Space-Filling Approach for Fast Window Query on Compressed Images

Kuo-Liang Chung, Yao-Hong Tsai, and Fei-Ching Hu

***Abstract*—Based on the space-filling approach, this paper presents a fast algorithm for window query on compressed images. Given a query window of size $n_1 \times n_2$, the proposed algorithm takes $O(n_l \log T + P)$ time to perform the window query, where $n_l = \max(n_1, n_2)$ and $T \times T$ is the image size; P is the number of outputted codes. The proposed algorithm improves the naive algorithm, which needs $O(n_1 n_2 \log T + P)$ time, significantly. Some experimentations are carried out to demonstrate the computational advantage of the proposed algorithm. From the experimental results, it is observed that the proposed algorithm has about 72–98% time improvement when compared to the naive algorithm.**

***Index Terms*—Compressed images, Hilbert scan, image database, maximal quadtree blocks, space-filling curve, window query.**

I. Introduction

THE ORIGINAL problem of space-filling curves was found by Cantor [7] that the interval $[0, 1]$, can be mapped bijectively onto the square $[0, 1]^2$. Peano [24] settled this problem by constructing a curve that passes through every entry of a two-dimensional region. Curves with this property are called space-filling curves or Peano curves. Afterwards, many variants of space-filling curves were invented by Hilbert [12], Moore [21], Lebesgue [18], and so on. The detailed descriptions are referred to see the book [27]. Applications of space-filling curves are studied in the area of image analysis , [15], [28], image compression [16], [17], [25], image encryption [6], [8], vector median filtering [26], ordered dither [30], database access analysis [4], [13], bandwidth reduction [5], and so on [27].

Hilbert was the first who made this phenomenon of space-filling curves luminous to the geometric imagination. Thus, among these space-filling curves, Hilbert curve [12] is the most well-known. One important feature of Hilbert curve is that it scans the neighboring entry in the image continuously. Then, the scanning order of Hilbert curve, named Hilbert scan, preserves the proximity property of the original image. Liu and Schrack [19], [20] presented an efficient algorithm to encode and decode the Hilbert order from two-dimensional (2-D) and three-dimensional (3-D) region into a one-dimensional (1-D) curve containing the image of the corresponding region and vice versa. Recently, some image compression methods based on Hilbert scan were developed [14], [16], [17]. For convenience, a gray image compressed by using Hilbert scan is simply called compressed image throughout this paper. Window query [2], [22], [23] is an important query operations in image databases. Given a compressed image, in this research, the window query wants to report the corresponding compressed subimage without the need to decompress the compressed image. After all, decompressing takes a lot of time and a window query does not need refer to the full image.

Employing the maximal blocks partition strategy [3], [29], some properties of Hilbert order, and the related fast mapping formula [19], this paper presents a fast algorithm for window query on compressed images. Given a query window of size $n_1 \times n_2$, the proposed algorithm takes $O(n_l \log T + P)$ time to perform the window query, where $n_l = \max(n_1, n_2)$ and $T \times T$ is the image size; P is the number of outputted codes. The proposed algorithm improves the naive algorithm, which needs $O(n_1 n_2 \log T + P)$ time. Some experimentations are carried out to demonstrate the computational advantage of the proposed algorithm. From the experimental results, it is observed that the proposed algorithm has about 72% to 98% time improvement when compared to the naive algorithm.

The rest of the paper is organized as follows. In Section II, the definition of the Hilbert curve and the mapping (encoding) formula from the x- and y-coordinates to the Hilbert order are described. In Section III, we describe the method for compressing gray images based on the Hilbert scan. Section IV presents the algorithm for generating maximal blocks for the given window. Section V presents the proposed algorithm for window query on compressed images. Some experimental results are demonstrated in Section VI. Finally, Section VII addresses some conclusions.

II. Hilbert Curve and the Encoding Formula

Consider a $T \times T$ gray image represented as an array in the domain $\{x, y | 0 \le x \le T-1, 0 \le y \le T-1\}$, where x and y are integers. A Hilbert curve [12] can be obtained by using a recursive procedure which passes through all the entries in the given image space. For example, the Hilbert curves of resolution $r = 1, 2$, and 3 are shown in Fig. 1, where $2^r = T$. Each pixel in the given 2-D image space is usually needed to be mapped into one point in the 1-D space Hilbert curve. Each position on the curve is denoted by an integer, say the Hilbert order h and the Hilbert orders along the Hilbert curve form a strictly increasing sequence $\langle 0, 1, 2, \cdots, T^2-1 \rangle$. For an image space, the original point of the x- and y-coordinates is defined at the lower-left corner of the image space. For example, the

Manuscript received November 13, 1998; revised June 13, 2000. This work was supported by NSC88-2213-E011-005/006. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Tsuhan Chen.

The authors are with the Department of Information Management, Institute of Computer Science and Information Engineering, National Taiwan University of Science and Technology, Taipei 10672, Taiwan, R.O.C. (e-mail: klchung@cs.ntust.edu.tw).

Publisher Item Identifier S 1057-7149(00)10071-5.

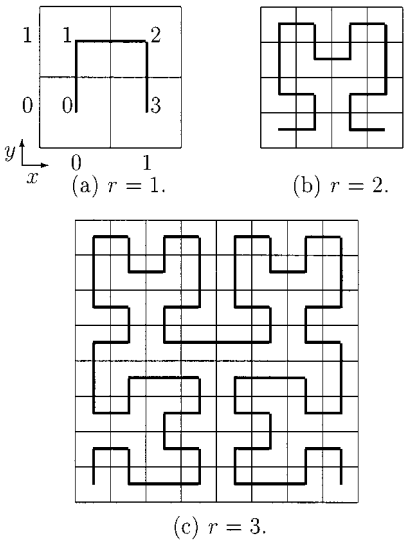


Fig. 1. Hilbert curves for resolution $r = 1, 2$, and 3.

corresponding Hilbert orders of the entries (0, 1) and (1, 1) in Fig. 1(a) are 1 and 2, respectively.

Using the encoding formula developed by Liu and Schrack [19], each entry (x, y) in the image space can be transformed into the corresponding Hilbert order h efficiently. We have assumed that the image size is of $T \times T$. Let $2^r = T$ and let x and y be represented by $x = (x_{r-1} \cdots x_1 x_0)_2$ and $y = (y_{r-1} \cdots y_1 y_0)_2$, i.e., $x = \sum_0^{r-1} 2^i x_i$ and $y = \sum_0^{r-1} 2^i y_i$, where $x_i, y_i \in \{0, 1\}$. The Hilbert order can be represented by a quaternary digit string $h = (h_{r-1} \cdots h_1 h_0)_4$, i.e. $h = \sum_0^{r-1} 4^i h_i$, where $h_i \in \{0, 1, 2, 3\}$. Let the two bits of a quaternary digit h_k in h be represented by h^*_{2k+1} and h^*_{2k}. The encoding formula is listed as follows:

$$h^*_{2k+1} = \overline{v}_{0,k}(v_{1,k} \oplus x_k) + v_{0,k}(v_{1,k} \oplus \overline{y}_k) \quad (1)$$

$$h^*_{2k} = x_k \oplus y_k \quad (2)$$

where $k = 0, 1, \cdots, r-1$ and $v_{0,k}$ and $v_{1,k}$ can be derived by the following iterative formula:

$$v_{1,r-1} = 0 \quad (3)$$

$$v_{0,r-1} = 0 \quad (4)$$

$$v_{1,j-1} = v_{1,j}(x_j \oplus y_j) + (\overline{x}_j \oplus y_j)(v_{0,j} \oplus \overline{y}_j) \quad (5)$$

$$v_{0,j-1} = v_{0,j}(v_{1,j} \oplus \overline{x}_j) + \overline{v}_{0,j}(v_{1,j} \oplus \overline{y}_j) \quad (6)$$

where $j = r-1, \cdots, 2, 1$. Based on the above formulas, (1)–(6), given the x- and y-coordinates of one pixel in the image space, the Hilbert order can be derived in $O(r) = O(\log T)$ time.

Return to Fig. 1(a). For the pixel at (1, 0), the coordinate values are $x = x_0 = 1$ and $y = y_0 = 0$. Since $r = 1$, we have $v_{1,0} = 0$ and $v_{0,0} = 0$ by (3) and (4). It follows that $h^*_1 = 1(0 \oplus 1) + 0(0 \oplus 1) = 1$ and $h^*_0 = 1 \oplus 0 = 1$ by (1) and (2), respectively. The corresponding Hilbert order is $h = (11)_2 = 3$.

III. Image Compression using Hilbert Scan

If a gray image of size $T \times T$ is scanned along the Hilbert curve, the generated gray levels of the pixels in the image form a 3-D curve, which is called the Hilbert scanned curve of gray levels (HSCG). In most cases, the difference of gray levels between two neighboring pixels in the image space is bounded by some small value. Due to the proximity property, the constructed HSCG can be partitioned into some segments by using the interpolation method.

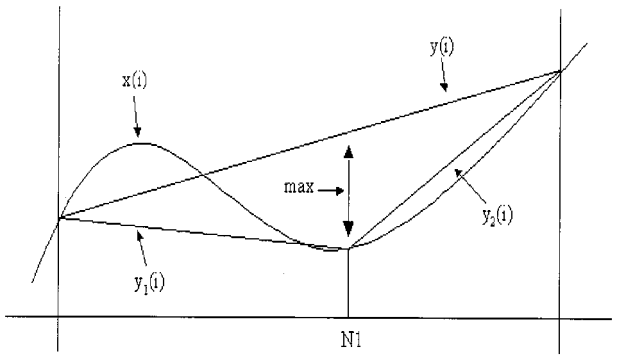
For each segment, the beginning point, ending point, and their gray levels form a code, which is used to represent the segment. In practice, the memory required in one segment is less than that required for representing those gray levels. Thus, if the original gray image is represented by a set of codes, which denote the generated segments, it achieves the compression effect. Kamata *et al.* [17] presented an efficient algorithm for compressing gray images using the Hilbert scan and the zero-order interpolation. Jian [14] presented a method using first-order interpolation, which is based on the technique of polygon approximation [1], [10] on the HSCG. Since the first-order interpolation method has a better fitting effect to the original gray image than that in the zero-order interpolation method, we thus adopt the former method in this paper.

The segmentation process in the first-order interpolation method is first to divide the generated HSCG into some partitions, each partition with length L. Since the total length of HSCG is $T \times T$, there are $N = (T \times T/L)$ partitions. The jth partition of HSCG for $1 \le j \le N$ is denoted by $x_j(i)$, where $i = 1, 2, \cdots, L$. Without loss of generality, we only focus on the jth partition and for simplicity, $x_j(i)$ is represented by $x(i)$.

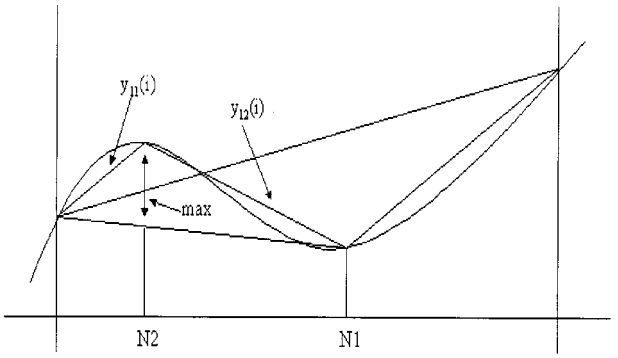
As shown in Fig. 2(a), let $y(i) = x(1) + (x(L) - x(1)/L - 1) \times (i-1)$ be the first-order approximation of $x(i)$. The division point $N1 = \{p | 1 \le p \le L, |x(p) - y(p)| \text{ is maximum}\}$ in the segment is used to divide $y(i)$ into two smaller segments which will approximate $x(i)$ better. The two divided segments are represented by $y_1(i) = x(1) + (x(N1) - x(1)/N1 - 1) \times (i-1)$ and $y_2(i) = x(N1+1) + (x(L) - x(N1+1)/L - N1 - 1) \times (i-1)$. Then, the condition $D \le T^*$ is tested, where $D = \max\{|x(i) - y_j(i)| \text{ for } j = 1 \text{ or } 2\}$ and T^* is the specified threshold. If $D > T^*$, the refinement process is repeated; otherwise it is stopped.

Fig. 2 shows the first two steps of the division (refinement) process of the approximation method. Fig. 2(a) shows the first division process to generate $y_1(i)$ and $y_2(i)$. In Fig. 2(b), $N2$ is the new division point and $y_{11}(i)$ and $y_{12}(i)$ are the new refined segments for representing $x(i)$ at the left of the division point $N1$.

When $D \le T^*$, the approximated polygon lines, i.e., the generated segments, will be quite similar in shape to the original HSCG because each division point is determined to be the point in the original HSCG with the highest curvature. We have known that there are N partitions. Suppose there are N^* segments in one partition, then the compressed output of the corresponding subimage with respect to that partition is a sequence of codes denoted by (B_j, E_j, G_B, G_E) for $1 \le j \le N^*$, where B_j and E_j are the beginning point and ending point of the jth segment; G_B and G_E are the gray levels of B_j and E_j, respectively. In fact, B_j and E_{j-1} (E_j and B_{j+1}) are the same points in the HSCG. Thus, only the beginning point (or ending point) and its gray level are sufficient to represent one code. As a result, the memory required in the outputted codes can be reduced to a half.



(a) The first division.



(b) The second division.

Fig. 2. First two steps of first-order interpolation.

IV. Maximal Quadtree Blocks

To speed up the window query in a quadtree-based image database, a well-known strategy is first to decompose each window into a set of square subwindows according to the quadtree decomposition [2], [22], [23]. These square subwindows are named maximal quadtree blocks, maximal blocks for simplicity. For a query window, the x- and y-coordinates of its lower-left corner is used to denote the window's location, which is called starting point of the window. Thus, a query window is denoted by $w(x, y, n_1, n_2)$, where n_1 is the height and n_2 is the width of the window. For example, the window W=$w(1, 6, 9, 8)$ in Fig. 3 is considered, which is highlighted by four thick lines on its boundary. There are 33 maximal blocks in W and 24 maximal blocks are of width 1; eight maximal blocks are of width 2, e.g., B_1 and B_2; one is of width 4, i.e., B_3. Each maximal block is a square block and is denoted by $MB(x, y, s)$, where (x, y) is the x- and y-coordinates of its lower-left corner and s is the width of the maximal block. Therefore, these 33 maximal blocks are denoted by $MB(1, 6, 1)$, $MB(1, 7, 1)$, $\cdots$, and $MB(4, 8, 4)$=B_3.

Given an arbitrary rectangular window of size $n_1 \times n_2$, Aref and Samet [3] presented an $O(n_l \log\log T)$-time algorithm for decomposing the query window into a set of maximal blocks, where $n_l = \max(n_1, n_2)$ and $T \times T$ is the size of the queried image. In addition, Dyer [11] showed that the number of generated maximal blocks is $O(n_l)$ in the worse case. Recently, an optimal algorithm was presented by Tsai *et al.* [29] for finding the maximal blocks in $O(n_l)$-time. For completeness of this

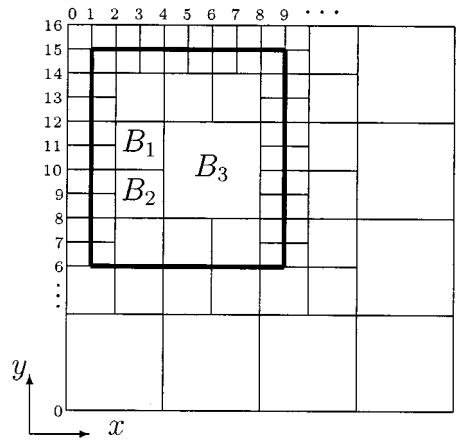


Fig. 3. Query window of size 9×8 on the image of size $2^4 \times 2^4$.

paper, we outline how their algorithm works. The window $W = w(1, 6, 9, 8)$ and the queried image shown in Fig. 3 are used to explain their algorithm.

The technique of their algorithm is repeatedly splitting a strip of maximal blocks from the four sides of the query window. Initially, the blocks of size 1×1 are generated and the variable k is set to 1. Parameters x, y, n_1, and n_2, are the x-coordinate, y-coordinate, height, and width of the window, respectively. Since $(x \bmod 2^k) = (1 \bmod 2) \neq 0$, nine maximal blocks $MB(1, i, 1)$ for $6 \leq i \leq 14$ are generated and the window W is split by moving out a vertical strip of size 9×1 from the left side of the window. The remaining window becomes $w(2, 6, 9, 7)$. Next, since $(y \bmod 2^k = (6 \bmod 2) = 0$, we do nothing. Since $(x + n_2 \bmod 2^k) = (2 + 7 \bmod 2) \neq 0$, nine maximal blocks $MB(2 + 7 - 1, i, 1)$ for $6 \leq i \leq 14$ are generated and the remaining window becomes $w(2, 6, 9, 6)$. Similarly, six maximal blocks $MB(i, 6 + 9 - 1, 1)$ for $2 \leq i \leq 7$ are generated since $(y + n_1 \bmod 2^k = (6 + 9 \bmod 2) \neq 0$. The remaining window is denoted by $w(2, 6, 8, 6)$. After performing the above four steps, k is increased by one, i.e., $k = 2$ and the same steps are repeated until the window becomes null. The other maximal blocks can be obtained by the same arguments.

V. Proposed Algorithm for Window Query on Compressed Images

After describing some related techniques mentioned in Sections II–IV, this section presents a novel approach using the previous techniques as primitives for window query on compressed images.

Suppose the Hilbert orders have been determined in the image space and the given window has been partitioned into a set of maximal blocks. First, the following observation is given.

Observation 1: For each maximal block, the associated Hilbert orders form a strictly increasing sequence.

Since the number of entries in the window is $n_1 \times n_2$, using the naive approach, the window query task can be done by querying each entry in the compressed image. For each entry, it takes $O(\log T)$ query time. For the window, it takes $O(n_1 n_2 \log T)$ query time. Applying the sorting algorithm [9] according to their Hilbert orders, these queried codes are sorted in an increasing sequence and it takes

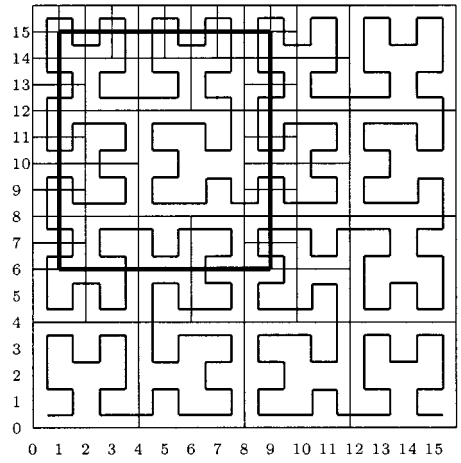


Fig. 4. Hilbert curve map for Fig. 3.

$O(n_1n_2\log(n_1n_2)) = O(n_1n_2\log T)$ since $n_1, n_2 < T$. Then these sorted codes are merged into a smaller set of codes. Totally, the naive algorithm takes $O(n_1n_2\log T + P)$ time to perform the window query, where P is the number of final outputted codes.

Looking Fig. 4, the following observation is very important to improve the naive algorithm.

Observation 2: For each maximal block, suppose the x- and y-coordinates of its lower-left corner has been known and the corresponding Hilbert order has been calculated. If the orientation of the maximal block can be determined, then the minimal Hilbert order and the maximal Hilbert order in the maximal block can be determined.

Following Observation 2, in what follows, we present an efficient method to determine the minimal Hilbert order and the maximal Hilbert order in one maximal block.

Given a maximal block $MB(x, y, s)$, the Hilbert order h_o of the entry (x, y) can be obtained by the formulas from (1)–(6). The orientation v of the Hilbert curve with respect to the beginning point and ending point in the maximal block can be obtained at the same time when h_o is computed. There are totally four types of orientations for the Hilbert curve, say $v = 0$ denoting the direction to the bottom; $v = 1$ denoting the direction to the right; $v = 2$ denoting the direction to the top; $v = 3$ denoting the direction to the left. Let $v = 2v_{1,j-1} + v_{0,j-1}$ be the orientation of a $2^j \times 2^j$ block. For the first type, $v = 0$, the beginning point is at the lower-left corner of the maximal block and the corresponding Hilbert order h_b is also h_o; the ending point is at the lower-right corner of the maximal block and the corresponding Hilbert order h_e is $h_o + (s \times s - 1)$.

Consequently, according to Table I, the minimal Hilbert order h_b and the maximal Hilbert order h_e in one maximal block can be determined using $O(\log T)$ time.

As an example, Fig. 5 is given to demonstrate how Table I works. Consider the maximal block $MB(4, 8, 4)$. For the entry $(4, 8)$, the coordinate values are $x = (0100)_2$ and $y = (1000)_2$. By (3) and (4), $v_{1,3} = 0$ and $v_{0,3} = 0$. Then, $v_{1,2} = 0(0 \oplus 1) + (1 \oplus 1)(0 \oplus 0) = 0$ and $v_{0,2} = 0(0 \oplus 1) + 1(0 \oplus 0) = 0$ by (5) and (6). Similarly, we have $v_{1,1} = 0$, $v_{0,1} = 1$, $v_{1,0} = 0$, and $v_{0,0} = 1$. By (1) and (2), the Hilbert order of $(4, 8)$ is

TABLE I
DETERMINING THE MINIMAL/MAXIMAL HILBERT ORDERS IN ONE MAXIMAL BLOCK

v	Beginning point	Ending point	h_b	h_e
0	lower–left	lower–right	h_o	$h_o + (s \times s - 1)$
1	upper–right	lower–right	$h_e - (s \times s - 1)$	$h_o + \sum_{l=0}^{\log s - 1} 2^l \times 2^l$
2	upper· right	upper–left	$h_e - (s \times s - 1)$	$h_o + \sum_{l=0}^{\log s - 1} 2^l \times 2^l$
3	lower–left	upper–left	h_o	$h_o + (s \times s - 1)$

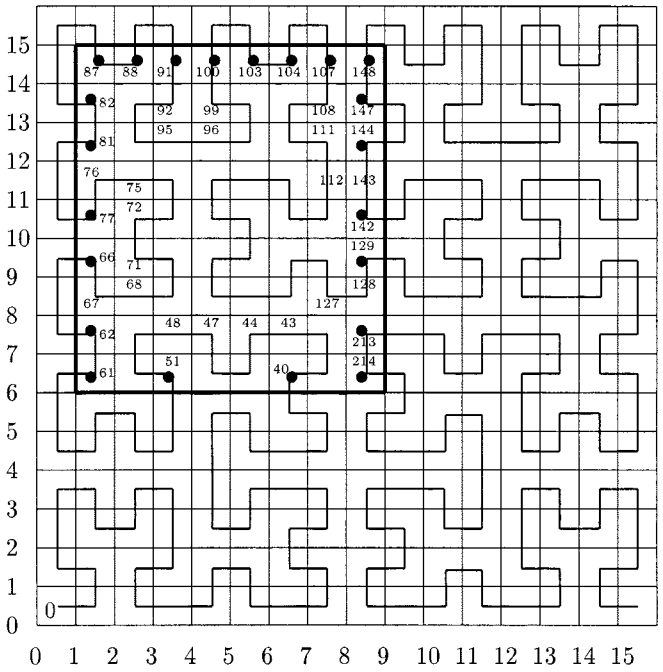


Fig. 5. Hilbert orders map for Fig. 4.

$h_o = (1322)_4 = 122$. Since the block is of size $2^2 \times 2^2$, $v = 2 \times v_{1,1} + v_{0,1} = 2 \times 0 + 1 = 1$ which denotes that the orientation of the corresponding maximal block is to the right. From Table I, we have that the minimal Hilbert order in the maximal block is $h_b = h_e - (4 \times 4 - 1) = 112$ and the maximal Hilbert order is $h_e = h_o + \sum_{l=0}^{1} 2^l \times 2^l = 127$.

After presenting how to determining the minimal and maximal Hilbert orders in one maximal block using $O(\log T)$ time, the proposed formal algorithm consisting of five steps is listed below.

Algorithm: Window Query: Input: An $n_1 \times n_2$ query window W and the compressed image I with N^* codes.

Output: The corresponding codes in W.

Step 1—(Generating Maximal Blocks): For the query window W, the set of maximal blocks, say M, corresponding to W is generated by using the linear-time algorithm proposed by Tsai *et al.* [5]. Note that for W, there are $O(n_l)$ maximal blocks in the worst case [11].

Step 2—(Computing the Related Hilbert Orders): For each maximal block, we compute the minimal and maximal Hilbert orders using Table I and it can be done in $O(\log T)$ time. For a 1×1 maximal block,the minimal and maximal Hilbert orders are set to be the same. Since for the window, there are at most $O(n_l)$ maximal blocks, this step takes $O(n_l \log T)$ time to determine the related Hilbert orders for these $O(n_l)$ maximal blocks.

TABLE II
SIMULATION FOR SORTING AND MERGING

Maximal Blocks	(1,6,1), (1,7,1), ..., (1,14,1); (8,6,1), (8,7,1), ..., (8,14,1); (2,14,1), (3,14,1), ..., (7,14,1); (2,6,2), (2,8,2), ..., (2,12,2); (4,6,2), (6,6,2); (4,12,2), (6,12,2); (4,8,4)
Hilbert orders	(61), (62), (67), (66), (77), (76), (81), (82), (87); (214), (213), (128), (129), (142), (143), (144), (147), (148); (88), (91), (100), (103), (104), (107); (48,51), (68,71), (72,75), (92,95); (44,47), (40,43); (96,99), (108,111); (112,127)
Sorting	(40,43), (44,47), (48,51), (61), (62), (66), (67), (68,71), (72,75), (76), (77), (81), (82), (87), (88), (91), (92,95), (96,99), (100), (103), (104), (107), (108,111), (112,127), (128), (129), (142), (143), (144), (147), (148), (213), (214)
Merging	(40,51),(61,62),(66,77), (81,82),(87,88),(91,100),(103,104),(107,129),(142,144),(147,148) (213,214)

Then, these calculated Hilbert orders form a sequence S. Up to here, the generated maximal blocks in M are transferred to the sequence S.

Step 3: (Sorting): Since the total maximal blocks in M are not obtained as the orders of the Hilbert scan, the sequence S needs to be sorted to be an increasing sequence. In this step, the quick-sort algorithm is used and it takes $O(n_l \log n_l)$ time. The sorted sequence is denoted by S^*.

Step 4: (Merging): If the difference of the Hilbert orders of the beginning point of the current block and the ending point of the previous block is exactly one, the two consecutive Hilbert subcurves in the two blocks can be merged to be one larger curve in order to reduce the number of Hilbert orders used. Only the Hilbert orders of the beginning point of the first block and the ending point of the last block are needed to represent the new merged block if some blocks can be merged. We thus merge the sequence S^* and obtain the merged sorted-sequence S_{new}. Since the sequence S^* needs to be traversed only once, this step takes $O(n_l)$ time. The size of S_{new} is bounded by $O(n_l)$.

Step 5: (Querying): For each pair of Hilbert orders in S_{new}, we now want to find the corresponding codes in the compressed image I. Consider the first pair in S_{new}, say B^* and E^* for $B^* < E^*$. Using B^* as the first key, the binary search [9] is used to find the Hilbert code in I, say $(B_j, E_j, G_{B_j}, G_{E_j})$, at which the condition $B_j \leq B* \leq E_j$ holds. If B^* is not equal to B_j, the code is split into $(B_j, B^*, G_{B_j}, G_{B^*})$ and $(B^*, E_j, G_{B^*}, G_{E_j})$, where $G_{B^*} = G_{B_j} + (G_{E_j} - G_{B_j}/E_j - B_j) \times (B^* - B_j)$. Similarly, using E^* as the second key, the binary search is used again to find the Hilbert code in I, say $(B_k, E_k, G_{B_k}, G_{E_k})$, at which the condition $B_k \leq E^* \leq E_k$ holds. If E^* is not equal to E_k, the code is split into $(B_k, E^*, G_{B_k}, G_{E^*})$ and $(E^*, E_k, G_{E^*}, G_{E_k})$, where $G_{E^*} = G_{B_k} + (G_{E_k} - G_{B_k}/E_k - B_k) \times (E^* - B_k)$. Then, we output the codes between B^* to E^*. Since there are at most T^2 segments and $O(n_l)$ maximal blocks, this step totally takes $O(n_l \log T^2) = O(n_l \log T)$ time for the binary search and $O(P)$ to output the queried result, where P is the number of outputted codes.

From the above five steps, we have the following main result.

Theorem 1: Given a query window of size $n_1 \times n_2$, the proposed algorithm takes $O(n_l \log T + P)$ time to perform the window query on a compressed image, where $n_l = \max(n_1, n_2)$; $T \times T$ is the image size, and P is the number of outputted codes.

Return to Fig. 5. A simulation for Steps 1–4 is given. Initially, the query window has 33 maximal blocks by Step 1. These maximal blocks are shown in the first row in Table II. In Step 2, we compute the minimal and maximal Hilbert orders for each maximal block. These pairs of Hilbert orders are shown in the second row in Table II. Then, they are sorted in an increasing order by Step 3. The sorted results are shown in the third row in Table II. By Step 4, 33 ordered pairs of Hilbert orders are merged into 11 pairs and they are shown in the last row in Table II. Thus, we can reduce the number of accessing times for the window query and achieve the goal of speeding up the window query.

Fig. 6. Four gray images.

VI. EXPERIMENTAL RESULTS

In this section, some experimentations are carried out to demonstrate the performance of the proposed algorithm and the naive algorithm. Both algorithms are implemented in Borland C++ builder and are executed on the *Pentium 233*-based *PC* with the same inputs. Four 256×256 gray images, say F16, kids, bridge, and boat (see Fig. 6), compressed by using Hilbert scan with parameters $L = 128$ and $T^* = 15$ [14] are used

TABLE III
EXPERIMENTAL RESULTS FOR SQUARE WINDOWS ON F16

Maximal Blocks	(1,6,1), (1,7,1), ..., (1,14,1); (8,6,1), (8,7,1), ..., (8,14,1); (2,14,1), (3,14,1), ..., (7,14,1); (2,6,2), (2,8,2), ..., (2,12,2); (4,6,2), (6,6,2); (4,12,2), (6,12,2); (4,8,4)
Hilbert orders	(61), (62), (67), (66), (77), (76), (81), (82), (87); (214), (213), (128), (129), (142), (143), (144), (147), (148); (88), (91), (100), (103), (104), (107); (48,51), (68,71), (72,75), (92,95); (44,47), (40,43); (96,99), (108,111); (112,127)
Sorting	(40,43), (44,47), (48,51), (61), (62), (66), (67), (68,71), (72,75), (76), (77), (81), (82), (87), (88), (91), (92,95), (96,99), (100), (103), (104), (107), (108,111), (112,127), (128), (129), (142), (143), (144), (147), (148), (213), (214)
Merging	(40,51),(61,62),(66,77), (81,82),(87,88),(91,100),(103,104),(107,129),(142,144),(147,148) (213,214)

as the inputs. Under the threshold $T^* = 15$, the number of bits required for representing one pixel in average is 3.595 with respect to the original 8 bits for representing one pixel. In addition, the signal to noise (SNR) ratio is 33.406.

First, we set $n_1 = n_2 = n$ and randomly choose the starting points to generate 100 query windows. From Theorem 1, the parameters T and P are the same for the proposed algorithm and the naive algorithm. Thus, the execution time required in the proposed algorithm and the naive algorithm are rewritten as $C_{ours} \times n$ for some constant C_{ours} and $C_{nav} \times n^2$ for some constant C_{nav}, respectively. The experimental results for square windows are shown in Tables III–VI for F16, kids, bridge, and boat, respectively. For square windows with different widths, each row in the four tables shows the total execution time for performing 100 window queries using the proposed algorithm and the naive algorithm, which are denoted by T_{ours} and T_{nav}, respectively. The symbol "sec" denotes second. From C_{ours} (C_{nav}), it is observed that the execution time of the proposed algorithm (the naive algorithm) are linearly proportional to n (n^2) within a small range centered around 0.0014 (0.000 28). Hence, the experimental results confirm the theoretical analysis. The improvement ratio of the execution time required in the proposed algorithm over the naive algorithm is denoted by $R_s = (T_{nav} - T_{ours}/T_{nav}) \times 100\%$ as shown in the final column of the following tables.

Second, the arbitrary rectangular windows are used as the inputs. Ten types of area are used in the experimentations and they are 3000, 5000, 7000, $\cdots$, and 21 000. For each specific area, 100 query windows are generated by randomly choosing the starting points and the width and height of the window. The comparison of execution time between the proposed algorithm and the naive algorithm are shown in Fig. 7(a)–(d), for F16, kids, bridge, and boat, respectively, where the horizontal axis in each figure denotes the window area, each unit being 10^3. From Tables III–VI and Fig. 7, it is observed that the proposed algorithm is faster than the naive algorithm and has about 72–98% time improvement. From Fig. 7, the larger the query window becomes, the better the performance is. This fits Theorem 1.

TABLE IV
EXPERIMENTAL RESULTS FOR SQUARE WINDOWS ON KIDS

Window size n	T_{ours} (sec)	C_{ours}	T_{nav} (sec)	C_{nav}	R_s
20	0.219	0.001369	0.891	0.000278	75.4209
40	0.312	0.000975	3.661	0.000286	91.4777
60	0.613	0.001277	7.977	0.000277	92.3154
80	0.922	0.001441	14.176	0.000277	93.4960
100	1.102	0.001378	22.219	0.000278	95.0403
120	1.434	0.001494	32.008	0.000278	95.5199
140	1.582	0.001413	43.609	0.000278	96.3723
160	1.875	0.001465	57.102	0.000279	96.7164
180	2.249	0.001562	72.251	0.000279	96.8872
200	2.512	0.001570	89.273	0.000279	97.1862

TABLE V
EXPERIMENTAL RESULTS FOR SQUARE WINDOWS ON BRIDGE

Window size n	T_{ours} (sec)	C_{ours}	T_{nav} (sec)	C_{nav}	R_s
20	0.199	0.001244	0.883	0.000276	77.4632
40	0.512	0.001600	3.469	0.000271	85.2407
60	0.637	0.001327	7.941	0.000276	91.9783
80	0.953	0.001489	14.141	0.000276	93.2607
100	1.152	0.001440	22.207	0.000278	94.8124
120	1.391	0.001449	32.043	0.000278	95.6590
140	1.152	0.001350	43.669	0.000279	96.5376
160	1.931	0.001509	57.012	0.000278	96.6130
180	2.227	0.001547	72.251	0.000279	96.9177
200	2.492	0.001558	89.238	0.000279	97.2075

TABLE VI
EXPERIMENTAL RESULTS FOR SQUARE WINDOWS ON BOAT

Window size n	T_{ours} (sec)	C_{ours}	T_{nav} (sec)	C_{nav}	R_s
20	0.203	0.001269	0.906	0.000283	77.5938
40	0.402	0.001256	3.578	0.000280	88.7647
60	0.594	0.001238	7.965	0.000277	92.5424
80	0.926	0.001447	14.141	0.000276	93.4517
100	1.148	0.001435	22.152	0.000277	94.8176
120	1.422	0.001481	31.977	0.000278	95.5531
140	1.527	0.001363	43.591	0.000278	96.4970
160	1.891	0.001477	57.019	0.000278	96.6838
180	2.152	0.001494	72.258	0.000279	97.0218
200	2.434	0.001521	89.367	0.000279	97.2764

VII. CONCLUSIONS

We have presented a fast algorithm for window query on Hilbert-scan-based compressed gray images. Using the strategy of maximal quadtree blocks to decompose the query window, each maximal block is used to perform the window query and the query time is reduced. Given a query window of size $n_1 \times n_2$, the proposed algorithm takes $O(n_l \log T + P)$ time to perform the window query, where $n_l = \max(n_1, n_2)$, $T \times T$ is the image size, and P is the number of outputted codes. The proposed algorithm improves the naive algorithm, which needs $O(n_1 n_2 \log T + P)$ time, significantly. The experimental results also confirm the theoretical analysis.

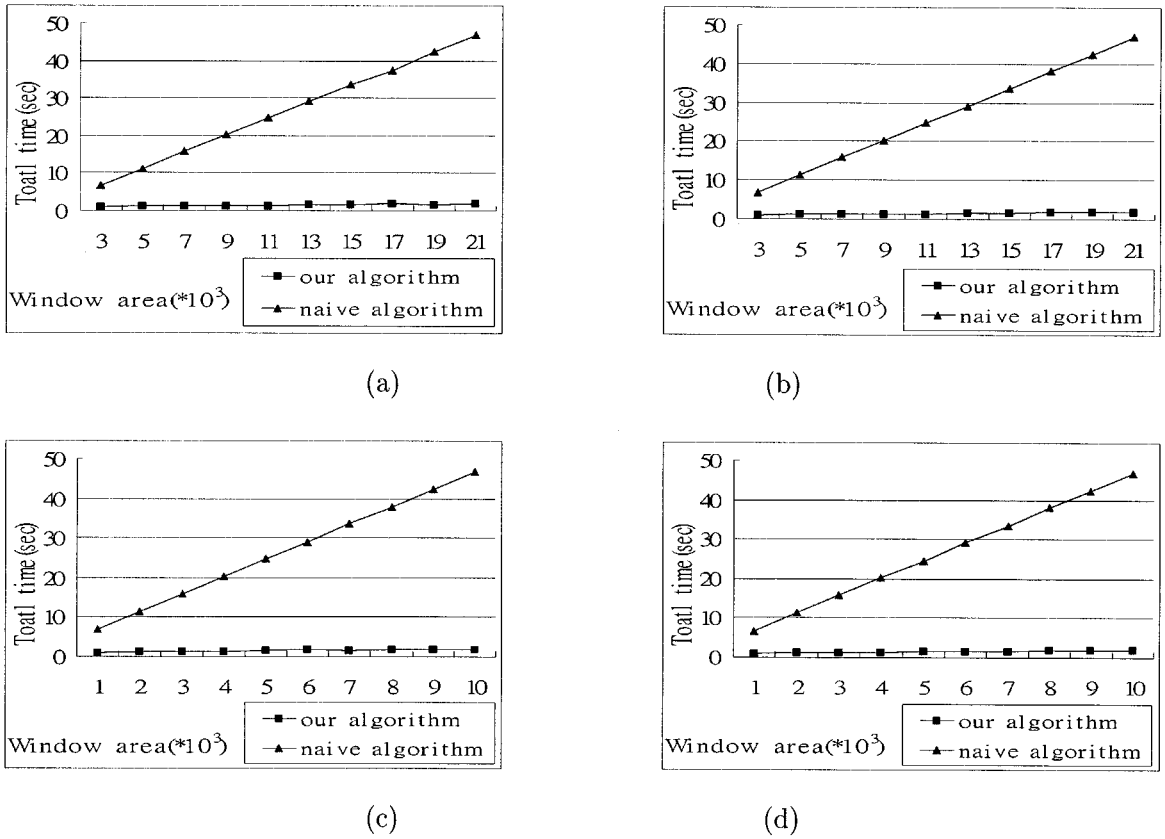


Fig. 7. Performance comparison for rectangular windows.

ACKNOWLEDGMENT

The authors are indebted to the anonymous reviewers, Editor-in-Chief A. Bovik, and Associate Editor T. Chen for their valuable suggestions that lead to the improved version of the paper.

REFERENCES

[1] C. Arcelli and G. Ramella, "Feature points for a polygonal representation of silhouettes," in *Progress in Image Analysis and Processing II*. Singapore: World Scientific, Sept. 1991, pp. 107–114.

[2] W. G. Aref and H. Samet, "Efficient processing of window queries in the pyramid data," in *Proc. 9th ACM-SIGMOD Symp. Principles Database Systems*, Apr. 1990, pp. 265–272.

[3] W. G. Aref and H. Samet, "Decomposing a window into maximal quadtree blocks," *Acta Inform.*, vol. 30, pp. 425–439, 1993.

[4] T. Asano, D. Ranjan, T. Roos, and E. Welzl, "Space-filling curves and their use in the design of geometric data structures," *Theor. Comput. Sci.*, vol. 181, pp. 3–15, 1997.

[5] T. Bially, "Space-filling curves: Their generation and their application to bandwidth reduction," *IEEE Trans. Inform. Theory*, vol. IT-15, no. 6, pp. 658–664, 1969.

[6] N. Bourbakis and C. Alexopoulos, "Picture data encryption using scan patterns," *Pattern Recognit.*, vol. 25, no. 6, pp. 567–581, 1992.

[7] G. Cantor, "Ein Beitrag zur Mannigfaltigkeitslehre," *Crelle J.*, vol. 84, pp. 242–258, 1878.

[8] K. L. Chung and L. C. Chang, "Encrypting binary images with higher security," *Pattern Recognit. Lett.*, vol. 19, pp. 461–468, 1998.

[9] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge, MA: MIT Press, 1990.

[10] J. G. Dunham, "Optimal uniform piecewise linear approximation of planar curves," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-8, no. 1, pp. 67–75, 1986.

[11] C. R. Dyer, "The space efficiency of quadtrees," *Comput. Graph. Image Process.*, vol. 19, no. 4, pp. 335–348, 1982.

[12] D. Hilbert, "Über die stetige Abbildung einer Linie auf ein Flächenstück," *Math. Ann.*, vol. 38, pp. 459–460, 1891.

[13] H. V. Jafadish, "Analysis of the Hilbert curve for representing two-dimensional space," *Inform. Process. Lett.*, vol. 62, pp. 17–22, 1997.

[14] F. C. Jian, "Hilbert curves and its applications on image processing," M.S. thesis, Dept. Elect. Eng., Nat. Taiwan Univ., Taiwan, R.O.C., June 1996.

[15] S. Kamata, M. Niimi, and E. Kawaguchi, "A method of an interactive analysis for multi-dimensional images using a Hilbert curve," *IEICE Trans.*, vol. J77-D-II, no. 7, pp. 1255–1264, 1993.

[16] S. Kamata, R. O. Eaxon, and E. Kawaguchi, "An implementation of the Hilbert scanning algorithm and its application to data compression," *IEICE Trans. Inform. Syst.*, vol. E76-D, no. 4, 1993.

[17] S. Kamata, M. Niimi, and E. Kawaguchi, "A gray image compression using a Hilbert scan," in *Proc. Int. Conf. Pattern Recognition '96*, 1996.

[18] H. Lebesgue, *Leçons sur l'Intégration et la Recherche des Fonctions Primitives*. Paris, France: Gauthier-Villars, 1904, pp. 44–45.

[19] X. Liu and G. F. Schrack, "Encoding and decoding the Hilbert order," *Softw. Pract. Exper.*, vol. 26, no. 12, pp. 1335–1346, 1996.

[20] X. Liu and G. F. Schrack, "An algorithm for encoding and decoding the 3-D Hilbert order," *IEEE Trans. Image Processing*, vol. 6, pp. 1333–1337, Sept. 1997.

[21] E. H. Moore, "On certain crinkly curves," *Trans. Amer. Math. Soc.*, vol. 1, pp. 72–90, 1900.

[22] E. Nardelli, "Efficient secondary memory processing of window queries on spatial data," *Inform. Sci.*, vol. 84, pp. 67–83, 1995.

[23] E. Nardelli and G. Proietti, "Time and space efficient secondary memory representation of quadtrees," *Inform. Syst.*, vol. 22, pp. 25–37, 1997.

[24] G. Peano, "Sur une courbe qui remplit toute une aire plane," *Math. Ann.*, vol. 36, pp. 157–160, 1890.

[25] M. K. Quweider and E. Salari, "Peano scanning partial distance search for vector quantization," *IEEE Signal Processing Lett.*, vol. 2, pp. 169–171, Sept. 1995.

[26] C. S. Refazzoni and A. Teschioni, "A new approach to vector median filtering based on space filling curves," *IEEE Trans. Image Processing*, vol. 6, pp. 1025–1037, July 1997.

[27] H. Sagan, *Space-Filling Curves*. New York: Springer-Verlag, 1994.

[28] R. J. Stevens, A. F. Lehar, and F. H. Preston, "Manipulation and presentation of multidimensional image data using the peano scan," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-5, pp. 520–526, 1983.
[29] Y. H. Tsai, K. L. Chung, and Y. H. Yang, "Optimal algorithm for decomposing a query window into maximal quadtree blocks," Dept. Inform. Manage. Inst. Inform. Eng., Nat. Taiwan Univ. Sci. Technol., Res. Rep., June 1998.
[30] Y. F. Zhang, "Space-filling curve ordered dither," *Comput. Graph.*, vol. 22, no. 4, pp. 559–563, 1998.

Kuo-Liang Chung received the B.S., M.S., and Ph.D. degrees from the Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, R.O.C.

He is a Professor with the Department of Information Management, Institute of Computer Science and Information Engineering, National Taiwan University of Science and Technology, Taipei. His current research interests include image processing, compression, video processing, graphics, and algorithms.

Yao-Hong Tsai received the M.S. and Ph.D. degrees in information management from the National Taiwan University of Science and Technology (NTUST), Taipei, Taiwan, R.O.C., in 1994 and 1998, respectively.

He now is a Researcher with the Advanced Technology Research Institute and Communications Research Laboratories, Industrial Technology Research Institute, NTUST. His current research interests include image processing, pattern recognition, and spatial data structures.

Fei-Ching Hu received the M.S. degree in computer science and information engineering from the National Taiwan University of Science and Technology, Taipei, Taiwan, R.O.C., in 1999.

She now is a System Engineer with Syscom Computer Engineering Company, Lake Mary, FL. Her current research interests include image processing and computer vision.