Visual Object Recognition

state-of-the-art techniques and performance evaluation

Lecture 6: Tree Search and Hashing

- High dimensional spaces
- kD-Trees and Best Bin First (BBF)
- Ball Trees
- K-means tree
- Geometric Hashing

Motivation

- Finding *the best match* to a query descriptor **q** in a database with N prototypes $\mathbf{p}_1 \ldots \mathbf{p}_N$ costs O(N).
- For a database with thousands or millions of descriptors this is expensive.
- A tree can find *several good matches* (near neighbours) in O(log N) time.
- A hash table can find *a good match* in O(1) time.

High dimensional spaces

- Distances in high dimensional spaces are higher on average!

Expected distance for two points in D-dim unit cube

- Small distances are unlikely in $\mathbb{R}^D$ for high D.

High dimensional spaces

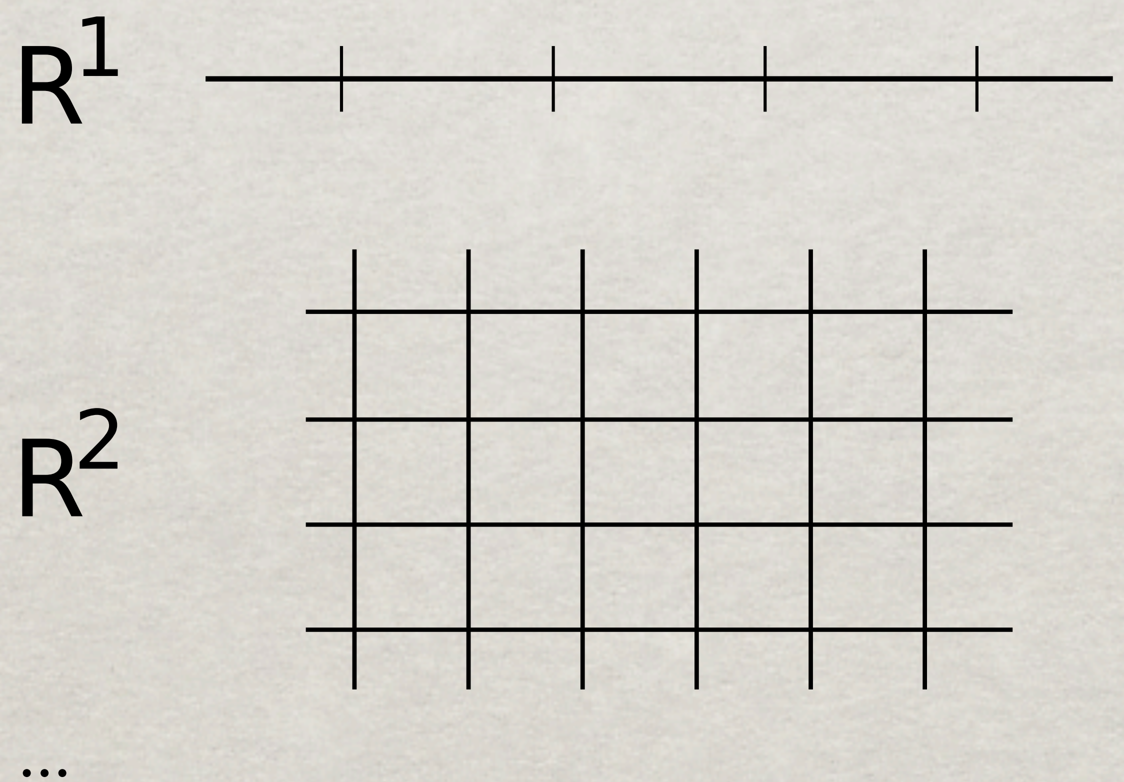
- Volume shrinks relative to area.
  Example: unit "ball"

dimension	volume	area	volume/area
1	$2r$	2	r
2	πr^2	$2\pi r$	$r/2$
3	$4\pi r^3/3$	$4\pi r^2$	$r/3$
4	$\pi^2 r^4/2$	$2\pi^2 r^3$	$r/4$

- This means that a decision region in $\mathbb{R}^D$ has increasingly more edges as D increases.

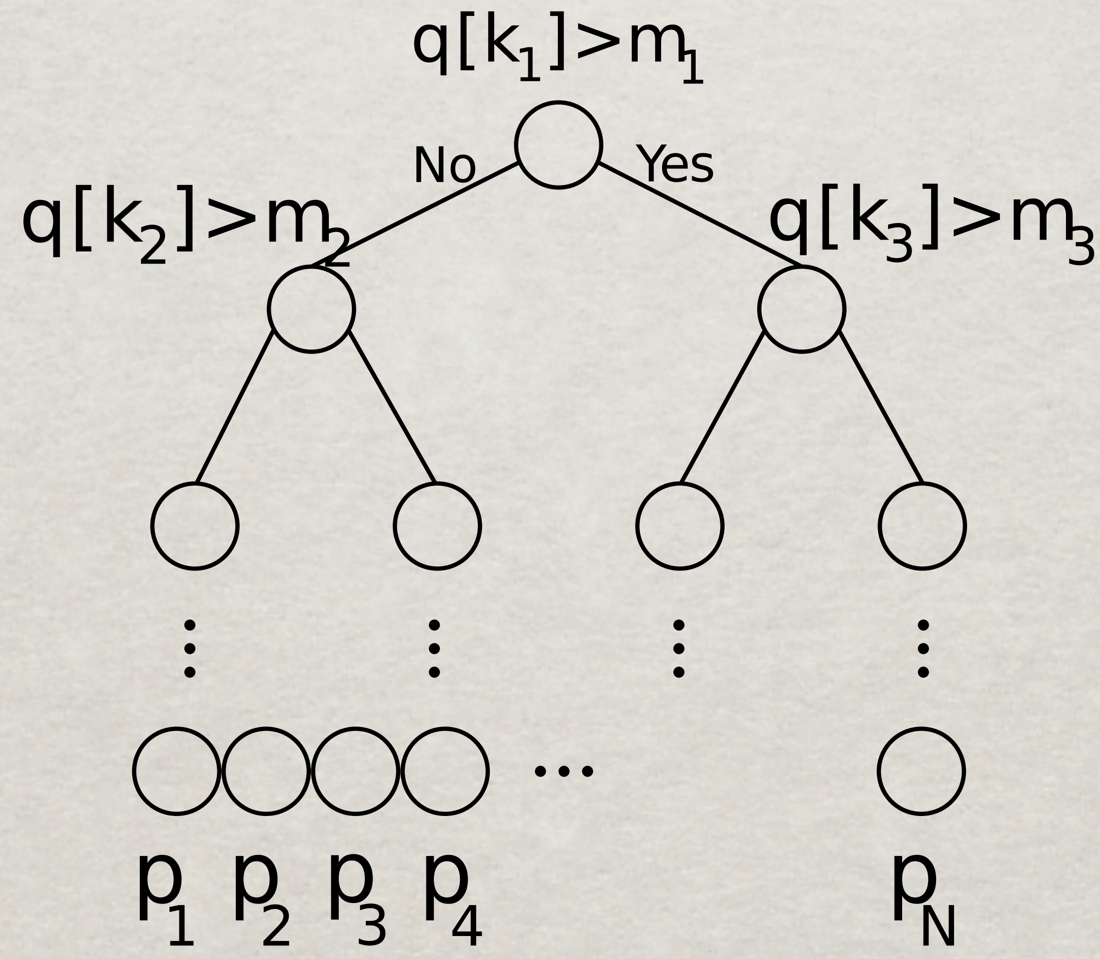
High dimensional spaces

- Box decision regions have 2 edges in R^1, 4 in R^2, 6 in R^3, 8 in R^4,... 2D in R^D



kD-Trees

- A binary tree for search in $\mathbb{R}^k$



kD-Trees

- Binary search only works in 1D,
in higher dimensions the kD-tree gives a
near neighbour.

- Tree construction algorithm:

  1. Select dimension k_n with largest variance
  2. Split dataset in two along selected dimension at median value, m_n.
  3. Repeat for each of the subsets.

kD-Trees

- Search for one neighbour is just one pass down the tree, and thus computation time is proportional to tree depth, d
- Tree depth $d = \lceil \log_2 N \rceil$
- To find more neighbours, the original algorithm suggested a depth-first search with branch pruning.
- If $e_{curr} < q[k_n] - m_n$ then skip branch.

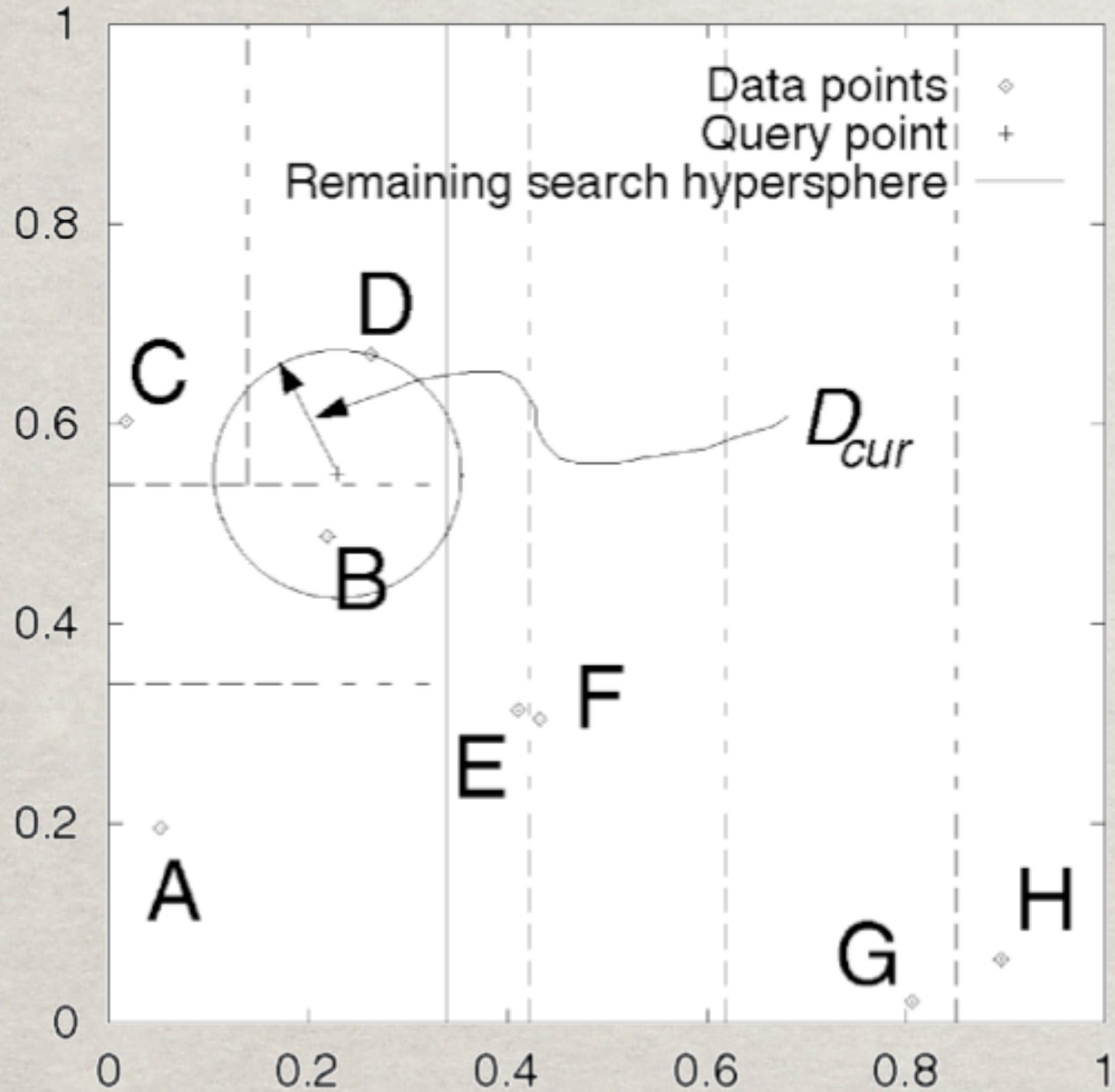
Best-bin-first

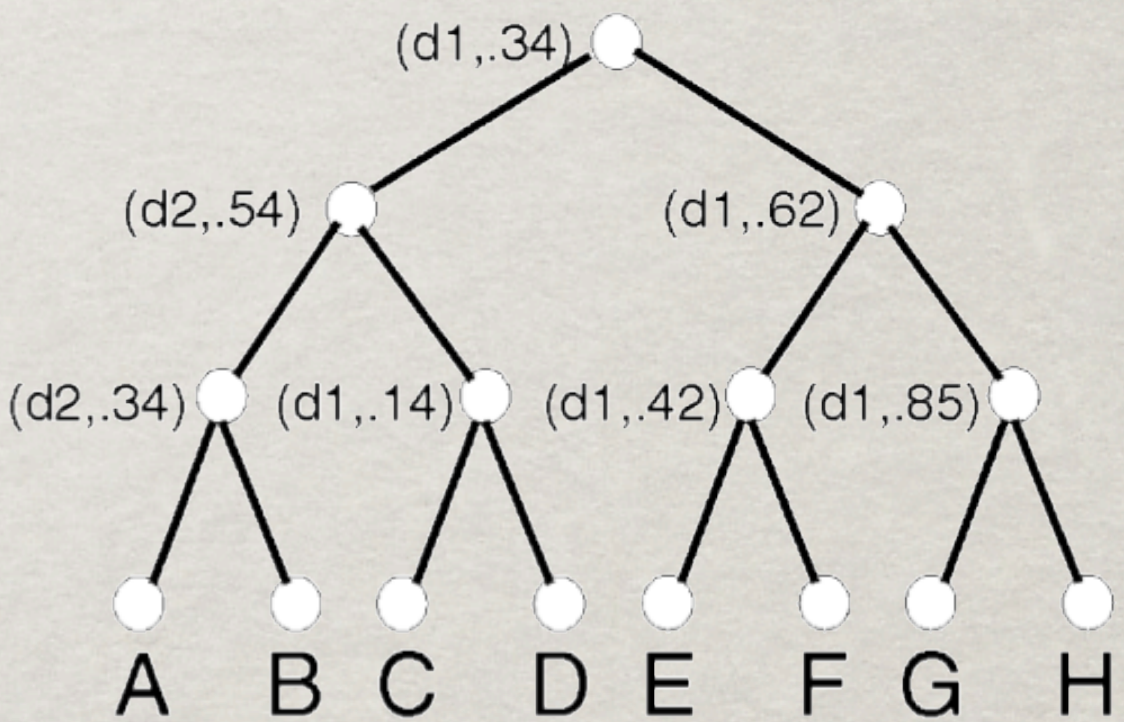
- Depth-first search works poorly in high dimensional spaces, and thus Beis&Lowe suggest a best-first search instead.

- Algorithm:

  1. At each node, store the distance $e_n=q[k_n]-m_n$ in a *priority queue*. Always insert lowest value first.
  2. Go down alternate branch of the first node in the queue if $e_n<e_{curr}$

Best-bin-first





Ball Trees

- Omohundro 89, *Metric Tree* Uhlmann 91

- A *range search* method:
  find all neighbours within a distance ϵ
  from query vector

- Each node in tree has a centre **p**, and a radius r

  p is average of all leaves
  r is maximum distance from p to a leaf

Ball Trees

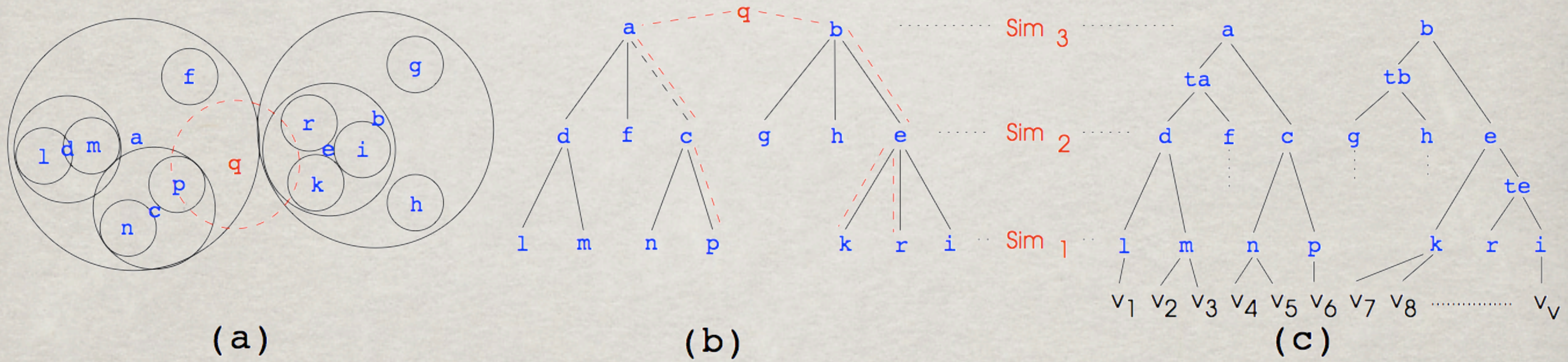
- An optimal ball tree is constructed bottom up. Very expensive. E.g. using *agglomerative clustering*:

    1. Set each sample to be one cluster
    2. Merge the two most similar clusters
    3. Repeat step 2 until no clusters are left.

- Agglomerative clustering generates a *dendrogram*, or similarity tree. This can be pruned using varius heuristics to form ball tree.

Ball Trees

- Example of a search:



Leibe&Mikolajczyk&Schiele, BMVC'06

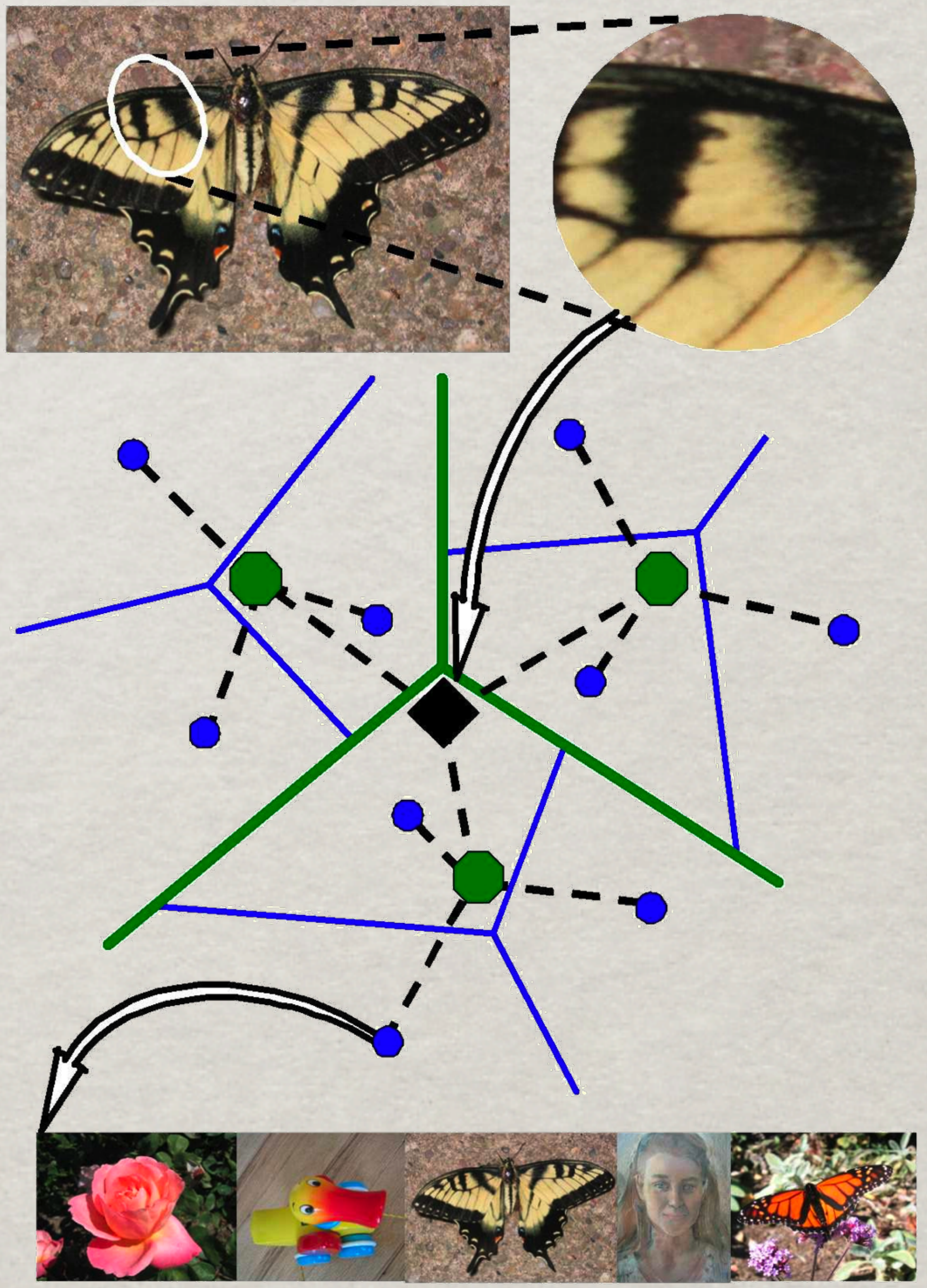
- At each node, the distances to circle centres are computed, and compared to the radius.

Ball Trees

- Advantage: Good if range search is needed. I.e. find all neighbours with $d<d_{max}$
- Disadvantages:
  - Tree construction algorithm does not scale to very large datasets
  - A ball in $\mathbb{R}^D$ is not such a useful region shape if sample density varies in the feature space.

K-means Tree

- David Nistér and Henrik Stewénius, *Scalable Recognition with a Vocabulary Tree*, CVPR06
- Hierarchical modification of the visual words idea from LE5



K-means Tree

- Building the tree:

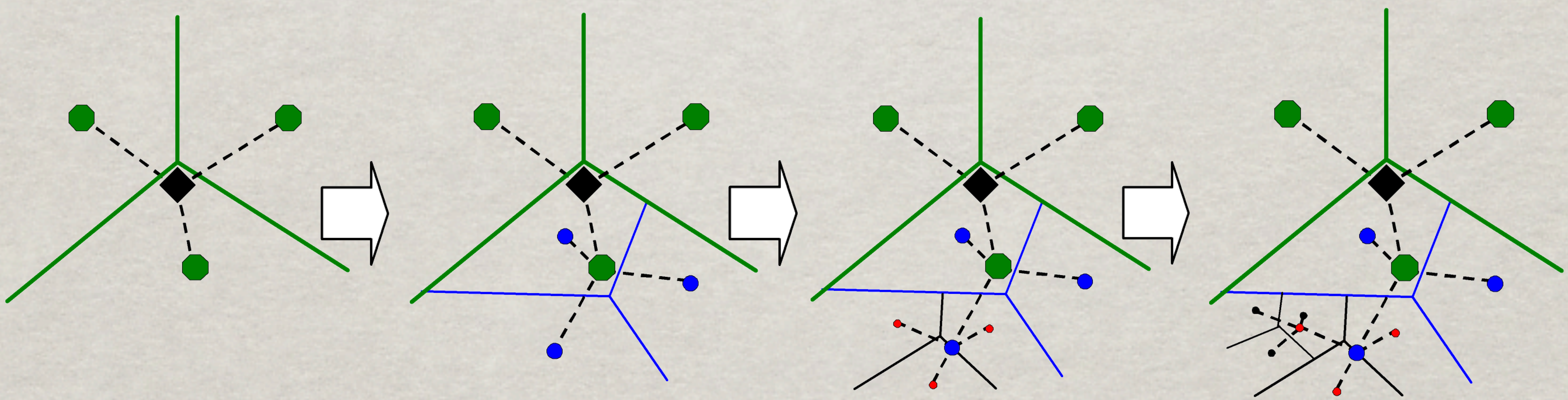
  1. Run K-means with e.g. K=10 on whole dataset.
  2. Partition dataset into K subsets using Voronoi regions
  3. Apply algorithm recursively on subsets.

- The tree gets branching factor K.

K-means Tree

- Using the tree:

    1. Compare query vector to prototypes at current level.
    2. Go down best branch

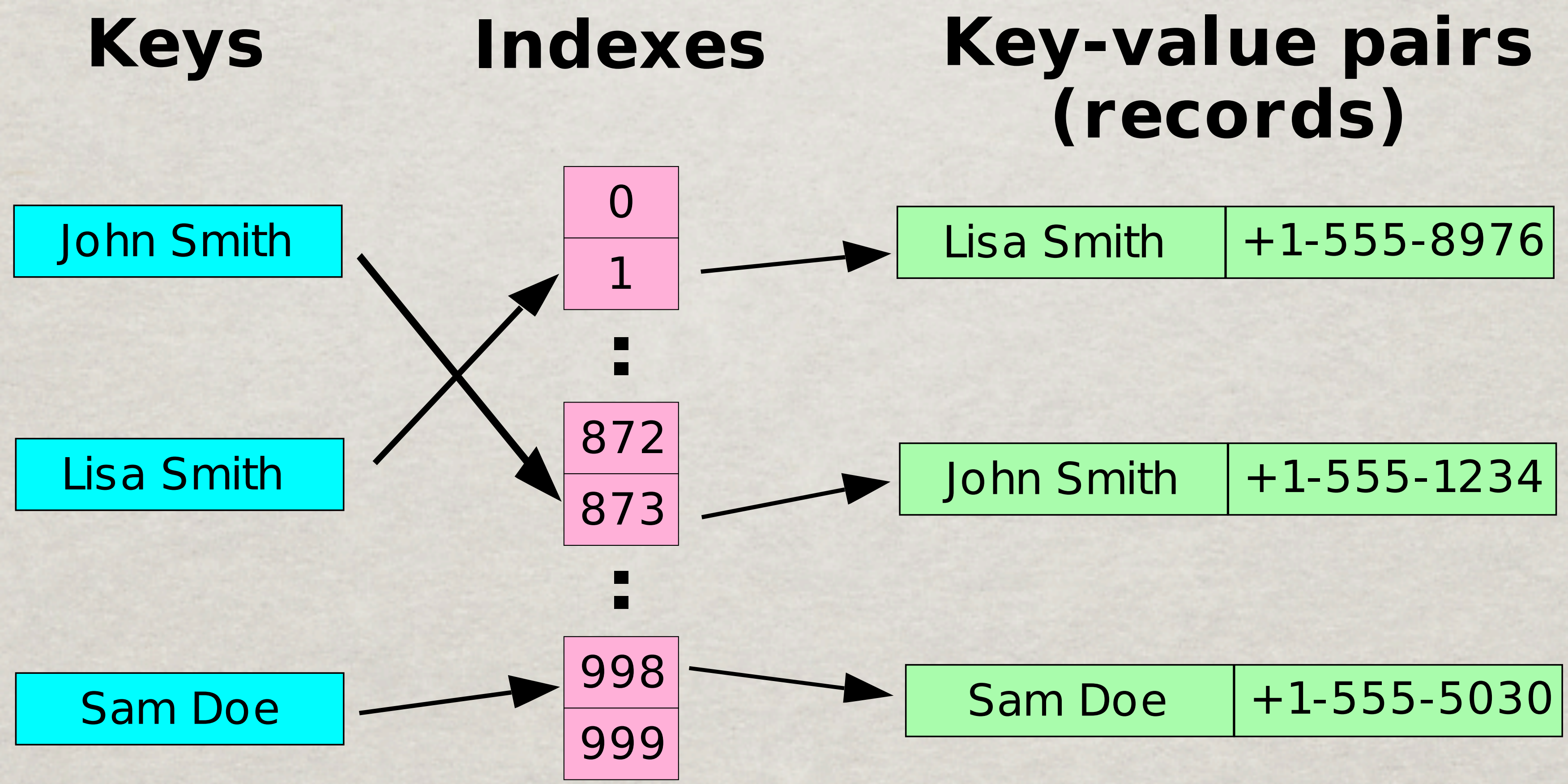


K-means Tree

- Used to compute a TF-IDF bag-of-words vector quickly.
- Much faster than non-hierarchical visual words algorithm.
- As in the kD-tree, the terminal leaf node is a near neighbour.
- Not so successful as a near neighbour search, as prototypes are far from region edge in $\mathbb{R}^D$.

Hash Tables

- An efficient way to perform lookup.



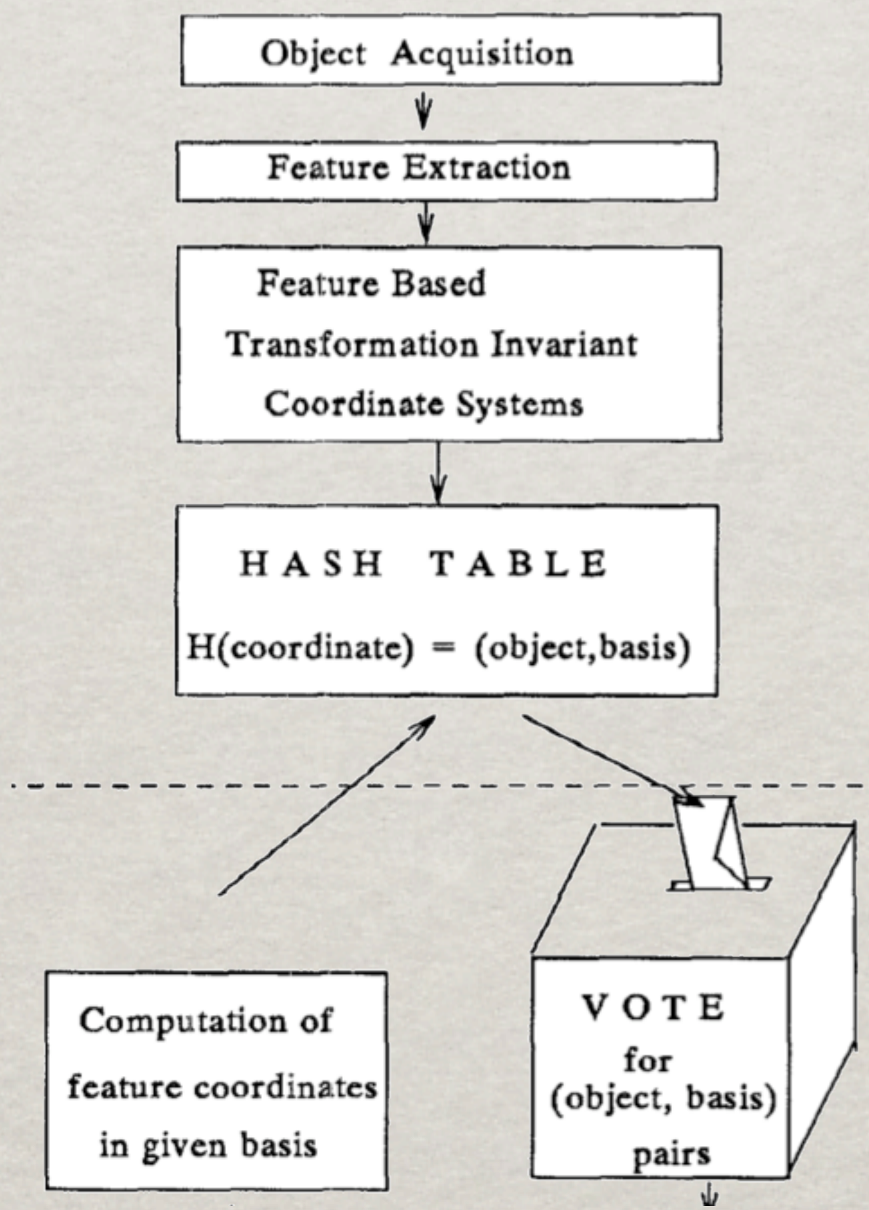
- Each key is converted to an index using a *hashing function*: index=H(key)

Hash Tables

- Lookup is O(1) instead of e.g. O(N) in a list, O(log N) in a sorted list/tree etc.
- Collisions can happen. i.e. different keys get the same index. Solved e.g. using *chaining* (linked lists), or *linear probing* (insertion at next free slot).
- Linear probing typically wants a <80% filled table.
- Hashing has poor cache locality.

Geometric Hashing

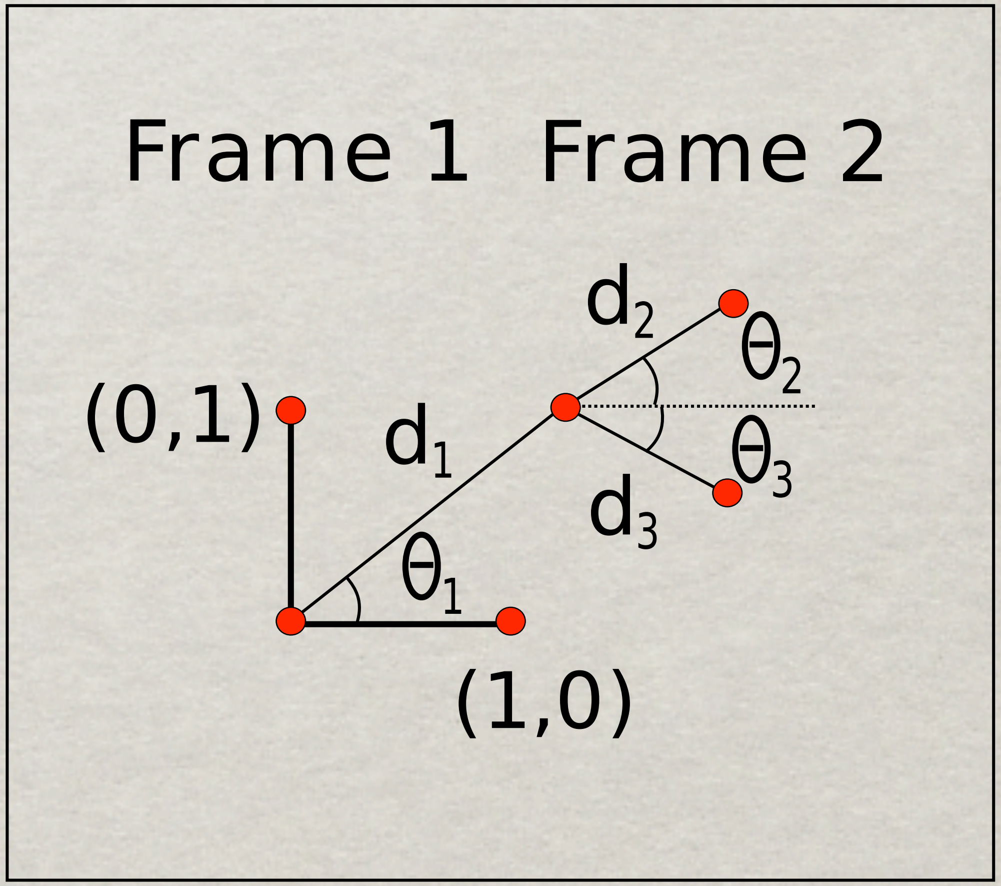
- Introduced in Lamdan&Wolfson ICCV'88



Geometric Hashing

- Modern example: Used for matching frames *without descriptors* by Chum & Matas, *Geometric Hashing with Local Affine Frames*, CVPR'06

- Use pairs of affine frames. Express frame 2 in frame 1. 25 bins for angle 16 for d_1, 6 for d_2 & d_3



- $9*10^6$ unique values for key to hash.

Geometric Hashing

- Design of H(key) is not discussed further.
- Hash tables suffer from the same basic problem as trees: Neighbouring bins might contain the closest match. More neighbours in high dimensional spaces.
- To deal with the neighbour problem, Chum&Matas construct 6 different tables (for 6 different frame constructions) and run them in parallel.

Projects

- Course is 8hp:
  5hp for lectures+articles+exam
  3hp for project.

- Project part is 2 weeks of:
  programming&research
  writing a small report.

Project suggestions

1. Comparison of kD-tree and K-means tree in terms of speed and accuracy.

2. Implementation of a bag-of-words recognition system (using existing feature detector code). Test how system parameters affect result.

3. Implementation and test of a new descriptor for a given detector.

Project suggestions

4. Implement and test a voting scheme for global geometric deformation (e.g. similarity, or affine transform) of feature locations.

5. Learn a matching metric, and compare it to least squares matching.

6. Compare Chi^2, EMD and least-squares on a problem of choice.

7. Your own suggestion.

Exam

Everyone should bring calendar next week, so we can decide on a date for the written exam.

Discussion

- Questions/comments on paper and lecture.