LECTURE NOTES ON

SPATIAL LOCALITY ALGORITHMS

MHPC MASTER

R. Valdarnini (SISSA)

INDEX

I. SPACE FILLING CURVES

I. 1 Introduction

In scientific computing multidimensional data structure occur in a variety of ways, here we give some examples:

- Discretization - based data and geometric models, such as quadtrees or octrees
- Vector, matrices, tensor etc. in linear algebra
- All kind of rasterized image data, such as pixel-based image data from computer tomography
- Coordinates in general (often as part of a graph)
- Tables, as for example in database
- Statistical data - in computer finance, for example, baskets of different stock or options might be considered as multidimensional data

- Examples of Algorithm and operations on multidimensional data

For the above list of multidimensional data the following algorithms and operations are applicable

- Matrix operations
- Traversal of data
- Storing and retrieving multi dimensional data sets
- selecting particular data items or data subsets
- Partioning of data, i.e. distributing the data sets into several parts, in particular for parallel processing. A strictly related problem is load balancing, i.e. the distribution of an equal amount of computational load among the different threads of a parallel machine.
- Accessing neighbor elements

In all of the above examples, the sequentialization of the data sets form a central subproblem which has a significant impact on the efficiency of the algorithm used for

- traversal of data
- storing and retrieving of data
- Nearest heighbors

- Requirements for efficient sequential order

To be efficient, a sequential order must satisfy several properties

- Necessary properties of sequential order

The first property is that must generate a unique index for each data item

A traversal of the index will lead to a traversal of the data set, where all data items are processed exactly once - In mathematical language the mapping between indices and data items should be bijective.

For a contiguous data set, the sequentialisation should also lead to a contiguous sequence of indices.

Finally, we will need sequential orders on data sets of different extensions. Hence, our indexing scheme needs to be adjustable according to some parameters. We then require a family of sequential orders.

- Requirements requiring efficiency

The following is a general list of requirements for an efficient sequentialisation, the relative importance will vary with applications

. The mapping between data and indices must be easy to compute

. Neighbor relation should be retained, if two data items are close in the space of multidimensional data, their indices should be close together and viceversa.

. Two variants of preserving locality are the continuity of the mapping for sequentialization and the clustering properties. Continuity ensures that data items with successive indices will be direct neighbors in the multidimensional space, as well. The clustering property is satisfied if a data set defined via a limited index range will not have a large extension in either direction of the data space.

. The sequentialization should not favour or penalise certain dimensions.

A 2D example

A classical example of a 2D multidimensional data in which are encountered the previous problem is the storage of a 2D matrix. Assuming Fortran as a reference

of programming language, 2D matrices are stored in row sequential order

$$\text{addres}(A_{i,j}) = i + j \cdot m \qquad \text{I.1}$$

where $\dim[A] = m \times n$

This addressing has several drawbacks:

. Neighbor relation are only preserved in one direction, neighbors in the column direction will have a distance of m elements. For higher-dimensional data structure, the neighbor properties get even worse

. Continuity of the sequentialization is violated at the boundary of the matrix

. There is no clustering of data. A given index range corresponds To few successive rows (or columns) and will thus correspond to a narrow band of the matrix.

The above discussion illustrates the need for the construction of an algorithm which introduces <u>sequential order</u> for multidimensional data structure. The mathematical concepts which are applied in scientific computing to solve this problem are the so-called <u>space Filling curves</u> (SFC).

I.2 DEFINITION OF SPACE FILLING CURVES

Introducing a sequential order on a d-dimensional array of elements defines a corresponding mapping from the range of array indices $\{1,2 \ldots n^d\}$ To the sequential indices $\{1,2 \ldots n^d\}$ and viceversa. In continuous terms we should derive a family of mapping from $[0,1]^d$ to $[0,1]$ and viceversa.

From the previous discussion a good mapping should be bijective and continuous. A bijective mapping ensures a one-to-one correspondence and a continuous mapping will avoid jumps between the image points of adjacent parameters, which will avoid 'holes' in the generated sequential order and improve locality and clustering of data.

In order to derive in a more rigorous way the definition of SFC let us first introduce the following definition

Def. I.1: Let $f: J \to R^n$ be a continuous mapping of the compact set $J \subset R$ into R^n. The respective image $f_\alpha(J)$ of such a mapping is then called a curve, and $x = f(t)$ $t \in J$ is called the parameter representation of the curve.

Accordingly, a curve is the image of any continuous mapping from a parameter interval into a 2D or higher dimensional space. Usually, the compact parameter f will simply be the unit interval $[0,1]$.

Hence, finding a bijective curve that maps the unit interval to the unit square means finding a curve that visit each point of the unit square (surjectivity) but does not intersect itself (to remain injective).

Netto (1879) proved that such curve cannot exist. In 1890, to demonstrate an earlier counterintuitive results that the number of points in a unit interval has the same cardinality as the number of points in a unit square (Cantor 1878), G. Peano introduced SFC as curves that pass through any points of a closed unit square.

The Peano curve is continuous and surjective (i.e. the curve visit each point of the unit square) but not injective. A SFC is then a thread which goes through all the points of space while visiting each point only once. The SFC imposes a linear order of points in a multidimensional space.

The Figure shows three iterations of the Peano curve. Similarly, Hilbert (1891) presented a curve with similar properties (Fig. I.2). All these curves will visit every point of the unit square

Def. I.2 (space filling curves). Given a curve $f_*(\mathcal{I})$ and the corresponding mapping $f: \mathcal{I} \to \mathbb{R}^n$ then $f_*(\mathcal{I})$ has a Jordan content (area, volume) larger than zero.

A SFC is then the image of a particular type of function which is surjective and whose range is the unit square $[0,1]^2$ Whereas functions which describe SFC are surjective, they need not also be injective, i.e. they may visit the same point in space more than once. We will confine our interest to SFC which are described by bijective functions. Moreover, only discrete curves will be considered, since they are used in computer science.

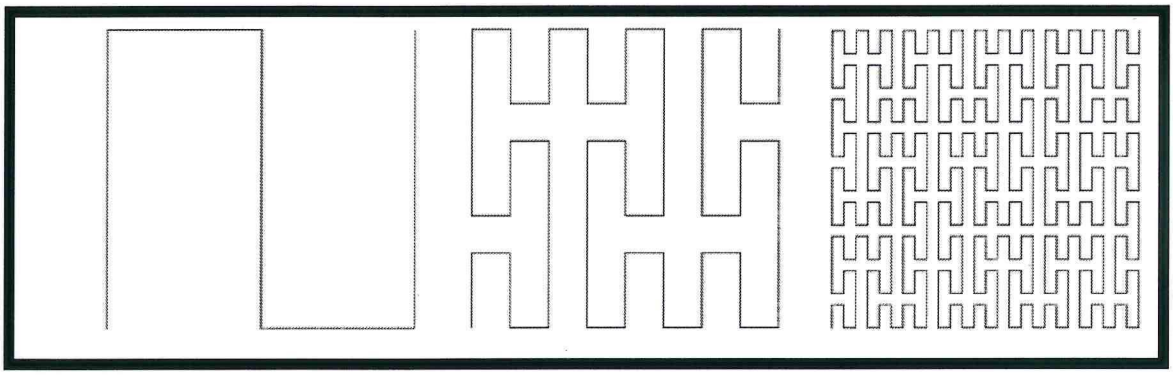


Fig.I1: Three iterations of the Peano curve construction

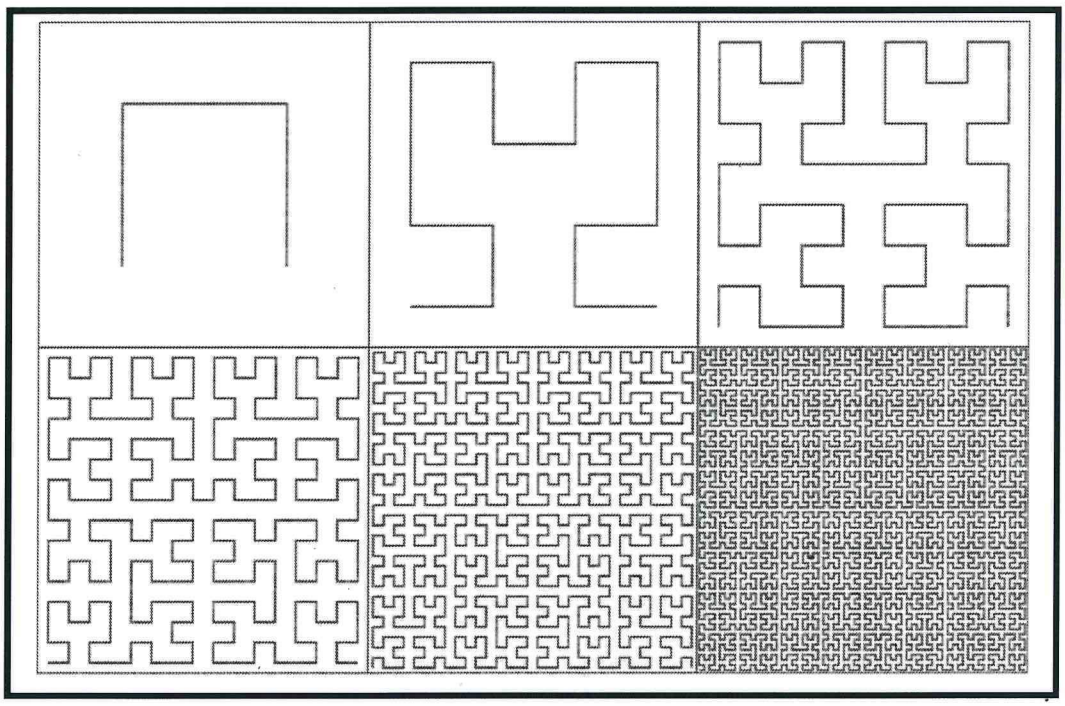


Fig.I2: Six iterations of the Hilbert curve construction

Discrete SFC : Consider a d-dimensional hypercube. Bisecting this hypercube k times along each dimension result in a d-dimensional matrix of $2^d \times 2^d \times \ldots = 2^{dk}$ non-overlapping hypercell of equal size.

A SFC is a mapping of these hypercells To a one-dimensional linear ordering. The process of mapping discrete multi-dimensional data into a one-dimensional ordering is often referred to as linearization. There are many different mappings that yield a linearization of multidimensional data. Here we will give examples for the two most important curves used in computer science: the Morton (or Z-) order curve and the Hilbert curve.

I.3 THE Z-ORDER CURVE

The Z-order curve is first attributed to Morton (1966) and it is widely used in many areas of research. Although the Hilbert curve has better compactness properties, nevertheless its relative simple algorithm renders its use particularly appealing. Examples of first to third order curve are shown in Fig. I3 which clearly illustrates self-similarity as the curve transforms from one order to the next.

The mapping of the Z-order is carried out by a process which is often referred to as bit interleaving. The coordinates are integerized according to the tresformation

$$x \rightarrow \text{floor}\left(x \cdot 2^p / L\right) \qquad \text{I-2}$$

where $0 \leq x < L$ and p is the order of the curve. There are then p bits for each coordinate and for the generic point $\underline{P}$

$$P = (x_1, x_2 \ldots x_p, y_1, \ldots y_p, z_1, \ldots z_p) \qquad \text{I-3}$$

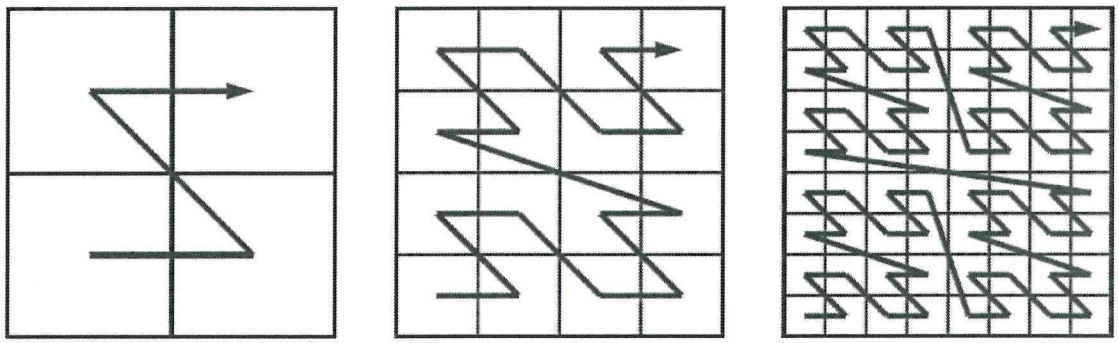


Fig.I3: Approximations of the Z-order curve in two dimensions from zero to third order

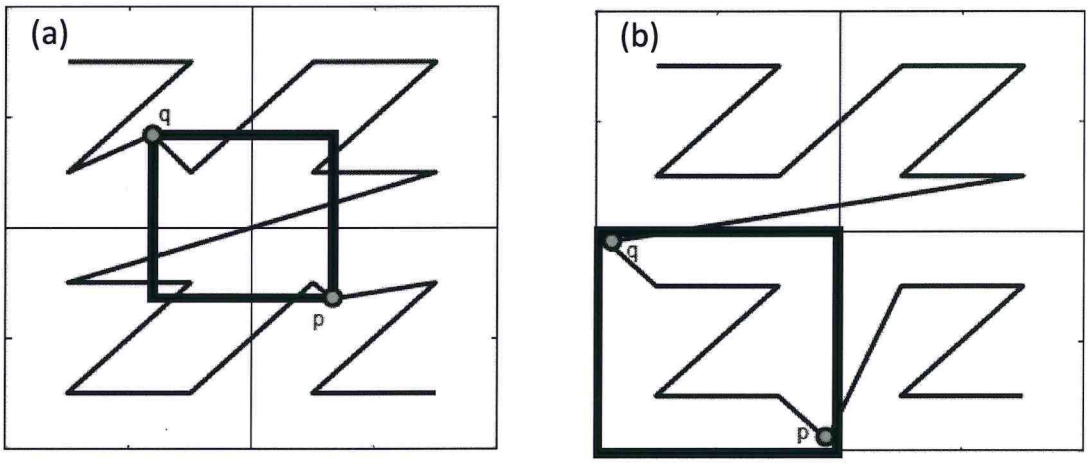


Fig.I4: The Morton order curve preceding the upper left corner, and following the lower right right corner of a box, will never intersect the box (a). The smallest quadtree box containing two points will also contain all points lying betweeen the two in Morton order (b).

The $\underline{z}$-order key, which result from bit interleaving, is

$$\underline{z} = (x_1 y_1, x_2 y_2 \ldots\ldots, x_p y_p) \qquad \text{I-4}$$

In n dimensions the $\underline{z}$-key comprises p sequences of n bits. More generically the p most significant bits can be expressed as $\underline{z}_1$ the next n as $\underline{z}_2$ and so on

$$\underline{z} = (\underline{z}_1, \underline{z}_2 \ldots \underline{z}_p) \qquad \text{I-5}$$

where $\underline{z}_i$ is in the range $[0, 2^{p-1}]$.
We can say that $\underline{z}_1$ is a first order approximation to $\underline{z}$, z_2 is a second order approximation and so on. As a practical example let us consider the second order curve of Fig. I3 (mid panel). We assume $L = 4$ and within the square we take 16 points uniformly placed in a lattice of 4×4. The point coordinates of the first quadrant (lower left) are

$$(x,y) = (0.5, 0.5) \quad (1.5, 0.5) \quad (0.5, 1.5) \quad (1.5, 1.5)$$
$$= \underline{I}$$

The coordinates of the points in the other quadrants are given by (in a compact form)

$$\underline{II} = \underline{I} + (2,0)$$

$$\underline{III} = \underline{I} + (0,2)$$

$$\underline{IV} = \underline{I} + (2,2)$$

The integerized coordinates of the points in quadrant $\underline{I}$ are

$$\underline{I} = (0,0) \quad (1,0) \quad (0,1) \quad (1,1)$$

and for $p=2$ their bits

$$\underline{I} = (00,00) \quad (10,00) \quad (00,10) \quad (10,10)$$

accordingly, the bit representation of the other points are

$$\underline{II} = \underline{I} + (01,00)$$

$$\underline{III} = \underline{I} + (00,01)$$

$$\underline{IV} = \underline{I} + (01,01)$$

Finally, the 16 2-keys of the 16 points are

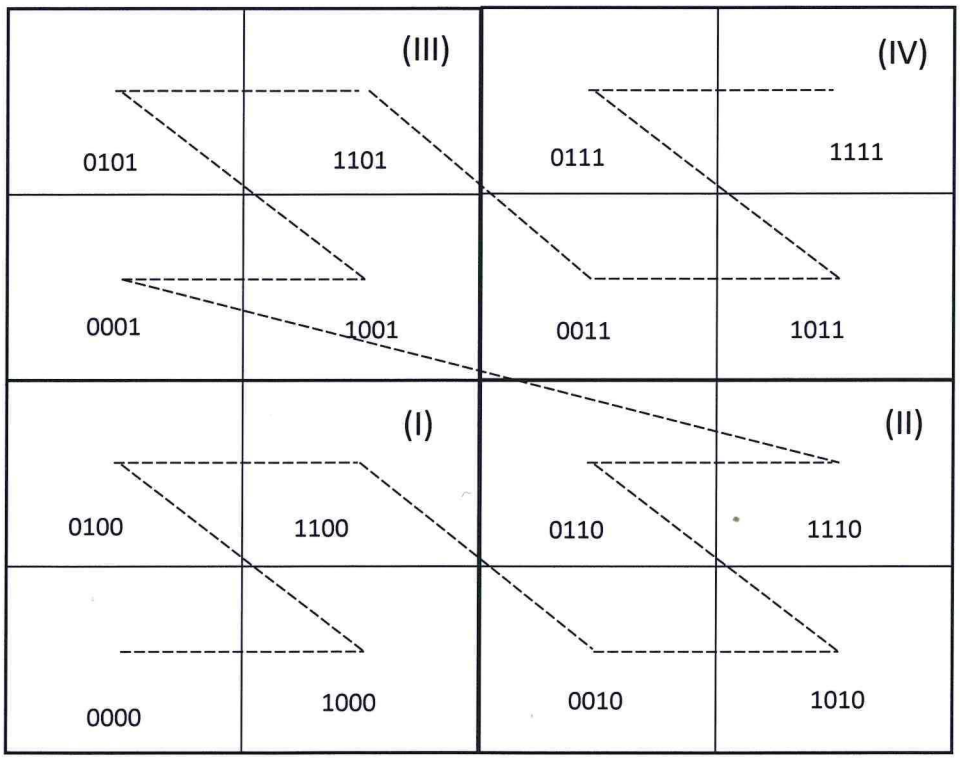
I = $(0000)_0$ $(1000)_1$ $(0100)_2$ $(1100)_3$

II = $(0010)_4$ $(1010)_5$ $(0110)_6$ $(1110)_7$

III = $(0001)_8$ $(1001)_9$ $(0101)_{10}$ $(1101)_{11}$

IV = $(0011)_{12}$ $(1011)_{13}$ $(0111)_{14}$ $(1111)_{15}$

In the panel below the different points are connected according to the order of



their z-values, with the curve assuming its characteristic shape.

The curve is also referred to as N-order because of the different order in which bit interleaving can be performed.

Here we have followed the notation of ordering the bits from the left, with the most significant to the right and at each level x-bits are placed first.

In a similar way one can define a z-key in 3 dimensions. The integerized coordinates (x, y, z) of a given point are scaled between 0 and $2^p - 1$. The bits are shuffled together to produce a $3p$-bit key

$$(x_{p-1} \ldots x_1, y_{p-1} \ldots y_1, z_{p-1} \ldots z_1) \quad \text{I.6}$$
$$\mapsto (x_0 y_0 z_0, x_1 y_1 z_1, \ldots x_p y_p z_p)$$

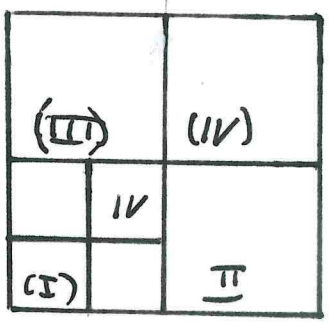
with the most significant bits being on the right.

The z-curve, or Morton order, cannot be called space-filling curves, as they lack the continuity required for a curve. This is clearly seen in the examples shown in Fig. I3.

However, as already outlined, they have several properties which render their use particularly convenient in several problems.

A strictly related problem to the construction of a Σ curve is the partitioning of data sets using quadtrees (2D) or octtrees (3D). These kinds will be examined in the chapter dedicated to tree structures. Another problem which has a deep connection to the Σ-order of a curve is the so-called nearest neighbor search. The problem seeks all the spatial neighbors of a given point which satisfy $\| x_i - x_k \| \leq R$. This problem will be discussed in detail later. However it is easily to see the deep connection between this problem and the Σ-order. Imagine, for example, that we construct a fourth order Σ-curve

Looking at the subdivision in the right figure, all of the points which lie in the subquadrant IV of quadrant I will share the same top 4 bits of the $\underline{z}$ curve: 1100. In three dimensions, for the l-th level from the root cell, all of the points with the top $3l$ bits lie in the same subcell. These particles are then easily identified by performing a bit masking on their $\underline{z}$ key. In this context, let us introduce two important properties of $\underline{z}$-curves. These are introduced in Fig. 4, panels (a) and (b), respectively.



I.4 HILBERT CURVE

Soon after Peano (1890), Hilbert (1891) gave a geometrical interpretation of space filling curves. To illustrate this concept we limit ourself in two dimensions. The Hilbert curve is a SFC used to preserve spatial locality as much as possible. Thus, the Hilbert curve has the important property that consecutively ordered points are adjacent in space. This is shown in Fig. I 5, where the H-curves of the first three orders are shown in two dimensions. At each step the curve is obtained by replicating the basic four quadrants, these are denoted by I, II, III, IV (clockwise, from left bottom).

At each n order the H-curve can be obtained by the previous n-1 order according to the prescriptions

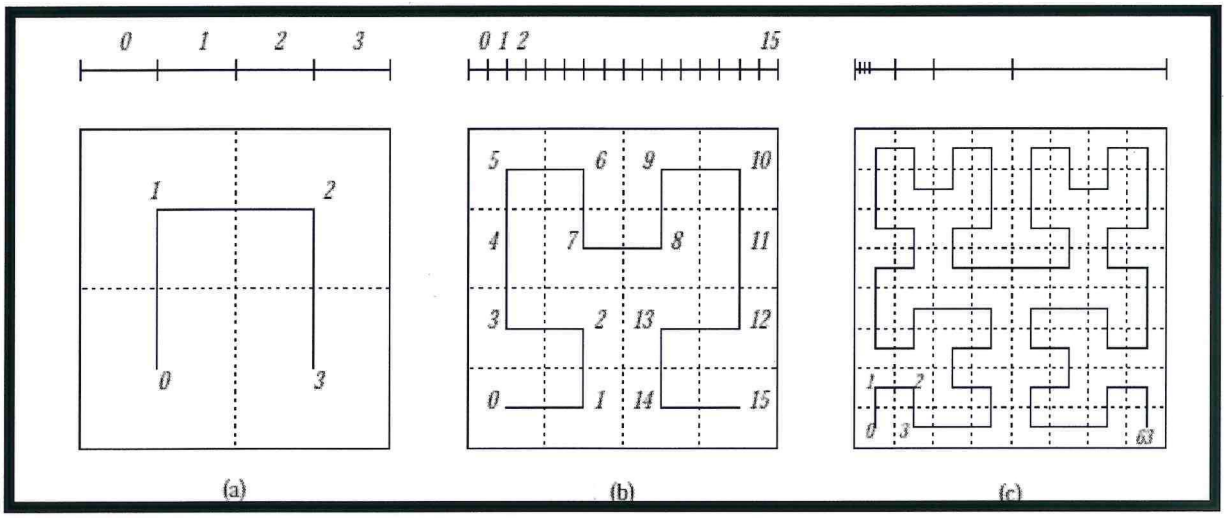


Fig.I5: Approximations of the Hilbert curve in two dimensions

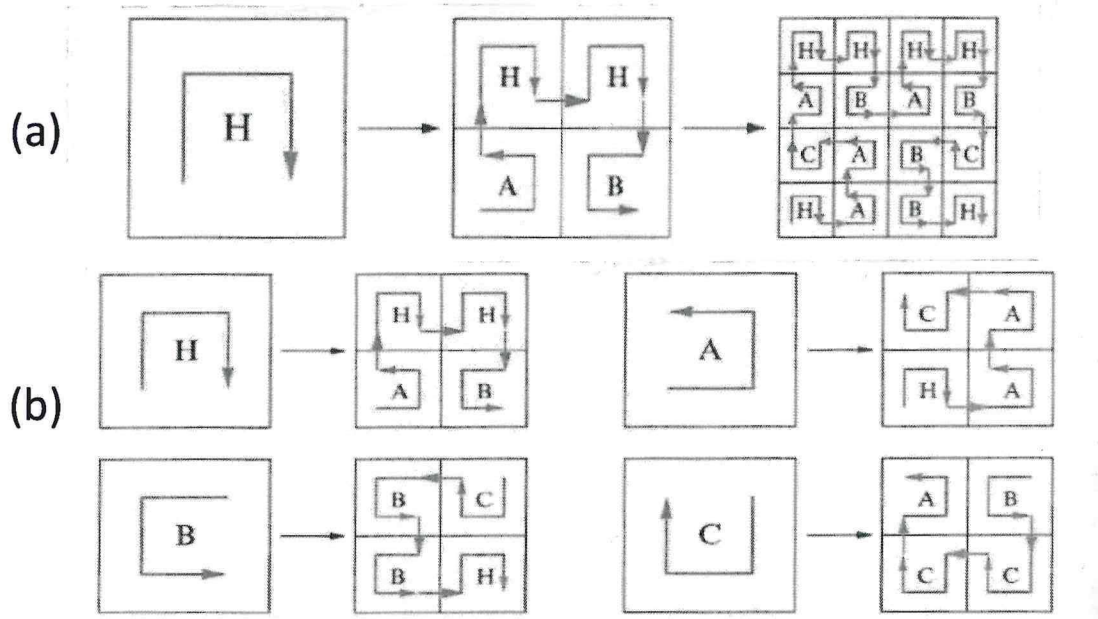


Fig.I6: (a) Basic patterns of the first three iterations of the Hilbert curve.
(b) Replacement scheme of the basic patterns in the iteration of the Hilbert curve.

I: rotate −90° + reflex
II and III: replica
IV: rotate +90° + reflex
} Hilbert curve rules

Because all of the rotations and sense computations are relative to the previously obtained ones in a particular quadrant, a repetition of these steps, starting from the basic curve (Fig. I 5 a), gives the curves Fig. I 5 b and Fig. I 5 c.
This can be put in a more systematic form using a grammer description.

I.4.1 Description of the Hilbert curve using grammer

To construct the iterations of the Hilbert curve, we recursively subdivide squares into subsquares, and construct the recursive patterns using the given prescriptions of rotation and reflexion.
There are four basic patterns; which we denote by H, A, B and G (Fig. I 6 (a)).

The replacement scheme is a closed scheme and there are no additional patterns.
The grammar is defined by the following rules.

- A set of non-terminal symbols $\{H, A, B, C\}$
  The non-terminal H shall be distinguished as the start symbol.
- A set of terminal symbols: $\{\uparrow, \downarrow, \leftarrow, \rightarrow\}$
  which describe the basic patterns and will also determine the connections between the subsquares by an iteration of the Hilbert curve.
- A set of production rules for the patterns.

I.7
$$\begin{cases} H \leftarrow A \uparrow H \rightarrow H \downarrow B \\ A \leftarrow H \rightarrow A \uparrow A \leftarrow C \\ B \leftarrow C \leftarrow B \downarrow B \rightarrow H \\ C \leftarrow B \downarrow C \leftarrow C \uparrow A \end{cases}$$

The productions determine for each basic pattern how it has to be replaced during the refinement.

An additional derivation rule: to construct a string from our grammar we are allowed to replace all non-terminal symbols at once in a derivation step. This ensures that a uniform refinement level is retained through the construction of the corresponding curve.

For the example of Fig. IG one obtains

$$
\begin{aligned}
&H \leftarrow A \uparrow H \rightarrow H \downarrow B \\
&\leftarrow H \rightarrow A \uparrow A \leftarrow G \uparrow A \uparrow H \rightarrow H \downarrow B \rightarrow A \uparrow H \rightarrow \\
&H \downarrow B \downarrow G \leftarrow B \downarrow B \rightarrow H
\end{aligned}
\qquad \text{I.8}
$$

where the arrow symbols describe the iterations of the Hilbert curve and can be interpreted as plotting instructions.

I.5 Mathematical description of the Hilbert curve

We will use sequences of nested 1D and 2D intervals To mathematically define the Hilbert curve. The construction shall proceed with constant velocity: the covering of one quarter of the parameter interval will correspond To the covering of one subsquare, this is true also for the recursive refinements. For a given parameter t we construct a sequence of intervals as follows (Fig. I 7):

. The 0-th level interval is $[0,1]$

. For each interval $[a, a+\Delta]$, its successor is chosen as one of the intervals

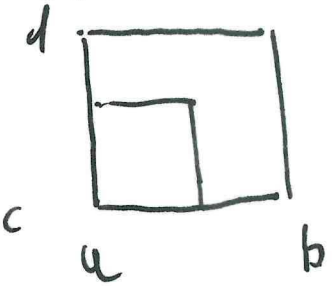
$$\left[a + (m-1)\frac{\Delta}{4}, \; a + m\frac{\Delta}{4} \right] \quad \text{where } m = 1,2,3,4 \qquad \text{I.9}$$

. The parameter t is contained within one of the sub-intervals

For example, the parameter $t = 1/3$ will generate the sequence

$$[0,1] \quad \left[\frac{1}{4}, \frac{2}{4}\right] \quad \left[\frac{5}{16}, \frac{6}{16}\right] \ldots \qquad \text{I.10}$$

In parallel with the sequence of nested intervals we construct a sequence of nested subsquares, where each subsquare will be one quarter of the parent square. Thus, the square $[a,b] \times [c,d]$ generates $\left[a, \frac{a+b}{2}\right] \times \left[c, \frac{c+d}{2}\right]$ and so on.



DEFINITION OF HILBERT CURVE:

Let us define the parameter representation $h(t)$ via the following algorithm. For each parameter $t \in [0,1]$, a sequence of nested intervals

$$I \supseteq [a_1,b_1] \supseteq \ldots \supseteq [a_n,b_n] \qquad I.11$$

exists, such that each interval is obtained by splitting its precedessors into four parts of equal size. Any such sequences of intervals can be mapped one by one to a sequence of nested 2D intervals. The iteration of the Hilbert curve determines which 2D interval is the image of which interval

. The constructed 2D nested intervals will converge to a uniquely defined point in $Q = [0,1] \times [0,1]$ - This point should be $h(t)$ The image of $h: \mathcal{I} \to Q$ is then a space filling curve.

I.5.1 quartenary representation

To represent these concepts into a formal mathematical description of the Hilbert mapping let us introduce the quartenary representation of the parameter $t \in [0,1]$

$$t = \sum_n 4^{-n} q_n \qquad \text{I.12}$$

then, for $t = 1/4 \to q_1 = 1 \; q_{n>1} = 0$; $t = 1/8 \to$

$q_1 = 0 \quad q_2 = 2 \quad q_{n>2} = 0$

$t = 1/4 = [0.1]$ $\qquad t = 1/8 = [0.01]$

and so on.

The quartenary digits represent the numbering of the subsquares in each iteration.

To compute the quartenary representation of any given parameter t one has to repeatedly multiply the parameter by 4

and cut off the integer part. The procedure ends when the remainder is zero

$$\text{for } t>0 \begin{cases} y = \text{floor}(4 \cdot t) \\ r = 4 \cdot t - y \\ t = r \end{cases} \qquad \text{I.13}$$

Turning back to the previous examples, if $t = 1/3$ one has at the first step $y_1 = 0$ $r = 1/2$ and at the second iteration $y_2 = 2$, $r = 0$ so that $\frac{1}{3} = [0_4.02]$. However $t = 2/5$ is a never ending series with $t = 2/5 = [0_4.121212\ldots]$.

I.5.2 Recursive mapping

Another fundamental property to be used when constructing Hilbert curves is self-similarity: at each iteration n the Hilbert curve consists of four parts each one filling one of the four subsquares, these curves are obtained by the curve at level n-1 by performing operations of scaling, rotation and traslation.

For each subsquare it is then necessary a trasformation operator that maps the unit square into the correct subsquare and performs the trasformations. We will denote these operators by W_0, W_1, W_2 and W_3 (Fig. I.7). The quartenary digits will tell us the intervals that contain the parameter t and will then determine the subsquares of the images, therefore indicating which operator must be applied. In matrix notation the four operators are

$$W_0 = \begin{pmatrix} 0 & 1/2 \\ 1/2 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \qquad W_1 = \begin{pmatrix} 1/2 & 0 \\ 0 & 1/2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ 1/2 \end{pmatrix}$$

$$W_2 = \begin{pmatrix} 1/2 & 0 \\ 0 & 1/2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 1/2 \\ 1/2 \end{pmatrix} \qquad W_3 = \begin{pmatrix} 0 & -1/2 \\ -1/2 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 1 \\ 1/2 \end{pmatrix}$$

I.14

All the operators perform a downsizing by one half, because all of the subsquares have $\frac{1}{2}$ of the original side length. In addition, each operator performs the following operations:

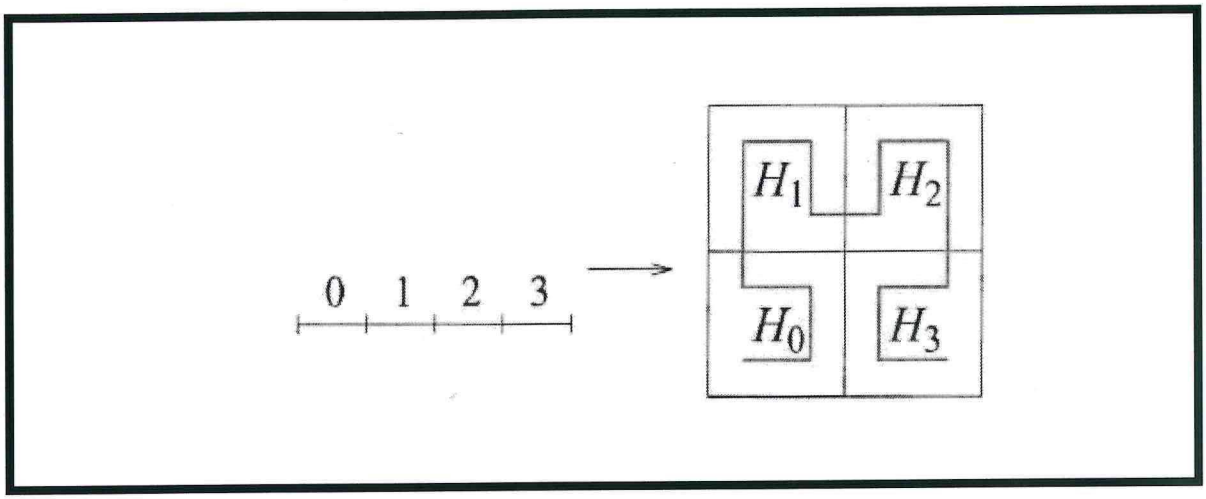


Fig.I7: Correspondence between intervals and transform operators for the Hilbert curve

Function hilbert – compute the Hilbert mapping (fixed number of digits)

```
Function hilbert (t, depth)
    Parameter: t: index parameter, t ∈ [0, 1]
               depth: depth of recursion (equiv. to number of quaternary digits)
begin
    if depth = 0 then
        return (0,0)
    else
        // compute next quaternary digit in q
        q := floor(4*t);
        r := 4*t - q;
        // recursive call to h()
        (x,y) := hilbert (r,depth-1);
        // apply operator H_q
        switch q do
            case 0: return ( y/2, x/2);
            case 1: return ( x/2, y/2 + 0.5);
            case 2: return ( x/2+0.5, y/2+0.5);
            case 3: return ( 1.0-y/2, 0.5-x/2);
        endsw
    end
end
```

Fig.I8: Computation of the Hilbert mapping in two dimensions up to a given recursion level depth

H_0: swaps x and y coordinates, which correspond to a clockwise rotation of 90°

H_1 and H_2: retain the original curve plus a traslation

H_3: an anticlockwise rotation by 90%, which is obtained by the swap $(x,y) \to (-y,-x)$ plus the traslation

We are now in position to calculate the values $h(t)$ for a given parameter t. This is accomplished via the following steps:

1) Compute the quartenary representation for the parameter $t = 0_4.y_1y_2\ldots$ The parameter lies in the y_1-th interval and $h(t)$ in the y_1 subsquare.

2) In the y_1 subsquare parameter t correspond To the local parameter $\tilde{t} = 0_4.y_2y_3\ldots$ We therefore compute $h(\tilde{t})$, which is the image of $\tilde{t}$ in the y_1 subsquare relative to the scaled down and rotated Hilbert curve in that subsquare.

Transform the local coordinates of $h(\tilde{t})$ into the coordinates in the original square

This is performed by applying the operator H_{q_1} to $h(f^2)$.

The recursion terminates when the quartenaries q_i which are left are zero:

$$h(f) = h(0_4 \cdot q_1 q_2 \dots) = H_{q_1} \circ H_{q_2} \circ H_{q_3} \dots H_{q_n}\begin{pmatrix}0\\0\end{pmatrix} \qquad \text{I.15}$$

The rhs generalizes to an infinite series in the case of infinite quartenaries.

An example of a pseudo code to compute an Hilbert mapping in 2D is given in Fig. I.8, where function Hilbert is a recursive implementation of the algorithm. This is applied for a given t and recursion level.

I.6 Hilbert curve in higher dimensions

The concept of SFC can be extended into higher dimensions. For example, in 3 dimensions, a 3D Hilbert curve is constructed by dividing the cube into 8 subcubes and order them so that sub cubes which are

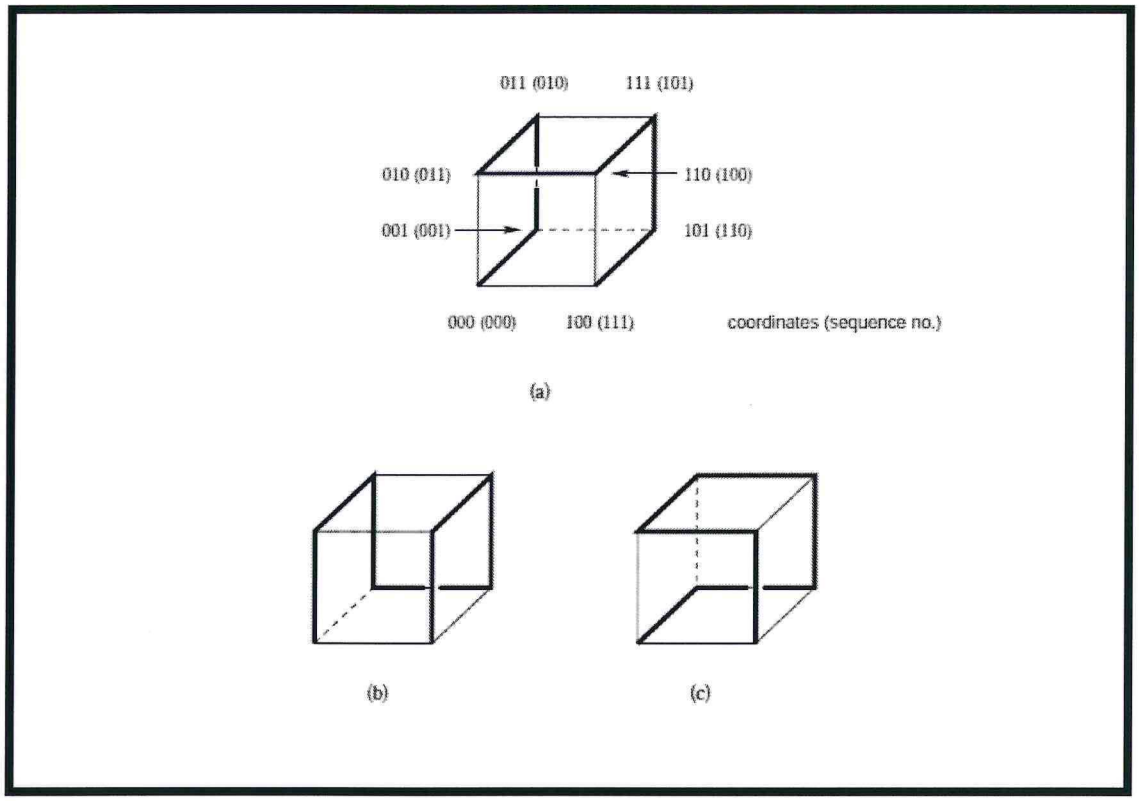


Fig.I9 : Hilbert first order curves in three dimensions

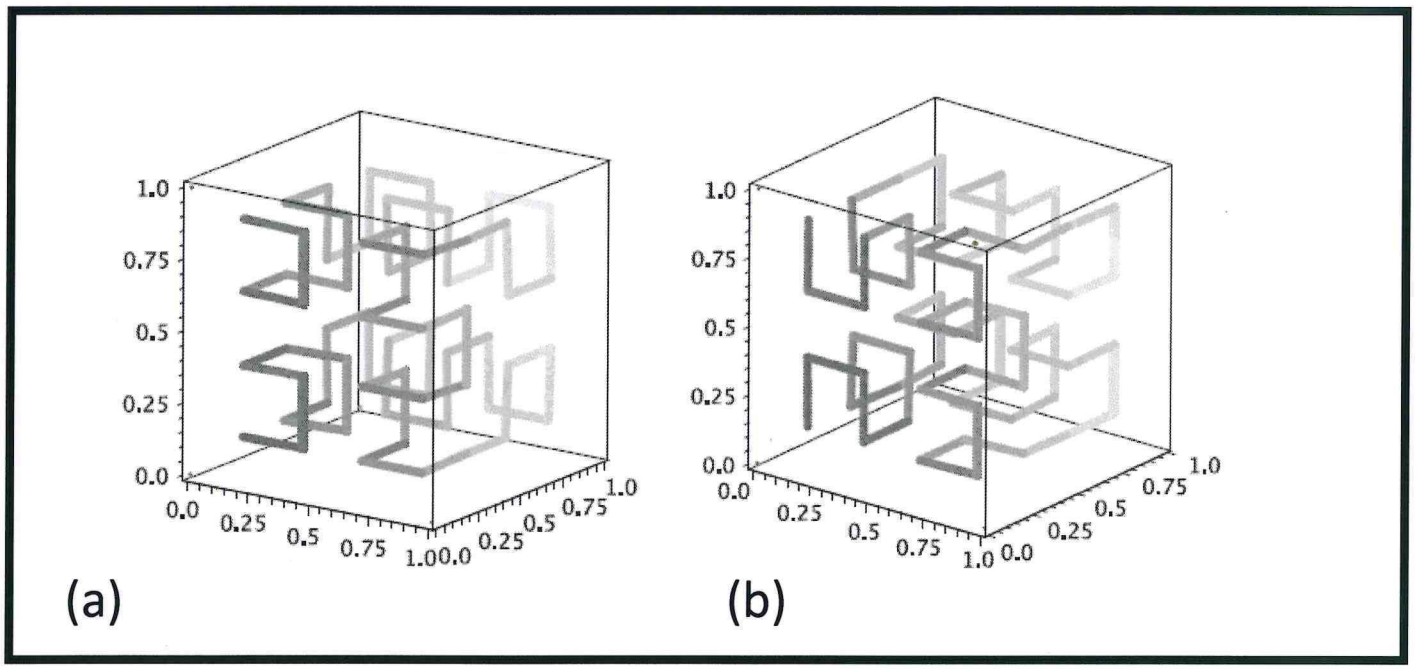


Fig.I10: Two possible variants of second order Hilbert curves

mapped to from adjacent subintervals share a common face. Three examples of first order curves are given in Fig. I 9. Note that a mapping not necessarily describes a symmetric curve (Fig. I 9c). Here we will consider as basic pattern the example of Fig. I 9a. Unlike the 2D case, Hilbert curves in 3D have a greater choice of constructing second order curves once that a first order curve has been drawn. Two possible variants are illustrated in Fig. I 10, where both curves have identical start and end points. The existence of these alternatives complicates the problem of the algorithm construction for a 3D Hilbert curve. In fact, it can be shown that there are 1,536 different variants to construct a single Hilbert curve. This number is obtained by considering $2^8 = 256$ different orientations times the six possible alternative paths.

$$H_0\begin{pmatrix}x\\y\\z\end{pmatrix}=\begin{pmatrix}\frac{1}{2}y+0\\\frac{1}{2}z+0\\\frac{1}{2}x+0\end{pmatrix} \qquad H_1\begin{pmatrix}x\\y\\z\end{pmatrix}=\begin{pmatrix}\frac{1}{2}z+0\\\frac{1}{2}x+\frac{1}{2}\\\frac{1}{2}y+0\end{pmatrix}$$

$$H_2\begin{pmatrix}x\\y\\z\end{pmatrix}=\begin{pmatrix}\frac{1}{2}z+\frac{1}{2}\\\frac{1}{2}x+\frac{1}{2}\\\frac{1}{2}y+0\end{pmatrix} \qquad H_3\begin{pmatrix}x\\y\\z\end{pmatrix}=\begin{pmatrix}-\frac{1}{2}x+1\\-\frac{1}{2}y+\frac{1}{2}\\-\frac{1}{2}z+0\end{pmatrix}$$

$$H_4\begin{pmatrix}x\\y\\z\end{pmatrix}=\begin{pmatrix}-\frac{1}{2}x+1\\-\frac{1}{2}y+\frac{1}{2}\\\frac{1}{2}z+\frac{1}{2}\end{pmatrix} \qquad H_5\begin{pmatrix}x\\y\\z\end{pmatrix}=\begin{pmatrix}-\frac{1}{2}z+1\\\frac{1}{2}x+\frac{1}{2}\\-\frac{1}{2}y+1\end{pmatrix}$$

$$H_6\begin{pmatrix}x\\y\\z\end{pmatrix}=\begin{pmatrix}-\frac{1}{2}z+\frac{1}{2}\\\frac{1}{2}x+\frac{1}{2}\\-\frac{1}{2}y+1\end{pmatrix} \qquad H_7\begin{pmatrix}x\\y\\z\end{pmatrix}=\begin{pmatrix}\frac{1}{2}y+0\\-\frac{1}{2}z+\frac{1}{2}\\-\frac{1}{2}x+1\end{pmatrix}$$

Fig.I11: Set of operators corresponding to the Hilbert curve of Fig.I10(a)

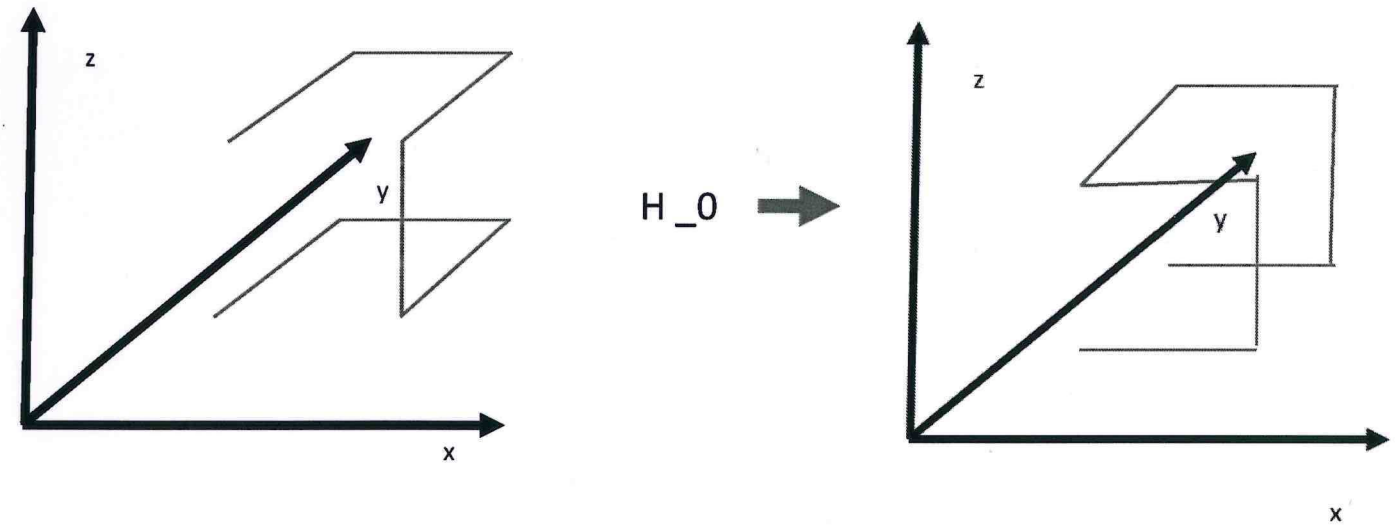


Fig.I12: Application of the operator H_0 to the Hilbert root cube

In order to translate the 2D mathematical description of the Hilbert curve in a 3D framework, one must resort to an octal representation of the parameter t. This being the natural choice as due to the recursive division into eight subcubes at each iteration. For a given parameter $t = 0_y.k_1 k_2 \ldots$ we must determine the operators H_k for the following representation

$$h(0_y.k_1 k_2 \ldots) = H_{k_1} \circ H_{k_2} \circ \ldots \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \quad \text{I.16}$$

where there are eight operators $(H_0 \text{ to } H_7)$ which describe the transformation of the unit cube into one of the eight subcubes. For the example of Fig. I 10 (a) we define the corresponding operators which perform the operations of rotation, reflexion and translation (Fig. I 11). For example, the H_0 operator reduces the unit curve into one-half and performs the swap $(x\,y\,z) \rightarrow (y, z, x)$. This operation is illustrated in Fig. I 12,

where H_0 is applied To the left (parent) curve and produces the right subcube.

II. TREE ACCESS METHODS

II.1 Introduction

A fundamental topic in many scientific applications is the retrieval of data from large data sets. Several examples include data mining, geographical information systems, multimedia information systems, time-series analysis. A characteristic of spatial data sets is that they are usually large and that the data are quite often distributed in a irregular manner. A spatial access method must then support an efficient spatial selection, for example range queries or nearest neighbor queries. A spatial access method then needs to take into account both spatial indexing and clustering techniques. The spatial indexing is necessary to avoid that the selection criterion is applied to every object in the database.

Moreover, many data sets are so large that they are stored on the hard disk and not in the computer memory, clustering is then needed to regroup these objects in order to avoid page faults. The clustering implies that objects which are close in reality according to some property, are also close in memory when stored.
To solve the problems posed by requiring an efficient spatial access method many algorithms have been developed. Here we describe tree methods.

II.2 General definitions

In computer science a tree is a data structure in which data access is organized hierarchically. The tree consists of a root node and potentially many levels of sub-trees. The algorithm finds data by repeatedly making choices at decision points called nodes.

A node can have as few as two branches or as many as several dozens. The branches are termed children. The maximum number of children per node is the order of the tree. The maximum number of access operation required to reach the desiclered record is the depth of the tree.
In a tree records are stored in location called leaves, i.e. a node with no children. A parent is a node that has at least a child, whilst siblings are those nodes with the same parent. Several examples of tree structures are given in Fig. II 1. We now consider a partial example to illustrate the basic principles that lie behind the use of trees. Let us consider a binary tree, i.e. a tree in which each node can have a maximum of two children. We now consider the following set of numbers:

(25, 63, 13, 72, 18, 32, 59, 67)

which are stored in a list.

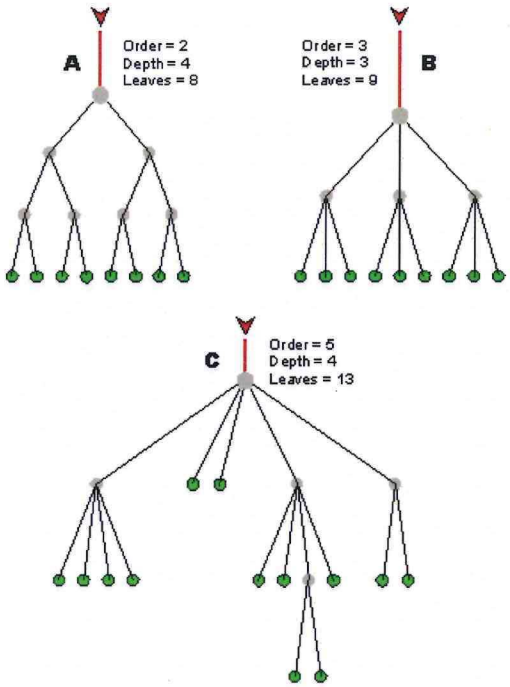


Fig.II1(a): Three examples of tree structures. Roots are at the top and leaves at the bottom, nodes are shown as dots.

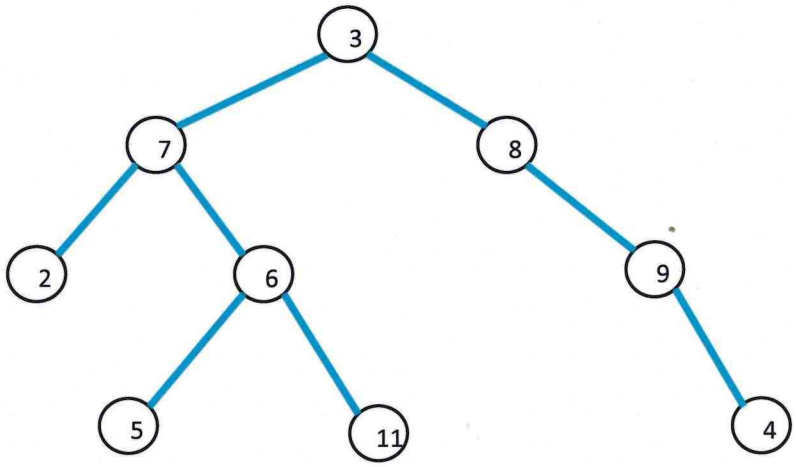


Fig.II1(b): An example of an ordered tree: 3 is the root node, 2 and 6 are siblings, 5 is a leaf, 2 and 6 are children of 7, 7 is the parent node of (2,6).

To search for a particular item we have to go through the list, thus the complexity would be $\sim O(n)$. Let us now store the data using a binary tree data structure. The data are stored following the rule that at any node the stored value is higher than those stored in the left child nodes and higher than those stored in the right child nodes (Fig. II 2). The root node is 59, but its choice is arbitrary. With this arrangement any search is taking at most four steps. The level of a node is the level of its parent plus one, the root has level zero.

II.3 Quadtrees and octrees

The quadtree is a generic name for all kinds of trees that are built by recursive division of space into four quadrants. More formally, by the term quadtree (Samet 2006) we mean a spatial data structure based on a disjoint regular partitioning of space.

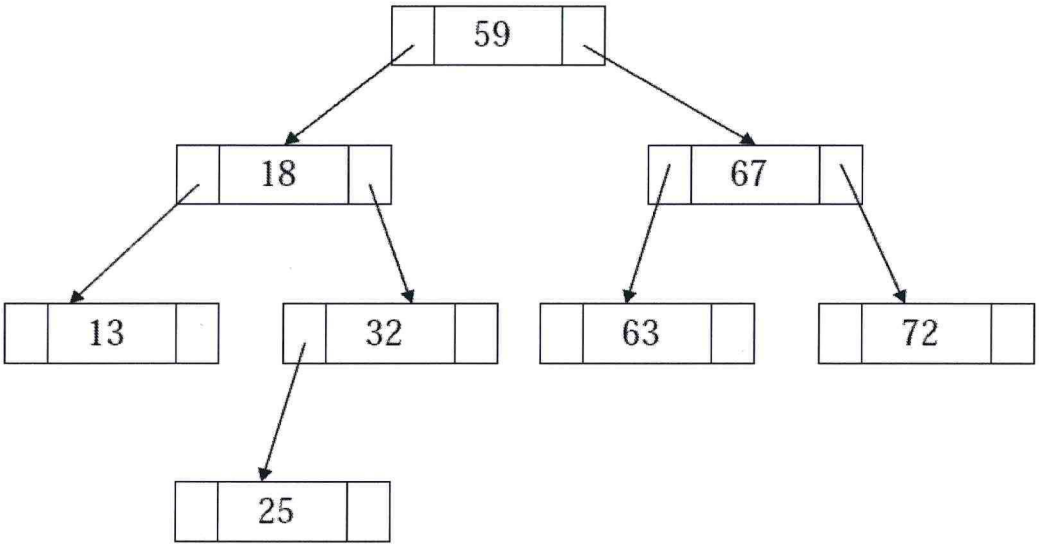


Fig.II2: Example of a binary tree

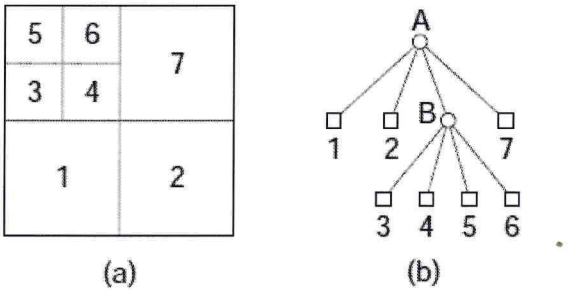


Fig.II3: Block decomposition (a) and tree structure (b) of a simple quadtree.

Each quadtree block (or cell) covers a portion of space that forms a hypercube in d dimensions, usually with a side length that is a power of two. Quadtree blocks may be further divided into 2^d sub-blocks of equal size. Fig. II 3 illustrates a simple example of a quadtree enclosing a set of points, together with the corresponding tree structure. The domain enclosing all the points forms the root of the tree. The subdivision is stopped when a predetermined number of points is left in the subdomain. Similarly, an octree can be defined in three dimensions or more (hyperoctree). We will generally speak of quadtrees. In quadtrees, the manner in which any subregion is bisected is independent on the location of the points.

This implies that the size of the quadtree is sensitive to the spatial distribution of points. Quadtrees were originally designed for the purpose of indexing two or three dimensional space. In practice, they are useful when d is small ($d \leq 4$-5). This is due to the fact that the fan-out of internal nodes is exponential with d, which renders their use impractical. For large values of d kd-trees are best suited (see later). Quadtrees are widely applied in many scientific problems where a regular partition of 2D space is a necessary condition to apply a solution method.

A family of quadtrees which are useful in geographical applications are the so called PM-quadtrees (Fig. II 4). This is a quadtree based dynamical data structure for storing objects

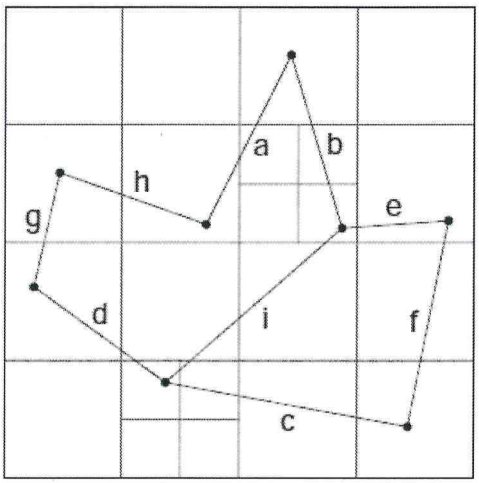


Fig.II4: A PM quadtree for line segments with a splitting threshold of 2, i.e. a sub-block is splitted if the number of objects is > 2.

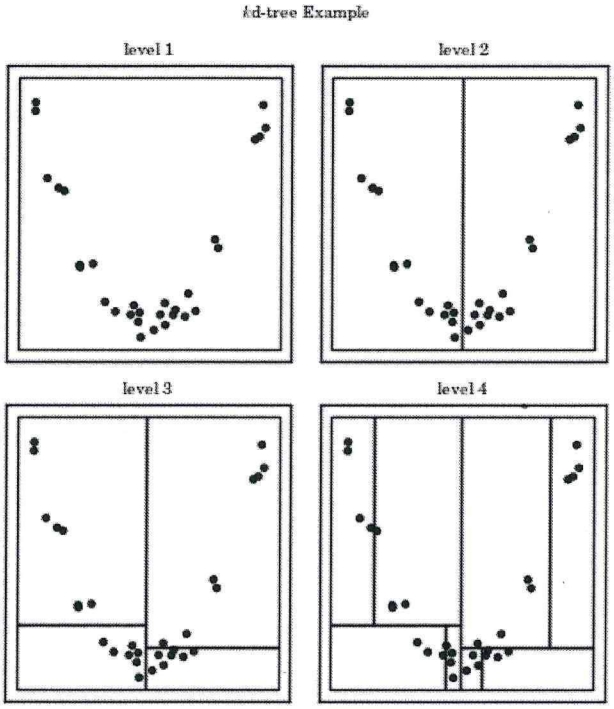


Fig.II5: Construction of a simple kd-tree, the partition ends when there are four or less points left in the node.

such as polygon maps. The vertices are stored as in the quadtree. The edges are segmented into q-edges which completely fall within the squares of the leaves. There are different classes of q-edges according to the intersections with square boundaries. The PM quadtree provides a reasonably efficient data structure for performing various operations: overlap of two maps, range searching, edge intersection.

II.4 kd-trees

A kd-tree is a data structure that is used for storing k-dimensional points. It is particularly useful in problems of range queries or nearest-neighbor searches. The root of the tree corresponds to the whole region of interest.

Each internal node of the kd-tree contains one point and also corresponds to a rectangular region. At each level of the iteration the rectangular region is divided into two parts by the x (y) coordinate of the chosen point. The coordinate axis is chosen alternate according to the depth of the tree. So that, for example in 2D, level 0 : x , level 1 : y, level 2 : x and so on. The point is chosen as the median of the sub-distribution, so that half of the remaining points will belong to the left subtrees and half to the right. The iteration continues until there are no points left and a leaf is reached. This method leads to a balanced tree in which each leaf node is about the same distance from the root.

II.5 R-tree family

The R-tree was introduced with the purpose of storing objects with a spatial extent, like lines, polygons, etc. Each internal node of the tree has the form (MBR, ptr) where MBR is the Minimum Bounding Rectangle that encompass all the MBR's of its child nodes in space. Examples of MBRs are given in Fig. II.6. Each node of an R-tree (with the expection of the root node) of class (m, M) contains between m and M pairs, with $m \leq \lceil \frac{M}{2} \rceil$ M is called the branching factor and is chosen to suit paging and I/O disk buffering. Fig. II.7 shows a simple example of an R-tree.

It must be noted that the MBR's that correspond to different nodes may be overlapping

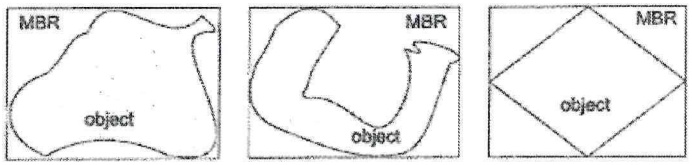


Fig.II6: A selection of polygons and their corresponding MBRs

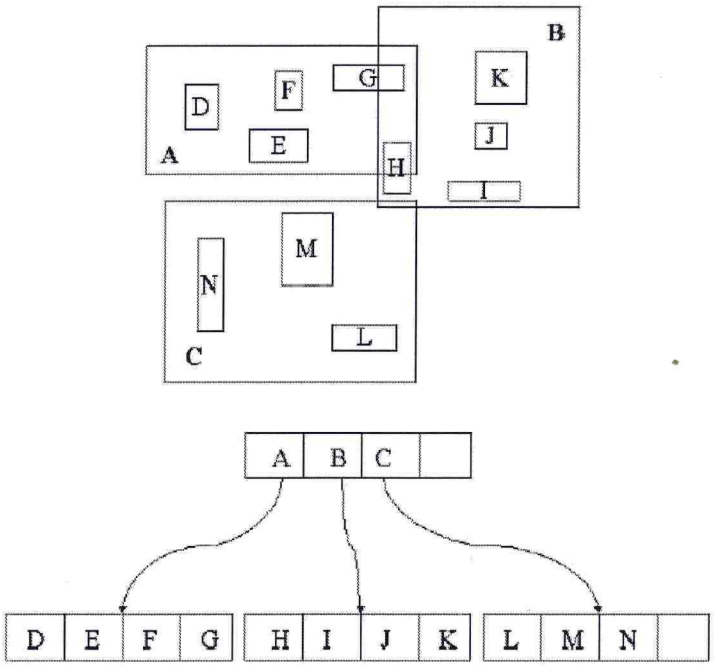


Fig.II7: An R-tree example. Data rectangles D through N are stored in leaf nodes, whereas A,B,C are hosted at the root nodes.

This implies that an MBR can be included in many nodes but can be associated to only one of them. As a result, a spatial search may visit many nodes, before finding a particular MBR object.
This has prompted to consider several variants to the original algorithm, here we mention only some variants.
The R^+-tree was proposed as a structure which avoids overlapping and thus improves performances during range queries. To avoid overlapping between MBRs at the same level, inserted objects have to be divided into two or more parts. A side effect is that a specific object may be duplicated and stored in various nodes. Moreover, no minimum space per node can be given. However, numerical experiments demonstrate a more efficient range searching.

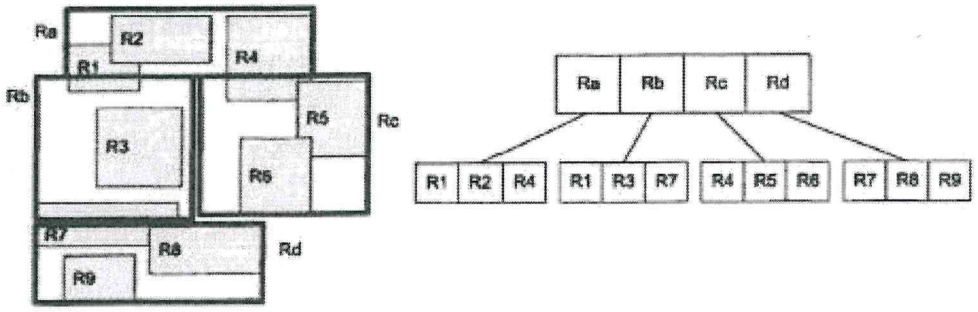


Fig.II8: An example of an R^+ - tree.

The Hilbert R-tree uses the centre point of the Hilbert value of the MBR To organize the objects. These are grouped according to their Hilbert value, a parent node contains the largest Hilbert value of the objects and the union of all the MBRs of the objects. The iteration is repeated on the higher level until a root node is reached. The drawback of the Hilbert R-tree is that it is possible that two objects which are close in space have different Hilbert values. Therefore, the two objects will not end up in the same node despite being close in space. This in turn implies large MBRs and reduces performances.

THE NEAREST NEIGHBOR PROBLEM

III. 1 Introduction

Let us consider a set P of N points p_i $i=1,..N$ Given a query object q, which may be a set member or not, we want to find the point p which satisfies $dist(p,q) = MIN$ over all $p \in P$. Here, $dist(a,b)$ is a metric distance. More generally, we may ask for the k nearest objects instead of a single one. Thus, the k-NN query asks for a list of objects with increasing distance from the query point q.

The Nearest Neighbor search is a fundamental geometric problem important in a variety of applications including data mining, graphics, statistics, computer vision etc.

For example, an important application of the NN problem is with a geographical database, where an user is interested in retrieving a certain location with respect to the given position.

The most common metric distance used for NN query is the L_p norm

$$L_p(x,y) = \left(\sum_j |x_j - y_j|^p \right)^{1/p} \quad \text{II.1}$$

where $\bar{x}, \bar{y}$ are the two coordinate points of the two objects in n-dimensions.

However, it must be stressed that not all of the NN queries make use of metric distance to establish a proximity between objects in the database. For example, let us consider an image database. In this case, one might be interested in the query "find all the images which are similar to a given image". The image properties will be traslated into a multidimensional vector and the NN processing will be performed according to some proximity criterion. For example: 'find all the images for which the levels of the three RGB channels differ by the query image less than....'. In this section we will consider only NN searches based on spatial distances.

Moreover, the k-NN problem can also be reformulated by considering a window, or range, query instead of a single query point q. In this case we seek all the points of P which fall within the window boundaries. In this case one refers to the window query problem.

We will provide several examples based on different methods in which the set of points are organized. In the first example we assume that data points are indexed using spatial access methods, in the other two the points are mapped into space filling curves.

III.2 Range search using tree methods

It is relatively easy to implement a window query using a tree. Here we illustrate the principle assuming that the data have been indexed using a quadtree, but the algorithm works as well using a kd-tree or an R-tree.

Fig. III. 1 shows an example of a window query superimposed onto a tree. Let us first assume that, starting from the root node, the space containing all the points has been subdivided into cells of decreasing size until there are no points left to examine.

A search algorithm then will work as follows: starting from the root cell check for $\Omega(c) \cap W$, if yes then open the root cell and repeat for the four sub-cells: {bl, br, tl, tr} (bottom/top left/right) continue the iteration until the cell leaf is a point and add it to the query list.

Here $\Omega(c)$ is the spatial region defined by cell boundaries $\{L_i, U_i\}$ where $i = x, y$ and L_i, U_i are the lower and upper boundaries, respectively. The window query $W = \{..\}$ is defined in a similar way.

The overlap criterion $\Omega(c) \cap W$ is true if the overlap condition (Fig. III. 2) is satisfied simultaneously along both the axes.

The procedure is illustrated in algorithm III. 1 where the search is performed by a first call to Window Query (W, root)

A III. 1 The Window query algorithm

Algorithm Window Query (window, cell)

```
/*  {L1,U1;L2,U2} : cell boundaries
/*  {QL1,QL2; UL1, UL2} : window boundaries
overlap = (L1 ≤ QU1 AND U1 > QL1) AND
          (L2 ≤ QU2 AND U2 > QL2)
if overlap is true then
   open childs of a cell
   for child in {bl, br, te, tr}
/* b=bottom  t=top  l=left  r=right
   if child is a particle
         add to the search list
      otherwise
         window query (window, child)
      end if
   end for
end if
```

There are many variants of the window query algorithm using spatial access methods.

We refer to Samet (2006) and Papadopoulos and Manolopoulos (2005) for a comprehensive discussion.

III.3 Nearest Neighbor search using SFC

Several works have been conducted to solve the query range utilizing SFC. The SFC approach is preferable in high dimensional space than tree access methods because of the growth of the nodes with the dimension d. Among different SFC curves, it has been found that the Hilbert curve posses the best clustering properties, that is the capability to preserve locality when a linear mapping is applied between multi-dimensional data and the one-dimensional space (Jagadish 1990, Mokbel et al. 2003). Many authors have proposed to use Hilbert curve, see Jin et al. (2011) for a recent algorithm and references cited therein. However, another SFC which is used to solve a range search problem is the Z-order, or Morton, curve.

The curve and its properties have been introduced in sect. II. Here we will present two examples of NN search, the first is an algorithm based on the $\underline{z}$-order curve whilst the second uses the Hilbert curve.

III.3.1 Nearest Neighbor search based on $\underline{z}$-order

A range search algorithm based on the Morton order, or $\underline{z}$-order, has been first proposed by Tropf & Herzog (1981, TH) and more recently by Connor and Kumar (2010). Here we will describe the implementation of TH.

Let us consider a set of points defined by integerized coordinates $0 \le x \le 8$ and $0 \le y \le 16$. We construct the $\underline{z}$-key of order 5 by interlacing the bits of x and y. Following TH, we adopt the convention (X, Y) so that the key of a point is given by

$$\underline{z} = x_5 y_5 x_4 y_4 \dots x_1 y_1 \qquad \text{III.2}$$

where x_5 (y_5) is the most significant bit of the x (y) value.

Fig. III.3 (Fig. 9 of TH) shows the data points together with their key values (note that in TH key values are termed record code).

For example, the point $(X,Y) = (8,16)$ has bits (01000, 10000) and key value 0110000000 = 384.

Data are arranged in a binary search tree, i.e. at each level of the tree all the points with key values k_i less than k(node) are on the left and all the points with $k_i > k$(node) are on the right, where k(node) is the key value of the node.

We now consider a search range defined by the window $(X,Y) = (3,5)$ and $(X,Y) = (5,10)$ with corresponding key values $(3,5) = (00011, 00101) =$

$= 00000\ 11011 = 27 =$ MIN and $(5,10) = (00101, 01010) =$

$= 00011\ 00110 = 102 =$ MAX.

These two key values define the two staircases of Fig. III.3, i.e. all the key values less than 27 and greater than 102.

If the search range now begins from the root node, in this example the point with

key value 58, the search then examines the minimum and maximum code in the range and if it is between the two both the subtrees must be examined, otherwise only one subtree must be opened. The algorithm is denoted by TH as range(P).
However, this algorithm is rather inefficient, since there are many keys between 27 and 102 which do not lie within the query range. For this reason TH proposed to improve the algorithm as follows: for a given cut off key k_c, in this case 58, we seek the maximum $k_{MAX} = MAX\{k_i\}$ of all the key values k_i which lie within the query and subject to the constraint $k_{MAX} \leq k_c$. In this case $k_{MAX} = 55$, TH adopt the term LITMAX for this key. Similarly, we seek the key $k_{min} = MIN\{k_i\}$ within the query such that $k_{MIN} > k_c$. This is the key value 74, which TH call BITMIN. These two keys introduce a new staircase in Fig. III 3 (the dashed line). The advantage is now that the search is done between MIN and LITMAX, and between

BITMIN and MAX, therefore pruning from the search range many keys previously considered. The new search algorithm is described by TH in Range (P, RMIN, RMAX) and with respect range (P) now the key range is dynamically shrinking as subtrees are opened. Obviously the most important aspect of the new algorithm is the computation of <ITMAX and BIGMIN. In TH this computation is described in detail, here we refer to a more simple description of the procedure (Raime, technical white paper 2012).

In binary form the key values of the boundary box take the values

$$\begin{cases} (3,5) = (00011, 00101) = 0000011011 \\ (5,10) = (00101, 01010) = 0001100110 \end{cases}$$

III-3

We first look at their 2 values and find the first most significant bits that differ in their value. In this case it is z_7 which corresponds to the y_4 bit. This tell us if we are looking at a vertical or horizontal division. In this case it is a y bit

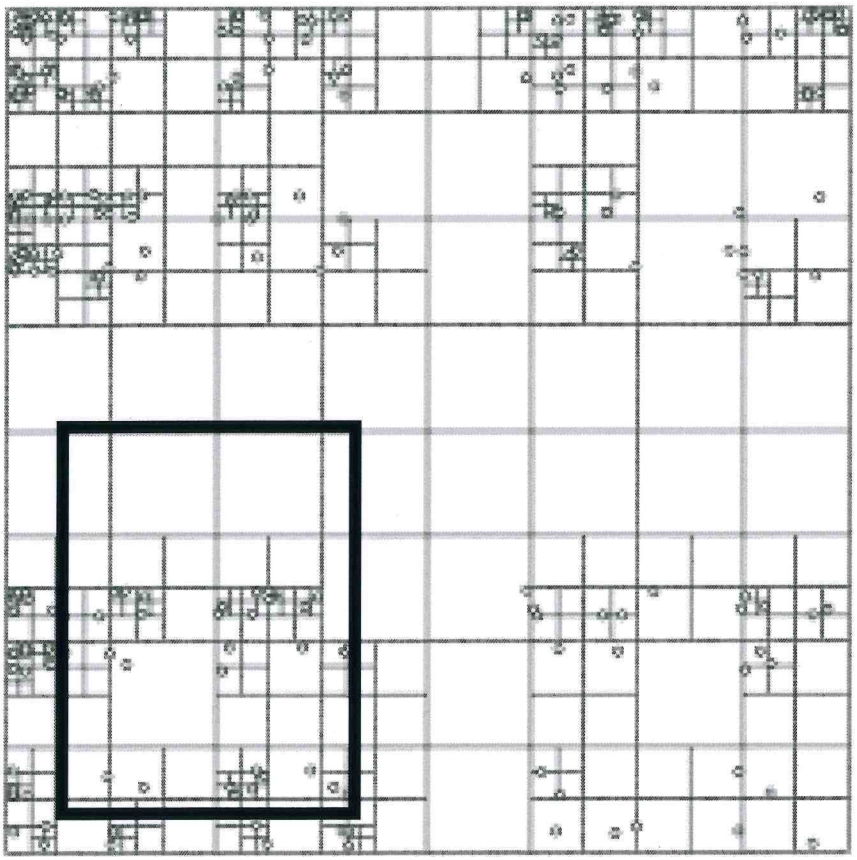


Fig.III1: Data points are organized into a tree and the window defines the search boundaries.

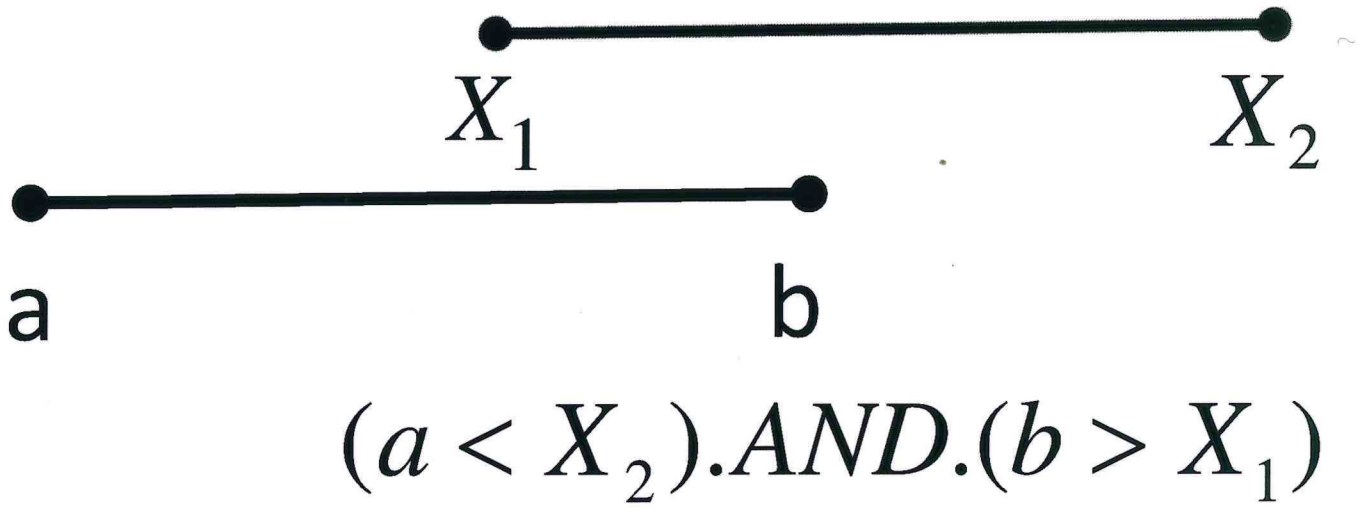


Fig.III2: The overlap criterion in one dimension between two segments

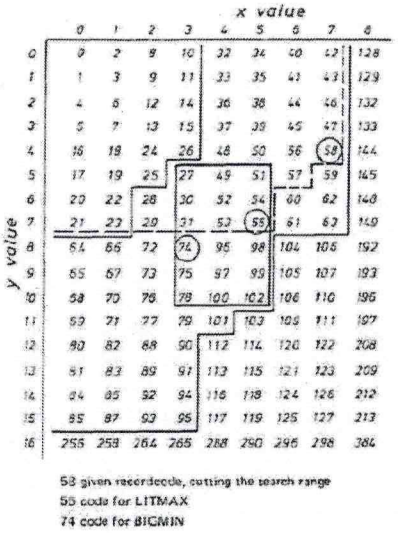


Fig.III3: Point distribution and key values as in Tropf & Herzog (1981). The search range is defined by the query (X,Y)=(3,5) and (X,Y)=(5,10). LITMAX=55 and BIGMIN=74 are the keys used to reduce the search range, which is cut by the key 58.

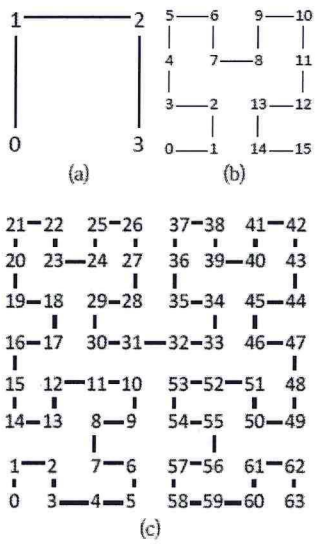


Fig.III4: Approximation of the Hilbert curve of first, second and third order. For the latter the range of the integer coordinates is 0 to 7, for example H=63 corresponds to (0,7).

which translates into a horizontal split. The upper and lower x boundaries are identified from the keys MIN and MAX : $x = 5$ and $x = 3$. We must now find the y key values of LIT MAX and BIGMIN. These are the highest key value in the upper part and the lowest key value in the lower part of the range cut.

LIT MAX's y value is determined as 'add the most significant bits' from the bounding box value, followed from a zero and then 1's.

BIGMIN is now defined as 'add the most significant bits' followed by 1 and then 0's.

For the considered example

III.4

$$\begin{cases} 27\text{'s } y = 00101 \\ 102\text{'s } y = 01010 \end{cases}$$

The first position (from the left) is zero and does not count, the other bits are all different so that

III.5

$$\begin{cases} \text{LIT MAX}(y) = 00\,111 \\ \text{BIGMIN}(y) = 01\,000 \end{cases}$$

The two key values are then

III.6

$$\begin{cases} \text{LIT MAX} = (00101, 00111) = 0000110111 = 55 \\ \text{BIGMIN} = (00011, 01000) = 0001001010 = 74 \end{cases}$$

If we want to further subdivide the range search, then the computation of LiTMAX and BiSMiN proceeds in a similar manner.

III.3.2 Nearest Neighbor search using the Hilbert curve

Many efforts have been dedicated over the years to solve the Nearest Neighbor search using SFC (Jin et al. 2011). Specifically, the Hilbert curve has been considered by many authors because of its superior clustering properties. Here we will illustrate as a specific example the work of Lawder & King (2001, LK). The reader is referred to the original paper, here we outline the general strategy of the algorithm.

Fig. III.4 shows the first three steps to construct a Hilbert curve for the 2-dimensional case. At unity square is initially divided into four sub quadrants and a first order curve is drawn through the centre points. Consecutive ordered quadrants share a common edge. In the next step each of the quadrants is divided into 4 sub quadrants and in each one a 'scaled-down' first order curve is drawn and

connected to the others. Fig III 4(c) shows the third step. The rule to construct the Hilbert curve at each step have been given in Sect. I. After k steps one has an approximation of a space filling curve of order k which passes through 2^n quadrants, the centre points are regarded as points in a space of finite granularity. We now describe how the mapping between one and n dimensions is calculated and how to utilize the curve to organize data.

- Hilbert curve mapping

The recursive way in which space is partitioned during the construction of the Hilbert curve can be expressed in terms of a tree structure as in Fig. III. 5. Each bottom node corresponds to a third order curve and at any level k a collection of nodes describes a curve of order k. Thus, the root node corresponds to the first order curve. We call a binary sequence number of a quadrant within a node as a key (derived-key in the terminology of [K]) or a quadrant number, the concatenated single

bit coordinates of a point

are called an h-point. We illustrate the mapping from the coordinates of a point to its key-value H with the example of point P in Fig. III 6 (Fig. 3 of [6]).

The H-key value of the point P = (6, 4) = (110, 100) is unknown and is designated by the string '??????'.

step 1 - the first top bits of the coordinates of point P form the h-point (11). From Fig. III 5 the h-point 11, at the tree level 1, is located in the quadrant number 10 (in bold)

The H-key of point P is now '10????'.

step 2 - Now concatenate the next lower (middle) bits to form the h-point 10. From Fig. III 5, at the tree level 2, this corresponds to the quadrant 11. The H-key of P is now '1011??'.

step 3 - the bottom bits of point P are concatenated to form the h-point 00. At the tree level 3 the quadrant identified

is 10, the final H-key is then $H = 101110 = 46$.

- Application of the Hilbert curve

Let us consider the H-curve of a set of points. The curve is cut into continuous sections each corresponding to a page storage in a data file. Pages are indexed by the keys of the points which are called 'page-keys'. The page-key of a page (= segment of the H curve) is the lowest H-key of any datum points lying on the page. The first page is an exception and is indexed by the page key of value zero. Pages are then ordered by their page keys. No page contains a point whose H-key is greater than that of any point of the subsequent page.

- Querying strategy

We want now to retrieve all the points lying within the hyper-rectangle query region defined by the lowest and upper bound coordinates $(l_1, l_2 \ldots l_4)$ and $(u_1, u_2 \ldots u_4)$.

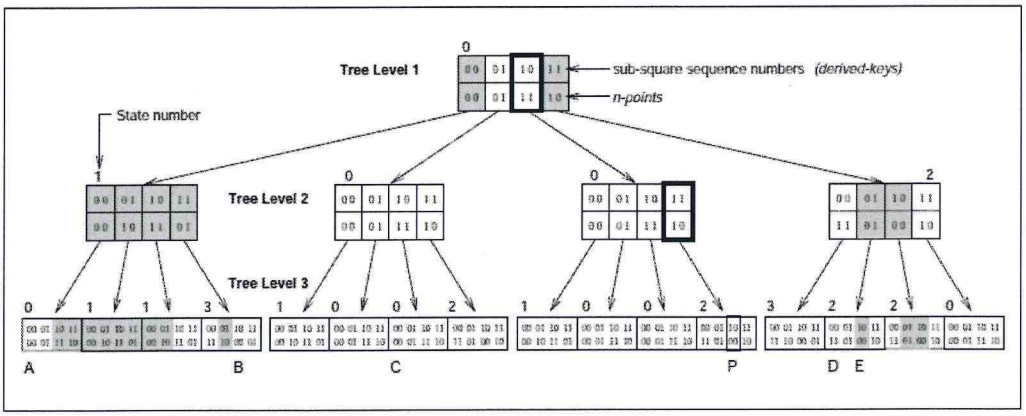


Fig.III5: A tree representation of a third order Hilbert curve in 2 dimensions (from Fig. 2 of Lawder & King , 2001).

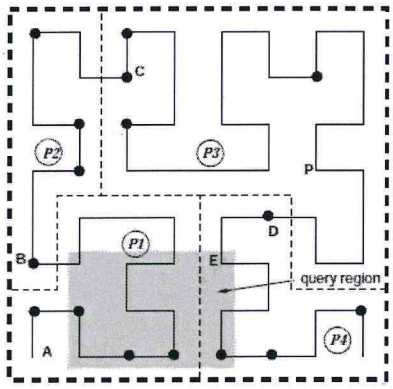
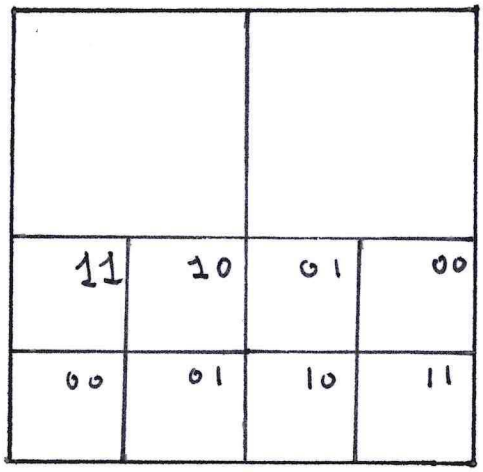


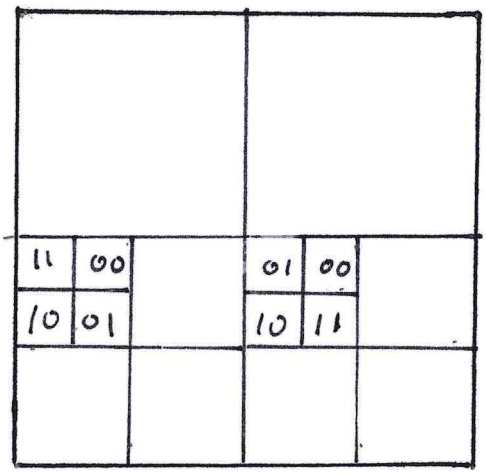
Fig.III6: A query region is superimposed onto the third order curve Hilbert curve of Fig.III4(c). A, B, C and D are 'page-key', E is the 'next-match' to B from the query. P has coordinates (6,4)= (110,100) and H=101110=46. The lower and upper boundaries of the query window are (1,0) and (4,2). From Fig.3 of Lawder & King (2001).

A query region intersects one or more curve sections each corresponding to a page. The Hilbert curve may enter, leave and re-enter a query region a number of times. In the example of Fig. III 6 the Hilbert curve is cut into four pages (P1, P2, P3 and P4) with page-keys A, B, C and D. These have coordinates

$$\begin{cases} A = (0,0) = 0 \\ B = (0,2) = 14 \\ C = (2,6) = 24 \\ D = (5,3) = 52 \end{cases}$$ III.7

where $(X,Y) = H$. The query region has boundaries defined by coordinates (1,0) and (4,2). Pages to be searched are identified in ascending order by the function calculate_next_match() = cnm(). The first time cnm() is called it returns the lowest H-key of any point lying within the query region. The page which contains this point is searched for all the points lying within the query region. The second time cnm() is called finds the lowest H-key of a point lying within the query region which is equal or greater





H key sequence: 0011-( ) = 12, 13, 14, 15

1101-( ) = 52, 53, 54, 55

Fig. III 7: Quadrants which are examined at each level during the calculation of next-match. The search region (dashed line) and the points which are relevant are depicted in the upper right square. At tree level 3 the H-key sequence of the numbered quadrants are indicated.

then the page-key of the page next in order to the page just searched. In the example of Fig. III 6 the H-key is E, since the page-key under consideration is B (page-key of P2). We call this key the next-match. When it is found the page which contains this its corresponding point is searched and all the points within the query region are retrieved. The process is iterated until no further next-match are found. It must be stressed that, unlike in the Z-order curve, the knowledge of the keys of the lower and upper bounds of the query regions is not useful since, as can be seen from Fig. III 6, the Hilbert curve may leave and re-enter the query region many times.

- Querying algorithm

The querying algorithm is now illustrated for the example of Fig. III 6. The query region is represented with a shaded area and the quadrants of the tree representation of Fig. III 5 which intersect the query region are shown in Fig. III 5 as shaded. We assume that cnm() has already been called once and

that page P1 has been identified and searched. If we now call chm() a second time then we are searching for the key (next-match) which is equal or minimally greater than the page key of the page next of P1. In this case is the page-key B = 001110 of page P2. This is the current key. The next match lying within the query region is the key $\bar{c} = (4,2) = 110110 = 54$. The search begins at the root of the tree and terminates at a leaf member which is the next match.

The value of next-match is initially unknown and is designated by '??????'. We have complemented the example of Fig. III 6 with Fig. III 7, which illustrates at each step the quadrants under consideration.

step 1- tree level 1: A binary search shows that quadrants 00 and 11 intersect the query region. Quadrant 00 is the lowest quadrant intersecting the query region. It also contains the current key since the top two bits of B are 00. Quadrant 00 may therefore contain

the next-match. We tentatively set next-match= '00????'.

Step 2 - tree level 2 - The search now proceeds to the node in level 2 pointed by quadrant 00. All of the quadrants intersect with the query region, but the lower two and the top right are excluded because the middle two bits of B are 11. The value of next-match is set to '0011??'.

step 3 - tree level 3. We now search in the sub quadrants identified by the quadrant 11 in step 2. The binary search shows that quadrant 01 intersect the query region, but the bottom two bits of the current key are 10, therefore placing the current key in the lower left quadrant. The next-match therefore does not lie in the node currently being searched.

Step 4 - tree level 1. The search now back tracks up to the root node. We restart by examining the quadrant next in order which intersect the query region: 11. next_match is now '11????"

Step 5 - tree level 2. The search proceeds down at level 2 and a binary search shows that quadrants 01 and 10 intersect with the query region. The lowest in order is 01. next-match is set to '11012?'.

Step 6 - tree level 3. At this level the binary search shows one sub-quadrant intersecting the query: 10. The next match key is now '111010' = (4,2) = K6.

Once that we have the value of next key we can find the corresponding page (P4 in the example) and retrieve all the points lying within the query.

REFERENCES

SECTION I and II

Bader, M., 2013, “Space-Filling Curves An Introduction with Applications in Scientific Computing”, Springer

Samet, H.M., 2006, "Foundations of multidimensional and metric data structures", Elsevier

Section III

Papadopoulos, A.N. and Manolopoulos, M., 2005, "Nearest neighbor search a database perspective", New York Springer

Articles
Connor, M. & Kumar, P., Fast construction of k-nearest neighbor graphs for point clouds, Visualization and Computer Graphics, IEEE Transactions on, vol. 16, no. 4, pp. 599 –608, 2010.

Jagadish, H.V., 1990, Linear Clustering of Objects with Multiple Attributes, Proceedings of the ACM SIGMOD Conference on Management of Data, ACM Press, New York, NY, pp. 332 – 342

Jin, Y. , Jing Dai, J. & Lu, C.-T., 2011, Efficient Range Query Using Multiple Hilbert Curves, Current Trends and Challenges in RFID, Ed. Prof. Turcu, C.

Available from:http://www.intechopen.com/books/current-trends-and-challenges-in-rfid/efficient-range-query-using-multiple-hilbert-curves

Lawder. J.K. & King, P.J.H., Querying Multi-Dimensional Data Indexed Using the Hilbert Space-Filling Curve, ACM SIGMOD Record, Vol. 30, No. 1, March, 2001, pp.19–24

Mokbel, M. F., Aref, W. G., & Kamel, I. (2003) Analysis of Multi-dimensional Space-Filling Curves, GeoInformatica, Vol. 7, No. 3, pp. 179-209

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

http://raima.com/wp-content/uploads/COTS_embedded_database_solving_dynamic_pois_2012.pdf