Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®

Information Systems 30 (2005) 205–226

www.elsevier.com/locate/infosys

Neighbor-finding based on space-filling curves ☆

Hue-Ling Chen, Ye-In Chang*

Department of Computer Science and Engineering, National Sun Yat-Sen University, Lien Hai Road, 70, Kaohsiung 804, Taiwan, ROC

Received 24 July 2002; received in revised form 14 November 2003; accepted 22 December 2003

Abstract

Nearest-neighbor-finding is one of the most important spatial operations in the field of spatial data structures concerned with proximity. Because the goal of the space-filling curves is to preserve the spatial proximity, the nearest neighbor queries can be handled by these space-filling curves. When data are ordered by the Peano curve, we can directly compute the sequence numbers of the neighboring blocks next to the query block in eight directions in the 2D-space based on its bit shuffling property. But when data are ordered by the RBG curve or the Hilbert curve, neighbor-finding is complex. However, we observe that there is some relationship between the RBG curve and the Peano curve, as with the Hilbert curve. Therefore, in this paper, we first show the strategy based on the Peano curve for the nearest-neighbor query. Next, we present the rules for transformation between the Peano curve and the other two curves, including the RBG curve and the Hilbert curve, such that we can also efficiently find the nearest neighbor by the strategies based on these two curves. From our simulation, we show that the strategy based on the Hilbert curve requires the least total time (the CPU-time and the I/O time) to process the nearest-neighbor query among our three strategies, since it can provide the good clustering property.

Keywords: Hilbert curve; Neighbor-finding; Peano curve; RBG curve; Space-filling curves; Spatial proximity

1. Introduction

With the proliferation of wireless communications and the rapid advances in geographic information systems (GIS), a very common spatial query is the "Nearest Neighbor Query" which seeks for the objects residing more closely to a given object. For example, when driving a car in a highway, the driver may ask where the nearest gas station is. In other words, nearest-neighbor-finding is one of the most important spatial operations in the field of spatial data structures regarding proximity.

Basically, spatial data consist of objects in space made up of points, lines, regions, rectangles, and data of higher dimensions. Access methods are required to support efficient manipulation of the multi-dimensional spatial objects in the secondary storage. Several schemes [1–7] for the linear mapping of a multi-dimensional space have been proposed for various applications, such as access methods for traditional databases, GIS and spatio-temporal databases. However, in spatial applications, the structure of databases become larger and more complex and means for extracting valuable

☆ Recommended by Nick Koudas.

*Corresponding author. Tel.: +886-7-525-2000x4334; fax: +886-7-525-4301.

E-mail address: changyi@cse.nsysu.edu.tw (Y.-I. Chang).

0306-4379/$ - see front matter © 2003 Elsevier Ltd. All rights reserved.
doi:10.1016/j.is.2003.12.002

information become more sophisticated [4]. The design of multi-dimensional access methods is difficult compared to one-dimensional cases because there is no total ordering that preserves spatial locality [5]. Once a total ordering is found for a given spatial database, one can use any one-dimensional access method which may yield good performance for multi-dimensional queries. Thus, what is needed is a mapping from a higher dimensional space to a one-dimensional (1D) space. In this mapping, the elements which are close in space are mapped into nearby points on the line to preserve the spatial locality. A space-filling curve passes through every point in the space once. It is one kind of mapping to give an one-to-one correspondence between the coordinates and the sequence numbers of the points on the curve [4].

Because the goal of the space-filling curves is to preserve spatial proximity, they can handle nearest-neighbor queries. Some examples of space-filling curves are the Peano curve, the RBG curve and the Hilbert curve. The *Peano curve* [6,7] (or the *Z*-order curve) has the *bit shuffling* property, which means interleaving the bits from two coordinates of the point in base 2 in the two-dimensional (2D)-space. An improvement of the Peano curve is the *Reflected Binary Gray-code* (*RBG*) *curve* [1,2] which uses Gray coding on the interleaved bits to reduce random accesses on the disk for range queries. Based on the Peano curve, the *Hilbert curve* was proposed with superior *data clustering* properties, which means that the locality between objects in the multi-dimensional space is preserved in the linear space [3–5,8,9].

When the data are linearly ordered by the Peano curve, we can directly compute the sequence numbers of the neighboring blocks next to the query block in eight directions based on its bit shuffling property, even the *order* is greater than 1. (Note that the order of a curve deals with how many sequence numbers be necessary to uniquely define that curve.) But with the data ordered by the RBG curve or the Hilbert curve, neighbor-finding would be complex. However, we also observe that there is some relationship between the RBG curve and the Peano curve, as with the Hilbert curve. Therefore, in this paper, we first show how these space-filling curves are derived and discuss the relationships between them. Next, we describe the effects on nearest-neighbor queries by using different curves. Then, we show the way to find the nearest neighbor by our strategy based on the Peano curve, and present the rules for transformation between the Peano curve and others. Finally, we compare the performance of our strategy for the nearest-neighbor-finding based on different space-filling curves.

The rest of paper is organized as follows. In Section 2, we briefly describe three space-filling curves, the Peano curve, the RBG curve, the Hilbert curve, and their spatial properties. In Section 3, we present our strategies for the nearest-neighbor-finding, based on the Peano curve and the transformation rules between the Peano curve and the others. In Section 4, we describe the simulation experiment performed to compare different strategies, and results obtained therefrom. Finally, we provide conclusions for this paper.

2. Space-filling curves

The design of multi-dimensional access methods is difficult compared to one-dimensional cases because there is no total ordering that preserves spatial locality [5]. One way is to look for a mapping which preserves spatial proximity at least to some extent. In this section, we discuss the properties of the Peano curve, the RBG curve, and the Hilbert curve.

2.1. Properties

A space-filling curve orders points linearly to preserve the distance between two points in the 2D-space. This means that points which are close in space and represent similar data should be stored together in the linear sequence. Some examples of space-filling curves are the Peano curve, the RBG curve and the Hilbert curve [1–3,5–9]. In general, a space-filling curve starts with a basic path on a k-dimensional square grid of side 2. The path visits every point in the grid exactly once without crossing itself. It has two free ends

which may be joined with other paths. The basic curve is said to be of order 1. To derive a curve of order i, each vertex of the basic curve is replaced by the curve of order $i-1$, which may be appropriately rotated and/or reflected to fit the new curve [3].

For the Peano curve, the 1D-number (one-dimensional sequence number) of a point is obtained by bit shuffling on the X and Y coordinates of the point in the 2D-space. The basic Peano curve of order 1 for a 2×2 grid, denoted as P_1, is shown in Fig. 1(a). To derive higher orders of the Peano curve, we replace each vertex of the basic curve with the previous order curve. Figs. 1(b) and (c) show the Peano curves of order 2 and 3, respectively [3,8].

For the RBG curve, numbers are coded into binary gray-codes such that successive numbers differ in exactly one bit position. Faloutsos [1–3] observed that in the RBG curve, the difference in only one bit position has a relationship with locality. He proposed a formula to transform the Peano curve into the RBG curve such that the property of locality can be preserved. The basic RBG curve of order 1 for a 2×2 grid, denoted as R_1 is shown in Fig. 2(a). Higher orders of this curve are derived by reflecting the previous order curve over the x-axis and then over the y-axis. Figs. 2(b) and (c) show the RBG curve of orders 2 and 3, respectively [1–3,8].

The Hilbert curve is a space-filling curve 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 [4], as shown in Fig. 3(a). As in the case of the previous two curves, it replicates in four quadrants. While replicating, the lower left quadrant is rotated clockwise 90°, the lower right quadrant is rotated anti-clockwise 90°, and the sense (the direction of traversal) of both lower quadrants are reversed. The two upper quadrants do not rotate and change the sense. Thus we obtain Fig. 3(b). Because all rotation and sense computations are relative to previously obtained ones in a particular quadrant, a repetition of this step gives rise to Fig. 3(c) [1–3,8,9].

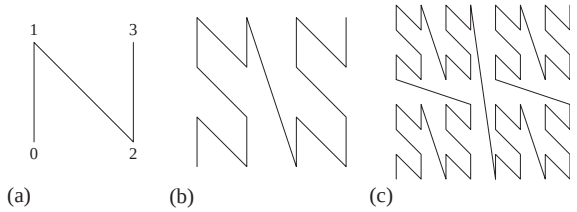

Fig. 1. Peano curves of order: (a) 1; (b) 2; (c) 3.

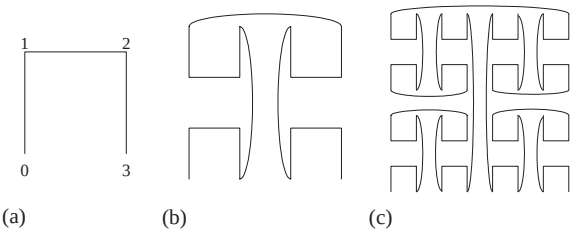

Fig. 2. RBG curves of order: (a) 1; (b) 2; (c) 3.

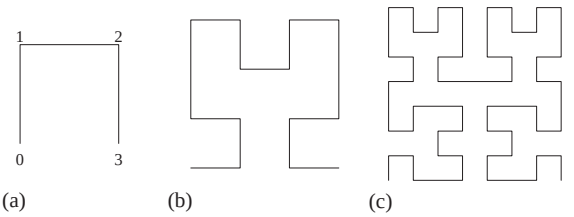

Fig. 3. Hilbert curves of order: (a) 1; (b) 2; (c) 3.

Space-filling curves are based on the assumption that any attribute value can be represented with some fixed number of bits. In the 2D-space, there are two attributes: the horizontal and vertical directions which are called the X- and Y-axis, respectively. The whole space is split into equal-sized sub-regions, and the direction of splitting alternates between the X- and Y-axis. A vertical split through the middle of the space amounts to discrimination in the value of x_0. If the resulting sub-regions are split horizontally, this corresponds to discrimination on the value of y_0. At this point, there are four equal-sized sub-regions, corresponding to the possible values of x_0 and y_0.

In general, each split on the X- or Y-axis is characterized by one bit. After splitting k times in each direction, a grid can be specified by $2k$ bits of two coordinates, $(x_{k-1}x_{k-2}\cdots x_1x_0, y_{k-1}y_{k-2}\cdots y_1y_0)_2$. The bit string obtained by bit

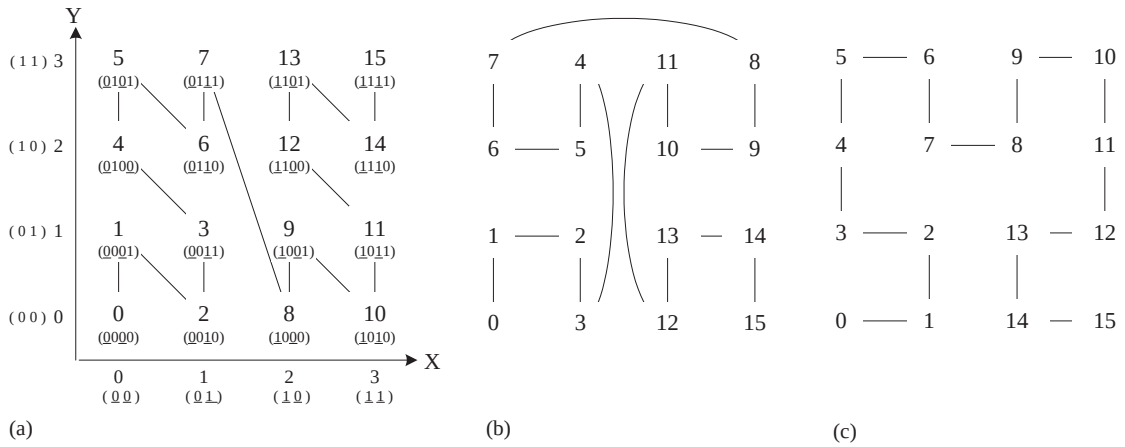

Fig. 4. Space-filling curves of order 2: (a) *z*-ordering: Peano curve; (b) *r*-ordering: RBG curve; (c) *h*-ordering: Hilbert curve.

shuffling on these $2k$ bits uniquely identifies one sub-region. The *z*-value is represented as the decimal value of the bit string. This means that there is a concise description of the shape, size and position of the sub-region. All points inside the sub-region have the same coordinates as the sub-region [7]. Consider the point or the region with $X = (1)_{10} = (0\ 1)_2 = (x_1x_0)_2$ and $Y = (3)_{10} = (1\ 1)_2 = (y_1y_0)_2$ in Fig. 4(a), where $(\,)_2$ represents the binary form of the number, and $(\,)_{10}$ represents the decimal form of the number. The point or the region has the bit string $(0\ 1\ 1\ 1)_2 = (x_1y_1x_0y_0)_2$, obtained by bit shuffling, which represents the splitting process on the *X*-and *Y*-axis. In the decimal representation, the *z*-value of the bit string $(0\ 1\ 1\ 1)_2$ is equal to $(7)_{10}$. Orenstein [6,7] used the *z*-values and *z*-ordering to refer to the ordering of the Peano curve (Fig. 4(a)), which is a total ordering of elements that combines with the highly constrained splitting policy and preserves proximity. Similarly, *r*-ordering and *r*-values are used for the RBG curve, and *h*-ordering and *h*-values are used for the Hilbert curve, as shown in Figs. 4(b) and (c), respectively.

3. Nearest-neighbor-finding

Nearest-neighbor-finding is one of the most important spatial operations in the field of spatial data structures which is concerned with proximity [10,11]. Because the goal of the space-filling curves is to preserve spatial proximity, nearest neighbor queries can be handled by the space-filling curves. An example of the nearest-neighbor query shown in Fig. 5 is to find the closest hydrant to the point of a fire by using three different curves. The entire grid is a map of a neighborhood, which is stored in a database. There exists *F*, which represents the point of the fire, and *H*, which represents a hydrant in the area. The lines represent street blocks and the numbers represent the orders that are stored in the database. In order to find the nearest data point *H* to the query point *F*, we briefly describe the conventional approach as follows.

In the conventional approach [3], we first compute the sequence number of *F*, and find *H* which has the closest sequence number (either in the preceding or succeeding order) to that of *F* along the curve path. Then, we calculate the actual distance *d* in the 2D-space between the query point and the retrieved data point, and issue a range query centered at the query point with radius *d*. In this way, we need to check all points and return the closest point to the query point. In Fig. 5(a) (Peano curve of order 2), we calculate the sequence number of the query point, i.e., F_9. We find the proceeding and succeeding sequence numbers on the curve path until one of the points corresponds

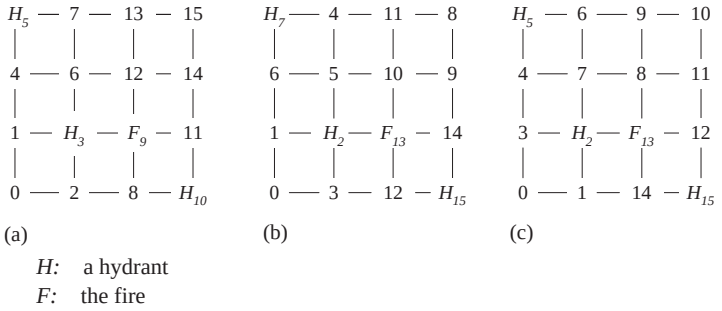

Fig. 5. An example of the nearest-neighbor query which is represented by three kinds of ordering: (a) *z*-ordering; (b) *r*-ordering; (c) *h*-ordering.

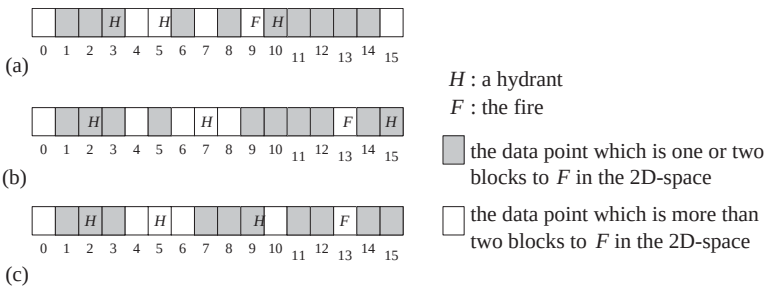

Fig. 6. The shaded regions of 1D diagrams illustrate the hydrants H which are within two blocks of the fire F in the 2D-space, as shown in Fig. 5: (a) *z*-ordering; (b) *r*-ordering; (c) *h*-ordering.

to H. In this case, H_{10} is the first one found to be the nearest neighbor to F_9. The distance from H_{10} to F_9 is 2 ($= d$) blocks in the 2D-space shown in Fig. 5(a). Then, we check all points which are within two blocks of F_9 to see if any closer hydrant exists by performing a range query. From Fig. 5(a), we see that a hydrant H_3 exists which is only one block away from F_9 in the 2D-space. But in the 1D-diagram shown in Fig. 6(a), which represents the ordering of the curve, we need to check 6 points (from 9 to 3) for the range query to obtain the data point H_3, which is the nearest neighbor (one block away from F_9) in the 2D-space. Similarly, in Fig. 5(b) (the RBG curve of order 2), H_2 is the nearest neighbor, which is only one block away from F_{13} in the 2D-space. But in the 1D-diagram shown in Fig. 6(b), we need to check 11 points (from 13 to 2) for the range query to obtain the data point H_2. In Fig. 5(c) (the Hilbert curve of order 2), H_2 is the nearest neighbor, which is only one block away from F_{13} in the 2D-space. But in the 1D-diagram shown in Fig. 6(c), we need to check 11 points (from 13 to 2) for the range query to obtain the data point H_2. Thus, when the data are linearly ordered by the space-filling curves, it is time-consuming in the nearest-neighbor-finding to check all the data in the database by using the conventional approach.

In this section, we first show how to find the nearest neighbor based on the property of the Peano curve. Then, we present the neighbor-finding algorithm based on the RBG curve and the Hilbert curve. We also present the transformation rules between the Peano curve and the other two curves in the neighbor-finding process.

3.1. Based on the Peano curve

In each of the space-filling curves, the block is characterized by the unique value, i.e., the

sequence number. The Peano curve uses the z-value to linearly order the blocks. As mentioned in Section 2, the z-value in base 2 of the Peano curve can be easily obtained by interleaving the bits from x_i bits and then y_i bits of two coordinates in base 2. (Note that the value in base 2 is represented with the base 2 bits, i.e., 0 or 1. A base 2 bit represents that bit times a power of 2.) The relationship between the z-value and the coordinates can be shown in Fig. 4(a). The value in parentheses denotes the value in base 2, and the mark (_) denotes the bit obtained from the X coordinate in base 2. However, it is not so straightforward to obtain the two coordinates in base 2 from the r-value of the RBG curve or from the h-value of the Hilbert curve in the same way. To obtain the two coordinates in base 2 from the r-value requires the reflection recursively on the basic RBG curve of order 1, as shown in Fig. 2. To obtain the two coordinates in base 2 from the h-value requires the reflection and rotation recursively on the basic Hilbert curve of order 1, as shown in Fig. 3. As compared to the r-value or the h-value, the z-value has a direct relationship with the coordinates through the bit shuffling, even when the order is greater than 1. Therefore, in this section, we first describe the process of neighbor-finding based on the z-value of the Peano curve.

To simplify our presentation, we let $z2$ be the z-value in base 2, i.e., the binary form, and $z10$ be the z-value in base 10, i.e., the decimal form. (Note that the value in base 10 is represented with base 10 digits from 0 to 9. A base 10 digit represents that digit times a power of 10.) For the query block numbered as $(Z)_{z10}$, it is the z-value of the block in the $2^n \times 2^n$ image space which is ordered by the Peano curve. The value $(Z)_{z2}$ is represented in base 2 with $2n$ $(= n + n)$ bits. So, the sequence numbers for the image space range from 0 to $2^{2n} - 1$. The nearest-neighbor algorithm based on the Peano curve is shown in Fig. 7.

In Step 1, we separate $(Z)_{z2}$, e.g., $(x_2y_2x_1y_1x_0y_0)_{z2}$, into two parts. One part concatenates the odd bits to the X-coordinate, i.e., $X_{c2} = (x_2x_1x_0)_{c2}$, where $c2$ denotes that the value of the coordinate is represented in base 2. The other part concatenates the even bits to the Y-coordinate, i.e., $Y_{c2} = (y_2y_1y_0)_{c2}$. Then, we convert the coordinates $(X, Y)_{c2}$ into $(X, Y)_{c10}$, where $c10$ denotes that the value of the coordinate is represented in base 10. In Step 2, we use $(X, Y)_{c10}$ to find the neighbor in a certain direction. The variables HD and VD mean the horizontal and vertical directions, respectively. For example, if we want to find the west neighbor, we let $HD =$ '*west*' and $VD =$ '*NULL*'. Then, we obtain the resulting coordinates $(NX, NY)_{c10}$ of the neighbor, e.g., $NX_{c10} = X_{c10} - 1$ and $NY_{c10} = Y_{c10}$ for the case of the west neighbor. It should be noticed that either NX_{c10} or NY_{c10} of the neighbor must not be beyond the range of the image space in Step 2. If either one of them is beyond the range, there is no neighbor in a certain direction. In Step 3, we convert $(NX, NY)_{c10}$ into $(NX, NY)_{c2}$, e.g., $(x'_2x'_1x'_0, y'_2y'_1y'_0)_{c2}$. Then, we interleave the bits from NX_{c2} and NY_{c2} to form the corresponding $(NZ)_{z2}$ for the west neighbor, i.e, $(x'_2y'_2x'_1y'_1x'_0y'_0)_{z2}$.

Taking Fig. 8(a) as an example, the width d of each block is 1. For the query block $(45)_{z10} = (x_2y_2x_1y_1x_0y_0)_{z2} = (\underline{1}\,0\,\underline{1}\,1\,\underline{0}\,1)_{z2}$, one of its equal-sized neighbors in the west direction can be obtained by following the steps shown in Fig. 9. Let $X((45)_{z10})$ and $Y((45)_{z10})$ denote the X-coordinate and the Y-coordinate of block $(45)_{z10} = (\underline{1}\,0\,\underline{1}\,1\,\underline{0}\,1)_{z2}$, respectively. First, we separate $(45)_{z10} = (\underline{1}\,0\,\underline{1}\,1\,\underline{0}\,1)_{z2}$ into $X((45)_{z10})$ and $Y((45)_{z10})$, i.e.,

$$\begin{aligned} X((45)_{z10}) &= X((\underline{1}\,0\,\underline{1}\,1\,\underline{0}\,1)_{z2}) \\ &= X((x_2y_2x_1y_1x_0y_0)_{z2}) \\ &= (x_2x_1x_0)_{c2} \\ &= (\underline{1}\,\underline{1}\,\underline{0})_{c2} = (6)_{c10}, \end{aligned}$$

$$\begin{aligned} Y((45)_{z10}) &= Y((\underline{1}\,0\,\underline{1}\,1\,\underline{0}\,1)_{z2}) \\ &= Y((x_2y_2x_1y_1x_0y_0)_{z2}) \\ &= (y_2y_1y_0)_{c2} \\ &= (0\,1\,1)_{c2} = (3)_{c10}. \end{aligned}$$

Next, the NX_{c10} and NY_{c10} values of the west neighbor are shown as follows:

$$NX_{c10} = X((45)_{z10}) - d = (6)_{c10} - (1)_{c10} = (5)_{c10},$$
$$NY_{c10} = Y((45)_{z10}) = (3)_{c10}.$$

```
Function Peano_NN((Z)_z2, HD, VD): string;
/* Find the nearest neighbor next to block (Z)_z2 based on the Peano curve in the certain direction. */
/* l_0 and h_0 are the lower and upper bounds of coordinates along the X-axis. */
/* l_1 and h_1 are the lower and upper bounds of coordinates along the Y-axis. */
/* Z_z2 is the z-value (in base 2) of the query block. */
/* HD and VD are the horizontal and vertical directions. */
/* (X,Y)_c2 is the coordinates (in base 2) of the query block. */
/* (X,Y)_c10 is the coordinates (in base 10) of the query block. */
/* (NX,NY)_c10 is the coordinates (in base 10) of the resulting neighboring block. */
/* (NX,NY)_c2 is the coordinates (in base 2) of the resulting neighboring block. */
/* (NZ)_z2 is the z-value (in base 2) of the resulting neighboring block. */

begin
    /* Step 1: Bit Separation */
    Separate the binary code of Z_z2 into the odd bits and the even bits;
    Concatenate the odd bits to X_c2;
    Concatenate the even bits to Y_c2;
    Convert X_c2 into X_c10;
    Convert Y_c2 into Y_c10;

    /* Step 2: Neighbor Decision */
    if (HD = 'west')
        then X_c10 := X_c10 - 1
    else if (HD = 'east')
        then X_c10 := X_c10 + 1
    if (VD = 'south')
        then Y_c10 := Y_c10 - 1
    else if (VD = 'north')
        then Y_c10 := Y_c10 + 1;
    if (X_c10 < l_0) or (X_c10 > h_0) or (Y_c10 < l_1) or (Y_c10 > h_1) then
        begin
            show an error message; /* There exists no neighbor. */
            Return(' ');
        end
    else begin
            NX_c10 := X_c10;
            NY_c10 := Y_c10;
        end;

    /* Step 3: Bit Shuffling */
    Convert NX_c10 into NX_c2;
    Convert NY_c10 into NY_c2;
    Interleave the bits from NX_c2 and NY_c2 to form (NZ)_z2;
    Return ((NZ)_z2);
end;
```

Fig. 7. Function *Peano_NN*.

There is no change in the Y coordinate of the west neighbor. Then, we convert $(NX, NY)_{c10} = (5, 3)_{c10}$ into $(NX, NY)_{c2} = (\underline{1}\,\underline{0}\,\underline{1}, 0\,1\,1)_{c2} = (x'_2x'_1x'_0, y'_2y'_1y'_0)_{c2}$. Finally, we interleave the bits from NX_{c2} and NY_{c2} to obtain the NZ_{z2} of the west neighbor equal to $(x'_2y'_2x'_1y'_1x'_0y'_0)_{z2} = (\underline{1}\,0\,\underline{0}\;1\,\underline{1}\,1)_{z2}$. Finally, we convert NZ_{z2} into NZ_{z10} to obtain the west neighbor, and the resulting block $(39)_{z10}$ is shown in Fig. 8(a). Similarly, the neighbors in the other directions can also be obtained by function *Peano_NN*, as shown in Fig. 7.

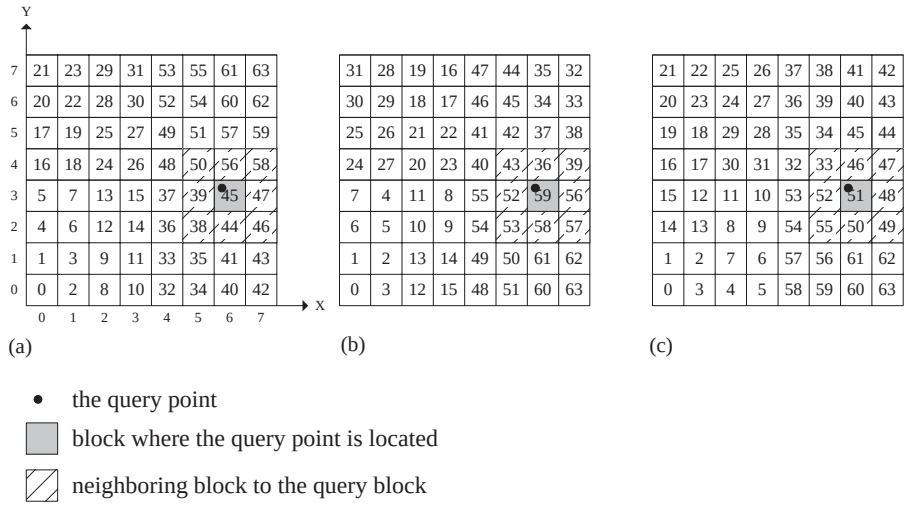

Fig. 8. The sequence numbers linearly ordered by the curves of order 3: (a) the Peano curve; (b) the RBG curve; (c) the Hilbert curve.

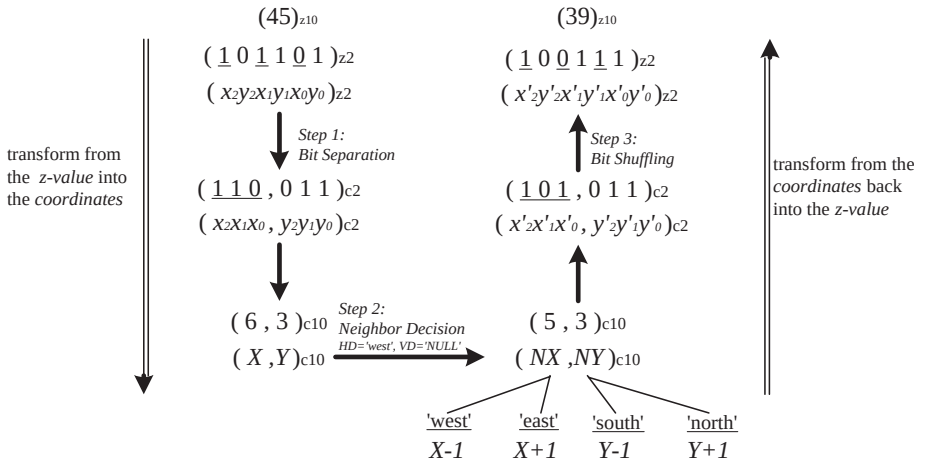

Fig. 9. Illustration of the west neighbor-finding process for the query block $(45)_{z10} = (1\,0\,1\,1\,0\,1)_{z2}$.

3.2. Neighbor-finding based on the other two curves

Based on the Peano curve, we can efficiently and directly find the neighbors next to the query block by function *Peano_NN* (Fig. 7). But it is not so straightforward to answer the nearest-neighbor queries based on the RBG curve or the Hilbert curve, as mentioned above. However, we observe that there is some relationship between the RBG curve and the Peano curve, and between the Hilbert curve and the Peano curve. The r-value in the RBG curve can be represented as the binary gray-code $(R)_{r2}$ (also called gray code), where $r2$ denotes the r-value in base 2. The formulas presented in [1,2,12] can transform $(R)_{r2}$ into its order in the binary form as $(z)_{z2}$. The h-value in the Hilbert curve can be obtained by some transformation, reflection, and rotation on the z-value. So, for the RBG curve (or the Hilbert curve), we can develop some transformation rules between the

Peano curve and itself, and use function *Peano_NN* to help answer the nearest-neighbor query. Therefore, we can first transform the *r*-value (or *h*-value) of the query block into the *z*-value. Then, we use the corresponding *z*-value to find the neighbors based on the Peano curve. We finally transform the *z*-value of the neighbor back into the corresponding *r*-value (or *h*-value) of the neighbors based on the transformation rules. In this section, we first briefly describe the process of the neighbor-finding based on the RBG curve. Then, we show how to find the nearest-neighbor based on the Hilbert curve.

3.2.1. Based on the RBG curve

As mentioned before, for the *r*-value of the 2D-space RBG curve in the $2^n \times 2^n$ image space, we can consider the value $(R)_{r2}$ as the gray code, where *r*2 denotes the *r*-value in base 2. That is the bit string $(R_{2n-1} \cdots R_1 R_0)_{r2}$. The neighbor-finding algorithm based on the RBG curve is shown in Fig. 10. Basically, we first transform $(R)_{r2}$ into the corresponding *z*-value, i.e., $(Z)_{z2}$, by the formulas presented in [1,2,12]. That is, we perform the transformation between the gray code and the binary code. Then, we use $(Z)_{z2}$ to generate one of the neighbors in a certain direction by function *Peano_NN*. Finally, we transform $(NZ)_{z2}$ of the neighbor back into the corresponding $(NR)_{r2}$ by again using the formulas presented in [1,2,12].

Take Fig. 8(b) as an example. It is a $2^3 \times 2^3$ image space which is ordered by the RBG curve of order 3. The query point is located in block $(59)_{r10}$, where *r*10 denotes the *r*-value in base 10. We use block $(59)_{r10}$ as the query block which has the same coordinates $(X, Y)_{c10} = (6, 3)_{c10}$ as the example shown in Fig. 8(a). We represent the r-value $(59)_{r10}$ in base 2 as the gray code, i.e., $(R_5 R_4 R_3 R_2 R_1 R_0)_{r2} = (1\,1\,1\,0\,1\,1)_{r2}$. The west neighbor finding process of block $(59)_{r10}$ is shown

```
Function RBG_NN((R)_r2, HD, VD): string;
/* Find the nearest neighbor next to block (R)_r2 based on the RBG curve in a certain direction. */
/* (R)_r2 is the gray code of length 2n-bits for the query block in the RBG curve. */
/* (Z)_z2 is the z-value in base 2 of length 2n-bits for the query block in the Peano curve. */
/* (NR)_r2 is the gray code of the neighboring block in the RBG curve. */
/* (NZ)_z2 is the z-value in base 2 of the neighboring block in the Peano curve. */

begin
    /* Step 1: Transformation from Gray Code into Binary Code */
    /* In other words, to transform the RBG curve into the Peano curve */
    /* ⊕ is the exclusive-or operator. */
    the left significant bit of (Z)_z2 := the left significant bit of (R)_r2;
    for the following bits of (Z)_z2
        (Z)_z2(2n-1-i) := (R)_r2(2n-1-i) ⊕ (Z)_z2(2n-i);

    /* Step 2: Neighbor Decision */
    (NZ)_z2 := Peano_NN((Z)_z2, HD, VD);

    /* Step 3: Transformation from Binary Code into Gray Code */
    /* In other words, to transform the Peano curve into the RBG curve */
    the left significant bit of (NR)_r2 := the left significant bit of (NZ)_z2;
    for the following bits of (NR)_r2
        (NR)_r2(2n-1-i) := (NZ)_z2(2n-1-i) ⊕ (NZ)_z2(2n-i);
    Return ((NR)_r2);
end;
```

Fig. 10. Function *RBG_NN*.

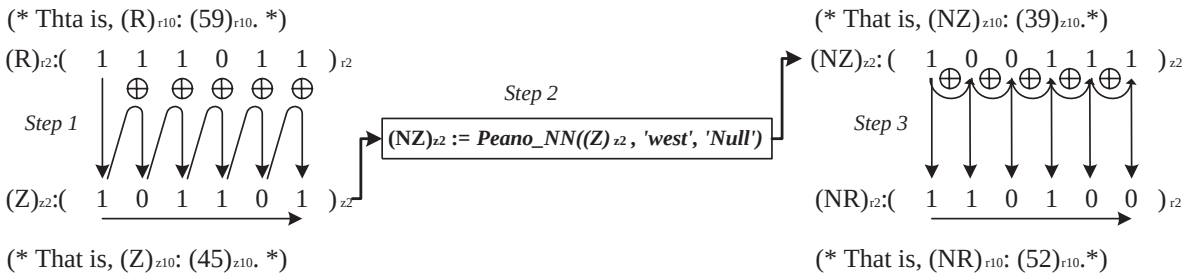

Fig. 11. Illustration of the west neighbor-finding process for the query block $(59)_{r10} = (1\,1\,1\,0\,1\,1)_{r2}$.

in Fig. 11. In Step 1, we transform the gray code $(R)_{r2} = (1\,1\,1\,0\,1\,1)_{r2}$ into the z-value in base 2, i.e., $(Z)_{z2} = (\underline{1}\,0\,\underline{1}\,1\,\underline{0}\,1)_{z2}$. In Step 2, we use the z-value $(\underline{1}\,0\,\underline{1}\,1\,\underline{0}\,1)_{z2} = (45)_{z10}$ to find the west neighbor by function *Peano_NN*. Then, we find the z-value $(\underline{1}\,0\,\underline{0}\,1\,\underline{1}\,1)_{z2} = (39)_{z10}$ of the neighboring block. In Step 3, we transform the z-value $(\underline{1}\,0\,\underline{0}\,1\,\underline{1}\,1)_{z2}$ back into the r-value $(1\,1\,0\,1\,0\,0)_{r2}$. Finally, we convert the r-value $(1\,1\,0\,1\,0\,0)_{r2}$ into the decimal $(52)_{r10}$ which is the neighbor lies the west of the query block $(59)_{r10}$ in Fig. 8(b). Note that block $(52)_{r10}$ also locates at $(NX, NY)_{c10} = (5, 3)_{c10}$, so we have the same answer as the previous example shown in Fig. 8(a).

3.2.2. Based on the Hilbert curve

For the h-value of the 2D-space Hilbert curve in the $2^n \times 2^n$ image space, we consider the value H in base 2 as a Hilbert code, i.e., $(h_{2n-1} \cdots h_1 h_0)_{h2}$, where $h2$ denotes the h-value in base 2. The pseudo codes for the neighbor-finding process is shown in Fig. 12, and the related source codes about the transformation rules between the Peano curve and the Hilbert curve are shown in Appendix A. Basically, similar to the previous function, from Step 1 to Step 4, we first transform the h-value into the z-value. Then, in Step 5, we use the z-value to generate the requested neighbor based on the Peano curve. Finally, from Step 6 to Step 9, we transform the z-value of the neighbor into the corresponding h-value. However, in this case, it is not so straightforward to do the transformation between the Hilbert curve and the Peano curve since the coordinates in base 2 from the h-value require the reflection and rotation recursively on the basic Hilbert curve of order 1.

Take Fig. 8(c) as an example. It is a $2^3 \times 2^3$ image space which is ordered by the Hilbert curve of order 3. The query point is located in block $(51)_{h10}$, where $h10$ denotes the h-value in base 10. We use block $(51)_{h10}$ as the query block, which has the same coordinates $(X, Y)_{c10} = (6, 3)_{c10}$ as the example shown in Fig. 8(a). The west neighbor-finding process of block $(51)_{h10}$ is shown in Fig. 13. We first represent the h-value, i.e., $(51)_{h10}$, in base 2 as the Hilbert code $(h_5 h_4 h_3 h_2 h_1 h_0)_{h2} = (1\,1\,0\,0\,1\,1)_{h2}$. We describe the neighbor-finding process of block $(51)_{h10}$ in the following steps.

Step 1: We divide the Hilbert code $(\underline{11}\,0011)_{h2}$ into pairs of bits from left to right and store the corresponding decimal value for each pair of bits in array d. Those are $d_0 = (\underline{11})_2 = 3$, $d_1 = (00)_2 = 0$, and $d_2 = (11)_2 = 3$. The decimal value denotes the sequence number (0, 1, 2, or 3) in one quadrant ordered by the Hilbert curve, as shown in Fig. 3(a). We also let another array e be same as array d, i.e., $e_0 = d_0 = 3$, $e_1 = d_1 = 0$, $e_2 = d_2 = 3$. (Note that array e is used for storing the temporary result in the following steps.) Array d is represented as $(d_0 d_1 d_2)_{hp} = (3\ 0\ 3)_{hp}$, where hp denotes the code composed of the sequence numbers in base 10 for *pairs* of bits in $(H)_{h2}$, and so is array e. Note that since all reflections and rotations in the Hilbert curve of order p, $p > 1$ are relative to the previously obtained curve of order $p - 1$, array d is the input used to store the sense of order p, and array e is the output used to store the sense of order $p - 1$. In other words, for the following step, there will be no change to array d to keep the information about reflection and rotation in the curve of order p. And array e will be repeatedly updated (if necessary) to show the sense in the curve of order $p - 1$.

```
Function Hilbert_NN((H)_h2, HD, VD): string;
/* Find the nearest neighbor next to block (H)_h2 based on the Hilbert curve in the certain direction. */
/* (H)_h2 is the Hilbert code of the query block. */
/* (Z)_z2 is the z-value in base 2 of the query block. */
/* (NH)_h2 is the Hilbert code of the neighboring block. */
/* (NZ)_z2 is the z-value in base 2 of the neighboring block. */
/* c, d, e are arrays to store the decimal values. */

begin
    /* Step 1: Bit Division */
    for each pair i of bits in (H)_h2 from left to right
    begin
        Convert them into the decimal value d_i;
        e_i:=d_i;
    end;

    /* Step 2: Reducing the Hilbert curve from order i to order 1, i > 1 */
     Call Rotate_Reflect(d, e);

    /* Step 3: Transformation from the Hilbert Curve into the Peano Curve */
    Call PHTransformation(e);

    /* Step 4: Bit Concatenation */
    for each e_i in e
        Concatenate the binary bits of e_i to (Z)_z2;

    /* Step 5: Neighbor Decision based on the Peano curve */
    (NZ)_z2 := Peano_NN((Z)_z2, HD, VD);

    /* Step 6: Bit Division */
    for each pair i of bits in (NZ)_z2 from left to right
        Convert them into the decimal value c_i;

    /* Step 7: Transformation from the Peano Curve into the Hilbert Curve */
    Call PHTransformation(c);

    /* Step 8: Generating the Hilbert curve of order i from order 1, i > 1 */
     Call Rotate_Reflect(c, c);

    /* Step 9: Bit Concatenation */
    for each c_i in c
        Concatenate the binary bits of c_i to (NH)_h2;
    Return ((NH)_h2);
end;
```

Fig. 12. Function *Hilbert_NN*.

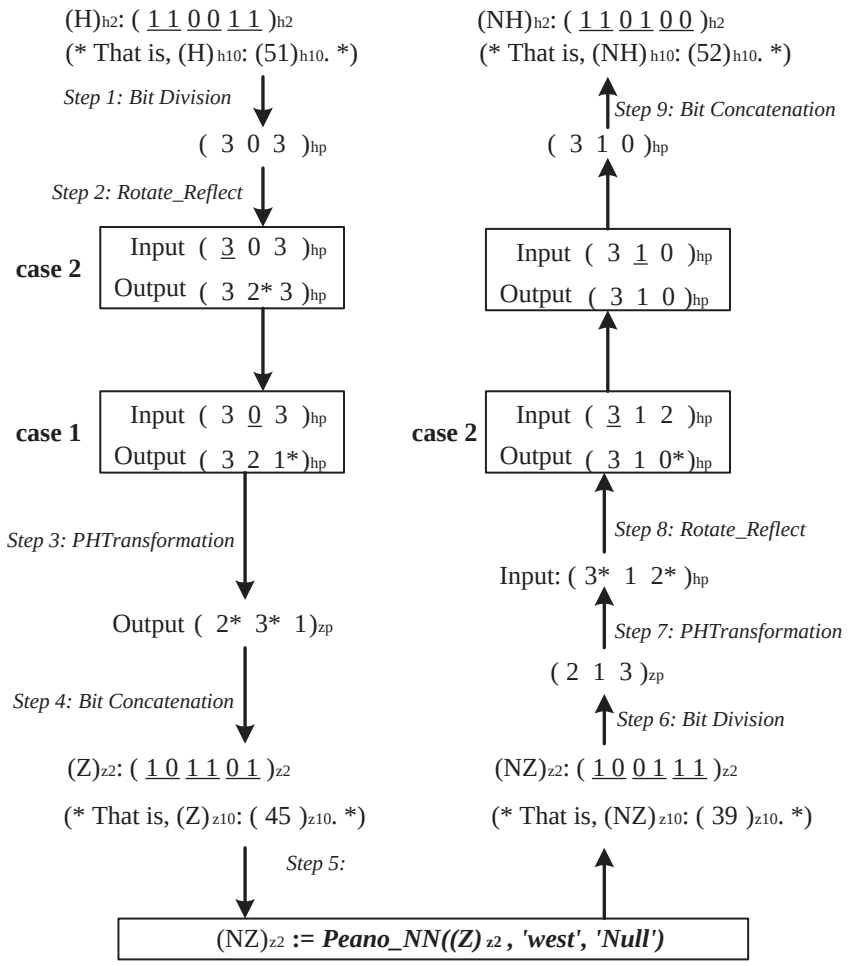

Fig. 13. Illustration of the west neighbor-finding process for the query block $(51)_{h10} = (1\,1\,0\,0\,1\,1)_{h2}$.

Step 2: We call procedure *Rotate_Reflect*, as shown in Fig. 14, to perform the task of reducing the Hilbert curve from order i ($i > 1$) to order 1. For each element in array d, we first consider the value of d_i to decide which case to follow in procedure *Rotate_Reflect*, as shown in Fig. 14. Arrays d and e are the input and output, respectively. In Case 1, if the value of d_i is 0, then we should change the value of the following element e_j from 1 to 3, or from 3 to 1, $j > i$. In Case 2, if the value of d_i is 3, then we should change the value of the following element e_j if the value is from 0 to 2, or from 2 to 0, $j > i$. In Fig. 13, the input is array $d = (3\ 0\ 3)_{hp}$ and the output is array $e = (3\ 0\ 3)_{hp}$ initially, where the mark _ represents the processing point. First, we process the first value of the input, i.e., d_0 (= 3). Since it is 3, which is Case 2, we change the second value of the output (e_1) from 0 to 2. Since there is no other 0 or 2 in the remaining e_j, $j > i$, we do not do any other update to the output for the input d_0. (Note that in Fig. 13, the mark * represents the changed value.) Next, we process the second value of the input, i.e., d_1 (= 0). Since it is 0, which is Case 1, we change the third value of the output (e_2) from 3 to 1. Up to this point, we have finished the process of all elements of the input, and the resulting output is $(3\,2\,1)_{hp}$.

```
Procedure Rotate_Reflect(input, output);
/* Transformation between the Hilbert curve of order p and order q */
/* input is an array to store the sequence numbers for the curve of order p. */
/* output is an array to store the sequence numbers for the curve of order q. */
begin
    for i = 1 to |input| - 1 do
    begin
        /* Case 1 */
        if (input[i] = 0) then
            for the following element j := i + 1 to |output| do
            begin
                if (output[j] = 1) then output[j] := 3
                else if (output[j] = 3) then output[j] := 1;
            end;
        /* Case 2 */
        else if (input[i] = 3) then
            for the following element j := i + 1 to |output| do
            begin
                if (output[j] = 0) then output[j] := 2
                else if (output[j] = 2) then output[j] := 0;
            end;
    end;
end;
```

Fig. 14. Procedure *Rotate_Reflect*.

The reason for the change operation for Case 1 and Case 2 is as follows. In order to preserve the proximity, the order p ($p > 1$) of the Hilbert curve is derived from the rotation and reflection on the basic curve of order 1 continuously in each quadrant. The sense, i.e., the direction of traversal [8], in each quadrant is not always the same. However, different from the Hilbert curve, the Peano curve recursively shows the same sense as the basic curve of order 1 to form the curve of order p, $p > 1$. So, during the process of the transformation between the h-value and the z-value, we need to perform the rotation and reflection in reverse order such that each divided quadrant will have the same sense as the basic Hilbert curve of order 1.

Take Fig. 15 as an example. Figs. 15(a) and (b) show the Hilbert curves of order 1 and 2, respectively. The ( ) denotes the sequence number of the quadrant in the Hilbert curve of order 1, in which the quadrant has four blocks. Each block has the sequence number ordered by the Hilbert

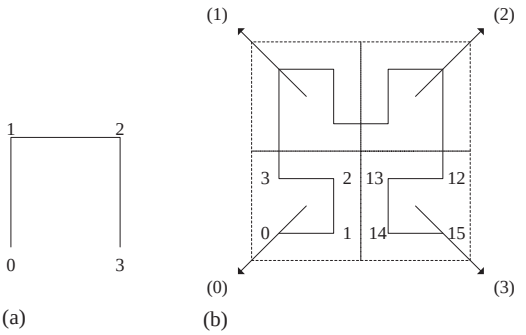

Fig. 15. Hilbert curves of order: (a) 1; (b) 2. (N_1): N_1 is the sequence number of the quadrant in the curve of order 1.

curve of order 2. We can observe that the sense of block (0) of order 2 in Fig. 15(b) is different from the sense of order 1 in Fig. 15(a), as is block (3). That is, in block (0), the position of sequence number 1 is different from its position in Fig. 15(a), as is the position of sequence number 3. It is just the sense of block (0) of order 2

(Fig. 15(b)) as derived from the sense of one of order 1 (Fig. 15(a)) by right-reflection (Fig. 16(a)) and 90°-clockwise-rotation (Fig. 16(b)). After processing the reflection and rotation, the result is shown in Fig. 16(c). This is Case 1 in procedure *Rotate_Reflect*. Similarly, for block (3) in Fig. 15(b), the position of sequence number 12 ($= 4 \times 3 + \underline{0}$) is different from its position (sequence number $\underline{0}$) in Fig. 15(a), so is the position of sequence number 14 ($= 4 \times 3 + \underline{2}$) vs. the position of sequence number 2. This is because the sense of block (3) (Fig. 15(b)) is derived from the sense of order 1 (Fig. 15(a)) by left-reflection (Fig. 17(a)) and 90°-anticlockwise-rotation (Fig. 17(b)). After processing the reflection and rotation, the result is shown in Fig. 17(c). This is Case 2 in procedure *Rotate_Reflect*. However, the sense of block (1) in Fig. 15(b) is the same as that of order 1 (Fig. 15(a)) without processing rotation and reflection, so is block (2). Therefore, to reduce the Hilbert curve of order p ($p > 1$) to the basic curve of order 1, we have to perform procedure *Rotate_Reflect*. In this way, after all senses in the Hilbert curve of order p ($p > 1$) have been reduced to the ones in the basic Hilbert curve of order 1, we can do the transformation from the Hilbert curve into the Peano curve.

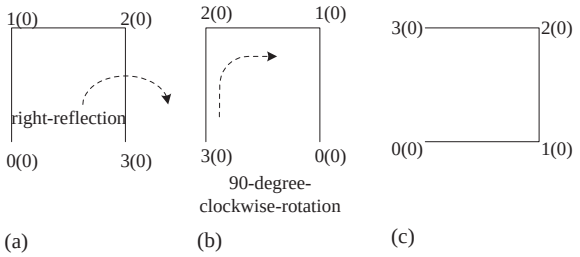

Fig. 16. The process of reflection and rotation on block (0) in the Hilbert curve of order 2 in Fig. 15: (a) before the right reflection; (b) after the right reflection; (c) after the 90°-clockwise-rotation. $N_2(N_1)$: N_1 is the sequence number of the block in the curve of order 1, and N_2 is the sequence number of the block in the curve of order 2.

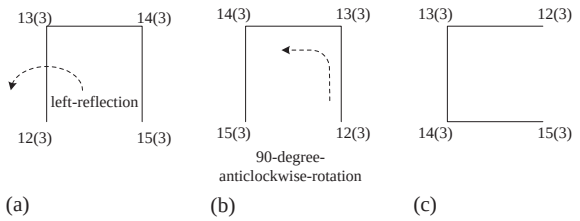

Fig. 17. The process of reflection and rotation on block (3) in the Hilbert curve of order 2 in Fig. 15: (a) before the left reflection; (b) after the left reflection; (c) after the 90°-anticlockwise-rotation. $N_2(N_1)$: N_1 is the sequence number of the block in the curve of order 1, and N_2 is the sequence number of the block in the curve of order 2.

Step 3: Up to this point, we have the code of $(3\ 2\ 1)_{hp}$, which is the code of the Hilbert curve of order 1. Note that in procedure *PHTransformation*, as shown in Fig. 18, we observe that the position of sequence number 2 in the basic Hilbert curve of order 1 (Fig. 19(a)) is different from its position in the basic Peano curve of order 1 (Fig. 19(b)), as is sequence number 3. Therefore, we change the position of sequence numbers 2 and 3 in the basic Hilbert curve of order 1 to those in the basic Peano curve to obtain the z-value. So, we transform $(3\ 2\ 1)_{hp}$ into the corresponding code of the z-value, i.e., $(2{*}3{*}1)_{zp}$, according to Table 1 by procedure *PHTransformation*, where *zp* denotes the code composed of sequence numbers in base 10 for *pairs* of bits in $(Z)_{z2}$.

Step 4: We transform each number of $(\underline{2}\ \underline{3}\ \underline{1})_{zp}$ into the corresponding pair of bits. Then, we concatenate them to form $(\underline{10}\ \underline{11}\ \underline{01})_{z2}$.

Step 5: We use the z-value $(1\,0\,1\,1\,0\,1)_{z2} = (45)_{z10}$ to find the nearest neighbor to the west based on the Peano curve by function *Peano_NN*. We find the west neighbor which is $(1\,0\,0\,1\,1\,1)_{z2} = (39)_{z10}$ in Fig. 8(a).

Step 6: We divide the code $(\underline{10}\,0111)_{z2}$ into pairs of bits from left to right and store the corresponding decimal value for each pair of bits in array c. These are $c_0 = (\underline{10})_2 = 2$, $c_1 = (01)_2 = 1$, and $c_2(11) = 3$. We represent array c as $(2\ 1\ 3)_{zp}$, which represents the code of sequence numbers in the Peano curve.

Step 7: In this step, based on the similar reason as Step 3, we change the position of sequence numbers 2 and 3 in the basic Peano curve of order 1 to those in the basic Hilbert curve of order 1. We transform the code of $(c_0c_1c_2)_{zp} = (2\ 1\ 3)_{zp}$ into the corresponding *hp* code in the basic Hilbert curve of order 1 according to Table 1 by using procedure *PHTransformation* in Fig. 18. At this point, we

```
Procedure PHTransformation(m);
/* Transformation between the Peano curve and the Hilbert curve */
/* m is an array to store the decimal values. */
begin
    for each i := 1 to |m| do
    begin
        if (m[i] = 2) then m[i] := 3
        else if (m[i] = 3) then m[i] := 2;
    end;
end;
```

Fig. 18. Procedure *PHTransformation*.

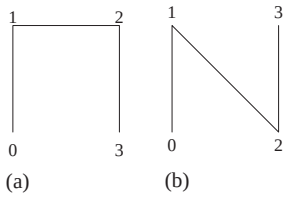

Fig. 19. Curves of order 1: (a) the Hilbert curve; (b) the Peano curve.

Table 1
Relationship between the sequence numbers in the Peano curve and those in the Hilbert curve

Peano curve	Hilbert curve
0	0
1	1
2	3
3	2

have the resulting array c as $(3^*\ 1\ 2^*)_{hp}$, which represents the code of sequence numbers in the basic Hilbert curve of order 1.

Step 8: In this step, with a reason similar to Step 2, we should generate the Hilbert curve of order p ($p > 1$) from the basic Hilbert curve of order 1. For each element in array c, we consider the value of c_i to decide which case to follow in procedure *Rotate_Reflect* (Fig. 14). Note that since all reflections and rotations in the Hilbert curve of order p, $p > 1$ are relative to the previously obtained curve of order $p-1$, the array c is not only the input, which represents the sense of order $p-1$, but also the output, which represents the sense of order p. First, we process the first value of the input, i.e., c_0 (= 3). Since it is 3, which is Case 2, we change only the third value of the output (c_2) from 2 to 0. Since there is no other 0 or 2 in the second value c_1 (= 1) and the remaining c_j, $j > i$, we do not do any other update to the output for the input c_0. Next, we process the second value of the input, i.e., c_1 (= 1). Since it is 1, we do not update the output. Up to this point, we have finished the process of all elements of the input, and the resulting output is $(3\ 1\ 0)_{hp}$, which represents the code of sequence numbers in the Hilbert curve of order p ($p > 1$).

Step 9: We transform each number of $(c_0c_1c_2)_{hp} = (3\ 1\ 0)_{hp}$ into the corresponding pair of bits. Then, we concatenate them to form $(\underline{11}\,\underline{01}\,\underline{00})_{h2}$. Finally, we obtain block $(52)_{h10} = (1\ 1\ 0\ 1\ 0\ 0)_{h2}$ as the west neighbor next to block $(51)_{h10}$ as shown in Fig. 8(c). Note that block $(52)_{r10}$ also locates at $(NX, NY)_{c10} = (5, 3)_{c10}$, which is the same answer as the previous example in Fig. 8(a).

4. Performance

In this section, we compare the performance of nearest-neighbor-finding strategies based on three space filling curves. Our experiments were performed on a Pentium III 733 MHz, 128 MB RAM, 7200 rpm IBM Hard Disk (model name: IC35L040AVVN07-0), and running Windows 2000.

4.1. The performance model

In the conventional approach as mentioned in Section 3, it requires the time to traverse the disk blocks by following the sequence numbers in the preceding and succeeding order of the query block in the 1D-space to find the neighboring blocks. The number of these traversed neighboring blocks are unpredictable and may be greater than eight blocks, since some unrelated disk blocks in the 1D-space may be accessed. Take the query block 13 in Fig. 6(c) based on the Hilbert curve as an example, it will access 11 blocks in the 1D-space to find one neighboring block 2, which is one block away from the query block 13 in the 2D-space. As compared to the conventional approach, our approach does not need to access unrelated disk blocks in the 1D-space. In our approach, first, we compute the sequence numbers of the neighboring blocks next to the query block in eight directions. It takes the CPU-time to compute these sequence numbers based on the space-filling curve. Next, once the sequence numbers of the eight neighboring blocks are obtained, we can fetch the corresponding blocks in the disk directly according to the sequence numbers. Then, we can find the nearest point to the query point only in these eight neighboring blocks in the 2D-space, which speeds up the search operation.

The I/O time, which means the time to access a block, consists of two parts: *positioning time* (PT) and *transfer time* (TT) [13]. The positioning time is the time to move to the right position in the disk, which includes the seek time and rotational delays. The transfer time is simply the time to transfer the requested block from the disk into main memory. Since it is more efficient to fetch a set of consecutive disk blocks than a randomly scattered set, in order to reduce additional positioning time, we can make use of the *clustering* property of the space filling curve to replace several exact match queries with one continuous range query to reduce the positioning time. The clustering property means that the locality between objects in the 2D-space can be preserved in the 1D-space. In our approach, since the adjacent blocks on the disk can be accessed with a single range query, we can combine eight exact match queries into some range queries to reduce the positioning time.

Take Fig. 8 as an example. In the case of the Peano curve as shown in Fig. 8(a), let's presume the query block be block 45. The sequence numbers of the neighboring blocks in eight directions are blocks 56, 58, 47, 46, 44, 38, 39, and 50. Then, we can combine these eight exact match queries into six range queries, including 38 to 39, 44, 46 to 47, 50, 56, and 58. The positioning time for six range queries is $6PT$. The transfer time to transfer these eight blocks from the disk is $8TT$. So, the total I/O time to access these eight disk blocks based on the Peano curve is $(6PT + 8TT)$. (Note that in this case, the number of blocks which the conventional approach has to access, is up to 21 (from 38 to 58) blocks in the worst case. Moreover, in the conventional approach, the higher the order of the space-filling curve is, the large the number of blocks accessed [3,5,14,15].)

In the case of the RBG curve as shown in Fig. 8(b), let's presume query block be block 59. The sequence numbers of the neighboring blocks in eight directions are blocks 36, 39, 56, 57, 58, 53, 52, and 43. Then, we can combine these eight exact match queries into five range queries, including 36, 39, 43, 52 to 53, and 56 to 58. So, the total I/O time to access these eight disk blocks based on the RBG curve is $(5PT + 8TT)$.

In the case of the Hilbert curve as shown in Fig. 8(c), let's assume the query block be block 51. The sequence numbers of the neighboring blocks in eight directions are blocks 46, 47, 48, 49, 50, 52, 55, and 33. Then, we can combine these eight exact match queries into four range queries, including 33, 46 to 50, 52, and 55. So, the total I/O time to access these eight disk blocks based on the Hilbert curve is $(4PT + 8TT)$. Therefore, as compared to the conventional approach, our approach can reduce the I/O time in accessing the unrelated blocks by directly accessing the eight related blocks. Moreover, in this example, among three space-filling curves, our strategy based on the Hilbert curve needs the least I/O time because the Hilbert curve provides the best clustering property [3–5,8,9].

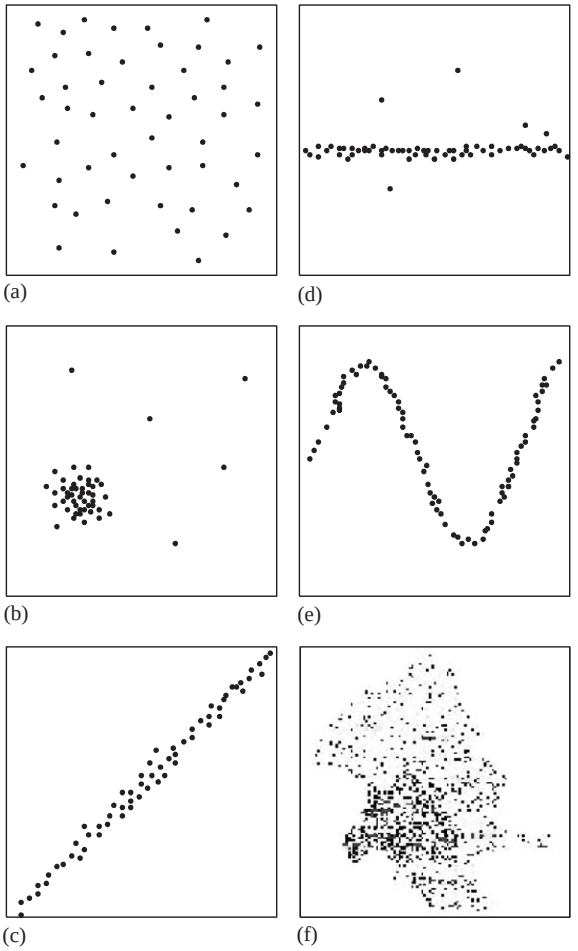

Fig. 20. Six cases of data distribution: (a) uniform; (b) centralized; (c) diagonal; (d) X-parallel; (e) sine; (f) the real map.

In this comparison, we considered the CPU-time and the I/O time as our performance measures. We focused on the point data. The experimental analysis was carried out on six distinct spatial datasets. Fig. 20 shows these six cases of data distribution which are described as follows [16]:

(a) *Uniform distribution*: The data objects are uniformly distributed in the overall data space.
(b) *Centralized distribution*: Most of the data objects are centralized in a small region.
(c) *Diagonal distribution*: The data objects follow a uniform distribution along the main diagonal.
(d) *X-parallel distribution*: Most of the data objects are located on a line which is parallel to the X-axis.
(e) *Sine distribution*: The data objects follow a sine curve.
(f) *Real map*: This is the point map of the population estimate in Taipei, Taiwan. There are 26 376 data points on this map. One point represents 100 people. This statistical information is retrieved from http://www.taiwandm.com.tw/c1-1.htm.

Each dataset contains 10 000 data points in the 2D-space which is 1000×1000. First, the data points were assigned to the corresponding disk block based on the space-filling curve. The points in the same disk block are linked. The capacity of each disk block, denoted as *block_capacity*, was assigned to be 10. Because of the limitation of *block_capacity*, a space-filling curve of different order p ($p > 1$) was required for the different datasets. Next, for each dataset in Figs. 20(a)–(f), 5000 query points were randomly generated to make the nearest-neighbor queries. Finally, we took the average of the CPU-time and the I/O time required to answer the nearest neighbor queries for each of the dataset as shown in Figs. 20(a)–(f).

4.2. Simulation results

Table 2 shows that different orders for the space-filling curve, which were required for the different datasets to satisfy the limitation of *block_capacity* ($= 10$), since they have different data distribution. For example, for the uniform data distribution in Fig. 20(a), the space-filling curve of order 5 was required to order 10 000 data points. This is because there are 1024 ($= 2^5 \times 2^5$) disk blocks which are used to store these points and each disk block contains 10 points on the average. For the centralization, diagonal, X-parallel, and sine data distributions in Figs. 20(b), (c), (d), and (e), respectively, most of their data points are distributed in the certain

Table 2
Different orders for the space-filling curve to organize different datasets

Data distribution	Uniform	Centralization	Diagonal	X-parallel	Sine	Real map
Order	5	6	7	5	7	6

Table 3
Comparison of the CPU-time (μs) on different datasets

Data distribution	Uniform	Centralized	Diagonal	X-parallel	Sine	Real map
Peano	42.0	48.0	54.0	42.2	54.0	50.2
RBG	70.0	84.2	94.2	72.0	96.0	84.2
Hilbert	98.2	114.0	128.2	100.2	130.2	112.2

Table 4
Comparison of the average positioning time of I/O time for our strategies

Data distribution	Uniform	Centralization	Diagonal	X-parallel	Sine	Real map
Peano	$5PT$	$5PT$	$6PT$	$5PT$	$6PT$	$5PT$
RBG	$5PT$	$5PT$	$5PT$	$5PT$	$5PT$	$5PT$
Hilbert	$4PT$	$4PT$	$4PT$	$4PT$	$4PT$	$4PT$

region or near the certain line. Because of the limitation of *block_capacity*, the space-filling curve of order (5 or 6 or 7) was required for each of these datasets. For the real map shown in Fig. 20, the space filling curve of order 6 was required to organize these data points.

First, we discuss the results of the comparison of the CPU-time of our strategies, as shown in Table 3. For all datasets, the CPU-time of the strategy based on the Peano curve is always less than that of the other two strategies. The main reason for this is as follows. For the strategy based on the Peano curve, it takes the CPU-time in function *Peano_NN* as shown in Fig. 7. In contrast, for the strategy based on the RBG curve, in addition to the CPU-time taken in function *Peano_NN*, it also takes the CPU-time in the transformation between the Peano curve and itself (Fig. 10). For the strategy based on the Hilbert curve, in addition to the CPU-time taken in function *Peano_NN*, it also takes the CPU-time in the transformation between the Peano curve and itself (Fig. 12). Thus, the strategy based on the Peano curve will take less CPU-time than the strategies based on the other two curves. Therefore, for the performance measure of the CPU-time, the strategy based on the Peano curve provides better result than the strategies based on the other two curves.

Next, we discuss the effect of different datasets to the needed CPU-time. For the strategy based on the Peano curve, the CPU-time is the same among the first five different datasets. This is because the strategy based on the Peano curve uses the bit shuffling directly on the coordinates and the z-value to find the neighboring blocks, even though the order is greater than 1. For the real map shown in Fig. 20(f), since the data points are uniform distributed, it takes the CPU-time for the uniform distribution, which is organized by the space filling curve of order 6.

However, for the strategy based on the RBG curve (or the Hilbert curve), the higher the order of the space-filling curve is, the longer the needed CPU-time is. It takes more CPU-time and becomes more complex than the strategy based

on the Peano curve in computing the sequence numbers of the neighboring blocks next to the query block, especially when the order is increased for the different datasets. For example, since the sine data distribution requires the curve of order 7, which is larger than the curve of order 6 required by the centralized distribution, the sine data distribution takes more CPU-time than the centralized distribution does.

Next, we discuss the performance of the I/O time among our strategies. The average positioning time for each of our three strategies on the different datasets is shown in Table 4. The transfer time for our strategies is the same, i.e., $8TT$. So, the average I/O time for the strategies based on the Peano curve, the RBG curve, and the Hilbert curve are $(5.3PT + 8TT)$, $(5PT + 8TT)$, and $(4PT + 8TT)$, respectively. In our simulation environment, one unit of the positioning time is average 8.8 ms, which is much longer than the CPU-time. (Note that this information is retrieved from http://www.digit-life.com/articles/digests/hddreview-0602-ibm.html.) For the performance of the I/O time, our strategy based on the Hilbert curve requires less time than the strategies based on the other two curves, because the Hilbert curve can provide the better clustering property than the other two curves [3,5,8]. But for the performance of the CPU time, our strategy based on the Hilbert curve requires more time than the strategies based on the other two curves because of the complicated transformation rules between the Peano curve and the Hilbert curve. Since the I/O time is generally several orders of magnitude longer than the CPU time, our strategy based on the Hilbert curve can provide better performance (the CPU-time + the I/O time) than the strategies based on the Peano curve and the RBG curve for processing nearest-neighbor queries.

Consequently, our strategy based on the Peano curve requires less CPU-time than the strategy based on the RBG curve (or the Hilbert curve) by using the bit shuffling property of the Peano curve. When the data are linearly ordered by the RBG curve (or the Hilbert curve), it takes most of the CPU-time in using the complicated transformation rules between the Peano curve and the RBG curve (or the Hilbert curve) to compute the sequence numbers of the eight neighboring blocks next to the query block in the 2D-space. Moreover, the strategy based on the Hilbert curve requires the least total time (the CPU-time + the I/O time) in the nearest-neighbor-finding, since the Hilbert curve provides the best clustering property among three space-filling curves.

5. Conclusion

In this paper, we have shown that the property of the Peano curve, bit shuffling directly on the coordinates in base 2, is useful in directly computing the sequence number of the eight neighboring blocks next to the query block in the 2D-space. Then, we have presented the transformation rules between the Peano curve and the RBG curve (or between the Peano curve and the Hilbert curve), which can be used when the data is linearly ordered by the RBG curve (or the Hilbert curve). Finally, we have compared the CPU-time and the I/O time among different nearest-neighbor strategies. From our simulation, among our three strategies, we have shown that the strategy based on the Peano curve requires the least CPU-time in computing the sequence number of the 8 neighboring blocks next to the query block, and the strategy based on the Hilbert curve requires the least I/O time to access eight neighboring blocks next to the query block because of its good clustering property. Since the I/O time is much longer than the CPU-time, the strategy based on the Hilbert curve requires the least total time (the CPU-time + the I/O time) to process the nearest-neighbor query among our three strategies.

Appendix A

The source codes of the transformation rules (Procedure *HilbertToPeano* and Procedure *PeanoToHilbert*) between the Peano curve and the Hilbert curve are shown in Figs. 21 and 22, respectively.

```
Procedure HilbertToPeano(IH);
/* Convert h-value to z-value */
/* IH is a bits array that represents the binary form of the h-value. That is, h_i, 0 ≤ i ≤ 2n − 1.*/
/* IB is a bits array that represents the binary form of the z-value. That is, b_i, 0 ≤ i ≤ 2n − 1.*/
/* ID is an integer array that represents the decimal form of 2 bits from left to right in IH. */
/* That is d_j, for 0 ≤ j ≤ n − 1, for each 2-bit h_i h_{i+1} as s_j, obtained according to the related table. */
/* IE is a temporary array that is the same array as ID. */
    begin
      /* Step 1: Bit Division */
      j = 0;
      while j <= (n − 1) do
        begin
          ID[j] := 2 * IH[j * 2] + IH[j * 2 + 1];
          IE[j] := ID[j];
          j := j + 1;
        end;
      /* Step 2: Rotate_Reflect */
      j := 0;
      while j < (n − 1) do
        begin
          if (ID[j] = 0) then
          /* Remove the right-reflection and 90°-clockwise rotation */
           /* on the bottom-left part of the higher order curve */
            begin
              k := j + 1;
              while k <= (n − 1) do
                begin
                  if (IE[k] = 1) then IE[k] := 3
                  else if (IE[k] = 3) then IE[k] := 1;
                  k := k + 1;
                end;
            end
          else
          if (ID[j] = 3) then
          /* Remove the left-reflection and 90°-anti-clockwise rotation */
           /* on the bottom-right part of the higher order curve */
            begin
              k := j + 1;
              while k <= (n − 1) do
                begin
                  if (IE[k] = 0) then IE[k] := 2
                  else if (IE[k] = 2) then IE[k] := 0;
                  k := k + 1;
                end;
            end;
          j := j + 1;
        end;
      /* Step 3: PHTransformation */
      /* Change the h-value in the Hilbert curve to the z-value in the Peano curve.*/
      j := 0;
      while j <= (n − 1) do
        begin
          if (IE[j] = 3) then IE[j] := 2
          else if (IE[j] = 2) then IE[j] := 3;
          IB[2 * j] := IE[j] div 2;
          IB[2 * j + 1] := IE[j] mod 2;
          j := j + 1;
        end;
      /* Step 4: Bit Concatenation */
       (The result is IB.)
    end;
```

Fig. 21. Procedure *HilbertToPeano*

```
Procedure PeanoToHilbert(IB);
/* Convert z-value to h-value */
/* IB is a bits array that represents the binary form of the z-value. That is, b_i, 0 ≤ i ≤ 2n − 1. */
/* IH is a bits array that represents the binary form of the h-value. That is, h_i, 0 ≤ i ≤ 2n − 1.*/
/* IC is an integer array that represents the decimal form of 2 bits from left to right in the IB. */
/* That is, c_j, for 0 ≤ j ≤ n − 1, for each 2-bit b_i b_{i+1} as t_j, obtained according to the related table. */
    begin
      /* Step 6: Bit Division */
       j = 0;
       while j <= (n − 1) do
          begin
             ID'[j] := 2 * IB[j * 2] + IB[j * 2 + 1];
             j := j + 1;
          end;
       /* Change the z-value in the Peano curve to the h-value in the Hilbert curve. */
       /* Step 7: PHTransformation */
       j := 0;
       while j <= (n − 1)do
          begin
             if (IC[j] = 3) then IC[j] := 2
             else if (IC[j] = 2) then IC[j] := 3;
             j := j + 1;
          end;
       j := 0;
       while j < (n − 1) do
          begin
             if (IC[j] = 0) then
             /* Make the right-reflection and 90°-clockwise rotation of the lower order curve */
             /* to form the bottom-left part of the higher order curve */
                begin
                   k := j + 1;
                   while k <= (n − 1) do
                      begin
                         if (IC[k] = 1) then IC[k] := 3
                         else if (IC[k] = 3) then IC[k] := 1;
                         k := k + 1;
                      end;
                end
             else
             if (IC[j] = 3) then
             /* Make the left-reflection and 90°-anti-clockwise rotation of the lower order curve */
             /* to form the bottom-right part of the higher order curve */
                begin
                   k := j + 1;
                   while k <= (n − 1) do
                      begin
                         if (IC[k] = 0) then IC[k] := 2
                         else if (IC[k] = 2) then IC[k] := 0;
                         k := k + 1;
                      end;
                end;
             j := j + 1;
          end;
       /* Step 8: Rotate_Reflect */
       j := 0;
       while j <= (n − 1) do
          begin
             IH[2 * j] := IC[j] div 2;
             IH[2 * j + 1] := IC[j] mod 2;
             j := j + 1;
          end;
       /* Step 9: Bit Concatenation */
       (The result is IH.)
    end;
```

Fig. 22. Procedure *PeanoToHilbert*.

References

[1] C. Faloutsos, Multiattribute hashing using gray codes, Proceedings of ACM SIGMOD International Conference on Management of Data, August 1986, pp. 227–238.
[2] C. Faloutsos, Gray codes for partial match and range queries, IEEE Trans. Software Eng. 14 (10) (1988) 1381–1393.
[3] C. Faloutsos, S. Roseman, Fractals for secondary key retrieval, Proceedings of ACM SIGACT-SIGMOD-SIGART Symposium on PODS, 1989, pp. 247–252.
[4] J.K. Lawder, P.J.H. King, Querying multi-dimensional data indexed using the Hilbert space-filling curve, ACM SIGMOD Record 30 (1) (2001) 19–24.
[5] B. Moon, H.V. Jagadish, C. Faloutsos, J.H. Saltz, Analysis of the clustering properties of the Hilbert space-filling curve, IEEE Trans. Knowledge Data Eng. 13 (1) (2001) 124–141.
[6] J.A. Orenstein, T.H. Merrett, A class of data structures for associative searching, Proceedings of Symposium on PODS, 1984, pp. 181–190.
[7] J.A. Orenstein, Spatial query processing in an object-oriented database system, Proceedings of ACM SIGMOD International Conference on Management of Data, 1986, pp. 326–336.
[8] H.V. Jagadish, Linear clustering of objects with multiple attributes, Proceedings of ACM SIGMOD International Conference on Management of Data, 1990, pp. 332–342.
[9] H.V. Jagadish, Analysis of the Hilbert curve for representing two-dimensional space, Inform. Process. Lett. 62 (1) (1997) 17–22.
[10] A. Corral, Y. Manolopoulos, Y. Theodoridis, M. Vassilakopoulos, Closest pair queries in spatial databases, Proceedings of ACM SIGMOD International Conference on Management of Data, 2000, pp. 189–200.
[11] R.H. Guting, An introduction to spatial database systems, Special Issue on Spatial Database Systems of VLDB J. 3 (4) (1994) 1–32.
[12] E.M. Remgold, J. Nievergelt, N. Deo, Combinatorial algorithms: Theory and Practice, Prentice-Hall Inc., Englewood Cliffs, NJ, 1977.
[13] B. Seeger, P.A. Larson, R. McFadyen, Reading a set of disk pages, Proceedings of the 19th Conference on VLDB, 1993, pp. 592–603.
[14] S. Liao, M.A. Lopez, S.T. Leutenegger, High dimensional similarity search with space filling curves, Proceedings of the International Conference on Data Engineering, 2001, pp. 615–622.
[15] M.F. Mokbel, W.G. Aref, I. Kamel, Performance of multi-dimensional space-filling curves, Proceedings of the 10th ACM International Symposium on Advances in GIS, 2002, pp. 149–154.
[16] B. Seeger, H.P. Kriegel, The buddy-tree: an efficient and robust access method for spatial data base systems, Proceedings of the 16th Conference on VLDB, 1990, pp. 590–601.