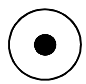
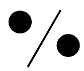
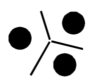
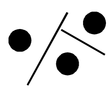
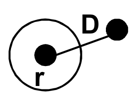
Data Structures for Computer Graphics

Searching in High-Dimensional Spaces

Lectured by Vlastimil Havran

Lecture Content

- Some principles and applications
- Tree-based data structures
  - VP-tree 
  - gH-tree 
  - GNAT 
  - mB-tree 
  - M-tree 

- Simple scan methods
- Distance matrix methods – AESA, LAESA

Applications

- Pattern recognition: fingerprints, speaker identity, optical characters, recognition of faces, etc.
- Plagiarism detection, near-duplicate detection
- Content based retrieval:
  - find a similar picture (SIFT=Scale invariant feature transforms or other feature descriptors)
  - volume data (magnetic resonance images, tomography, CAD shapes, time series)
- Searching for similar DNA sequences
- Spelling correction
- Description of a set of objects via feature vectors
- Searching performed in a set of feature vectors

Implementations

- Point queries (exact match)
- Range queries (similar objects)
- (Approximate) nearest neighbor queries (similar objects)
- All-closest-pairs queries (=spatial join query) … finding all pairs of objects that are sufficiently similar

Problem = Curse of Dimensionality

R=rectangle

- For dimensions > 15 to 20
  - data structures such as kd-trees and R-trees cease to work well
- For many tasks the dimensionality is in order of thousands
  - kd-trees get near linear query time for high dimensions
  - become slower than a naïve solution

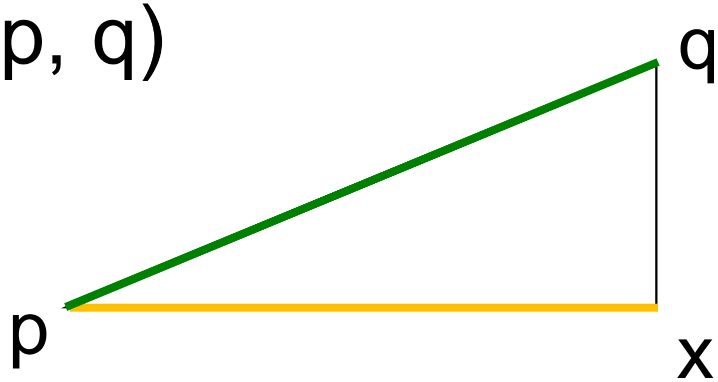
Recall Metric Spaces and Distance Functions

- Examples:
  - *Positiveness*: for all x,y in X, d(x,y) >= 0
  - *Symmetry*: for all x,y in X, d(x,y)=d(y,x)
  - *Reflexivity*: for all x in X, d(x,x) = 0
  - *triangular inequality*:

  for all x,y,z in X, d(x,y) <= d(x,z) + d(z,y)
- Examples:
  - Arbitrary metric spaces, some distance functions
  - Vector spaces with Euclidean distance
  - String with Hamming or Levenshtein distance

Triangle Inequality

- d(p, x) + d(q, x) >= d(p, q)



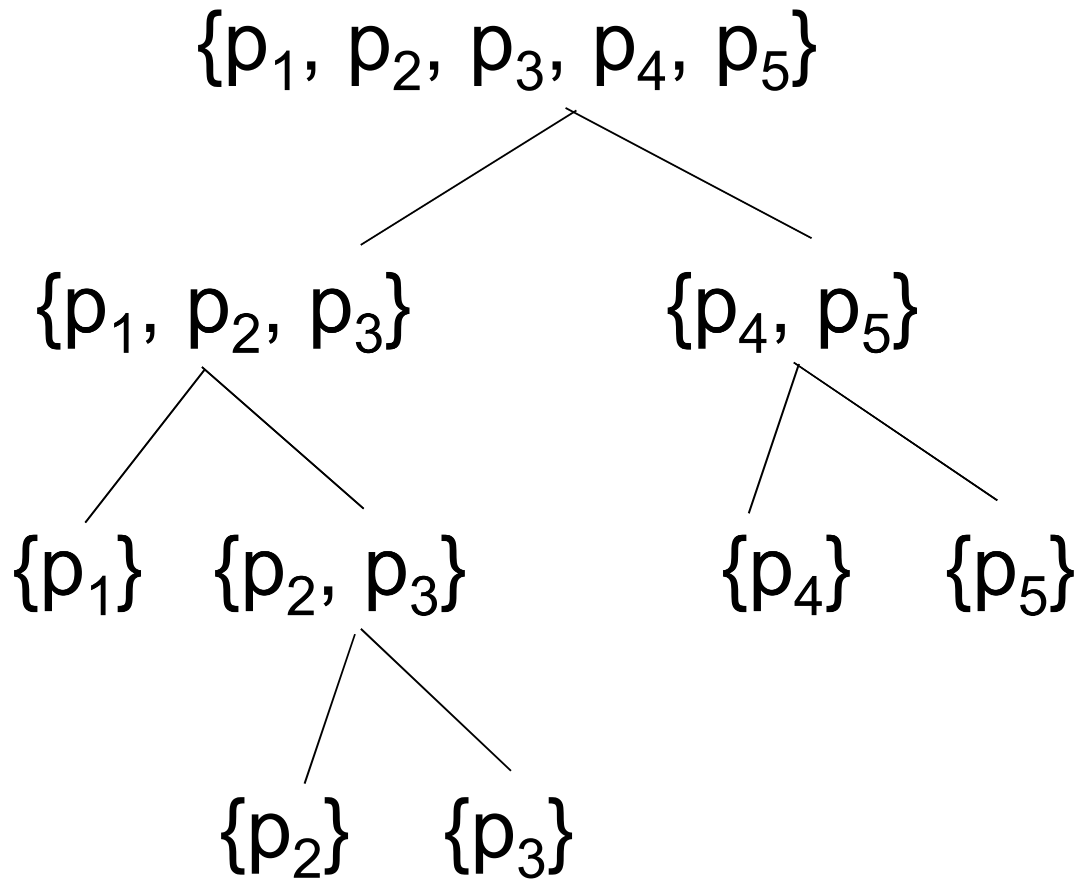
- For every element x such that is in distance from p by d(p, x) and we know d(p, q) the triangle inequality implies for d(q, x) that

$$d(q, x) \geq \underline{d(p, q)} - \underline{d(p, x)}$$

Branch and Bound Technique

- We represent the original set $S = \{p_1, p_2, p_3, \ldots, p_n\}$ by a tree
- Every node corresponds to a subset of S
- Root corresponds to S
- Every node contains some information about its subtrees that allows to provide **lower bound** for any query with the subset in the whole subtree

Range Search and NN-search

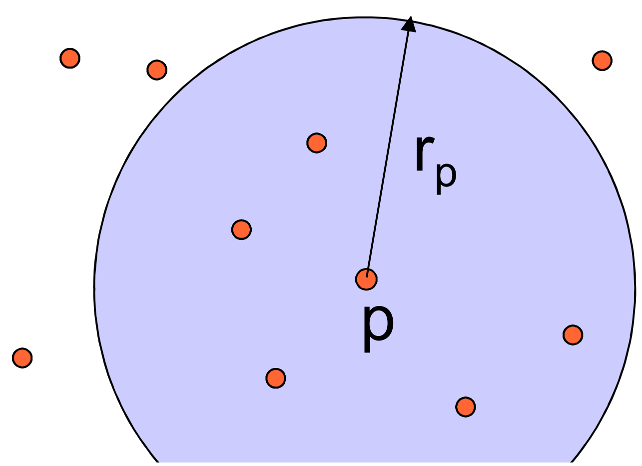


Vantage-Point Trees (VP-trees)

ConstructionAlgorithm(set of objects)

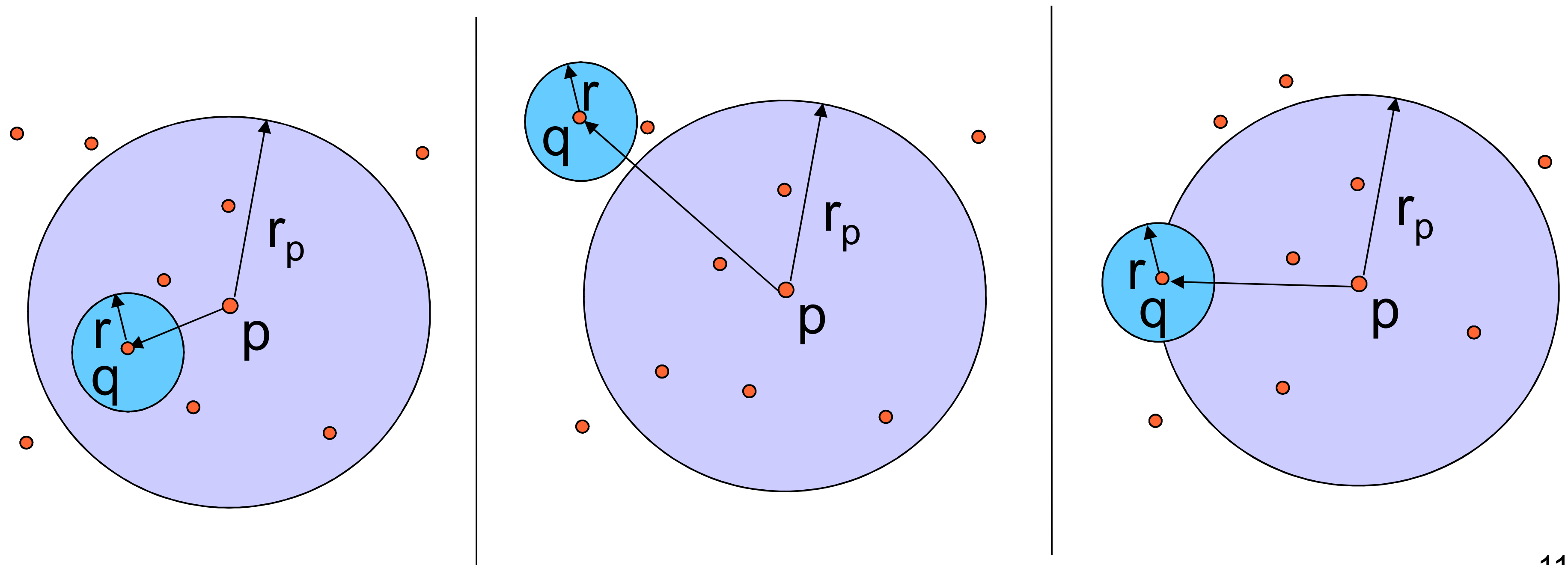
- IF there is a single object, construct a leaf and return.

  ELSE choose randomly some object p in a set. ENDIF
- Choose partitioning radius r_p
- Put all p_i such that $d(p_i, p) <= r$ into "inner" part, other points to the "outer" part of a ball
- Recurse

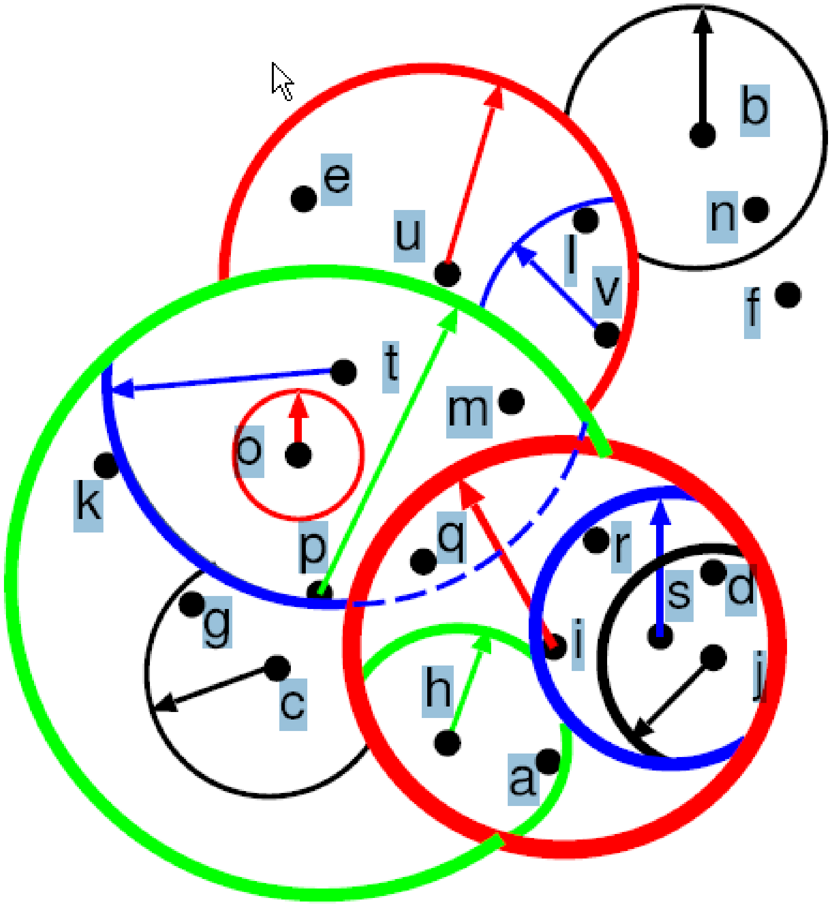


For Range Search

- Circular range search with radius r
- IF $d(p, q) < (r_p - r)$, THEN prune the outer branch
- IF $d(p, q) > (r_p + r)$, THEN prune the inner branch
- Otherwise it holds: $r_p - r < d(p, q) < r_p + r$ and we have to visit both branches



VP-tree Example



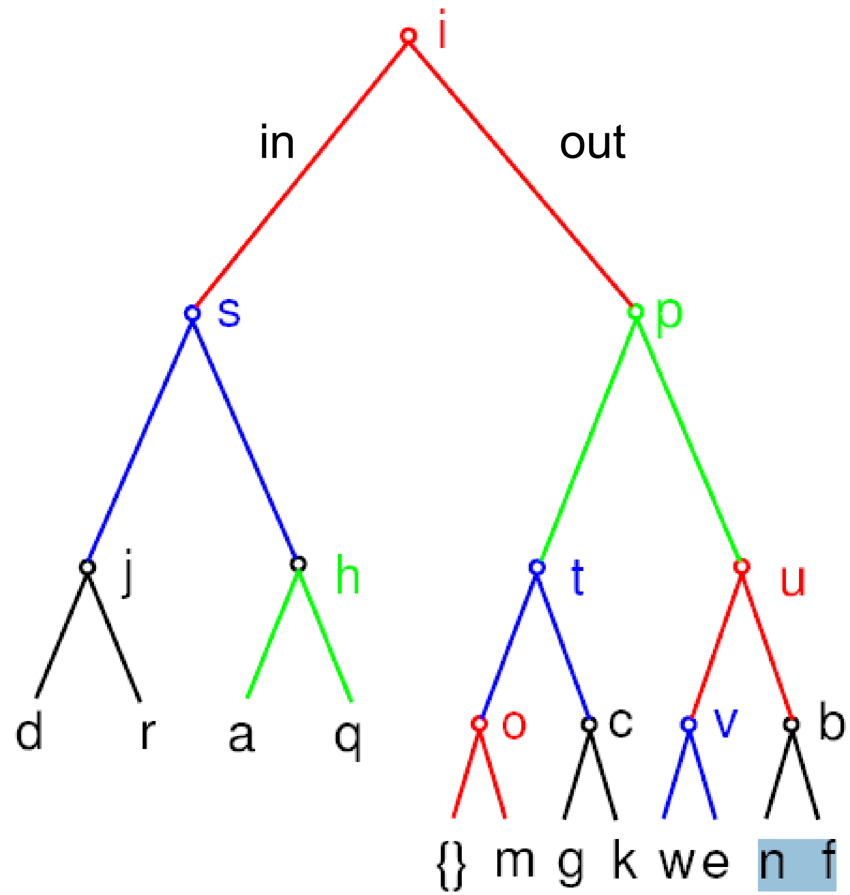


Image from SIGGRAPH 2007 course notes by Samet

Variants of VP-trees

- Burkhard-Keller tree
  - pivot used to divide the space into m rings
  - m-ary tree at each node
- MVP-tree
  - collapse several levels of VP-tree to a single node
  - use the same pivot for different nodes in one level
- Post-office tree
  - use (r_p + eps) for inner branch
  - (r_p − eps) for outer branch

Generalized Hyperplane Tree (gH-tree)

We use generalized hyperplane partitioning method based on two pivots

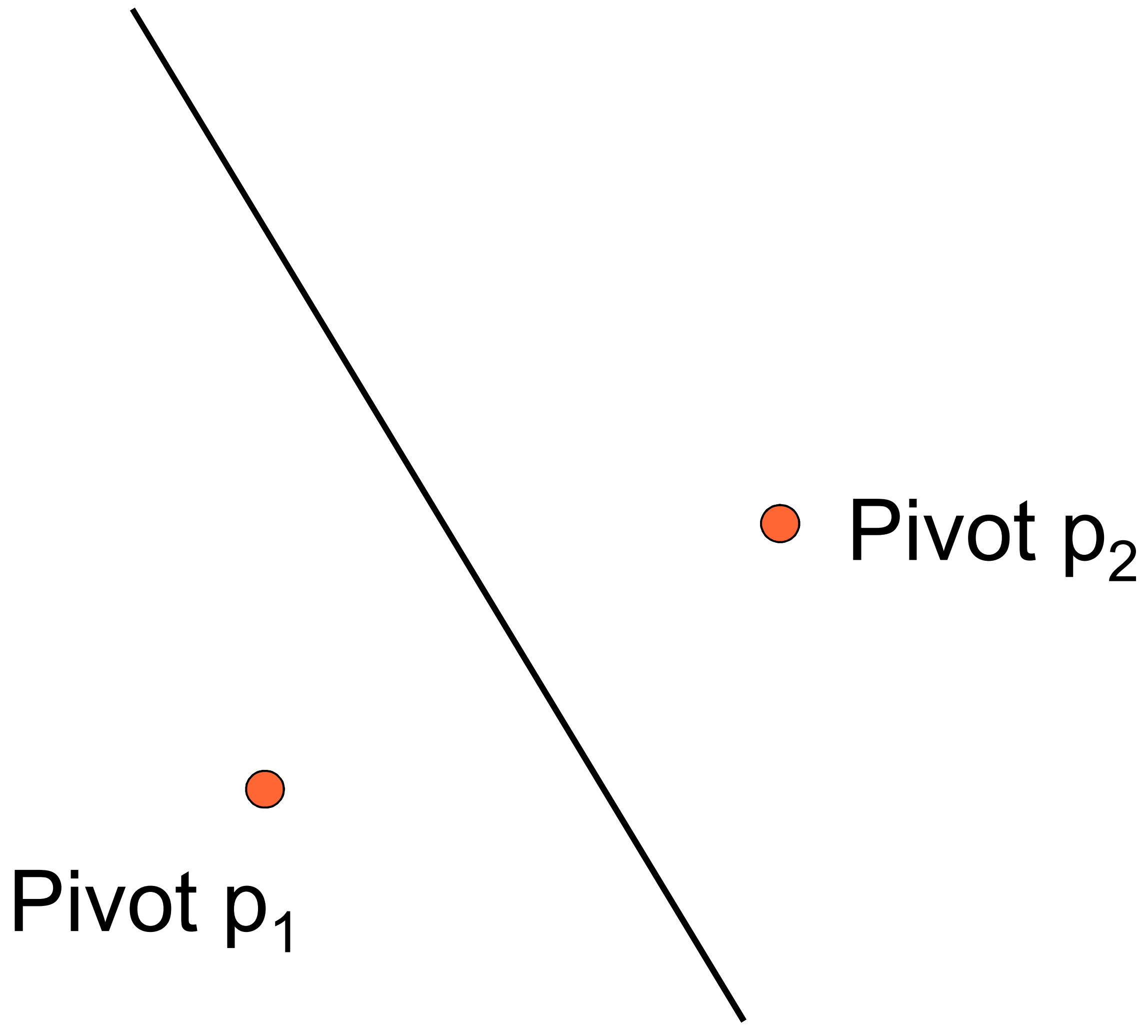
- Select p_1 and p_2 from S and partition S into two subsets S_1 and S_2 so for the objects we apply this rule:

  $S_1 = \{ o \text{ in } S \setminus \{p_1, p_2\} \text{ and } d(o, p_1) <= d(o, p_2) \}$

  $S_2 = \{ o \text{ in } S \setminus \{p_1, p_2\} \text{ and } d(o, p_1) > d(o, p_2) \}$

- Apply recursively, yielding a binary tree

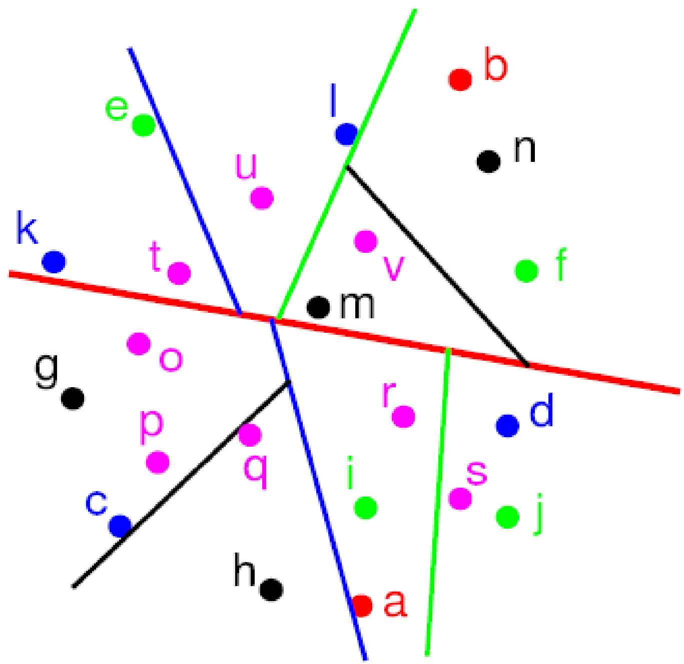
General Hyperplane Concept



Properties of gH-trees

- Each interior node contains two pivots, pivot p_1 and pivot p_2
- A hyperplane corresponds to all points o satisfying condition: $d(p_1,o) = d(p_2, o)$
- Objects in S_1 are closer to p_1
- Objects in S_2 are closer to p_2
- The regions of a tree are implicit (defined by pivot objects) instead of being explicit

gH-tree Example



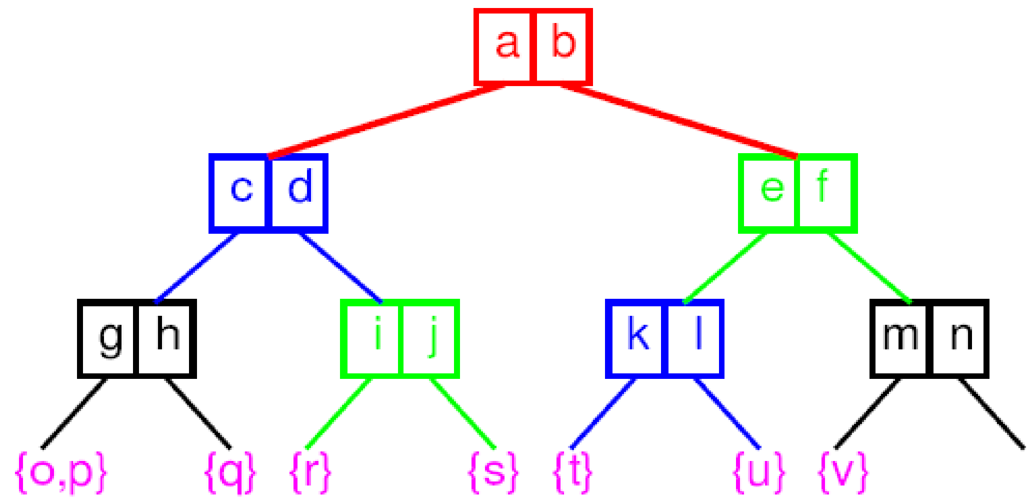


Image from SIGGRAPH 2007 course notes by Samet

Circular Search with gH-tree

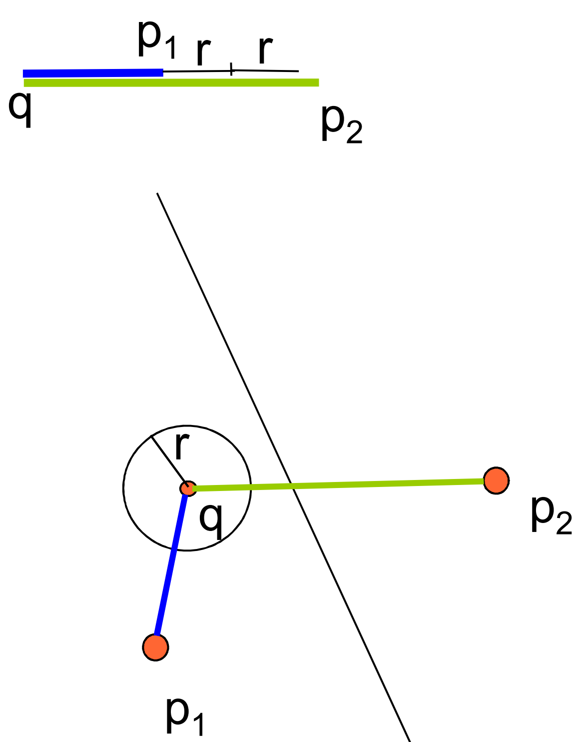
- Query "q" with radius "r"
- Left subtree containing pivot p_1 is visited if and only if when:

$$d(q,p_1) - d(q,p_2) < 2r$$

- Right subtree containing pivot p_2 is visited if and only if when:

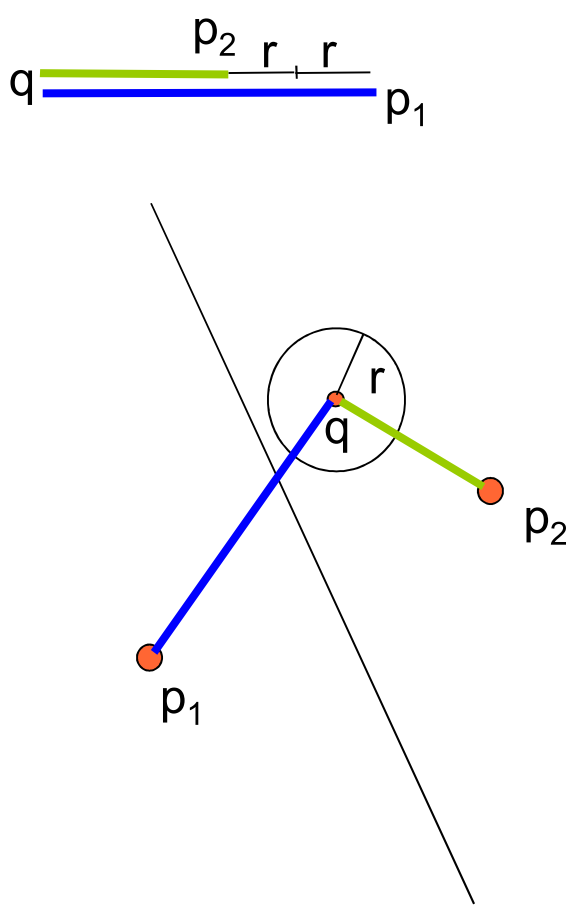
$$d(q,p_2) - d(q,p_1) < 2r$$

- These rules are approximate, but work conservatively.



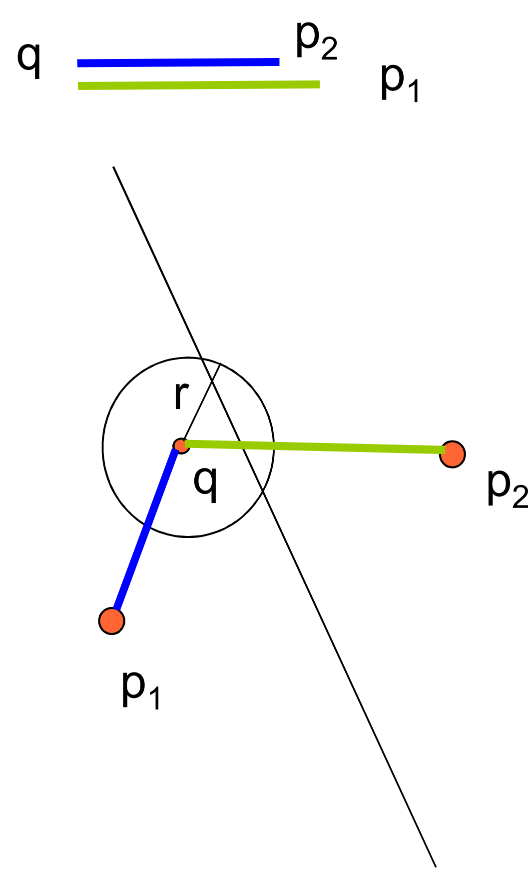
$0 > d(q,p1) - d(q,p2) < 2r$

$d(q,p2) - d(q,p1) > 2r$



$0 > d(q,p2) - d(q,p1) < 2r$

$d(q,p1) - d(q,p2) > 2r$



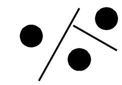
$0 < d(q,p2) - d(q,p1) < 2r$

$0 > d(q,p1) - d(q,p2) < 2r$

Geometric Near-neighbor Access Tree (GNAT)

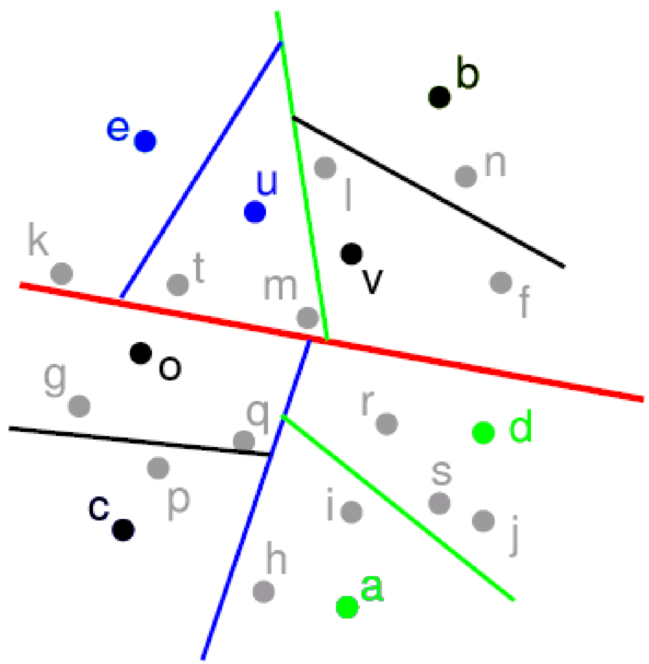
- Generalization of gH-tree
- We use more than two pivots to partition the data set at each node – m pivots
- Possible Heuristics
  - Pickup 3*m pivots randomly
  - First pivot randomly from 3*m pivots
  - Second pivot: the farthest one from the first pivot
  - Third pivot: the farthest one from the first and second pivot
  - N-th pivot: similarly - total sum from all previous pivots is maximized.

mB-tree – Monotonous Bisector Tree



- Similar to the gH-tree
- Inherits one pivot from ancestor node
- Advantage - fewer distances computations
- Deeper tree

mB-Tree Example



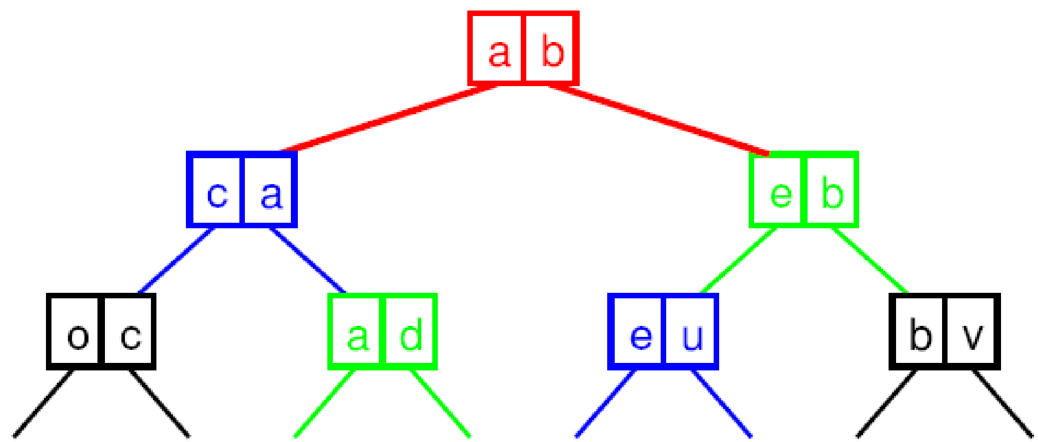
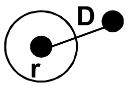


Image from SIGGRAPH 2007 course notes by Samet

Nearest Neighbor Search with Tree Structures

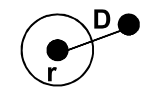
- Applies to VP-tree, gH-tree, GNAT, mB-tree
- We use depth first search starting from a root with **a priority queue** (best fit search)
- The search is finished once we have all the nodes to be visited farther than the closest object found so far

M-tree



- Dynamic data structure similar to GNAT
- All objects stored only in leaf nodes, some objects used as a pivots at the same time
- Inner node n has 2 pivot entries. Entry:
  - p – pivot
  - r – corresponding covering radius
  - D – distance value from p to the parent pivot
  - T – reference to a child node of n

M-tree Construction



- Unlike previous tree-based methods constructed from bottom to top – can be used for dynamic data
- The insertion of point "p" uses heuristics, for example:
  – Insert "p" to such a leaf which covers it (radius)
  – If there is not such a leaf or more such leaves contain p, pickup such a leaf which has the closest distance to "p".
  – Upon insertion update covering radii up to the root node
- Once a leaf has too many entries, then it is split - two pivots are selected and are added to the parent node, which can cause another split
- The details and other heuristics in the paper: Ciaccia et al., 1997: *M-tree: an efficient access method for similarity search in metric spaces*.

Simple Methods using Sequential Scan over dimensions

- **Partial Sum:** When the partial sum of squared differences of a candidate already exceeds the squared distance to the nearest neighbor so far, the candidate is rejected
- **Sampling:** We select a predefined part of each feature vector and pre-select the candidates for which we further compute the distance – this yields approximation (without guarantee)

Simple Methods contd.

- Recall that the distance is a sum of terms.
- For **partial sum** we sum **all terms** until we exceed already found minimum distance. The result is *exact*.
- **Sampling:** we sum only **some terms** so we cannot guarantee the exactness. We can try to select such dimensions that we maximize the distance results. The result is *only approximate*.

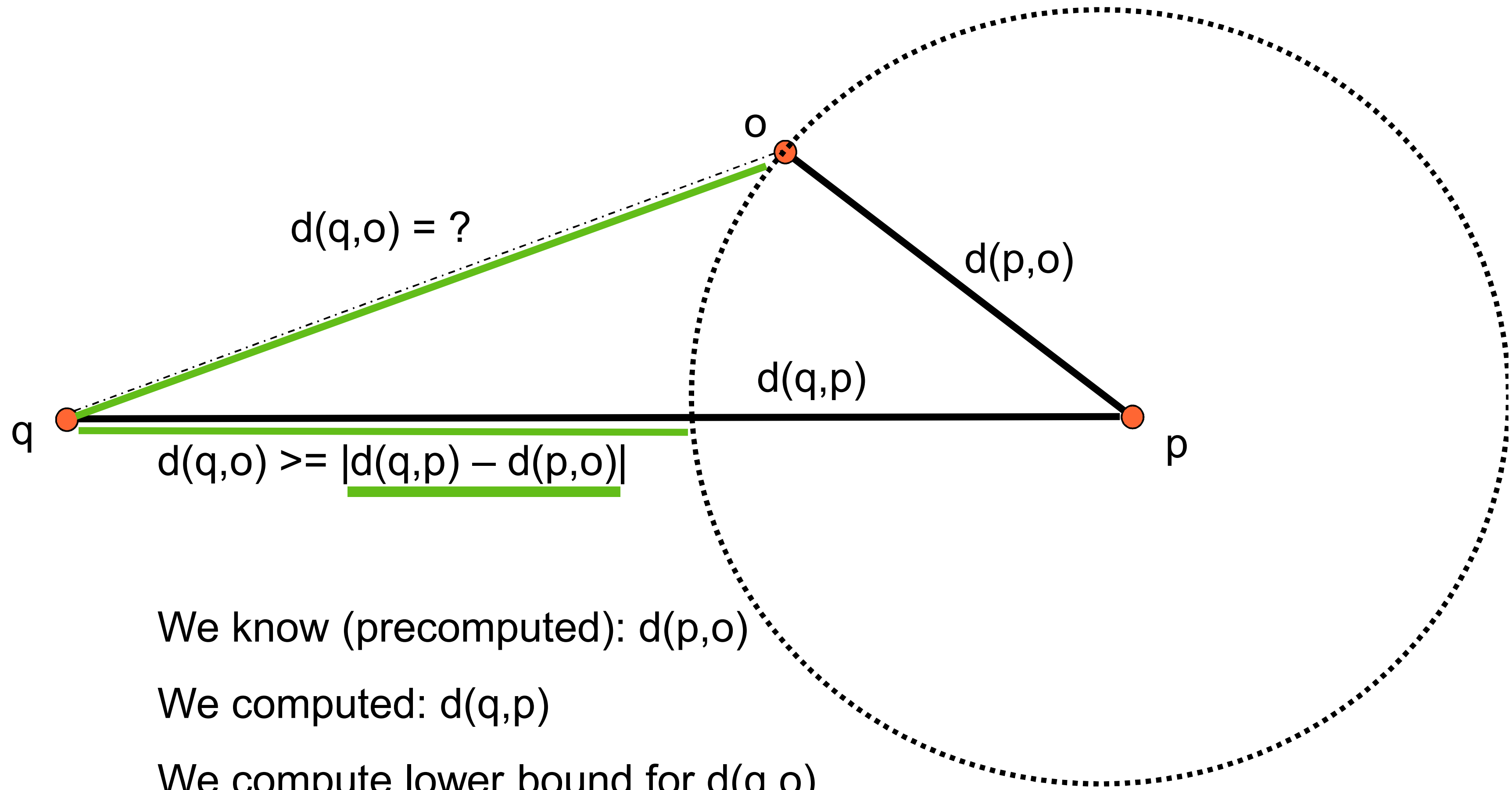
Array-based Distance Methods

- Based on computing distance between some or all data entries in the structure
- Based on the known distances we can prune the search extensively
- They can be time efficient but memory demanding in order $O(N^2)$
- Two methods: AESA and LAESA

AESA – Approximating and Eliminating Search Algorithm

- It precomputes all the distances between the objects
- Hence the space complexity is $O(N^2)$
- During nearest neighbor search it selects an arbitrary object (pivot p) and establishes lower bound distances to all other objects (o)
- The **number of distance computations for search** can be **remarkably low**
- It can also be used for range searching and kNN search

AESA – Graphical Illustration



We know (precomputed): d(p,o)

We computed: d(q,p)

We compute lower bound for d(q,o)

AESA - Use of Triangle Inequality

- For a query "q", an object "o", and a pivot "p" we know:

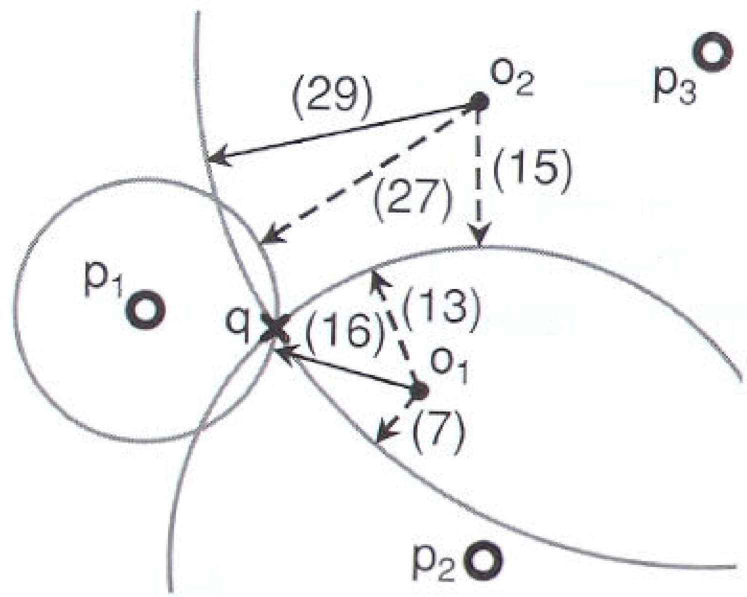
  $d(q,o) >= |d(q,p) - d(p,o)|$

- If we have more pivots, the greatest lower bound d(q,o) is computed as:

  $d(q,o) = Max(|d(q,p_i) - d(p_i,o)|)$

  for all pivots i where we have

  already the distance $d(q,p_i)$



AESA NN-search

- Mark all objects as candidate NN-neighbors
- Given a query "q" pickup arbitrary object "p" and add it to a set "P" (set of pivots)
- Compute closest distance so far e = d(q,p)
- While more than one candidate is possible NN-neighbor do in a loop:
  - Computing greatest lower bound $d(q,o) = Max(|d(q,p_i) - d(p_i,o)|$ for all possible remaining candidates and all pivots p_i in "P"
  - Exclude those remaining candidates from the computation that are farther than "e" from the query (including those in "P")
  - Select another pivot (the estimated closest candidate) and compute new distance d(q,p) and add it to the set "P"

AESA conclusion

- If you have a small number of candidates and enough memory – very low number of distance calculations
- The use of AESA makes sense only if the number of queries is substantially higher than the number of data entries in the distance array
- It can be used in dynamic version – computing distances on the demand

LAESA – linear AESA

- Selects only limited number M of pivots given by a user
- The space complexity is therefore only O(N*M) where M is the number of selected pivots
- The pivots are selected in such a way that they are maximally separated
- The search becomes more complicated

LAESA versus AESA

- The difference during the search is that we do not exclude from candidate objects the pivot objects.
- The search is faster with increasing M
- We can tradeoff the space complexity and the search complexity

Dimension Reduction Techniques

- Principle is to project the original space to some other space, for example a plane that is described by fewer coordinates
- The selection of a projection plane is of crucial importance for the algorithm performance – we reduce such dimensions to not to lose too much of the information in the data.
- Currently active research area (PCA, LPCA, …) although used for many years

Literature

- G.R. Hjaltason and H. Samet: *Index-Driven Similarity Search in Metric Spaces*, 2003
- E. Chavez, G. Navarro, R. Baeza-Yates, J. L. Marroquin: Searching in Metric Spaces, 2000.
- H. Samet: Foundations of Multidimensional and Metric Data Structures, 2006. (chapter 4)

Software:

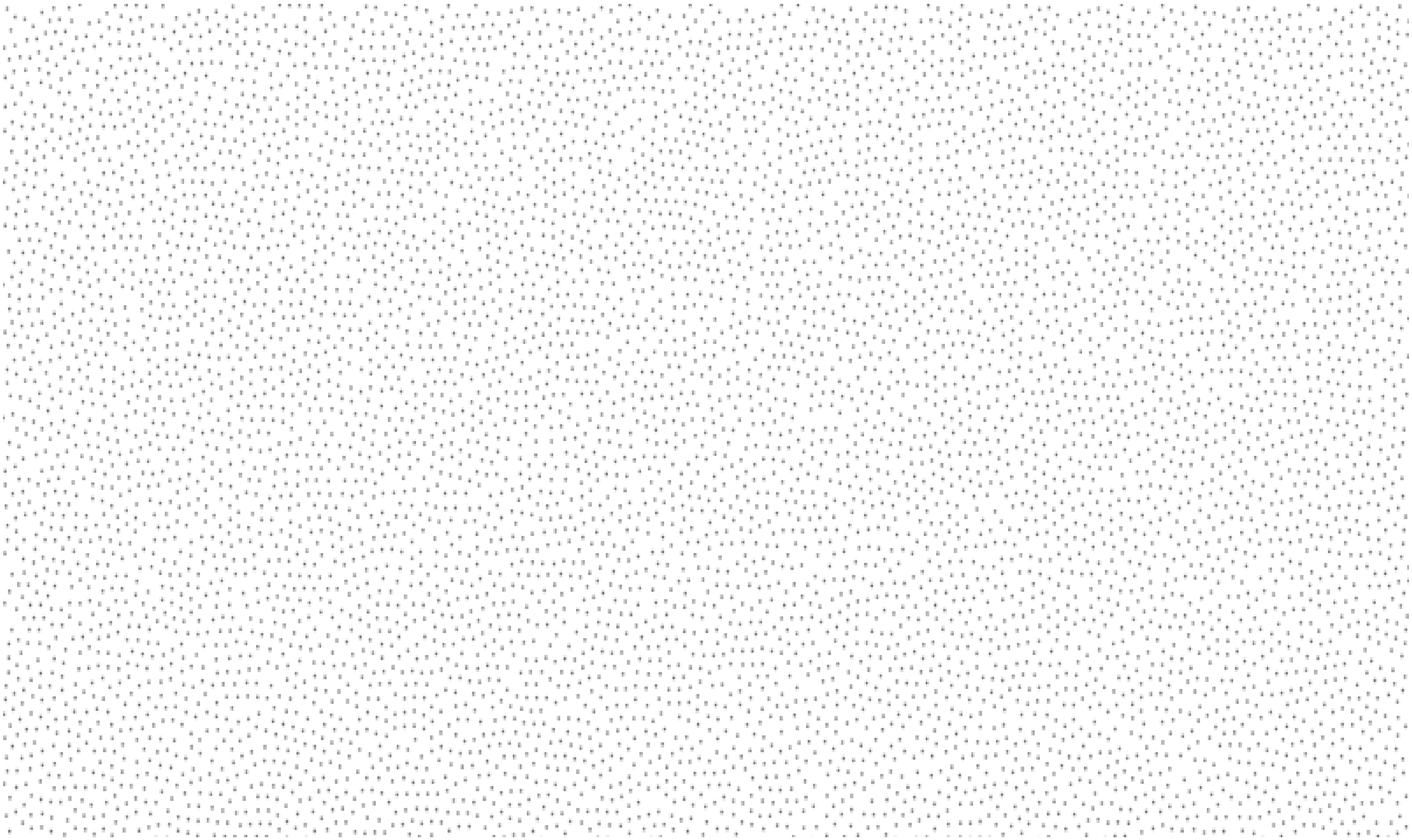
Metric Spaces Library: http://www.sisap.org

Introduction to Sampling

- Many applications require sampling of different types.
- For many reasons uniform equidistant sampling is not a right choice.
- A Poisson-disk point set is a set of points taken from a uniform distribution in which no two points are closer than some minimum distance "R".
- Blue noise characteristics
  - density proportional to f over a finite frequency range.
  - power density increases 3dB per octave

Poisson-disk Point Set Example

Minimum low frequency components and no spikes in energy.



Generation: Hierarchical Dart Throwing

- Initial active squares
- Check if the square is covered

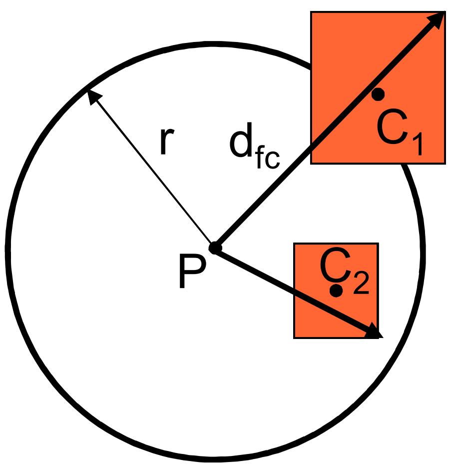
$$d^2_{fc} = (|x_c - x_p| + b/2)^2 + (|y_c - y_p| + b/2)^2$$

COVERED if $d^2_{fc} < r^2$ center

$C=(x_c,y_c)$ … square center

$P=(x_p,y_p)$ … disk center

b … size of a square



Sampling Algorithm Overview

- Put base level squared on active list 0 (the base level)
- Initialize the point set to be empty
- **While** there are active squares
  - Choose an active square S with probability proportional to the area.
  - Let “i” be the index of the active list containing “S”.
  - Remove S from the active lists.
  - Choose a random point, P, inside square S.
  - **IF** P satisfies the minimum distance condition **THEN** **(use grid index, O(1))**

    add P to the point set.
  - **ELSE**
    - ➔Split S into four child squares.
    - ➔Check each child square to see if it is covered
    - ➔Put each non-covered child of S on active list i+1
  - **ENDIF**

Algorithm Summary

- In practice O(N) complexity for sampling N samples when we have O(1) search to find nearest neighbors.
- Practically 30 times faster than other algorithms published so far.
- Details in the paper:

K. B. White, D. Cline, P.K. Egbert: *Poisson Disk Point Sets by Hierarchical Dart Throwing*, 2007.

Thank you for your attention!