Using State Diagrams for Hilbert Curve Mappings [⋆]

J K Lawder

School of Computer Science and Information Systems,
Birkbeck College, University of London,
Malet Street, London WC1E 7HX, United Kingdom
jkl@dcs.bbk.ac.uk

Abstract. The Hilbert Curve describes a method of mapping between one and n dimensions. Such mappings are of interest in a number of application domains including image processing and, more recently, in the indexing of multi-dimensional data. Relatively little work, however, has been devoted to techniques for mapping in more that 2 dimensions. This paper presents a technique for constructing state diagrams to facilitate mappings and is a specialization of an incomplete generic process described by Bially. Although the storage requirements for state diagrams increase exponentially with the number of dimensions, they are useful in up to about 9 dimensions.

1 Introduction

The Hilbert Curve [13] is a space-filling curve which can be used to define a one-one mapping between points in cartesian product space and points on a line. Thus the points are ordered. This paper describes how to construct state diagrams which facilitate a simple and practical way of determining the ordinal position of a point or the coordinates of a point with a given position. The use of state diagrams is practicable for spaces in up to about 10 dimensions, beyond which memory requirements become prohibitive.

We base our approach to contructing state diagrams on a technique developed by Bially [2]. Bially's technique is not specifically oriented towards the Hilbert Curve, or any other. It comprises an incomplete set of rules requiring a manual process of trial-and-error, which becomes increasingly arduous as the number of dimensions rises. Our adaptation, however, enables the automatic construction of state diagrams for Hilbert Curves in any number of dimensions, save for practical limitations imposed by their storage requirements.

An alternative approach avoiding the problem of storage is proposed by Butz [3] who calculates the coordinates of a point corresponding to an arbitrary ordinal position on the curve. (With a little effort, it is possible to derive the algorithm for the inverse of this mapping and a solution is given by Lawder [15, 16]). By 'arbitrary', we mean as distinct from generating points sequentially from

[⋆] Technical Report no. JL2/00, August 15, 2000, updated on August 23, 2000 their sequence numbers starting at sequence number 0. An ability to generate the coordinates of points sequentially is not adequate in some applications, such as multi-dimensional indexing.

It is much easier, however, to employ a table driven approach to performing the mappings than it is to calculate them. This also works for arbitrary points and in both directions of mapping. A simple approach is particularly advantageous in facilitating the design and implementation of more complicated algorithms, within which mappings or partial mappings are embedded. An example of this is the querying algorithms employed where multi-dimensional data is mapped to one dimension in the data storage application developed by Lawder [15].

A number of other techniques have been proposed in the literature for performing Hilbert Curve mappings. These have mostly been confined to the 2-dimensional case, often only generating the coordinates of points in the sequence that the Hilbert curve passes through them and do not always enable inverse mappings to be determined.

Recursive procedures which map one-dimensional values to 2-dimensional points sequentially for the purpose of drawing the curve but which neither provide mappings for arbitrary points nor provide inverse mappings are given by Wirth [20], Goldschlager [10], Cole [4] and Witten and Wyvill [21]. Table driven versions are given by Griffiths [11, 12].

An iterative table driven version which does enable the mapping of arbitrary one-dimensional values to 2-dimensional points is given by Fisher [9]. A table driven version which additionally facilitates the inverse mapping, from arbitrary 2-dimensional points to one-dimensional values, is given by Cole [5, 6].

A mathematical formula for mapping from one-dimensional values to 2-dimensional points is defined by Sagan [19], who also gives a BASIC program which implements it for the fourth order Hilbert curve. It appears that a different formula would be required for each number of dimensions through which the Hilbert curve passes.

Kamata et al [14] apply the Hilbert curve to the analysis of images. They refer to a technique for sequentially mapping one-dimensional values to coordinates of n-dimensional points which appears in Japanese in an earlier publication by the same authors. Where arbitrary mappings are required they utilize the method described by Butz.

Liu and Schrack [17] provide formulae for mapping from coordinates of 2-dimensional points to one-dimensional values and the inverse which they found in experimentation to be significantly more computationally efficient than the algorithms given by Fisher in [9].

The remainder of this paper is divided into three sections and a conclusion. In section 2 we describe the concept of the Hilbert Curve. In section 3 we give an example of a state diagram , together with the algorithm which uses it in mapping calculations. Section 4 details the state diagram construction process for the Hilbert Curve.

2 Hilbert's Space-filling Curve

An understanding of the way in which a Hilbert Curve is drawn is rapidly gained from Fig. 1 showing the first 3 steps of an infinite process for the 2-dimensional case. A square is initially divided into 4 sub-squares which are then ordered such that any pair of consecutive sub-squares share a common edge. The ordering is illustrated by drawing a line through their centre-points and this line is called a *first-order* curve. Figure 1(b) shows the next step in which each sub-square created in the first step is then divided into 4 sub-squares. The sub-squares within the first and last squares of the first step are ordered differently to ensure the adjacency property is preserved. Different orderings of sets of 4 sub-squares give rise to different orientations of first order curves and are regarded as being distinct states in a state diagram.

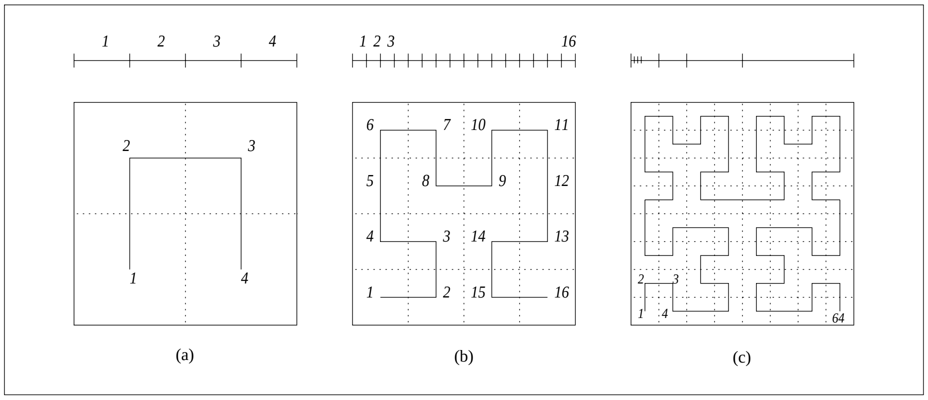


Fig. 1. Approximations of the Hilbert Curve in 2 Dimensions

In practical applications, the process can be terminated after k steps to produce an *approximation* of a space-filling curve of order k. This curve passes through 2^{kn} sub-squares, the centre-points of which are regarded as points in a space of finite granularity. The transformation of a curve of order $k-1$ to a curve of order k can be viewed as replacing each point on the former with a first order curve.

The concept of the Hilbert Curve can be generalized into higher numbers of dimensions and an example of a second order curve in 3 dimensions is given in Figure 2. We see in both the 2 and 3 dimensional cases that the Hilbert Curve manifests an interesting and useful property in which consecutively ordered points are always adjacent to each other in space.

In the remainder of this paper, we refer to the sequence number of a point on a Hilbert Curve as a *derived-key*.

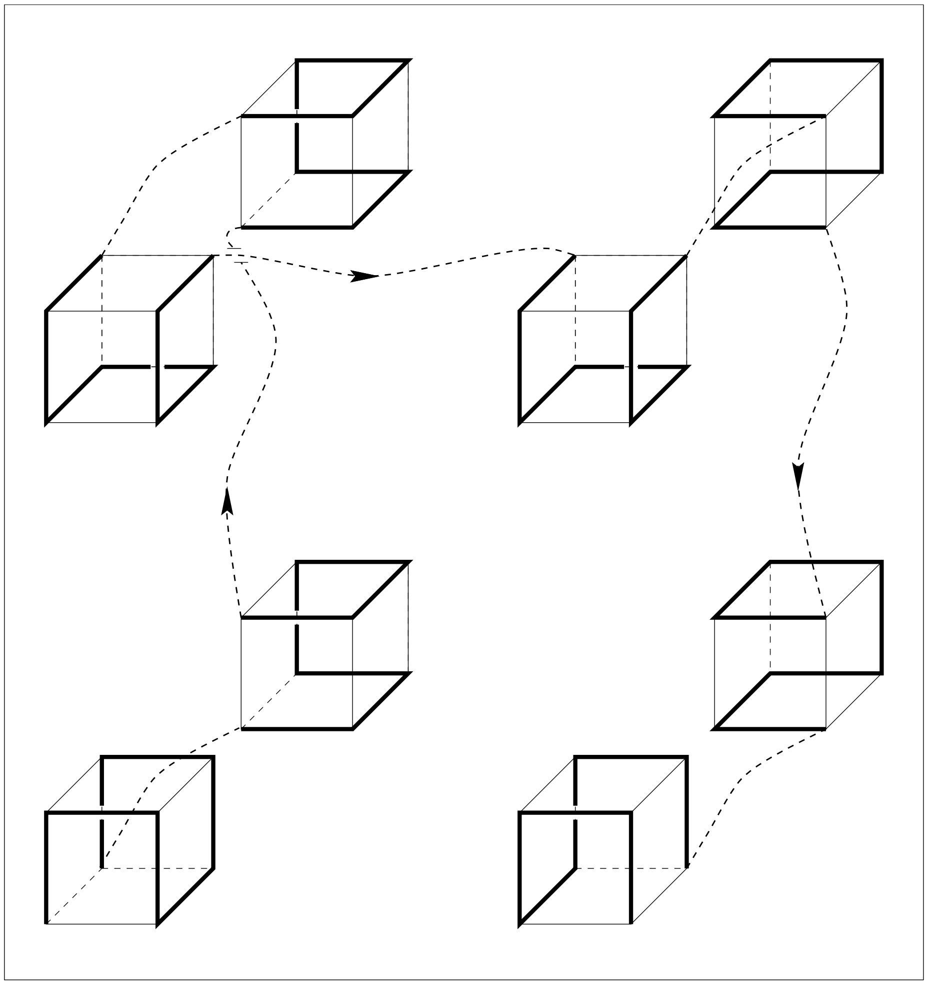


Fig. 2. Approximations of the Hilbert Curve in 3 Dimensions

state	*derived-key*							
no.	000	001	010	011	100	101	110	111
0	000	001	011	010	110	111	101	100
	1	2	2	3	3	5	5	4
1	000	010	110	100	101	111	011	001
	2	0	0	8	8	7	7	6
2	000	100	101	001	011	111	110	010
	0	1	1	9	9	11	11	10
3	011	010	000	001	101	100	110	111
	11	6	6	0	0	9	9	8
4	101	111	011	001	000	010	110	100
	9	7	7	11	11	0	0	5
5	110	010	011	111	101	001	000	100
	10	8	8	6	6	4	4	0
6	011	111	110	010	000	100	101	001
	3	11	11	5	5	1	1	7
7	101	100	110	111	011	010	000	001
	4	9	9	10	10	6	6	1
8	110	100	000	010	011	001	101	111
	5	10	10	1	1	3	3	9
9	101	001	000	100	110	010	011	111
	7	4	4	2	2	8	8	3
10	110	111	101	100	000	001	011	010
	8	5	5	7	7	2	2	11
11	011	001	101	111	110	100	000	010
	6	3	3	4	4	10	10	2

Table 1. Hilbert Curve State Diagram: for Mapping from One Dimension (*derived-keys*) to 3-dimensional Points

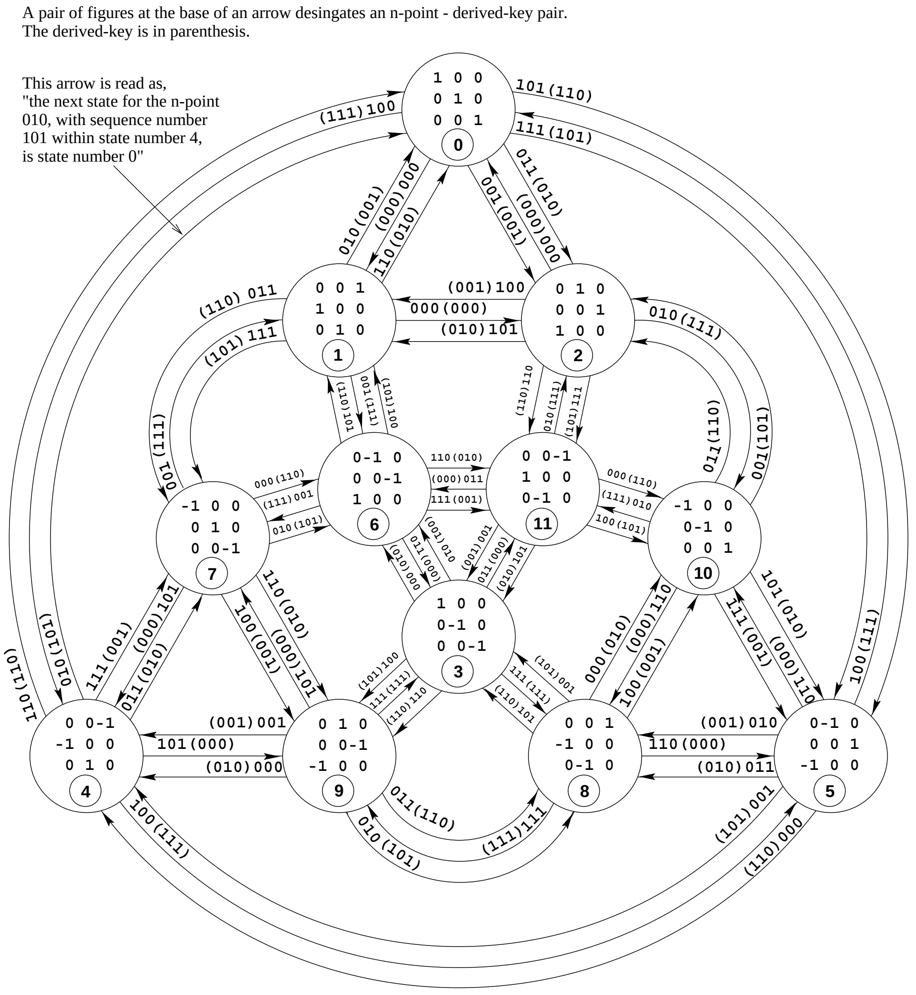
3 An Algorithm Using State Diagrams

In what follows, we use the term *n-point* for the concatenation of the (single bit) coordinates of a point on the first order curve.

An example of a graphical representation of a state diagram for the Hilbert Curve in 3 dimensions is given in Figure 3. This state diagram is then expressed in tablular form in Tables 1 and 2.

Table 1 contains a row for each state and is used in the mapping from one to n dimensions. Within a row, points, expressed as n-points, are ordered by sequence number, or derived-key value. Associated with each n-point is a (decimal) 'next-state' number designationg which state it transforms to in a curve of the next higher order. Table 2 is similar but is used for the inverse mapping and so derived-keys are ordered by their corresponding n-point values in each state. These tables may be stored as arrays in a software implementation of the mapping algorithms.

The manner in which the derived-key, D, of a point, P, is calculated using a state diagram is given in Algorithm 1, where the most significant bit of a


A pair of figures at the base of an arrow desingates an n-point - derived-key pair.
The derived-key is in parenthesis.
This arrow is read as,
"the next state for the n-point
010, with sequence number
101 within state number 4,
is state number 0"

from state	(Y)	X_1	to state
0	011	010	3
	100	110	
1	011	100	8
	100	101	
2	011	001	9
	100	011	
3	011	001	0
	100	101	
4	011	001	11
	100	000	
5	011	111	6
	100	101	
6	011	010	5
	100	000	
7	011	111	10
	100	011	
8	011	010	1
	100	011	
9	011	100	2
	100	110	
10	011	100	7
	100	000	
11	011	111	4
	100	110	

Fig. 3. A State Diagram for the Hilbert Curve in 3 Dimensions with supplementary table listing links not shown (for clarity) in the illustration

state	*n-point*							
no.	000	001	010	011	100	101	110	111
0	000	001	011	010	111	110	100	101
	1	2	3	2	4	5	3	5
1	000	111	001	110	011	100	010	101
	2	6	0	7	8	8	0	7
2	000	011	111	100	001	010	110	101
	0	9	10	9	1	1	11	11
3	010	011	001	000	101	100	110	111
	6	0	6	11	9	0	9	8
4	100	011	101	010	111	000	110	001
	11	11	0	7	5	9	0	7
5	110	101	001	010	111	100	000	011
	4	4	8	8	0	6	10	6
6	100	111	011	000	101	110	010	001
	5	7	5	3	1	1	11	11
7	110	111	101	100	001	000	010	011
	6	1	6	10	9	4	9	10
8	010	101	011	100	001	110	000	111
	10	3	1	1	10	3	5	9
9	010	001	101	110	011	000	100	111
	4	4	8	8	2	7	2	3
10	100	101	111	110	011	010	000	001
	7	2	11	2	7	5	8	5
11	110	001	111	000	101	010	100	011
	10	3	2	6	10	3	4	4

Table 2. Hilbert Curve State Diagram: for Mapping from 3-dimensional Points to One Dimension (*derived-keys*)

value is designated 'position 1'. The inverse mapping, from a derived-key to the coordinates of a point, is carried out in a similar manner.

4 Constructing a State Diagram for the Hilbert Curve

Bially's method for creating state diagrams is carried out in 2 stages. The first stage entails following a set of rules which results in the production of a *state diagram generator table*. This table principally describes how a particular first order curve transforms into a second order curve. The table is then used as a tool for creating the state diagram itself.

An example of a generator table for the Hilbert Curve in 3 dimensions is given in Table 3.

In this section we specialize and extend Bially's rules to enable the construction of generator tables for Hilbert Curves and then give an algorithm which uses the table to construct the state diagram itself.

Algorithm 1 Finding the *derived-key* of a Point using the State Diagram

```
1:  $current\_level \Leftarrow 1$
2:  $current\_state \Leftarrow 0$
3:  $D \Leftarrow$ the empty bit-string
4:  repeat
5:    $p \Leftarrow$ one bit in position $current\_level$ taken from each coordinate in $P$, concatenated into an $n$-point
6:    $d \Leftarrow$ the $n$-bit derived-key taken from the $current\_state$ corresponding to $p$
7:    append $d$ to $D$
8:    if $current\_level <$ the order of the curve then
9:      $current\_state \Leftarrow$ the next state for the ordered pair $\langle p, d \rangle$ within the $current\_state$
10:   end if
11:   $current\_level \Leftarrow current\_level + 1$
12: until $current\_level >$ the order of the curve
```

Our rules for generating state diagrams enable us to perform a particular mapping for each value of n. We believe that these mappings describe valid Hilbert curves and have satisfied ourselves empirically that they produce correct results. They constitute our definitions of the Hilbert curves used throughout this paper. However, we leave a formal proof of the correctness of our techniques as a matter for future work.

4.1 Method of Constructing the Generator Table

A generator table contains 2^n rows, one for each point on a first order curve, and corresponds to a particular state in the diagram; we label it 'state 0' or S_0. The columns are labeled Y, X_1, X_2, δY and $\overline{T(Y)}$. In describing the rules for populating the table, we provide some brief comments on the semantics of the various columns, as these are generally absent from Bially's paper.

We noted above that Bially's procedure is generic and therefore is applicable where the coordinates and derived-keys are expressed as integers of any radix. An important characteristic of our specialization of the rules for the Hilbert Curve is that we express integers in a binary radix. Thus all numbers in the first three columns are composed of the digits 0 and 1 whereas in Bially's original paper they are composed of digits in the range $[0, \ldots, r-1]$, where r is the radix used. We do not refer to rules which are followed automatically as a consequence of using a binary radix.

Column Y Contains the sequence numbers of points lying on a first order curve arranged in ascending order. For the Hilbert Curve they lie in the range $[0, \ldots, 2^n - 1]$.

Column X_1 Contains sets of coordinates of points on a first order curve expressed as n-points. Points in successive rows must be adjacent to each other in space and so successive n-points differ in one bit only.

Y	X_1	X_2	δY	$T(Y)$		
000	000	000	001	0	0	1
		001		1	0	0
				0	1	0
001	001	000	010	0	1	0
		010		0	0	1
				1	0	0
010	011	000	010	0	1	0
		010		0	0	1
				1	0	0
011	010	011	100	1	0	0
		111		0	-1	0
				0	0	-1
100	110	011	100	1	0	0
		111		0	-1	0
				0	0	-1
101	111	110	010	0	-1	0
		100		0	0	-1
				1	0	0
110	101	110	010	0	-1	0
		100		0	0	-1
				1	0	0
111	100	101	001	0	0	-1
		100		1	0	0
				0	-1	0

Table 3. State Diagram Generator Table for the Hilbert Curve in 3 Dimensions

For the Hilbert Curve, this requirement is satisfied by setting column X_1 values equal to the 'Gray-codes' of their corresponding column Y values and so each row's value is simply calculated as $Y \oplus \ Y/2$. The Gray-code sequence is discussed by Reingold et al in [18]. The usefulness of Gray-codes as column X_1 values was noted by Faloutsos in his technical report [8].

We note that the first column X_1 value equals $0\ldots0$ and the last equals $10\ldots0$ and that these two points are adjacent since they differ in one bit only.

Column X_2 Each row contains a pair of n-points corresponding to the first and last points on the first order curve to which the point in column X_1 transforms in the construction of a second order curve.

The first value in the first row equals the first column X_1 entry and the second value in the last row equals the last column X_1 entry. Two values in the same row differ in one bit and adjacent values in different rows differ in one bit only; the same bit in which their corresponding entries in column X_1 differ, but in the opposite sence.

In 2 dimensions, the following manually generated sequence, for example, satisfies these constraints:

$$[\,00,\,01,\,00,\,10,\,00,\,10,\,11,\,10\,]$$

as does the following sequence in 3 dimensions:

$$[000,\,001,\,000,\,010,\,000,\,010,\,011,\,111,\,011,\,111,\,110,\,100,\,110,\,100,\,101,\,100].$$

Generating these sequences manually becomes increasingly difficult, however, as the number of dimensions increases but a pattern emerges enabling them to be expressed algorithmically, for any value of n.

We derive a column X_2 sequence for a curve of n dimensions from the sequence for $n-1$ dimensions, initializing the sequence for one dimension as $[\,0,\,1,\,0,\,1\,]$. The sequence for n dimensions is then determined as follows;

1. Initialize the sequence for n dimensions equal to the sequence for $n-1$ dimensions.
2. Set the value of the last member of the sequence (for $n-1$ dimensions) equal to the value of the penultimate member.
3. Prefix the last member of the sequence with a bit of value 1 and all other members each with a bit of value 0.
4. Double the size of the sequence by reflecting it such that the last member equals the first and so on.
5. Invert the values of the most significant bits of all of the members in the upper half of the sequence.

The rules ensure that the last point of a first order curve within a second order curve is adjacent to the first point of the next first order curve.

Column δY This column is implied only in Bially's paper and his rules need no particular adaptation to suit the Hilbert Curve. Each row contains a number in which each digit is the magnitude of the difference between the corresponding digits in the same row's entries in column X_2. For the Hilbert Curve only one bit in any δY value is non-zero.

The entry in column δY for any row indicates in which dimension the start and end points of the first order curve at the second order level have different coordinate values. A corresponding value for the entries in the first and last rows in column X_1 would be equal to the last entry in column X_1, since the first entry contains zero-valued bits only.

Column $\overline{T(Y)}$ Contains transformation matrices which define how the first order curves which are partially defined in column X_2 differ from the first order curve defined by columns Y and X_1, ie state S_0.

A matrix is initially a permutation matrix which, applied to its column δY value produces the last column X_1 value. This is ambiguous but resolved simply for the Hilbert Curve as follows;

1. Set the first row of a matrix equal to the matrix's column δY value.
2. for each of the remaining rows, set row i equal to the value in row $i-1$ circular right-shifted one bit position.

Permutation matrices for all curves are then adapted to become transformation matrices as follows; if the *i-th* digit of the first of its row's pair of X_2 entries is non-zero then the non-zero element of the *i-th* column within the matrix is set to -1.

Generally, a transformation matrix implies some state S_i, $S_i \neq S_0$, by enabling any point P_i in state S_i to be transformed into the equivalent point P_0 in state S_0 which has the same distance from the beginning of the curve in state S_0 as does point P_i in state S_i. The distance, ie sequence number, of point P_i from the start of the curve in state S_i is then determined by looking up the column Y value for point P_0 in state S_0.

It follows that applying a row's matrix to its first entry in column X_2 should transform it to the first entry in column X_1 and applying it to its second entry should transform it to the last entry in column X_1.

4.2 Using the Generator Table to Construct a State Diagram

The method by which a state diagram generator table is used to produce a state diagram is not detailed in Bially's paper although it is addressed from a mathematical perspective in his thesis [1]. In this section we present an algorithm for state diagram generation in terms which more readily enable it to be implemented as a computer program. Our algorithm applies specifically where the generator table has been constructed in the manner described in the previous section.

The generator table is used to produce a temporary list of all of the states which together define a state diagram. Once created, not all of the information within it is needed in the final state diagram. The list is therefore traversed to extract relevant information only.

The first state placed in the temporary list is the state encapsulated by the generator table itself; by the mapping defined by columns Y and X_1 and by the *next-states* for each point as defined by transformation matrices in column $\overline{T(Y)}$.

There is no relationship between consecutive states within the list of states. They are simply placed in it in the order that they are encountered in the calculation process.

A member of the temporary list defines a state and is a record containing the following information:

1. A State Number identifying the state.
2. A set of 2^n triples, one for each point on a first order curve. Each triple is of the form: $\langle Y_u^i,\ X1_u^i,\ tm_u^i \rangle$. Y_u^i is a sequence number, in the range $[0,\ldots,2^n-1]$, of a point in state number u, ie S_u. $X1_u^i$ is the n-point representation of the coordinates of the point, in the same range. tm_u^i is the number of the *next-state*, ie first order curve, to which the point transforms

to in a second order curve. Each distinct tm_u^i number corresponds to a transformation matrix, examples of which are found in column $\overline{T(Y)}$ of the generator table, defining a distinct state.
3. A Transformation Matrix encapsulating how the state differs from the first state in the list.
4. A pointer to the next member in the list.

The procedure for building the temporary list of states is given in Algorithm 2.

A state diagram, derived from the temporary list of states, can be implemented as an array of states with one element for each state in the list. A state is identified by its number, which can be implied by its position within the array of states, and defined by its list of triples. Each list of triples can also be stored as an array. The elements of these arrays may be expressed compactly as pairs since one of the attributes, Y_u^i or $X1_u^i$, of a triple can be implied by its position in the array, depending on how the triples are sorted.

Thus two state diagrams can be produced from the state list. One is required for performing a mapping from one dimension to n dimensions. The triples within it are sorted by Y_u^i values, which may be implied.. The other is required for performing a mapping from n dimensions to one dimension. The triples are sorted by $X1_u^i$ values, which may be implied.

Transformation Matrix Operations

Two different transformation matrix calculations are employed in Algorithm 2; in line numbers 24 and 27. We conclude this section with a description of why and how these calculations are performed.

We note from the previous section that transformation matrices in the generator table (column $\overline{T(Y)}$) can be encapsulated by two values, namely the first of a row's pair of column X_2 values and the same row's column δY value. This concept also applies to transformation matrices which do not appear in the generator table and this simplifies calculations using matrices. Thus a transformation matrix for state S_u is represented by a pair; $\langle X2_u, \delta Y_u \rangle$.

We also note that a δY value for some state S_u encapsulates how much right circular shifting needs to be applied to each row in the identity matrix to create the matrix for state S_u. Thus δY_u can be stored as an integer representing the right hand side operand of the shift operation. For example, $10\ldots0$ is stored as 0, $010\ldots0$ is stored as 1, $0010\ldots0$ is stored as 2, and so on.

The calculation in line number 24 finds the n-point j (equal to $X1_0^p$) in state S_0 which has the same sequence number (ie p) as the n-point i in state S_u. Once p is found, i assigned to $X1_u^p$.

The calculation of j is carried out in the following manner:

1. $j \Leftarrow i \oplus X2_u$
2. $j \Leftarrow j$ *left* circular shift δY_u bits

The calculation in line number 27 determines the pair $\langle X2_w, \delta Y_w \rangle$ encapsulating the transformation matrix for state S_w, which is the next-state for $X1_u^p$

Algorithm 2 Algorithm to Create a List of States

```
{Using the generator table, initialize the first member, ie state, of the list}
1: current_state ⇐ 0
2: next_state_num ⇐ 1
3: for all $i$ such that $0 \leq i < 2^n$ do
4:   $Y_0^i \Leftarrow i$
5:   $X1_0^i \Leftarrow X_1$ value from row $i$ of the generator table
6: end for
{The $tm_0^i$ attributes of the triples are initially undefined}
7: current_state transformation matrix ⇐ the identity matrix
{Initialize the $tm_0^i$ attributes, ie next-state numbers, of the triples in state 0}
8: for all $i$ such that $0 \leq i < 2^n$ do
9:   if no state exists in the state list whose transformation matrix equals that found
     in row $i$ of the generator table then
10:    append a new state to the list
11:    new state's number ⇐ next_state_num
12:    next_state_num ⇐ next_state_num + 1
13:    new state's transformation matrix ⇐ matrix from row $i$ of the generator table
14:    $tm_0^i \Leftarrow$ new state's number
15:  else
16:    $tm_0^i \Leftarrow$ the number of the state found in the list
17:  end if
18: end for
{State 0 is now fully defined. A new state has been added to the list for each distinct
transformation matrix, other than the identity matrix, found in column $\overline{T(Y)}$ of
the generator table}
{Initialize the attributes of the triples in all of the states remaining in the list,
appending any new states required in the process}
19: for all states, $u$, in the list (excluding state 0) do
20:   for all $i$ such that $0 \leq i < 2^n$ do
21:     $Y_u^i \Leftarrow i$
22:   end for
23:   for all $i$ such that $0 \leq i < 2^n$ do
24:     $j \Leftarrow i *$ the transformation matrix for state $u$
25:     $p \Leftarrow$ the row in state 0 such that $X1_0^p = j$
26:     $X1_u^p <= i$ {ie assign $i$ to the $X1$ value in row $p$ of the current state $u$}
27:     $TM \Leftarrow$ (transformation matrix corresponding to $tm_0^p$) $*$ (transformation ma-
        trix corresponding to state $u$)
        {NB $TM$ is a transformation matrix defining a state, not a state number}
28:     if no state exists in the state list whose transformation matrix equals $TM$
        then
29:       Add a new state to the list
30:       State Number of the new state ⇐ next unused number
31:       Transformation matrix of the new state ⇐ $TM$
32:     end if
33:     $tm_u^p \Leftarrow$ the State Number of the state whose transformation matrix equals
        $TM$
34:   end for
35: end for
```

in state S_u. S_q, encapsulated by the pair $\langle\, X2_q,\, \delta Y_q\, \rangle$, is the next-state corresponding to tm_0^p. The calculation is carried out as follows:

1. $\delta Y_w \Leftarrow (\delta Y_q + \delta Y_u) \bmod n$
2. $temp \Leftarrow X2_q$ right circular shift δY_u
3. $X2_w \Leftarrow temp$ exculsive-or $X2_u$

5 Conclusions

The existence of state diagrams for the Hilbert Curve facilitates a convenient and simple means of mapping between one and n dimensions in which proximity between points in one dimensions are preserved in n dimensions and to some degree the inverse is also true.

The storage requirements for state diagrams increase exponentially with the number of dimensions, both in terms of the number of states and in terms of the size of a single state. Our rules result in a number of states given by

$$n2^{n-1}$$

According to Bially, this is the minimum possible. The size of a single state is determined by the number of *derived-key* – next-state pairs in Table 1, for example, and is given by the expression 2^n.

The complexity of the mapping algorithm given in Algorithm 1 can be seen to be $O(kn)$ and this is the same as for the algorithm given by Butz which relies on calculation alone, although the latter includes a higher constant factor. In experiments, we found that on currently available hardware, mappings can be performed more quickly using state diagrams although their advantage diminishes with increasing values of n.

The mapping technique described by Butz is similar to the state diagram approach at the fundamental level and insights gained in developing rules for state diagram generation enabled us to introduce some useful improvements to the detail of Butz' algorithm. These are described in [15, 16].

A further benefit of the state diagram approach is that variations to the detail of the rules governing the generator tables may be found and the resulting state diagrams therefore define slightly different interpretations of the Hilbert Curve. Exploration of this is largely a matter for further research.

References

1. Theodore Bially. *A Class of Dimension Changing Mappings and its Application to Bandwidth Compression.* PhD thesis, Polytechnic Institute of Brooklyn, 1967.
2. Theodore Bially. Space-filling curves: Their generation and their application to bandwidth reduction. *IEEE Transactions on Information Theory*, IT-15(6):658–664, Nov 1969.
3. Arthur R. Butz. Alternative algorithm for hilbert's space-filling curve. *IEEE Transactions on Computers*, 20:424–426, April 1971.

4. A. J. Cole. A note on space filling curves. *Software – Practice and Experience*, 13(12):1181–1189, December 1983.
5. A.J. Cole. Direct transformations between sets of integers and hilbert polygons. *International Journal of Computer Mathematics*, 20:115–122, 1986.
6. A.J. Cole. Compaction techniques for raster scan graphics using space-filling curves. *The Computer Journal*, 30(1):87–92, 1987.
7. Christos Faloutsos and Shari Roseman. Fractals for secondary key retrieval. In *Proceedings of the Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, March 29-31, 1989, Philadelphia, Pennsylvania*, pages 247–252. ACM Press, 1989.
8. Christos Faloutsos and Shari Roseman. Fractals for secondary key retrieval (an extended version of [7]). Technical Report UMIACS-TR-89-47, University of Maryland, 1989. http://www.cs.cmu.edu/ christos/cpub.html.
9. A. J. Fisher. A new algorithm for generating Hilbert curves. *Software – Practice and Experience*, 16(1):5–12, January 1986.
10. Leslie M. Goldschlager. Short algorithms for space-filling curves. *Software – Practice and Experience*, 11(1):99, January 1981. Short Communication.
11. J.G. Griffiths. Table-driven algorithms for generating space-filling curves. *Computer Aided Design*, 17(1):37–41, Jan/Feb 1985.
12. J.G. Griffiths. An algorithm for displaying a class of space-filling curves. *Software Practice and Experience*, 16(5):403–411, May 1986.
13. David Hilbert. Ueber stetige abbildung einer linie auf ein flachenstuck. *Mathematische Annalen*, 38:459–460, 1891.
14. S. Kamata, E. Kawaguchi, and M. Niimi. An interactive analysis method for multi-dimensional images using a hilbert curve. *Systems and Computers in Japan*, 26(3):83–92, 1995.
15. Jonathan Lawder. *The Application of Space-Filling Curves to the Storage and Retrieval of Multi-dimensional Data*. PhD thesis, Birkbeck College, University of London, 2000.
16. Jonathan Lawder. Calculation of mappings between one and n-dimensional values using the hilbert space-filling curve. Technical Report JL1/00, Birkbeck College, University of London, 2000.
17. Xian Liu and Günther Schrack. Encoding and decoding the Hilbert order. *Software – Practice and Experience*, 26(12):1335–1346, December 1996.
18. E.M. Reingold, J. Nievergelt, and N. Deo. *Combinatorial Algorithms: Theory and Practice*. Prentice Hall, 1977.
19. Hans Sagan. *Space-Filling Curves*. Springer-Verlag, 1994.
20. N. Wirth. *Algorithms + data structures = programs*. Prentice Hall, 1976.
21. Ian H. Witten and Brian Wyvill. On the generation and use of space-filling curves. *Software – Practice and Experience*, 13(6):519–525, June 1983.