
Z-ordering

Christos Faloutsos

www.cs.cmu.edu/~christos

Roadmap

1) Roots: System R and Ingres
2) Implementation: buffering, indexing, q-opt
3) Transactions: locking, recovery
4) Distributed DBMSs
5) Parallel DBMSs: Gamma, Alphasort
6) OO/ODBMS
7) Data Analysis - data mining
8) Benchmarks
9) vision statements
   extras (streams/sensors, graphs, multimedia, web, fractals)

Detailed roadmap

1) Roots: System R and Ingres
2) Implementation: buffering, indexing, q-opt
   - OS support for DBMS
     - R-trees and GiST
     - Z-ordering
       - Buffering
         - ...
   3) Transactions: locking, recovery

Reference


z-ordering - Detailed outline

- What is the problem / S.A.M.
- z-ordering
  - main idea - 3 methods
  - use w/ B-trees; algorithms (range, knn queries ...)
  - non-point (eg., region) data
  - analysis; variations

Spatial Access Methods - problem

- Given a collection of geometric objects (points, lines, polygons, ...)
- organize them on disk, to answer spatial queries (like??)
Spatial Access Methods - problem

- Given a collection of geometric objects (points, lines, polygons, ...)
- organize them on disk, to answer
  - point queries
  - range queries
  - k-nn queries
  - spatial joins (‘all pairs’ queries)

SAMs - motivation

- Q: applications?

SAMs: solutions

- z-ordering
- R-trees
- (grid files)

Q: how would you organize, e.g., n-dim points, on disk? (C points per disk page)

z-ordering - Detailed outline

- What is the problem / S.A.M.
- z-ordering
  - main idea - 3 methods
  - use w/ B-trees; algorithms (range, knn queries ...)
  - non-point (eg., region) data
  - analysis; variations

z-ordering

Q: how would you organize, e.g., n-dim points, on disk? (C points per disk page)

Hint: reduce the problem to 1-d points(!)

Q1: why?
A:
Q2: how?
z-ordering

Q: how would you organize, e.g., \( n \)-dim points, on disk? (C points per disk page)
H: hint: reduce the problem to 1-d points (!!)
Q1: why?
A: B-trees!
Q2: how?

Q2: how?
A: assume finite granularity; z-ordering = bit-shuffling = N-trees = Morton keys = geo-coding = ...

Q2.1: how to map \( n \)-d cells to 1-d cells?
A: row-wise
Q: is it good?
A: great for ‘x’ axis; bad for ‘y’ axis

Q: is it good?
A: great for ‘x’ axis; bad for ‘y’ axis
z-ordering

Q: How about the ‘snake’ curve?

A: still problems:

\[2^{32}\]

z-ordering

Q: Why are those curves ‘bad’?

A: no distance preservation (~ clustering)

Q: solution?

z-ordering/bit-shuffling/linear-quadtrees

· few long jumps;
· scoops out the whole quadrant before leaving it
· a.k.a. space filling curves

Q: solution? (w/ good clustering, and easy to compute, for 2-d and n-d?)

A: z-ordering/bit-shuffling/linear-quadtrees

z-ordering/bit-shuffling/linear-quadtrees

Q: How to generate this curve \(z = f(x,y)\) ?

A: 3 (equivalent) answers!
z-ordering

z-ordering/bit-shuffling/linear-quadtrees
Q: How to generate this curve \((z = f(x,y))\)?
A1: ‘z’ (or ‘N’) shapes, RECURSIVELY

\[
\begin{array}{c}
\text{order-1} \\
\text{order-2} \\
\vdots \\
\text{order \((n+1)\)}
\end{array}
\]

z-ordering

z-ordering/bit-shuffling/linear-quadtrees
Q: How to generate this curve \((z = f(x,y))\)?
A: 3 (equivalent) answers!

Method #2?

z-ordering

z-ordering/bit-shuffling/linear-quadtrees
Q: How to generate this curve \((z = f(x,y))\)?
A: 3 (equivalent) answers!

z-ordering

z-ordering/bit-shuffling/linear-quadtrees
Q: How to generate this curve \((z = f(x,y))\)?
A: 3 (equivalent) answers!

How about the reverse:

\[
\begin{array}{c}
\text{order-1} \\
\text{order-2} \\
\vdots \\
\text{order \((n+1)\)}
\end{array}
\]

bit-shuffling

\[
\begin{array}{c}
x \\
y \\
z = (0\ 1\ 0\ 1)_2 = 5
\end{array}
\]

How about the reverse:

\[
\begin{array}{c}
x \\
y \\
z = (0\ 1\ 0\ 1)_2 = 5
\end{array}
\]

How about \(n\)-d spaces?
**z-ordering**

*z-ordering/bit-shuffling/linear-quadtrees*

Q: How to generate this curve \( z = f(x,y) \)?

A: 3 (equivalent) answers!

Method #1:

\[
\begin{array}{ccc}
1 & 0 & 0 \\
0 & 0 & 1 \\
0 & 1 & 0 \\
0 & 1 & 1 \\
\end{array}
\]

Method #2:

\[
\text{shuffle(11;10)} = 1
\]

Method #3:

\[
\text{assign } N \rightarrow 1, S \rightarrow 0 \text{ etc.}
\]

---

**z-ordering**

*linear-quadtrees*: assign \( N \rightarrow 1, S \rightarrow 0 \) e.t.c.

Drill: z-value of magenta cell, with the three methods?

\[
\begin{array}{ccc}
1 & 0 & 0 \\
0 & 0 & 1 \\
0 & 1 & 0 \\
0 & 1 & 1 \\
\end{array}
\]

- method\#1: 14
- method\#2: shuffle(11;10) = (1110)\(_2\) = 14
- method\#3: EN;ES = ...) = 14
z-ordering - Detailed outline

- z-ordering
  - main idea - 3 methods
  - use w/ B-trees; algorithms (range, knn queries ...)
  - non-point (eg., region) data
  - analysis; variations

z-ordering - usage & algo’s

Q1: How to store on disk?
A: treat z-value as primary key; feed to B-tree

Q2: queries? (eg.: find city at (0,3) )?
A: find z-value; search B-tree

z-ordering - usage & algo’s

MAJOR ADVANTAGES w/ B-tree:
- already inside commercial systems (no coding/debugging!)
- concurrency & recovery is ready

z-ordering - usage & algo’s

Q2: queries? (eg.: find city at (0,3) )?
z-ordering - usage & algo's

Q2: range queries?

A: compute ranges of z-values; use B-tree

Q2': range queries - how to break a query

A: recursively, quadtree-style; decompose

Q2'': range queries - how to reduce # of qualifying of ranges?

A: Augment the query!
Q2”: range queries - how to break a query into ranges?
A: recursively, quadtree-style; decompose only non-full quadrants

- z-ordering
  - main idea - 3 methods
  - use w/ B-trees; algorithms (range, knn queries ...)
  - non-point (eg., region) data
  - analysis; variations

Q3: k-nn queries? (say, 1-nn)?
A: traverse B-tree; find nn wrt z-values and ...

... ask a range query.

... ask a range query.
z-ordering - usage & algo's

Q4: all-pairs queries? (all pairs of cities within 10 miles from each other?)

PGH

SF

(we’ll see ‘spatial joins’ later: find all PA counties that intersect a lake)

z-ordering - Detailed outline

• z-ordering
  – main idea - 3 methods
  – use w/ B-trees; algorithms (range, knn queries ...)
  – non-point (eg., region) data
  – analysis; variations

z-ordering - regions

Q: z-value for a region?

A

B

z_B = ??

z_C = ??

z-ordering - regions

Q: z-value for a region?

A

B

z_B = ??

z_C = ??

z-ordering - regions

Q: z-value for a region?

z_B = 11**

z_C = ??

z-ordering - regions

Q: z-value for a region?

z_B = 11**

z_C = {0010; 1000}

“don’t care”
z-ordering - regions

Q: How to store in B-tree?
Q: How to search (range etc queries)

A

B

z-ordering - regions

Q: How to store in B-tree? A: sort (*<0<1)
Q: How to search (range etc queries)

A

B

<table>
<thead>
<tr>
<th>z</th>
<th>obj-id</th>
<th>etc</th>
</tr>
</thead>
<tbody>
<tr>
<td>0010</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>0101</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>11**</td>
<td>B</td>
<td></td>
</tr>
</tbody>
</table>

z-ordering - regions

Q: How to search (range etc queries) - eg  'red' range query

A

B

z-ordering - regions

Q: How to search (range etc queries) - eg  'red' range query
A: break query in z-values; check B-tree

A

B

C

z-ordering - regions

Almost identical to range queries for point data, except for the “don’t cares” - i.e.,

A

B

z-ordering - regions

Almost identical to range queries for point data, except for the “don’t cares” - i.e.,

\[ z1 = 1100 \] ?? \[ 11** = z2 \]

Specifically: does \( z1 \) contain/avoid/intersect \( z2 \)?

Q: what is the criterion to decide?
z-ordering - regions

z1= 1100 ?? 11** = z2
Specifically: does z1 contain/avoid/intersect z2?
Q: what is the criterion to decide?
A: Prefix property: let r1, r2 be the corresponding regions, and let r1 be the smallest (=> z1 has fewest **). Then:

1. if r2 contains r1, then z2 contains z1
2. if r2 contains r1, then z1 contains z2
3. if r2 contains r1, then z1 and z2 are disjoint

z-ordering - regions

Drill (True/False). Given:
- z1= 011001**
- z2= 01******
- z3= 0100****

T/F r2 contains r1
T/F r3 contains r1
T/F r3 contains r2

z-ordering - regions

Drill (True/False). Given:
- z1= 011001**
- z2= 01******
- z3= 0100****

T/F r2 contains r1 - TRUE (prefix property)
T/F r3 contains r1 - FALSE (disjoint)
T/F r3 contains r2 - FALSE (r2 contains r3)
Spatial joins: find (quickly) all counties intersecting lakes

Naive algorithm: \( O(N \times M) \)

Something faster?

Solution: merge the lists of (sorted) z-values, looking for the prefix property

footnote\[#1\]: ‘*’ needs careful treatment
footnote\[#2\]: need dup. elimination

Spatial joins: find (quickly) all counties intersecting lakes

\[
\begin{array}{ccc}
0010 & \text{ALG} \\
\ldots & \ldots \\
1000 & \text{WAS} \\
11** & \text{ALG} \\
0011 & \text{Erie} \\
0101 & \text{Erie} \\
\ldots & \ldots \\
10** & \text{Ont.}
\end{array}
\]

Spatial joins: find (quickly) all counties intersecting lakes

- z-ordering
  - main idea - 3 methods
  - use w/ B-trees; algorithms (range, knn queries ...)
  - non-point (eg., region) data
  - analysis; variations

z-ordering - Detailed outline
z-ordering - variations

Q: is z-ordering the best we can do?

A: probably not - occasional long ‘jumps’

Q: then?

A1: Gray codes

A2: Hilbert curve! (a.k.a. Hilbert-Peano curve)

‘Looks’ better (never long jumps). How to derive it?

order-1

order-2

... order (n+1)
z-ordering - variations

Q: function for the Hilbert curve \( h = f(x, y) \)?
A: bit-shuffling, followed by post-processing, to account for rotations. Linear on # bits.
See textbook, for pointers to code/algorithms (eg., [Jagadish, 90])

Q: how about Hilbert curve in 3-d? n-d?
A: Exists (and is not unique!). Eg., 3-d, order-1 Hilbert curves (Hamiltonian paths on cube)

z-ordering - analysis

Q: How many pieces (‘quad-tree blocks’) per region?
A: proportional to perimeter (surface etc)

(How long is the coastline, say, of England? Paradox: The answer changes with the yardstick -> fractals ...)

z-ordering - analysis

Q: How many pieces (‘quad-tree blocks’) per region?
A: proportional to perimeter (surface etc)
Q: Should we decompose a region to full detail (and store in B-tree)?

A: NO! approximation with 1-3 pieces/z-values is best [Orenstein90]

Q: how to measure the ‘goodness’ of a curve?

A: e.g., avg. # of runs, for range queries

Q: So, is Hilbert really better?

A: 27% fewer runs, for 2-d (similar for 3-d)

Q: are there formulas for #runs, #of quadtree blocks etc?

A: Yes ([Jagadish; Moon+ etc])

Hilbert and z-ordering curves: “space filling curves”: eventually, they visit every point in n-d space - therefore:

order-1

order-2

... order (n+1)
z-ordering - fun observations

... they show that the plane has as many points as a line (→ headaches for 1900’s mathematics/topology). (fractals, again!)

<table>
<thead>
<tr>
<th>order-1</th>
<th>order-2</th>
<th>... order (n+1)</th>
</tr>
</thead>
</table>

z-ordering - fun observations

Observation #2: Hilbert (like) curve for video encoding [Y. Matias*, CRYPTO ‘87]:

Given a frame, visit its pixels in randomized hilbert order; compress; and transmit

z-ordering - fun observations

In general, Hilbert curve is great for preserving distances, clustering, vector quantization etc

z-ordering - Detailed outline

- z-ordering
  - main idea - 3 methods
  - use w/ B-trees; algorithms (range, knn queries ...)
  - non-point (eg., region) data
  - analysis; variations
  - Conclusions

Conclusions

- z-ordering is a great idea (n-d points → 1-d points; feed to B-trees)
- used by TIGER system and (most probably) by other GIS products
- works great with low-dim points