CS 164 & CS 266: Computational Geometry

Lecture 15

Range counting, kD-trees, and quadtrees

David Eppstein
University of California, Irvine

Fall Quarter, 2023

Range counting and range search

Range query data structures

Input: a set of points

Example: a database of people listing them by age and weight, with these numbers interpreted as coordinates

Preprocessing stage: Build a data structure representing them

Preferably linear or near-linear space

Query stage: Handle queries that ask for a list of all points in some shape, or that ask for aggregate information about those points

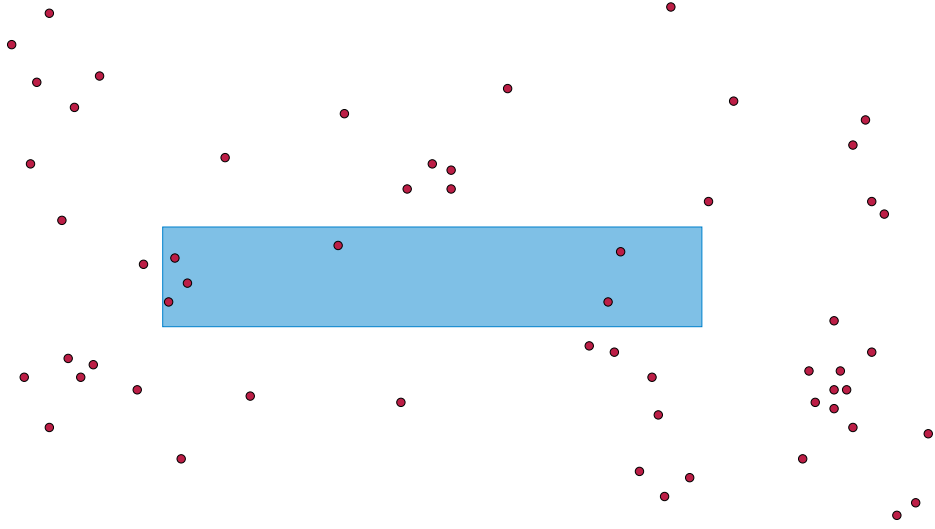
Example: How many points are in the shape?

Goal: query time $\ll$ time to re-scan whole set

The shapes used for the queries are called ranges

Example: Rectangle range counting

Query: How many points are inside a given rectangle?



Scanning all data per query would give $O(n)$ time

Warmup: 1d range searching

Data structure for n points on a line, to count points in an interval

Use a sorted array

Answer queries by binary search for interval endpoints then subtract their positions

Preprocessing time $O(n \log n)$ by sorting

Space $O(n)$ to store sorted array

Query time $O(\log n)$

KD-trees

What is a kD-tree?

The name stands for k-dimensional

but we'll mainly be covering only $k = 2$ (still called kD-tree)

Each node of tree is a (possibly infinite) rectangle)

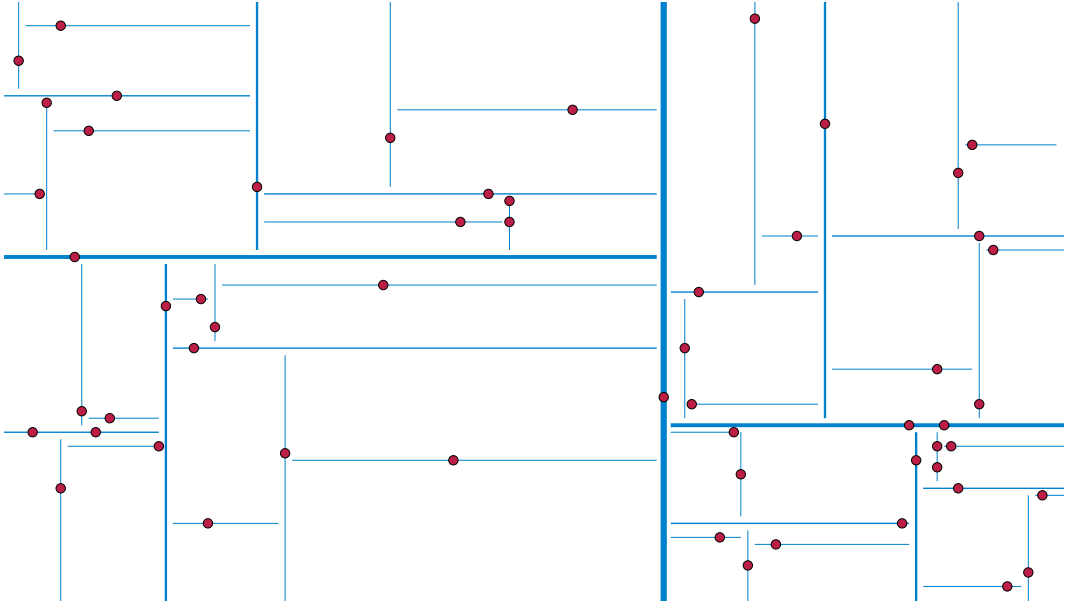
Root = whole plane

Children = smaller rectangles, split by vertical or horizontal line

Alternate vertical/horizontal, split at median data point

Stop splitting at empty rectangles

Example



Representation

Each rectangle = an object

Instance variables:

- Whether split line is vertical or horizontal
- Coordinate of split line (x for vertical, y for horizontal)
- Pointers to two child rectangles, if they are nonempty
- Array of points on split line, sorted by other coordinate
- Aggregate information about all points in the rectangle (for instance, how many points are in the rectangle)

Preprocessing (construction)

Initialize an object for the root rectangle (the whole plane), split vertically, with a list of all its points, not yet sorted

Each time we create an object for a rectangle and a list of points:

- Find the median point for the split direction
- Partition the points into the subsets whose coordinate is smaller than median, equal to median, or greater than median
- Recursively create child rectangles for the smaller and larger subsets
- Sort the points equal to the median and store them in a sorted array

Time for everything except the sorting: $T(n) = O(n) + 2T(n/2) = O(n \log n)$

Time for sorting: $\sum O(n_i \log n_i) = O(n \log n)$

Alternative method for preprocessing

Sort all the points twice:
Once by their x-coordinate and once by their y-coordinate

When we create each child rectangle,
pass in its two sorted lists of its points

To find the median coordinate, use one of the sorted lists

When partitioning the points into subsets,
preserve their sorted order
so they don't need to be sorted again

Still $O(n \log n)$ total

To handle range counting queries

Define rectangle by L, R, T, B: left, right, top, bottom coords

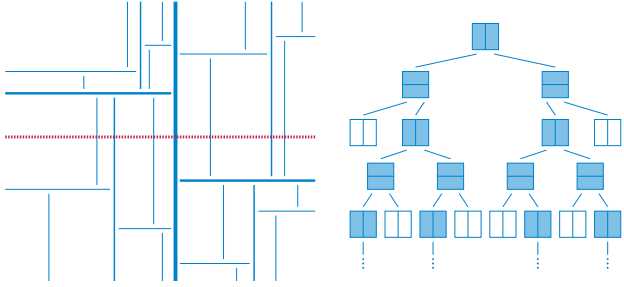
Within recursion, can change to $\pm\infty$: boundary no longer relevant

Define query(node, L, R, T, B):

- If $L = B = -\infty$ and $R = T = +\infty$:
  - Return number of points stored for current node
- Else if it's an x-splitting node
  - If split coordinate $< L$:
    - Recurse into right child
  - Else if split coordinate $> R$:
    - Recurse into left child
  - Else
    - Recurse left with $R = +\infty$
    - Recurse right with $L = -\infty$
    - Add results to # points on split line
- Else handle symmetric cases for y-splitting node

Key lemma for analysis

Every horizontal or vertical line in the plane (including the sides of any query rectangle) is crossed by $O(\sqrt{n})$ split lines



Proof idea: Only splits perpendicular to the line can cross it

The whole tree has height $\log_2 n$, but the number of rectangles crossed by the line is doubled only at alternating levels

If we double $\frac{1}{2} \log n$ times, we get $2^{(log_2 n)/2} = \sqrt{n}$

Query analysis

Because of the replacement of range boundaries by $\pm\infty$, we never recurse into children of rectangles that are entirely covered by the query range

For each rectangle that we recurse into, its parent crosses the range boundary or entirely covers the whole range

Only $O(\log n)$ rectangles can entirely cover the whole range, because they form a path in the tree down from the root to the first one that doesn't

Only $O(\sqrt{n})$ rectangles can cross the range boundary by the lemma

Range counting: $O(\sqrt{n})$; range reporting: $O(\sqrt{n} + k)$

Generalizes to $n^{(d-1)/d} + k$ in d dimensions

Quadtrees

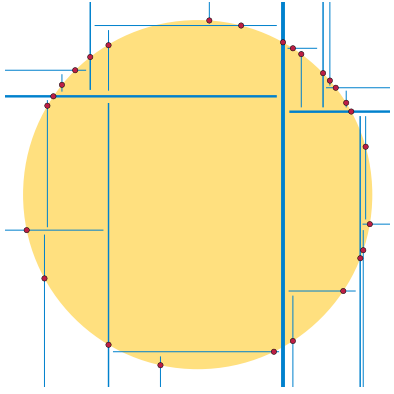
Problem with kD-trees

They can be bad for ranges that are not axis-parallel rectangles

Range boundary can cross all of the k-D tree nodes

Gives query time $\Theta(n)$ even when output size is $O(1)$

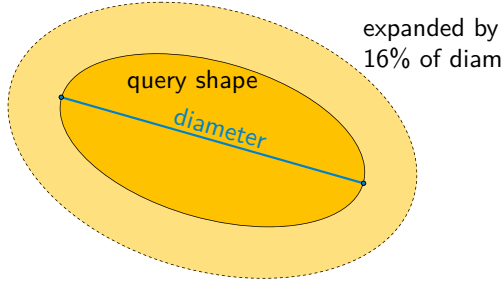
Part of the problem: many long thin rectangles



Two ideas for improvement

Approximate ranges: when we do a query, the result

- Counts or includes all points inside query range
- Does not include points farther than $\varepsilon \times$ diameter from range
- Might or might not include points between range and $\varepsilon \times$ diam

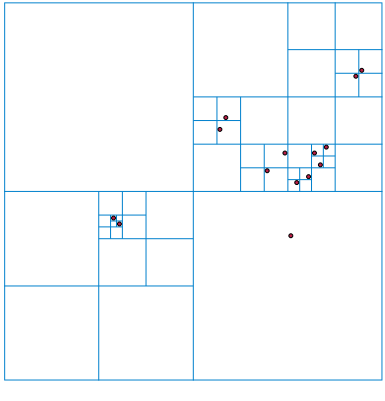


Also, use a different recursive partition with better shapes

Quadtree

(More precisely, "point quadtree"; there are other kinds)

Recursively divide squares into four smaller squares



Simplifying assumptions:

- Point coordinates are integers in range $0 \ldots 2^b - 1$ for some b
- Squares have side lengths 2^k for $0 \leq k \leq b$
- Coordinates of square sides are integer $+ \frac{1}{2}$ so points avoid square sides

Representation and construction

Each square stores:

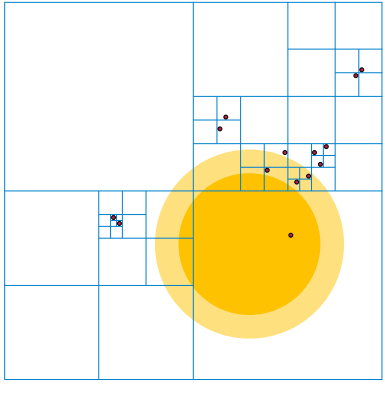
- Its square
- Whether it is empty, has one point, or has multiple points
- If one point, what is that point?
- If multiple points, four child squares
- Aggregate info for points (for instance how many)

Start with a big power-of-two-size square containing all of the points, and a list of all its points

Test whether list is empty, one point, or more than one

If more than one, partition points into four quadrants and recursively construct four child squares

Approximate range counting



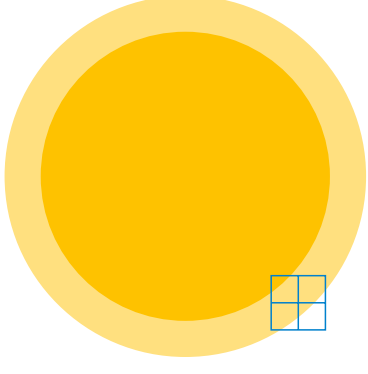
To query a quadtree square for given approximate range:

- If square is empty or avoids inner range, return zero
- If square is inside outer range, return its # points
- If square has one point, check it against range
- Otherwise, recurse into children and sum results

Query analysis lemma

Say that a square crosses the range when it contains parts of both inner and outer range boundaries, but none of its children do

Then $O(1/\varepsilon)$ quadtree squares cross the range



Reason: Each square has side length at least $d/\sqrt{2}$, where $d = \text{diam} \cdot \varepsilon$ is the distance between inner and outer boundaries (otherwise it would be too small to reach both boundaries)

Shell between inner and outer has area $O(\text{diam}^2\varepsilon)$; this square covers area $\Omega(\text{diam}^2\varepsilon^2)$ of the shell

Squares are disjoint, and each has $\Omega(\varepsilon)$ fraction of whole area, so # squares is $O(1/\varepsilon)$

Query analysis

Query time is dominated by two terms:

- How many steps to get from the root to the crossing squares
- How many crossing squares

First term covers time in recursion above level of crossing squares

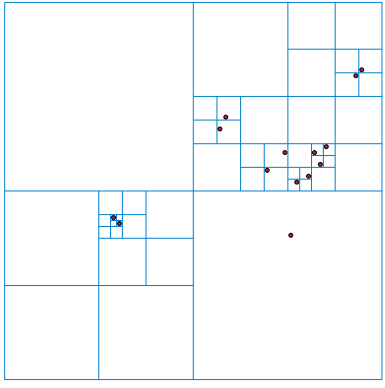
As soon as recursion reaches a child of a crossing square, it stops

Time for recursion $O(b + 1/\varepsilon)$
where b is number of bits in integer coordinates of given points

Generalizes to $b + 1/\varepsilon^{d-1}$ in d dimensions

Two improvements to quadtree

Problem: Too much space



Construction can perform many levels of recursion without splitting anything

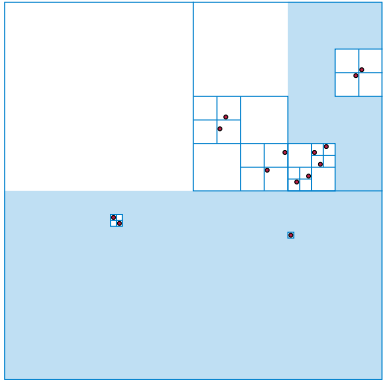
(like on lower left of example)

So space is not linear!

It is $O(nb)$

where b = numerical precision

Solution: Compressed quadtree



When constructing a quadtree node, shrink its square to smallest power-of-two square containing its points

Every non-leaf square has more than one non-empty child

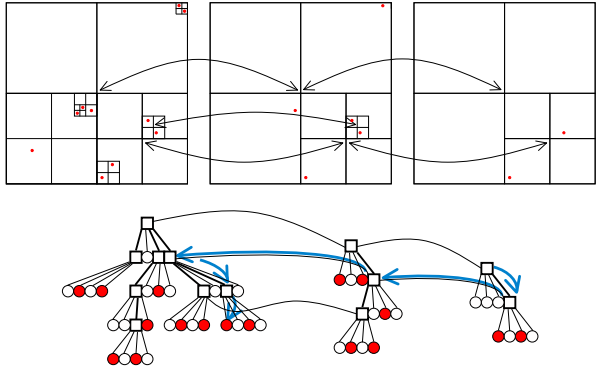
Total # squares is $O(n)$

Also improves time to recursively list all points within a square to $O(\#\text{ points})$, needed for fast range reporting queries

Construction time $O(n \log n)$, from mesh generation lecture

Problem: Time depending on numerical precision

Solution: "Skip quadtree". Randomly sample half the points, build recursive structure on sample, and link its compressed quadtree squares to same squares of compressed quadtree of larger point set [Eppstein et al. 2008]



Start query in quadtree for smallest sample; when reaching a leaf in a sampled quadtree, step to same square in next larger sample

Query time $O(\log n + 1/\varepsilon)$ or $O(\log n + 1/\varepsilon + k)$

References

David Eppstein, Michael T. Goodrich, and Jonathan Zheng Sun. Skip quadtrees: dynamic data structures for multidimensional data. *Int. J. Comput. Geom. Appl.*, 18 (1–2):131–160, 2008. doi: 10.1142/S0218195908002568.