The Skip Quadtree:
A Simple Dynamic Data Structure
For Multidimensional Data

David Eppstein, Michael T. Goodrich, and Jonathan Z. Sun

Univ. of California, Irvine
Donald Bren School of Information and Computer Sciences
The Problem:

Organize a set of many low-dimensional input points

Handle (approximate) range listing queries, nearest neighbor queries, etc

Typical solutions:

Recursively subdivide space into a hierarchy of nested convex cells at each level, split cells by lines into smaller cells until all leaf cells have at most one point each

Handle queries by top-down search: if current cell is out of range, backtrack else recursively search its children

But how to choose splits?
Quadtree

All cells are squares

To subdivide:

split into four equal squares

Problems:

Superlogarithmic depth

Superlinear size

No guaranteed query time
(recursion too deep)
Compressed Quadtree

Keep only interesting squares from quadtree

Square is interesting if root or has >1 nonempty child

Problems:

Superlogarithmic depth

No guaranteed query time (recursion too deep)

Unclear how to dynamize
kD-tree

All cells are rectangles

To subdivide:

split at median coordinate alternating horizontal and vertical

Problems:

High aspect ratio cells

No guaranteed query time (too many cells in range)

Dynamization is amortized (with approx median splits)
BAR-tree

All cells have ≤ 6 sides horizontal, vertical, slope 1

Bounded aspect ratio guaranteed

To subdivide:

split at median point
choose best of 3 split slopes

Problems:

Complex implementation

Dynamization is amortized (with approx median splits)
Skip Quadtree

Key idea:

Impose extra sampling hierarchy (analogous to skiplist) on top of compressed quadtree

Keeps the advantages as compressed quadtree...

Simple structure

Well shaped cells

...but allows logarithmic-time searches and updates

Basic version is randomized

Time bounds are high probability and expected

But deterministic also possible (with same time bounds)
New Results

\(O(\log n)\) time:

Insert or delete a point from input set

Locate query point in compressed quadtree

\(O(\varepsilon^{1-d} + \log n)\) time:

(1+\(\varepsilon\))-approximate fat range query

Approximation to range is decomposed into
\(O(\varepsilon^{1-d})\) compressed quadtree cells

\(O(\varepsilon^{1-d}(\log n + \log 1/\varepsilon))\) time

(1+\(\varepsilon\))-approximate nearest neighbor query

(like spherical range query with unknown radius)
The skip quadtree

Assign a non-negative integer level to each input point
probability $2^{1-i}$ of being assigned level $i$

For each $i$, build a compressed quadtree $Q_i$ of points with levels $\leq i$

Each interesting square stores seven pointers:

- next larger interesting square in $Q_i$ (if not root)
- four children (smaller squares or solitary points)
- same square in $Q_{i-1}$ (always exists unless $i = 0$)
- same square in $Q_{i+1}$ (if it exists)
The skip quadtree, visually
To locate a query point in a skip quadtree:

Start at the last nonempty level

Repeat:

if current square has a child containing query, move to it
else move to same square in next lower level

until finding smallest square containing query point in \(Q_0\)

In expectation, \(O(1)\) steps within each level
so \(O(\log n)\) steps overall
To insert a new point into a skip quadtree:

- Assign a level to the point
- Locate the point (finds smallest interesting square containing it in all levels)
- Perform $O(1)$ local changes in each modified level

To delete a point from a skip quadtree:

- Same as insertion in reverse
To perform range queries:

Simulate standard subdivision-data-structure search in $Q_0$:
repeatedly replace squares by children intersecting range until remaining squares approximately cover the range

Problem:
long chain of replacements of square by one child

Instead, use skip structure to find descendant at end of chain like point location, $O(\log n)$ time using skip structure

To perform nearest neighbor queries:

Similar to range query

Use priority queue to keep track of which square to expand
Conclusions

New data structure combines quadtree and skiplist

All advantages of similar subdivision-based structures:

- easy to implement
- fast updates and queries
- well shaped cells
- generalizes to arbitrary dimension

Future work

Distributed version (to appear at PODC)