A Closed-Form Algorithm for Converting Hilbert Space-Filling Curve Indices*

Chih-Sheng Chen[†,‡], Shen-Yi Lin[†], Min-Hsuan Fan[†], and Chua-Huang Huang [†]

Abstract—**We use the tensor product theory to formulate a closed-form algorithm for converting Hilbert space-filling curve indices of individual points. A two-dimensional Hilbert space-filling curve is specified as a permutation which rearranges two-dimensional $2^n \times 2^n$ data elements stored in the row-major order as in C language or the column-major order as in FORTRAN language to the order of traversing a two-dimensional Hilbert space-filling curve. The closed-form algorithm converts the row-major index or the column index of a single point to the index of Hilbert space-filling curve order. The time complexity of the closed-form algorithm is a function of the length of the binary representation of the index and its space complexity is bounded by a constant. In addition, the closed-form tensor product formula can be directly translated into computer programs which can be used in various applications such as image compression. The process of program generation is explained in the paper.**

Keywords: Hilbert space-filling curve, tensor product, closed-form, program generation

1 Introduction

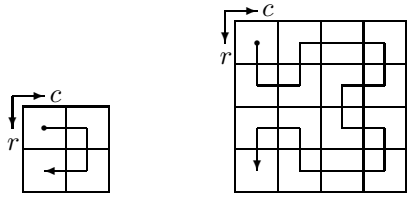
The Hilbert space-filling curve is a space-filling curve that traverses every point once on a two-dimensional $2^n \times 2^n$ square grid and without crossing the path [1]. Typically, a $2^n \times 2^n$ Hilbert space-filling curve is recursively constructed from $2^{n-1} \times 2^{n-1}$ Hilbert space-filling curve. For example, the 2×2, H_1, and the 4×4, H_2, Hilbert space-filling curves are shown in Figure 1. H_2 is a curve connecting four copies of H_1 in different orientations. It is first described by David Hilbert in 1891 and used to express the locality of two-dimensional data in a one-dimensional space [2]. Digital data stored in the Hilbert space-filling curve order have important advantages of locality efficiency between neighboring points, so various applications use it to arrange data elements in a linear order and preserves spatial locality. In image processing, applying the Hilbert space-filling curve to images, and get better compression rate than that of a raster scanned image [3, 4, 5]. Using the properties of the Hilbert space-filling curve, VLSI component layout [6, 7] can fully exploit the available silicon area and optimal area complexity. Multi-dimensional data structure R-tree uses the Hilbert space-filling curve indexing to cluster the data and makes searching more efficient [8, 9, 10, 11, 12]. Furthermore, the Hilbert space-filling curve is beneficial in the design of small antennas because it can pack the maximum length of a line in a given area [13]. Recently, three-dimensional space-filling curve are used to detect and classify functional Magnetic Resonance Imaging (fMRI) activation patterns of clinical medical images [14].

Since Hilbert space-filling curves are extensively used in various applications, different methods of curve generation have been suggested for reducing computation time and/or space complexities. Jagadish analyzes the clustering properties of Hilbert space-filling curves [15]. He shows that the Hilbert space-filling curve order can achieve the best clustering, *i.e.*, it is the best space-filling curve order in minimizing the number of clusters. Butz uses an iterative algorithm to compute a mapping function with byte-oriented technique such as exclusive OR, shifting, *etc.* [16, 17]. Sagan presents an arithmetic method for the generation of the nodes and produces an approximating polygon to represent the Hilbert space-filling curve [18]. Ohno and Ohyama describe Hilbert space-filling curves with the Lindenmayer system which be can used to generate self-similar fractals [19]. Quinqueton and Berthod present an algorithm for computing all addresses of scanning path by recursive procedure [20]. Kamata *et al.* propose a non-recursive algorithm for N-dimensional Hilbert space-filling curve using look-up tables [21, 22]. Using tensor product formulation present, we design both recursive and iterative coding algorithms which scan all space points of two-dimensional and three-dimensional Hilbert space-filling curves [23, 24].

Some application problems, such as finding nearest neighbor points and retrieving partial of satellite picture in geographic information system, are not required to scan all data elements of a Hilbert space-filling curve, *i.e.*, Liu and Schrack present a haphazard point mapping algorithm between one-dimensional and two-dimensional data [25]. This algorithm does not depend on look-up tables and is

*This work was supported in part by National Science Council, Taiwan, R.O.C. under grant NSC 92-2218-E-035-013.

[†]Department of Information Engineering and Computer Science, Feng Chia University, Taichung, Taiwan 40724 Email: {chenc, sylin, mfan, chh}@pmlab.iecs.fcu.edu.tw [‡]Department of Health Administration, Tzu Chi College of Technology, Hualien, Taiwan 97005 Email: chen@tccn.edu.tw



H_1 H_2

Figure 1: 2×2 and 4×4 Hilbert space-filling curves

faster then Fisher's interpretation [26]. Recently, Chen *et al.* design an encoding and decoding algorithms based on the replication of the Hilbert matrix [27]. These algorithms work better when the data area is smaller then the entire space.

In this paper, we use the tensor product theory to formulate a closed-form algorithm for converting Hilbert space-filling curve indices of individual points. A point on the $m \times n$ two-dimensional space is usually indexed as a pair of coordinates, (r, c), where $0 \leq r < m$ and $0 \leq c < m$. Usually, the data elements of $m \times n$ the two-dimensional are stored in computer memory of linear address space in the row-major order, as in C language, or in the column-major order, as in FORTRAN language. For $2^n \times 2^n$ two-dimensional space, conversion of a row-major or column-major index (r, c) to its corresponding Hilbert space-filling curve index h is expressed as a tensor product formula. The time complexity of the closed-form algorithm is a function of the length of the binary representation of the point index and its space complexity is bounded by a constant. In addition, the tensor product formula can be directly translated into a high-level language program with bit-wise operators and few subtractions.

The paper is organized as the following. We briefly review the algebraic theory of tensor product and other related operations in Section 2. In Section 3, we derive the closed-form algorithm for two-dimensional Hilbert space-filling curve based on previous research. Program generation of the closed-form Hilbert space-filling curve algorithm from the tensor product formula is explained in Section 4. The time and space complexities of the algorithm are discussed in Session 5. Concluding remarks and future works are given in Section 6.

2 Overview of Tensor Product Operations

In this section, we give an brief overview of the algebraic operations and some of their properties used in formulating the closed-form for Hilbert space-filling curve permutation. The operations explained include tensor product, direct sum, vector reversal, and stride permutation.

Tensor product is a matrix operation which builds a "large" matrix from two "small" matrices. It is defined as below:

Definition 2.1 (Tensor Product) *If A is an $m \times n$ matrix and B is an $p \times q$ matrix, then the tensor product of A and B is the block matrix obtained by replacing each element $a_{i,j}$ by $a_{i,j}B$, denoted by $A \otimes B$, is an $mp \times nq$ matrix defined as*

$$A \otimes B = \begin{bmatrix} a_{0,0}B_{p\times q} & \cdots & a_{0,n-1}B_{p\times q} \\ \vdots & \ddots & \vdots \\ a_{m-1,0}B_{p\times q} & \cdots & a_{m-1,n-1}B_{p\times q} \end{bmatrix}.$$

Let F^m be the vector space of m-tuples over field F and let $F^{m\times n}$ be the vector space of $m \times n$ matrices. The collection of elements $\{e_i^m | 0 \leq i < m\}$, where e_i^m is the vector with a one in the i-th position and zeros elsewhere, form the standard basis for F^m.

Definition 2.2 (Tensor Bases) *Let F^n be the vector space of n-tuples over the field F, a collection of elements $\{e_{i_1}^{n_1} \otimes e_{i_2}^{n_2} \otimes \cdots \otimes e_{i_k}^{n_k} | 0 \leq i_1 < n_1, 0 \leq i_2 < n_2, \cdots, 0 \leq i_k < n_k\}$ are called a tensor bases of $F^{n_1} \otimes F^{n_2} \otimes \cdots \otimes F^{n_k}$.*

A tensor basis can be linearized (or factorized) as below:

$$e_{i_1}^{n_1} \otimes e_{i_2}^{n_2} \otimes \cdots \otimes e_{i_k}^{n_k} = e_{i_1 n_2 \cdots n_k + \cdots + i_{k-1} n_k + i_k}^{n_1 n_2 \cdots n_k}.$$

The direct sum is a matrix operation that transforms several matrices into one block diagonal matrix, that is to say, the operation could build a "large" matrix from some "small" matrices .

Definition 2.3 (Direct Sum) *Let A and B be two matrices $m \times n$ and $p \times q$, respectively. The direct sum of A and B is an $(m + p) \times (n + q)$ matrix defined as*

$$A \oplus B = \begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix}.$$

If B is a $p \times q$ matrix, $I_n \otimes B$ is the direct sum of n copies of B, where I_n is the $n \times n$ identity matrix.

$$I_n \otimes B = \bigoplus_{k=0}^{n-1} B = \begin{bmatrix} B & & \\ & \ddots & \\ & & B \end{bmatrix}_{pn \times qn}.$$

Three permutations are used in formulating the closed-form of Hilbert space-filling curve permutation. They are stride permutation, reverse permutation, and Gray permutation.

Definition 2.4 (Stride Permutation) *A stride permutation L_n^{mn} is defined by*

$$L_n^{mn}(e_i^m \otimes e_j^n) = e_j^n \otimes e_i^m.$$

L_n^{mn} is referred to as the stride permutation which permutes the tensor product of two vector bases. If an $m \times n$ matrix is stored in the column-major order, its basis is isomorphic to $e_i^m \otimes e_j^n$. Stride permutation is exactly the transposition operation transforming the matrix from the column-major order allocation to the row-major order. Stride permutation also corresponds to the exchange of a coordinate system.

Definition 2.5 (Reverse Permutation) *A reverse permutation J_n is defined by*

$$J_n e_i^n = e_{(n-1)-i}^n.$$

J_n maps the basis element e_i^n to the basis element $e^n_{-(i+1) \ mod \ n}$. Reverse permutation corresponds the reverse of a coordinate system. It is also used in the definition of Gray permutation. Reverse permutation for a vector of length 2^k can be viewed as a binary complement operation, since $J_{2^k}(e^2_{i_{k-1}} \otimes \cdots \otimes e^2_{i_0}) = e^2_{\overline{i_{k-1}}} \otimes \cdots \otimes e^2_{\overline{i_0}}$, where $\overline{i_j}$ is the binary complement of i_j, $0 \le j < n$.

Definition 2.6 (Gray Permutation) *The n-bit Gray permutation G_{2^n} is defined as*

$$G_2 = I_2, \quad G_{2^n} = (I_{2^{n-1}} \oplus J_{2^{n-1}})(I_2 \otimes G_{2^{n-1}}).$$

J_n maps the basis element e_i^n to the basis element $e^n_{-(i+1) \ mod \ n}$. Reverse permutation corresponds the reverse of a coordinate system. It is also used in the definition of Gray permutation. Reverse permutation for a vector of length 2^k can be viewed as a binary complement operation, since $J_{2^k}(e^2_{i_{k-1}} \otimes \cdots \otimes e^2_{i_0}) = e^2_{\overline{i_{k-1}}} \otimes \cdots \otimes e^2_{\overline{i_0}}$, where $\overline{i_j}$ is the binary complement of i_j, $0 \le j < n$.

Definition 2.7 (Selection Operation) *A selection operation $E_{k,k}^{4^n,4^n}$ is a $4^n \times 4^n$, $0 \le k < 2^n - 1$, matrix with the k-th diagonal elements being 1 and zeros, elsewhere.*

$E_{k,k}^{4^n,4^n}$ is termed as a *selection operator* and selected one of the input elements which on which the computation is performed.

$$E_{k,k}^{4^n,4^n} = E_{r,r}^{2^n,2^n} \otimes E_{c,c}^{2^n,2^n},$$

where $k = r \times 2^n + c$.

There are some properties of tensor products, direct sums, stride permutations L_n^{mn}, and reverse permutations used in this paper. The readers are referred to [24] for these properties.

3 Closed-Form Algorithm for Hilbert Space-Filling Curve

In the previous papers [23, 24], we develop recursive and iteration tensor product formulations for two-dimensional and three-dimensional Hilbert space-filling curves. Suppose the points of a $2^n \times 2^n$ two-dimensional gird are initially stored in the row-major or column-major order. The recursive tensor product formula of rearranging the grid points in the Hilbert space-filling curve order is given as the following:

$$\begin{array}{rrcl} & H_1 & = & G_2, \\ n > 1: & H_n & = & (I_4 \otimes H_{n-1}) R_n G_n B_n \\ & & = & (I_4 \otimes H_{n-1}) \\ & & & (T_{2^{2(n-1)}} \oplus I_{2^{2(n-1)}} \oplus I_{2^{2(n-1)}} \oplus \overline{T}_{2^{2(n-1)}}) \\ & & & (G_2 \otimes I_{2^{2(n-1)}})(I_2 \otimes L_2^{2^n} \otimes I_{2^{n-1}}). \end{array}$$

The construction of the $2^n \times 2^n$ Hilbert space-filling curve is carried out in four steps. In the first step, operation B_n is to reallocate the initial row-major ordering data to 2×2 blocks. Secondly, G_n is to permute the blocks using 2×2 Gray permutation, and the third step, R_n is to rotate and reflect the block elements according to a given orientation, for each block. We use $T_{2^{2(n-1)}}$ and $\overline{T}_{2^{2(n-1)})}$ to denote transposition and anti-diagonal transposition operation of $2^{n-1} \times 2^{n-1}$ blocks, respectively. Note that $T_{2^{2(n-1)}}$ is $L_{2^{n-1}}^{2^{2(n-1)}}$ and $\overline{T}_{2^{2(n-1)})}$ is $(J_{2^{n-1}} \otimes J_{2^{n-1}}) L_{2^{n-1}}^{2^{2(n-1)}}$. Finally, the four blocks are recursively applied the $2^{n-1} \times 2^{n-1}$ Hilbert space-filling curve permutation. The recursive tensor product formula of the Hilbert space-filling curve permutation can be expanded repeatedly to derive the iterative tensor product formula as:

$$\begin{array}{rcl} H_n & = & \prod_{i=0}^{n-1} I_{4^i} \otimes \\ & & [(T_{2^{2(n-i-1)}} \oplus I_{2^{2(n-i-1)}} \oplus I_{2^{2(n-i-1)}} \oplus \overline{T}_{2^{2(n-i-1)}}) \\ & & (G_2 \otimes I_{2^{2(n-i-1)}})(I_2 \otimes L_2^{2^{n-i}} \otimes I_{2^{n-i-1}})]. \end{array}$$

Both recursive and iteration tensor product formulas for Hilbert space-filling curves converts the entire space points. Since some applications need not process all points of a multi-dimensional space, we will modify the iterative tensor product formula for Hilbert space-filling curves to derive a closed-form algorithm for converting Hilbert space-filling curve indices of individual points.

A selection operation is needed for the tensor product formulation because the closed-form algorithm deals with a single point only. The iterative two-dimensional closed-form tensor product formula for converting a Hilbert space-filling curve point can be deduced from the two-

dimensional iterative tensor product formula as:

$$
\begin{aligned}
& H_n E_{k,k}^{4^n,4^n} \\
&= \left\{ \prod_{i=0}^{n-1} I_{4^i} \otimes \left[\begin{array}{l} (T_{2^{2(n-i-1)}} \oplus I_{2^{2(n-i-1)}} \oplus \\ I_{2^{2(n-i-1)}} \oplus \overline{T}_{2^{2(n-i-1)}}) \\ (G_2 \otimes I_{2^{2(n-i-1)}}) \\ (I_2 \otimes L_2^{2^{n-i}} \otimes I_{2^{n-i-1}}) \end{array} \right] \right\} \left(E_{k,k}^{4^n,4^n} \right) \\
&= \left\{ \prod_{i=0}^{n-1} I_{4^i} \otimes \left[\begin{array}{l} (T_{2^{2(n-i-1)}} \oplus I_{2^{2(n-i-1)}} \oplus \\ I_{2^{2(n-i-1)}} \oplus \overline{T}_{2^{2(n-i-1)}}) \\ (G_2 \otimes I_{2^{2(n-i-1)}}) \\ (I_2 \otimes L_2^{2^{n-i}} \otimes I_{2^{n-i-1}}) \end{array} \right] \right\} \\
& \quad \left(E_{r,r}^{2^n,2^n} \otimes E_{c,c}^{2^n,2^n} \right),
\end{aligned}
$$

where operation $E_{k,k}^{4^n,4^n}$ is to select one of the points of the $2^n \times 2^n$ gird and the remainder of the tensor product formula is the operations of iterative closed-form algorithm applying to the point. Initially, we assume the points (data) on the two-dimensional $2^n \times 2^n$ grid are stored in the row-major order so the point of index (r, c), $0 \le r, c < 2^n$, is represented as the tensor basis $e_r^{2^n} \otimes e_c^{2^n}$.

The iterative closed-form algorithm can be explained as the following construction steps:

1. **Selection operation:** $E_{k,k}^{4^n,4^n}$
   The selection operation $E_{k,k}^{4^n,4^n}$ can be factorized to $E_{r,r}^{2^n,2^n} \otimes E_{c,c}^{2^n,2^n}$, where $k = r \times 2^n + c$. Let indices r and c be expressed in their binary representations $r_0 r_1 \cdots r_{n-2} r_{n-1}$ and $c_0 c_1 \cdots c_{n-2} c_{n-1}$, respectively. Applying $E_{k,k}^{4^n,4^n}$ to the initial row-major order tensor basis, we obtain the following basis:

$$
\begin{aligned}
& (E_{k,k}^{4^n,4^n})\,[1 \cdots 1]_{4^n \times 1}^T \\
= \; & (E_{r,r}^{2^n,2^n} \otimes E_{c,c}^{2^n,2^n})([1 \cdots 1]_{2^n \times 1}^T \otimes [1 \cdots 1]_{2^n \times 1}^T) \\
= \; & e_r^{2^n} \otimes e_c^{2^n} \\
= \; & e_{r_0}^2 \otimes e_{r_1}^2 \otimes \cdots \otimes r_{r_{n-2}}^2 \otimes r_{r_{n-1}}^2 \otimes \\
& e_{c_0}^2 \otimes e_{c_1}^2 \otimes \cdots \otimes r_{c_{n-2}}^2 \otimes r_{c_{n-1}}^2.
\end{aligned}
$$

2. **Iterative operation:**
   $\prod_{i=0}^{n-1} I_{4^i} \otimes \left[\begin{array}{l} (T_{2^{2(n-i-1)}} \oplus I_{2^{2(n-i-1)}} \oplus \\ I_{2^{2(n-i-1)}} \oplus \overline{T}_{2^{2(n-i-1)}}) \\ (G_2 \otimes I_{2^{2(n-i-1)}})(I_2 \otimes L_2^{2^{n-i}} \otimes I_{2^{n-i-1}}) \end{array} \right]$
   The iterative operation $\prod_{i=0}^{n-1}$ has the loop index i step from 0 to $n-1$. The tensor basis at the beginning of the i-th iteration is $e_{h_i}^{4^i} \otimes e_{r_i}^2 \otimes \cdots \otimes r_{r_{n-2}}^2 \otimes r_{r_{n-1}}^2 \otimes e_{c_i}^2 \otimes \cdots \otimes r_{c_{n-2}}^2 \otimes r_{c_{n-1}}^2$, where $h_0 = 0$ and $h_i = \sum_{t=0}^{i-1} (2r'_t + c'_t) \times 4^{i-t-1}$, for $i > 0$. Also, r'_t and c'_t, for $0 \le t < i$, are the results obtained from the previous iterations. If we do not consider the selection operation, the iterative operation is applied to the entire $2^n \times 2^n$ grid. At the beginning of the i-th iterations, the points of the grid are rearranged into 4^i blocks such that the blocks are stored in the order of $2^i \times 2^i$ Hilbert space-filling curve and points in each block are stored in the row-major order with a sequences of Gray permutation and coordinate transformation determined by the block index. Note that I_{4^i} in the iteration body can be distributed to the terms on the left hand side of the tensor product operator, *i.e.*,

$$
\begin{aligned}
& I_{4^i} \otimes \\
& [(T_{2^{2(n-i-1)}} \oplus I_{2^{2(n-i-1)}} \oplus I_{2^{2(n-i-1)}} \oplus \overline{T}_{2^{2(n-i-1)}}) \\
& (G_2 \otimes I_{2^{2(n-i-1)}})(I_2 \otimes L_2^{2^{n-i}} \otimes I_{2^{n-i-1}})] \\
= \; & [I_{4^i} \otimes (T_{2^{2(n-i-1)}} \oplus I_{2^{2(n-i-1)}} \oplus I_{2^{2(n-i-1)}} \oplus \\
& \overline{T}_{2^{2(n-i-1)}})][I_{4^i} \otimes (G_2 \otimes I_{2^{2(n-i-1)}})] \\
& [I_{4^i} \otimes (I_2 \otimes L_2^{2^{n-i}} \otimes I_{2^{n-i-1}})]
\end{aligned}
$$

We will explain block allocation, Gray permutation, and coordinate transformation below.

(a) **Block allocation:** $I_{4^i} \otimes (I_2 \otimes L_2^{2^{n-i}} \otimes I_{2^{n-i-1}})$.
Block allocation rearranges a block of size $2^{n-i} \times 2^{n-i}$ to four blocks of size $2^{n-i-1} \times 2^{n-i-1}$ each. Applying the block allocation to the tensor basis, we obtain the following basis:

$$
\begin{aligned}
& [I_{4^i} \otimes (I_2 \otimes L_2^{2^{n-i}} \otimes I_{2^{n-i-1}})] \\
& (e_{h_i}^{4^i} \otimes e_{r_i}^2 \otimes e_{r_{i+1}}^2 \otimes \cdots \otimes e_{r_{n-1}}^2 \otimes \\
& e_{c_i}^2 \otimes e_{c_{i+1}}^2 \otimes \cdots \otimes e_{c_{n-1}}^2) \\
= \; & [I_{4^i} e_{h_i}^{4^i}] \otimes [(I_2 \otimes L_2^{2^{n-i}} \otimes I_{2^{n-i-1}}) \\
& (e_{r_i}^2 \otimes e_{r_{i+1}}^2 \otimes \cdots \otimes e_{r_{n-1}}^2 \otimes \\
& e_{c_i}^2 \otimes e_{c_{i+1}}^2 \otimes \cdots \otimes e_{c_{n-1}}^2)] \\
= \; & e_{h_i}^{4^i} \otimes e_{r_i}^2 \otimes e_{c_i}^2 \otimes e_{r_{i+1}}^2 \otimes \cdots \otimes e_{r_{n-1}}^2 \otimes \\
& e_{c_{i+1}}^2 \otimes \cdots \otimes e_{c_{n-1}}^2.
\end{aligned}
$$

(b) **Gray permutation:** $I_{4^i} \otimes (G_2 \otimes I_{2^{2(n-i-1)}})$.
The result of block allocation is 4^{i+1} blocks of size $2^{n-i-1} \times 2^{n-i-1}$ each. Given (r', c'), the four blocks indexed by (r_i, c_i), where $0 \le r_i, c_i \le 1$, are permuted with 2-bit Gray permutation $G(2)$. Applying the gray permutation to the resulting tensor basis of block allocation, we obtain:

$$
\begin{aligned}
& [I_{4^i} \otimes (G_2 \otimes I_{2^{2(n-i-1)}})] \\
& (e_{h_i}^{4^i} \otimes e_{r_i}^2 \otimes e_{c_i}^2 \otimes e_{r_{i+1}}^2 \otimes \cdots \otimes e_{r_{n-1}}^2 \otimes \\
& e_{c_{i+1}}^2 \otimes \cdots \otimes e_{c_{n-1}}^2) \\
= \; & e_{h_i}^{4^i} \otimes e_{r'_i}^2 \otimes e_{c'_i}^2 \otimes e_{r_{i+1}}^2 \otimes \cdots \otimes e_{r_{n-1}}^2 \otimes \\
& e_{c_{i+1}}^2 \otimes \cdots \otimes e_{c_{n-1}}^2.
\end{aligned}
$$

The mapping of Gray permutation G_2 is depicted as: $(0,0) \to (0,0)$, $(0,1) \to (0,1)$, $(1,0) \to (1,1)$, and $(1,1) \to (1,0)$.

(c) **Coordinate transformation:**
$I_{4^i} \otimes (T_{2^{2(n-i-1)}} \oplus I_{2^{2(n-i-1)}} \oplus I_{2^{2(n-i-1)}} \oplus \overline{T}_{2^{2(n-i-1)}})$.
The four blocks obtained from Gray permutation are not aligned well because the Hilbert

space-filling curve in each of the blocks are not connected "adjacently". Hence, we need to adjust the orientation of blocks so the ending point of a block and the starting point of its following block are neighboring to each other. Let $R_i = \prod_{k=i+1}^{n-1} r_k$ and $C_i = \prod_{k=i+1}^{n-1} c_k$. The orientation adjustment for the block with index (r'_i, c'_i) is expressed as the following coordinate transformations:

i. for $(r'_i, c'_i) = (0,0)$, $(R_i, C_i) \rightarrow (C_i, R_i)$, *i.e.,* $2^{n-i-1} \times 2^{n-i-1}$ transposition operation denoted as $T_{2^{2(n-i-1)}}$,

ii. for $(r'_i, c'_i) = (0,1)$ and $(r'_i, c'_i) = (1,0)$, $(R_i, C_i) \rightarrow (R_i, C_i)$, *i.e.,* $2^{n-i-1} \times 2^{n-i-1}$ identity operation denoted as $I_{2^{2(n-i-1)}}$, and

iii. for $(r'_i, c'_i) = (1,1)$, $(R_i, C_i) \rightarrow (-C_i, -R_i)$, *i.e.,* $2^{n-i-1} \times 2^{n-i-1}$ anti-diagonal transposition operation denoted as $\overline{T}_{2^{2(n-i-1)}}$,

The coordinate transformation is expressed as $T_{2^{2(n-i-1)}} \oplus I_{2^{2(n-i-1)}} \oplus I_{2^{2(n-i-1)}} \oplus \overline{T}_{2^{2(n-i-1)}}$. Recall that $T_{2^{2(n-i-1)}}$ is $L^{2^{2(n-i-1)}}_{2^{n-i-1}}$ and $\overline{T}_{2^{2(n-i-1)}}$ is $(J_{2^{n-i-1}} \otimes J_{2^{n-i-1}})L^{2^{2(n-i-1)}}_{2^{n-i-1}}$. The effect of $T_{2^{2(n-i-1)}}$ and $\overline{T}_{2^{2(n-i-1)}}$ on tensor basis $e^2_{r_{i+1}} \otimes \cdots \otimes e^2_{r_{n-1}} \otimes e^2_{c_{i+1}} \otimes \cdots \otimes e^2_{c_{n-1}}$ is as below:

$$
\begin{aligned}
& T_{2^{2(n-i-1)}}(e^2_{r_{i+1}} \otimes \cdots \otimes e^2_{r_{n-1}} \otimes \\
& e^2_{c_{i+1}} \otimes \cdots \otimes e^2_{c_{n-1}}) \\
= \; & L^{2^{2(n-i-1)}}_{2^{n-i-1}}(e^2_{r_{i+1}} \otimes \cdots \otimes e^2_{r_{n-1}} \otimes \\
& e^2_{c_{i+1}} \otimes \cdots \otimes e^2_{c_{n-1}}) \\
= \; & e^2_{c_{i+1}} \otimes \cdots \otimes e^2_{c_{n-1}} \otimes e^2_{r_{i+1}} \otimes \cdots \otimes e^2_{r_{n-1}} \\
= \; & e^{2^{n-i-1}}_{C_i} \otimes e^{2^{n-i-1}}_{R_i},
\end{aligned}
$$

$$
\begin{aligned}
& \overline{T}_{2^{2(n-i-1)}}(e^2_{r_{i+1}} \otimes \cdots \otimes e^2_{r_{n-1}} \otimes \\
& e^2_{c_{i+1}} \otimes \cdots \otimes e^2_{c_{n-1}}) \\
= \; & (J_{2^{n-i-1}} \otimes J_{2^{n-i-1}}) \\
& L^{2^{2(n-i-1)}}_{2^{n-i-1}}(e^2_{r_{i+1}} \otimes \cdots \otimes e^2_{r_{n-1}} \otimes \\
& e^2_{c_{i+1}} \otimes \cdots \otimes e^2_{c_{n-1}}) \\
= \; & (J_{2^{n-i-1}} \otimes J_{2^{n-i-1}})(e^2_{c_{i+1}} \otimes \cdots \otimes e^2_{c_{n-1}} \otimes \\
& e^2_{r_{i+1}} \otimes \cdots \otimes e^2_{r_{n-1}}) \\
= \; & e^2_{c_{i+1}} \otimes \cdots \otimes e^2_{c_{n-1}} \otimes e^2_{r_{i+1}} \otimes \cdots \otimes e^2_{r_{n-1}} \\
= \; & e^{2^{n-i-1}}_{2^{n-i-1}-C_i-1} \otimes e^{2^{n-i-1}}_{2^{n-i-1}-R_i-1}.
\end{aligned}
$$

At the end of coordinate transformation, tensor basis $e^{4^i}_{h_i} \otimes e^{2^i}_{r'_i} \otimes e^{2^i}_{c'_i}$ is exactly $e^{4^{i+1}}_{h_{i+1}}$, *i.e.,* $h_{i+1} = 4h_i + 2r'_i + c'_i$, for the next iteration.

The closed-form algorithm maps the row-major index of a point to its corresponding Hilbert space-filling curve index. We will explain program generation of the closed-form algorithm in the following section.

4 Closed-Form Program Generation

We use C programming language to illustrate the generated program. Function `closedFormHilbert2D()` has three unsigned integer parameters `n`, `r`, and `c`, where `n` is the length of the binary representation of row index `r` and column index `c`. We assume the maximum length of the row/column index is less than or equal to half of the length of an unsigned integer which is 16 in most computers. Function `closedFormHilbert2D()` will return an unsigned integer which is the corresponding Hilbert space-filling curve index of (`r`, `c`).

We will use bit-wise operations to convert the row-major index (`r`, `c`) to the Hilbert space-filling curve index `h`. Program generation of `closedFormHilbert2D()` follows the steps of tensor product formulation explained in the previous section.

1. **Selection operation:**
   The tensor basis $e^2_{r_0} \otimes e^2_{r_1} \otimes \cdots \otimes r^2_{r_{n-2}} \otimes r^2_{r_{n-1}} \otimes e^2_{c_0} \otimes e^2_{c_1} \otimes \cdots \otimes r^2_{c_{n-2}} \otimes r^2_{c_{n-1}}$ is encoded in parameters `r` and `c` of `n` bits long. The least significant `n` bits of `r` and `c` correspond to the binary representation $r_0 r_1 \cdots r_{n-2} r_{n-1}$ and $c_0 c_1 \cdots c_{n-2} c_{n-1}$, respectively. The resulting Hilbert space-filling curve index is stored in a local variable `h` which is initialized to 0 and its result is of length $2n$ bits. At beginning of an iteration the partial result of Hilbert space-filling curve index h_i is recorded in `h`. We use `pos` to record the position of bits r_i and c_i in an iteration. Variable `mask` is used later in the anti-diagonal transposition operation and it is initialized to all 1's at the $n-1$ least significant bits and 0's elsewhere. Variable declaration and initialization of `h`, `pos`, and `mask` are shown below:

   ```
   unsigned h = 0;
   unsigned pos = 1 << (n-1);
   unsigned mask = pow(2, n-1)-1;
   ```

2. **Iterative operation:**
   The iterative operation $\prod_{i=0}^{n-1}$ is a matrix product which is simply translated to the following a `for` loop:

   ```
   for (i=0; i<n; i++) { };
   ```

   with the loop body performing block allocation, Gray permutation, and coordinate transformation. These three operations manipulate tensor basis $e^2_{r_i} \otimes e^2_{r_{i+1}} \otimes \cdots \otimes e^2_{r_{n-1}} \otimes e^2_{c_i} \otimes e^2_{c_{i+1}} \otimes \cdots \otimes e^2_{c_{n-1}}$ and the partial result of Hilbert space-filling curve index h_i is stored in `h`.

   (a) **Block allocation:**
   Block allocation maps $e^2_{r_i} \otimes e^2_{r_{i+1}} \otimes \cdots \otimes e^2_{r_{n-1}} \otimes$

$e^2_{c_i} \otimes e^2_{c_{i+1}} \otimes \cdots \otimes e^2_{c_{n-1}}$ to $e^2_{r_i} \otimes e^2_{c_i} \otimes e^2_{r_{i+1}} \otimes \cdots \otimes e^2_{r_{n-1}} \otimes e^2_{c_{i+1}} \otimes \cdots \otimes e^2_{c_{n-1}}$. We extract r_i and c_i and store them in variables `ri` and `ci`, respectively:

```
ri = (r & pos) >> (n-i-1);
ci = (c & pos) >> (n-i-1);
```

Also, the rest of tensor basis $e^2_{r_{i+1}} \otimes \cdots \otimes e^2_{r_{n-1}} \otimes e^2_{c_{i+1}} \otimes \cdots \otimes e^2_{c_{n-1}}$ is kept in `r` and `c` with the following two assignments:

```
r = r & mask; c = c & mask;
```

(b) **Gray permutation:**
The Gray permutation is simply transformed to a conditional statement of four cases:

```
if (ri==0 && ci==0)
     {ri = 0; ci = 0;} // Case 1
else if (ri==0 && ci==1)
     {ri = 0; ci = 1;} // Case 2
else if (ri==1 && ci==0)
     {ri = 1; ci = 1;} // Case 3
else {ri = 1; ci = 0;} // Case 4
```

Note that the assignments in the first two cases are not necessary, we leave them there to show the mapping of the Gray permutation.

(c) **Coordinate transformation:**
Coordinate transformation is coded within the four cases of the Gray permutation. For Case 2 and Case 4, *i.e.*, $(r'_i, c'_i) = (0, 1)$ and $(1, 0)$, coordinate transformation is the identity mapping, $I_{2^{2(n-i-1)}}$, so we will do nothing at all. For Case 1, *i.e.*, $(r'_i, c'_i) = (0, 0)$, the transposition operation, $T_{2^{2(n-i-1)}}$, is implemented by swapping `r` and `c`:

```
tmp = r; r = c; c = tmp;
```

For Case 3, the anit-diagonal transposition operation, $\overline{T}_{2^{2(n-i-1)}}$, can be implemented by swapping `r` and `c` followed by complement operations: is implemented by swapping `r` and `c`:

```
tmp = r; r = c; c = tmp;
r = ~r & mask; c = ~c & mask;
```

At the end of iteration `i`, `pos` and `mask` are manipulated. Also, `ri` and `ci` are appended to the partial result `h` as below:

```
pos =  pos >> 1; mask = mask >> 1;
h = (((h << 1) | ri) << 1) | ci;
```

When the loop terminates, `h` is the Hilbert space-fill curve index mapped by the input row-major index `(r, c)`. Finally, `h` is returned by function `closedFormHilbert2D()`.

We summarized the generated program as the followings:

```
unsigned closedFormHilbert2D(unsigned n,
               unsigned r, unsigned c) {

  unsigned h = 0;
  unsigned pos = 1 << (n-1);
  unsigned mask = pow(2, n-1)-1;
  unsigned i, tmp;

  for (i=0; i<n; i++) {
    ri = (r & pos) >> (n-i-1);
    ci = (c & pos) >> (n-i-1);
    r = r & mask; c = c & mask;
    if (ri==0 && ci==0) {      // Case 1
      ri = 0; ci = 0;
      tmp = r; r = c; c = tmp; }
    else if (ri==0 && ci==1) { // Case 2
      ri = 0; ci = 1; }
    else if (ri==1 && ci==0) { // Case 3
      ri = 1; ci = 1;
      tmp = r; r = c; c = tmp;
      r = ~r & mask; c = ~c & mask; }
    else {                     // Case 4
      ri = 1; ci = 0; }
    pos =  pos >> 1; mask = mask >> 1;
    h = (((h << 1) | ri) << 1) | ci;
  }
  return h;
}
```

In some application problems, the mapping of a Hilbert space-filling curve index to its corresponding row-major or column-major index is needed. We simply show the tensor product of formula of the inverse closed-form algorithm:

$$\begin{aligned} H_n^{-1} \quad = \quad & \prod_{i=n-1}^{0} I_{4^i} \otimes \\ & [(I_2 \otimes L^{2^{n-i}}_{2^{n-i-1}} \otimes I_{2^{n-i-1}}) \\ & (T_{2^{2(n-i-1)}} \oplus I_{2^{2(n-i-1)}} \oplus I_{2^{2(n-i-1)}} \oplus \overline{T}_{2^{2(n-i-1)}}) \\ & (G_2 \otimes I_{2^{2(n-i-1)}})]. \end{aligned}$$

The inverse closed-form algorithm is applied to tensor basis $E^{4^n,4^n}_{h,h}$ corresponding to a Hilbert space-filling curve index h. Program generation of the inverse closed-form algorithm is omitted in this paper.

5 Performance Analysis

The time and space complexities of the two-dimensional Hilbert space-filling curve algorithm are $O(n4^n)$ and $O(4^n)$, respectively, because it scans the entire $2^n \times 2^n$ data space and performs data movement [24]. However, the closed-form algorithm only converts the index of a selected point. It is improper to compare time and space complexities of these two algorithms.

The time complexity of the closed-form algorithm is $O(n)$, *i.e.* the bit length of r and c, if we count all the three operations: block allocation, Gray permutation, and coordinate transformation, in each iteration.

However, block allocation can be interpreted as bit extraction of Hilbert space-filling curve index. The closed-form algorithm may be modified to delayed the block allocation so the tensor basis at the beginning of the i-th iteration is $e^2_{r'_0} \otimes \cdots \otimes e^2_{r'_{i-1}} \otimes e^2_{r_i} \otimes \cdots \otimes e^2_{r_{n-1}} \otimes e^2_{c'_0} \otimes \cdots \otimes e^2_{c'_{i-1}} \otimes e^2_{c_i} \otimes \cdots \otimes e^2_{c_{n-1}}$. The Hilbert space-filling curve index is synthesized upon termination of the loop by interleaving $e^2_{r'_0} \otimes \cdots \otimes e^2_{r'_{i-1}} \otimes e^2_{r'_i} \otimes \cdots \otimes e^2_{r'_{n-1}}$ and $e^2_{c'_0} \otimes \cdots \otimes e^2_{c'_{i-1}} \otimes e^2_{c'_i} \otimes \cdots \otimes e^2_{c'_{n-1}}$. For the modified closed-form algorithm, the time complexity counts only the number of transposition operation, for $(r'_i, c'_i) = (0, 0)$, and anti-diagonal transposition operation, for $(r'_i, c'_i) = (0, 1)$, *i.e.*, $O(parity(\overline{c}))$. The space complexity of the closed-form algorithm is $O(1)$, bounded by a constant, since only a number of local variables are declared in function `closedFormHilbert2D()`. However, if we consider the bit length of n, the actual space complexity should be $O(n)$.

Chen *et al.* present an algorithm for encoding and decoding a Hilbert space-filling curve index of time complexity $\lceil \log_2 max(r, c) \rceil$ [27]. In fact, the computational sequence of this algorithm is the same as the algorithm in this paper. The complexity discrepancy between these two algorithms is caused by how the operations are counted. We do not only present an algorithm of converting Hilbert space filling-curve indices, but we also show how the algorithm is derived from a tensor product formula. This approach gives a theoretical background for designing variants of space-filling curves.

6 Conclusions

We use a tensor product based algebraic theory to model two-dimensional iterative closed-form tensor product formulation for converting Hilbert space-filling curve indices. The inverse two-dimensional iterative closed-form Hilbert space-filling curve permutation algorithm is derived from the iterative tensor product formula. These formulas can be used to generate iterative and inverse iterative closed-form programs with bit-wise opertions.

Space-filling curves are used in various applications such as image compression, VLSI component layout, and R-tree indexing. For the example of image compression, data collection can be rearranged according to two-dimensional space-filling curve order to enhance locality relationship and improve compression ratio. Furthermore, the inverse space-filling curve permutations can be used in image decompression. The tensor product theory is also suitable for expressing other space-filling curve such as two-dimensional Peano, Moore, and Wunderlich space-filling curves and three-dimensional Hilbert space-filling curve. In the future work, we will develop general-form tensor product formulas of these space-filling curves.

References

[1] G. Peano, "Sur une courbe qui remplit touteune aire plane," *Mathematische Annalen*, vol. 36, pp. 157–160, 1890.

[2] D. Hilbert, "Über die stetige abbildung einer linie auf Flächenstück," *Mathematische Annalen*, vol. 38, pp. 459–460, 1891.

[3] S. Biswas, "Hilbert scan and image compression," in *Proceedings of the 15th International Conference on Pattern Recognition*, vol. 3, 2000, pp. 207–210.

[4] G. Melnikov and A. K. Katsaggelos, "A non-uniform segmentation optimal hybrid fractal/DCT image compression algorithm," in *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 5, 1998, pp. 2573–2576.

[5] N. Memon, D. L. Neuhoff, and S. Shende, "An analysis of some common scanning techniques for lossless image coding," *IEEE Transactions on Image Processing*, vol. 9, pp. 1837–1848, 2000.

[6] B. O'Sullivan, "Applying partial evaluation to VLSI design rule checking," 1995, uRL: citeseer.ist.psu.edu/371669.html.

[7] S. Rovetta and R. Zunino, "VLSI circuits with fractal layout for spatial image decorrelation," in *Proceedings of the 1999 IEEE International Symposium on Circuits and Systems*, vol. 4, 1999, pp. 110–113.

[8] D. M. Gavrila, "R-tree index optimization," in *Proceedings of the Sixth International Symposium on Spatial Data Handling*, vol. 2, 1994, pp. 771–791.

[9] I. Kamel and C. Faloutsos, "On packing R-trees," in *Proceedings of the Second International Conference on Information and Knowledge Management*, 1993, pp. 490–499.

[10] I. Kamel and C. Faloutsos, "Hilbert R-tree: An improved R-tree using fractals," in *Proceedings of the Twentieth International Conference on Very Large Databases*, 1994, pp. 500–509.

[11] S. T. Leutenegger, J. M. Edgington, and M. A. Lopez, "STR: A simple and efficient algorithm for R-tree packing," in *Proceedings of the Thirteenth International Conference on Data Engineering*, 1997, pp. 497–506.

[12] S. T. Leutenegger and M. A. Lopez, "The effect of buffering on the performance of R-trees," *Knowledge and Data Engineering*, vol. 12, no. 1, pp. 33–44, 2000.

[13] J. Gonzalez-Arbesu, S. Blanch, and J. Romeu, "Are space-filling curves efficient small antennas?" *IEEE Antennas and Wireless Propagation Letters*, vol. 2, pp. 147 –150, 2003.

[14] D. Kontos, V. Megalooikonomou, N. Ghubade, and C. Faloutsos, "Detecting discriminative functional MRI activation patterns using space filling curves," in *Proceedings of the 25th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, 2003, pp. 963–967.

[15] H. V. Jagadish, "Linear clustering of objects with multiple attributes," in *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, 1990, pp. 332–342.

[16] A. R. Butz, "Space filling curves and mathematical programming," *Information and Control*, vol. 12, no. 4, pp. 314–330, 1968.

[17] A. R. Butz, "Convergence with Hilbert's Space Filling Curve," *Journal of Computer and System Sciences*, vol. 3, no. 2, pp. 128–146, 1969.

[18] H. Sagan, "On the geometrization of the Peano curve and the arithmetization of the Hilbert curve," *International Journal of Mathematical Education in Science and Technology*, vol. 23, no. 3, pp. 403–411, 1992.

[19] Y. Ohno and K. Ohyama, "A catalog of symmetric self-similar space-filling curves," *Journal of Recreational Mathematics*, vol. 23, pp. 161–173, 1991.

[20] J. Quinqueton and M. Berthod, "A locally adaptive Peano scanning algorithm," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 3, no. 4, pp. 403–412, 1981.

[21] S. Kamata, R. O. Eason, and Y. Bandou, "A new algorithm for N-dimensional Hilbert scanning," *IEEE Transactions on Image Processing*, vol. 8, no. 7, pp. 964–973, 1999.

[22] S. Kamata, M. Niimi, R. O. Eason, and E. Kawaguchi, "An implementation of an N-dimensional Hilbert scanning algorithm," in *Proceedings of the 9th Scandlnavian Conference on Image Analysis*, 1995, pp. 431–440.

[23] C.-S. Chen, S.-Y. Lin, and C.-H. Huang, "Algebraic formulation and program generation of three-dimensional Hilbert space-filling curves," in *Proceedings of the 2004 International Conference on Imaging Science, Systems, and Technology*, 2004, pp. 254–260.

[24] S.-Y. Lin, C.-S. Chen, L. Liu, and C.-H. Huang, "Tensor product formulation for Hilbert space-filling curves," *Journal of Information Science and Engineering*, vol. 24, no. 1, pp. 261–275, 2008.

[25] X. Liu and G. F. Schrack, "An algorithm for encoding and decoding the 3-D Hilbert order," *IEEE Transactions on Image Processing*, vol. 6, no. 9, pp. 1333–1337, 1997.

[26] A. J. Fisher, "A new algorithm for generating Hilbert curves," *Software: Practice and Experience*, vol. 16, no. 1, pp. 5–12, 1986.

[27] N. Chen, N. Wang, and B. Shi, "A new algorithm for encoding and decoding the Hilbert order," *Software: Practice and Experience*, vol. 37, no. 8, pp. 897–908, 2007.