Generating and Analyzing N-dimensional Hilbert Cell

FENG Yucai  LI Chenyang

ABSTRACT In this paper, two algorithms are presented for generating two code scan lists of an N-dimensional Hilbert cell, and a formal proof of the backward encoding algorithm is given. On the basis of the self-similarity properties of a Hilbert curve, this paper gives a novel algorithm for generating a static evolvement rule table through analyzing a Hilbert cell. By looking up the static evolvement rule table, the N-dimensional Hilbert mappings are efficiently implemented.

KEYWORDS algorithm; Hilbert curve; space-filling curve

CLC NUMBER TP751

Introduction

In 1890, Italian mathematician G. Peano presented a family of curves which pass through all points in a space\(^2\). Since the publication of this discovery, many researchers have worked on this problem. Curves of this type have been called Peano curves or space-filling curves. It is well known that space-filling curves describe a method of one-to-one mapping between N-dimensional space and one-dimensional space. Such mapping is of interest in a number of application domains including image processing\(^3\,\,4\) and, more recently, in the indexing of multi-dimensional data\(^5\,\,6\). Among these space-filling curves, Z-order curve\(^7\) and Hilbert curve\(^8\) are the most famous, for Z-order curve is the simplest in encoding and Hilbert curve achieves the best clustering\(^9\). For the most cases, interest in applications of space-filling curves has been confined to Z-order mapping due to its simple. Little work has been devoted to techniques of Hilbert curve in more than 2 dimensions due to the complexity. Kamata, Richard and Yukihiro presented a representative N-dimensional Hilbert mapping algorithm\(^10\), which analyzes a Hilbert curve from top to bottom, but its time complexity is much higher than Z-order mapping, for the method has to analyze the Hilbert curve dynamically when mapping. An N-dimensional Hilbert cell is the most basic element of a Hilbert curve, and the most important characteristic of the Hilbert curve is self-similarity. Thus by studying the information hiding in an N-dimensional Hilbert cell, we can obtain the evolving rules of an N-dimensional Hilbert curve and then implement an N-dimensional Hilbert mapping. In this paper, we introduce the evolving rules of generating an N-dimensional Hilbert cell (i.e., code scan list) from \(n-1\) dimensions to \(n\) dimensions. In the light of the rules, we present a pair of algorithms for generating an N-dimensional Hilbert cell, which include the forward encoding algorithm and the backward encoding algorithm. By use of on them, we can perform efficiently mappings in two directions. Furthermore, we give a formal proof of the backward encoding algorithm. On the basis of the cell code list, we propose a novel algorithm for generating a static evolvement rule table, through which we can generate N-dimensional Hilbert curves and implement N-dimensional Hilbert mappings efficiently. Using a static evolvement rule table, we
have analyzed a Hilbert curve from bottom to top, the result is better than that obtained with the Kamata method. So it can greatly improve efficiency of N-dimensional Hilbert mappings.

1 Available approaches

There are two ways to implement Hilbert mapping. One is table driven approach; the other is calculation approach.

The table driven approach performs mappings through looking up the code scan list. An iterative table driven version which does enable the mapping of 1-dimensional (1D) values to 2-dimensional (2D) points is given by Fisher[32]. A table driven version which additionally facilitates the inverse mapping, from 2D points to 1D values, is given by Cole[10, 11]. Guohua Jin and John Mellor-Crummey presented the framework SFCGen to efficiently generate multidimensional space-filling curves[12]. The main shortcoming of the table driven approach is high space complexity.

The calculation approach performs mappings through a one-to-one calculation. With the Butz algorithm[31] the coordinates of a point corresponding to an arbitrary ordinal position on the curve can be calculated. A nonrecursive algorithm is given by Faloutsos and Roseman[33], which performs mappings through analyzing the relationship between Z-order code and Hilbert code. Liu and Schrack[35] provided formulae for mapping from coordinates of 2D points to 1D values, and they extended it to 3-dimensional (3D) mapping in Reference [16].

Most of the above are confined to the 2D case. In recent years, algorithms for N-dimensional Hilbert mapping have been proposed, such as the Kamata algorithm[38] and the Lawder algorithm[37]. The Kamata algorithm implements mappings through analyzing a Hilbert curve dynamically, and the Lawder algorithm implements mappings through constructing state diagrams and generator table. But their time and space complexities are still much higher than that of Z-order mapping. That is the reason why Hilbert mapping has not been used widely.

2 Analyzing a Hilbert curve

2.1 Concepts of a Hilbert curve

It is well known that cells replicate themselves according to the genetic information to grow into a complex organism. Generating a Hilbert curve can be described as a similar course.

Firstly, let $R^N$ be an N-dimensional space. The coordinates in $R^N$ are denoted as $X_1, \ldots, X_N$, and the mapped coordinate value in $R^1$ as Hilbert code ($H$-code).

Here, a Hilbert curve is called as an N-dimensional $m$th-generation Hilbert curve, if it passes through $2^m \times 2^m \times \cdots \times 2^m$ (i.e., $2^{mn}$) subhypercubes of a hypercube in $R^N$. In general, it can be expressed by using $H_m^N$. Fig. 1 shows the $H_1^2$, $H_2^2$ and $H_3^2$.

Fig. 1 Original Hilbert curves

![Original Hilbert curves](image)

Definition 1 (N-dimensional Hilbert cell; $C^N$)

An N-dimensional Hilbert cell is an N-dimensional 1th-generation Hilbert curve (i.e., $H_1^N$).

Let $C^N$ be an N-dimensional Hilbert cell, it is shown in Fig. 2.

Fig. 2 Examples of 2D and 3D Hilbert cells

![Examples of 2D and 3D Hilbert cells](image)

Definition 2 (N-dimensional Hilbert gene)

The Hilbert gene is a list of coordinate transformation commands, which directs how to generate $H_m^N$ from $H_{m-1}^N$.

The coordinate transformation includes two types; exchange ("\leftrightarrow"); reverse ("\uparrow\downarrow"). Fig. 3(a) shows
an initial state of \( C \), Fig. 3 (b) shows the transformed result of \( C \) through exchange \((X, \leftrightarrow X_2)\), and Fig. 3 (c) shows the transformed result of \( C \) through reverse \((X, \leftrightarrow X_2; X', X')\).

Fig. 3 Examples of coordinate transformation

We describe a gene list by using \( G[i] \), and use \( H[i] \) to express the transformed result of \( H[i] \). Here \( i \) denotes the sequence number of Hilbert order in \( R_i \) (i.e., the location of \( H[i] \) in \( H[i] \)). Table 1 is an example of a 2D Hilbert gene list (‘–’ denotes no transformation).

Table 1 \( G[i] \): 2D Hilbert gene list

| Location | Exchange | Reverse |
|----------|----------|---------|
| 0        | \( X_1 \leftrightarrow X_2 \) | –       |
| 1        | –        | –       |
| 2        | –        | –       |
| 3        | \( X_1 \leftrightarrow X_2 \) | \( X', X' \) |

The list shows how to implement \( H[i] \rightarrow H[i] \). For example, \( H[1] \rightarrow H[0] \) through \( X_1 \leftrightarrow X_2 \), which is shown in Fig. 3(b).

2.2 Growing of a Hilbert curve

\( C^N \) is the most basic element of a Hilbert curve like the cell of an organism. And \( G[i] \) is formulated to construct a complex Hilbert curve from \( C^N \) like the gene list of an organism. Now, a 2D Hilbert curve is taken as an example to describe the course of growing of a Hilbert curve.

From Fig. 4 (b), we can see that \( H[i] \) is generated through \( X_1 \leftrightarrow X_2 \) of \( C \), and \( H[i] \) is constructed by assembling the four of \( H[i] \) \((i = 0, 1, 2, 3)\).

Assemble \( H[i] \) according to the order of \( i \) to generate \( H[i] \), then set \( H[i] \) as the initial element and repeating the above steps, we can generate \( H[i] \), shown in Fig. 4(c).

From this example, we know how to generate an \( N \)-dimensional Hilbert curve from \( H[i] \) (i.e., from bottom to top). Given \( C^N \) and \( G[i] \), it can decide a Hilbert curve wholly.

3 Generating \( N \)-dimensional Hilbert cell

In this section, we describe the details of our algorithms for generating an \( N \)-dimensional Hilbert cell. We introduce two evolving algorithms; forward encoding algorithm and backward encoding algorithm. The forward encoding algorithm can transform a point from \( R^N \) to \( R^1 \) efficiently. And the backward encoding algorithm transforms a point from \( R^1 \) to \( R^N \) more efficiently.

3.1 Generating \( N \)-dimensional Hilbert cell

We look at a Hilbert cell code list from two different perspectives; forward encoding (i.e., encoding from \( R^N \) to \( R^1 \)), as shown in Table 2; and backward encoding (i.e., encoding from \( R^1 \) to \( R^N \)), as shown in Table 3.

The forward encoding algorithm can be described as follows:

\[
C^i = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \quad \text{for } i = 0, 1, 2, \ldots, n = 2, 3, \ldots
\]

\[
C^i = \text{Merge} \left( \begin{pmatrix} 0 & C^{i-1} \\ 1 & \text{Reverse}(C^{i-1}) \end{pmatrix} \right)
\]

Fig. 4 \( H[1] \rightarrow H[2] \rightarrow H[3] \) according to \( G[i] \)
The backward encoding algorithm can be described as follows:

\[
\begin{align*}
C^1 &= \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\
C^n &= \text{Merge} \left( 1 \oplus \text{Reverse} \left( X_{n-1} \right) \right) \\
&\quad \cdot C^{n-1} \quad \cdot n = 2, 3, \ldots
\end{align*}
\]

where the mark ‘$\oplus$’ denotes concatenation, e.g., $11 \oplus 00 = 1100$ and the function $\text{Reverse}(x)$ denotes bit-by-bit reverse, e.g., $\text{Reverse}(10) = 01$.

The generating course of a 3D cell using forward encoding algorithm can be described as follows:

\[
\begin{align*}
X_1 &\quad X_2 &\quad X_3 \\
C_1[0] &= 0 &\quad C_1[0][0] &= 0 &\quad C_1[0][0][0] &= 000 \\
C_1[1] &= 1 &\quad C_1[1][0] &= 0 &\quad C_1[1][0][0] &= 001 \\
C_2[0][1] &= 11 &\quad C_2[0][1][0] &= 011 &\quad C_2[0][1][0][0] &= 010 \\
C_2[1][1] &= 10 &\quad C_2[1][1][0] &= 010 &\quad C_2[1][1][0][0] &= 011 \\
C_3[0][0][1] &= 111 &\quad C_3[0][1][0][1] &= 100 &\quad C_3[1][1][1][1] &= 101
\end{align*}
\]

The generating course of a 3D cell using backward encoding algorithm can be described as follows:

**Step 1** $0 \oplus C^0$ (i.e., add dimension $X_1; 0$)

\[
\begin{align*}
\text{H-code} &\quad X_1 &\quad X_2 \\
0 &\quad 0 &\quad 0 &\quad 0 &\quad 0 \\
1 &\quad 0 &\quad 1 &\quad 0 &\quad 0 \\
2 &\quad 0 &\quad 1 &\quad 1 &\quad 0 \\
3 &\quad 1 &\quad 0 &\quad 0 &\quad 0 \\
\end{align*}
\]

**Step 2** $1 \oplus \text{Reverse} \left( X_2 \right)$ (i.e., reverse dimension $X_2$ of $C^0$ and add dimension $X_3; 1$)

\[
\begin{align*}
\text{H-code} &\quad X_1 &\quad X_2 &\quad X_3 \\
0 &\quad 0 &\quad 0 &\quad 0 &\quad 0 \\
1 &\quad 0 &\quad 1 &\quad 0 &\quad 0 \\
2 &\quad 1 &\quad 1 &\quad 0 &\quad 0 \\
3 &\quad 0 &\quad 1 &\quad 0 &\quad 0 \\
\end{align*}
\]

**Step 3** Merge (i.e., sequential replication)

\[
\begin{align*}
\text{H-code} &\quad X_1 &\quad X_2 &\quad X_3 \\
0 &\quad 0 &\quad 0 &\quad 0 &\quad 0 \\
1 &\quad 0 &\quad 1 &\quad 0 &\quad 0 \\
2 &\quad 1 &\quad 1 &\quad 0 &\quad 0 \\
3 &\quad 0 &\quad 1 &\quad 0 &\quad 0 \\
\end{align*}
\]

The forward scan can map from $\mathbb{R}^N$ into $\mathbb{R}^p$ efficiently, and the backward scan can map from $\mathbb{R}^p$ to $\mathbb{R}^N$ more efficiently. The forward encoding algorithm is space-efficient, but difficult to express on computer. So we focus on the backward encoding algorithm. In the case of backward encoding algorithm, we express a cell code list by using a 1-dimensional integer array (32 bits).

$C^n[H\text{-code}] = X_N \cdots X_1$

$H\text{-code}$ is the coordinate value in $\mathbb{R}^1$. $X_N, \ldots, X_1$ are the coordinate values in $\mathbb{R}^N$. For example, $C^1$ can be described as $C^1[0] = 000$, $C^1[1] = 001$, $C^1[2] = 011$, $C^1[3] = 010$, $C^1[4] = 110$, $C^1[5] = 111$, $C^1[6] = 101$, $C^1[7] = 100$, as shown in Table 3.

### 3.2 Analyzing of algorithm’s correctness

In this subsection, we give four basic properties of an $N$-dimensional Hilbert cell. It can be seen from the definition of an $N$-dimensional Hilbert cell.

The properties are described as follows.

**Property 1** $C^N$ is symmetric positively and negatively about dimension of $X_e$.

**Property 2** The difference between the coordinates values of the entry point and the exit point is only in one dimension.
Property 3 Points of $C^N$ are sequential and strictly monotone in $R^1$ and unique in $R^N$.

Property 4 The neighbor in $R^1$ (i.e., $H$-code) must be neighbor in $R^N$.

The criterion of the correctness is whether $C^N$ generated by the algorithms has the four properties. Here, we only give the proof of the backward encoding algorithm using mathematical induction.

Proof

Case $N=2$:

(1) For the values of $X_{-1} \cdots X_1$; 0,1 and 1,0 are reverse, the points $(00,01; 11,10)$ are positive and negative symmetry about dimension of $X_2$. The positive and negative parts are divided by $'1'$.

(2) (entry point, exit point) of $C^2$ is $(00,10)$, which is different only in dimension of $X_2$.

(3) The points $(0,1,2,3)$ in $R^1$ of $C^2$ are sequential and strictly monotone.

The points $(00,01,11,10)$ in $R^2$ are unique.

(4) It is obvious that $C^2$ agrees with property 4 (e.g., 1 is neighbor of 0, 2 in $R^1$, the corresponding point 01 is neighbor of 00, 11 in $R^2$ too).

So $C^2$ agrees with the four properties.

Case $N=n-1$:

Supposing that $C^{n-1}$ agrees with the four properties.

Case $N=n$:

(1) For property 1 of $C^{n-1}$, the points in $R^{n-1}$ of $C^{n-1}$ can be described as $(0V_1, 0V_2, \cdots, 0V_{2^{n-2}}, 1V_{2^{n-2}}, \cdots, 1V_2, 1V_1)$. The positive and negative parts are divided by $'1'$.

The Step 1 of the backward encoding algorithm is $0 \oplus C^{n-1}$, so the result of the step 1 is $(00V_1, \cdots, 00V_{2^{n-2}}, 01V_{2^{n-2}}, \cdots, 01V_1)$. The Step 2 of the backward encoding algorithm is $1 \oplus \text{Reverse}(X_{n-1}$ of $C^{n-1})$, so the result of the Step 2 is $(11V_1, \cdots, 11V_{2^{n-2}}, 10V_{2^{n-2}}, \cdots, 10V_1)$. The Step 3 of the backward encoding algorithm is merge, so the result of $C^n$ is $(00V_1, \cdots, 00V_{2^{n-2}}, 01V_{2^{n-2}}, \cdots, 01V_1; 11V_1, \cdots, 11V_{2^{n-2}}, 10V_{2^{n-2}}, \cdots, 10V_1)$. For the values of $X_{n-1} \cdots X_1; \cdots, 1V_1, 1V_1, \cdots, 1V_{2^{n-2}}, 0V_{2^{n-2}}, \cdots, 0V_1$ are reverse too, we can draw a conclusion that points are positive and negative symmetry about dimension of $X_n$. So $C^n$ agrees with property 1.

(2) (entry point, exit point) of $C^n$ is $(00V_1, 10V_1)$, which is different only in dimension of $X_n$. So $C^n$ agrees with property 2.

(3) It is obvious that the points of $(0,1,\cdots, 2^{n-1}-1)$ in $R^1$ of $C^n$ are sequential and strictly monotone.

For property 3 of $C^{n-1}$, $V_1, V_2, \cdots, V_{2^{n-2}}$ must be unique. As a result, The points of $(00V_1, \cdots, 00V_{2^{n-2}}, 01V_{2^{n-2}}, \cdots, 01V_1; 11V_1, \cdots, 11V_{2^{n-2}}, 10V_{2^{n-2}}, \cdots, 10V_1)$ in $R^n$ are unique. So $C^n$ agrees with property 3.

(4) For $(0V_1, 0V_2, \cdots, 0V_{2^{n-2}}, 1V_{2^{n-2}}, \cdots, 1V_2, 1V_1)$ of $C^{n-1}$ are agree with property 4, $V_1, V_2, \cdots, V_{2^{n-2}}$ must be agree with property 4. That is to say $V_i$ must be neighbor of $V_{i-1}$ and $V_{i+1}$ in $R^{n-2}$.

From above conclusion, it is obvious that $(00V_1, \cdots, 00V_{2^{n-2}}, 01V_{2^{n-2}}, \cdots, 01V_1; 11V_1, \cdots, 11V_{2^{n-2}}, 10V_{2^{n-2}}, \cdots, 10V_1)$ of $C^n$ is agree with property 4.

In conclusion, $C^n$ agrees with the four properties too, when $C^n$ agrees with the four properties.

Now, we have therefore proved that $C^N (N=1,2,3,\cdots)$ generated by the backward encoding algorithm agrees with the four properties.

4 Generating gene list by analyzing N-dimensional Hilbert cell

An N-dimensional Hilbert cell is the most basic element of a Hilbert curve. And the most important characteristic property of the Hilbert curve is self-similarity. Thus by studying the information hiding in an N-dimensional Hilbert cell, we can obtain the evolving rules of an N-dimensional Hilbert curve (i.e., Hilbert gene).

There are two steps to obtain the Hilbert gene; the first step is confirming the entry and exit of $H^x[i]$; the second step is generating $G[i]$ by analyzing the entry and exit of $H^y[i]$.
4.1 Confirming the entry and exit of $H^*_i$

The algorithm for confirming the entry and exit of $H^*_i$ is based on analyzing a Hilbert cell. We express the entry and exit of $H^*_i$ by using two integers:

$$H^*_i[0] = X_{N-i} \cdots X_1$$
$$H^*_i[1] = X_{N-i} \cdots X_1$$

where $H^*_i[0]$ is the entry of $H^*_i$; and $H^*_i[1]$ is the exit of $H^*_i$; 'i' is the order in $R^1$; $X_N, \cdots, X_1$ is the coordinate values in $R^N$.

Now, we give the algorithm for confirming the entry and exit of $H^*_i$ ($i=0, \cdots, 2^N-1$) as follows;

Algorithm 1:
//positive part
if (i < $2^N$/2)
{
if (i == 0)
{
$H^*_i[0] = C^*_0$;
$H^*_i[1] = (C^*_0 \cdot C^*_1)$;
}
else
{
$H^*_i[0] = H^*_i[i-1][1] \cdot (C^*_0 \cdot C^*_1)$;
if ($H^*_i[0] \& (C^*_0 \cdot C^*_1[i+1]) = C^*_0 \cdot (C^*_0 \cdot C^*_1[i+1])$)
{
//the entry of $H^*_i$ can connect to $H^*_i[i+1]$ directly
$H^*_i[1] = H^*_i[0] \cdot (C^*_0 \cdot C^*_1[i+1])$;
}
else
{
// the entry of $H^*_i$ can connect to $H^*_i[i+1]$ directly
$H^*_i[1]$ has $N-1$ alternative options;
}
}
else
{
if ($i = = 2^N$/2)
{
$H^*_i[0] = H^*_i[2^N-i-1][1] \cdot (0x00000001 <<<n-1)$;
$H^*_i[1] = H^*_i[i][0] \cdot (H^*_i[2^N-i-1][0] \cdot H^*_i[2^N-i-1][1])$;
}
else
{
$H^*_i[0] = H^*_i[i-1][1] \cdot (H^*_i[2^N-i-1][0] \cdot H^*_i[2^N-i-1][1] \cdot H^*_i[2^N-i-1][1])$;
}
}

The mark ' \cdot ' is XOR ($0 \cdot 1 = 1; 0 \cdot 0 = 0; 1 \cdot 1 = 0$), and the mark ' & ' is AND ($0 \& 1 = 0; 0 \& 0 = 0; 1 \& 1 = 1$).

4.2 Generating gene list

Now, we can confirm the entry and exit of $H^*_i$ ($i=0, \cdots, 2^N-1$). Next, we give an algorithm for obtaining the gene list by analyzing the entry and exit of $H^*_i$. In section 3, we introduce that there are two types of coordinate transformation: exchange and reverse. Here, We express exchange transformation by using $G[i][0]$, and express reverse transformation by using $G[i][1]$. The algorithm for obtaining exchange transformation can be described as follows;

$G[i][0] = (C^*_0 \cdot C^*_1[2^N-i-1]) \cdot H^*_i[i][1]$;

For example, Entry-exit of $C^*_i$ is in $X_i$, for the entry of $C^*_i$ (i.e., $C^*_i[0]$) is 000, and the exit of $C^*_i$ (i.e., $C^*_i[7]$) is 100. Entry-exit of $H^*_i$ is in $X_i$, for the entry of $H^*_i[0]$ is 000, and the exit of $H^*_i[0]$ is 001. So the exchange transformation must occur between $X_i$ and $X_i$ (i.e., $G[0][0]=101$).

The algorithm for obtaining reverse transformation can be described as follows;

$G[i][1] = C^*_i[0] \cdot H^*_i[i][0]$;

For example, the entry of $C^*_i$ (i.e., $C^*_i[0]$) is 000, and the entry of $H^*_i[3]$ is 101. So the reverse transformation must occur in $X_3$ and $X_3$ (i.e., $G[3][1]=101$).
It can be expressed as follows:

\[
\begin{align*}
G[0][0] &= 101, \quad G[0][1] = 000; \quad G[1][0] = 110, \quad G[1][1] = 000; \\
G[2][0] &= 000, \quad G[2][1] = 000; \quad G[3][0] = 101, \quad G[3][1] = 101; \\
G[4][0] &= 101, \quad G[4][1] = 000; \quad G[5][0] = 000, \quad G[5][1] = 000; \\
G[6][0] &= 110, \quad G[6][1] = 000; \quad G[7][0] = 101, \quad G[7][1] = 101.
\end{align*}
\]

5 Hilbert mapping

Now, we take a 2D Hilbert mapping as an example and give main steps of our method. Given \(C_2[0]=(00,01,11,10)\) and \(G[i] \) (i.e., 3D Hilbert gene list, as shown in Table 1).

\'i\' is the H-code in \(R^2\); \(C_2[0]=00\) is the coordinate values of \(X_2X_1\) in \(R^2\);

Given that there is a point \(p=(X_2, X_1)\) in \(R^2\), and it’s mapping in \(R^1\) is \(H\). And supposing it is a 2-dimensional 3rd-generation Hilbert curve.

5.1 Mapping from \(R^2\) to \(R^1\)

Given that there is a point \(p=(001,010)\) in \(R^2\) \((X_2, X_1)\), and it’s mapping in \(R^1\) is \(H=XXXXXXXX\); \(p\) is a point in \(H_3\) (i.e., 2D 3rd-generation Hilbert curve), as shown in Fig. 5.

![Fig. 5 Mapping between R² and R¹](image)

The course of mapping can be described as follows:

(1) Getting the upper bits of \(p=(001,010)\); \(z=00\);

(2) Looking up \(z\) in \(C_2[i]; i=0\);

(3) Appending \(i=0\) to \(H, H=00XXXX\);

(4) Getting \(G[i]; G[0]=X_1\leftrightarrow X_2\); (5) Performing coordinate transformations; \(p=(10,01)\);

if \(X_2=01, X_1=10\), do the exchange of \(X_1\leftrightarrow X_2\), the result is \(X_2=10, X_1=01\), so \(p=(10,01)\).

The following steps are an iterative course:

(6) Getting the upper bits of \(p=(10,01); z=10\);

(7) Looking up \(z\) in \(C_2[i]; i=3\);

(8) Appending \(i=3\) to \(H; H=0011XXX\);

(9) Getting \(G[i]; G[3]=X_1\leftrightarrow X_2\), \(X'_1\), \(X'_2\);

(10) Performing coordinate transformations; \(p=(0,1)\);

Note: if \(X_2=0, X_1=1\) do the exchange of \(X_1\leftrightarrow X_2\), the result is \(X_2=1, X_1=0\), so \(p=(1,0)\);

then \(X_2=1, X_1=0,\) do the reverse of \(X'_1, X'_2\), the result is \(X_2=1, X_1=0\), so \(p=(0,1)\).

(11) Getting the upper bits of \(p=(0,1); z=01\);

(12) Looking up \(z\) in \(C_2[i]; i=1\);

(13) Appending \(i=1\) to \(H; H=001101\);

The result is 001101.

5.2 Mapping from \(R^1\) to \(R^2\)

Given that there is a point \(H=001101\) in \(R^1\), and it’s mapping in \(R^2\) is \(p(X_2, X_1)=(XXX, XXX)\), as shown in Fig. 5. The course of mapping can be described as follows:

(1) Getting the lower 2 bits of \(H=001101; i=01=1\);

(2) Getting \(C_2[1]; C_2[1]=01\);

(3) Appending \(C_2[1]\) to \(X_2, X_1\); \(p=(XX0, XX1)\);

(4) Getting the lower 2 bits of \(H=0011; i=11=3\);

(5) Getting \(G[3]; X_1((X_2, X'_1, X'_2)\);

(6) Performing coordinate transformations; \(p=(XX0, XX1)\);

Note: if \(X_2=XX0, X_1=XX1\) do the exchange of \(X_1\leftrightarrow X_2\), the result is \(X_2=XX1, X_1=XX0\), so \(p=(XX1, XX0)\);

then \(X_2=XX1, X_1=XX0\) do the reverse of \(X_1, X'_2\), the result is \(X_2=XX0, X_1=XX1\), so \(p=(XX0, XX1)\).

The following steps are an iterative course;

(7) Getting \(C_2[3]; C_2[3]=10\);
(8) Appending \( C^3[3] \) to \( X_1, X_1^* \); \( p = (X_{10}, X_{01}) \);

(9) Getting 2 bits of \( H = 00; i = 00 = 0; \)

(10) Getting \( G[0]; X_1 \leftrightarrow X_2; \)

(11) Performing coordinate transformations; \( p = (X_{01}, X_{10}) \);

Note: \( X_2 = X_{10}, X_1 = X_{01} \) do the exchange of \( X \leftrightarrow X_2 \), the result is \( X_i = X_{01}, X_{10} \), so \( p = (X_{01}, X_{10}) \).

(12) Getting \( C[-i]; C[0] = 00; \)

(13) Appending \( C[-0] \) to \( X_i \), \( X_1; \) \( p = (001, 010) \).

The result is \((001, 010)\).

### 6 Discussion

Supposing \( n \) is the point number of an \( N \)-dimensional cell, \( N \) is the number of dimensions, and \( m \) is the number of the generation. The relationship between \( n \) and \( N \) can be described as

\[
n = 2^N
\]

In the computation, the Kamata method requires \((N\times U + N^2)\times 2^{N+4}\) bytes \((U \) is a natural number that satisfies \( 2^{m-1} \leq N2^{m-1} \leq 2^U \)).

Compared with it, the space costs of our algorithm are \(3\times 4\times 2^N\) bytes.

For example, in the case of \( N = 2, 3, 4, 5, 10, 20 \), Table 4 shows a comparison of space-efficiency with our method and the Kamata method.

| \(N\) | Ours(bytes) | Kamata's(bytes) |
|------|-------------|-----------------|
| 2    | 48          | 8               |
| 3    | 96          | 84              |
| 4    | 192         | 576             |
| 5    | 384         | 3,840           |
| 10   | 12,288      | 15,073,280      |
| 20   | 12,582,912  | 60,473,139,527,680 |

From Table 4, we can see that the storage size of the Kamata method is too large to implement. As a result, the Kamata method has to generate looking up tables dynamically. So we just list time costs of ours, as shown in Table 5.

| \(N\) | Time costs(ms) |
|------|----------------|
| 2    | 0.026          |
| 3    | 0.027          |
| 4    | 0.030          |
| 5    | 0.034          |
| 10   | 0.277          |
| 20   | 0.651          |

### 7 Conclusions

In this paper, we presented two algorithms for generating an \( N \)-dimensional Hilbert cell. And we give a formal proof of the backward encoding algorithm. By analyzing the \( N \)-dimensional Hilbert cell, we propose a novel algorithm for generating Hilbert gene, which analyzes a Hilbert curve from bottom to top, based on a static evolvement rule table \((i.e., \text{the Hilbert gene})\). From Section 6, we can see that the time complexity of our mapping algorithm is \( O(m) \). It is the same as \( Z \)-order mapping in general.

### REFERENCES

1. Peano G (1890) Sur une courbe qui remplit toute une aire plane. *Math. Ann.* 36: 157-160
2. Biswas S (2004) One-dimensional B-B polynomial and Hilbert scan for graylevel image coding. *Pattern Recognition*, 37: 789-800
3. Stevens R J, Lehar A F, Preston F H (1983) Manipulation and presentation of multi-dimensional image data using the peano scan. *IEEE Trans. Pattern Anal. Machine Intell.*, PAMI-5: 520-526
4. Chen H, Chang Y (2005) Neighbor-finding based on space-filling curves. *Information Systems*, 30: 205-226
5. Mokbel M F, Aref W G (2001) Irregularity in multi-dimensional space-filling curves with applications in multimedia databases. The 10th ACM SIGMIS Information and Knowledge Management, Georgia, USA.
6. Hilbert D (1891) ber die stetige Abbildung einer Linie
auf ein Flächenstück. Math. Ann., 38, 459-460

7 Moon B, Jagadish H V, Faloutsos C, et al. (2001) Analysis of the clustering properties of the Hilbert space-filling curve. IEEE Transactions on Knowledge and Data Engineering, 13, 124-141

8 Kamata S, Eason R O, Bandou Y (1999) A new algorithm for N-dimensional Hilbert scanning. IEEE Trans. on Image Processing, 8(7)

9 Fisher A J (1986) A new algorithm for generating Hilbert curves. Software-Practice and Experience, 16, 5-12

10 Cole A J (1986) Direct transformations between sets of integers and Hilbert polygons. International Journal of Computer Mathematics, 20, 115-122

11 Cole A J (1987) Compaction techniques for raster scan graphics using space-filling curves. The Computer Journal, 30, 87-92

12 Jin G, Mellor-Crummey J (2005) A framework for efficient generation of multi-dimensional space-filling curves by recursion. ACM Transactions on Mathematical Software (TOMS) archive, 31

13 Butz A R (1971) Alternative algorithm for Hilbert’s space-filling curve. IEEE Transactions on Computers, 20, 424-426

14 Faloutsos C, Roseman S (1989) Fractals for secondary key retrieval. The 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Philadelphia.

15 Liu X, Schrack G (1996) Encoding and decoding the Hilbert order. Software-Practice and Experience, 26, 1335-1346

16 Liu X, Schrack G (1997) An algorithm for encoding and decoding the 3D Hilbert order. IEEE Transactions on Image Processing, 6, 1333-1337

17 Lawder J K (2000) Using state diagrams for Hilbert curve mappings. Technical Report JL2/00, Birkbeck College, University of London.

Notes to Contributors

Contributions are welcomed on one of the following subjects or in related areas:

- GIS
- GPS
- RS
- Cartology
- Geodynamic
- Geo-surveying
- Photogrammetry
- Graphics
- Physical geo-surveying
- Engineering surveying
- Mapping apparatus

Paper submitted in the electronic text on a diskette should be sent along with two printed copies. The main text should be preceded by the abstract of no more than 300 words, followed by key words. Full references should be listed in the order of the citations in the text under the heading “References”, guided by standard publication format. The name of the fund and project series number for articles of funded projects should also be given.

Contribution submitted for publication will be peer reviewed, i.e., the entire article will be examined by qualified expert independent of the author before publication.

The authors assume sole responsibility for their dissertations. Canned theses are not allowed. Once the paper is submitted, the editors are authorized to make necessary literate processing. Authors who receive no notice in three months after submission may contact us for inquiry.