Querying Multi-dimensional Data Indexed Using the Hilbert Space-Filling Curve*

J. K. Lawder
Birkbeck College, University of London
jkl@dcs.bbk.ac.uk

P. J. H. King
Birkbeck College, University of London
pjhk@dcs.bbk.ac.uk

Abstract

Mapping to one-dimensional values and then using a one-dimensional indexing method has been proposed as a way of indexing multi-dimensional data. Most previous related work uses the Z-Order Curve but more recently the Hilbert Curve has been considered since it has superior clustering properties. Any approach, however, can only be of practical value if there are effective methods for executing range and partial match queries. This paper describes such a method for the Hilbert Curve.

1 Introduction

Indexing of multi-dimensional data has been the focus of a considerable amount of research effort over many years but no generally agreed paradigm has emerged to compare with the impact of the B-Tree, for example, on the indexing of one-dimensional data. An extensive review appears in [5]. At the same time, the need for efficient methods is ever more important in an environment where databases become larger and more complex in their structures and aspirations for extracting valuable information become more sophisticated.

Mapping multi-dimensional data to one dimension, enabling simple and well-understood one-dimensional access methods to be exploited, has been suggested as a solution in the literature, for example by Faloutsos [3, 4]. One way of effecting such a mapping is to utilize space-filling curves which pass through every point in a space once so giving a one-one correspondence between the coordinates of the points and the one-dimensional sequence numbers of the points on the curve.

For the most part, interest in space-filling curves has been confined to the Z-Order Curve, for example in the work of Orenstein and Manola [15] and more recently of Ramsak et al [16]. The possibility of using other curves, such as the Hilbert Curve, has also been suggested but most previous work has been of a theoretical nature [7, 8, 14]. These studies show that the Hilbert Curve manifests superior data clustering properties when compared with other curves.

In order for the application of the Hilbert Curve in the indexing of multi-dimensional data to be viable, however, the existence of a technique for querying data is essential. Techniques which work for the Z-Order Curve, for example that of Tropf and Herzog [17], cannot simply be applied to the Hilbert Curve. This paper reports on a technique which has successfully been developed as part of the design and implementation of the first fully functioning data storage and retrieval application utilizing the Hilbert Curve by Lawder [10].

In sections 2, 3 and 4 we briefly describe the Hilbert Curve, how mappings between one and n dimensions are calculated and how we utilize the curve in a data storage application. In section 5 we describe our strategy for querying data and its implementation.

2 The Hilbert Curve

Space-filling curves were a topic of interest for leading pure mathematicians in the late 19th century and the first graphical representation of one was given by David Hilbert in 1891 [6].

An understanding of the way in which a Hilbert Curve is drawn is gained from Figs. 1–3 showing the first 3 steps of an infinite process for the 2-dimensional case. In Fig. 1(a) a square is initially divided into 4 quadrants and a *first-order curve* is drawn through their centre points. The quadrants are ordered such that any two which are adjacent in the ordering share a common edge. In the next step, shown in Fig. 1(b), each of the quadrants of Fig. 1(a) is divided into 4 and, in all, 4 'scaled-down' first order curves are drawn and connected together. Note that the first and last first order curves have different orientations to the one drawn in the first step so that the adjacency property of consecutive squares is maintained. Fig. 3 shows the third step.

In practical applications the process is terminated after k steps to produce an *approximation* of a space-filling curve of order k. This passes through 2^{2k} quadrants, the centre points of which are regarded as points in a space of finite granularity. Generalizing the concept into n dimensions, squares and quadrants

* Technical Report no. JL3/00, September 2000

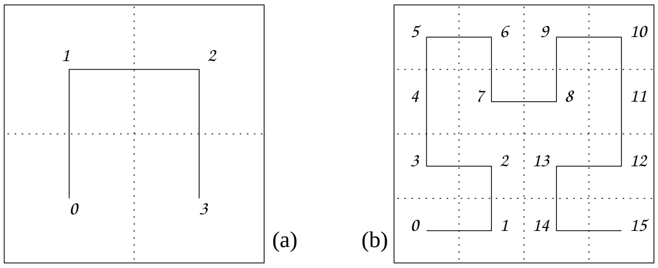


Figure 1: Approximations of the Hilbert Curve

are replaced by hyper-rectangles, successive hyper-rectangles share common hyper-faces and a curve passes through 2^{nk} points. In the working software developed by Lawder [10], 32nd order curves are used, enabling 32-bit coordinate values, and n takes a value of up to 16, but this could quite easily be increased.

An important property of the Hilbert Curve is that consecutively ordered points are adjacent in space.

3 Hilbert Curve Mapping

The recursive way in which space is partitioned during the Hilbert Curve construction process can be expressed in a tree structure as in Fig. 2. This conceptual view aids an understanding of the mapping process. Each node corresponds to a first order curve and a collection of nodes at any tree level, k, describes a curve of order k, where the root resides at level 1. Thus the root node corresponds to the first order curve of Fig. 1(a) and the leaf nodes correspond to the set of first order curves comprising the third order curve of Fig. 3.

Alternatively, a node is viewed as a sub-space enclosing 2^n nested sub-spaces, except that a leaf node encloses 2^n points.

We call a binary sequence number of a quadrant within a node (equivalent to a point on a first order curve) a *derived-key*, or a *quadrant number*, and the concatenated (single-bit) coordinates of a point on a first order curve an *n-point*. We also call the sequence number of a point on a curve of any order k a derived-key. This value contains nk bits; the same as the sum of the bits in all of the coordinates of a point.

We illustrate how the mapping from the coordinates of a point to its derived-key takes place with the example of point **P** shown on Fig. 3. Its coordinates are $\langle 110, 100 \rangle$. Initially the derived-key of **P** is unknown and designated by the bit string '??????'.

Step 1: Concatenate the top bits of the coordinates of **P** to form the n-point 11. This locates **P** in quadrant number 10 in the tree's root node. This quadrant is emphasized in bold in Fig. 2. The derived-key of **P** is now '10????'.

Step 2: Concatenate the next lower (ie middle) bits of

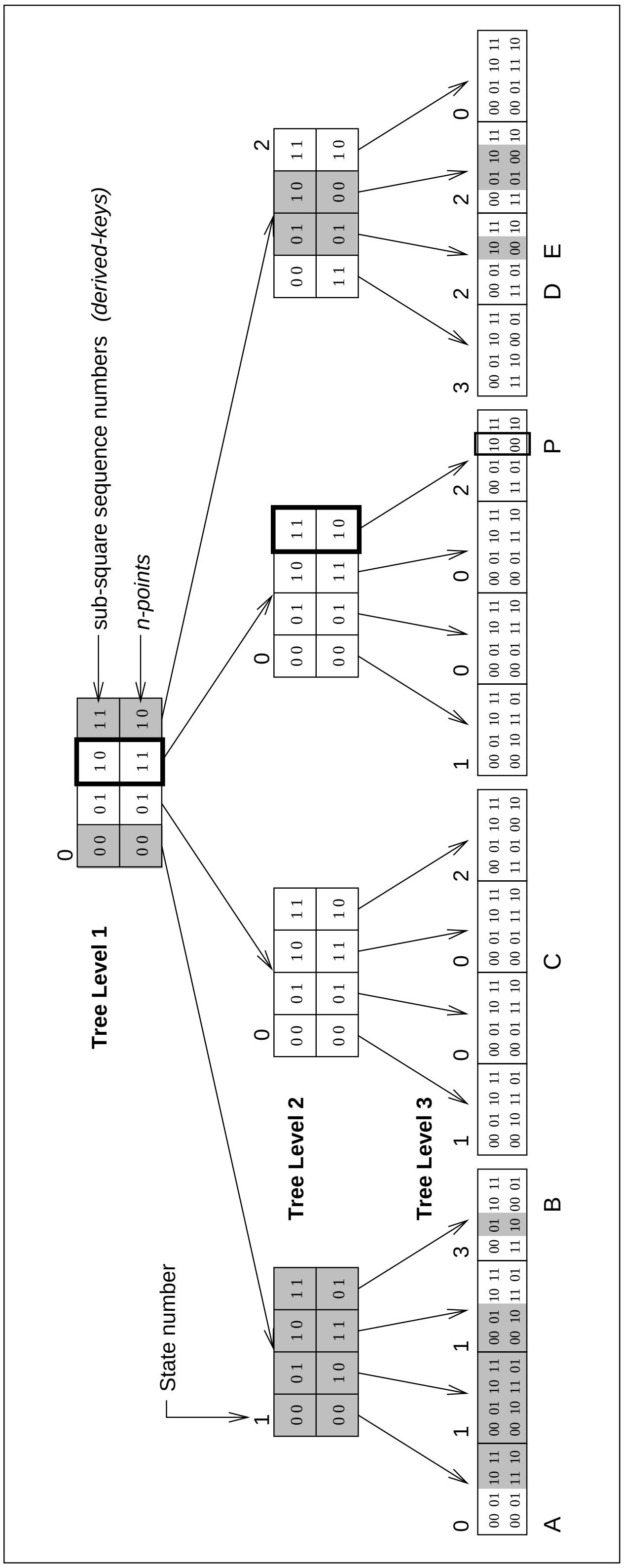


Figure 2: A Tree Representation of the Third Order Hilbert Curve in 2 Dimensions

the coordinates of **P** to form the n-point 10. Locate this n-point within the tree level 2 node pointed to by the quadrant identified in the root, in step 1. This places the point in the quadrant number is 11. The derived-key of **P** is now '1011??'.

Step 3: Concatenate the bottom bits of the occordinates of **P** to form the n-point 00. Locate this n-point within the tree level 3 node pointed to by the quadrant identified in the previous step. This places the point in the quadrant number 10. The derived-key of **P** is now '101110'.

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

It is not practicable to store the tree representation of the Hilbert Curve in memory for calculating mappings for any useful value of order of curve. We note, however, that a tree contains a finite number of types of node. Different types of node (equivalent to different orientations of first order curve) can be regarded as being different states enabling the tree structure to be expressed more compactly as a state diagram.

A method for constructing Hilbert Curve state diagrams is described by Lawder [13] who adapts and extends a generic technique originally proposed by Bially [1]. In higher than about 9 dimensions, state diagrams also become too large to accommodate in memory and mappings need to be calculated instead, for example in the manner detailed by Butz [2] and developed by Lawder [12].

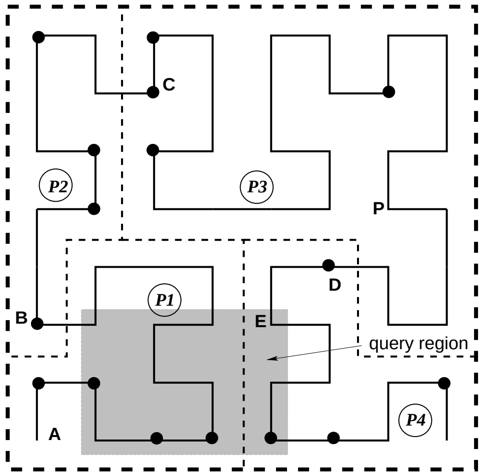
4 Application of the Hilbert Curve

We now describe how the Hilbert Curve is used in a practical application. We refer to actual records placed in a data file as *datum-points*.

Conceptually, the curve is cut into contiguous sections each corresponding to a page of storage in the data file. Each page has a fixed data capacity but the length of a curve section varies according to the local density of datum-points; thus each section or page holds roughly the same number of datum-points. Pages are indexed by derived-keys of datum-points, which we call *page-keys*. Generally, a page's page-key is the lowest derived-key of any datum-point lying on the page. The first page is an exception and is indexed by the derived-key of value zero whether or not it corresponds to a datum-point. An example is shown in Fig. 3 where the page capacity is 4 datum-points.

Pages are thus logically ordered by their page-keys which are placed in a one-dimensional index together with their corresponding page addresses. No page contains a datum-point whose derived-key is greater than that of any datum-point on a successor page.

We note that the approach described here partitions data rather than the space in which it lies. It adapts



A, B, C & D are 'page-keys', E is the 'next-match' to B for the query

Figure 3: Partitioning into Pages

easily on update, ie insertion and deletion of datum-points. For example, if a page becomes full, half of the datum-points whose derived-keys are greater or less than those of the other half can be moved to a new page or some other similarly defined portion may be moved to a logically adjacent page. In other words, precisely the same approach used in the one-dimensional B-Tree can be used for multi-dimensional data.

5 Query Execution

5.1 The Querying Strategy

Our application facilitates the retrieval of datum-points from hyper-rectangular query regions defined by lower and upper bound coordinates, $\langle l_1, l_2, \ldots l_n \rangle$ and $\langle u_1, u_2, \ldots u_n \rangle$ with $min_i \leq l_i \leq u_i \leq max_i$. A query region intersects one or more curve sections each corresponding to a page. The Hilbert Curve may enter, leave and re-enter a query region a number of times. Our strategy identifies for retrieval and searching only those pages which intersect the query region, e.g. given the query region shown shaded in Fig. 3, only pages $P1$ and $P4$ are identified. Intervening curve sections corresponding to pages $P2$ and $P3$ are effectively 'stepped over'.

Pages to be searched are identified lazily, in ascending page-key order. This is effected by a function called *calculate_next_match()*. The first time it is called the lowest derived-key of any point lying within the query region is identified. The index is searched and the page which contains this point, if it is a datum-point, is identified, retrieved and searched for datum-points lying within the query region.

The second time that *calculate_next_match()* is called, it attempts to find the lowest derived-key of a point lying within the query region which is equal to or greater than the page-key of the successor page to the one just searched. We call this derived-key the *next-match*. If a next-match is found the page which may contain its corresponding datum-point is also identified, retrieved and searched. The process continues in this manner until no further next-matches can be found. The number of times which *calculate_next_match()* is called always equals the number of pages intersecting with the query region plus one.

In contrast to the Z-Order Curve, knowledge of the derived-key values of the lower and upper bound points of the query region does not assist in the query process. This necessitates a different approach to query execution where the Hilbert Curve is used for mapping between one and n dimensions.

5.2 The Querying Algorithm

We illustrate the operation of *calculate_next_match()* with an example, in terms of descending the tree representation of the Hilbert Curve described in section 3.

Figure 3 shows an example query region as a shaded area. The quadrants at all levels within the tree representation of the Hilbert Curve intersecting with this query region are also shown shaded in Fig. 2. We assume that *calculate_next_match()* has already been called once and that the first page intersecting the query region, $P1$, has already been identified, retrieved and searched. On calling *calculate_next_match()* a second time, we are searching for the next-match which is equal to or minimally greater than the page-key of the successor page to the one just searched. This page-key is the derived-key of point **B** shown in Figs. 2 and 3 and we call it the *current-key*. Its value is 001110. Note that the next-match must be a derived-key corresponding to a point lying within the query region. In this example, the next-match is 110110, the derived-key of point **E**.

The search begins at the root of the tree, descends and terminates at the member of a leaf which is the next-match. Back-tracking takes place in our example but it is not always required. At each level we carry out a binary search on the derived-keys of the quadrants within a node. A binary search finds a quadrant intersecting with the query region and whose derived-key, ie number, is as small as possible while also containing points whose derived-keys are equal to or greater than the current-key. That the next-match should be as small as possible accounts for the sorting of quadrants within a node by derived-key rather than by n-point value. Binary searching is particularly important for applications in higher dimensions since a node contains 2^n 'quadrants'.

In our example, the value of the next-match is initially unknown: ??????. Our search for the next-match proceeds as follows:

Step 1: Tree Level 1: A binary search of the root node shows that quadrant number 00 is the lowest numbered quadrant intersecting with the query region. It also contains the point whose derived-key is the current-key. We know this because the top 2 bits of the current-key are also 00. Quadrant 00 may therefore contain a point which lies within the query region and whose derived-key is equal to or minimally greater than the current-key. The next-match is tentatively modified to 00????.

We note that the higher numbered 2 quadrants (ie 10 and 11) also intersect with the query region and that if the next-match is not found in a leaf which is a descendent of quadrant number 00, then it will be found in one of these.

Step 2: Tree Level 2: The search now proceeds down one level to the node in level 2 pointed to by quadrant numbered 00 in the root. (In effect, the search space is restricted to one of the quadrants in the root node). Although all of the quadrants in the node at level 2 intersect with the query region, the binary search first rejects the lower numbered 2 quadrants (ie 00 and 01) and then the lower of the other two (ie 10) because the current-key lies within quadrant numbered 11. We know this because the middle 2 bits of the current-key are 11. We must therefore search for the next-match in this quadrant. The next-match is tentatively modified to 0011??.

Note that there is no higher numbered quadrant within the node intersecting with the query region.

Step 3: Tree Level 3: The search now proceeds down one level to the node in level 3 pointed to by quadrant numbered 11 in the node searched in step 2. The binary search finds that the query region intersects the lower numbered 2 quadrants only but that the bottom 2 bits of the current-key (ie 10) place it in the upper 2 numbered quadrants. The next-match therefore does not lie in the node currently being searched. No point lying within the intersection of the current node and the query region can have a derived-key equal to or greater than the current-key.

Step 4: Tree Level 1: The search back-tracks up to the root node which was found in step 1 to contain at least one quadrant intersecting with the query region and containing points with derived-keys all greater than the current-key. Two bits of the next-match are removed for each level of ascension; thus it is re-set to ??????. Binary searching resumes and quadrant number 11 is found to be the lowest numbered quadrant intersecting the query region, enabling the next-match to be recalculated as 11????.

(More generally, back-tracking ascends to the lowest level in the tree possible, where a sub-set of quadrants intersect with the query region but enclose only points mapping to higher derived-keys than the current key. If no such sub-set of quadrants exist at any level, back-tracking cannot take place and this signifies that no next-match to the query exists. The query then terminates. Note that back-tracking occurs at most once only and that if it is possible then a next-match is guaranteed to exist. Additionally, no further regard to the value of the current-key is required.)

Step 5: Tree Level 2: The search now proceeds down one level to the in level 2 node pointed to by quadrant number 11 in the root. Binary searching determines the lowest numbered quadrant intersecting with the query region to be quadrant number 01. The next-match is modified to 1101??.

Step 6: Tree Level 3: The search now proceeds in the level 3 node pointed to by quadrant numberd 01 in the node searched in step 5. Binary searching determines the lowest numbered quadrant intersecting with the query region to be quadrant number 10. The next-match is now determined to be 110110 and the calculation is complete.

The search process is expressed more formally as an algorithm in [11] and in further detail still in [10].

5.3 Binary Searching in Nodes

Broadly, the approach described above can also be applied to some curves other than the Hilbert Curve, for example the Z-Order Curve. However, the Hilbert Curve has at least $n2^{n-1}$ different node types which complicates binary searching. In contrast, the Z-Order Curve has a single node type only. Solving the problem of how to perform binary searches within nodes in a tree containing different node types is crucial for the viability of the application of the Hilbert Curve. In this sub-section we describe our solution.

Before searching a node we define the the *current-query-region* as the intersection of the query region and the sub-space defined by the node. As with the original query region the current-query-region is defined by lower and upper bound coordinates.

If the top bit of a coordinate is in 'bit position 1', then during a search of a node at tree level j, we encapsulate the current-query-region with a pair of n-points made up of bits taken from position j in the lower and upper bound coordinates. They locate the current-query-region within a search space defined by a node.

At the end of each iteration of a binary search of quadrants within a node, we discard half of the quadrants and search a smaller sub-set in the next iteration. Apart from ensuring we focus on sets of quadrants enclosing one or more points (at the leaf level) mapping to derived-keys equal to or greater than the current-key, we must focus on those intersecting with the query region. The question immediately arises of how do we know from quadrant derived-keys whether lower and/or upper halves of numbered quadrants in a sub-set intersect with the query region?

Solving this requires a function which returns a quadrant's n-point given its derived-key. The function also needs to know which type of node (ie orientation of first-order Hilbert Curve) is currently being searched. We call this function *d_to_c()*. Implementation is particularly simple where state diagrams are employed.

As a result of the symmetrical nature of the Hilbert Curve, we note the following. If the derived-keys of a sub-set of quadrants are in the range

$$[\ \textit{lowest}, \ldots, \textit{max-lower},\ \ \textit{min-higher}, \ldots, \textit{highest}\ ]$$

then all of the quadrants whose derived-keys are in the lower sub-range $[\ \textit{lowest}, \ldots, \textit{max-lower}\ ]$ have the same value, 0 or 1, in their coordinates in one particular dimension, i. Similarly, all of the quadrants whose derived-keys are in the higher sub-range $[\ \textit{min-higher}, \ldots, \textit{highest}\ ]$ have the opposite coordinate value, 1 or 0, in the same dimension, i. This characteristic applies in exactly one dimension only.

We also recall that quadrants whose derived-keys are consecutive are adjacent in space. Thus an n-point variable *pd* (short for 'partitioning dimension'), containing a single non-zero bit corresponding to dimension i and dividing this range into two, is evaluated as

$$\textit{pd} \Leftarrow \ \mathrm{d\_to\_c}\,(\textit{max-lower}) \ \oplus\ \mathrm{d\_to\_c}\,(\textit{min-higher})$$

where the symbol $\oplus$ is the bitwise exclusive-or operator.

It now remains to be found whether the quadrants whose derived-keys are in the lower sub-range all have the value 0 or 1 in dimension i. This is done by testing the value of the expression

$$\mathrm{d\_to\_c}\,(\textit{max-lower}) \ \wedge\ \textit{pd}$$

where the symbol $\wedge$ is the bitwise 'and' operator. If it evaluates to non-zero, then these quadrants all have the value 1 in dimension i, otherwise they have the value 0.

Finally, we are able to determine whether the current-query-region intersects the quadrants whose derived-keys are in the lower sub-range. For there to be an intersection, either (or both) of the lower and upper bounds of the current-query-region must have the same value in its coordinate for dimension i as these quadrants.

We determine whether the current-query-region intersects the half of the sub-set of quadrants of current interest whose derived-keys are in the higher sub-range in a similar manner.

We conclude this section by illustrating the operation of the binary search with an example showing how step 5 in the example from section 5.2 is executed.

The query lower and upper bound coordinates are $\langle\, 010,\, 000\, \rangle$ and $\langle\, 100,\, 010\, \rangle$ respectively.

At the end of step 4, the search space is restricted to quadrant number 11 in the root node. This corresponds to a region with lower and upper bound coordinates of $\langle\, 100,\, 000\, \rangle$ and $\langle\, 111,\, 011\, \rangle$.

In restricting the query region we compare each coordinate of its bounds with those of the restricted search space. Query lower bound coordinates which are less than the restricted search space equivalents are increased and upper bound coordinates which are greater than the restricted search space equivalents are reduced. The current-query-region is then bounded by the points $\langle\, 100, 000\,\rangle$ and $\langle\, 100, 010\,\rangle$.

In step 5, the n-points encapsulating the current-query-region are formed from the second from top bits taken from its coordinates. Thus the lower bound n-point is 00 and the uper bound n-point is 01.

In the first iteration of the binary search of the quadrants of the node in tree level 2, we determine whether the query intersects the lower numbered pair (ie 00 and 01) as follows: Firstly, *pd* is calculated as d_to_c(01) $\oplus$ d_to_c(10) which evaluates to 01 $\oplus$ 00, ie 01. This signifies that quadrant numbers 00 and 01 have the same coordinate values in the y dimension. Secondly, d_to_c(01) $\wedge$ *pd* evaluates to 01 $\wedge$ 01, ie 01. This signifies that quadrant numbers 00 and 01 have the value of 1 for their coordinates in the y dimension.

Since the n-point encapsulating the upper bound of the current-query-region also has a value of 1 for its y coordinate, the current-query-region must intersect with the lower numbered two quadrants within the node. (Note that since the lower bound's n-point has a zero-valued y coordinate, the current-query-region must also intersect with the higher numbered quadrants; this is confirmed by Fig. 2).

In a similar way, the second iteration of the binary search now finds the lowest quadrant number, from the sub-set of quadrant numbers 00 and 01, intersecting with the current-query-region. As we see from Fig. 2, the result is quadrant number 01.

6 Conclusion

The application and algorithms described in this paper have been implemented as fully functioning software, hitherto in up to 16 dimensions but this could easily be extended. Preliminary experimentation using artificially generated data produces promising results indicating that the concepts scale well with increasing data volumes and numbers of dimensions.

References

[1] T. Bially. Space-Filling Curves: Their Generation and Their Application to Bandwidth Reduction. *IEEE Transactions on Information Theory*, IT-15(6):658–664, Nov 1969.

[2] A.R. Butz. Alternative Algorithm for Hilbert's Space-Filling Curve. *IEEE Transactions on Computers*, 20:424–426, April 1971.

[3] C. Faloutsos. Gray Codes for Partial Match and Range Queries. *IEEE Transactions on Software Engineering*, 14(10):1381–1393, Oct 1988.

[4] C. Faloutsos and S. Roseman. Fractals for Secondary Key Retrieval. *Proceedings of the Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 247-252, 1989.

[5] V. Gaede and O. Günther. Multidimensional Access Methods. *ACM Computing Surveys*, 30(2):170–231, June 1998.

[6] D. Hilbert. Ueber stetige Abbildung einer Linie auf ein Flachenstuck. *Mathematische Annalen*, 38:459–460, 1891.

[7] H.V. Jagadish. Linear Clustering of Objects with Multiple Atributes. *Procs. of the 1990 ACM SIGMOD Int. Conf. on Management of Data*, 19(2):332–342, 1990.

[8] H.V. Jagadish. Analysis of the Hilbert curve for representing two-dimensional space. *Information Processing Letters*, 62(1):17–22, April 1997.

[9] A. Kumar. A Study of Spatial Clustering Techniques. In: *Procs. of the 5th Int. Conf. on Database and Expert Systems Applications (DEXA '94)*, vol 856, Lecture Notes in Computer Science, pages 57–71, Sept 1994.

[10] J.K. Lawder. The Application of Space-Filling Curves to the Storage and Retrieval of Multi-dimensional Data. PhD thesis, Birkbeck College, University of London, 2000.

[11] J.K. Lawder and P. J. H. King. Using Space-Filling Curves for Multi-dimensional Indexing. In: *Advances in Databases, 17th British National Conference on Databases (BNCOD 17)*, vol 1832, Lecture Notes in Computer Science, pages 20–35, July 2000.

[12] J.K. 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.

[13] J.K. Lawder. Using State Diagrams for Hilbert Curve Mappings. Technical Report JL2/00, Birkbeck College, University of London, 2000.

[14] B. Moon and H.V. Jagadish and C. Faloutsos and J.H. Saltz. Analysis of the Clustering Properties of the Hilbert Space-Filling Curve. Technical Report 99-10, University of Arizona, 1999.

[15] J.A. Orenstein and F.A. Manola. PROBE: Spatial Data Modeling and Query Processing in an Image Database Application. *IEEE Transactions on Software Engineering*, 14(5):611–629, 1988.

[16] F. Ramsak, V. Markl, R. Fenk, K. Elhardt and R. Bayer. Integrating the UB-Tree into a Database System Kernel. In: *Procs. of the 26th Int. Conf. on Very Large Databases*, pages 263–272, 2000

[17] H. Tropf and H. Herzog. Multi-dimensional Range Search in Dynamically Balanced Trees. *Angewandte Informatik*, 23(2):71–77, 1981.