



WIKIPEDIA  
The Free Encyclopedia

Main page  
Contents  
Featured content  
Current events  
Random article  
Donate to Wikipedia  
Wikipedia store

Interaction  
Help  
About Wikipedia  
Community portal  
Recent changes  
Contact page

Tools  
What links here  
Related changes  
Upload file  
Special pages  
Permanent link  
Page information  
Wikidata item  
Cite this page


Print/export  
Create a book  
Download as PDF  
Printable version

Languages   
Čeština  
Deutsch  
Français  
Српски / srpski  
 Edit links

Create account Log in

Article [Talk](#)

[Read](#) [Edit](#) [More](#) ▾

Search 

# Z-order curve

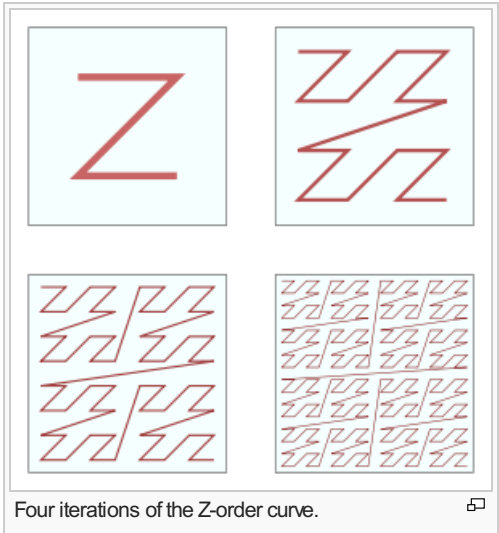
From Wikipedia, the free encyclopedia  
(Redirected from [Z-order \(curve\)](#))

*Not to be confused with [Z curve](#) or [Z-order](#).*

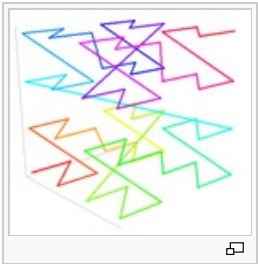
In [mathematical analysis](#) and [computer science](#), **Z-order**, **Morton order**, or **Morton code** is a [function](#) which maps multidimensional data to one dimension while preserving locality of the data points. It was introduced in 1966 by [G. M. Morton](#).<sup>[1]</sup> The z-value of a point in multidimensions is simply calculated by interleaving the [binary](#) representations of its coordinate values. Once the data are sorted into this ordering, any one-dimensional data structure can be used such as [binary search trees](#), [B-trees](#), [skip lists](#) or (with low significant bits truncated) [hash tables](#). The resulting ordering can equivalently be described as the order one would get from a depth-first traversal of a [quadtree](#).

## Contents [hide]

- 1 Coordinate values
- 2 Efficiently building quadtrees
- 3 Use with one-dimensional data structures for range searching
- 4 Related structures
- 5 Applications in linear algebra
- 6 See also
- 7 References
- 8 External links



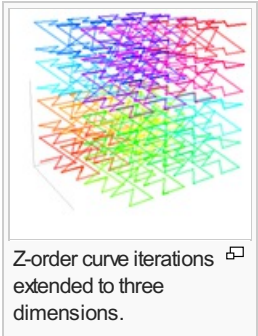
Four iterations of the Z-order curve.



## Coordinate values [\[edit\]](#)

The figure below shows the Z-values for the two dimensional case with integer coordinates  $0 \leq x \leq 7$ ,  $0 \leq y \leq 7$  (shown both in decimal and binary). [Interleaving](#) the binary coordinate values yields binary z-values as shown. Connecting the z-values in their numerical order produces the recursively Z-shaped curve. Two-dimensional Z-values are also called as quadkey ones.

	x:		0	1	2	3	4	5	6	7
			000	001	010	011	100	101	110	111
y:	0	000000	000001	000100	000101	010000	010001	010100	010101	
	000									
1	001	000010	000011	000110	000111	010010	010011	010110	010111	
2	010	001000	001001	001100	001101	011000	011001	011100	011101	
3	011	001010	001011	001110	001111	011010	011011	011110	011111	
4	100	100000	100001	100100	100101	110000	110001	110100	110101	
5	101	100010	100011	100110	100111	110010	110011	110110	110111	
6	110	101000	101001	101100	101101	111000	111001	111100	111101	
7	111	101010	101011	101110	101111	111010	111011	111110	111111	



Z-order curve iterations extended to three dimensions.

The Z-values of x's are described as binary numbers:

```
x[] = {0b000000, 0b000001, 0b000100, 0b000101, 0b010000, 0b010001, 0b010100,
0b010101}
```

The sum and subtraction of two x's are calculated by using [bitwise operations](#):

```
x[i+j] = ((x[i] | 0b101010) + x[j]) & 0b01010101
x[i-j] = (x[i] - x[j]) & 0b01010101 if i >= j
```

## Efficiently building quadtrees [\[edit\]](#)

The Z-ordering can be used to efficiently build a quadtree for a set of points.<sup>[2]</sup> The basic idea is to sort the input set according to Z-order. Once sorted, the points can either be stored in a binary search tree and used directly, which is called a linear quadtree,<sup>[3]</sup> or they can be used to build a pointer based quadtree.

The input points are usually scaled in each dimension to be positive integers, either as a fixed point representation over the unit range [0, 1] or corresponding to the machine word size. Both representations are equivalent and allow for the highest order non-zero bit to be found in constant time. Each square in the quadtree has a side length which is a power of two, and corner coordinates which are multiples of the side length. Given any two points, the *derived square* for the two points is the smallest square covering both points. The interleaving of bits from the x and y components of each point is called the *shuffle* of x and y, and can be extended to higher dimensions.<sup>[2]</sup>

Points can be sorted according to their shuffle without explicitly interleaving the bits. To do this, for each dimension, the most significant bit of the [exclusive or](#) of the coordinates of the two points for that dimension is examined. The dimension for which the most significant bit is largest is then used to compare the two points to determine their shuffle order.

The exclusive or operation masks off the higher order bits for which the two coordinates are identical. Since the shuffle interleaves bits from higher order to lower order, identifying the coordinate with the largest most significant bit, identifies the first bit in the shuffle order which differs, and that coordinate can be used to compare the two points.<sup>[4]</sup> This is shown in the following Python code:

```
def cmp_zorder(a, b):
    j = 0
    k = 0
    x = 0
    for k in range(dim):
        y = a[k] ^ b[k]
        if less_msb(x, y):
            j = k
            x = y
    return a[j] - b[j]
```

One way to determine whether the most significant smaller is to compare the floor of the base-2 logarithm of each point. It turns out the following operation is equivalent, and only requires exclusive or operations:<sup>[4]</sup>

```
def less_msb(x, y):
    return x < y and x < (x ^ y)
```

It is also possible to compare floating point numbers using the same technique. The *less\_msb* function is modified to first compare the exponents. Only when they are equal is the standard *less\_msb* function used on the mantissas.<sup>[5]</sup>

Once the points are in sorted order, two properties make it easy to build a quadtree: The first is that the points contained in a square of the quadtree form a contiguous interval in the sorted order. The second is that if more than one child of a square contains an input point, the square is the *derived square* for two adjacent points in the sorted order.

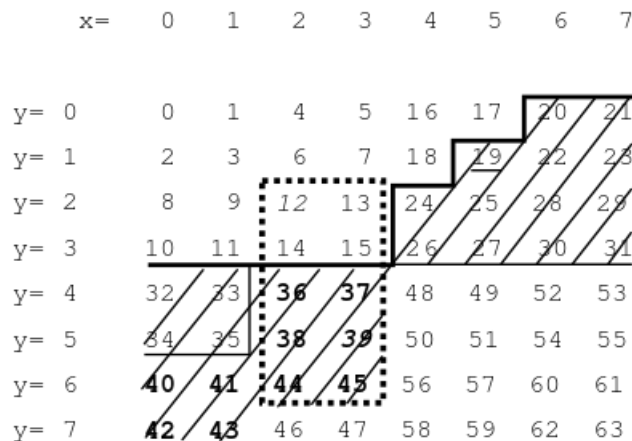
For each adjacent pair of points, the derived square is computed and its side length determined. For each derived square, the interval containing it is bounded by the first larger square to the right and to the left in sorted order.<sup>[2]</sup> Each such interval corresponds to a square in the quadtree. The result of this is a compressed

quadtree, where only nodes containing input points or two or more children are present. A non-compressed quadtree can be built by restoring the missing nodes, if desired.

Rather than building a pointer based quadtree, the points can be maintained in sorted order in a data structure such as a binary search tree. This allows points to be added and deleted in  $O(\log n)$  time. Two quadtrees can be merged by merging the two sorted sets of points, and removing duplicates. Point location can be done by searching for the points preceding and following the query point in the sorted order. If the quadtree is compressed, the predecessor node found may be an arbitrary leaf inside the compressed node of interest. In this case, it is necessary to find the predecessor of the least common ancestor of the query point and the leaf found.<sup>[6]</sup>

## Use with one-dimensional data structures for range searching <sup>[edit]</sup>

Although preserving locality well, for efficient range searches an algorithm is necessary for calculating, from a point encountered in the data structure, the next Z-value which is in the multidimensional search range:



In this example, the range being queried ( $x = 2, \dots, 3, y = 2, \dots, 6$ ) is indicated by the dotted rectangle. Its highest Z-value (MAX) is 45. In this example, the value  $F = 19$  is encountered when searching a data structure in increasing Z-value direction, so we would have to search in the interval between  $F$  and MAX (hatched area). To speed up the search, one would calculate the next Z-value which is in the search range, called BIGMIN (36 in the example) and only search in the interval between BIGMIN and MAX (bold values), thus skipping most of the hatched area. Searching in decreasing direction is analogous with LITMAX which is the highest Z-value in the query range lower than  $F$ . The BIGMIN problem has first been stated and its solution shown in Tropf and Herzog.<sup>[7]</sup> This solution is also used in [UB-trees](#) ("GetNextZ-address"). As the approach does not depend on the one dimensional data structure chosen, there is still free choice of structuring the data, so well known methods such as balanced trees can be used to cope with dynamic data (in contrast for example to [R-trees](#) where special considerations are necessary). Similarly, this independence makes it easier to incorporate the method into existing databases.

Applying the method hierarchically (according to the data structure at hand), optionally in both increasing and decreasing direction, yields highly efficient multidimensional range search which is important in both commercial and technical applications, e.g. as a procedure underlying nearest neighbour searches. Z-order is one of the few multidimensional access methods that has found its way into commercial database systems ([Oracle database](#) 1995,<sup>[8]</sup> [Transbase](#) 2000 <sup>[9]</sup>).

As long ago as 1966, G.M.Morton proposed Z-order for file sequencing of a static two dimensional geographical database. Areal data units are contained in one or a few quadratic frames represented by their sizes and lower right corner Z-values, the sizes complying with the Z-order hierarchy at the corner position. With high probability, changing to an adjacent frame is done with one or a few relatively small scanning steps.

## Related structures <sup>[edit]</sup>

As an alternative, the [Hilbert curve](#) has been suggested as it has a better order-preserving behaviour, but here the calculations are much more complicated, leading to significant processor overhead. BIGMIN source code for both Z-curve and Hilbert-curve were described in a patent by H. Tropf.<sup>[10]</sup>

For a recent overview on multidimensional data processing, including e.g. nearest neighbour searches, see [Hanan Samet's](#) textbook.<sup>[11]</sup>

## Applications in linear algebra <sup>[edit]</sup>

The **Strassen algorithm** for matrix multiplication is based on splitting the matrices in four blocks, and then recursively splitting each of these blocks in four smaller blocks, until the blocks are single elements (or more practically: until reaching matrices so small that the trivial algorithm is faster). Arranging the matrix elements in Z-order then improves locality, and has the additional advantage (compared to row- or column-major ordering) that the subroutine for multiplying two blocks does not need to know the total size of the matrix, but only the size of the blocks and their location in memory. Effective use of Strassen multiplication with Z-order has been demonstrated, see Valsalam and Skjellum's 2002 paper.<sup>[12]</sup>

## See also <sup>[edit]</sup>

- [Space filling curve](#)
- [UB-tree](#)
- [Hilbert curve](#)
- [Hilbert R-tree](#)
- [Spatial index](#)
- [Geohash](#)
- [Locality preserving hashing](#)
- [Matrix representation](#)
- [Linear algebra](#)

## References <sup>[edit]</sup>

- ↑ Morton, G. M. (1966), *A computer Oriented Geodetic Data Base; and a New Technique in File Sequencing*, Technical Report, Ottawa, Canada: IBM Ltd.
- ↑  <sup>***a*** ***b*** ***c***</sup> Bern, M.; Eppstein, D.; Teng, S.-H. (1999), "Parallel construction of quadrees and quality triangulations", *Int. J. Comp. Geom. & Appl.* **9** (6): 517–532, doi:10.1142/S0218195999000303 .
- ↑ Gargantini, I. (1982), "An effective way to represent quadrees", *Communications of the ACM* **25** (12): 905–910, doi:10.1145/358728.358741 .
- ↑  <sup>***a*** ***b***</sup> Chan, T. (2002), "Closest-point problems simplified on the RAM", *ACM-SIAM Symposium on Discrete Algorithms* .
- ↑ Connor, M.; Kumar, P (2009), "Fast construction of k-nearest neighbour graphs for point clouds", *IEEE Transactions on Visualization and Computer Graphics*  (PDF)
- ↑ Har-Peled, S. (2010), *Data structures for geometric approximation*  (PDF)
- ↑ Tropf, H.; Herzog, H. (1981), "Multidimensional Range Search in Dynamically Balanced Trees"  (PDF), *Angewandte Informatik* **2**: 71–77.
- ↑ Gaede, Volker; Guenther, Oliver (1998), "Multidimensional access methods"  (PDF), *ACM Computing Surveys* **30** (2): 170–231, doi:10.1145/280277.280279 .
- ↑ Ramsak, Frank; Markl, Volker; Fenk, Robert; Zirkel, Martin; Elhardt, Klaus; Bayer, Rudolf (2000), "Integrating the UB-tree into a Database System Kernel", *Int. Conf. on Very Large Databases (VLDB)*  (PDF), pp. 263–272.
- ↑  <sup>***US*** ***7321890***</sup> Tropf, H., "Database system and method for organizing data elements according to a Hilbert curve", issued January 22, 2008.
- ↑ Samet, H. (2006), *Foundations of Multidimensional and Metric Data Structures*, San Francisco: Morgan-Kaufmann.
- ↑ Vinod Valsalam, Anthony Skjellum: A framework for high-performance matrix multiplication based on hierarchical abstractions, algorithms and optimized low-level kernels. Concurrency and Computation: Practice and Experience 14(10): 805-839 (2002)<sup>[1]</sup> <sup>[2]</sup>

## External links <sup>[edit]</sup>

- [STANN: A library for approximate nearest neighbor search, using Z-order curve](#)
- [Methods for programming bit interleaving](#) , Sean Eron Anderson, Stanford University

Categories:	<a href="#">Fractal curves</a>	<a href="#">Database algorithms</a>	<a href="#">Geometric data structures</a>	<a href="#">Database index techniques</a>
	<a href="#">Linear algebra</a>			

This page was last modified on 17 April 2015, at 03:15.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#). Wikipedia® is a registered trademark of the [Wikimedia Foundation, Inc.](#), a non-profit organization.

[Privacy policy](#) [About Wikipedia](#) [Disclaimers](#) [Contact Wikipedia](#) [Developers](#) [Mobile view](#)

