# Linear Clustering of Objects with Multiple Attributes

*H V Jagadish*

AT&T Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

There is often a need to map a multi-dimensional space on to a one-dimensional space For example, this kind of mapping has been proposed to permit the use of one-dimensional indexing techniques to a multi-dimensional index space such as in a spatial database This kind of mapping is also of value in assigning physical storage, such as assigning buckets to records that have been indexed on multiple attributes, to minimize the disk access effort

In this paper, we discuss what the desired properties of such a mapping are, and evaluate, through analysis and simulation, several mappings that have been proposed in the past We present a mapping based on Hilbert's space-filling curve, which out-performs previously proposed mappings on average over a variety of different operating conditions

## 1. INTRODUCTION

There is often a need to map points from a multi-dimensional space into a single dimension In the database context, there are two significant applications in this regard The first is multi-attribute indexing Elements from a multi-dimensional attribute space are hashed (or otherwise indexed) on to a linear range of block addresses on disk Since selections may often be specified only on some of the attributes, or include ranges of attributes, it is desirable that points that are close together in multi-dimensional attribute space also be close together in the one-dimensional space For example, with multi-attribute hashing [19], record signatures are mapped to buckets which are stored on disk We would prefer to access consecutive rather than randomly scattered buckets in response to a relational query Similarly, with a grid file [13], there is a mapping from grid squares to disk blocks, and we would like to perform this mapping to minimize the number of disk blocks accessed, and would prefer that these blocks be sequential if possible



A second application arises due to a multi-dimensional indexing technique proposed by Orenstein [15] The idea is to develop a single numeric index (on a one-dimensional number line) for each point in multi-dimensional space, such that for any given object, the range of indices, from the smallest index to the largest, includes few points not in the object itself Other applications, not relevant to databases, are for bandwidth-reduction of digitally sampled signals [2] and for graphics display generation [17]

Mapping from a higher to a lower dimension is not at all unusual As early as in our first programming course, we learn how to take a two-dimensional matrix and map it into a linear range of memory addresses The typical solution is to use the row major format, scanning the matrix row by row

More sophisticated mapping functions have been proposed in the literature One, based on interleaving bits from the coordinates, was proposed in [15] An improvement was suggested by Faloutsos in [5], using Gray coding on the interleaved bits A third method, based on the Peano curve [18], sometimes also attributed to Hilbert [9], has been proposed in [7] In this paper, we develop and extend this proposal based on this space-filling curve To remain consistent with [7], we shall call this curve the Hilbert curve

In general, if the multi-dimensional data has to be accessed for computation or for queries, then the expected access pattern has to be taken into account when selecting the mapping from multi-dimensional space to one-dimensional space For example, the simple scan by row may be the best organization when dealing with a raster-scan printer or display device However, it will perform poorly if most accesses are in the form of queries that specify a selection by column

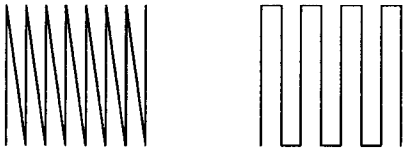
In this paper we study this mapping process from multiple dimensions to one dimension, and examine different criteria to evaluate a mapping function for the class of database applications outlined above We show that the choice of parameter values can greatly affect the relative performance of these mappings However, under most circumstances, the Peano mapping function proposed here outperforms the others

In Section 2 we introduce this mapping function, after first describing the other functions known in the literature Criteria for evaluating goodness are discussed in Section 3, along with an analysis of the mapping functions for the criteria proposed Section 4 describes some simulation experiments performed to compare different techniques, and the results obtained therefrom

For ease of exposition, Sections 2-4 all deal with a square two-dimensional space In Section 5, we show how the ideas of the preceding sections can be extended to multiple dimensions, and unequal extent in the different dimensions Finally, we conclude with some closing remarks in Section 6

## 2. MAPPING FUNCTIONS

In this section we discuss techniques to map a point from a given two-dimensional space to a one-dimensional space We begin with a brief mention of two straightforward schemes, and then go on, in Sections 2 1 and 2 2, to describe the two techniques previously proposed in the literature Finally, we present our proposal in Section 2 3 For simplicity, we assume square regions with dimensions that are powers of 2



(a) (b)

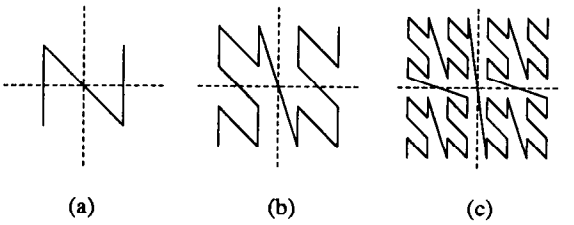
**Figure 1** (a) A Column-wise Scan, and (b) A Column-wise Snake Scan

The simplest possible mapping function is, of course, a scan, column-wise (or row-wise) See Fig 1a The assigned coordinate in one-dimensional space for a given point in two-dimensional space is the sequential position on the scan line of the point That is, given a point with coordinate $(x,y)$, it is assigned a linear coordinate of $x*ydim + y$, where $ydim$ is the number of different coordinate values along the $y$ dimension [1]

An easy improvement is to reverse the scan direction for alternate columns, as shown in Fig 1b Now, the linear coordinate of $(x,y)$ is still obtained as $x*ydim + y$ for even values of $x$ For odd values of $x$, the formula becomes $(x+1)*ydim - y - 1$

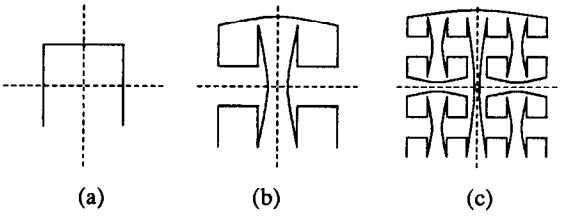
### 2.1 The z-curve

Proposed first by Orenstein [14-16], in this mapping the one-dimensional coordinate of a point is obtained by simply interleaving the bits of a binary representation of the X and Y coordinates of the point in two-dimensional space[2] Fig 2 shows the resulting mapping for a few different size 2-D grids The zig-zag shape (which would resemble the letter z if the X and Y axes were transposed) gives rise to the name, z-curve



(a) (b) (c)

**Figure 2.** Recursive development of the z-curve



(a) (b) (c)

**Figure 3** Recursive development of the gray-code mapping

One could also define the z-curve recursively One can think of dividing the given region into quadrants and drawing a curve such as Fig 2a Then each quadrant is divided in turn into four, and the same basic curve repeated in each, in place of each node in the previous step One more recursive step, again dividing each node into 4 joined by the basic curve, gives rise to Fig 2c This sort of recursive construction is useful to keep in mind for the rest of this section, when considering other mappings

### 2.2 Gray Coding

In a (binary) Gray code, numbers are coded into binary representations such that successive numbers differ in exactly one bit position Faloutsos [5, 6] observed that difference in only one bit position had a relationship with locality and proposed that the numbers produces by bit interleaving the two-dimensional coordinates of a point, as in the previous technique, be Gray coded to obtain the one-dimensional coordinate The curves that result are shown in Fig 3[3]

Once more the curve can be defined recursively As in the case of the z-curve above, begin with the curve of Fig 3a Divide each quadrant into 4 and replicate While replicating, rotate the two upper quadrants through 180 ° Thus we obtain Fig 3b Now divide into 4 once again, with replication and upper quadrant rotation to get Fig 3c , and so on

### 2.3 Hilbert Curve

It appears desirable, in the database context, to map from multiple dimensions to one dimension in a way that preserves "locality" as much as possible (The notion of locality will be made more precise in the next section For now an intuitive understanding will suffice) However, it is easy to show that it is not possible to map an $m$-dimensional grid into an $n$-

1 In C style, all coordinate ranges in this paper begin from zero An $n$-point coordinate range would be from 0 to $n-1$ inclusive

2 Actually, how the bits are combined depends on a *shuffle function* In the extreme case, one could group all the bits from one coordinate before all the bits from the other to obtain a column scan However, bit-interleaving is the "suggested" typical use, and we shall only consider this shuffle function for this paper

3 As in the case of z-curves, other Gray code mappings can be derived For this paper, we focus only on this ' typical ' Gray code mapping

dimensional grid, for any $n < m$ in such a way that two points that are close together in the former are always close together in the latter Considering nearest neighbors along the grid axes only, we can immediately see that each grid point has $2m$ nearest neighbors in the former, and only $2n$ nearest neighbors in the latter So at least $2(n-m)$ points have been forced to move at least one unit of distance away Similar arguments apply to points two units of distance away, three units away, etc

Therefore, the best that one can hope to do in a mapping from two dimensions to one dimension is to have two of the four nearest neighbors in the 2-D grid to be nearest neighbors in the linear mapping In other words, we would like to have a linear mapping in which successive points are nearest neighbors in the 2-D grid Looking at the mappings above, we find that this is the case neither for the z-curve nor for the Gray code The "jumps" in the linear traversal can easily be seen in Figs 2 and 3 We would like to find a mapping that avoids these jumps, if possible

The column scan, on the other hand, has only one jump per column, and this is fixed by doing the snake scan Therefore, at least as far as the nearest neighbor criterion is concerned, one can do no better than the snake scan Observe, however, that each node has two other nearest neighbors, and these tend to be quite far away in the scan mappings We would like a mapping in which these two other nearest neighbors are usually mapped to a point not too far away in the linear traversal

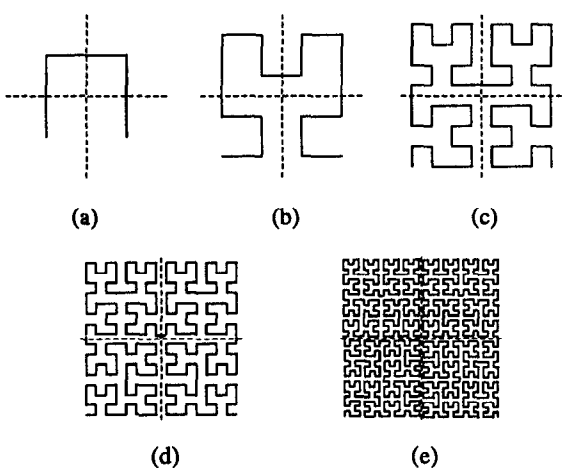



**Figure 4.** Recursive development of the Hilbert curve

These subjective criteria are met by the following mapping based upon the "space-filling" curve of Hilbert Begin with Fig 4a As in the previous cases, replicate in four quadrants When replicating, the lower left quadrant is rotated clockwise 90 °, the lower right quadrant is rotated anti-clockwise 90 °, and the *sense* (or direction of traversal) of both lower quadrants is reversed The two upper quadrants have no rotation and no change of sense Thus we obtain Fig 4b Remembering that all rotation and sense computations are relative to previously obtained rotation and sense in a particular quadrant, a repetition of this step gives rise to Fig 4c Further repetitions give Fig 4d and 4e

It is straightforward to obtain the linear coordinate along the curve for any given X and Y coordinate value in the 2-D grid In the spirit of the recursive construction above, the linear coordinate is obtained two bits at a time The most significant bits of the X and Y give the 2 m s b of the answer, along with the rotation and sense to be applied to the rest of the computation The next bit each from X and Y can be decoded in the light of the current rotation and sense to give the next two bits of the answer, and to provide a new rotation and sense that applies to the rest of the computation Thus the $n$-bit result is obtained in $n/2$ such iterations, with each iteration comprising only a handful of instructions that can all be executed in-memory See the Appendix for a simple code fragment that performs this computation See [3] for an alternative (byte-oriented) technique for computing this result The main point to note is that the cost of conversion is low of the order of a few dozen machine cycles The entire computation can be done in memory only a few very small tables are required

## 3. ANALYSIS OF GOODNESS

### 3.1 The Measures

Access to the data in a database context is typically to a subset specified by means of a selection criterion The selection itself may either be in the form of exact values specified on some (but not all) attributes, or in terms of ranges specified on some (possibly all) attributes We may also have a mix, with some attributes fully specified, some range-specified, and some not specified at all

A selection with some attributes fully specified and others not specified is a partially specified exact match query, or an associative search on one of multiple possible keys A selection with ranges specified on all attributes, could be a conjunctive selection in a traditional database (For example, "Find all employees who are more than 50 years old and earn between 50K and 100K dollars a year") It is also typical in a spatial database, where the ranges together specify a rectangular bounding box, as discussed in $R$-trees [8], $R^+$-trees [21], $R^*$-trees [1], and $P$-trees [10] See [20] for a tutorial exposition of this topic

In this and the following section, we shall consider the following two types of queries as representative

i One dimension exactly specified in a two-dimensional space, and the other dimension not specified, so that the selected points constitute a straight (horizontal or vertical) line through the grid

ii Both dimensions specified as ranges in the two-dimensional grid, giving a selected area that is rectangular

A linear traversal of the multi-dimensional space specifies the order in which the objects, or index entries, are placed on disk For each of the above access types, the appropriate measures of cost, for a proposed linear traversal, are

i Number of disk blocks fetched

ii Number of non-consecutive disk blocks fetched We prefer to fetch a set of consecutive disk blocks rather

than a randomly scattered set, due to the additional seek time involved

iii Size of the linear span for a given selection (difference between the maximum and minimum linear coordinate in the selected region) This measure is relevant only to Orenstein's idea of "reduce indexing to problem in one dimension"

The number of disk blocks fetched depends on the the capacity of each block in terms of the number of grid points that can be stored in each. For purposes of analysis, we shall use instead the number of runs of consecutive *grid points*, as a convenient measure of performance of the mapping algorithms under study This measure is exactly measure (ii) above, if each grid point is mapped to exactly one disk block[4] It is also highly correlated with measure (i), since (with many grid points to a disk block), consecutive points are likely to be in the same block while points across a discontinuity are likely to be in a different disk block Please note that this simplified measure is used only to render the analysis tractable Some weaknesses in this simplification are discussed in Section 3 4 and will be evident in Section 4

## 3.2 Partial Exact Match Selection

Since partial exact matches are not typical of spatial databases, the linear range of such a selection is not likely to be of interest, and we will not discuss it here Instead, let us see how many continuous runs of grid points are used for horizontal line and vertical line selections, in each of the five mapping techniques described in Section 2 Assume a square grid with $2^m \times 2^m$ points

For the column-wise scan, a vertical selection picks a single column corresponding to a single continuous run A horizontal selection picks a single row, which cuts across one grid point in each column, giving $2^m$ runs for each selection Averaging over every possible partial exact match selection, that is, over all rows and columns, there are $\frac{1}{2}(2^m + 1) = 2^{m-1} + \frac{1}{2}$ continuous runs per selection

For the column-wise snake scan, the results are the same as above, except that only $2^{m-1}$ runs are required for the top row, rather than $2^m$, since every pair of columns has a snake connection at the top Similarly, the bottom row requires only $2^{m-1}+1$ runs Thus the average number of runs required is

$$\frac{[1\times 2^m] + [2^m\times(2^m-2) + 2^{m-1} + (2^{m-1}+1)]}{2^m + 2^m} = 2^{m-1} + 2^{-m-1}$$

For the z-curve, a vertical selection picks a single column in which every pair of of points is connected and no more, so that there are $2^{m-1}$ runs A horizontal selection finds no points in the same row connected, so that $2^m$ runs are required The average becomes $1\,5\times 2^{m-1}$

For the Gray code mapping function, consider the basic unit, shown in Fig 3a A vertical selection on either column requires only 1 continuous run When this unit is replicated, as in Fig 3b, the number of runs required doubles for every column except two in which the transitions to the next level are made, saving one discontinuity We thus have two columns with 2 runs, and two with 1 run Replicating further, as in Fig 3c, doubles the number of runs once again, except for two columns Thus, the total number of runs in all the columns combined is 2 in a square grid of size 2, 6 in a square grid of size 4, 22 in a square grid of size 8, and so on Mathematically, let $V_k$ be the number of runs with grid size $2^k$, so that $V_1$ is 2, $V_2$ is 4, etc Then, we have

$$\begin{aligned} V_k &= 4V_{k-1} - 2 \\ &= 4^2V_{k-2} - 2\times 4 - 2 \\ &= 4^{k-1}V_1 - 2\times 4^{k-2} - 2\times 4^{k-3} - \quad - 2 \\ &= 2^{2k-1} - 2(4^{k-2} + 4^{k-3} + \quad + 1) \\ &= 2^{2k-1} - 2/3\times 4^{k-1} + 2/3 \\ &= 2/3(2^{2k-1} + 1) \end{aligned}$$

In the horizontal direction, $H_1$ is 3 rather than 2, and there is only one new horizontal link established in each doubling, so the recursion is

$$\begin{aligned} H_k &= 4H_{k-1} - 1 \\ &= 3\times 4^{k-1} - 1/3\times 4^{k-1} + 1/3 \\ &= 4/3\times 2^{2k-1} + 1/3 \end{aligned}$$

In particular, the desired average number of continuous runs over the $2^m$ possible vertical and $2^m$ horizontal selections is $(H_m + V_m)/2^{m+1}$, which is $2^{m-1} + 2^{-m-1}$

Since the Hilbert curve is rotated as it is developed, there is no need to separately compute the horizontal and vertical number of runs Instead, note that the total number of runs in a 2 by 2, $R_1$, is 5 When the size is increased by quadrupling, three new links are established Therefore

$$\begin{aligned} R_k &= 4R_{k-1} - 3 \\ &= 5\times 4^{k-1} - 3\times 4^{k-1} + 1 \\ &= 2\times 2^{2k-1} + 1 \end{aligned}$$

The desired average is $R_m/2^{m+1} = 2^{m-1} + 2^{-m-1}$

We thus find that the snake scan, the Gray code, and the Hilbert code, all three tie for the best performance, at an average of $2^{m-1} + 2^{-m-1}$ continuous runs, for an exact match selection on one of two attributes in a 2-D grid of size $2^m \times 2^m$ If the small lower order term is ignored, the simple column scan also weighs in with a similar performance All of these algorithms require on average about half as many continuous runs as the number of points (which is $2^m$) That is, if we have one disk block to a grid point, the number of random seeks is only one half the total number of disk blocks fetched in such a selection, on average

The z-curve mapping has a significantly worse performance at $1\,5\times 2^{m-1}$, which works out to three-quarters as many continuous runs as the number of grid points (It must be pointed out that this mapping was not proposed for this type of selection in the first place, so its poor performance

4 This assumption has been made in [6] and is valid for applications where the size of the information at each grid point can be tuned to match the size of a disk block, such as the size of a multi-attribute hash bucket or one cell in a grid file However, such matching may not always be possible when one or more attributes take on (successive) discrete values For example, pixels in an image with each pixel having a certain fixed sized record attached to it Similarly, in extensible hashing [4,11,12], there usually are many grid points to a disk block

here is not surprising). In fact, it has been shown [6] that the Gray code will always do better than the z-curve, with this type of selection and this cost measure

Between the three best mappings, however, there is no one that clearly wins It is easy to construct examples in which one or the other mapping performs better than the others One difference to notice is that the snake scan has an extreme asymmetry on performance between horizontal and vertical selections, the Hilbert code is almost perfectly symmetric (it can be shown that the total number of continuous runs in the two directions differ exactly by 1 for every $k$), and the Gray code is in between Therefore, if the objective is to minimize the average, when the probability of vertical selection is even slightly greater (less) than that of a horizontal selection, one should pick the snake scan (a rotated (row-wise) snake scan) However, one is typically interested in having a low variance as well For low variance, irrespective of probability, one should choose the Hilbert code With a composite objective function that minimizes both expectation and variance, one may pick any of the three mappings depending on how asymmetric the probabilities are

### 3.3 Range Selections

Rather than specify arbitrary ranges on both attributes, we simplify the analysis by considering only 2×2 square regions While there is no evidence that results derived for small square regions are applicable to arbitrary rectangular regions, there is reason to expect that a technique that does well on small regions, irrespective of the region alignment on even boundaries, is also likely to do well on large regions as well The cost measure is once more the number of continuous runs required to cover the 4 grid points in the region

For the column scan exactly 2 runs are required for every 2×2 region selected For the column snake scan, only 1 run is required for $2^m-1$ regions out of a total possible $(2^m-1)^2$ number of regions The average works out to $2-(2^m-1)^{-1}$, which is roughly 2 once again

For the z-curve, only one run is required if the selected region is aligned on an even boundary in both horizontal and vertical directions The probability of this occurring is roughly one half for each direction (It is actually a shade better than one half, since there are $2^{m-1}$ even alignment positions, and $2^{m-1}-1$ odd alignment positions We neglect this low order difference to simplify the computation) If the selected square is off alignment in both directions, 4 runs are required If the selected square is aligned horizontally, but not vertically, 3 or 4 runs may be required, depending on whether a multiple of 4 boundary was crossed or not Finally, if the selected square is aligned vertically, but misaligned horizontally, 2 runs are required So 1 run is required with probability 0 25, 2 runs with probability 0 25, 3 runs with probability 0 125, and 4 runs with probability 0 375, giving an expected number of runs of 2 625

For the Gray code, similarly, 1 run is required if the square is perfectly aligned, and 2 runs are needed if aligned vertically but misaligned horizontally If the selected square is misaligned vertically, 3 runs are required if the horizontal alignment is on an even boundary, and 4 runs if it is not So 1, 2, 3, or 4 runs are required, each with a probability of 0 25, for an overall 2 5 expected number of runs

For the Hilbert code, as in the previous two cases, even alignment in both directions requires only 1 run Due to the rotations of the basic template, computing the the number of runs requires much book-keeping when the alignment is off By recursively growing the size of the grid, a series of values can be obtained and summed, to produce a number very close to 2 That is, for the Hilbert code, the expected number of runs is 2 (The derivation is conceptually straightforward, but involves lengthy computation with careful book-keeping As such it is not presented here The interested reader should contact the author directly)

Let us turn now to the linear span spanned by a selected square region The column scan clearly has a linear span of $2^m+1$, being one full column of length $2^m$, and one element in addition, for every 2×2 square region selected In the snake scan, the linear span can vary between 3 (when the selected square covers the end at which two columns are connected) and $2^{m+1}-1$ (when it covers the other end of such a column pair) The average is easily obtained to be $2^m+1$, exactly the same as in the case of a simple column scan

For all three recursively defined mappings, the linear span can be computed by quadrature Divide the given $2^m\times2^m$ grid into quadrants With probability roughly $2^{-m}$, the selected 2×2 square will cross a vertical quadrant boundary, and with a similar probability it will cross a horizontal quadrant boundary For the z-curve, if a vertical quadrant boundary is crossed, the linear span will include roughly two full quadrants, or $2^{2m-1}$ points If a horizontal quadrant boundary is crossed, the number of grid points in the span is small by comparison If no quadrant boundary is crossed, then the selected square lies entirely within some one quadrant of the grid Divide this quadrant into four quadrants, and repeat the process With probability $2^{-m+1}$, the linear span will be $2^{2m-3}$ Continuing thus, we obtain a series that can be summed to get $2^m$ (Note that this is an approximate answer the actual average linear span will be somewhat higher)

The same computation applies to the Gray and Hilbert mappings as well, except that when the vertical quadrant boundary is crossed, the minimum linear span established is a function of the vertical coordinate as well, and ranges from almost 0 to nearly $2^{2m}$, but with an average of $2^{2m-1}$ The summation proceeds as before In the case of the Hilbert mapping, due to the rotation in some quadrants, the words "horizontal" and "vertical" may have to be flipped However, the mathematics remains the same

Thus we find for range selections that there is little to choose between the mappings in terms of linear span For a constant size of area selected, in all cases, the linear span obtained is of the order of the side of the grid, or the square root of the area of the grid In terms of obtaining a low variance, the simple column scan may actually turn out to be the best of the lot

In terms of the number of continuous runs, one again the scan mappings did the best along with the Hilbert code The Gray code came in next, and the z-curve after that

### 3.4 Caveat on Scan Mapping

All the preceding analysis could easily lead one to believe that a linear scan or snake scan mapping is all that one requires that all the discussion regarding more complex mappings like the z-curve, Gray and Hilbert, are pointless mathematical exercises While this is certainly true when the cost measure is the linear span, a caveat with regard to the other two cost measures is worth repeating

The cost measure used in the analysis, as a substitute for the number of disk blocks (randomly accessed disk blocks) fetched, is the number of continuous runs of grid points To the extent that each grid point maps to a single disk block, this analysis accurately measures the number of random disk accesses required (the total number of disk accesses being identical for all mappings) However, to the extent that multiple grid points may be mapped to a single block, the analysis above is only partly applicable In particular, small discontinuities are likely to be immaterial only large ones count Since many of the discontinuities are small in the recursively defined mappings, when these discontinuities are ignored the performance of these mappings will improve considerably On the other hand, almost all discontinuities are large in the scan mappings, and their performance remains essentially unaltered

For example, in a partial exact match selection, suppose that a continuous run is not interrupted if there is a discontinuity of one grid point (That is, a sequence "2,3,4,6,8,9" is considered a single run, rather than three runs "2,3,4", "6", and "8,9", since the breaks involve no more than one grid point each "5" and "7") Then the z-curve mapping has every alternate pair of grid points "connected" in a horizontal selection, improving its performance for horizontal selection, and overall, to $2^{m-1}$, as good as the scan mapping Similarly, if discontinuities of up to two grid points are ignored, then the Gray and Hilbert mappings improve by more than a factor of 2

We shall see these effects through simulation experiments in the next section In particular, Figs 5 and 6 highlight the weakness of the scan mapping

## 4. EXPERIMENTAL RESULTS

To compare the actual performance of the different mapping algorithms, several experiments were run A summary of the results obtained are presented here In all experiments the performance of the simple column scan and the snake column scan differed little, with the snake scan doing slightly better To minimize clutter only the results for the snake scan have been shown Where the results obtained exactly match the analysis in the previous section, only a mention is made, no curve is plotted The curves in the plots are identified by the letter S for scan, Z for z-curve, G for Gray, and H for Hilbert

Figures 5 through 7 show experiments in which each possible exact match selection was made Fig 5 shows the

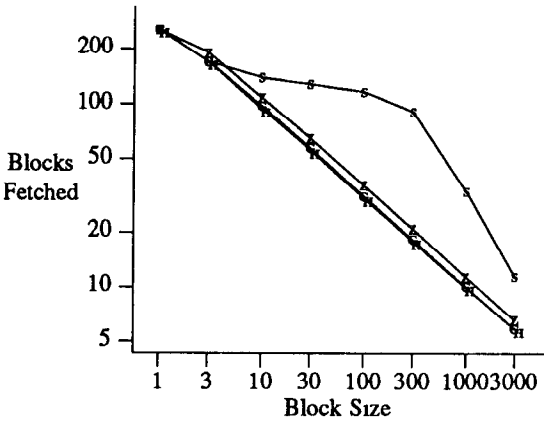



**Figure 5.** Number of blocks fetched decreases as the block size is increased on a grid size of 256 x 256

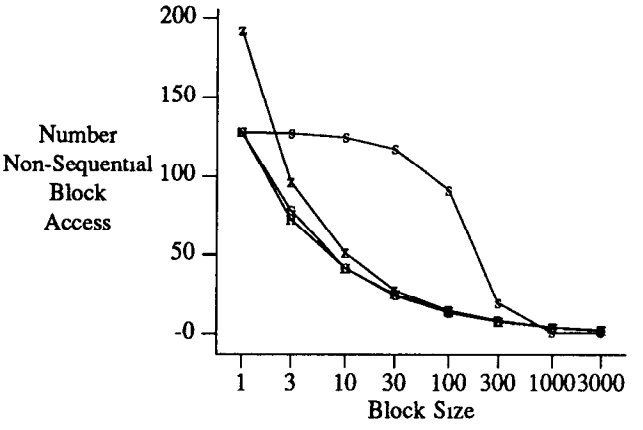



**Figure 6.** Number of non-sequential blocks fetched decreases as the block size is increased on a grid size of 256 x 256

variation in the number of blocks fetched as the block size is increased on a 256x256 grid (There are 512 possibilities, 256 vertical selections and 256 horizontal selections) The averages are plotted When each disk block holds exactly one grid point, 256 blocks are fetched irrespective of the mapping used As the block size increases to a few grid points, notice how the recursively defined mappings are able to capitalize on this and reduce the number of blocks fetched The Gray and Hilbert mappings both perform equally well (the curves are hard to distinguish in the figure) and better than the others The z-curve does almost as well also The scan mapping, however, is unable to capitalize on the increased block size and starts faring worse and worse relative to the others, up to a point Once the block size becomes large enough, including a whole column of grid points at a time, the scan mapping also begins to benefit At the extreme, if the entire database is resident in a single block, all the algorithms once again perform identically, requiring a single block fetch

The average number of runs of disk blocks fetched is shown in Fig 6 With block size one the numbers obtained perfectly match the analytical predictions There are exactly 192 runs for the z-curve, and 128 for the other three

mappings As the block size is increased, the average number of runs decreases As in Fig 5, this decrease is sufficiently slow for the snake scan that the z-curve is very soon able to do better than it. Once more, as the block size gets very large, all the mappings start looking alike

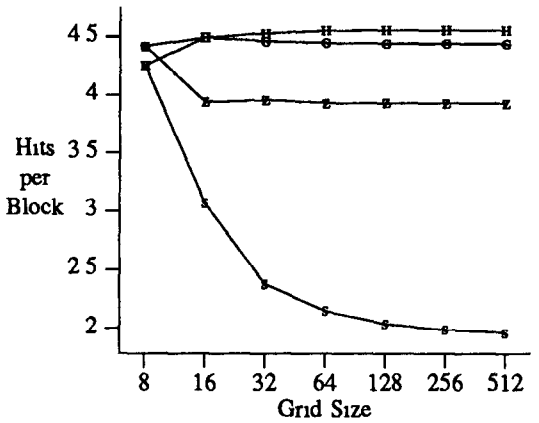



**Figure 7.** Number of relevant items fetched on average per block of size 30 items remains fairly constant as the size of the database is varied

The size of the grid was varied from 8x8 to 512x512 and the same trends were observed for all sizes Fig 7 is a plot as the grid size is varied of the number of hits per block, that is, the number of points that belong to the selected set on average per block fetched (This number can be obtained by dividing the number of points to be fetched, which is the length of one side of the grid, by the number of blocks that were fetched) As before, the curves show the average over all possible selections

The most heartening news in Fig 7 is that for all three recursive algorithms, the number of hits per block remains constant as the size of grid is increased In other words, irrespective of the size of the database, it is possible to have 4 to 5 relevant items in each block of size 30 Therefore, the effort required to fetch a selection remains proportional to the size of the selected set, but is independent of the size of the database In contrast, the scan mapping does worse and worse as the size of the database increases

The remaining experiments deal with range selections Figs 8 and 9 plot the averages obtained over every square of a specified size (which is varied on the X-axis in these figures) in a grid of size 128x128. Similar results were obtained for different size grids (not plotted)

Fig 8 is intended to measure the number of continuous runs of disk blocks fetched Since the number of disk blocks fetched varies as the size of the selected square is increased, this number has been normalized by the total number of disk blocks fetched, and plotted as the *fraction* of disk blocks read that are sequentially accessed The results plotted perfectly match the analytical predictions Considering the leftmost point in the curve, where the selected square size is 2x2, a total of 4 blocks have to be fetched, one for each of the 4 grid points For both the Hilbert code and the snake scan mapping, exactly half the reads, that is 2 block fetches, are non-

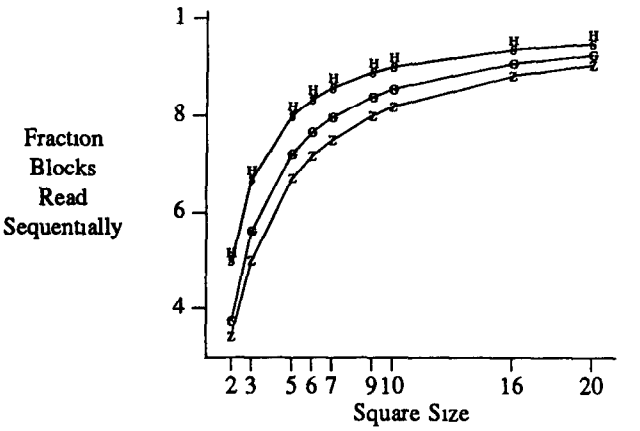



**Figure 8.** Fraction of blocks (of size 1) read sequentially as the size of the selected square is varied

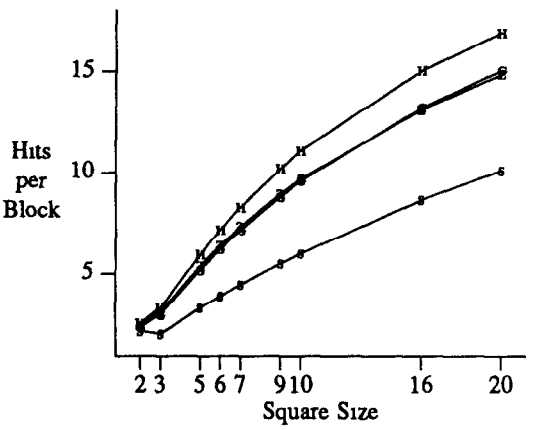



**Figure 9.** Number of relevant items fetched on average per block of size 30 items increases as the size of the selected square is increased on a grid 128 x 128

sequential, and the other half are sequential Similarly, the Gray code and the z-curve have points a little below 0 4 and a little above 0 3 respectively, corresponding to the analytically predicted 2 5 and 2 625 runs

As the size of the selected square is increased, all mappings do better In fact, the number of non-sequential blocks read appears to be proportional to the square root of the area selected (This square root relationship is easy to establish analytically for the scan mapping) The difference in performance between the curves continues with the Hilbert and snake scan curves doing the best with performance that is so close that the curves cannot be told apart in the figure

As the block size is varied, results similar to Figs 5 and 6 are obtained for range selections as well If the size of the database is varied, as in Fig 7, the performance remains roughly constant These curves have not been plotted Instead, in Fig 9, we measure the performance of all the mappings in terms of hits per block for a block size of 30 The results obtained here are even more heartening than in

Fig 7 As the size of selected area is increased, the number of hits per block goes up for all the mappings The Hilbert code mapping does the the best, followed by the Gray and z-curve mappings both of which do about equally well The snake scan comes in last These differences become larger as the size of the selected square is increased

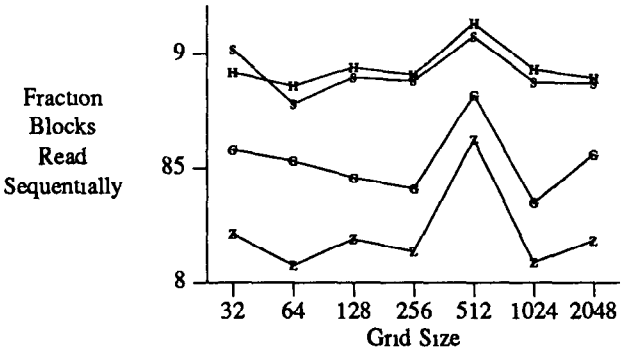



**Figure 10.** Fraction of blocks (of size 1) read sequentially as the size of the database is varied

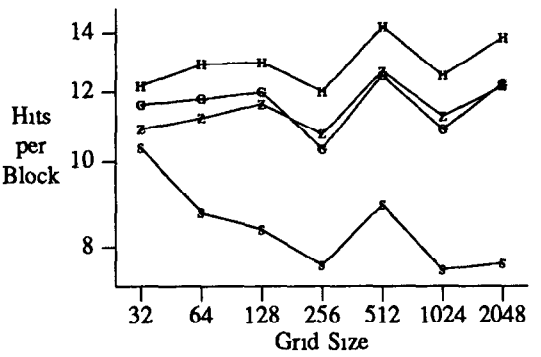



**Figure 11** Number of relevant items fetched on average per block of size 30 items remains fairly constant as the size of the database is varied

There is no reason why range selections should be restricted to square areas Several experiments were run with randomly generated rectangles, and some of the results are presented in Figures 10 through 13 In each case, 100 rectangles were randomly generated from a weakly exponential distribution that tended to keep the size of each rectangle small The size of selected rectangle (or selectivity) is varied in Fig 12, to check for sensitivity The results presented are the average of 100 random rectangles

In Fig 10, we measure the fraction of blocks (of size 1) read sequentially Just as in Fig 8, we find that the Hilbert mapping and the snake scan vie for the first place The Hilbert mapping seems to do just a shade better now The Gray code mapping comes in a distant third, with the z-curve even further behind On the X-axis in this figure, the size of the database is varied The fraction blocks read sequentially remains roughly constant as this size is varied

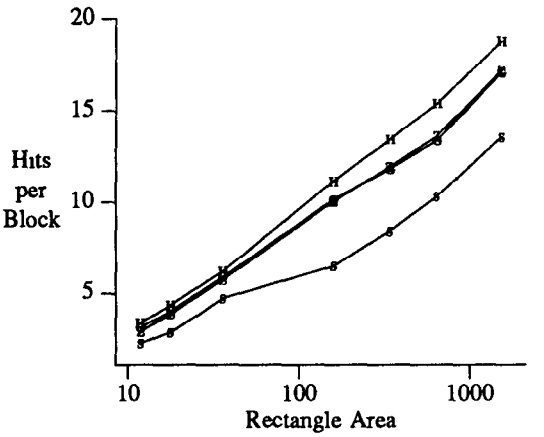



**Figure 12.** Number of relevant items fetched on average per block of size 30 items increases as the area of the selected rectangles is increased on a grid 512 × 512

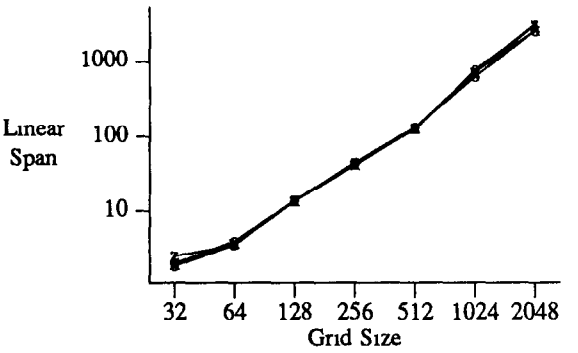



**Figure 13.** Average linear range spanned by a random rectangle as the size of the database is varied

Fig 11 measures the numbers of relevant items in each disk block (of size 30 items) fetched As one may expect from Fig 9, the Hilbert mapping performs the best, the Gray code and z-curve mappings compete for second place, and a column snake scan comes in last As the size of the database is varied, the performance of the three recursively defined mappings remains roughly constant, while that of the scan mapping deteriorates

Fig 12 shows the variation in hits per block as the size of the selected rectangle is varied We find that as the rectangle size is increased, each of the mapping techniques performs better, but their relative performance remains the same The trends are what one would expect on the basis of Fig 9, which plots the same curve for the case perfect squares

Finally, in Fig 13, we plot the linear span of the selected rectangular regions in the 2-D grid, normalized by the area of the region This span turned out to be almost identical on average for all the mappings used As predicted by the analysis, this span increases with grid size Moreover, the linear span (without any normalization) was found, in a separate experiment not plotted here, to vary by less than 25%

as the size of the selected rectangles was varied 2 orders of magnitude in a fixed size grid

## 5. EXTENSIONS

### 5.1 Multiple Dimensions

An $n$-bit binary reflective Gray code constitutes a tour of all the vertices of an $n$-dimensional hypercube, traversing only the edges between these vertices To develop an $n$-dimensional Hilbert mapping we begin by defining a basic unit in $n$ dimensions, of size 2x2x2x , using a binary reflective Gray code[5] Fig 14 shows the primitive that is used to construct a Hilbert mapping for $n = 3$

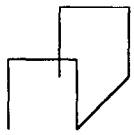


**Figure 14.** The Primitive for a 3-dimensional Hilbert Curve

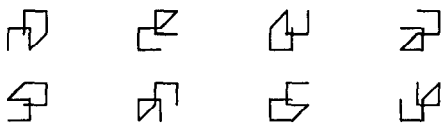


**Figure 15.** Different ways of covering a 2x2x2 cube and traversing one unit in the Z direction

The curve of Fig 14 has its end points separated by one unit in the Z direction The same separation of end points can be obtained through other curves such as those shown in Fig 15 One difference lies in the value of the X and Y coordinates of the points (differentiating the columns of Fig 15) Another difference lies in the basic curve used the top row of Fig 15 shows the same primitive rotated around the Z axis The bottom row shows rotations of a different primitive We have chosen the curve of Fig 14 as our basic primitive, and will use the top row of Fig 15 as necessary, to construct a Hilbert mapping from three dimensions to one

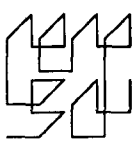


**Figure 16.** The Front Half of a Three-dimensional Hilbert Curve on a Cube 4x4

We now develop the recursive construction step Take the basic unit of Fig 14 and replicate one basic unit for each of its 8 nodes The first unit is at the front bottom left It must have its curve starting from the front bottom left corner, and must end next to front top left corner, making a transition in the Y direction An appropriate rotation and sense of the basic primitive can be found to achieve this See Fig 16 The second unit must have a transition along the X axis, and so on Thus the rotation and sense of each replicated unit is determined Once determined for the first recursive step, the same set of rotations and sense changes can be used for as many recursive steps as necessary to generate a mapping of a large enough grid

In a similar fashion, one can also extend the Gray code and z-curve mappings to multiple dimensions No experiments were run on multi-dimensional mappings However, the analysis presented in Section 3 can easily be extended to apply to any arbitrary dimensions The essential results remain the same If anything, the difference between the mappings gets exaggerated To understand why this happens, consider a scan mapping (Z changes fastest, Y next, and X most slowly) in 3 dimensions Two points that differ by one in the X coordinate value, will be mapped $n^2$ apart in the linear map, as opposed to only $n$ apart in the two dimensional case (where $n$ is the size of the grid in each dimension) Normalizing for the total number of points on the grid, we have gone from a difference that is $N^{1/2}$ to something $N^{2/3}$, where $N$ is the total number of points in the grid (equal to $n^{dimension}$) This trend continues as the dimension is increased further

### 5.2 Non-Square Grids

For ease of exposition, we have throughout assumed a square grid Often different attributes may have different size domains For example, most CRT screens are not square, but the pixels used on them are, so that different numbers of pixels are on the grid in the X and Y dimension For another example, consider two attributes age and sex, in some personnel database While there probably will be dozens of different integer values for the age attribute, there can possibly be only two values for the sex attribute This attribute space cannot conveniently be forced into a square grid

Consider a rectangular grid of size $2^m$ in one dimension and $2^n$ in the other, $n>m$ From a bit interleaving standpoint, we can ask what to do with the extra $n-m$ bits in one dimension These bits could be tacked on as the most significant bits of the $2m+n$ bit result word, or as the least significant bits, or they could be interspersed throughout We show how each of these options can be implemented, one by one

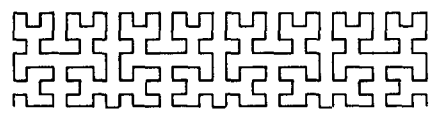


**Figure 17.** The 16x16 grid of Fig 4d is repeated four times to obtain a 64x16 grid

A $2^m \times 2^m$ square area can be covered with the usual Hilbert curve The displacement between the beginning and end of this curve is always exactly $2^m$ in the X direction Replicate this pattern $2^{n-m}$ times (rotating it first, if the Y dimension is bigger) We thus obtain a nearest neighbor connected curve

5 This does not imply that a multi-dimensional Hilbert mapping is the same as a Gray code mapping Recall that even in the two dimensional case, the Gray and Hilbert codes used the same elementary unit, in Fig 3a and Fig 4a The difference is in the construction

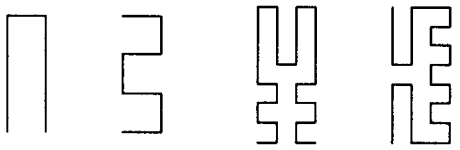



**Figure 18.** (a) A 4×2 primitive, (b) Its rotation by 90°, (c) These primitives used to construct an 8×4 grid, and (d) Its rotation

that covers the given area See Fig 17

A second option is to develop a primitive that is $2^{n-m+1}\times 2$ instead of just 2×2 Create a $2^{m-1}\times 2^{m-1}$ grid In the last step, instead of replicating the regular 2×2 primitive, use this special primitive Note that "rotation" of this primitive is a little strange rotating the primitive of Fig 18a gives Fig 18b This special definition of rotation is required because the aspect ratio of the primitive cannot change upon rotation, it has to be a change only in the connection points Every shape can be considered 4 pieces joined by 3 connections A rotation rotates the pieces, each piece independently The primitive of Fig 18a has each of 4 pieces a vertical segment, with three connections, one between each pair of successive pieces In the rotation, each piece becomes a horizontal segment, with three connections between them This idea becomes more clear comparing Fig 18c to 18d Fig 18c is constructed by the standard recursive generation rule for the Hilbert curve The first quadrant has a 18b piece in it, which rotated becomes the 18a piece The next quadrant in 18c is the 18a piece, which rotated becomes the 18b piece, and so on, to result in Fig 18d after the pieces are hooked together with the appropriate connections

An arbitrary interspersal of the bits in the two coordinates is achieved by defining primitives of size $2^k\times 2$, for any value of $k$ Wherever extra bits are to be interspersed, instead of performing the usual quadruplication with the 2×2 primitive, use this special primitive instead These ideas extend to multiple dimensions in a straightforward way

## 6. CONCLUSIONS

In this paper we introduced a mapping, based on the Hilbert curve, from multi-dimensional space to a line We discussed database applications in which such mappings are of value, and developed measures to evaluate their performance Through algebraic analysis, and through computer simulation, we showed that under most circumstances, the Hilbert mapping performed as well as or better than the best of alternative mappings suggested in the literature

In particular, this paper demonstrated that it is possible to develop a mapping of multi-attribute space to disk blocks such that the number of disk blocks retrieved on a selection is a function only of the size of the selected set and not of the size of the database Moreover, the number of disk blocks retrieved grows less than proportionately with the size of the selected set The same results still hold true if one counts number of non-sequential disk blocks fetched instead of total number of disk blocks fetched

These results are especially heartening in light of the proof, in [22], which established in general (given a uniform distribution of key values), that a $k$ attribute selection on a database with $N$ records has a file access cost of $O(N^{\frac{k-1}{k}})$ Our experiments seem to indicate that, for the specific classes of queries studied, we can do much better than this worst case

## APPENDIX

### Code Fragment for Generating the Two Dimensional Hilbert Mapping

```
int rotation_table[4] = {3, 0, 0, 1} ,
int sense_table[4] = {-1, 1, 1, -1} ,
int quad_table[4][2][2] = { {{0,1},{3,2}},
        {{1,2},{0,3}}, {{2,3},{1,0}}, {{3,0},{2,1}} } ,
{
 rotation = 0 ,                 /* Initially no rotation */
 sense = 1 ,                    /* Initially positive sense */
 num = 0 ,
 for (k = side/2 , k > 0 , k = k/2 ) {
  xbit = x/k ,   /* Get the m s b of x */
  ybit = y/k ,
  x -= k*xbit ,   /* Take away the current m s b */
  y -= k*ybit ,
  quad = quad_table[rotation][xbit][ybit] ,
                 /* Which quadrant am I in? */
  num += (sense == -1 ) ? k*k*(3-quad)  k*k*quad ,

  rotation += rotation_table[quad] ,
                 /* Fix rotation value for next time */
  if (rotation >= 4) rotation -= 4 ,
                 /* Addition is modulo 4 */
  sense *= sense_table[quad] ,
                 /* Fix sense value for next time */
 }
}
```

The above code fragment produces in *num* the linear position of the grid point with coordinates $x$ and $y$, in square grid with side of size *side*

## ACKNOWLEDGEMENTS

I would like to thank Bruce Hillyer for a very careful reading of the manuscript

## REFERENCES

[1] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger, "The R*-tree An Efficient and Robust Access Method for Points and Rectangles," *Proc ACM-SIGMOD Conf on the Management of Data*, Atlantic City, NJ, May 1990

[2] T Bially, "Space-Filling Curves Their Generation and Their Application to Bandwidth Reduction," *IEEE Trans on Information Theory*, **IT-15**(6), Nov 1969, 658-664

[3] A R Butz, "Alternative Algorithm for Hilbert's Space-Filling Curve," *IEEE Transaction on Computers*, C-20, Apr 1971, 424-426

[4] R Fagin, J Nievergelt, N Pippenger, and H R Strong, "Extendible Hashing — A Fast Access Method for Dynamic Files," *ACM Trans on Database Systems*, 4(3), Sept. 1979, 315-344

[5] C Faloutsos, "Multiattribute Hashing Using Gray Codes," *Proc ACM-SIGMOD Int'l Conf on the Management of Data*, Washington, D C , 1985, 227-238

[6] C Faloutsos, "Gray Codes for Partial Match and Range Queries," *IEEE Trans on Software Engineering*, 1987

[7] C Faloutsos and S Roseman, "Fractals for Secondary Key Retrieval," *Proc of the ACM Conf on the Principles of Database Systems*, March 1989, 247-252

[8] A Guttman, "R Trees A Dynamic Index Structure for Spatial Searching," *Proc ACM SIGMOD Int'l Conf on the Management of Data*, 1984, 47-57

[9] D Hilbert, "Uber die steitige Abbildung einer Linie auf ein Flachenstuck," *Math Ann* , **38**, 1891

[10] H V Jagadish, "Spatial Search with Polyhedra," *Proc Int'l Conf on Data Engineering*, Los Angeles, CA, Feb 1990

[11] P-A Larson, "Dynamic Hashing," *BIT*, **18**, 1978, 184-201

[12] W Litwin, "Linear Hashing A New Tool for File and Table Addressing," *Proc of the Sixth Int'l Conf on Very Large Databases*, Montreal, 1980, 212-223

[13] J Nievergelt, H Hinterberger, and K C Sevcik, "The Grid file An Adaptable Symmetric Multikey File Structure," *ACM Trans on Database Systems*, 9(1), 1984

[14] J A Orenstein and T H Merett, "A Class of Data Structures for Associative Searching," *Proc Third SIGACT News SIGMOD Symposium on the Principles of Database Systems*, 1984, 181-190

[15] J A Orenstein, "Spatial Query Processing in an Object-Oriented Database System," *Proc ACM SIGMOD Int'l Conf on the Management of Data*, 1986, 326-336

[16] J A Orenstein, "Redundancy in Spatial Databases," *Proc ACM SIGMOD Int'l Conf on the Management of Data*, Portland, OR, May-June 1989

[17] E A Patrick, D R Anderson, and F K Bechtel, "Mapping Multidimensional Space to One Dimension for Computer Output Display," *IEEE Trans Computers*, C-17, Oct 1968, 949-953

[18] G Peano, "Sur une courbe qui remplit toute une aire plaine," *Math Ann* , **36**, 1890

[19] J B Rothnie and T Lozano, "Attribute Based File Organization in a Paged Memory Environment," *Communication of the ACM*, 17(2), Feb 1974, 63-69

[20] H Samet, "Hierarchical Representation of Collections of Small Rectangles," *ACM Computing Surveys*, 20(4), December 1988, 271-309

[21] T Sellis, N Roussopoulos, and C Faloutsos, "The R+ Tree A Dynamic Index for Multidimensional Objects," *Proc 13th Int'l Conf on Very Large Databases*, Brighton, U K , Sep 1987, 507-518

[22] Y Tanaka, "Adaptive Segmentation Schemes for Large Relational Database Machines," *Proc 3rd International Conference on Database Machines*, Munich, FRG, 1983