Spatial databases

Rasmus Pagh

Reading: RG 28, blog post, [BKOS00, sec. 5.3]

Today

- Spatial databases
- Multi-dimensional indexing:
  - Grid files
  - kD-trees
  - R-trees
- Spatial indexing in Oracle
- More multi-dimensional indexing:
  - Range trees
  - Space-filling curves

Spatial databases

Examples:

- Geographic Information Systems (GIS)
- Computer-Aided Design (CAD)
- Multi-media databases (*feature vectors*)
- Traffic monitoring

More generally, spatial/multidimensional indexing techniques may be relevant to all queries that contain a range or point condition on more than one attribute.

Spatial data

Two main types:

- Point data (GIS, feature vectors, OLAP)
- Region data: Objects have some spatial extent, e.g. polygons.

- We will focus on point data, but some of the techniques we will talk about also work for region data.
- We will talk mostly about 2D, but all ideas extend (with some cost) to higher dimensions.

Spatial queries

Examples:

- Orthogonal range queries:
  - Select all points with coordinates in given ranges.
- Nearest neighbor queries:
  - Find the nearest point to a given query point.
- Spatial join:
  - Join with spatial condition, e.g. ”are closer than 1 km”. Not discussed today.

Problem session

- Consider an orthogonal range query:

```
SELECT x,y FROM map
WHERE (x BETWEEN 1000 AND 2000)
  AND (y BETWEEN 2000 AND 3000)
```

- Assume there are *n* rows in map, and that we have covering B-tree indexes on (x,y) and (y,x).
- What strategy may the DBMS use to perform the query, and what will be the number of I/Os?
  - Use the selectivity of the range conditions in your answer.

Conclusion – B-trees

- B-trees may give an acceptable solution if one of the range conditions has very high selectivity.
- However, often the case that both range conditions have low selectivity, while their conjunction has high selectivity.
- Ideally, we would like an index whose time depends on the number of points satisfying *both* range conditions.

Grid files, in a picture

Grid file properties

- Simple implementation – reduction to clustered index on cell ID.
  - Especially easy when the grid is uniform.
- Weak point: The number of points in a cell may vary a lot when points are not uniformly distributed.
  - Sometimes need 1 I/O to retrieve few points.
  - Sometimes need many I/Os to retrieve the points in a single cell.
- More rubust implementation:
  - Clustered B-tree on (x-grid-coord,y-coord)
  - Make sure there are at most ≈$(NB)^{0.5}$ points for each x-grid-coordinate.

Non-uniform grid file search

Grid file analysis (robust impl)

- Worst-case: First and last search may cost $(N/B)^{0.5}$ I/Os, without returning any results.
- Each additional search costs $O(\log_B N)$ I/Os (assume this is O(1)), plus the cost of reporting results.
- Worst-case: Search in at most $(N/B)^{0.5}$ x-grid-coords.

- Total cost of $(N/B)^{0.5}$ I/Os, plus the cost of reading the results.

kd-trees

- Generalization of ordinary search tree.
  - External memory version sometimes called kdB-tree.
- An internal node splits the data along some dimension.
  - In 2D, the splitting alternates between horisontal and vertical.
- Similar to *Quad-trees*, implemented in Oracle (deprecated feature).
  - Quad-trees split on two dimensions at each internal node.

kd-tree in a picture

kd-tree properties

- Simple generalization of search trees.
- Can adapt to different densities in various regions of the space.
- Efficient external memory variant.

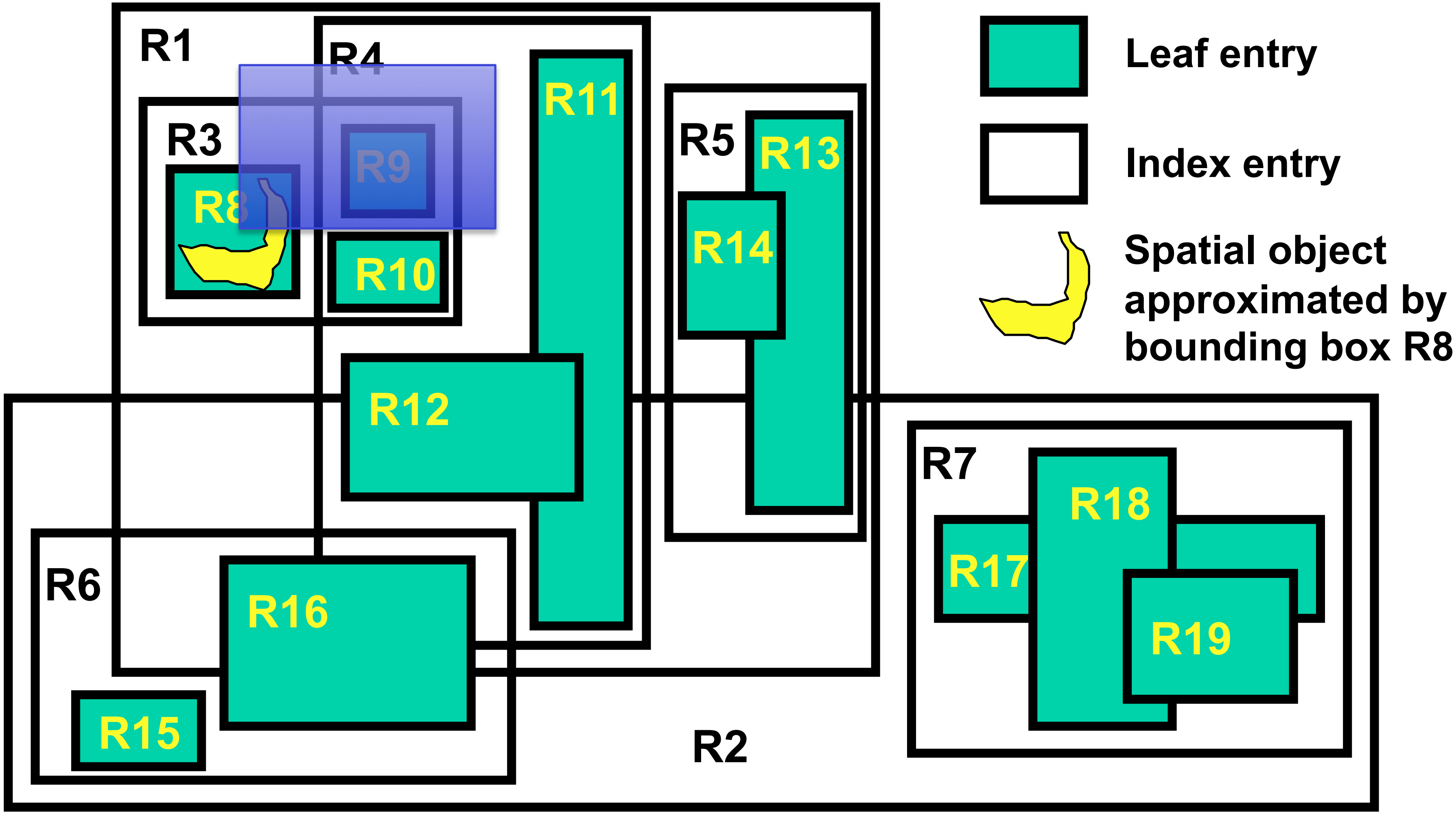
- <u>Weak point</u>: Very rectangular range queries may take long, and return only few points.
  - A 2D query on N points may visit up to $N^{1/2}$ leaves, even when there are no results.

R-trees

- *Another* generalization of B-trees.
- An internal node splits the points (or regions) into a number of rectangles.
  - A rectangle is a ”multidimensional interval”.
  - Rectangles may overlap.
- Balancing conditions, and how balance is maintained, is similar to B-trees.
  - Especially, depth is low.
  - However, searches may need to explore *several* children of an internal node, so search time can be larger.

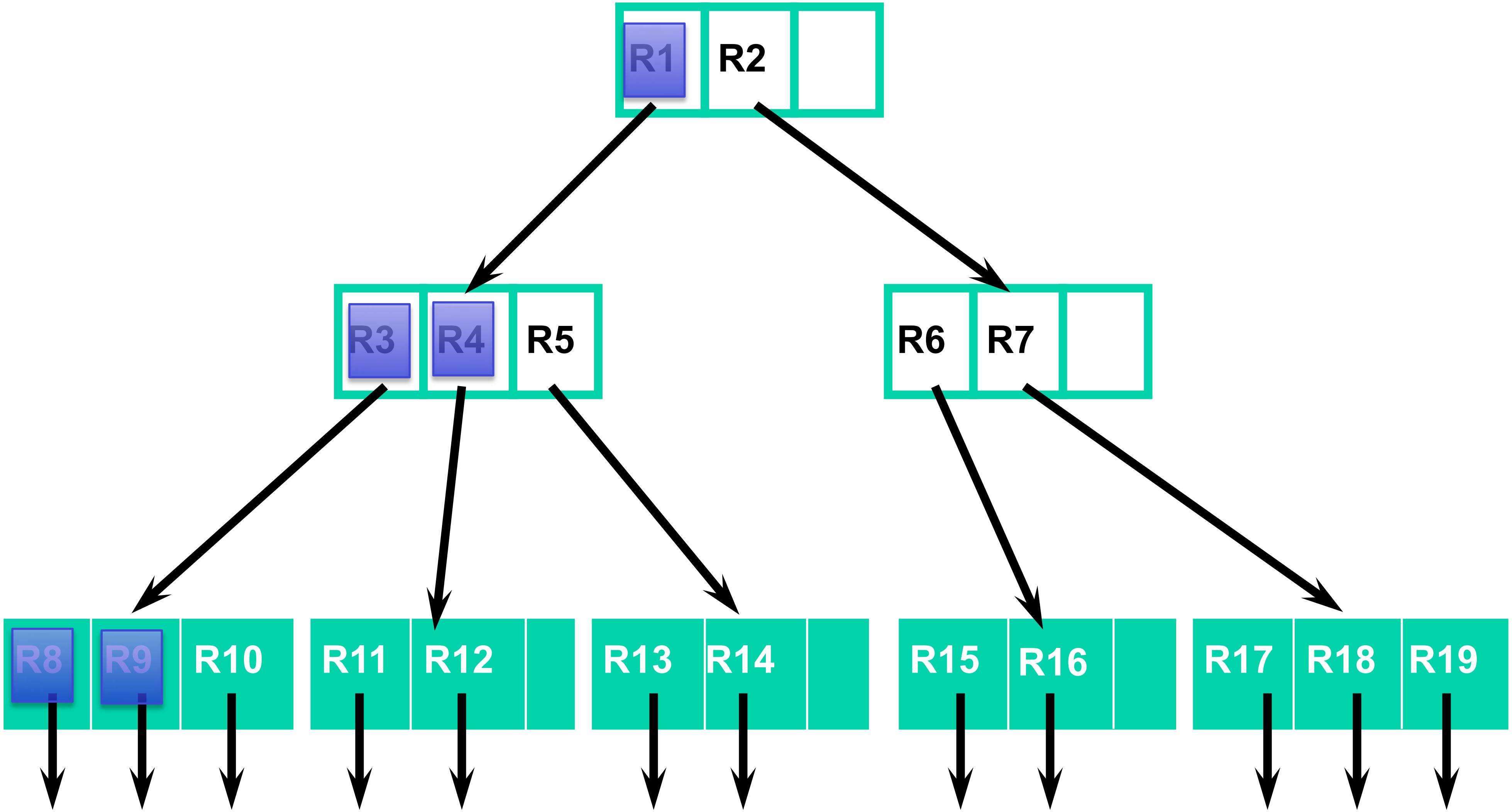
R-tree example

(slide by Ramakrishnan and Gehrke)



R-tree example

(slide by Ramakrishnan and Gehrke)



R-tree properties

- Theoretically, not known to be stronger than kd-trees.
  - Except in special cases.
- The most widely implemented spatial tree index.
  - Flexible
  - Performs well in low dimensions

Spatial indexing in Oracle

- Based on R-trees. Limited to 2D, 3D,4D.
- Built-in data types for various geometric objects, e.g. `SDO_POINT_TYPE` (3D point).
- Can create point using `SDO_POINT_TYPE(x,y,z)`.
- To create index, first insert ”suitable information” in `user_sdo_geom_metadata`.
- Then create index using e.g.

```
CREATE INDEX spatial_idx ON R(position)
INDEXTYPE IS MDSYS.SPATIAL_INDEX;
```

Spatial query in Oracle

Find tuples where the point position is in the rectangle defined by (5,6),(12,12):

- ```
  SELECT *
  FROM R WHERE
  SDO_INSIDE(R.position,
      SDO_GEOMETRY(2003,
      NULL, NULL,
      SDO_ELEM_INFO_ARRAY(1,1003,3),
      SDO_ORDINATE_ARRAY(5,6,12,12)))
  = 'TRUE';
  ```
  ```

Range trees

- We next consider *range trees*, which provide fast multi-dimensional range queries at the cost of higher space usage.
  – Performance acceptable only in low dimensions.
- In the lecture, we will see a simpler variant that allow range trees to be implemented using a collection of standard B-trees!

Ranges vs prefixes

- Covering ranges by prefixes:
  - Suppose a and b are w-bit integers.
  - Any range [a;b] can be split into at most 2w intervals where each interval consists of all integers with a particular prefix.
- Often the intervals used in ”OLAP” queries naturally correspond to prefixes. E.g.
  - ”location=Denmark”
  - ”location=Denmark:Copenhagen”
  - ”location=Denmark:Copenhagen:Amager”
- Thus: Enough to solve the case where a prefix is specified in each dimension.

Storing points redundantly

- Basic idea:
  - Store each point several times, using all different combinations of prefixes as key.
- Example:
  - p=(DK:CPH:Amgr, Shirts:White).
  - Store according to the 12 keys:

DK:CPH:Amgr; Shirts:White	DK:CPH:Amgr; Shirts	DK:CPH:Amgr; *
DK:CPH; Shirts:White	DK:CPH; Shirts	DK:CPH; *
DK;Shirts:White	DK;Shirts	DK;*
*;Shirts:White	*;Shirts	*;*

Querying

- Prefix querying is very easy:
  Simply use the prefixes as key in some index structure (e.g. a B-tree).
  - Time efficient!
  - But general range queries may require a relatively large number of prefix queries.
- Space analysis:
  - If there are w possible prefixes in each of d dimensions, each point is stored w^d times.
  - Space is factor w^d from optimal. May be fine when d is small.

Problem session

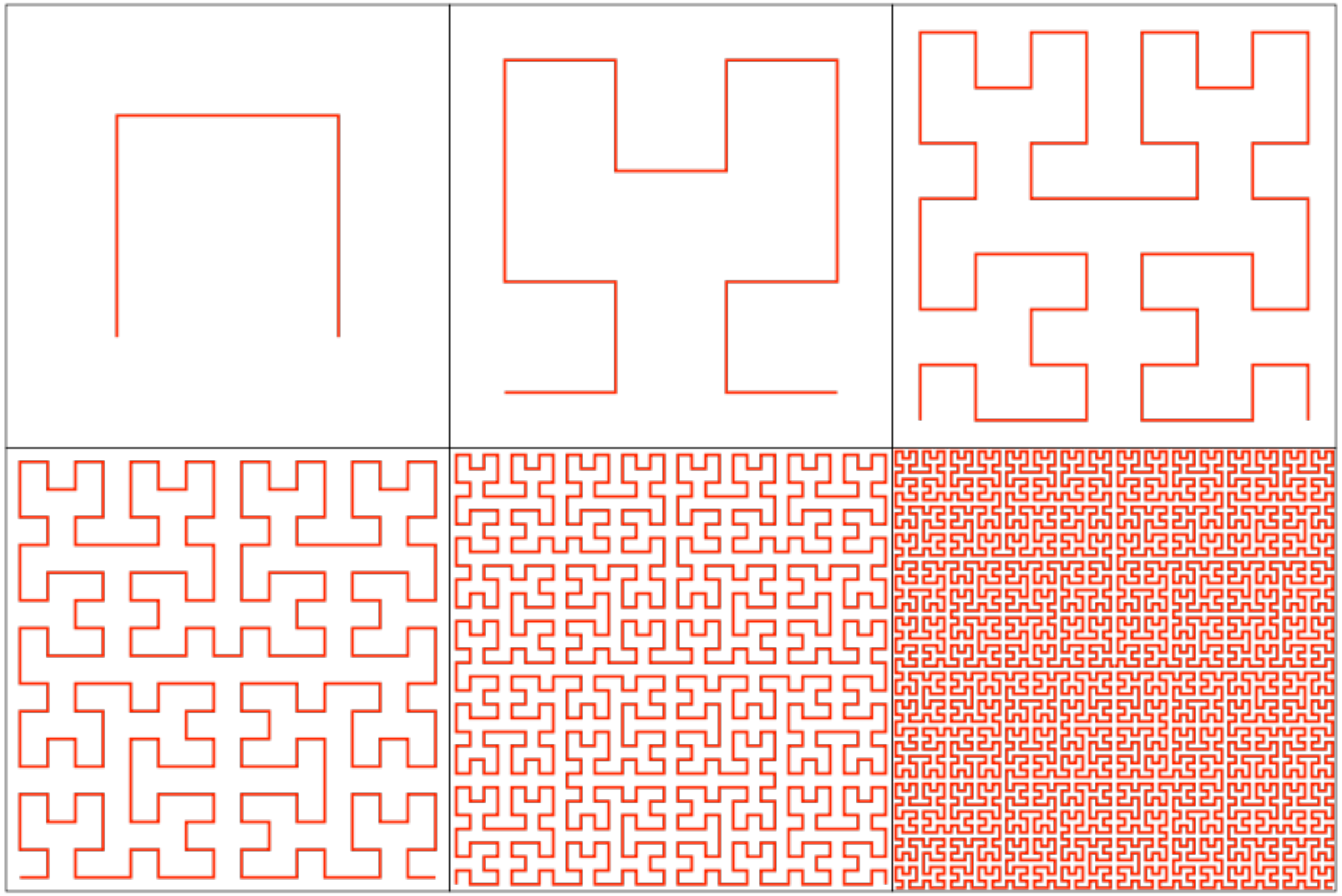
- We revisit the setting from before, where we consider points of the form (Country:City:Site, ItemType:Color).
  - 4 possible location prefixes, 3 item prefixes
  - Basic idea says 12 keys should be used
- Come up with a better way of storing the points:
  - With same query efficiency.
  - Only 3 keys per point
  - **Hint**: Composite keys and range queries.

Range trees wrap-up

- Space overhead may be reduced to w^{d-1} using this idea.
- It is even known how to reduce the space overhead to w^{d-2}, but then the scheme is not external memory efficient.
- Summary:
  - Range trees are mainly applicable where a considerable space overhead is acceptable.
  - Best for prefix queries, but also *reliable* performance for range queries. Especially good in 2D (and 3D).

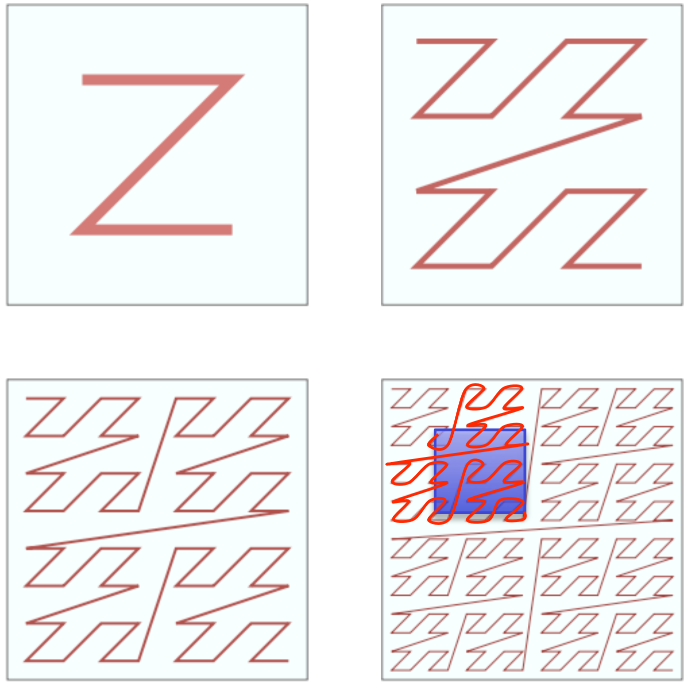
Space-filling curves

Idea: Create 1-to-1 correspondence between points in 2D and 1D that ”preserves locality”.



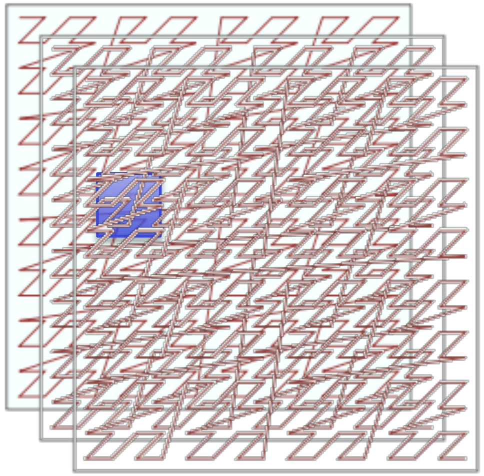
Z-ordering

- Simplest space-filling curve
- Consider point given by binary coordinates: (00101110, 01101011)
- Mapped to the number formed by interleaving: 0001110011101101.
- Mapping a 2D range query: Determine the smallest interval containing range.
  - Z-order: Top-left and bottom right corners determine the extremes.



Weak points of space-filling curves

- *Some* points that are close in 2D will be far apart when mapping to 1D.
- Chance of running into this problem can be minimized by adding a random shift to all coordinates.
  - Alternatively, consider a number of space-filling curves slightly shifted along both coordinates.



Approximate nearest neigbor

- Exact near neighbor queries are difficult, especially
  - when data changes, and
  - there may be many point at almost minimal distance to the query point.
- Often: Enough to find a neighbor that is not much further away than the nearest neighbor.
  - Allows much more efficient solutions.
  - The ratio between distances can be guaranteed.

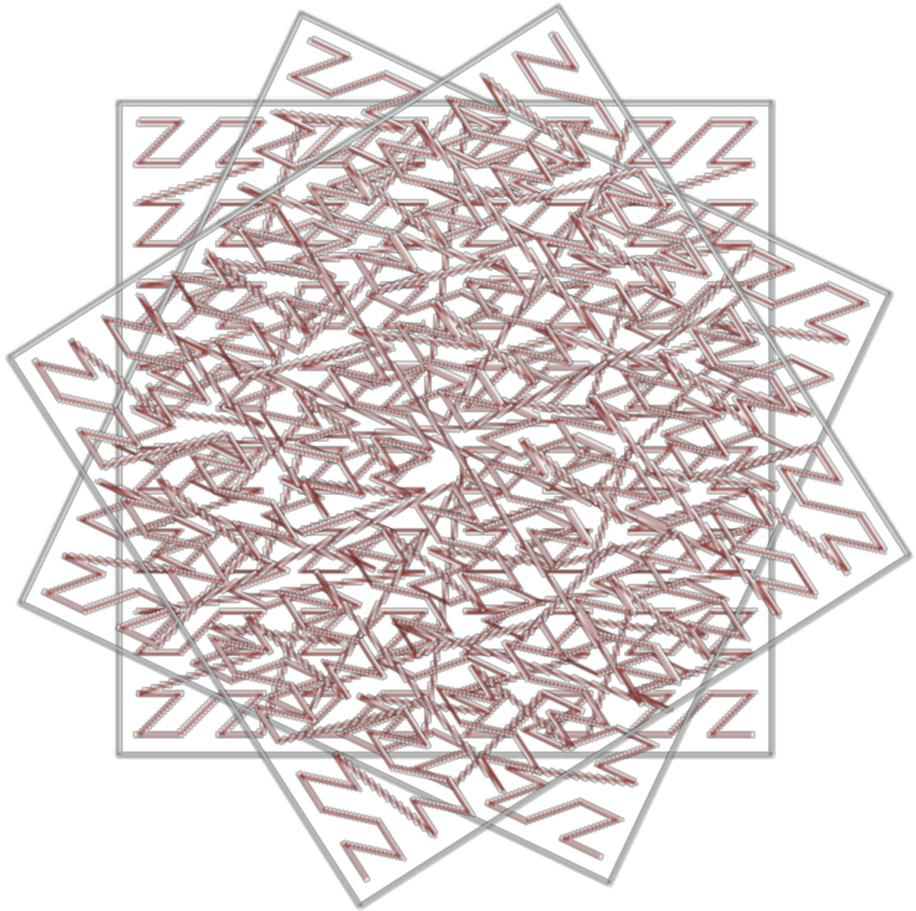
Approximate NN picture

Approximate NN using Z-order

- Points that agree in the most significant bits are close. Vice versa?
- If the coordinates of two points differ by d_1 and d_2 along the two dimensions, we *expect* the least significant $2\log(\max(d_1,d_2))$ bits of the corresponding 1D values to differ.
  - By using several curves, we can make this hold for at least one curve (for any point pair).
  - The largest difference in any dimension is what counts (L_∞ norm).
- Candidates for being near neighbors of a query point p are simply the predecessor and successor of p in the curve order.

Rotations

- To make L_∞ norm close to the normal euclidian distance, we may consider several curves that are rotations of the Z-curve.



Spatial indexing summary

- Many different indexes, with different strengths and weaknesses.
- Distinguishing features include:
  - Linear or super-linear space?
  - Good for any point distribution?
  - Support for queries: Range q.,near neighbor q., stabbing q., intersection q.,…?
  - Exact or approximate results?
  - Fast updates, or meant for static use?
- Most common in practice: R-trees, kd/quad-trees, (space-filling curves).

Exercise

- Hand-out: ”R-trees for triangles”
  - We will go through question a).
  - Question b) is more challenging, and may be fun for you to think about later.