SOFTWARE—PRACTICE AND EXPERIENCE, VOL. 26(12), 1335–1346 (DECEMBER 1996)

Encoding and Decoding the Hilbert Order

XIAN LIU*† AND GÜNTHER SCHRACK
Department of Electrical Engineering, The University of British Columbia, Vancouver, Canada V6T 1Z4
(e-mail: schrack@ee.ubc.ca)

SUMMARY

Explicit formulas are presented to generate the coordinates of a point on the two-dimensional Hilbert curve from its location code and vice versa. Execution-time assessments suggest that the proposed algorithms are faster than the ones published previously.

KEY WORDS: space-filling curves ; Hilbert curve; Hilbert order

INTRODUCTION

Space-filling curves, and in particular the Hilbert curve, continue to attract much interest, as demonstrated by a steady stream of articles appearing in the literature. A space-filling curve, defined on an integer lattice, defines a corresponding *order*, e.g. the Hilbert order. By numbering the points visited sequentially on the curve, *location codes* are defined (see Figures 4, 5, 6, 11). For plotting purposes, several sequential algorithms for the Hilbert order (as well as other orders) have been proposed for generating the coordinates in sequence.[1-3]

In recent years, applications from diverse fields that employ the Hilbert order have been described. For instance, it was reported recently that Hilbert curves may significantly improve the performance of spatial data processing systems.[4,5] For such applications, *random-access algorithms* are required, i.e. encoding and decoding without previous context or historical information. An *encoding algorithm* derives the location code from a given coordinate point, whereas a *decoding algorithm* derives the coordinates from a given location code.

Apparently, only the algorithms proposed by Fisher[6] are random-access algorithms; they are serial-bit processes and use two short look-up tables.

In this paper, new encoding and decoding algorithms that do not depend on look-up tables are proposed. They are stated in terms of an explicit formulation, and in addition, according to experimental evidence, they are faster than Fisher's.

ENCODING

The Hilbert order is to be defined in the domain $D = \{x, y \in \mathbf{I} | 0 \leq x < 2^r, 0 \leq y < 2^r\}$, a subset of the two-dimensional integer space. The parameter r is the specified *resolution* or *level* of the spatial order. Small values of r correspond to low resolutions and low levels.

*Current address: Bell-Northern Research Ltd., P.O. Box 3511, Station C, Ottawa, Canada K1Y 4H7 (e-mail: xianliu@bnr.ca)
†Corresponding author

CCC 0038–0644/96/121335–12
©1996 by John Wiley & Sons, Ltd.
Received 23 September 1994
Revised 19 October 1995 and 26 February 1996

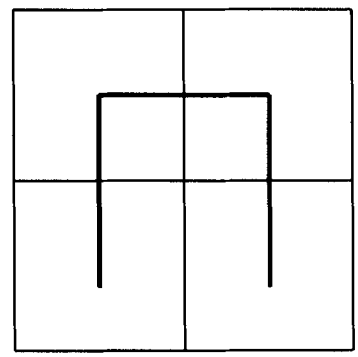

Figure 1. The Hilbert curve of resolution 1

The Hilbert curves of resolution 1, 2 and 3 and their corresponding orders are depicted in Figures 1 to 6.

Given the coordinates of a point P in the lattice system D, the corresponding location code of P is to be determined. Let the coordinate values x and y be expressed by bit-strings: $x = x_{r-1} \ldots x_1 x_0$ and $y = y_{r-1} \ldots y_1 y_0$, i.e., $x = \Sigma_0^{r-1} 2^i x_i, y = \Sigma_0^{r-1} 2^i y_i$, where $x_i, y_i \in \{0, 1\}$, and the corresponding location code by a quaternary digit string $h = h_{r-1} \ldots h_1 h_0$, i.e. $h = \Sigma_0^{r-1} 4^i h_i$, $h_i \epsilon \{0, 1, 2, 3\}$.

A pattern that specifies the indexing strategy for the four quadrants of a square domain will be called a *unit*. It is a simple graph with three edges and four *nodes* labelled by quaternary indices, the *node indices*, as illustrated in Figure 7. Two features are associated with a unit, orientation and aspect. The *orientation* indicates the current state of rotation of a unit, whereas the *aspect* specifies the relation between the quaternary digits h_i of a location code h and the node indices. The nodes of a unit are successively indexed clockwise, but the sequence of the digits of h may run either clockwise or counterclockwise. The orientation of the unit at the lowest level (i.e. level 1, corresponding to the resolution $r = 1$) is called the *principal orientation* of the Hilbert order.

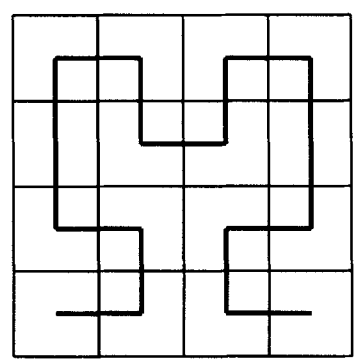

Figure 2. The Hilbert curve of resolution 2

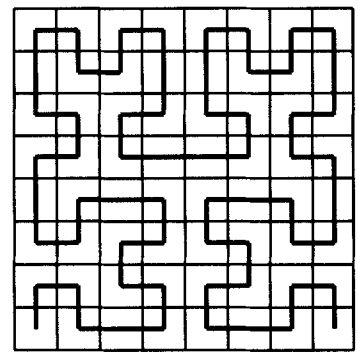

Figure 3. The Hilbert curve of resolution 3

Both orientation and aspect determine the *structure* of a unit. The orientation is a key feature in the iterative construction process and its calculation can be isolated from that of the location code and the aspect. According to the inherent property of the Hilbert order, the units generated from nodes 0, 1, 2, 3 of a unit at level k ($k = 1, 2, \ldots, r-1$) always rotate 90 degrees clockwise, 0 degrees, 0 degrees, and 90 degrees counterclockwise, respectively, relative to the orientation of the generating unit.

The aspect of a unit depends only on its orientation; or, equivalently, on the node of the level below from which the unit is generated. The aspect of a unit generated from nodes 1 or 2 of the unit at the lower level remains unchanged, whereas that generated from nodes 0 and 3 is always reversed.

As will be shown, a quaternary digit h_i of a location code depends on the binary digits x_i and y_i of the coordinates as well as the structure of its unit at level $r-i$. Because an orientation depends on the $i-1$ most significant binary digits of the coordinates, each quaternary digit h_i is a function $h_i(x_k, y_k)$, $k = r-1, r-2, \ldots, i$, $i = r-1, \ldots, 1, 0$.

The structure of level 1, at the principal orientation, is specified by default, see Figure 7. It determines the overall appearance of the order considered. Level 2 is generated by constructing four units, labelled U_k ($k = 0, 1, 2, 3$), from the corresponding indexed nodes of the unit at level 1 (see Figure 8).

The orientation of a unit may be specified by a normalized vector m, where $m \in \{(0,1)^T, (-1,0)^T, (0,-1)^T, (1,0)^T\}$. Two units with their normal vector are depicted in Figure 9. The orientation of a unit could be obtained from the previous level by multiplying vector m by a 2×2 rotation matrix. An alternative approach, however, using *directional indices*, is preferable. Indexing the directions is feasible because the rotational freedom in the two-

1	2
0	3

Figure 4. The Hilbert order of resolution 1

5	6	9	10
4	7	8	11
3	2	13	12
0	1	14	15

Figure 5. The Hilbert order of resolution 2

dimensional space is restricted to the directions of the four principal axes, x, y, $-x$, and $-y$, hence one of four index values is sufficient to characterize a unit's orientation. The representation of the directional vectors m by the directional indices v is defined arbitrarily (see Figure 10).

Because an index v has four possible values, it can be represented by two bits v_1 and v_0 such that $v = 2v_1 + v_0$. Similarly, two bits n_{2k-1} and n_{2k-2} are used to represent a node index. The four orientations are mapped onto the nodes according to the inherent structure of the Hilbert order. Thus, n_{2k-1} and n_{2k-2} can be expressed in terms of v_1, v_0, x_{k-1} and y_{k-1} by*:

$$\begin{aligned} n_{2k-1} &= \overline{v}_1\overline{v}_0 x_{k-1} + \overline{v}_1 v_0 y_{k-1} + v_1\overline{v}_0\overline{x}_{k-1} + v_1 v_0\overline{y}_{k-1} \\ &= \overline{v}_0(v_1 \oplus x_{k-1}) + v_0(v_1 \oplus y_{k-1}) \qquad (1) \\ n_{2k-2} &= v_0 \oplus x_{k-1} \oplus y_{k-1} \qquad (2) \\ & \quad k = 1, \ldots, r. \end{aligned}$$

The iteration of v can be carried out by either multiplying or adding an incremental factor that is governed by the inherent structure of a particular Hilbert order. Here, addition is chosen as it appears to facilitate analysis as well as implementation. Denote the incremental factor by dv, then the transition rules may be expressed explicitly by two `if`-clauses:

```
    if  nodeindex  n = 0  then  dv = 3;
elseif  nodeindex  n = 3  then  dv = 1
  else  dv = 0;
```

Because $dv \in \{0, 1, 3\}$, two bits suffice to represent dv. With the two bits denoted as dv_1 and dv_0 such that $dv = 2dv_1 + dv_0$, the relation defined by the `if`-clauses above may be written as a table (Table I). It can also be formulated explicitly as two Boolean expressions:

$$dv_1 = \overline{n}_{2k-1}\overline{n}_{2k-2} \qquad (3)$$

$$dv_0 = n_{2k-1} \oplus \overline{n}_{2k-2} \qquad (4)$$

The updated value v' of the directional index v is obtained by modulo-4 addition:

$$v' = (v + dv) \bmod 4 \qquad (5)$$

* The following notation will be used for Boolean variables p and q:
p *and* q is denoted as pq,
p *inclusive or* q is denoted as $p + q$,
p *exclusive or* q is denoted as $p \oplus q$, and
the complement of p is denoted as $\overline{p}$.

Table I. Incremental factors of v

n_{2k-1}	n_{2k-2}	dv_1	dv_0
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	1

Let v' be represented by two bits v'_1 and v'_0; then formula (5) becomes:

$$v'_1 = (v_0\, dv_0) \oplus v_1 \oplus dv_1 \tag{6}$$
$$v'_0 = v_0 \oplus dv_0 \tag{7}$$

Substituting formulas (3) and (4) into (6) and (7) yields

$$v'_1 = [v_0(n_{2k-1} \oplus \overline{n}_{2k-2})] \oplus v_1 \oplus (\overline{n}_{2k-1}\overline{n}_{2k-2}) \tag{8}$$
$$v'_0 = v_0 \oplus n_{2k-1} \oplus \overline{n}_{2k-2} \tag{9}$$

To obtain the final expressions of v'_1 and v'_0 in terms of v_1, v_0, x_{k-1}, and y_{k-1}, substitute formulas (1) and (2) into (8) and (9) and simplify. The directional index bits for all levels can then be calculated by an iteration, as follows:

$$\begin{aligned} v_{1,r-1} &= 0 \\ v_{0,r-1} &= 0 \\ v_{1,j-1} &= v_{1,j}(x_j \oplus y_j) + v_{0,j}x_j y_j + \overline{v}_{0,j}\overline{x}_j\,\overline{y}_j \\ &= v_{1,j}(x_j \oplus y_j) + (\overline{x}_j \oplus y_j)(v_{0,j} \oplus \overline{y}_j) \qquad (10) \\ v_{0,j-1} &= v_{0,j}(v_{1,j} \oplus \overline{x}_j) + \overline{v}_{0,j}(v_{1,j} \oplus \overline{y}_j) \qquad (11) \\ & \quad j = r-1, \ldots, 2, 1. \end{aligned}$$

Finally, to determine the location code of a point at coordinates x and y, each quaternary digit h_i is calculated as a function of x_i, y_i, and the structure of the unit at level $r - i$. There are four possible cases, one for each orientation. Changing notation, let the two bits of a digit of h be denoted by h_{2k+1} and h_{2k}, respectively. Then

$$\begin{aligned} h_{2k+1} &= \overline{v}_{1,k}\overline{v}_{0,k}x_k + \overline{v}_{1,k}v_{0,k}\overline{y}_k + v_{1,k}\overline{v}_{0,k}\overline{x}_k + v_{1,k}v_{0,k}y_k \\ &= \overline{v}_{0,k}(v_{1,k} \oplus x_k) + v_{0,k}(v_{1,k} \oplus \overline{y}_k) \qquad (12) \\ h_{2k} &= x_k \oplus y_k \qquad (13) \\ & \quad k = 0, 1, \ldots, r-1. \end{aligned}$$

The location code is obtained by interleaving the bits h_{2k+1} and h_{2k}. As an example, the complete set of location codes of the Hilbert order of resolution $r = 4$ is shown in Figure 11.

21	22	25	26	37	38	41	42
20	23	24	27	36	39	40	43
19	18	29	28	35	34	45	44
16	17	30	31	32	33	46	47
15	12	11	10	53	52	51	48
14	13	8	9	54	55	50	49
1	2	7	6	57	56	61	62
0	3	4	5	58	59	60	63

Figure 6. The Hilbert order of resolution 3

DECODING

Decoding is the inverse process of encoding: given the location code of a particular point, the corresponding Cartesian coordinate pair (x, y) is to be determined. An explicit formula for decoding can be derived by the same methodology as used for encoding. Using the previous notation again,

$$n_{2k-1} = v_0 \oplus h_{2k-1} \tag{14}$$

$$n_{2k-2} = v_0 \oplus h_{2k-2} \tag{15}$$

$$k = 1, 2, \ldots, r.$$

The transition rules for iterating the directional index v are identical to those for the encoding process. Substituting formulas (14) and (15) into (8) and (9), followed by a lengthy simplification and recasting into an iteration yields

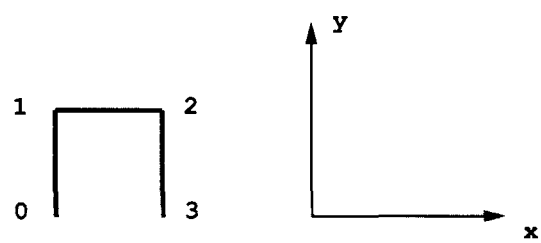

Figure 7. Orientation and aspect of level 1

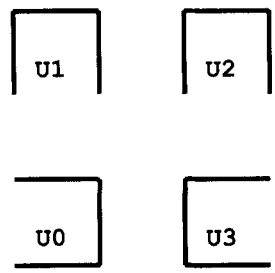

Figure 8. Orientations of units of level 2

$$
\begin{aligned}
v_{1,r-1} &= 0 \\
v_{0,r-1} &= 0 \\
v_{1,j-1} &= v_{1,j} \oplus (\overline{h}_{2j}\, \overline{h}_{2j+1}) && (16)\\
v_{0,j-1} &= v_{0,j} \oplus h_{2j} \oplus \overline{h}_{2j+1} && (17)\\
& \quad j = r-1, \ldots, 2, 1.
\end{aligned}
$$

To determine the coordinates of a point given by its location code, each bit x_i and y_i of the coordinates is calculated as a function of the quaternary digit of h and the structure of the unit at level $r - i$. From the four possible cases result the formulas

$$
\begin{aligned}
x_k &= \overline{v}_{1,k}\overline{v}_{0,k}h_{2k+1} + \overline{v}_{1,k}v_{0,k}(\overline{h}_{2k+1} \oplus h_{2k}) + v_{1,k}\overline{v}_{0,k}\overline{h}_{2k+1} + v_{1,k}v_{0,k}(h_{2k+1} \oplus h_{2k}) \\
&= (v_{0,k}\overline{h}_{2k}) \oplus v_{1,k} \oplus h_{2k+1} && (18)\\
y_k &= \overline{v}_{1,k}\overline{v}_{0,k}(h_{2k+1} \oplus h_{2k}) + \overline{v}_{1,k}v_{0,k}\overline{h}_{2k+1} + v_{1,k}\overline{v}_{0,k}(\overline{h}_{2k+1} \oplus h_{2k}) + v_{1,k}v_{0,k}h_{2k+1} \\
&= (v_{0,k} + h_{2k}) \oplus v_{1,k} \oplus h_{2k+1} && (19)\\
& \quad k = 0, 1, \ldots, r-1.
\end{aligned}
$$

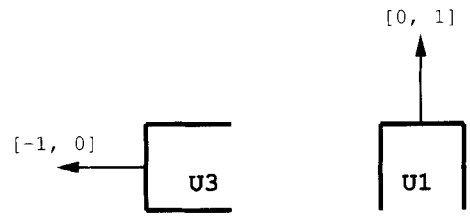

Figure 9. Two units with directional vectors at level 2

vector m	*index* v
$(0,1)^T$	0
$(-1,0)^T$	1
$(0,-1)^T$	2
$(1,0)^T$	3

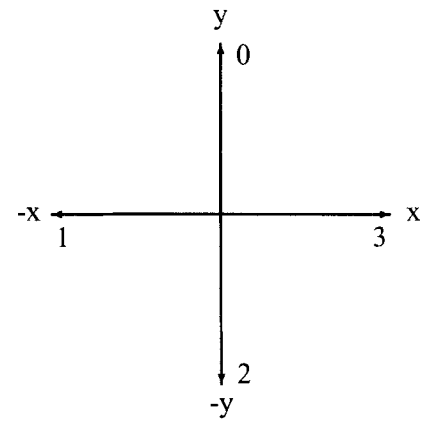

Figure 10. Directional indices

IMPLEMENTATIONS AND EXECUTION-TIME ASSESSMENTS

The operands in the foregoing formulas are binary digits. Many programming languages support the bit-wise logical operations `and`, `or`, `exclusive or`, and `complement` on entire machine words. Formulas (12), (13), (18), and (19) can therefore be implemented as single assignment statements. The same is true for formulas (10), (11), (16), and (17); however, they need to be embedded in a loop. Two integer variables, h_{odd} and h_{even}, are introduced; they comprise the bits at the odd and even digit positions of the location code, respectively. Thus, when interleaved bit-wise, they will yield the location code.

The two algorithms, implemented in the C language, follow. All variables are assumed to have been declared as integers; the resolution r is treated as a constant parameter. Since complementing operations are involved, it is necessary to mask out leading 1-bits in the complemented variables. In addition, for efficiency, all repetitive operations have been moved outside the iteration loop.

The encoding algorithm expects the coordinates x and y as input parameters; it delivers the variables $h_{odd} = h_{2r-1} \ldots h_3 h_1$ and $h_{even} = h_{2r-2} \ldots h_2 h_0$, and, after interleaving, the location code h:

```
mask = (1 << r) - 1;
heven = x ^ y;
notx = ~ x & mask;
noty = ~ y & mask;
temp = notx ^ y;
v1 = v0 = 0;
for (k = 1; k < r; ++ k) {
  v1 = ((v1 & heven) | ((v0 ^ noty) & temp)) >> 1;
  v0 = ((v0 & (v1 ^ notx)) | (~ v0 & (v1 ^ noty))) >> 1;
}
hodd = (~ v0 & (v1 ^ x)) | (v0 & (v1 ^noty));
h = interleave(hodd,heven);
```

y																
15	85	86	89	90	101	102	105	106	149	150	153	154	165	166	169	170
14	84	87	88	91	100	103	104	107	148	151	152	155	164	167	168	171
13	83	82	93	92	99	98	109	108	147	146	157	156	163	162	173	172
12	80	81	94	95	96	97	110	111	144	145	158	159	160	161	174	175
11	79	76	75	74	117	116	115	112	143	140	139	138	181	180	179	176
10	78	77	72	73	118	119	114	113	142	141	136	137	182	183	178	177
9	65	66	71	70	121	120	125	126	129	130	135	134	185	184	189	190
8	64	67	68	69	122	123	124	127	128	131	132	133	186	187	188	191
7	63	62	49	48	47	44	43	42	213	212	211	208	207	206	193	192
6	60	61	50	51	46	45	40	41	214	215	210	209	204	205	194	195
5	59	56	55	52	33	34	39	38	217	216	221	222	203	200	199	196
4	58	57	54	53	32	35	36	37	218	219	220	223	202	201	198	197
3	5	6	9	10	31	28	27	26	29	228	227	224	245	246	249	250
2	4	7	8	11	30	29	24	25	230	231	226	225	244	247	248	251
1	3	2	13	12	17	18	23	22	233	232	237	238	243	242	253	252
0	0	1	14	15	16	19	20	21	234	235	236	239	240	241	254	255
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15 x

Figure 11. The Hilbert order of resolution 4

The function `interleave` assembles h by interleaving h_{odd} and h_{even}.

The decoding algorithm expects the variables h_{odd} and h_{even} as input parameters, obtained from the location code h by deleaving; it delivers the coordinates x and y:

```
deleave (h, &hodd, &heven);
mask = (1 << r) - 1;
v1 = v0 = 0;
temp1 = ~ (heven | hodd) & mask;
temp0 = ~ (heven ^ hodd) & mask;
for (k = 1; k < r; ++ k) {
  v1 = (v1^temp1) >> 1;
  v0 = (v0^temp0) >> 1;
}
x = (v0 & ~ heven)^v1^hodd;
y = (v0  |  heven)^v1^hodd;
```

For interleaving and deleaving the bits of a variable (the functions `interleave` and `deleave` above), the second author of this paper has designed a set of efficient algorithms[7] by introducing specialized data structures. A detailed discussion is not within the scope of this paper.

For verification, execution-time assessment, and comparison, Fisher's[6] and the proposed algorithms were implemented in the C language. The input and output variables of Fisher's encoding algorithm ('Berthil') and decoding algorithm ('Hilbert') are the same as those for above algorithms.

Fisher's and the proposed algorithms are of linear order $O(r)$ because of the loop each one requires. Fisher's algorithms carry out all calculations within the loop, including interleaving and deleaving. In contrast, the proposed algorithms separate the calculation of the directional indices from the remaining calculations which is the reason for an improved execution-time behaviour of the proposed algorithms.

For the timing experiments, the encoding algorithms were embedded in a double loop $x, y = 0, \ldots, 2^r - 1$, i.e., timed for all grid values of each resolution for the set $r = 7, \ldots, 12$. Similarly, the decoding algorithms were embedded in the single loop $h = 0, \ldots, 4^r - 1$. The algorithms were executed and timed using the C library function `clock`. The total time was divided by 4^r and normalized to the value of the proposed algorithm for $r = 7$. The experiments were carried out on Sun SPARC stations IPC, IPX, stations 2, 5, and 10, with close results for each platform. The results for the IPX and station 5 machines are plotted in Figures 12 and 13 for encoding and decoding, respectively.

For the reasons stated above, and despite the fact that the proposed encoding algorithm has to post-process its results h_{odd} and h_{even} to obtain h, and the decoding algorithm needs to pre-process h, the encoding algorithm is still 13% and 18% faster (depending on the type of machine) than Fisher's, whereas the decoding algorithm is 83% and 98% faster.

Logical operations on entire machine-words restrict the applicability of the implemented algorithms to resolutions $\lfloor w/2 \rfloor$, where w is the word length of a machine. For example, for

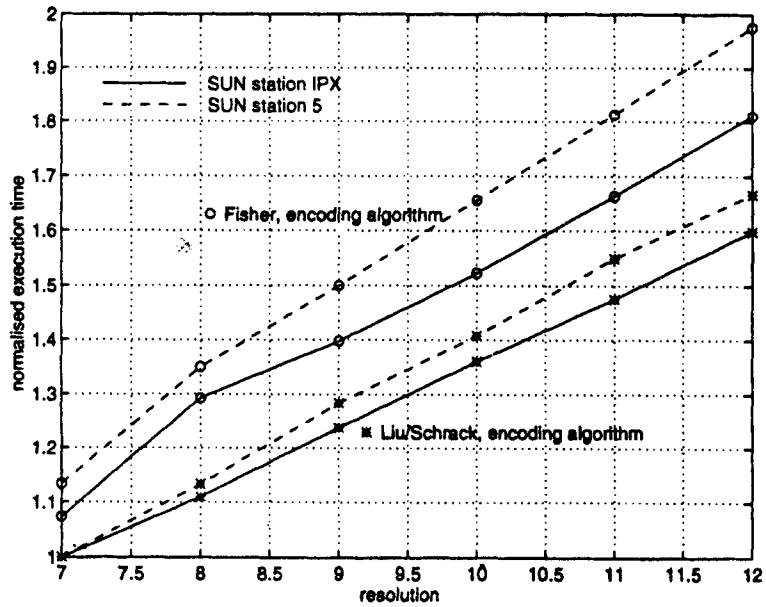

Figure 12. Comparison of encoding algorithms for the Hilbert order

variables declared `unsigned int` and a 32-bit machine, the maximum resolution is 16, i.e., corresponding to a domain of size $2^{16} \times 2^{16}$.

CONCLUSION

New explicit formulas to calculate the location code of an arbitrary point on the Hilbert curve of a given order from its two Cartesian coordinates (encoding) and the inverse operation of obtaining the coordinates from the location code (decoding) have been designed. A Hilbert curve could be generated recursively, based on 'units' that change direction from one level to the next. To incorporate the directional changes, directional indices have been introduced and the resulting formulas have been incorporated into iterative algorithms.

It has been shown that only the directional indices require calculation by iteration, the calculation of the location code or the coordinates do not. This approach has improved the execution-time behaviour over previously published algorithms. As has been determined experimentally, the encoding algorithm is at least 10% faster, the decoding algorithm at least 80%.

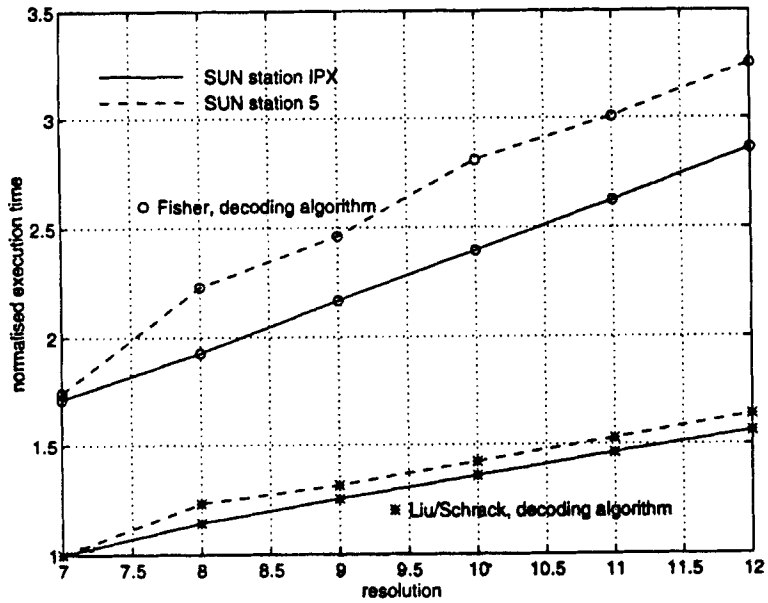

Figure 13. Comparison of decoding algorithms for the Hilbert order

REFERENCES

1. L. M. Goldschlager, 'Short algorithms for space-filling curves', *Software—Practice and Experience*, **11**, 99 (1981).
2. J. G. Griffiths, 'Table-driven algorithms for generating space-filling curves', *Computer-Aided Design*, **17**, 37–41 (1985).
3. N. Wirth, *Algorithm+Data Structures=Programs*, Prentice-Hall, Englewood Cliffs, New Jersey, 1976 (see also the second edition, 1986).
4. D. J. Abel and D. M. Mark, 'A comparative analysis of some two-dimensional orderings', *Int. J. GIS.*, **4**, 21–31 (1990).
5. C. Faloutsos and S. Roseman, 'Fractals for secondary key retrieval', *Proc. ACM Conf. on the Principles of Database Systems*, 247–252, (1989).
6. A. J. Fisher, 'A new algorithm for generating Hilbert curves', *Software—Practice and Experience*, **16**, 5–12 (1986).
7. G. Schrack, 'Encoding and decoding algorithms for linear quadtrees', *Tech. Rpt.*, Dept. of Electrical Engineering, The University of British Columbia, 1995.