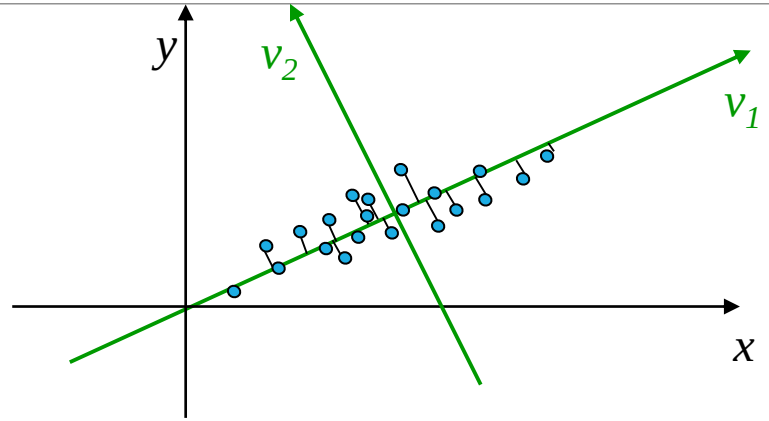
PCA – pictorial representation

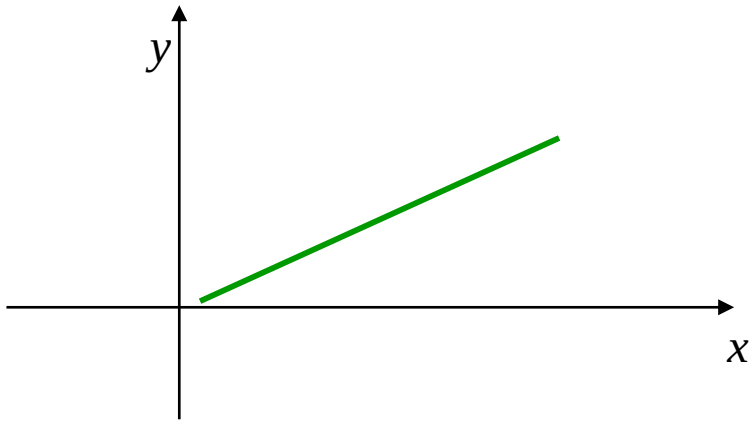
PCA finds an orthogonal basis that best represents given data set.



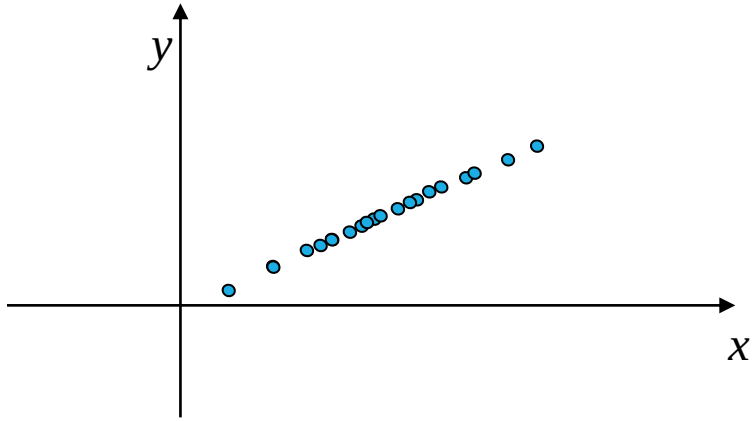
The sum of distances^2 from the x' axis is minimized.

PCA – pictorial representation





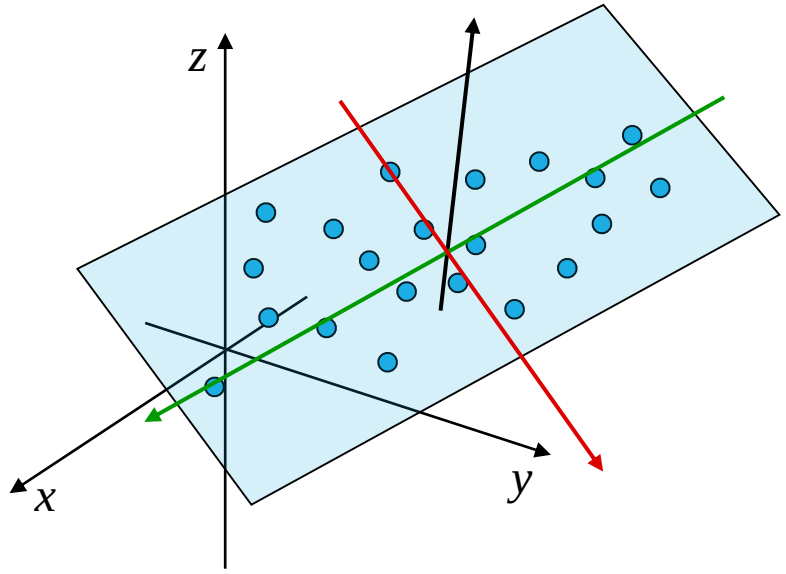
This line segment approximates the original data set



The projected data set approximates the original data set

PCA – pictorial representation

PCA finds an orthogonal basis that best represents given data set.

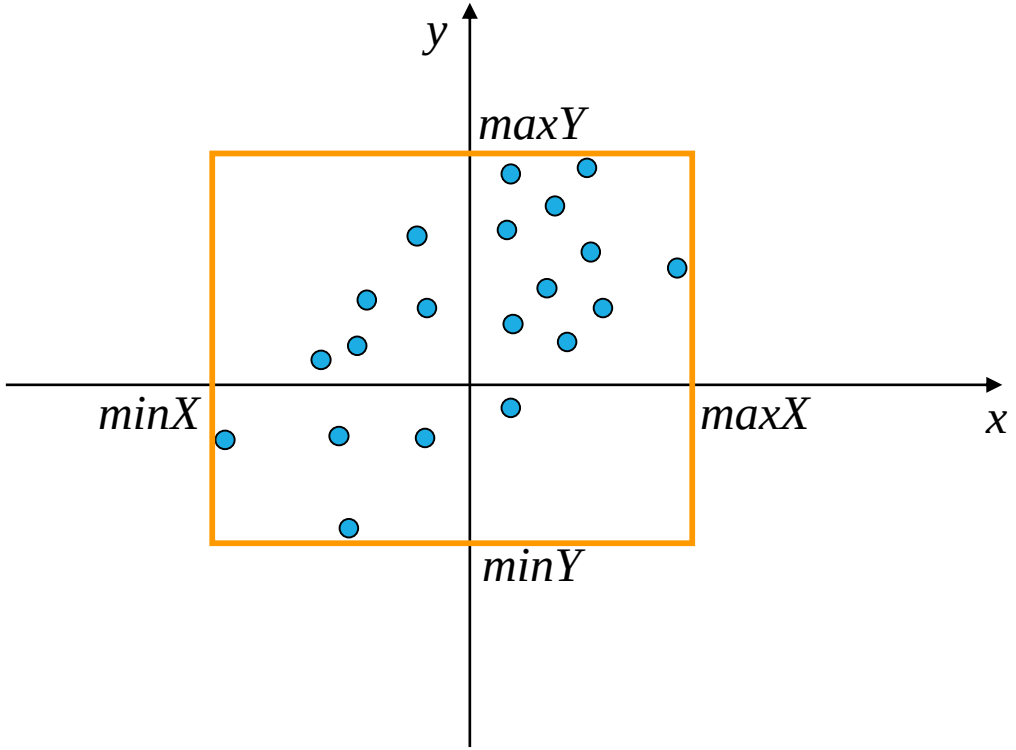


3D point set in standard basis

PCA finds a best approximating plane (again, in terms of $\Sigma distances^2$)

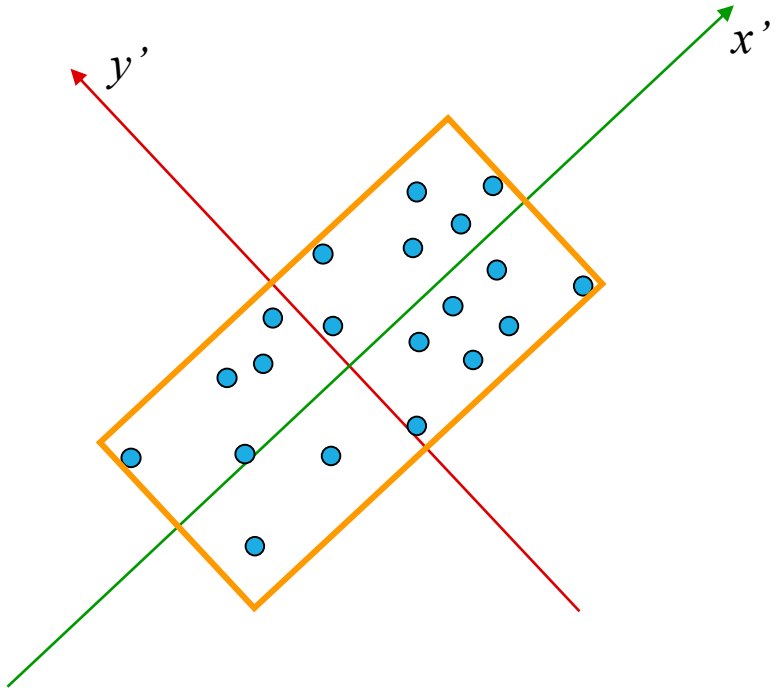
Application: finding tight bounding box

An axis-aligned bounding box: agrees with the axes



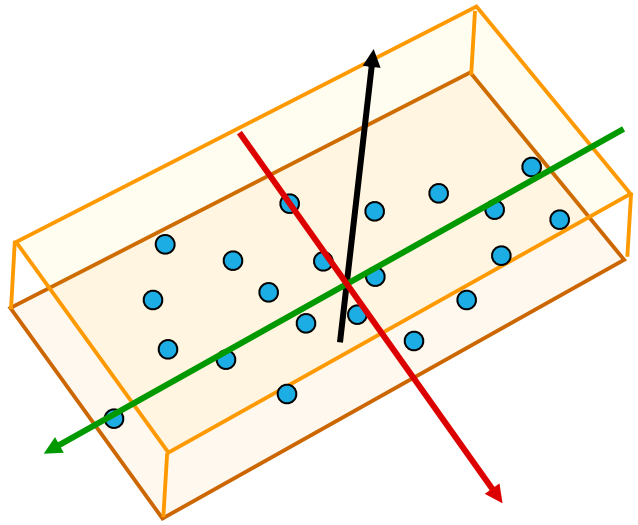
Application: finding tight bounding box

Oriented bounding box: we find better axes!



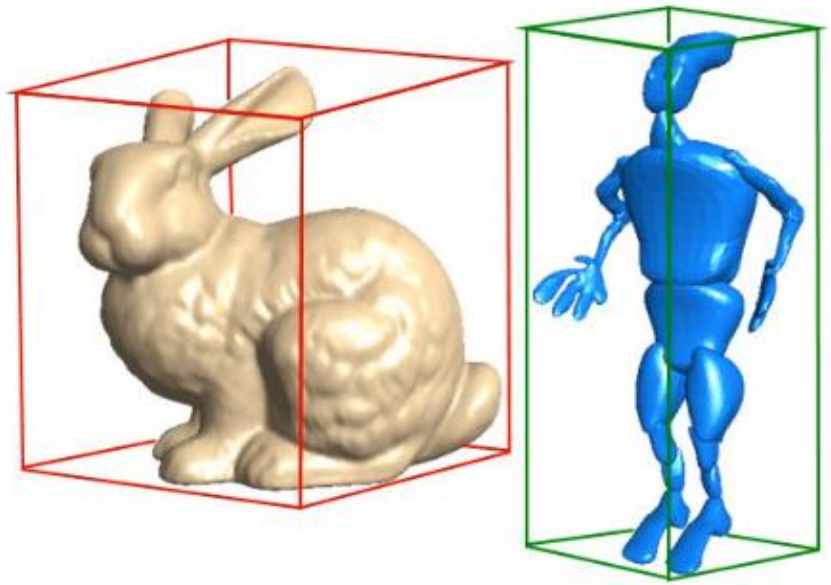
Application: finding tight bounding box

Oriented bounding box: we find better axes!



Use of bounding boxes (bounding volumes)

- Serve as very simple “approximation” of the object
- Fast collision detection, visibility queries
- Whenever we need to know the dimensions (size) of the object



- The models consist of thousands of polygons
- To quickly test that they don’t intersect, the bounding boxes are tested
- Sometimes a hierarchy of bounding boxes are used
- The tighter the bounding box, the less “false alarms” we have
- A collection of bounding boxes, covering different parts of an object, could be used for more accurate collision detection

KD Trees

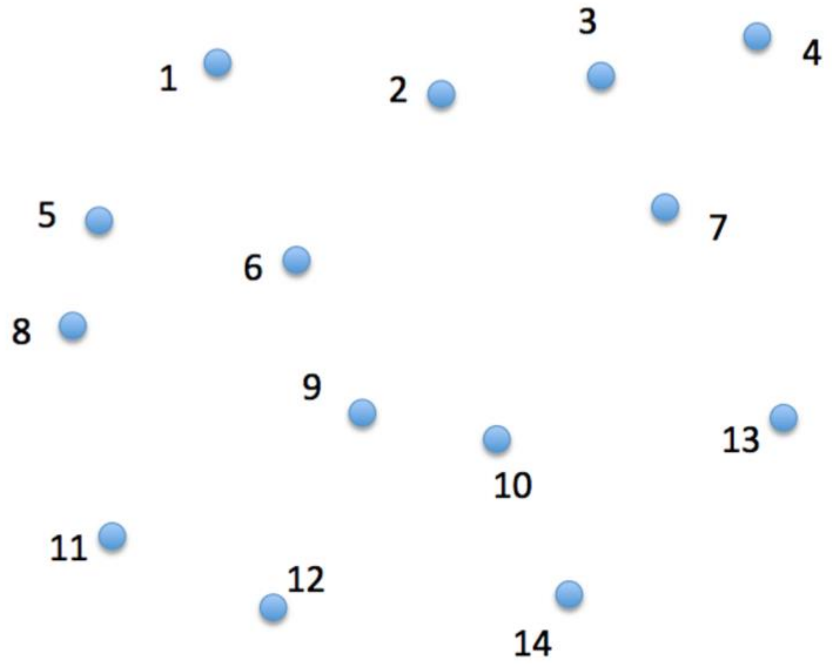
KD-tree (*K*-dimensional tree) is a data structure that allows to organize a point cloud in the form of a binary tree.

The basic idea is to recursively and alternatively project the points onto the *x*, *y*, *z*, *x*, *y*, *z*, etc., axes, to order the points along each axis and to split the set into two halves.

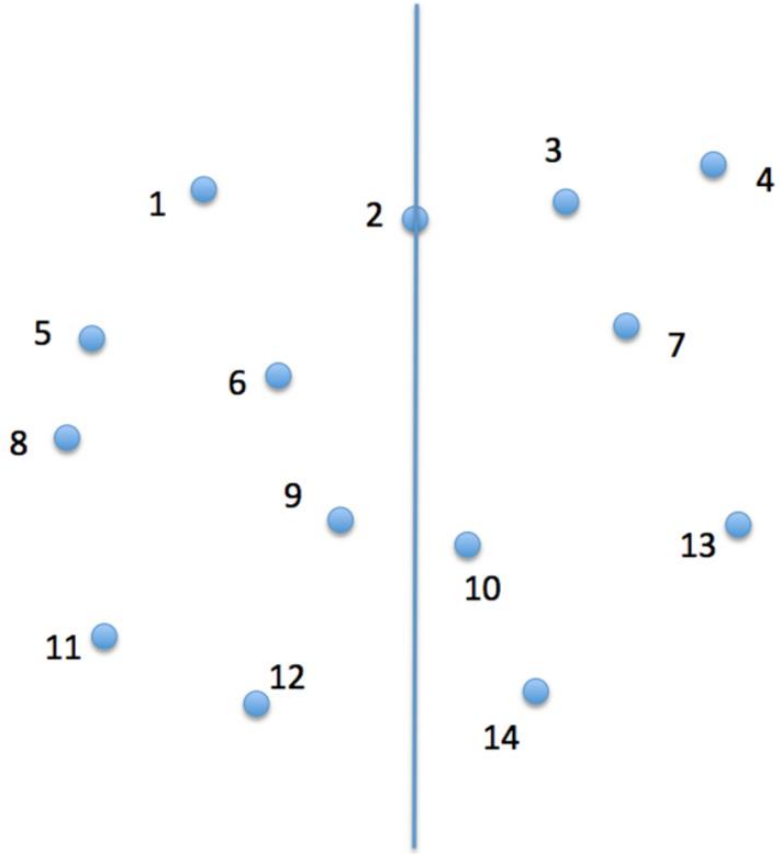
This point-cloud organization facilitates and accelerates the search of nearest neighbors (at the price of kd-tree construction).

A more elaborate method (requiring more pre-processing time) is to search for the principal direction and split the data using a plane orthogonal to this direction, and apply this strategy recursively.

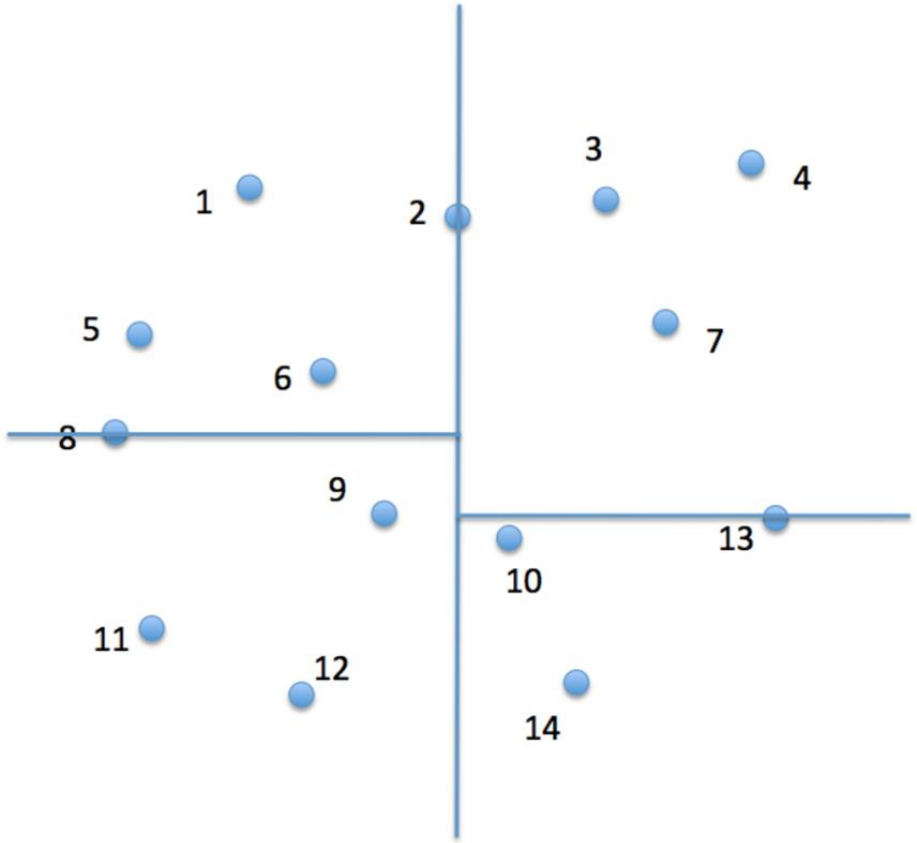
An Example of a 2D-tree (1)



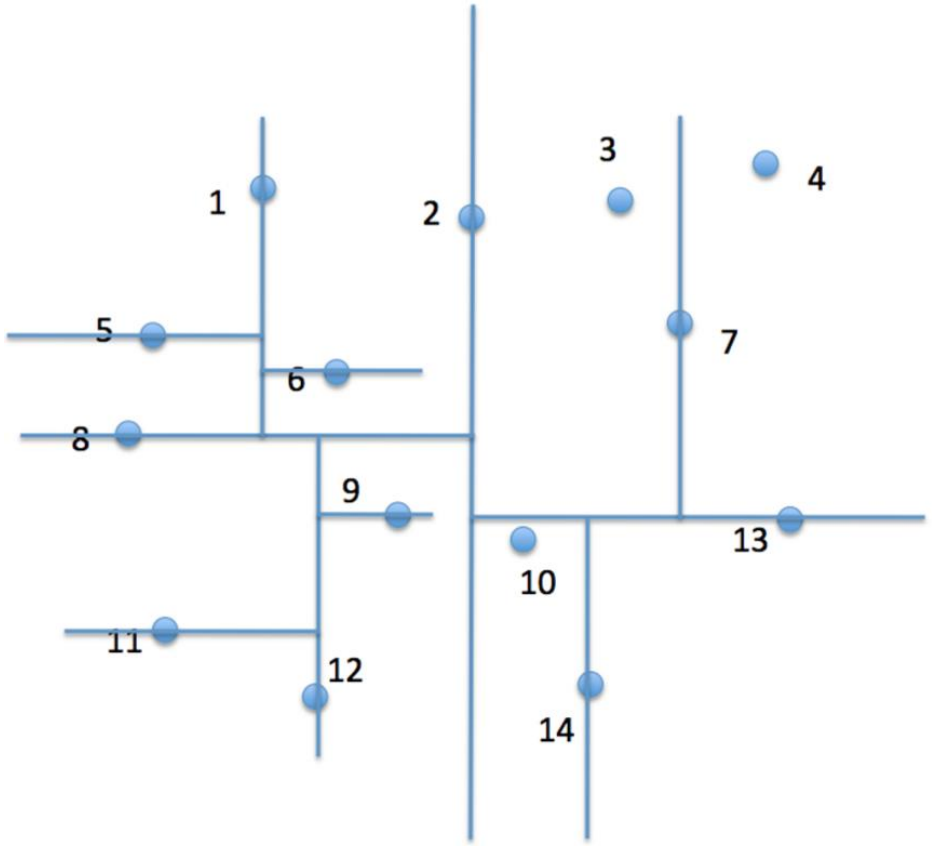
An Example of a 2D-tree (2)



An Example of a 2D-tree (3)



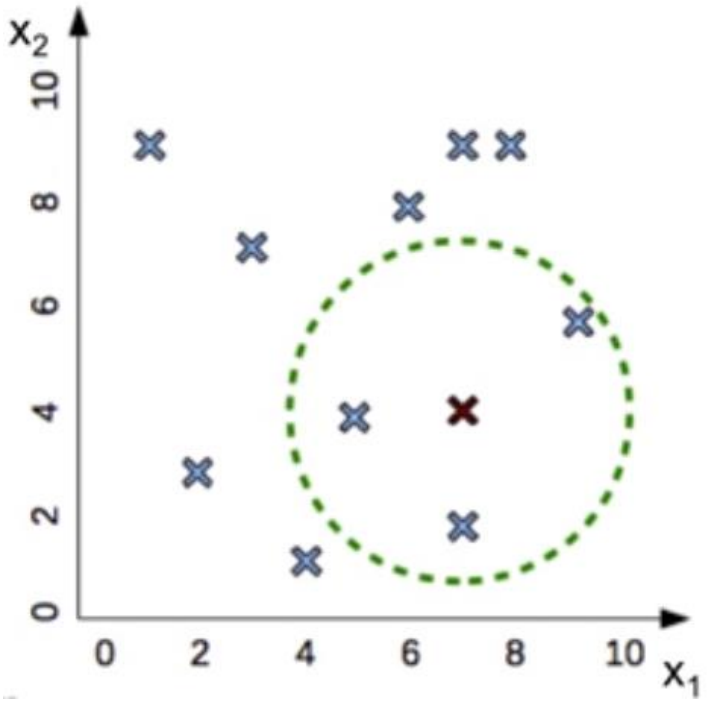
An Example of a 2D-tree (4)



An Example of a 2D-tree

We show how to build a KD tree for the following set of 2D points:

(1,9) (2,3) (4,1) (3,7) (5,4), (6,8) (7,2) (8,8) (7,9) (9,6)

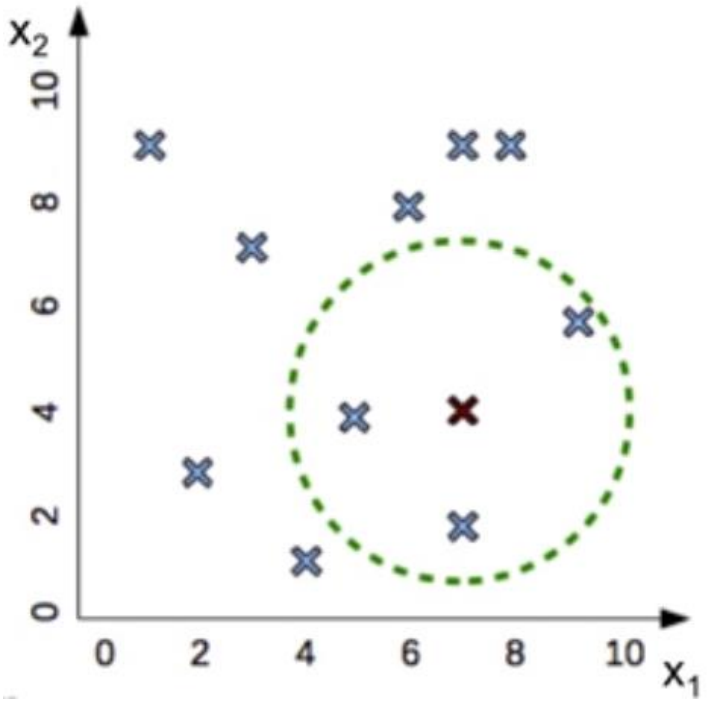


An Example of a 2D-tree

General procedure:

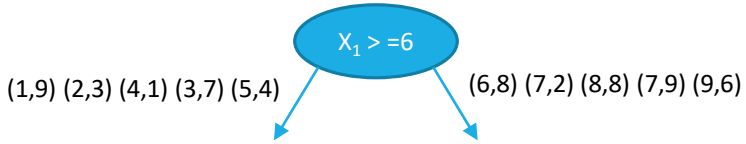
If we have data in n dimensions (n=2 in this example),

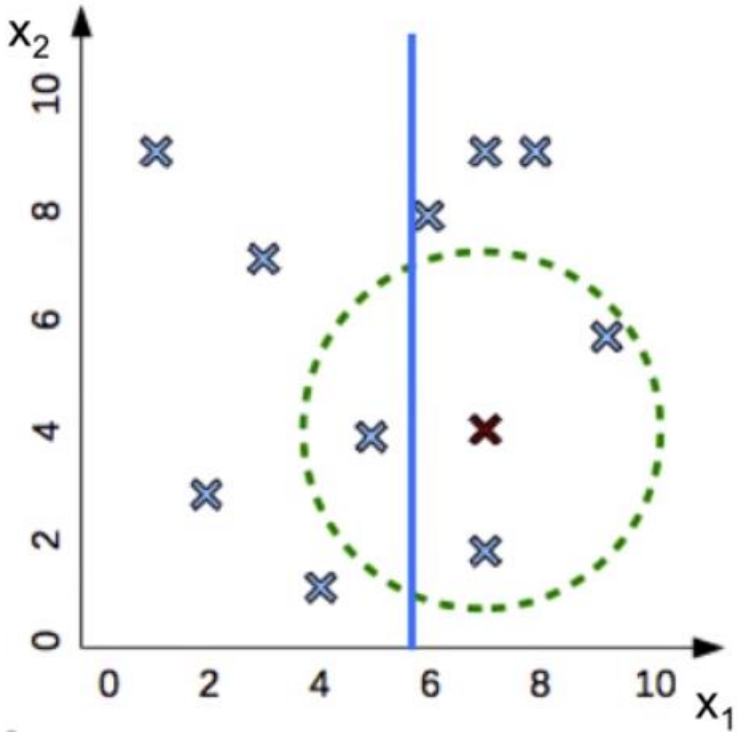
1. pick one of the dimensions (alternating among all dimensions)
2. find median
3. split data based on median
4. repeat Steps 1-3



An Example of a 2D-tree

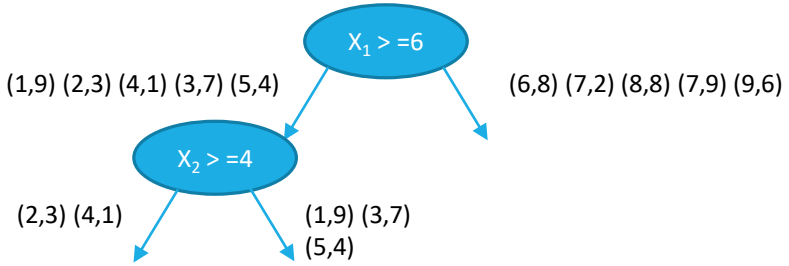
First, we pick the $\mathbf{x_1}$-coordinate to split the data. Median in terms of $\mathbf{x_1}$ is 6.

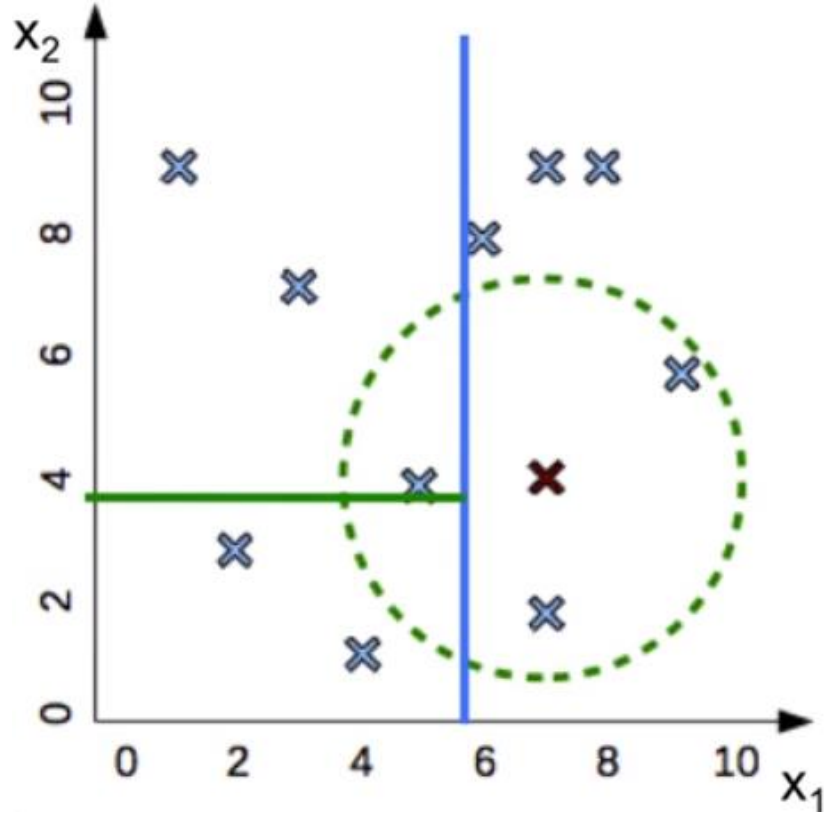




An Example of a 2D-tree

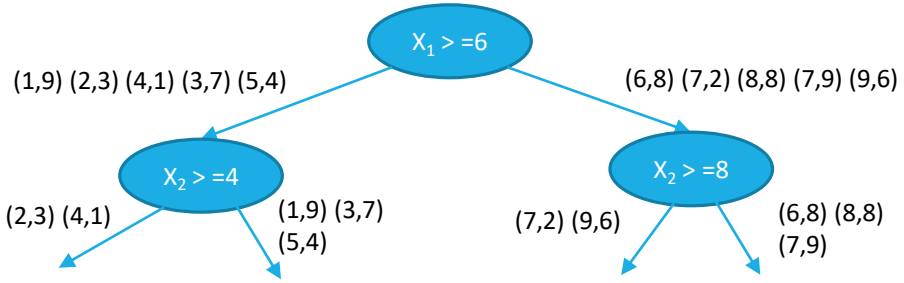
Next, we pick the $\mathbf{x_2}$-coordinate to split the data in each subset.

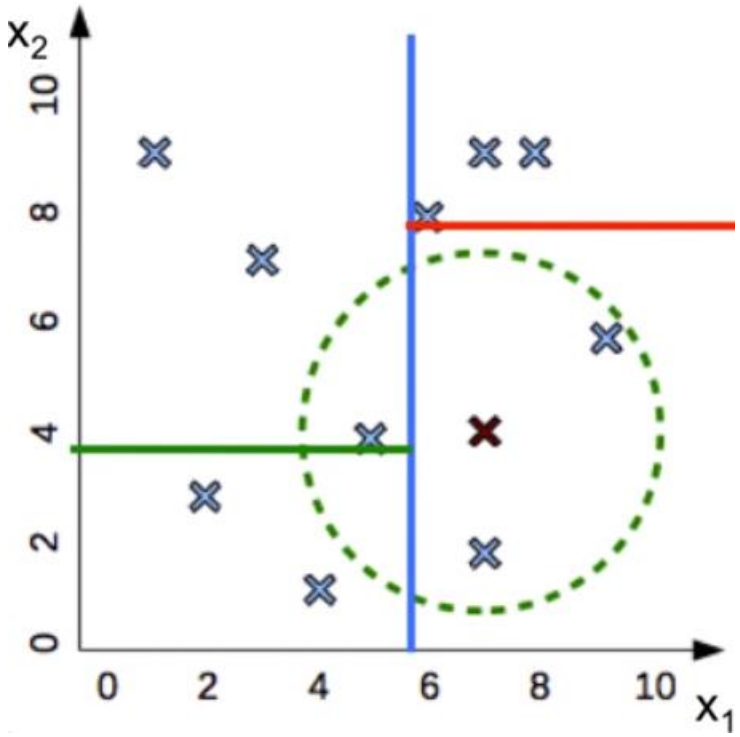




An Example of a 2D-tree

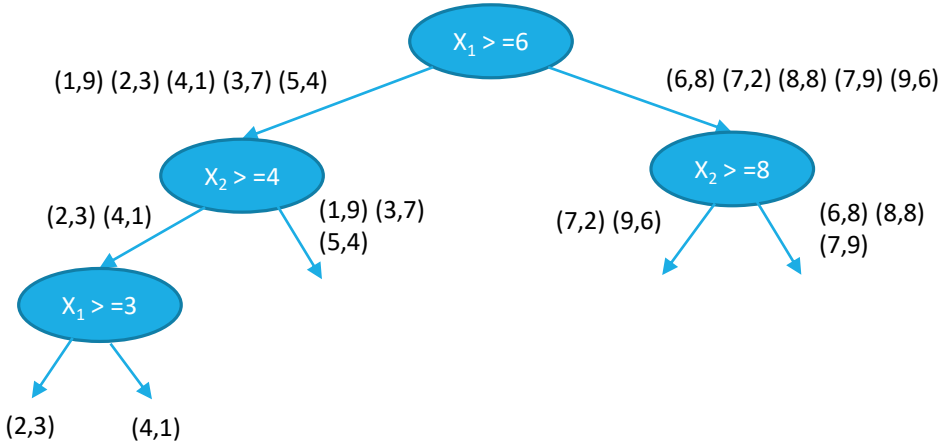
Next, we pick the $\mathbf{x_2}$-coordinate to split the data in each subset.





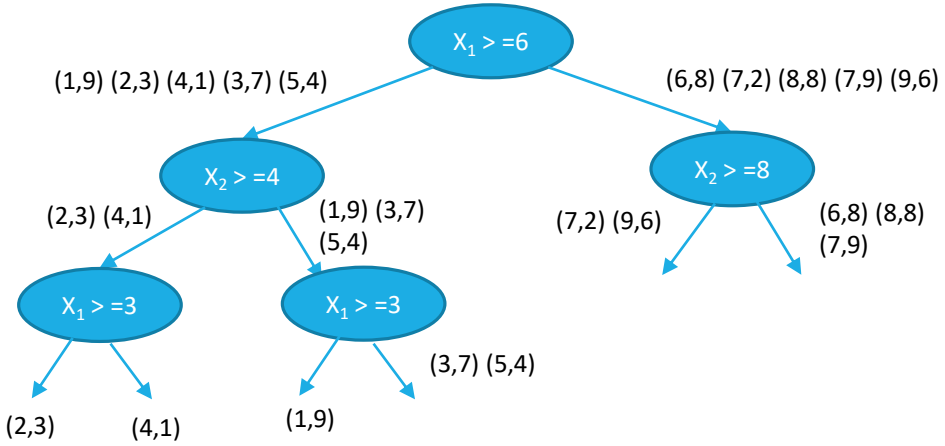
An Example of a 2D-tree

Next, we pick the $\mathbf{x_1}$-coordinate to split the data in each subset (we alternate between x_1 and x_2).



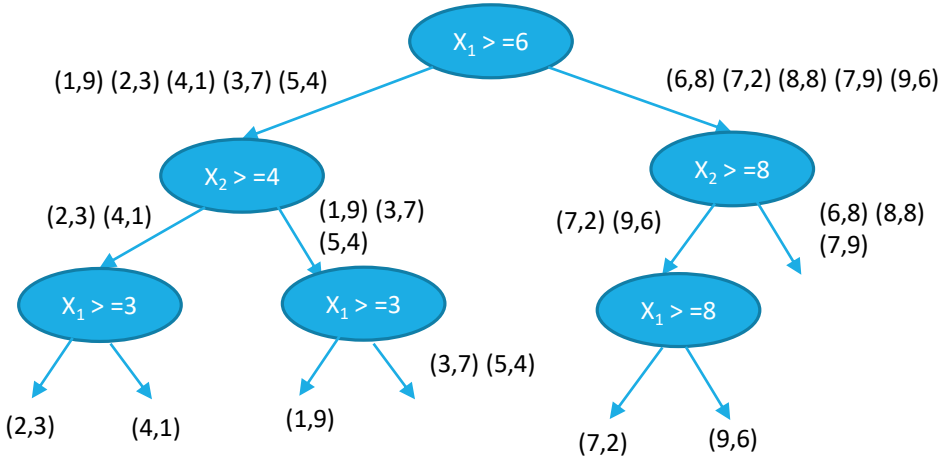
An Example of a 2D-tree

Next, we pick the $\mathbf{x_1}$-coordinate to split the data in each subset (we alternate between x_1 and x_2).



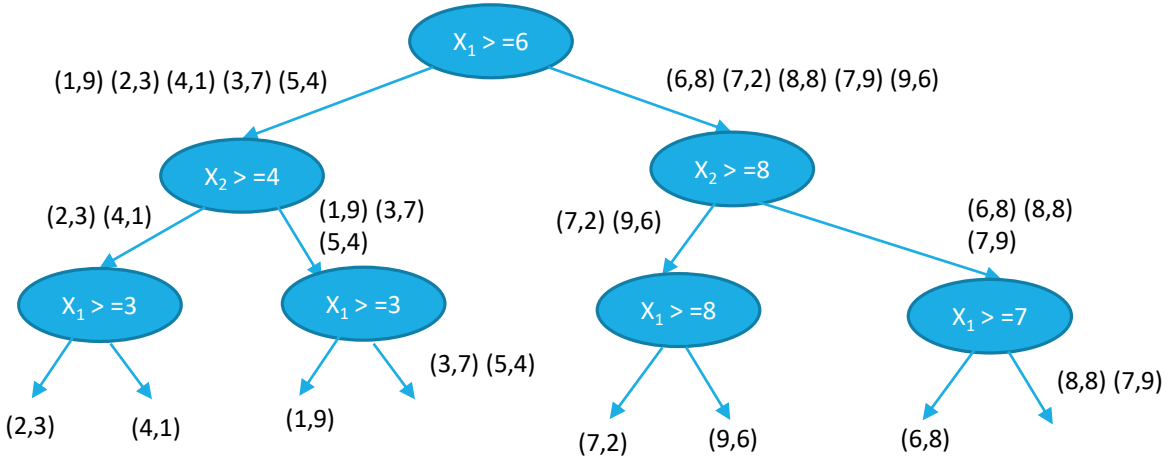
An Example of a 2D-tree

Next, we pick the $\mathbf{x_1}$-coordinate to split the data in each subset (we alternate between x_1 and x_2).



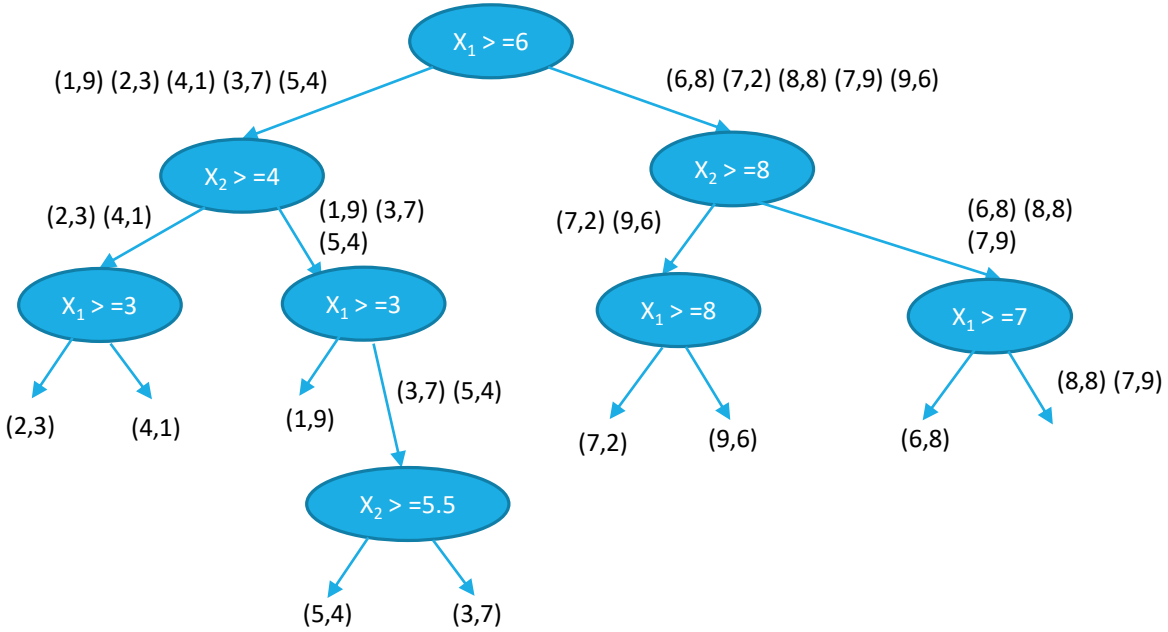
An Example of a 2D-tree

Next, we pick the $\mathbf{x_1}$-coordinate to split the data in each subset (we alternate between x_1 and x_2).



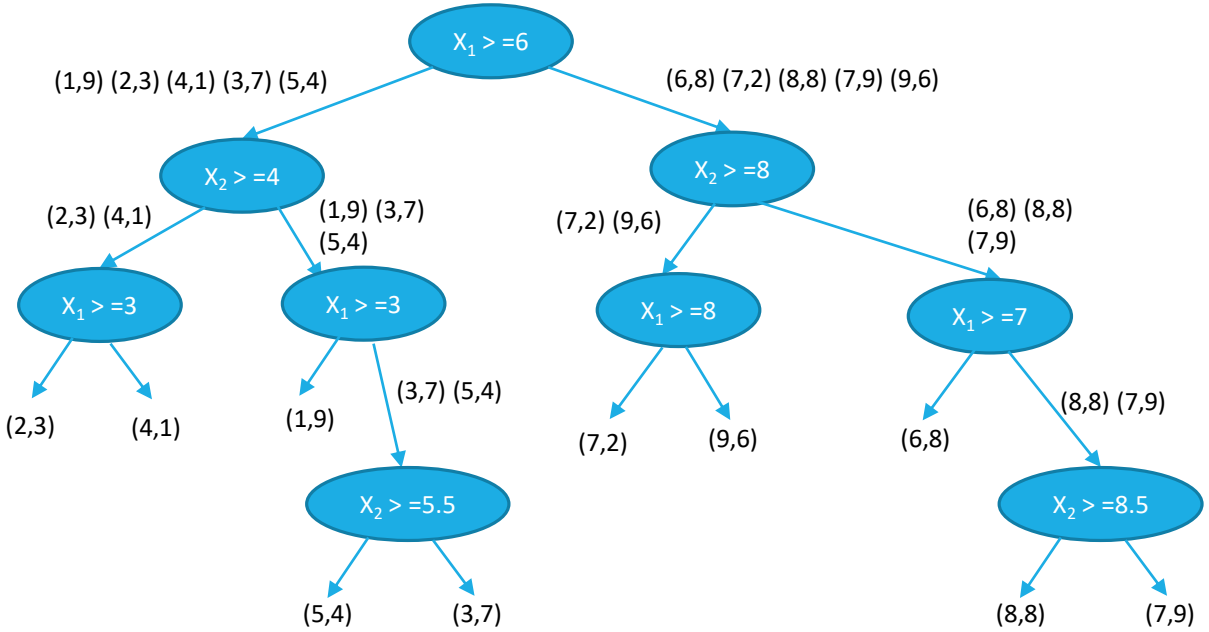
An Example of a 2D-tree

Next, we pick the $\mathbf{x_2}$-coordinate to split the data in each subset (we alternate between x_1 and x_2).



An Example of a 2D-tree

Next, we pick the $\mathbf{x_2}$-coordinate to split the data in each subset (we alternate between x_1 and x_2).



Animated Example of KD Tree

https://www.youtube.com/watch?v=Y4ZgLlDfKDg

https://www.youtube.com/watch?v=Z4dNLvno-EY

Application of KD Tree

- KD trees are useful for Nearest Neighbor search in K-dimension; it is a quick approximation, not 100% accurate.
- We can leave Buckets of Points at the leaves instead of a single point, if we don't keep dividing all the way to the end.
- For 3D Point Clouds we can search for Nearest Neighbors & Create a Mesh from a Point Cloud. (However, the nearest neighbor search is not accurate since we are not using Distance to Split.)
- A better strategy for splitting is to use Principal Components of Points to determine the axis to split. We will not discuss these advanced strategies in this course. But feel free to read up on your own if you find this topic interesting.
- Finding Nearest Neighbors is also useful for Interaction with Point Could data, e.g., to mark or select points and regions through a user interface.

The K-means Algorithm

What is a cluster: a group of points whose inter-point distance are small compared to distances to points outside the cluster.

Cluster centers: $\mu_1,\ldots,\mu_m$.

Goal: find an assignment of points to clusters as well as a set of mean-vectors μ_k.

Notations: For each point x_j there is a *binary indicator variable* $r_{jk} \in \{0,1\}$.

Objective: minimize the following *distortion measure*: (**Minimize the Total of the Sum of Squares of Each Cluster from its Cluster Center --- an NP-hard problem**!)

$$J = \sum_{j=1}^{n} \sum_{k=1}^{m} r_{jk} \|\boldsymbol{x}_j - \boldsymbol{\mu}_k\|^2$$

A K-means Algorithm
(One of the Heuristic Algorithms: Not Optimal!)

Initialization: Choose *m* and initial values for $\mu_1,\ldots,\mu_m$.

First step: Assign the *j*-th point to the closest cluster center:

$$r_{jk} = \begin{cases} 1 & \text{if } k = \arg\min_l \|\boldsymbol{x}_j - \mu_l\|^2 \\ 0 & \text{otherwise} \end{cases}$$

Second Step: Compute the New cluster centers: $\boldsymbol{\mu}_k = \frac{\sum_{j=1}^{n} r_{jk}\boldsymbol{x}_j}{\sum_{j=1}^{n} r_{jk}}$

Convergence: Repeat until no more change in the assignments.

Numerical Example of K-means Algorithm

Consider 5 Points: P1 = (0, 0), P2 = (0.5, 0.5), P3= (2, 2), P4 = (3, 3) and P5 = (4, 4)

Initialization: Choose *m =2* and initial values for Cluster Centers μ_1 *=(0,0) &* μ_2 = (0.5, 0.5)

First step: P1 closer to First Cluster Center, other points closer to 2nd Cluster Center, so:

r_{11} = 1, & r_{22} = r_{32} = r_{42} = r_{52} = 1 other r's are equal to 0

$$r_{jk} = \begin{cases} 1 & \text{if } k = \arg\min_l \|\boldsymbol{x}_j - \mu_l\|^2 \\ 0 & \text{otherwise} \end{cases}$$

New Cluster Centers: μ_1 *=(0,0) &* μ_2 = (0.5+2+3+4, 0.5+2+3+4)/4 = (2.375, 2.375)

Second Step: Compute the New Cluster Centers:

$$\boldsymbol{\mu}_k = \frac{\sum_{j=1}^{n} r_{jk}\boldsymbol{x}_j}{\sum_{j=1}^{n} r_{jk}}$$

Convergence: Repeat until no more change in the assignments.

Numerical Example of K-means Algorithm

SECOND ITERATION:

5 Points: P1 = (0, 0), P2 = (0.5, 0.5), P3= (2, 2), P4 = (3, 3) and P5 = (4, 4)

Current Cluster Centers: μ_1 *=(0,0) &* μ_2 = (2.375, 2.375)

First step: P1 & P2 closer to First Cluster Center, other points closer to 2nd Cluster Center, so:

$r_{11} = r_{21} = 1$, & $r_{32} = r_{42} = r_{52} = 1$ other r's are equal to 0

$$r_{jk} = \begin{cases} 1 & \text{if } k = \arg\min_l \|\boldsymbol{x}_j - \mu_l\|^2 \\ 0 & \text{otherwise} \end{cases}$$

New Cluster Centers: μ_1 *=(0+0.5,0+0.5)/2 = (0.25, 0.25) &* μ_2 = (2+3+4, 2+3+4)/3 = (3, 3)

Second Step: Compute the New Cluster Centers:

$$\boldsymbol{\mu}_k = \frac{\sum_{j=1}^{n} r_{jk}\boldsymbol{x}_j}{\sum_{j=1}^{n} r_{jk}}$$

Convergence: Repeat until no more change in the assignments.

Numerical Example of K-means Algorithm

THIRD ITERATION:

5 Points: P1 = (0, 0), P2 = (0.5, 0.5), P3= (2, 2), P4 = (3, 3) and P5 = (4, 4)

Current Cluster Centers: μ_1 *=(0.25,0.25)* & μ_2 = (3, 3)

First step: P1 & P2 closer to First Cluster Center, other p

$$r_{jk} = \begin{cases} 1 & \text{if } k = \arg\min_l \|\boldsymbol{x}_j - \mu_l\|^2 \\ 0 & \text{otherwise} \end{cases}$$

r_{11} = r_{21} =1, & r_{32} = r_{42} = r_{52} = 1 other r's are equal to 0

New Cluster Centers: μ_1 *=(0+0.5,0+0.5)/2 = (0.25, 0.25)* & μ_2 = (2+3+4, 2+3+4)/3 = (3, 3)

Convergence: Repeat until no more change in the assignments. **NO CHANGE SO CONVERGED!**

Uniqueness of the proposed K-means Algorithm

The Proposed K-means algorithm **DOES NOT PRODUCE UNIQUE RESULTS** it is only an approximation

5 Points: P1 = (0, 0), P2 = (0.5, 0.5), P3= (2, 2), P4 = (3, 3) and P5 = (4, 4)

Suppose we want K=3 Clusters & we choose:

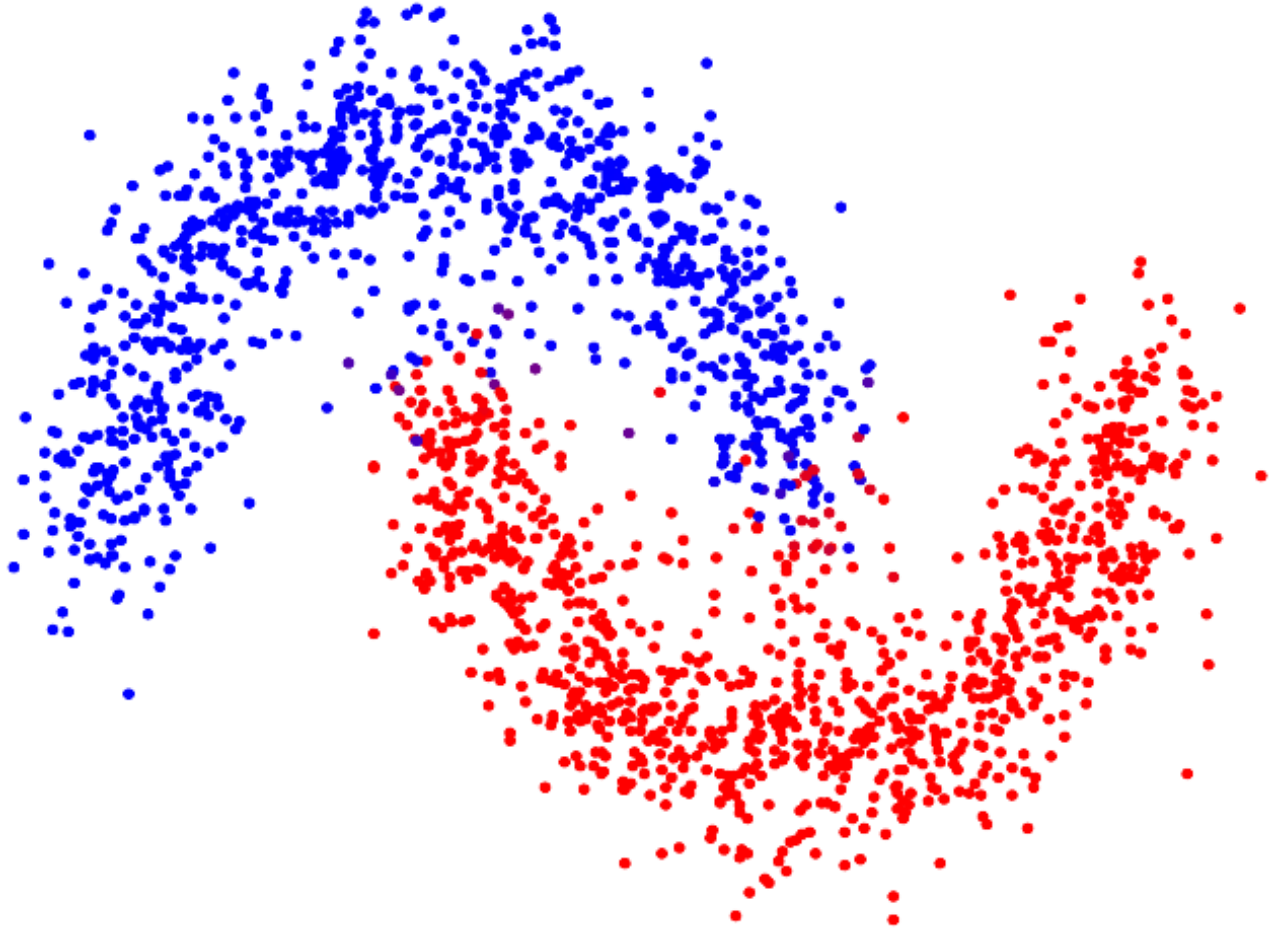
(i) $\mu_1 = (0, 0); \mu_2 = (0.5, 0.5)$ & $\mu_3 = (3, 3)$

(ii) $\mu_1 = (0, 0); \mu_2 = (2, 2)$ & $\mu_3 = (3, 3)$

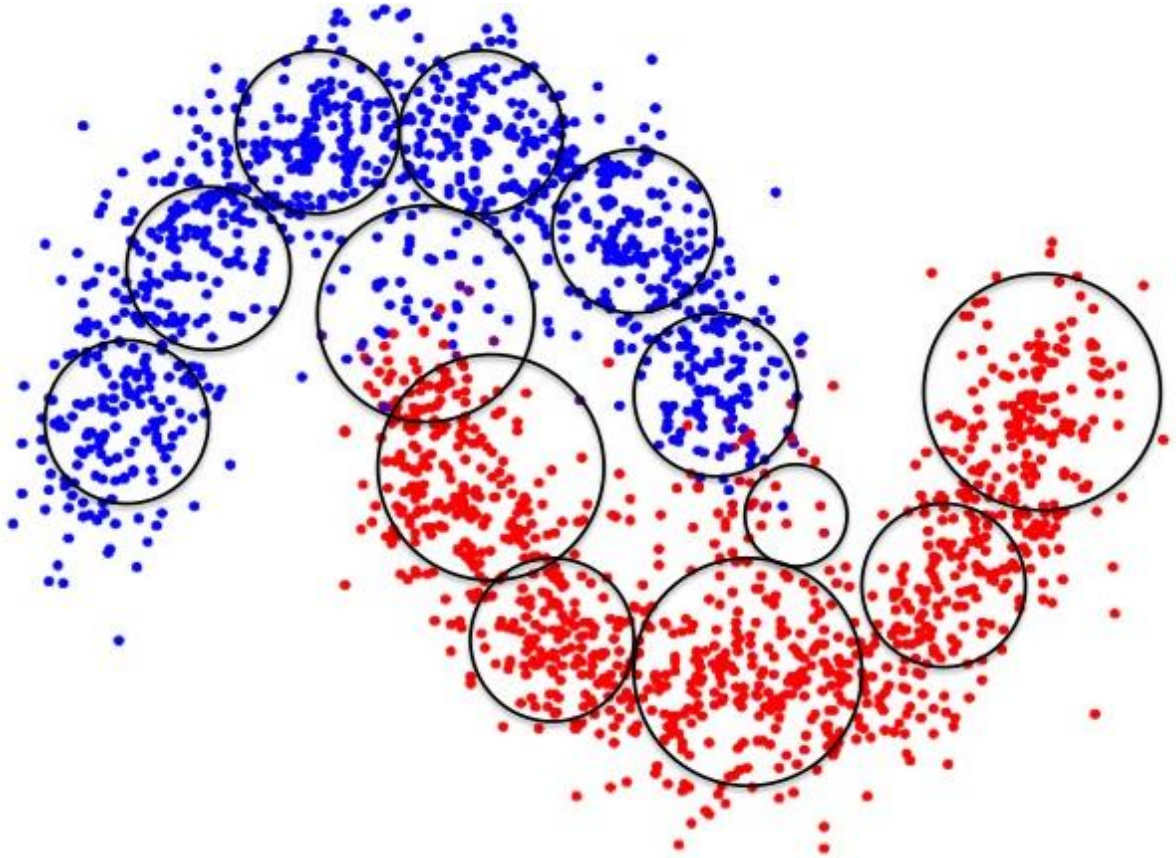
In this case we get different clusters.

The clusters produced by our algorithm does not Minimize the function J on Slide 55. It takes $O(2^n)$ computations to guarantee such an optimal solution right now. So we only get an approximate solution.

How to Represent This Point Cloud?



Spherical Clusters



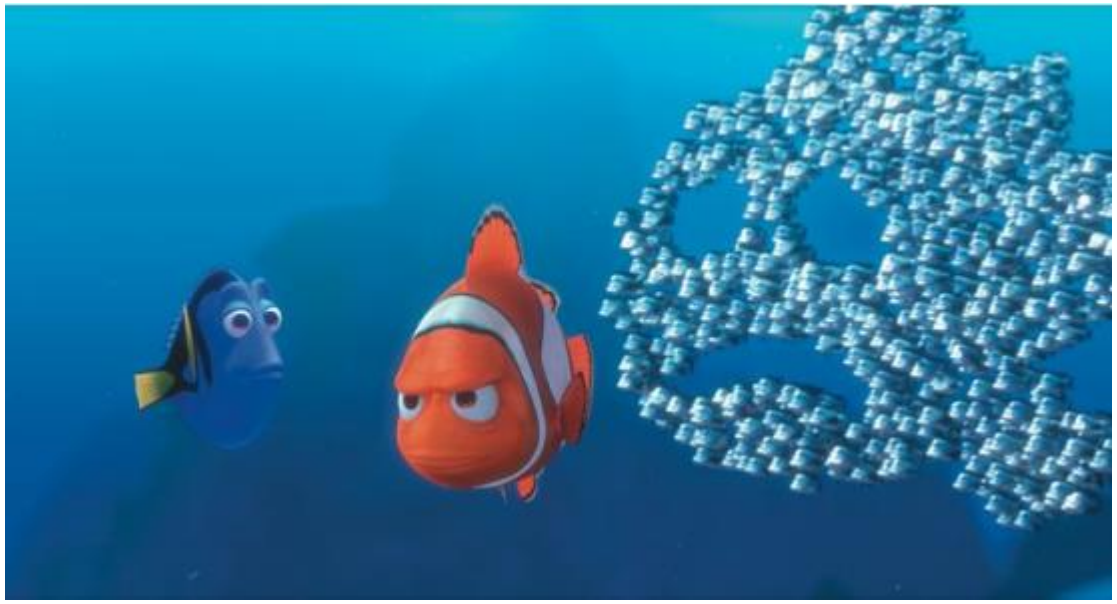
K-means Demo

<< Show the MATLAB K-Means demo by changing the value of K >>

Application of K-Means for Point Clouds

Problem: 3D clouds can be extremely dense

- Large storage space
- Long post-processing times

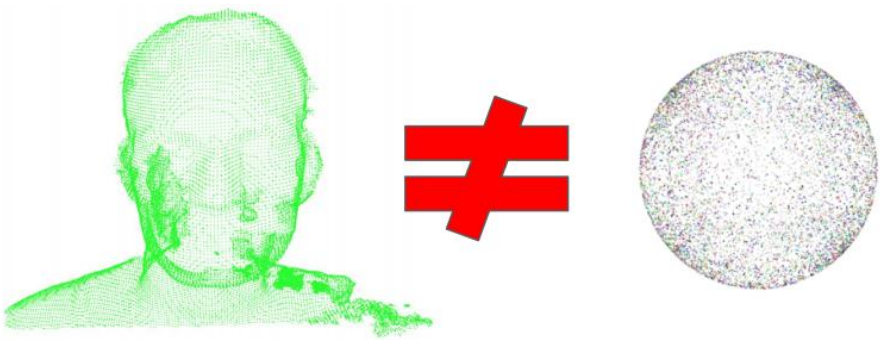


Solution: Remove unnecessary points

- Hard to correctly identify representative points and still keep small features and boundaries

Application of K-Means for Point Clouds

- Clouds have >200,000 points
- Need to preserve features like boundaries



- Simplification Steps:

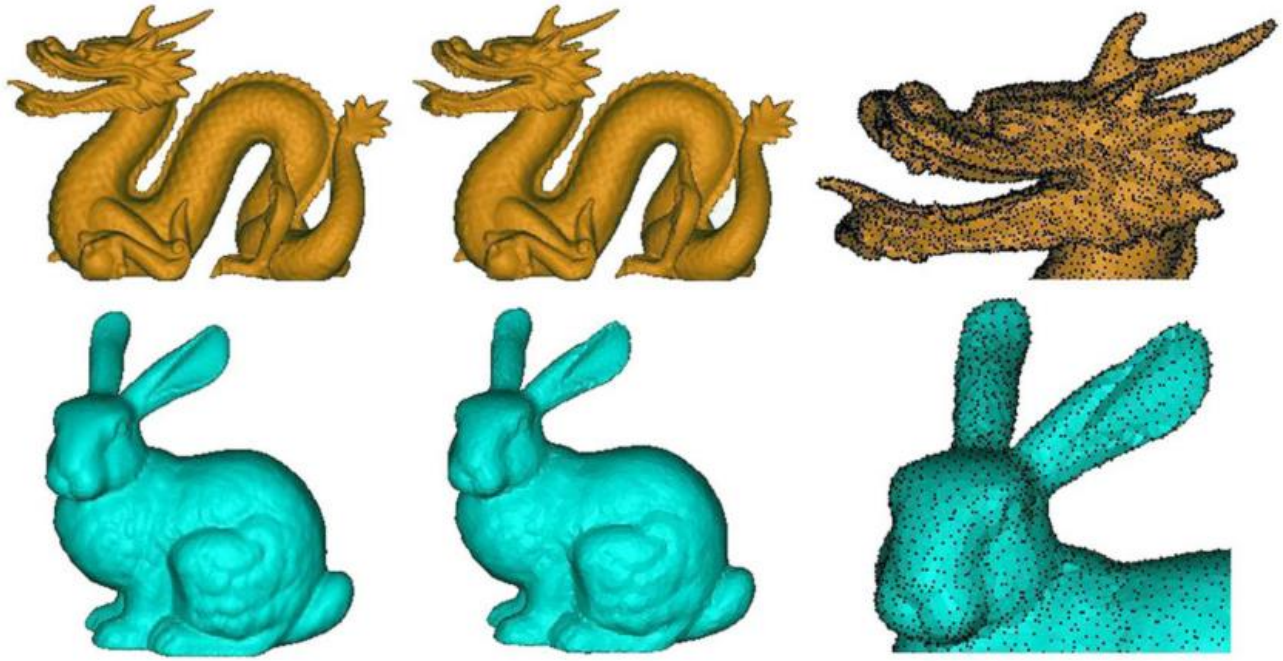
1. Create clusters using k-means clustering algorithm
2. Check original boundary integrity
3. Partition clusters into sub-clusters recursively
4. Refine clusters to balance density distribution of points

Application of K-Means for Point Clouds

Results in impressive compression:

Dragon: 7.6% (435,545 to 32,925)

Bunny: 12.8% (34,834 to 6,500)



Gaussian Mixture Model

One important characteristic of K-means is that it is a hard clustering method, which means that it will associate each point to one and only one cluster. A limitation to this approach is that there is no uncertainty measure or probability that tells us how much a data point is associated with a specific cluster. Gaussian Mixture Model, or simply GMM, can give a data point's probabilities of belonging to different clusters.

A Gaussian mixture model is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters.

One can think of mixture models as generalizing k-means clustering to incorporate information about the covariance structure of the data as well as the centers of the latent Gaussians.

https://video.search.yahoo.com/search/video?fr=opensearch&p=gaussian+mixture+models+for+clustering#id=4&vid=3002a8dd3bf5e9deca7b92be7031f364&action=click

Gaussian Mixture Model

The Gaussian (also known as a multivariate normal distribution) of a random D dimensional point p can be specified using μ and $\sum$, which are its expected value (mean), and covariance matrix respectively. Therefore the likelihood of a single point x with respects to the k[th] Gaussian is given by

$$\mathcal{N}_k(x) = \frac{1}{(2\pi)^{\frac{D}{2}} |\sum|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-\mu_k)' \sum_k^{-1}(x-\mu_k)}$$

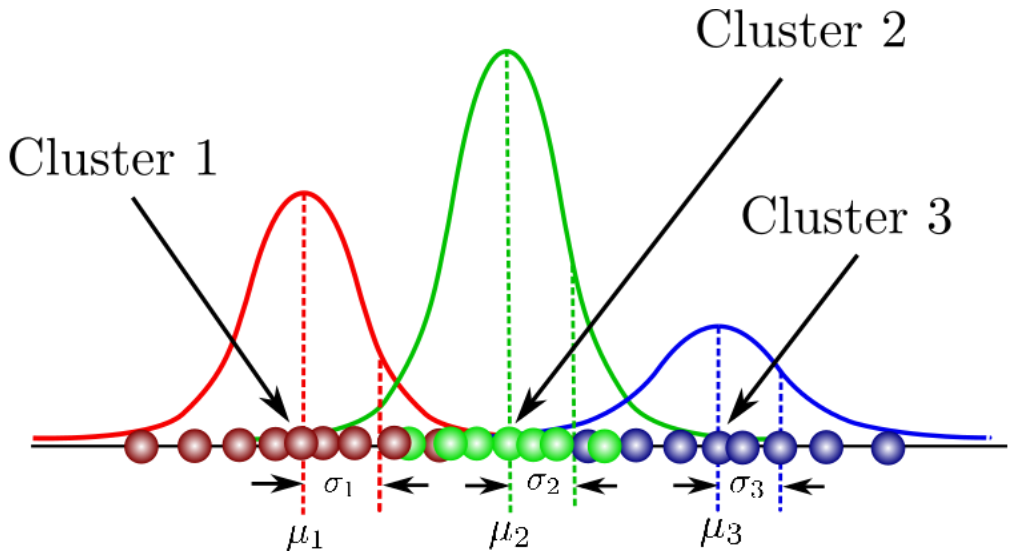
and the likelihood of x associated with the GMM

$$\mathcal{N}_\lambda(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}_k(x)$$

π_k are the mixture weights (informally representing the number of points/ the amount of influence of each Gaussian).

Gaussian Mixture Model

1D condition:



$$\sum_{k=1}^{K} \mathcal{N}_k(p) = 1$$

Gaussian Mixture Model

Let's say that you have some D dimensional data points.

In addition, let us assume that each one of these data points came from one of K different sources. Each source is a Gaussian in a D dimensional space.

The challenge is to find the Gaussians parameters that will maximize their expectation. This is usually done using the Expectation Maximization (EM) algorithm.

We need to find $\lambda = (\pi_k, \mu_k, \sum)$ for the Gaussian distributions that best fits the data.

Some Notation

Given a data point x_n, the probability it came from Gaussian k:

$$p = (z_{nk} = 1|\boldsymbol{x}_n)$$

z is a latent variable that takes only two possible values. It is one when x came from Gaussian k, and zero otherwise.

So we have

$$\pi_k = p(z_k = 1)$$

Which means that the overall probability of observing a point that comes from Gaussian *k* is equivalent to the mixing coefficient for that Gaussian.

Some Notation

Let z be the set of all possible latent variables *z*, hence:

$$\boldsymbol{z} = \{z_1, z_2 \dots z_k\}$$

Each *z* occurs independently of others and that they can only take the value of one when *k* is equal to the cluster the point comes from. Therefore:

$$p(z) = p(z_1 = 1)^{z_1} p(z_2 = 1)^{z_2} \dots p(z_k = 1)^{z_k} = \prod_{k=1}^{K} \pi_k^{z_k}$$

The probability of given data coming from Guassian k:

$$p(\boldsymbol{x}_n|z) = \prod_{k=1}^{K} \mathcal{N}(\boldsymbol{x}_n|\mu_k, \sum_k)^{z_k}$$

Then according to the Bayes rule:

$$p(\boldsymbol{x}_n, z) = p(\boldsymbol{x}_n|z)p(z)$$

Some Notation

Marginalization:

$$p(\boldsymbol{x}_n) = \sum_{k=1}^{K} p(\boldsymbol{x}_n|z)p(z) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}_n|\mu_k, \sum_k)$$

To determine the optimal values for these parameters we need to determine the maximum likelihood of the model. We can find the likelihood as the joint probability of all observations x_n, defined by:

$$p(X) = \prod_{n=1}^{N} p(\boldsymbol{x}_n) = \prod_{n=1}^{N} \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}_n|\mu_k, \sum_k)$$

$$\ln p(\boldsymbol{X}) = \sum_{n=1}^{N} \ln \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x_n}|\mu_k, \sum_k)$$

From Bayes rule:

$$p(z_k = 1|\boldsymbol{x_n}) = \frac{p(\boldsymbol{x}_n|z_k = 1)p(z_k=1)}{\sum_{j=1}^{K} p\left(\boldsymbol{x}_n\middle|z_j = 1\right)p(z_j=1)}$$

Some Notation

So we have:

$$p(z_k = 1|\boldsymbol{x}_n) = \frac{\pi_k \mathcal{N}(\boldsymbol{x}_n|\mu_k,\sum_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x}_n|\mu_j,\sum_j)} = \gamma(z_{nk}) \text{ (1)}$$

Expectation Maximization

Step 1: Initialise λ accordingly. For instance, we can use the results obtained by a previous K-Means run as a good starting point for our algorithm.

Step 2: Evaluate

$$p(Z|\boldsymbol{X}, \boldsymbol{\lambda})$$

We have already found an expression for this. If we take the expectation of z_{nk}, then we get:

$$p(z_{nk}|\boldsymbol{X}, \lambda) = \mathbb{E}[z_{nk}] = \sum_{j=1}^{K} z_{nj}\gamma(z_{nj}) = \gamma(z_{nk})$$

This is exactly what we got in equation (1).

Expectation Maximization

Step 3: Find the revised parameters λ^*using:

$$\lambda^* = \text{argmax}_\lambda \mathcal{Q}(\lambda^*, \lambda)$$

Where

$$\mathcal{Q}(\lambda^*, \lambda) = \mathbb{E}[\ln p(X, Z|\lambda^*)] = \sum_Z p(\boldsymbol{Z}|X, \lambda) \ln p(\boldsymbol{X}, \boldsymbol{Z}|\lambda^*)$$

$$p(\boldsymbol{X}, \boldsymbol{Z}|\lambda^*) = \prod_{n=1}^{N} \prod_{k=1}^{K} \pi^{z_{nk}} \mathcal{N}(\boldsymbol{x}_n|\mu_k, \sum_k)^{z_{nk}}$$

Now we can determine the optimal parameters.

$$\mathcal{Q}(\lambda^*, \lambda) = \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma(z_{nk})[\ln \pi_k + \ln \mathcal{N}(\boldsymbol{x}_n|\mu_k, \sum_k)] - \lambda(\sum_{k=1}^{K} \pi_k - 1)$$

We can determine the parameters by using maximum likelihood. Take the derivative of Q with respect to π and let it equal 0.

Expectation Maximization

$$\frac{\partial Q(\lambda^*,\lambda)}{\partial \pi_k} = \sum_{n=1}^{N} \frac{\gamma(z_{nk})}{\pi_k} - \lambda = 0$$

Then by rearranging the terms and applying a summation over k to both sides of the equation, we obtain:

$$\sum_{n=1}^{N} \gamma(z_{nk}) = \pi_k \lambda \Longrightarrow \sum_{k=1}^{K} \sum_{n=1}^{N} \gamma(z_{nk}) = \sum_{k=1}^{K} \pi_k \lambda$$

So:

$$\pi_k = \sum_{n=1}^{N} \frac{\gamma(z_{nk})}{N}$$

And:

$$\mu_k^* = \frac{\sum_{n=1}^{N} \gamma(z_{nk}) \mathbf{x_n}}{\sum_{n=1}^{N} \gamma(z_{nk})}, \sum_k^* = \sum_{n=1}^{N} \gamma(z_{nk}) \, (\boldsymbol{x_n} - \mu_k)(\boldsymbol{x_n} - \mu_k)^T$$

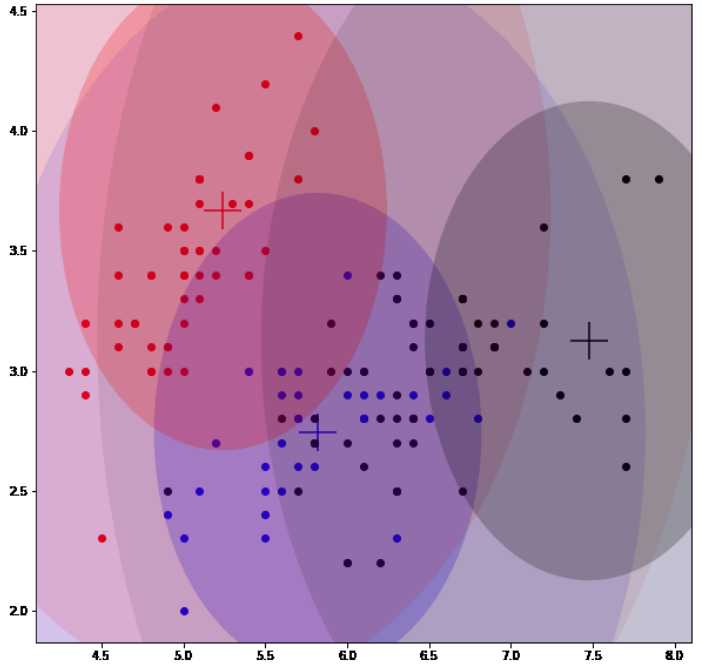
Expectation Maximization

Those revised values will be used in the next EM iteration to determine γ.

Repeat step 2 and 3 until convergence.

A visualization of the process:

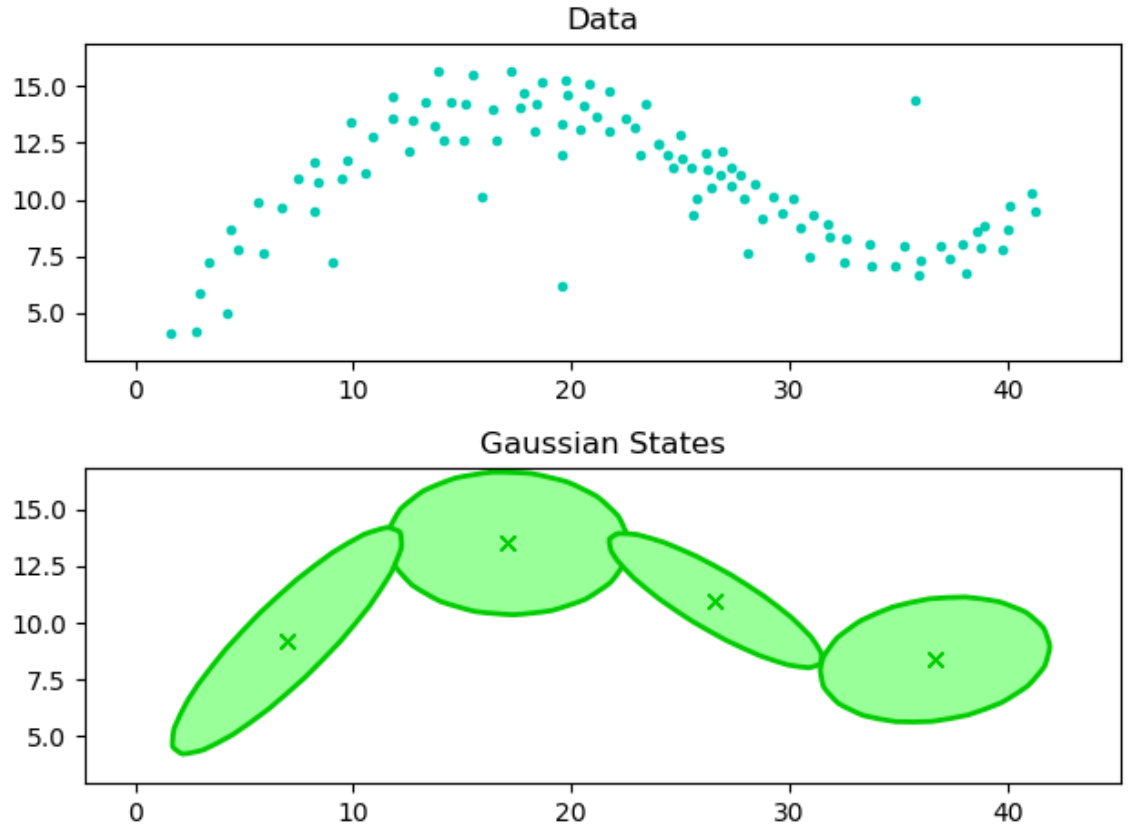
Initialized by K means to 3 clusters



GMM Example

Here is an example of clustering using GMM.

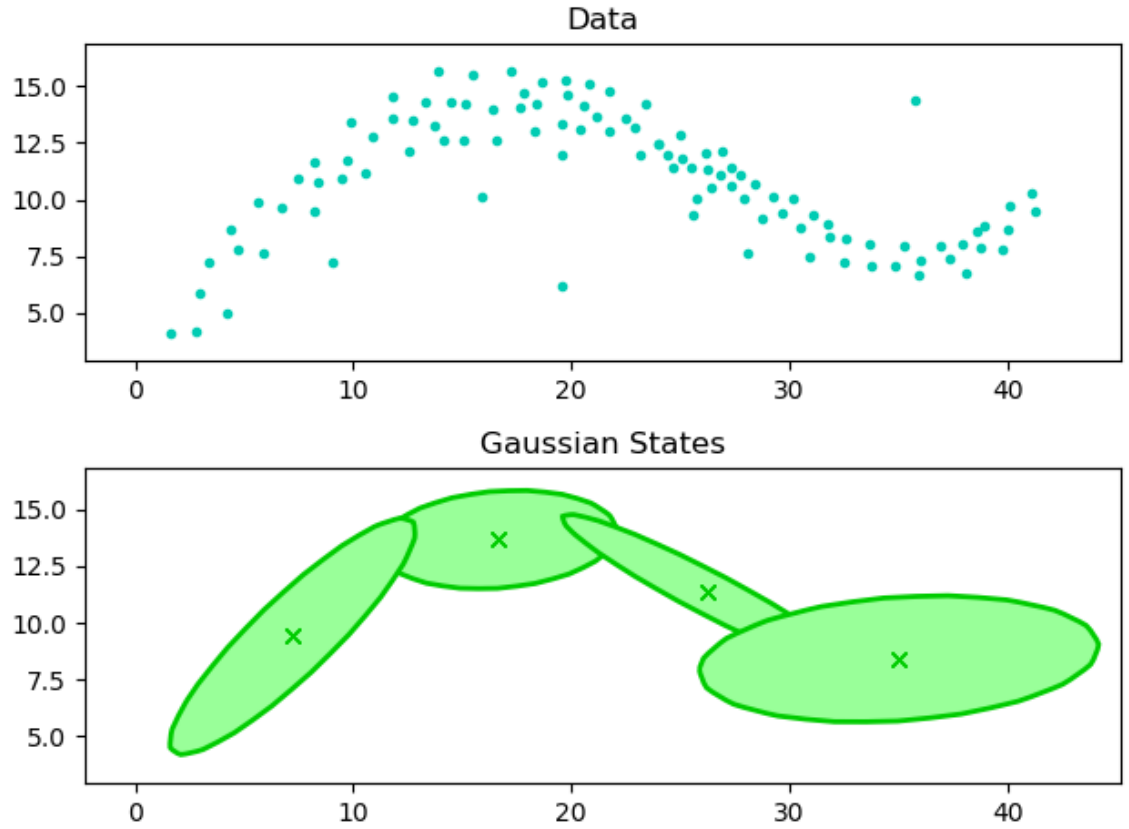
Initialized using K-means



GMM Example

Here is an example of clustering using GMM.

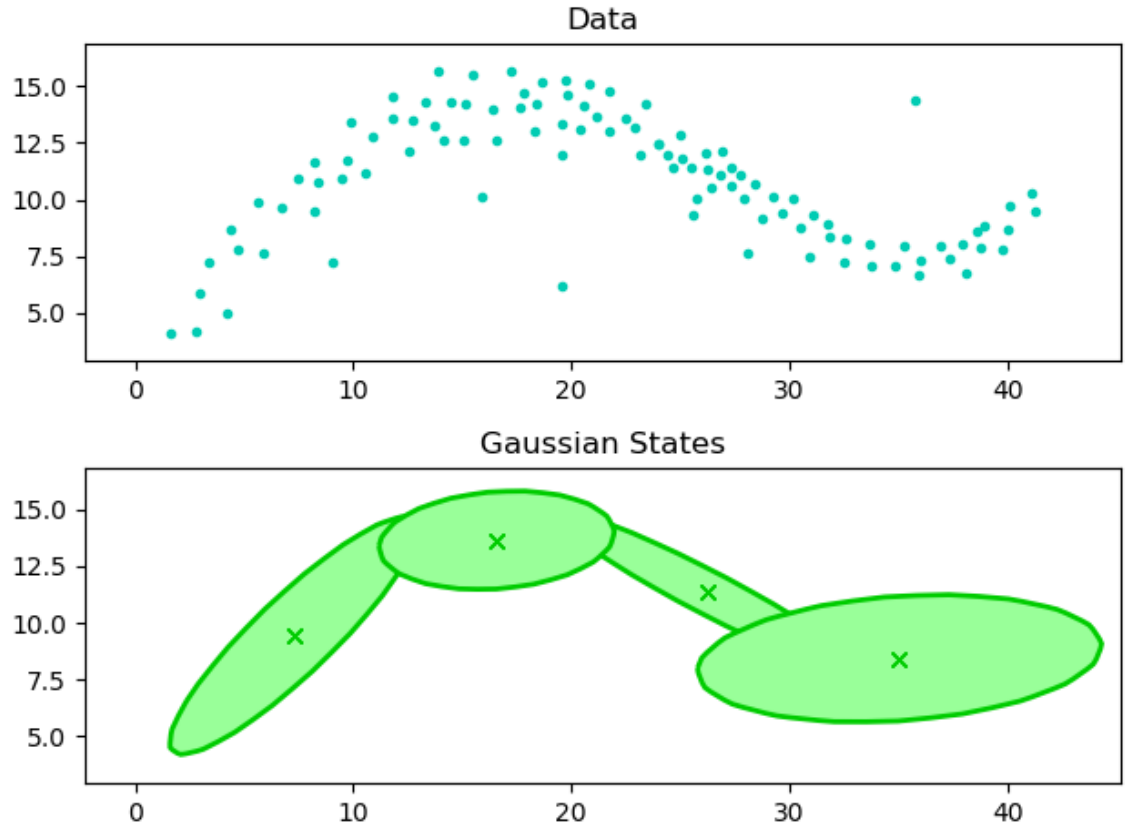
After 10 iterations



GMM Example

Here is an example of clustering using GMM.

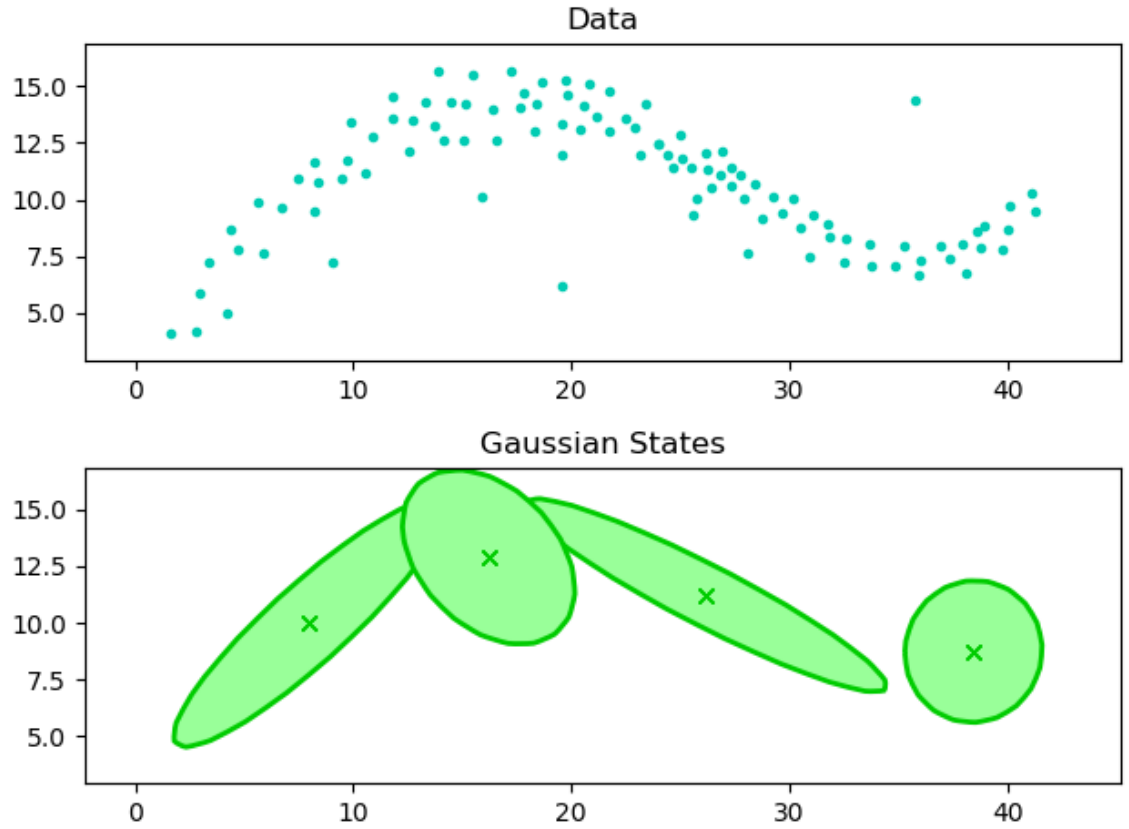
After 20 iterations



GMM Example

Here is an example of clustering using GMM.

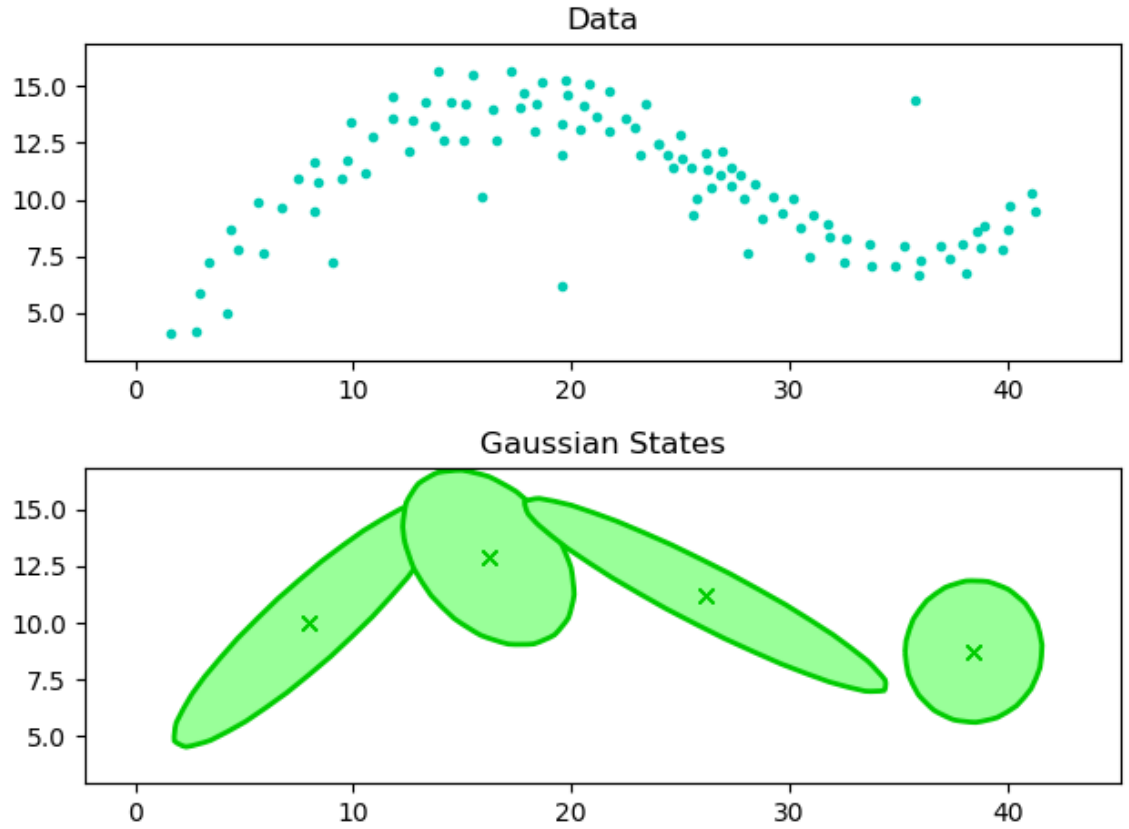
After 30 iterations



GMM Example

Here is an example of clustering using GMM.

After 40 iterations,
Final result:



Laser Range Scanning

Laser scanners capture 3D point and color samples

- Connectivity is implicit

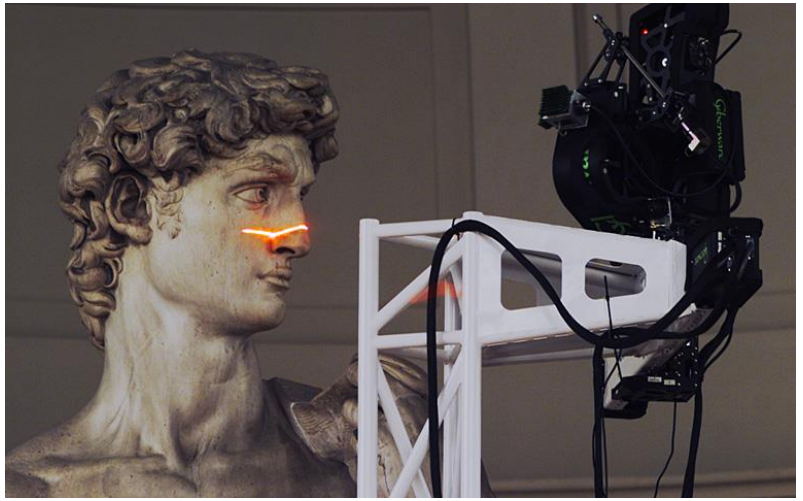
They can scan at resolutions of 0.25mm or better

They produce huge data sets of point samples (multi GB)

- Multiple sets of samples for any one object

What do you have to do to make it work with existing systems?

What are some problems?



Digitizing David

Point-Based Rendering (Splatting)

Instead of drawing triangles, just draw lots of dots (or small circles, or something)

What are the advantages of this?

What problems do you need to solve?

- Think about all the operations performed in a standard graphics pipeline, and how they might cause problems with points
- There are some unique problems

(a)
Points

(b)
Polygons – same number of primitives as (a)
Same rendering time as (a)

(c)
Polygons – same number of vertices as (a)
Twice the rendering time of (a)

QSplat

(Rusinkiewicz and Levoy SIGGRAPH 2000)

Primary goal is interactive rendering of very large point-data sets

Built for the Digital Michelangelo Project

Splat Shape

Several options

- Square (OpenGL “point”)
- Circle (triangle fan or texture mapped square)
- Gaussian (have to do two-pass)

Can squash splats depending on viewing angle

- Sometimes causes holes at silhouettes, can be fixed by bounding squash factor

Splat Shape

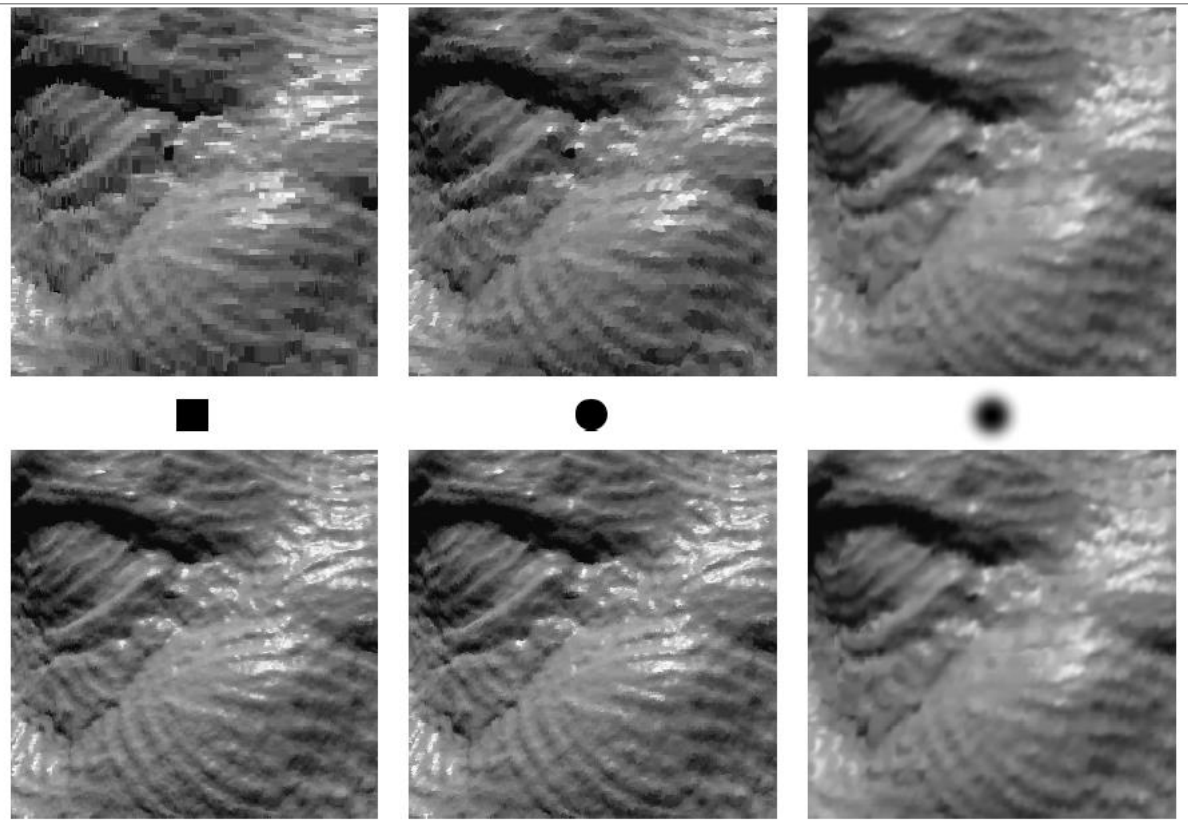


Figure 3: Choices for splat shape. We show a scene rendered using squares, circles, and Gaussians as splat kernels. In the top row, each image uses the same recursion threshold of 20 pixels. Relative to squares, circles take roughly twice as long to render, and Gaussians take approximately four times as long. The Gaussians, however, exhibit significantly less aliasing. In the bottom row, the threshold for each image is adjusted to produce the same rendering time in each case. According to this criterion, the square kernels appear to offer the highest quality.

Rendering Large Data Sets

Methodologies for dealing with this problem

1) Visibility Culling – includes frustum culling, back-face culling, occlusion culling

2) LOD Control – discrete or fine-grained control

3) Geometric Compression – saves on storage costs, but must be decoded to render

4) Point Rendering – use a simpler primitive, the point, instead of triangles

Many algorithms use some of these techniques; QSplat uses all of them

Conclusions

QSplat accomplishes its goal of interactive rendering of very large data sets

QSplat's performance both in preprocessing and rendering is competitive with the fastest progressive display algorithms and mesh simplification algorithms

Geometric compression achieved by QSplat is close to that of current geometric compression techniques

QSplat can be implemented independent of 3D graphics hardware

Software available on graphics.stanford.edu/software/qsplat/

Example Videos

3D Point Cloud with Qsplat:

https://www.youtube.com/watch?v=uYHq5QzX-U4

Point Based Rendering for Games with Qsplat:

https://www.youtube.com/watch?v=VgnNgBwlz6c

MATLAB

Reading and visualizing point cloud in MATLAB (short tutorial):

https://www.youtube.com/watch?v=njIB2CeBV84

Recent Work in MRC Lab, UAlberta

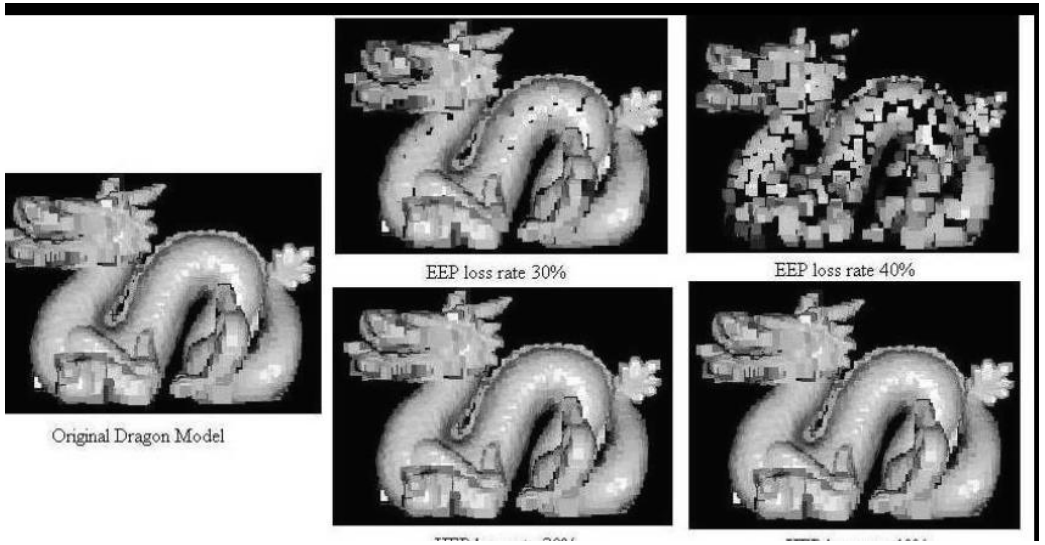
Stereo Point Clouds for 3D Editing & Demarcation of Features on 3D Surfaces

Dynamic Point Cloud Animation

Robust Point Cloud Transmission

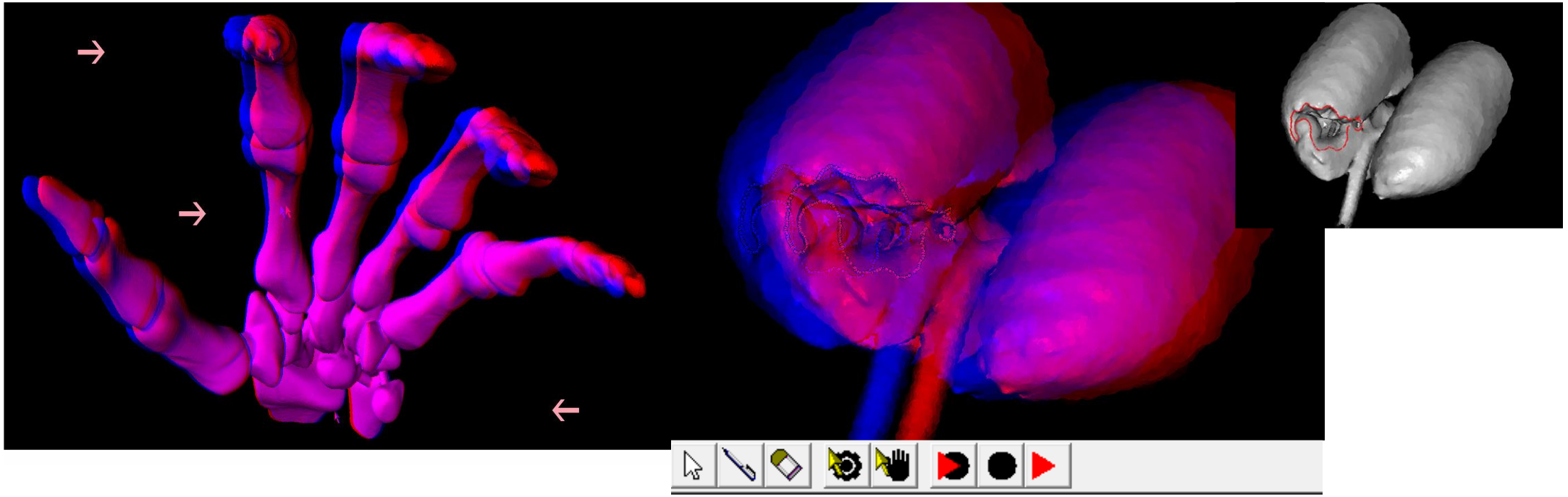
Robust Point Cloud Transmission

We use Forward Error Correction (FEC) based on Reed-Solomon (RS) Codes



Stereo and Multi-view Point Clouds

In Stereo, Closer Objects have Higher Disparity between Left and Right Views & Farther Objects have Lower Disparity



Dynamic Point Cloud Transmission

Created New Data Structures for Efficient Storage and Retrieval of Moving Point Cloud Models