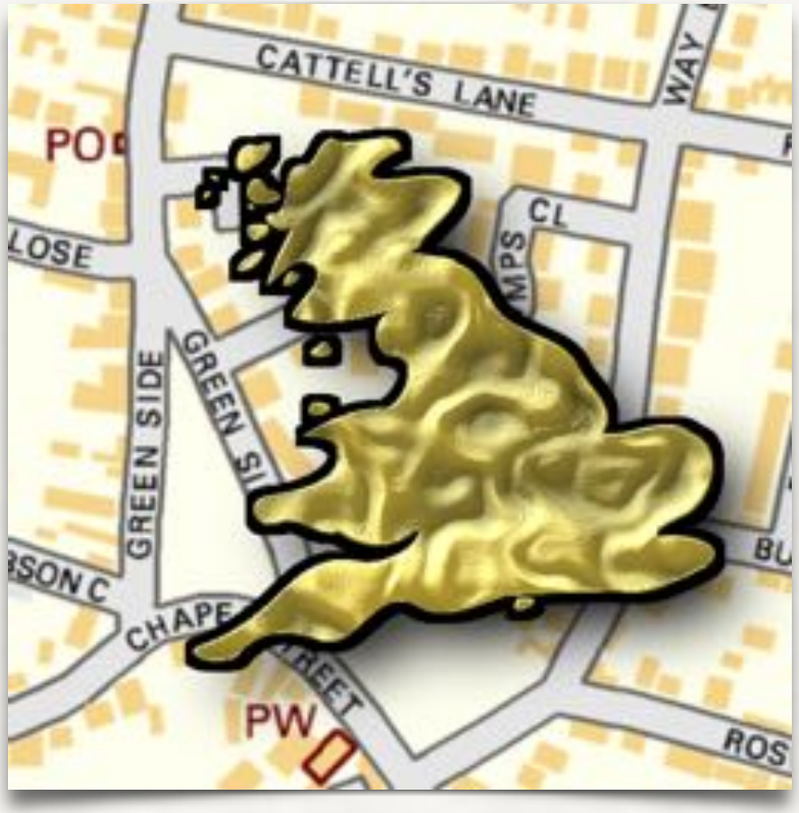
THE Z-CURVE AND STANDARD CONTAINERS

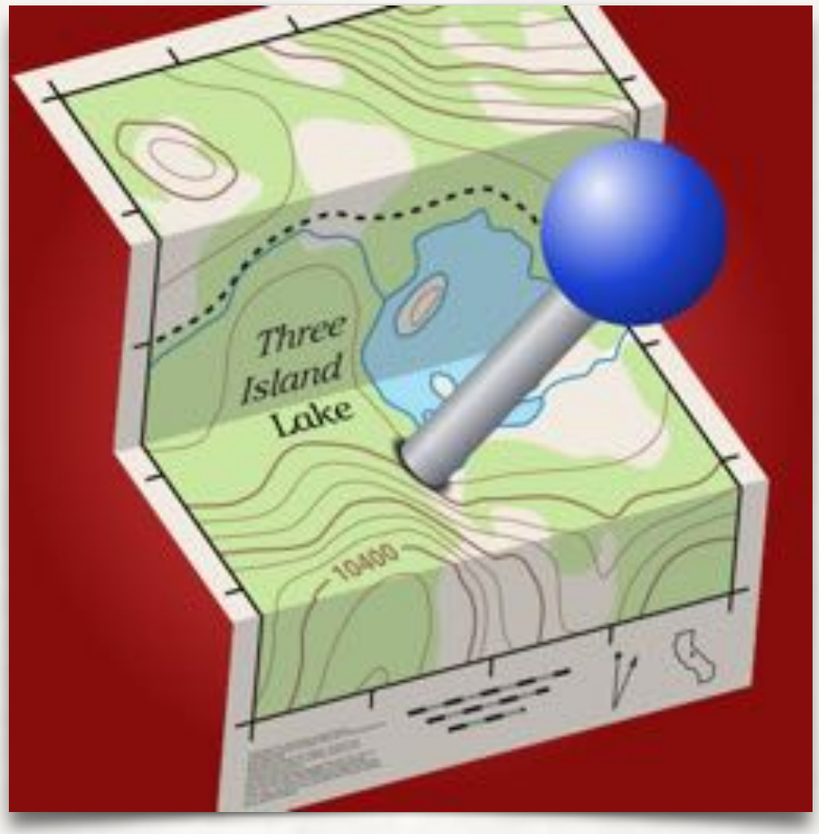
PHIL ENDECOTT

PHIL ENDECOTT

phil@chezphil.org



UK Map App



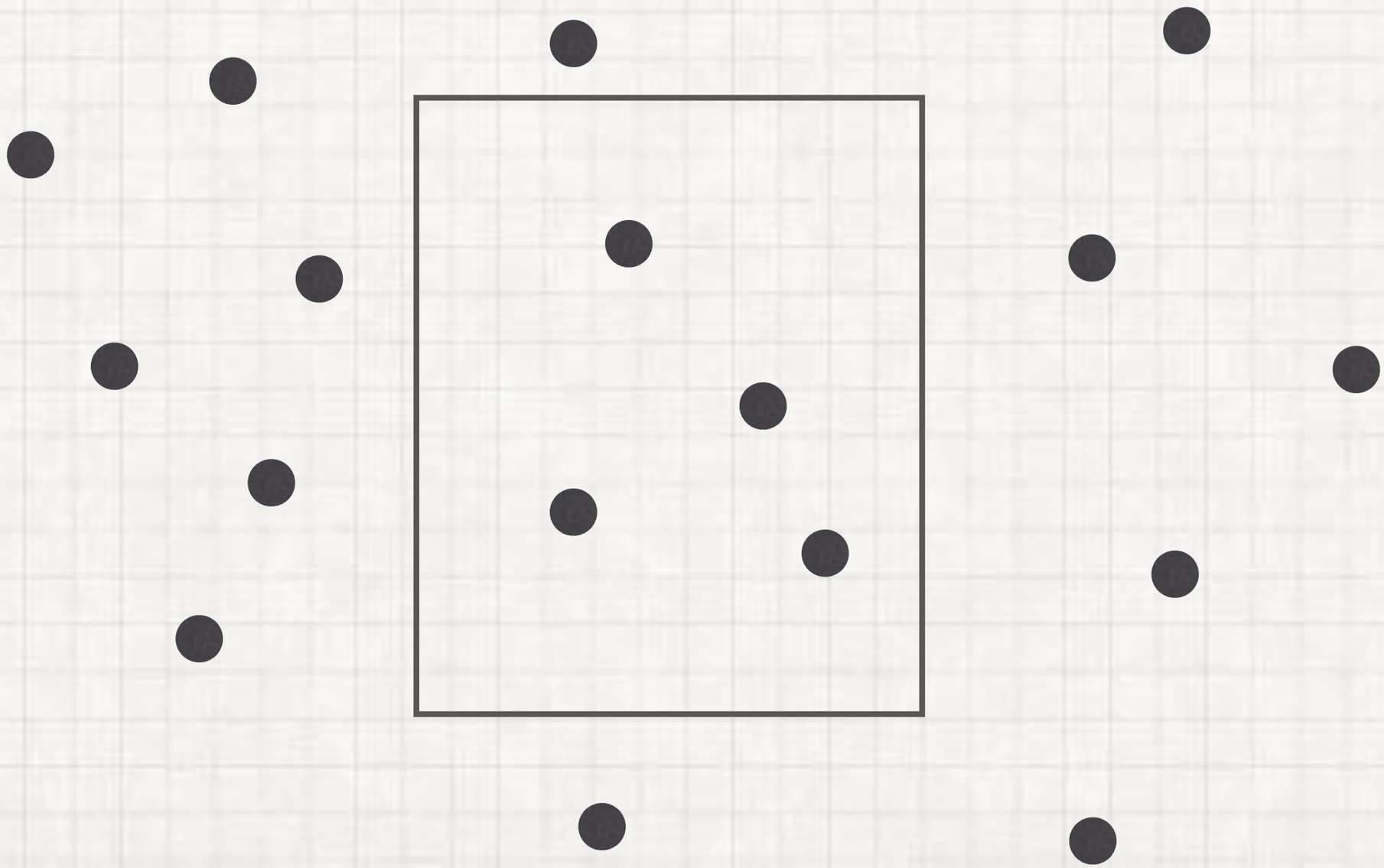
Topo Maps


Ruadh-stac Mòr
Spidean Coire nan Clach (Beinn Eighe) - Sail Mhòr
Spidean Coire nan Clach (Beinn Eighe) - Coinneach Mhòr
Spidean Coire nan Clach (Beinn Eighe) - Coinneach Mhòr (Mid Cliff)
Spidean Coire nan Clach (Beinn Eighe) - Sail Mhòr
Stuc Coire an Laoigh
Stuc a' Choire
2 km


Show Key
Altitude
Distance
1000
800
600
400
200
0
0 km
2 km
4 km
6 km
8 km
10 km
12 km
Distance

MOTIVATING PROBLEM

STORE A SET OF 2D POINTS SUCH THAT WE CAN EFFICIENTLY ITERATE OVER THE CONTENT OF AN AXIS-ALIGNED RECTANGLE.



MOTIVATING PROBLEM

COMPUTATIONAL COMPLEXITY

- If there are N items in the container and M items in the rectangle, the complexity of iterating those M items has:
    - a lower bound of O(M)
    - an upper bound of O(N)

STANDARD CONTAINERS ARE GREAT

- std::vector, std::list, std::set, std::map
- Available everywhere
- Everyone understands them
- Quality implementations
- Well documented
- Have the right computational complexity etc.
- Work with standard algorithms

STANDARD CONTAINERS ARE GREAT

AND OTHER CONTAINERS BORROW THEIR GREAT FEATURES

- boost::flat_set, flat_map
- boost::intrusive
- boost::interprocess
- boost::container::static_vector, small_vector
- Google's in-memory b-tree

BUT....

- Standard associative containers require an ordering predicate, i.e. operator<
- This is inherently one-dimensional
- Most often, multidimensional data is stored in specialised containers

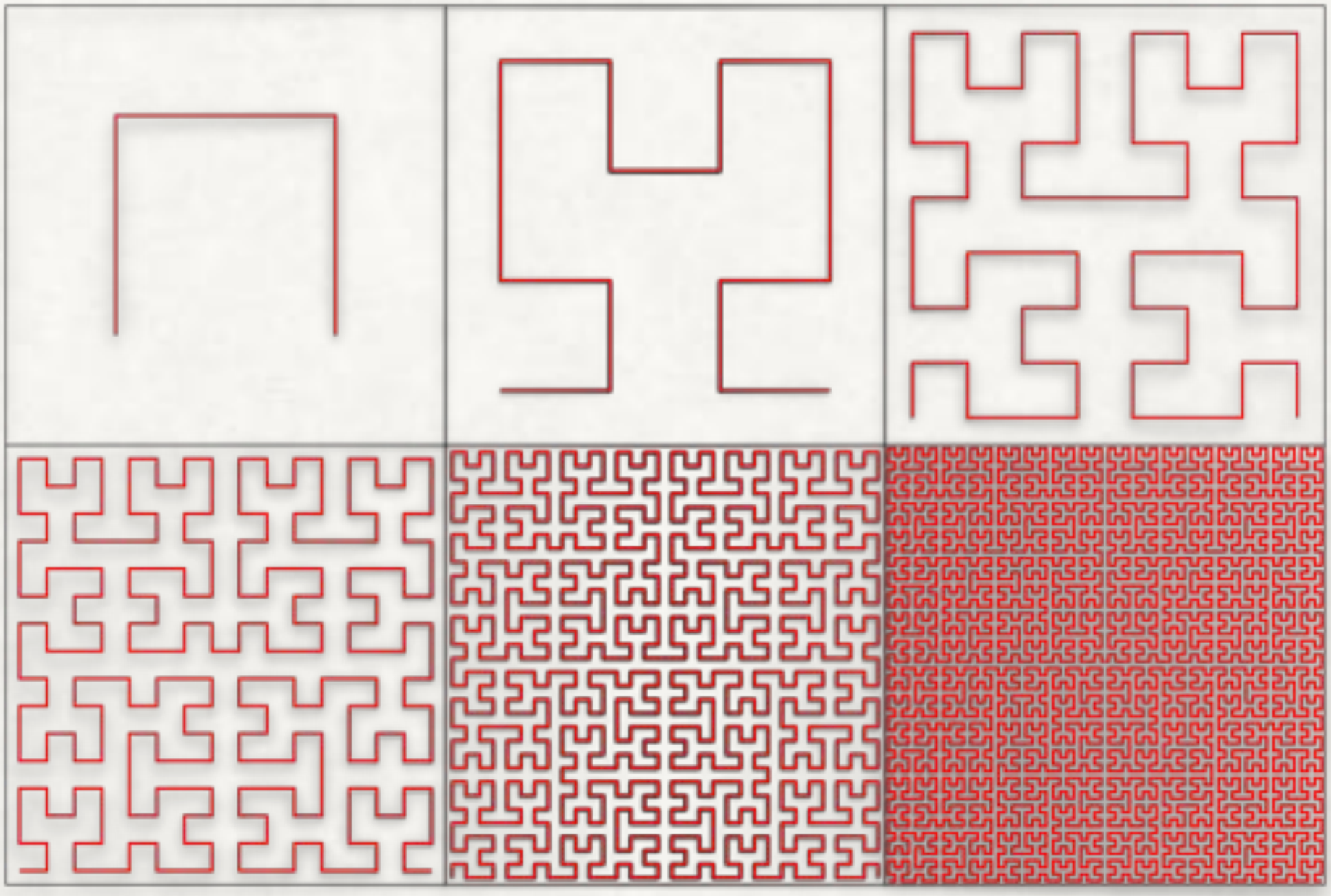
MULTIDIMENSIONAL CONTAINERS

- Few good open-source implementations
- Inherently complex
- Not obvious which data structure to use

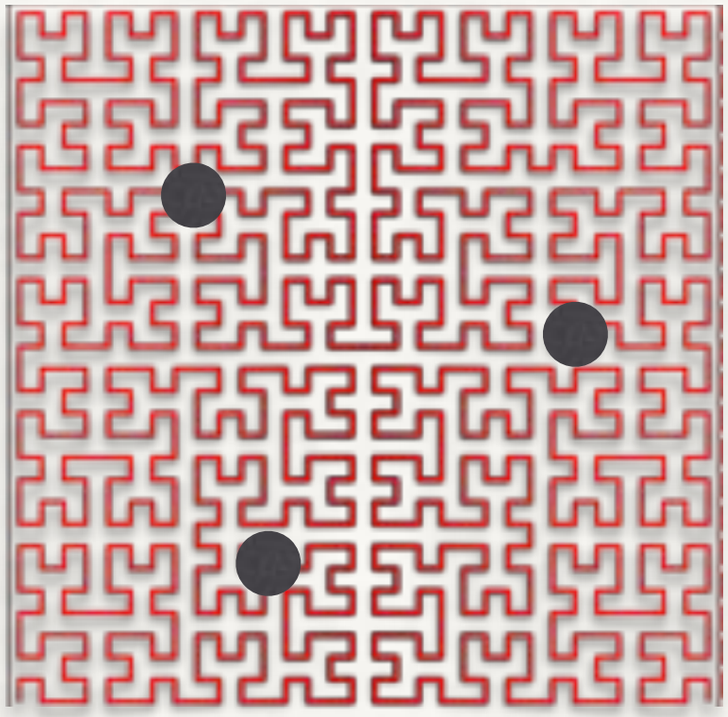
ADAPTERS

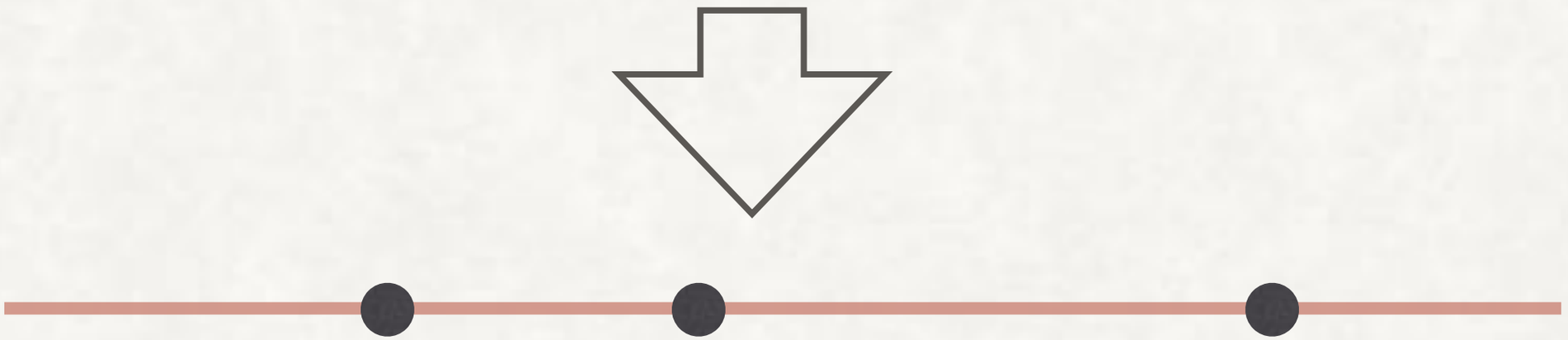
- Can we create an adapter that wraps a 1D associative container so that it stores 2D data?
- **adapt2d< std::map<point,foo> >**
- **adapt2d< boost::flat_map<point,foo> >**
- **adapt2d< boost::intrusive::map<foo> >**

SPACE FILLING CURVES



SPACE FILLING CURVES



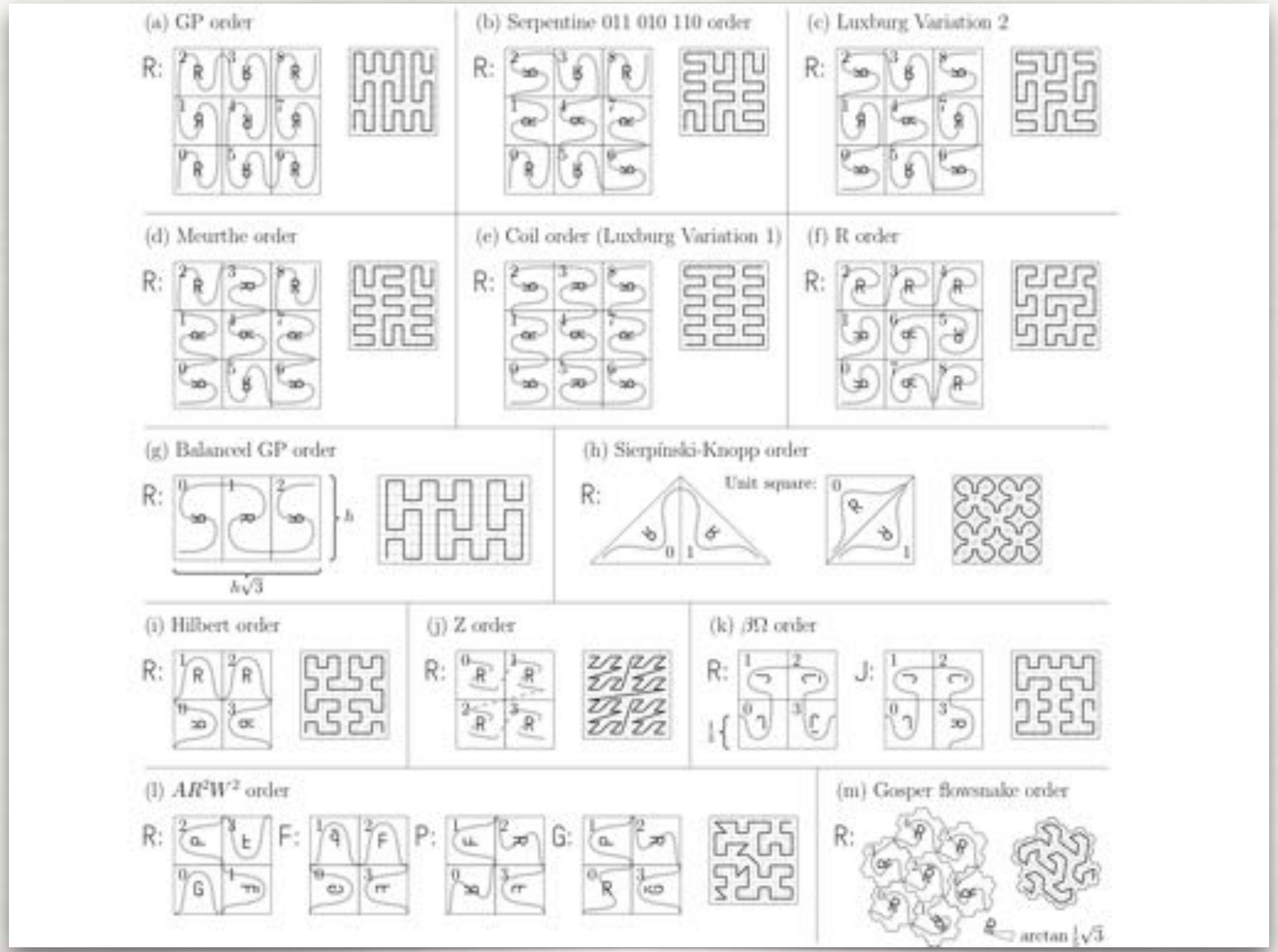


SPACE FILLING CURVES

- Curve is defined by a function that converts (x,y) to a distance along the curve, which is one-dimensional
- (And the inverse function)
- Idea is that we use the distance along the curve with the ordering predicate in a standard 1D container

WHICH CURVE TO USE?

THERE ARE PLENTY TO CHOOSE FROM



WHICH CURVE TO USE?

HERE ARE TWO OF MY FAVOURITES

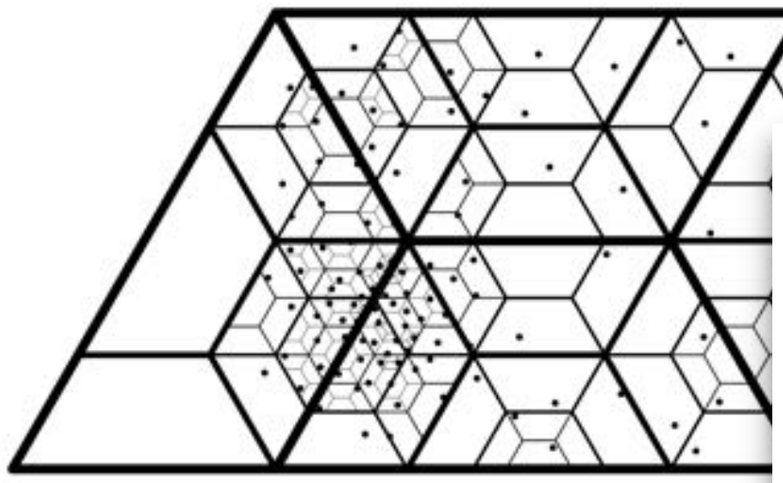


Figure 2: A recursive tiling into trapez

a recursive tiling based on trapezoids, expanded down to the l
at most one data point. We store the data points in such a wa
level of recursion, the data points within that tile are stored a
the disk. Hopefully, if we get it right, we can now cover the reg

Journal of Computational Geometry jocg.org

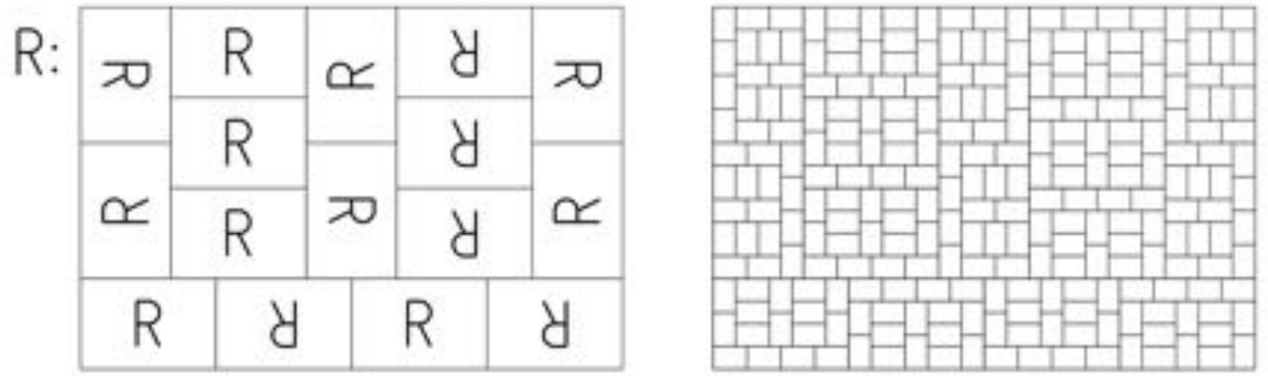


Figure 10: Definition of the Daun tiling, and level-two expansion. Figure 22 at the end of this article shows the level-three expansion.

Proof. We have $\alpha > 1$, otherwise we would get a regular grid of squares with Arrwwid number four. Therefore the solution to Equation 2 for which $n_{wh} > 0$, must have $n_{wh} < \sqrt{t}$ and $\alpha = (\sqrt{t} - n_{hh})/n_{wh}$. □

The above two lemmas brought an exhaustive search for increasing values of t within reach, trying all eligible values of α for each t. This led to the following result:

Theorem 3. *The smallest uniform rectangular tiling (with fewest tiles in the defining*

WHICH CURVE TO USE?

EXPERTS HAVE TRIED TO MEASURE THEIR PROPERTIES

Order	WL_∞	WL_2	WL_1	WBA	ABA	WBP	ABP	WOA	AOA	WOP	AD_∞
Sierpiński-Knopp order	4	4	8	3.000	1.78	3.000	1.42	1.789	1.25	1.629	1.77′
Balanced GP	4.619	4.619	8.619	2.000	1.44	2.155	1.19	1.769	1.31	1.807	1.72′
GP (Serp. 000 000 000)	8	8	$10^2/_3$	2.000	1.44	2.722	1.28	1.835	1.32	2.395	2.13′
Serpentine 011 010 110	5.625	6.250	10.000	2.500	1.44″	2.500	1.20	2.222	1.32′	2.036	1.71′
Luxburg 2 (101 010 101)	$5^5/_8$	$6^1/_4$	10	2.500	1.49′	2.500	1.24	2.222	1.35′	2.036	1.81′
Meurthe (110 110 110)	5.333	5.667	10.667	2.500	1.41″	2.667	1.17	2.000	1.30′	2.018	1.64′
Coil (Serp. 111 111 111)	$6^2/_3$	$6^2/_3$	$10^2/_3$	2.500	1.41′	2.667	1.17	2.222	1.29′	2.424	1.63′
Hilbert	6	6	9	2.400	1.44	2.400	1.19	1.929	1.30	1.955	1.67′
$\beta\Omega$	5.000	5.000	9.000	2.222	1.42	2.250	1.17	1.800	1.29	1.933	1.64′
AR^2W^2	5.400	6.046	12.000	3.055	1.49′	3.125	1.22	2.344	1.33	2.255	1.70′
Z-order	∞	∞	∞	∞	2.92	∞	2.40′	∞	2.46	∞	3.80″
Gosper flowsnake	6.35	6.35	12.70	$\geq$3.18		$\geq$3.18					

Table 1: Bounds for different measures and curves. New curves printed in bold. For the A-measures the standard deviation is indicated behind the number: no symbol when less than 0.5%; one mark when between 0.5% and 1.0%, two marks when between 1.0% and 2.0%.

even better locality in these measures than Luxburg's second variant (Serpentine 101 010 101). Even better locality is achieved by Wierum's $\beta\Omega$-curve (matching or improving on Hilbert's curve in all measures) and still better by our new Peano variant: balanced GP. The latter approaches the locality of the Sierpiński-Knopp order, which is still conjectured to be optimal.

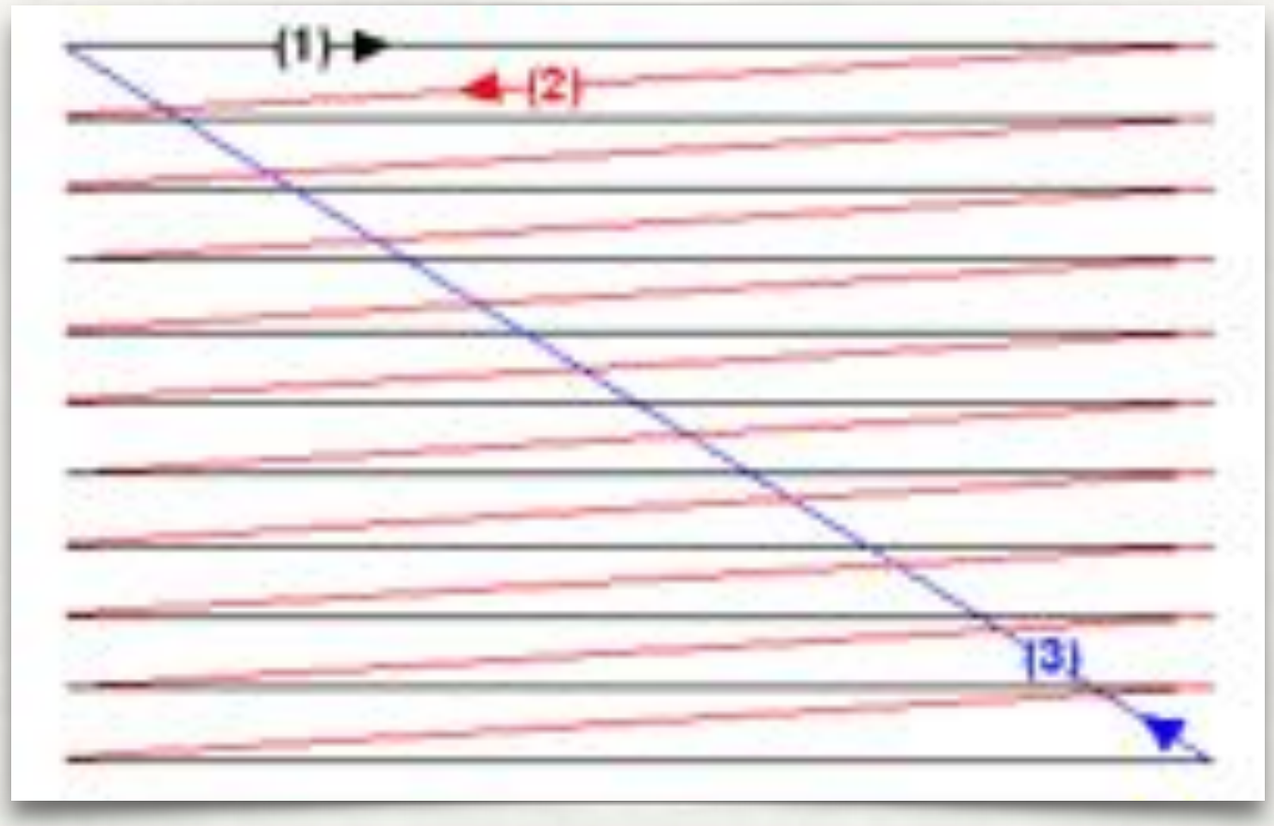
However, it appears that the optimal locality of the Sierpiński-Knopp order comes at a

BUT IN PRACTICE....

- The functions that define those exotic-looking curves, and their inverses, are horribly complex and slow to compute.
- I suppose you might consider using them if lookup were particularly slow, e.g. over the 'net.
- In practice there is only one curve considering.
- (Or maybe two)

ASIDE: RASTER SCAN ORDER

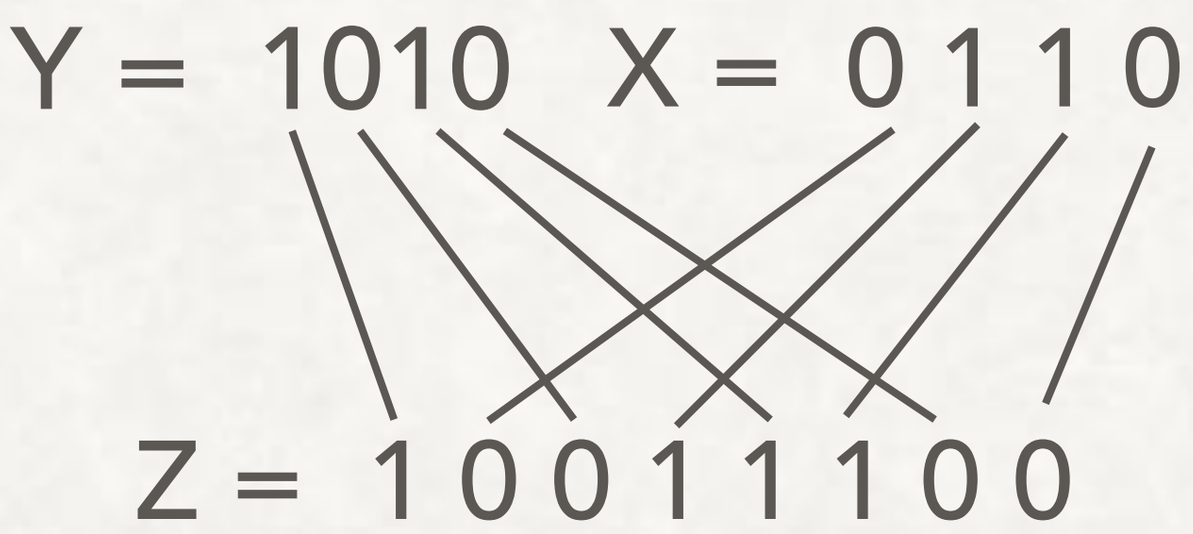
- Is this a space-filling curve?
- It's not fractal
- It's what you get if you store a std::pair in a std::set
- It's still a useful way of ordering data in some cases



THE "Z" OR MORTON CURVE

THE "Z" OR MORTON CURVE

- It looks like a fractal "Z" if you use the wrong coordinate system.
- Unlike the Hilbert, Peano and other complex curves it has edges of greater than unit length.
- It's easy to compute: you just bitwise-interleave the X and Y values:



BITWISE INTERLEAVING

- Quickest way to (de-)interleave seems to be a 256-byte lookup table.
- In the container you can store:
    - The interleaved value
    - The non-interleaved values
    - Both

NOT BITWISE INTERLEAVING

- A few years after implementing an adaptor based on that, I discovered:

Closest-Point Problems Simplified on the RAM

Timothy M. Chan*

Basic proximity problems for low-dimensional point sets, such as closest pair (CP) and approximate nearest neighbor (ANN), have been studied extensively in the computational geometry literature, with well over a hundred papers published (we merely cite the survey by Smid [10] and omit most references). Generally, optimal algorithms designed for worst-case input require hierarchical spatial structures with sophisticated balancing conditions (we mention, for example, the BBD trees of Arya *et al.*, balanced quadtrees, and Callahan and Kosaraju's fair-split trees); dynamization of these structures is even more involved (relying on Sleator and Tarjan's dynamic trees or Frederickson's topology trees).

In this note, we point out that much simpler algorithms with the same performance are possible using standard, though nonalgebraic, RAM operations. This is interesting, considering that nonalgebraic operations have been used before in the literature (e.g., in the original version of the BBD tree [2], as well as in various randomized CP methods).

The CP algorithm can be stated completely in one paragraph. Assume coordinates are positive integers

r lies in a quadtree box of diameter $O(r)$ after some shift [6].

This algorithm is not original. The approximate version was most recently proposed by Lopez and Liao [7, 8], although sorting along shuffle order, or generally space-filling curves, has been suggested often in other applied areas such as databases and pattern recognition. In computational geometry, it was used by Bern *et al.* [3] (for constructing unbalanced and balanced quadtrees, which we are trying to do without here), but is largely overlooked as a theoretical tool (hence the reason for writing this note). Shifting, on the other hand, is a well-known technique in approximation.

One objection is that we can't directly compute $\sigma(p)$. But we can decide whether $\sigma(p) < \sigma(q)$ by a straightforward procedure:

$i \leftarrow 1$;
for $j = 2, \ldots, d$ do
    if $\lfloor p_i \oplus q_i \rfloor < \lfloor p_j \oplus q_j \rfloor$ then $i \leftarrow j$;
return $p_i < q_i$.

Here $\oplus$ denotes bitwise exclusive-or and $\lfloor x \rfloor$ denotes

NOT BITWISE INTERLEAVING

```
template <typename POINT>
bool z_less(POINT a, POINT b)
{
  auto xdif = a.x ^ b.x, ydif = a.y ^ b.y;
  if (ydif <= xdif && ydif < (xdif ^ ydif))
    return a.x < b.x;
  else
    return a.y < b.y;
}
```

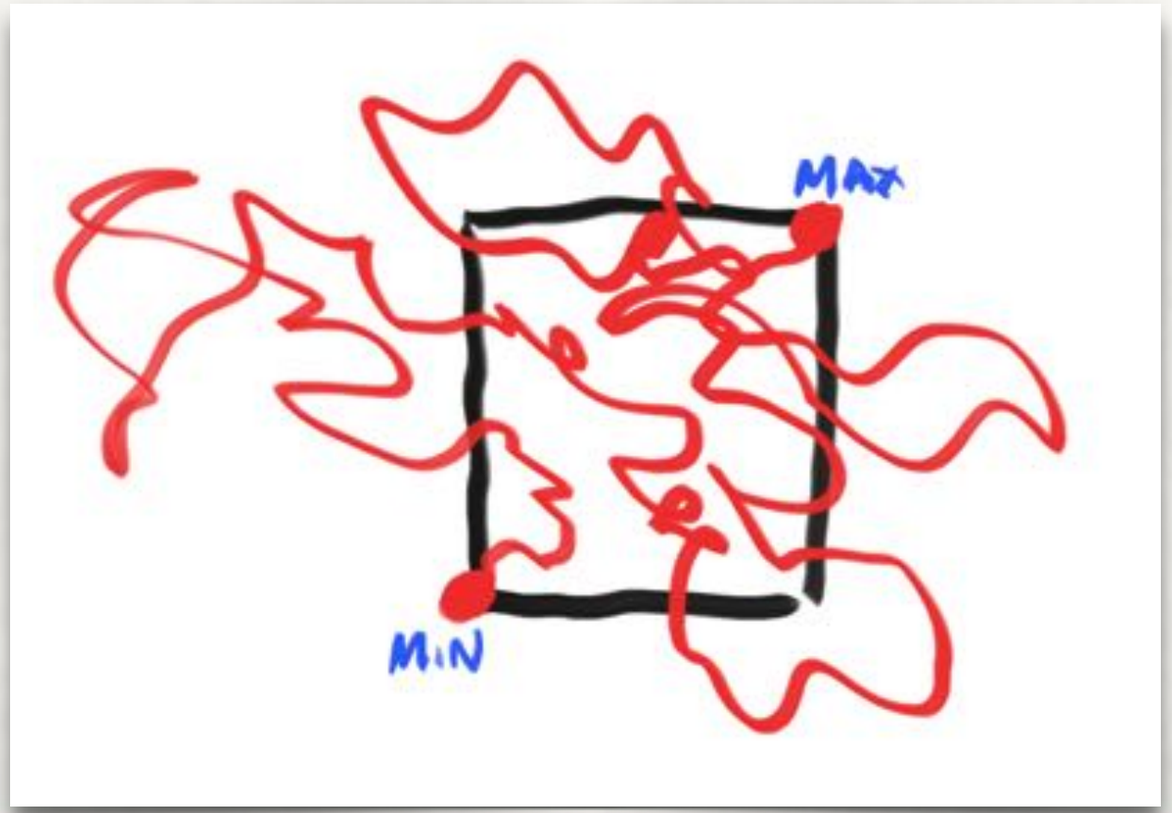
NOT BITWISE INTERLEAVING

- **std::map< Point, foo, zless<Point> >**

ALL DONE?

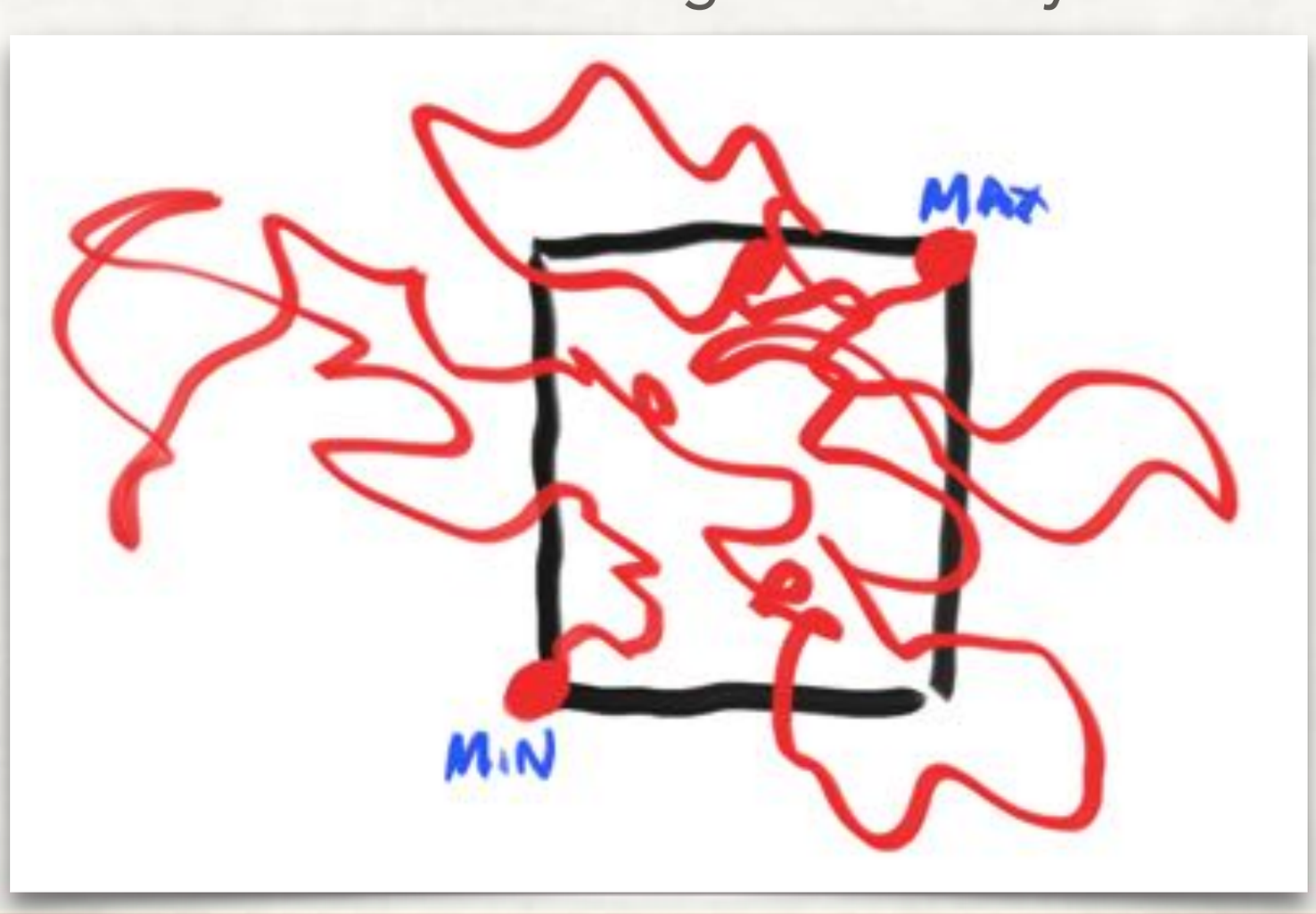
(NO)

- There is more to do in order to iterate over the content of a rectangular region, because generally the curve extends outside the rectangle.
- A useful property of the Z curve is that the curve is constrained between the bottom-left and top-right of the rectangle:



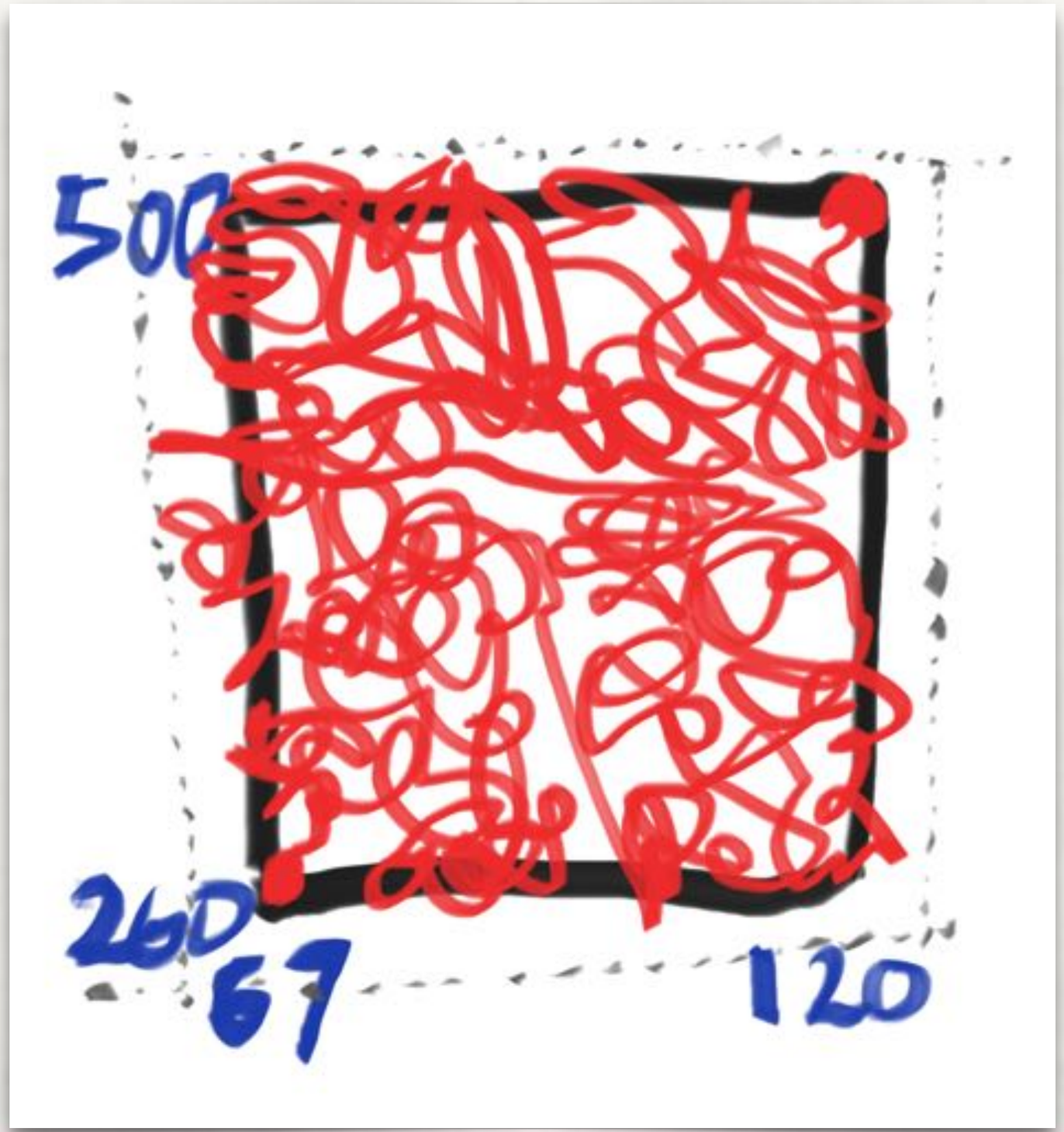
THINKING OUTSIDE THE BOX

- Visiting everything between MIN and MAX will visit everything in the box
- But also potentially lots of other things.
- One option is simply to filter out those things when they are encountered.



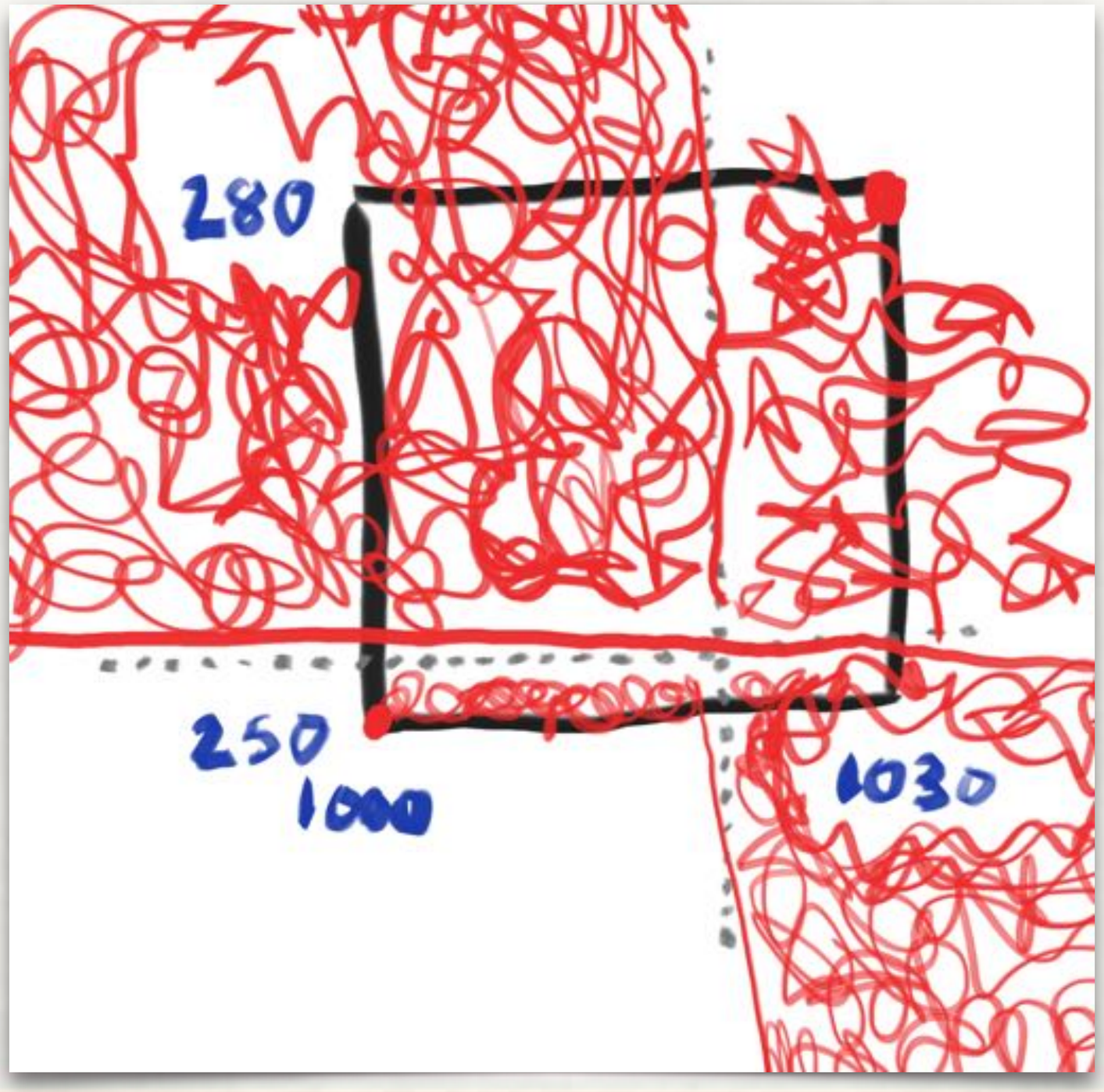
HOW FAR OUTSIDE THE BOX?

SOMETIMES THE CURVE DOESN'T WANDER FAR



HOW FAR OUTSIDE THE BOX?

IF YOUR BOX STRADDLES A LARGE POWER OF TWO IT WILL GO TO THE MOON AND BACK



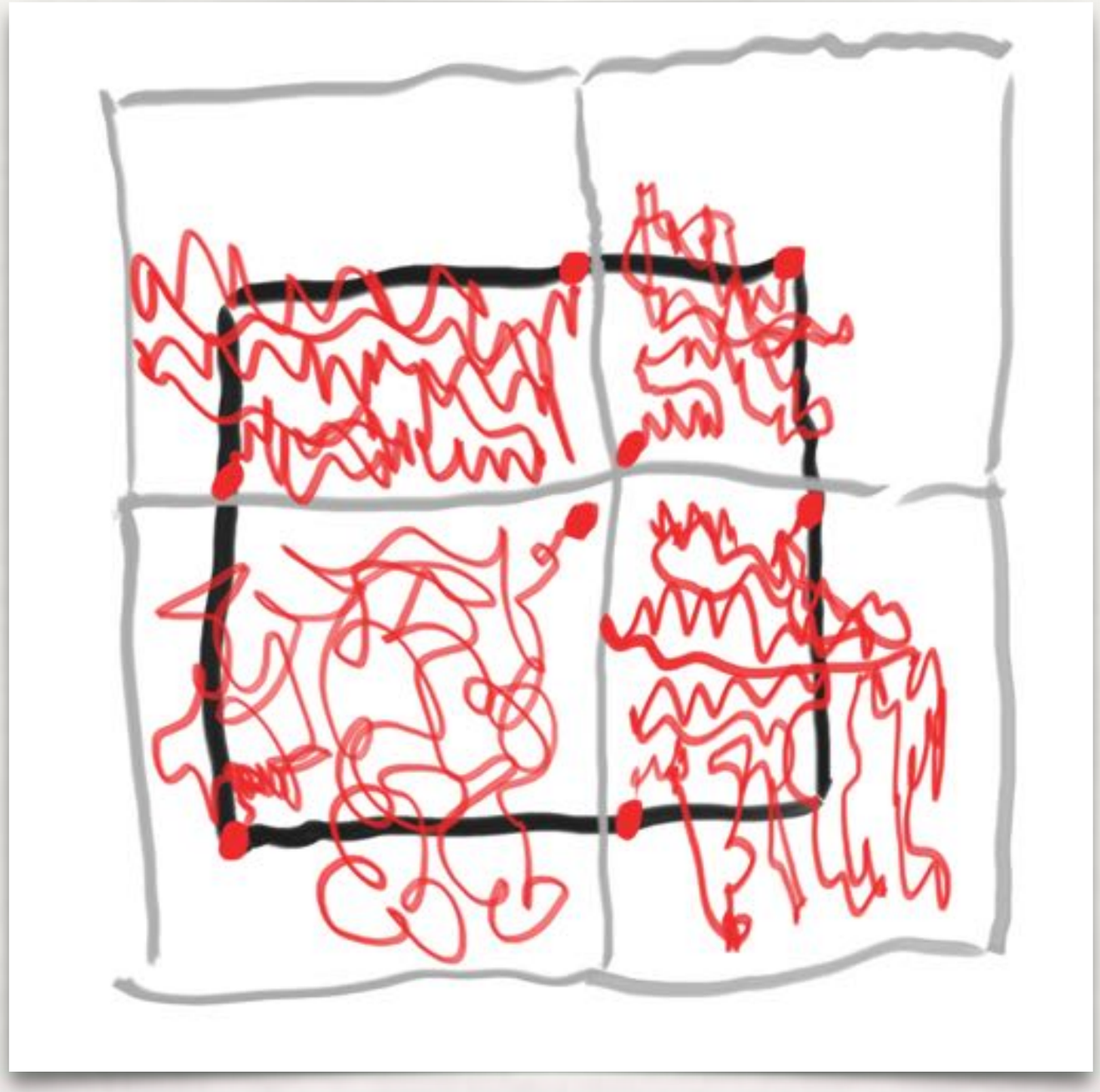
HOW FAR OUTSIDE THE BOX?

- Maybe the length of curve outside the box is (amortised) bounded by some multiple of the size of the box, or something?

- No, sorry :-(

KEEPING IT IN THE BOX

- One option is to divide your box into 4 sub-ranges, splitting at the multiples of the largest powers of two



KEEPING IT IN THE BOX

- This limits the visited space to four times the area of the box, if the box is square.

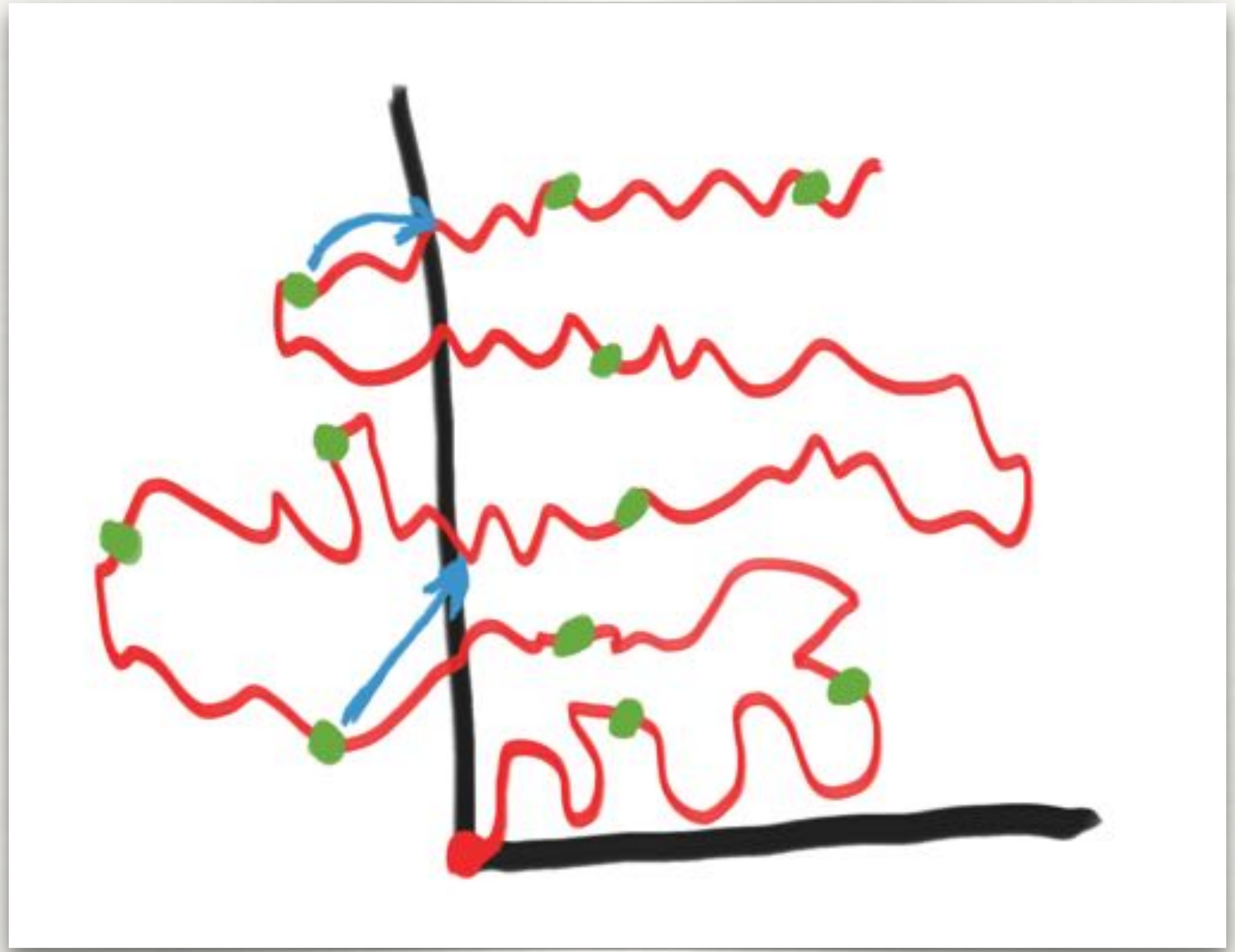
KEEPING IT IN THE BOX

- But the area visited is less important than the number of items visited, unless the items are uniformly distributed.
- Consider a cluster of items just outside a box which is itself almost empty.
- Computational complexity is worst case O(N)

BIGMIN

- The alternative way to constrain the iteration to the box is the so-called "BIGMIN" function.
- It dates from the original FORTRAN implementation when identifiers of more than six characters were considered witchcraft.
- No-one understands how it works, but it does.

BIGMIN



BIGMIN

- Given a rectangle, and a point that's outside the rectangle but on the rectangle's Z-curve, BIGMIN returns the next point on the Z-curve that is on the boundary of the rectangle.
- So when iteration reaches an item that's outside the rectangle we apply BIGMIN and then skip forward, bypassing any other items on the same "loop".
- Skipping forward is probably O(log N).

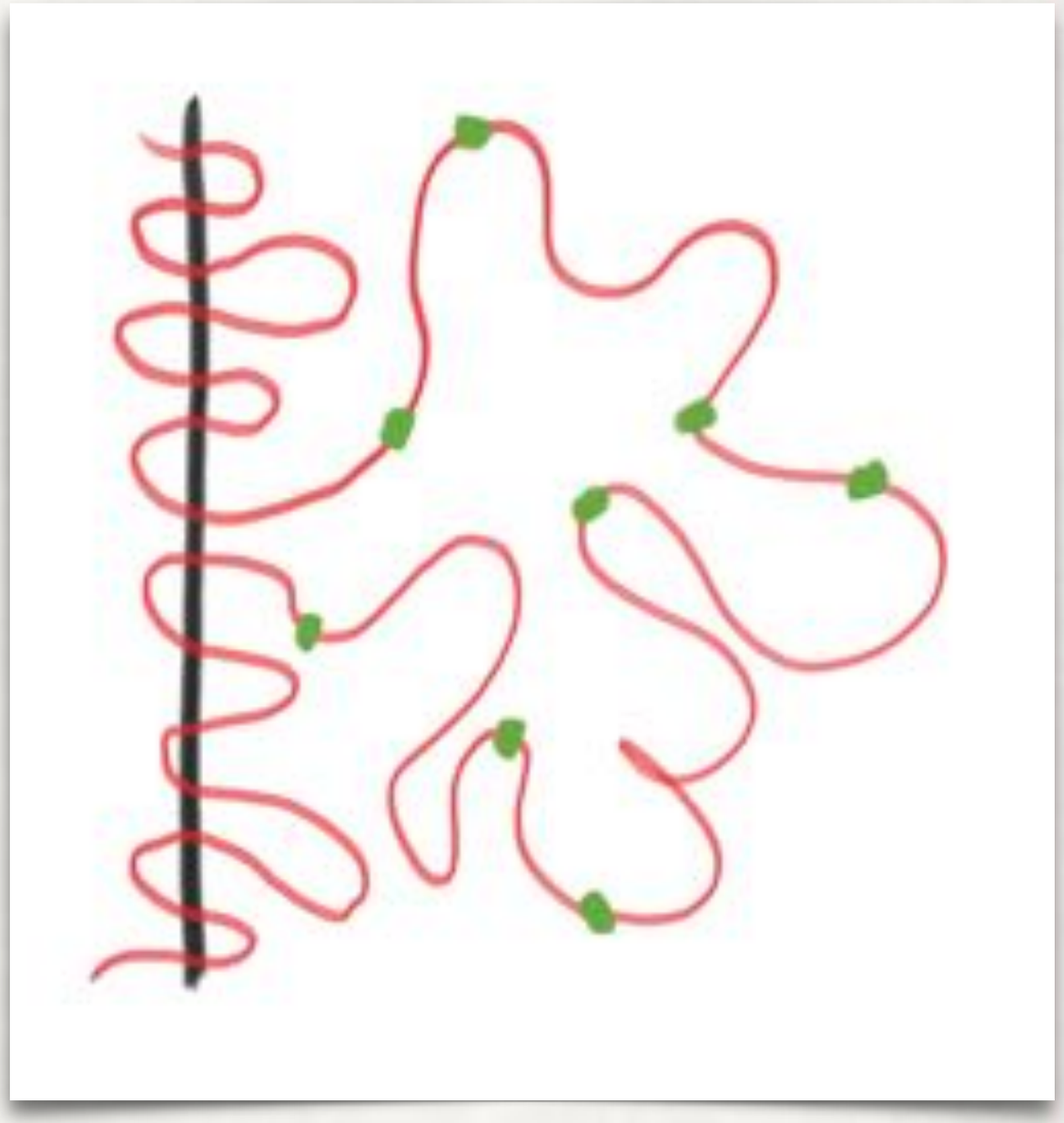
BIGMIN

BIGMIN decision table

Dividing-record-code	Actual bits of Range-minimum-code (MIN)	Range-maximum-code (MAX)	Action
0	0	0	No action; continue.
0	0	1	BIGMIN = LOAD ("1000...", MIN); MAX = LOAD ("0111...", MAX); continue.
(0	1	0)	This case not possible because MIN < = MAX.
0	1	1	BIGMIN = MIN; finish.
1	0	0	Finish.
1	0	1	MIN = LOAD ("1000...", MIN); continue.
(1	1	0)	This case not possible because MIN < = MAX.
1	1	1	No action; continue.

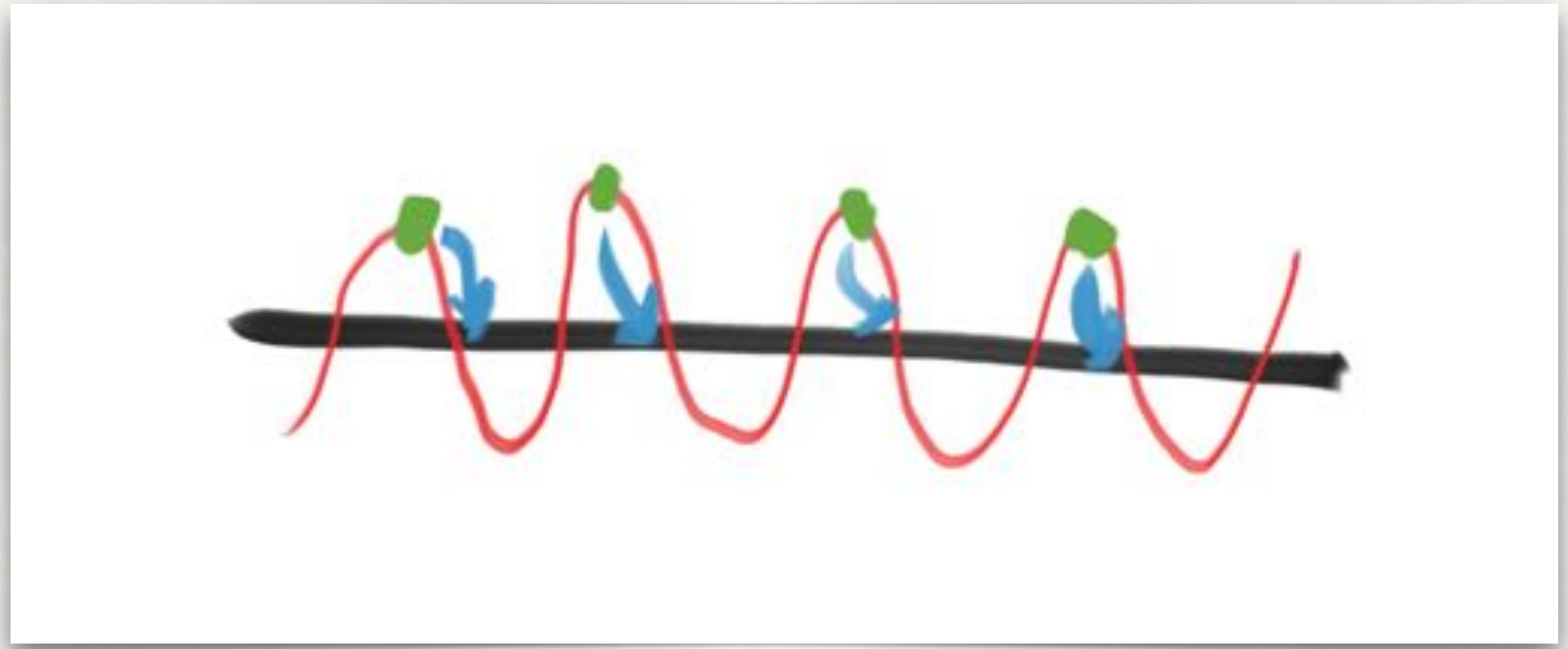
BEST CASE FOR BIGMIN

- Thinking about the "loops" outside the box, BIGMIN works best when there are:
    - Short loops with no items on them;
    - Long loops with many items that can all be skipped in one go.
- This is what should happen with a fractal curve like the Z-curve. It's exactly what doesn't happen with raster scan.



WORST CASE FOR BIGMIN

- The worst case is when there is just one item on each "loop".
- This is worse than just filtering out these items - it makes the iteration O(N log N) rather than O(N)



LINEAR LOWER BOUND

- A variant of std::lower_bound that does a short linear search before falling back to the logarithmic search.
- If you use it to iterate through the whole container, complexity is better than O(N).
- Kludge needed to work with std::map's member lower_bound.

A 2D CONTAINER ADAPTER

- Point and Rectangle classes.
- Two "magic" Z-curve functions, z_less and bigmin.
- linear_lower_bound.
- Type metafunction to change associative container's comparison to z_less.
- adapt2d template.
- Iterator using boost::iterator_facade

CODE

http://chezphil.org/tmp/adapt2d.cc

CONCLUSIONS

- I've been using this technique for storing 2D data for about 10 years.
- I think its greatest strength is that you can apply it to many different underlying containers. I've used:
    - Read-only memory-mapped files.
    - Flat maps (i.e. sorted vectors).
    - Containers with special allocators.
- Performance is good in practice.
- But worst-case computational complexity is O(N).

REFERENCES

- Good starting point for space filling curves in general:

  http://www.win.tue.nl/~hermanh/doku.php?id=recursive_tilings_and_space-filling_curves

- An early paper describing how to use the Z-curve, including the BIGMIN function:

  Tropf, H.; Herzog, H. (1981), "Multidimensional Range Search in Dynamically Balanced Trees", Angewandte Informatik 2: 71–77.

- How to order points without actually interleaving the bits:

  Chan, T. (2002), "Closest-point problems simplified on the RAM", ACM-SIAM Symposium on Discrete Algorithms.