Using Space-filling Curves for Multi-dimensional Indexing

J K Lawder and P J H King

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

Abstract. This paper presents and discusses a radically different approach to multi-dimensional indexing based on the concept of the space-filling curve. It reports the novel algorithms which had to be developed to create the first actual implementation of a system based on this approach, on some comparative performance tests, and on its actual use within the TriStarp Group at Birkbeck to provide a Triple Store repository. An important result that goes beyond this requirement, however, is that the performance improvement over the Grid File is greater the higher the dimension.

1 Introduction

Underlying any dbms is some form of repository management system or data store. The classic and dominant model for such repositories is that of some form of logical record or data aggregate type with a collection of instances conforming to that type usually termed a file. Such file systems are, of course, also used directly in many applications. The data model of a dbms may be radically different from this underlying repository model, the mapping between the two being a function of the dbms. With the well known relational systems, however, the mapping from the simple tabular data model to a collection of such repository files is generally straightforward.

The indexing of files can be either one-dimensional or multi-dimensional and we briefly review the two approaches in sections 2 and 3 of this paper. We then present our work on multi-dimensional indexing using the Hilbert Curve which was suggested by Faloutsos [4] although the idea was not fully developed. We are not aware of any actual implementation based on this approach prior to our own, but there has been some useful theoretical work supported by simulation on the clustering properties of the curve [5], [11], [13] and [10]. These studies generally agree on the superior clustering properties of the Hilbert Curve.

Section 4 introduces the concept of a space-filling curve using the Hilbert Curve as illustration. Sections 5 to 8 then discuss our own work on the Hilbert Curve in some detail, followed by Sect. 9, which discusses other types of curve, and Sect. 10 which summarizes our conclusions and further planned research.

2 One-dimensional File Structures

With one-dimensional file structures, the record type specifies the fields or attributes which a record may contain. In one of these fields, usually termed the primary key, a value is required in all records and all values must be distinct.

A characteristic of one-dimensional systems is that the primary key values are used to determine the placement of records in physical storage organized as a collection of pages, a page containing a number of records. The page is located using some form of physical address, or value from which such an address can be determined, known as a *page-key*. Given a primary key value, there is an algorithmic process of some kind, a B-tree index, hashing, etc which yields a page-key thus determining the page on which the corresponding record is to be placed in the case of a store operation or is to be found, if it exists, in the case of a retrieval operation. We assume in this paper that the reader is familiar with such one-dimensional file structures and with aspects such as page splitting, page merging, etc. The B-tree in particular is now well developed and has come to be the dominant technology for one-dimensional file systems.

Whilst one-dimensional indexing provides for efficient retrieval on the primary key, retrieval by the specification of other attribute values, if not to be by exhaustive search, requires the maintenance of supplementary or secondary indexes. For these the values are not required to be unique and, given a particular value, the index simply provides the primary keys for those records having this value which are then retrieved in the usual way. Although some gain can be had by operating on primary key lists prior to actual retrieval such as intersecting lists from different secondary indexes, organizing the placement of records to optimize retrieval other than for the primary key is not possible.

3 Multi-dimensional Indexing

Multi-dimensional indexing is based on the notion that more than one field or attribute of a record type should be specified as constituting the *primary key set* for records of that type and that all keys in the key set should play an equal role in determining the physical placement of a record in store and hence in optimization issues. Whilst every member of the key set must be defined for a record, uniqueness is required only for the set as a whole and not of its individual components. To fix ideas consider the indexing of objects at points in three-dimensional space; objects can lie on the same plane or line and thus can have any one or any two key values in common, but no two objects can occupy the same point and thus all three co-ordinates cannot be the same.

With multi-dimensional indexing we have the important facility of the retrieval of sets of records on partially specified keys. Thus with a three component key set we may wish for example, to retrieve all those records for two specified values of the key set but with any value for the other one; or with just one component specified and any values for the other two; or with intervals in all three dimensions, all the records whose key set values lie within the defined cuboid. Unlike the one-dimensional file structures therefore, with multidimensional structures the algorithms for retrieval are considerably more complex than those which determine the placement of a given record in physical storage and their development is a matter of research.

A landmark paper in work on multi-dimensional indexing is that of Nievergelt et al [14] in which the potential values of key sets are regarded as points in an n-dimensional cartesian product space. This product space we term the *key-space* and the points which correspond to actual records we term *datum-points*. The key-space is divided into cuboids with each cuboid being in one-one correspondence with a page of data in physical store, the records on that page being those that correspond to the datum-points in the cuboid. When a page which becomes full is split the cuboid is split correspondingly and similarly with recombination of under-used pages. This paper left open a number of questions relating to the structuring of the index to the key-space and the choice of dimensions for splitting. These were addressed by Derakhshan [3] whose work resulted in the repository system used by the TriStarp Group for the past decade in both its three and four dimensional versions. An evaluation and critique of this implementation is to be found in Lawder [12].

An important development along the same lines as the Grid File is the Bang File of Freeston [6]. In this approach the key-space cuboids are not disjoint but nested and whilst there are data pages in one-one correspondence with the cuboids, for only the innermost cuboids do they contain the records for all datum-points within it, in general containing only the records for the datum-points within it less those in the contained cuboids. This approach avoids the recombination deadlock problems which can occur with the Grid File and a method of indexing has also been developed. However no retrieval method for partial match or interval queries has yet been published and nor is such a method readily apparent.

A review of multi-dimensional indexing methods, including Guttman's R-Tree [8], is given by Gaede and Günther [7].

4 Hilbert's Space-Filling Curve

Space filling curves became a topic of interest to leading pure mathematicians in the late 19th century, the first paper being that of Peano in 1890 [16]. A more readable paper, however, is that of David Hilbert in 1891 [9] who gave the first geometrical interpretation. We briefly summarise his argument, using the same diagrams he used.

Without loss of generality we consider a mapping between the points of a square and a finite line using the definition of a point on the line as the limit of an infinite sequence of nested intervals whose length tends to zero and of a point in two dimensions as being the limit of an infinite sequence of nested squares whose area tends to zero. Figure 1 reproduces the figures in Hilbert's paper, except that we number the points from 0 rather than from 1. Figure 1(a) shows the initial square and line each divided into four, the numbers showing corre-

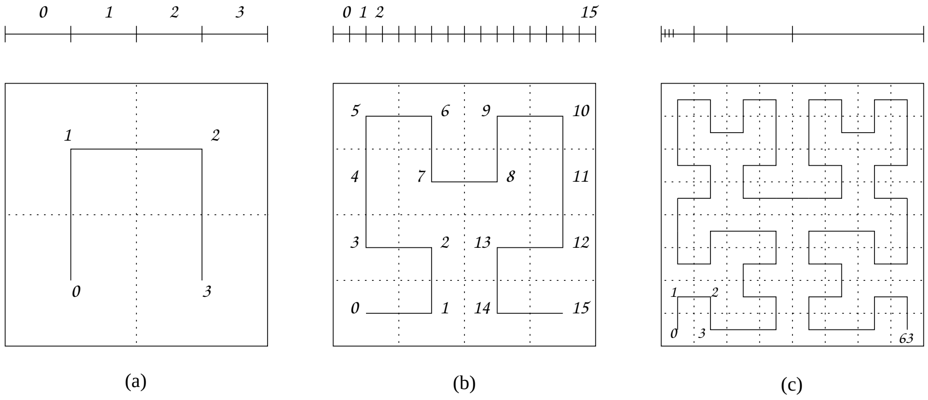


Fig. 1. Approximations of the Hilbert curve in 2 dimensions

spondences between sub-squares and line intervals established so that adjacent line intervals always correspond to adjacent sub-squares; the line connecting the centres of the sub-squares emphasizes their ordering. Figure 1(b) shows the next step in which each square and its corresponding line interval have been further subdivided with re-orientations of the sub-square sequences for the first and last sub-squares of Fig. 1(a) to ensure that the adjacency property is everywhere preserved. Figure 1(c) indicates the third step in the sequence.

If this process is continued to infinity then for any arbitrary point in the square there is a unique infinite sequence of nested squares for which this point is its limit. The corresponding infinite sequence of nested intervals on the line also defines a unique point. This argument also goes the other way and thus a one-one mapping is established between all points of the line and all points of the square. The points on the line are plainly ordered and this ordering thus gives the points of the square an ordering now known as the Hilbert Curve. Since this curve passes through every point in the square once and once only it is said to be space filling. The curve is everywhere contiunuous and nowhere differentiable. Intuitively one can see that the continuity results from the persistence of the adjacency property at every step and the non-differentiability from the sharp changes of direction at each step. For more on the mathematics and on other curves see Sagan [17].

But these mathematicians were concerned with limits at infinity whereas computer scientists are more concerned with the finite albeit large. We therefore say the curve in Fig. 1(a) is the First Order Hilbert Curve and is in a finite two dimensional space comprising only four points. It clearly“fills” this space. Likewise Fig. 1(b) shows the Second Order Hilbert Curve which fills a finite two-dimensional space of 16 points, and so on.

To specify the coordinates of the points in the space of Fig. 1(a) we need only a single bit but for the ordinal positions on the line two bits. The ordinal mapping established by the First Order Hilbert Curve is thus:

$$\langle\, 0,0 \,\rangle \rightarrow 00, \langle\, 0,1 \,\rangle \rightarrow 01, \langle\, 1,1 \,\rangle \rightarrow 10 \text{ and } \langle\, 1,0 \,\rangle \rightarrow 11.$$

For the points in the space of the Second Order Curve we require two bits for the coordinates and four bits for the ordinal numbering and the mapping established by this curve begins:

$$\langle\, 00,00 \,\rangle \rightarrow 0000, \langle\, 01,00 \,\rangle \rightarrow 0001, \langle\, 01,01 \,\rangle \rightarrow 0010, \langle\, 00,01 \,\rangle \rightarrow 0011, \ \ldots$$

In practice we are interested in somewhat higher order curves. With 32 bit coordinates we would need the 32nd Order Hilbert Curve and would have 64 bit ordinal numbers. We therefore need some algorithmic procedures for the mapping which, given coordinates, produces the ordinal number and vice versa. These are discussed in Sect. 7.

Thus far we have discussed only two dimensions since in this paper we aim only to illustrate concepts. However the Hilbert Curve extends readily to any number of dimensions. In three the First Order Curve passes through the eight sub-cubes into which the initial cube is divided and the mapping is to a line divided into eight intervals, the adjacency property now being that consecutive sub-cubes have a common surface. For four dimensions the First Order Curve orders 16 cuboids with consecutive cuboids having a common three dimensional hypersurface. And so on! For more information on how the algorithms scale up and the general formulae see the thesis of Lawder [12], the working software of which can be parameterised to up to 16 dimensions and which could be modified to go beyond.

5 Indexing Using Space-Filling Curves

To construct an index to a multi-dimensional file we use a finite Hilbert Curve of appropriate order to give a linear ordering to the datum-points in the key-space. These datum-points are then partitioned by dividing the curve into consecutive sections. Each section is then put into correspondence with a page of storage which contains the actual records. The sequence number or *derived-key* of the first datum-point on a curve section is used as the corresponding page-key which gives the pages a logical ordering. The page-keys are then indexed using a conventional B-Tree or some variant which gives the physical page addresses.

An important point to note in our approach is that it indexes partitions of data rather than partitions of the key-space. This contrasts with most alternative approaches, including the Grid and Bang Files. Our approach enables us to avoid problems arising from partitions overlapping within the index, which are common in other approaches, including those based on the R-Tree and the Bang File.

6 A Tree Representation of the Hilbert Curve

As a result of the recursive way in which a space-filling curve is constructed, the correspondence between the points on a finite space-filling curve and their sequence numbers can be expressed as a tree structure, its height corresponding to the order of the curve. Not only does this provide an insight into how mappings are performed but, more importantly, a tree-like conceptual view greatly aids the development of algorithms which are used to facilitate the execution of queries. We use the 2-dimensional Hilbert Curve in describing the construction of the tree but the process is applicable to other curves and to higher dimensions. In what follows, we use the term *n-point* for the concatenation of the coordinates of a point on the first order curve; note that on a first order curve coordinates are single bit values.

We begin by placing a first order curve at the root of the tree. If we express the coordinates of points lying on a first order curve as n-points then the correspondence between one-dimensional sub-interval sequence numbers and coordinates given in Sect. 4 above results in a root node comprising the set of ordered pairs: $\langle 00, 00 \rangle$, $\langle 01, 01 \rangle$, $\langle 10, 11 \rangle$ and $\langle 11, 10 \rangle$. In this notation, the first value of each pair is the sequence number and the second value is the n-point representation of a point lying at the centre of a sub-square.

In the transformation to a second order curve each of these ordered pairs becomes the parent of a node similar to the root and so the height of the tree increases to 2. The middle two child nodes express a first order mapping which is the same as that of their parent and the other two express mappings which are different. An example of a tree whose height is 3 is given in Fig. 2. Tree level 1 corresponds to Fig. 1(a), level 2 corresponds to Fig. 1(b) and level 3 corresponds to Fig. 1(c).

The process of growing the tree can then be continued for each node at the lowest current level as the order of the curve increments. We note that the fanout of the tree equals the number of points on a first order curve.

The set of ordered pairs in all of the leaves corresponds to the finite set of points through which the curve passes while non-leaf nodes correspond to sub-squares which contain a sub-set of points at the leaf level.

7 Mapping Between n and One Dimensions

An algorithm by which the derived-key, D, of a point, P, is calculated is readily illustrated with the aid of the tree representation of the Hilbert curve and entails a traversal from root to leaf. This is given in Algorithm 1, where the most significant bit of a 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.

It is not a practical option, however, to base the mapping implementation on storing the tree representation of a curve explicitly and traversing it from root to leaf since the number of nodes would be excessive. Furthermore, the size of a node increases exponentially with the number of dimensions. There is, however, a finite number of distinct node types, ie orientations of the first order curve, and this number is independent of the height of a tree. This is apparent from Fig. 2. Thus it is possible to represent the tree as a state diagram in which each node type corresponds to a state. Once the height of the tree exceeds a relatively

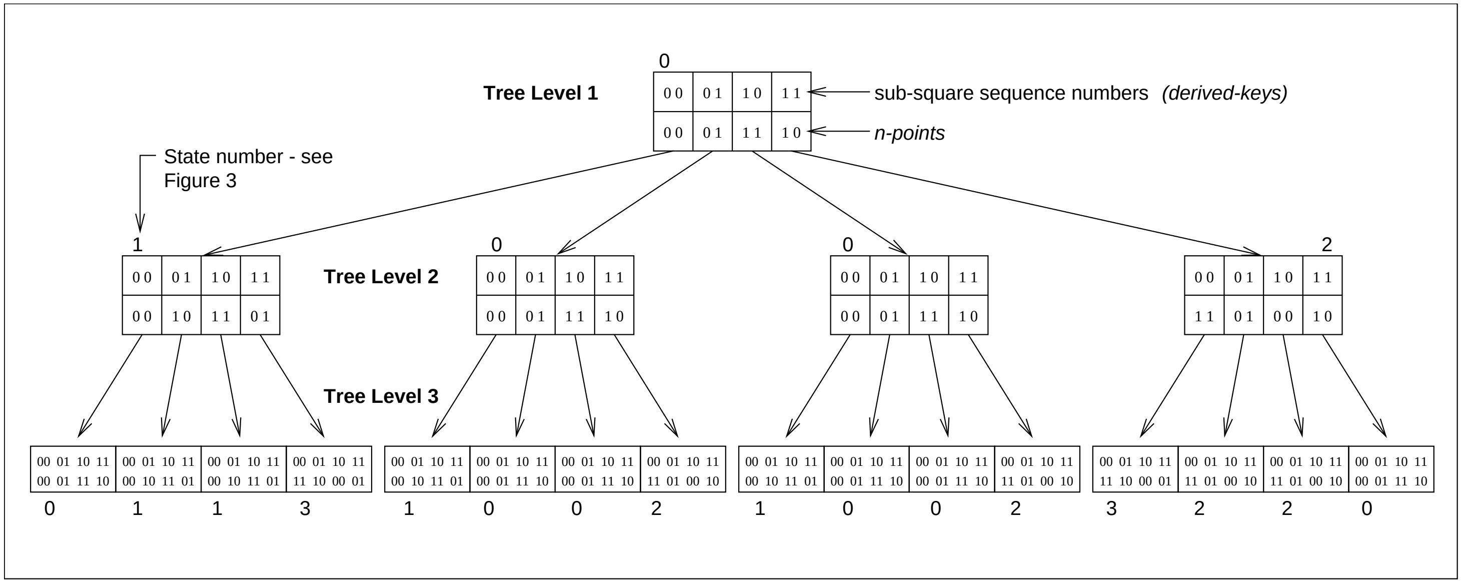


Fig. 2. The tree representation of the third order Hilbert Curve in 2 dimensions

Algorithm 1 Finding the *derived-key* of a Point by Traversing the Tree Representation of the Hilbert Curve

```
$current\_level \Leftarrow 1$
$current\_node \Leftarrow root$
$D \Leftarrow$ the empty bit-string
repeat
  $p \Leftarrow$ one bit in position $current\_level$ taken from each coordinate in $P$, concatenated into an n-point
  $d \Leftarrow$ the $n$-bit derived-key taken from $current\_node$ corresponding to $p$
  append $d$ to $D$
  if $current\_level < leaf\ level$ then
    $current\_node \Leftarrow$ node pointed to by the ordered pair $\langle p, d \rangle$ within $current\_node$
  end if
  $current\_level \Leftarrow current\_level + 1$
until $current\_level > leaf\ level$
```

low threshold, there are more nodes than states. A state diagram thus enables us to express the tree in a compacted form, since the states are not replicated in the diagram. The state diagram for 2 dimensions is given in Fig. 3.

The use of state diagrams for Hilbert Curve mappings was suggested in a technical report by Faloutsos [4] which refers to a method for generating state diagrams proposed by Bially [1]. Bially's technique is not specifically oriented towards the Hilbert Curve and comprises an incomplete set of rules requiring a manual process of trial-and-error in the generation of state diagrams.

Lawder [12] adapts Bially's technique to enable the automatic construction of state diagrams for the Hilbert Curve. State diagrams are useful in up to 10 dimensions for the Hilbert Curve, beyond which memory requirements become prohibitive. In higher dimensional space, the calculation method of mapping from one to n dimensions given by Butz [2] is used. Some improvements to Butz' method are given by Lawder, together with details of the inverse mapping.

8 Query Execution

8.1 Overall Approach

We noted earlier that a page represents a section of curve, ie a contiguous set of points, and will contain the records for datum-points which lie on that section. A query region, which is a hyper-rectangle, will always overlap one or more sections of the curve with intervening lengths joining them lying outside of the region. In other words, the curve may enter, leave and re-enter the query region a number of times.

Our querying algorithm therefore identifies every page whose curve section overlaps one or more of the curve sections of the query region. These pages are identified in ascending page-key order and any page whose curve section lies wholly outside the query region is, in effect, 'stepped over'. An example

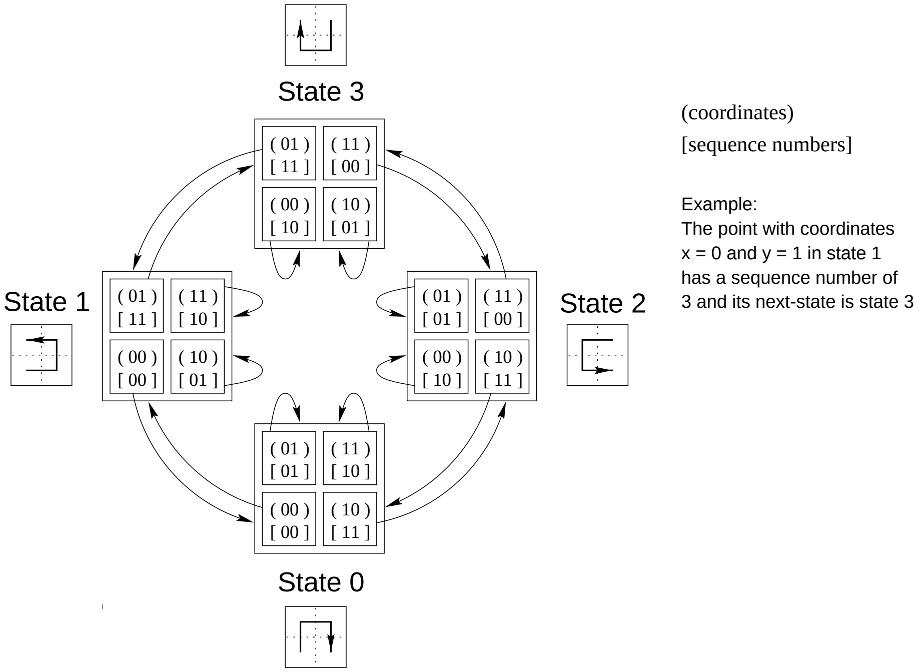


Fig. 3. A state diagram for the Hilbert curve in 2 dimensions

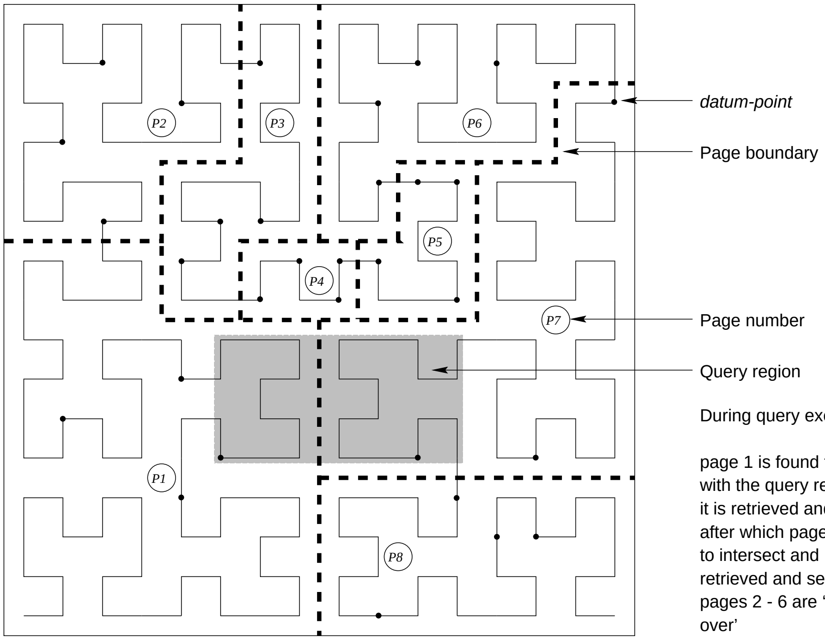
illustrating the concept is given in Fig. 4. To effect the stepping-over, a function is required which, given as input a derived-key value, i, will calculate and output the lowest derived-key which is equal to or greater than i and which corresponds to a point lying in the query region. We call this function *calculate_next_match* and the output value the *next-match*.

Once a page, intersecting with the query region, has been identified, retrieved and searched, the page-key of the next page is used as input to the *calculate_next_match* function. We then retrieve for searching the page which will contain the point mapping to the output of the function, if it is a datum-point. We identify the first page to be searched after calculating the lowest derived-key of any point in the query region by supplying the value of zero as input to the *calculate_next_match* function.

A description of the *calculate_next_match* function in more detail is given in the next section.

8.2 Algorithm for Calculating the *next-match* on a Hilbert Curve

We refer to the input to the *calculate_next_match* function as the *current-page-key* and refer to the sub-spaces resulting from a first order division of a space as *quadrants* regardless of the number of dimensions. Recall that a first order curve passing through a space connects the centre-points of these quadrants, which



During query execution:

page 1 is found to intersect with the query region and it is retrieved and seached, after which page 7 is found to intersect and is then also retrieved and searched; pages 2 - 6 are 'stepped over'

Fig. 4. Example range query on points mapped to the Hilbert curve

total 2^n in number. Also recall that the sequence number of a quadrant is also a derived-key and that the derived-key of a quadrant is an n-bit value.

Our search for a next-match, if one exists, is equivalent to finding the appropriate path in a descent, from root to leaf, of the tree representation of the partitioning of space induced by the Hilbert Curve. This descent is an iterative process. During each iteration, a node is searched to attempt to find within it a quadrant intersecting with the query region and mapping to the lowest derived-key equal to or greater than that of the quadrant holding the point corresponding to the current-page-key. The child node pointed to by this quadrant may then be searched in the next iteration but, as described below, sometimes the search for a next-match can be simplified and accelerated, sometimes back-tracking is required and sometimes the search fails thus signifying completion of the query execution.

The searching of a node is not straightforward when using the Hilbert curve since quadrants can be ordered differently in different nodes. The manner in which this problem is resolved is detailed by Lawder [12].

We recall that nodes correspond to sub-spaces and we call a node which is being searched during any particular iteration of the algorithm the *current-search-space*. As the tree is descended, at the end of each iteration, the current-search-space is restricted to one of its quadrants and thus is reduced in size by a factor of 2^n since a node contains this many quadrants. It is convenient also to restrict the query region to that part which intersects with the current-search-space. Note that the root node of the tree corresponds to the whole of the key-space.

On successful completion of each iteration of the search process, n bits of the value of the next-match are identified and provisionally appended to any previously calculated bits. These n bits are the sequence number of the quadrant found following a successful search of the current-search-space. Thus the highest, or most significant, n bits of the next-match designate the sequence number of the quadrant within the whole key-space in which the point corresponding to the next-match lies.

If the search of a node results in a quadrant which is a sub-space of the query region then the remainder of the search for a next-match may be simplified. One of two possible situations obtains. In one, the quadrant contains the point whose derived-key is the current-page-key. We therefore know that the current-page-key is itself the next-match and so the search may terminate immediately. In the other, the current-page-key is not the derived-key of a point within the quadrant and is a lower value than the derived-keys of all of the points within the quadrant. Therefore, the next-match must be the lowest derived-key of any point within the quadrant. All bits within the next-match which would otherwise have been calculated during searches of nodes at lower levels of the tree are now known to be zero-valued and again the search may terminate immediately.

Alternatively, the search of a node may result in a quadrant which intersects with but is not a sub-space of the query region. Again, one of two possible situations obtains. In one, the quadrant contains the point corresponding to the current-page-key and so the quadrant sequence number is appended to the next-match, as described above, and the search continues in the next lower level of the tree. In the other, the quadrant does not contain the point corresponding to the current-page-key and so the remainder of the search can be simplified since a next-match is guaranteed to exist. It will be the lowest derived-key of any point lying within the intersection of the quadrant and the query region. Thus the value of the current-page-key is of no further interest and there will be no possibility of a requirement to back-track, as described below.

Sometimes the search of a node fails to find any quadrant both intersecting with the query region and also having a derived-key equal to or greater than that of the quadrant containing the point corresponding to the current-page-key. Determining the next-match then requires back-tracking to a higher level in the tree where a node was previously found to contain at least one quadrant intersecting with the query region and containing points mapping to higher derived-keys than the current-page-key. If no such node, to which back-tracking may take place, was found previously then no next-match can exist and the search terminates. The lack of a next-match also signifies that the query process itself is complete.

Sub-sets of quadrants to which back-tracking can return may be found at more than one level of the tree as it is descended. In order for a next-match to be minimally higher than the current-page-key, back-tracking, if required, entails ascending the tree by the least possible number of levels, ie to the ‘most recently’ identified such sub-set of quadrants. For each level of ascension of the tree, n low order bits of the next-match are removed and must be recalculated once descent resumes.

If back-tracking is possible and takes place, then a similar situation arises as described above where the search results in a quadrant not containing the point corresponding to the current-page-key. Additionally, the quadrant may be a sub-space of the query region, also as described above.

The algorithm implemented as the *calculate_next_match* function is given in Algorithm 2 and for simplicity assumes a tree representation of the Hilbert Curve.

Algorithm 2 Algorithm to calculate the Hilbert *next-match*

```
current_level $\Leftarrow$ 1
current-search-space $\Leftarrow$ root
repeat
  $X \Leftarrow$ the derived-key of the quadrant in the current-search-space containing the
  current-page-key
  $Y \Leftarrow$ the lowest derived-key of any quadrant in the current-search-space inter-
  secting with the current-query-region, such that $Y \geq X$
  {this entails a binary search of the current-search-space, the recording of sub-
  spaces to which back-tracking may be required and may aslo entail backtracking
  (if possible)}
  current-search-space $\Leftarrow$ the quadrant whose derived-key $= Y$
  current-query-region $\Leftarrow$ the intersection of the current-query-region and the
  current-search-space
  append $Y$ to the next-match
  if $X = Y$ then
    if current-query-region = current-search-space then
      return the current-page-key as the next-match
    end if
  else
    if current-query-region = current-search-space then
      all remaining unresolve bits of next-match $\Leftarrow 0$
      return the next-match
    else
      calculate the next-match as the lowest derived-key of any point in the
      current-query-region
      return the next-match
    end if
  end if
  current_level $\Leftarrow$ current_level $+1$
until current_level $>$ leaf level
```

9 Other Space-Filling Curves

The approach described in this paper can be used for finite space-filling curves other than the finite Hilbert Curve by the substitution of appropriate mapping functions and some minor modifications to the *calculate_next_match* function.

Most notable is the Z-order curve, which has been applied, for indexing regions (eg the Bang File) and for spatial data (eg Orenstein's PROBE Project [15]). A Z-order derived-key is 'assembled' very simply by cyclically taking a bit from each coordinate of a point and appending it to those taken previously. This is sometimes referred to as 'bit interleaving'. Thus the 2 dimensional point P with coordinates

$$\langle\, x_1 x_2 x_3 \ldots x_k,\ y_1 y_2 y_3 \ldots y_k \,\rangle$$

maps to a derived-key of

$$x_1 y_1 x_2 y_2 x_3 y_3 \ldots x_k y_k,$$

where each x_i is a bit in a coodinate in dimension x, and similarly each y_i. The curve is illustrated in Fig. 5.

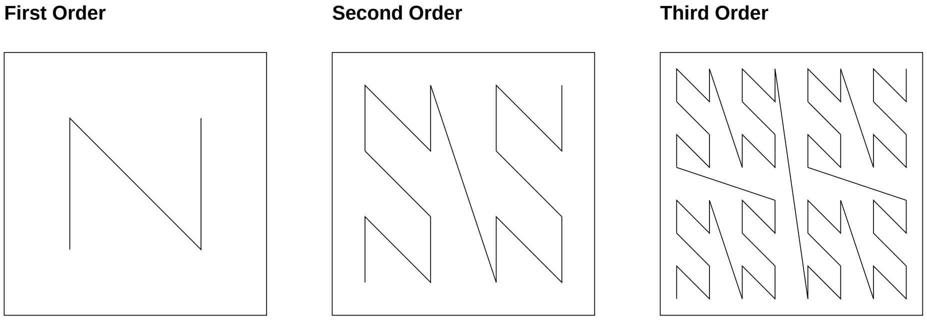


Fig. 5. Approximations of the Z-order curve in 2 dimensions

The direct relationship between individual bits in coordinates and Z-order derived-keys enables us to adopt a different algorithm for the *calculate_next_match* function which is more computationally efficient. This relies on manipulating bit values rather than conceptually descending a tree.

Where the page-key is not the next-match it must be incremented which entails changing a zero-valued bit to 1. In order to increment the page-key minimally, the algorithm first finds the lowest possible highest bit which must change and lower bits are then reduced in value, where possible. The method is given in outline in Algorithm 3, in which a *page-key-point* is the point corresponding to a page-key.

Other curves, but of less interest than the Z-order Curve, include the Gray-code, Scan and Snake Curves and are discussed by Lawder [12].

Algorithm 3 Algorithm to calculate the Z-order *next-match*

```
next-match $\Leftarrow$ page-key
check each coordinate in the page-key-point to find the highest bit in the next-match
    which must change:
    (a) from 1 to 0 because a page-key-point coordinate $>$ the corresponding coordinate
        in the range upper bound
        or
    (b) from 0 to 1 because a page-key-point coordinate $<$ the corresponding coordinate
        in the range lower bound
where the highest bit to change arises from condition (a) in some coordinate:
        find the lowest but higher bit in the next-match determined by some other
        coordinate which can be changed from 0 to 1, such that if all lower bits are
        set to 0, the coordinate value remains leq the corresponding coordinate in the
        range upper bound
set to 1 the bit identified in the next-match as being the highest zero-valued bit
    which must be incremented
set all lower bits in the next-match to 0
for each coordinate of the point corresponding to the next-match, if any is $<$ the cor-
    responding coordinate of the range lower bound, then increment the corresponding
    bits in the next-match to the values in the range lower bound
```

10 Conclusions and Future Work

The work reported in this paper has taken the idea of using space-filling curves for multi-dimensional indexing and the Hilbert Curve in particular and resolved the outstanding problems required to make such an approach practically usable; in particular the key problem of discovering a general algorithm for the all important query process. This has led to a working system which provides for the multi-dimensional indexing of records in up to 16 dimensions and which could readily be extended to higher dimensions if required.

This software has now been used practically for three dimensions as a Triple Store repository within the TriStarp Group and we have carried out preliminary experiments using randomly generated data to compare the system with the Group's existing multi-dimensional indexing sytstem based on the Grid File. These tests have indicated that the space-filling curve approach is more to be preferred the higher the dimensionality required. Comparisons with Guttman's R-Tree using 2-dimensional randomly generated spatial data showed a 75% saving in time taken to populate a data store and over 90% saving in time taken to execute range queries. Further details of performance tests undertaken can be found in Lawder's thesis [12]. We have also modified our software to give versions which use other space-filling curves and made some comparisons with the Hilbert Curve software. Our provisional conclusion is that none is better than the Hilbert Curve but more work is needed on this aspect.

Considerable scope remains for carrying out further experimentation and a number of potentially significant improvements need investigating. One interesting possibility is not to regard the pages as covering the whole curve. This could be effected by indexing each page using the minimum and maximum derived-keys for the datum-points which it contains. The maximum derived-key of one page and the minimum derived-key of the next then specify a section of curve which contains no datum-points. This should reduce the number of pages which have to be searched for a query and in some cases require no pages at all to be searched.

Thus far our attention has been principally focussed on point data as datum-points of records. We are aware, however, that multi-dimensional indexing using space-filling curves has potential application to spatial data which is another area meriting further work. We would be pleased to make our software available for bona fide experimental use to anyone with a multi-dimensional indexing requirement.

References

1. 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.
2. Arthur R. Butz. Alternative Algorithm for Hilbert's Space-Filling Curve. *IEEE Transactions on Computers*, 20:424–426, April 1971.
3. Mir Derakhshan. A Development of the Grid File for the Storage of Binary Relations. PhD thesis, Birkbeck College, University of London, 1989.
4. Christos Faloutsos and Shari Roseman. Fractals for Secondary Key Retrieval. Technical Report UMIACS-TR-89-47, University of Maryland, 1989. http://www.cs.cmu.edu/ christos/cpub.html.
5. Christos Faloutsos and Yi Rong. DOT: A Spatial Access Method Using Fractals. In: *Proceedings of the Seventh International Conference on Data Engineering, April 8-12, 1991, Kobe, Japan*, pages 152-159. IEEE Computer Society.
6. M. Freeston. The BANG File: A New Kind of Grid File. In: Umeshwar Dayal and Irving L. Traiger (eds): *Proceedings of the Association for Computing Machinery Special Interest Group on Management of Data Annual Conference*, May 27-29, 1987, San Francisco, California, pages 260-269. ACM Press.
7. Volker Gaede and Oliver Günther. Multidimensional Access Methods. ACM Computing Surveys, 30(2):170–231, June 1998.
8. Antonin Guttman. R-Trees: A Dynamic Index Structure for Spatial Seaching. In: *SIGMOD '84: Proceedings of the Annual Meeting, volume 14(2) of SIGMOD Record*, pages 47–57. ACM, 1984.
9. David Hilbert. Ueber stetige Abbildung einer Linie auf ein Flachenstuck. *Mathematische Annalen*. 38:459–460, 1891.
10. H.V. Jagadish. Analysis of the Hilbert curve for representing two-dimensional space. *Information Processing Letters*, 62(1):17–22, April 1997.
11. Akhil Kumar. A Study of Spatial Clustering Techniques. In: Dimitris Karagiannis (ed): *Proceedings of the 5th International Conference on Database and Expert Systems Applications (DEXA '94)*, volume 856 of Lecture Notes in Computer Science, pages 57–71. Springer-Verlag, Sept 1994.

12. Jonathan Lawder. The Application of Space-Filling Curves to the Storage and Retrieval of Multi-dimensional Data (Submitted for PhD). Technical Report JL/1/99, Birkbeck College, University of London, 1999.
13. Bongki Moon and H.V. Jagadish and Christos Faloutsos and Joel H. Saltz. Analysis of the Clustering Properties of the Hilbert Space-Filling Curve. Technical Report CS-TR-3611 / UMIACS-TR-96-20, University of Maryland, 1996.
14. Jürg Nievergelt and Hans Hinterberger and Kenneth C. Sevcik. The Grid File: An Adaptable, Symmetric Multikey File Structure. *ACM Transactions on Database Systems (TODS)*, 9(1):38–71, 1984.
15. Jack 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. Giuseppe Peano. Sur une courbe, qui remplit toute une aire plane (On a curve which completely fills a planar region). *Mathematische Annalen*, 36:157–160, 1890.
17. Hans Sagan. Space-Filling Curves. Springer-Verlag, 1994.