DOT: A Spatial Access Method Using Fractals

Christos Faloutsos†
Yi Rong

Department of Computer Science
University of Maryland at College Park

Abstract

Existing Database Management Systems (DBMSs) do not handle efficiently multi-dimensional data such as boxes, polygons, or even points in a multi-dimensional space. We examine access methods for these data with two design goals in mind: (a) efficiency in terms of search speed and space overhead and (b) ability to be integrated in a DBMS easily. We propose a method to map multidimensional objects into points in a 1-dimensional space; thus, traditional primary-key access methods can be applied, with very few extensions on the part of the DBMS. We propose such mappings based on fractals; we implemented the whole method on top of a B^+-tree, along with several mappings. Simulation experiments on several distributions of the input data show that a modified Hilbert curve gives the best results, even when compared to R-trees [7].

1. Introduction.

Existing Database Management Systems (DBMSs) handle efficiently numbers and character strings, but not multi-dimensional data such as boxes, polygons, or even points in a multi-dimensional space. Multi-dimensional data arise in many applications, including: Cartography [21], Computer-Aided Design (CAD) and VLSI design systems [14], [8], computer vision and robotics [1], traditional databases (a record with k attributes corresponds to a point in a k-d space), rule indexing in expert database systems [20], etc..

Our goal here is to design a method with the following two characteristics:

1) Fast response on geometric queries.
2) Ability to be integrated easily in a DBMS.

We mainly focus on *range queries* (or *region queries*) on a collection of rectangles: Given a rectangular region (user window), find all the objects that intersect it. A special case of the range query is the *point query*: Given a certain point in the space, find all the objects that contain it. To avoid many disk accesses, a good method should ideally store on the same page those rectangles that are geometrically "similar".

We restrict the geometric objects to be rectangles that are aligned with the axes; the reason is that a general geometric shape is frequently represented by its minimum enclosing rectangle.

The main idea in the proposed method is to use two consecutive transformations, to map spatial objects into 1-dimensional points. Thus, we can use ANY primary key access method (e.g. B-tree). For the rest of this work, the method will be called **DOT** (for **DO**uble **T**ransformation).

Good distance-preserving mappings are essential for the performance of the method. We examine space-filling curves, also known as "fractals" [10]. These curves, such as the Peano curve and the Hilbert curve, define a path that traverses the points in a $N \times N$ square grid. These curves can be generalized for higher dimensionality spaces.

The paper is organized as follows: Section 2 gives a brief survey and classification of known spatial access methods. Section 3 describes the proposed approach and alternative distance preserving mappings. Section 4 compares the alternatives using analysis, exhaustive enumeration, as well as simulation with a B^+-tree as the underlying primary-key file structure. It also compares the alternative designs of DOT to R-trees [7], which is one of the main spatial access methods. Section 5 presents the conclusions and future research directions.

---

† Also with University of Maryland Institute for Advanced Computer Studies (UMIACS).

This research was sponsored partially by the National Science Foundation under the grants DCR-86-16833, IRI-8719458 and IRI-8958546 (PYI-89 award).

2. Survey

The problem is to store and retrieve spatial objects on secondary store (disk). As mentioned before, a general object is represented by its minimum enclosing rectangle. Access methods for such rectangles form three classes [18], [19]. We examine the first two in more detail, because they are necessary to describe the proposed DOT method.

Class 1: Methods that transform the rectangles into points in a space of higher dimensionality [9]. For example, a 2-d rectangle with sides parallel to the axes is characterized by four coordinates, and thus it can be considered as a point in a 4-d space. Therefore, secondary key access methods can be used. Figure 2.1 shows an example with line segments (1-d rectangles) and their transformations. The Figure also shows the transformation of a range query, say, (q_{start}, q_{end}): $x_{start} < q_{end}$ and $x_{end} > q_{start}$. Without loss of generality, the (1-d) address space is normalized to the segment (0,1). Note that there are no points outside the triangle {(0,0), (0,1), (1,0)} in the transformed space.

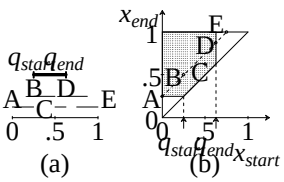

Figure 2.1.
Some line segments (2.1a) and their transformations (2.1b) along with the transformation of the query (q_{start}, q_{end}).

Class 2: Methods that use *space filling curves* to map a k-d space onto a 1-d space. Such methods have been suggested, among others, by Orenstein [11], [12]. The idea is to transform each k-dimensional object to a set of line segments, using Peano's space filling curve [15]. The Peano curve induces an ordering of the k-d points, giving a unique value ($z-value$) to each grid point (pixel). The term $z-ordering$ denotes the way that rectangles are mapped to a set of line segments on the Peano curve. The $z-value$ of a pixel is the binary value of the string that is created by interleaving the binary representations of its x- and y-coordinates. For example, rectangle "A" in Figure 2.2 corresponds to one pixel and has the $z-value$ of 0101; a larger quadrant has as its $z-value$ the common prefix of the $z-values$ of the pixels it contains (eg., rectangle "B" in Figure 2.2 has the $z-value$ of 11). The transformation of a rectangle is a set of $z-values$ (variable-length bit strings), each corresponding to a quadrant that the rectangle completely covers. In Figure 2.2, rectangle "C" has to be split; the corresponding $z-values$ of the pieces are 0010 and 1000 respectively. Overlapping objects can be detected using the so-called "spatial join" [11] which is a simple extension of the natural join. This way minimal changes are required in database systems to support spatial operations.

The only problem with this approach is the proliferation of records to store, because each rectangle is divided into many pieces. The proposed DOT method avoids this problem, as we shall see later.

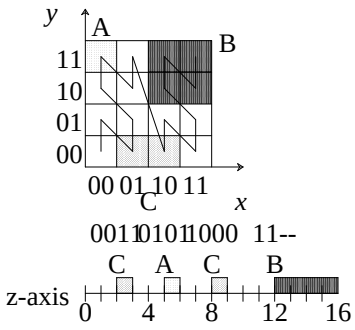

Figure 2.2.
Illustration of the $z-ordering$ method [12].
Shuffling function: the point (x_1x_2, y_1y_2) gives $(x_1y_1x_2y_2)$

Class 3: Methods that divide the original space into appropriate sub-regions. The sub-regions may be overlapping, such as in R-trees [7] [16] or they may be disjoint, such as in the "cell"-trees [6] or in the R^+-trees [19].

3. Proposed approach

The proposed method suggests pipelining the transformations of Class 1 and Class 2 of the survey section. The resulting method enjoys the best of both worlds, avoiding the drawbacks of its individual transformation:

- like the $z-ordering$ of Class 2, it can be easily integrated in DBMS;
- like the members of Class 1, it does not have to divide the rectangles into pieces that fit the quadrants of the address space (while the $z-ordering$ does have to), thus avoiding the proliferation of the records to be stored.

The method works as follows (see Figure 3.1):

Step 0. A spatial object from a k-d space is represented by its minimum enclosing rectangle.

Step 1. The rectangle is transformed into a point in a $2k$ -d space [9]. This transformation is called **first transformation**.

Step 2. A distance preserving mapping [11] is used to map this point to a point in an 1-dimensional space. This transformation is called **second transformation**.

Definitions: The k-dimensional space that the rectangles are in will be called **initial** space; the $2k$-dimensional space resulting after Step 1 will be called **intermediate** space, and the 1-dimensional space after Step 2 will be called **final** space.

The **x-value** of a rectangle is defined as the value of the corresponding point in the final space, when the mapping "x" is used as the second transformation. "x" can be any mapping, eg., as we shall see next, the tri-Hilbert one, the tri-Peano one etc. For example, for the mapping in Figure 3.1, the x-values of segments A and B are 0 and 7 respectively.

In all the drawings, the grid cells in the intermediate space are represented by their upper-left corner (marked by little squares). This means that all the points that fall in a given grid cell are mapped to the upper-left corner of this cell. Of course, this digitization need not compromise the accuracy of the representation: the coordinates of the rectangles are already represented with finite accuracy, and the grid in the intermediate space can have arbitrarily fine granularity.

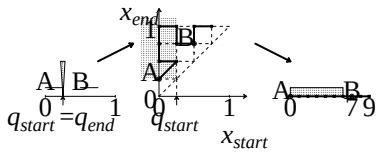

Figure 3.1. Illustration of the proposed approach ("Double Transformation").

Step 0 may be slightly modified, if there is too much "dead space" between the object and its enclosing rectangle, e.g, a concave lake in a map, or a beltway road. With minor changes in the upcoming algorithms, the object can be divided into a few pieces, either manually or automatically. Each of the pieces can be represented by its minimum enclosing rectangle, thus reducing the dead space at the expense of replicating information [13]) Notice that this part is completely optional in the proposed method, as opposed to the Class 2 methods, where the partitioning of each object is mandatory.

The second transformation must preserve the distance as much as possible. We suggest mappings based on the Hilbert curve, which outperformed its competitors in our experiments on 2-, 3- and 4- dimensional grids [4]. Traditional distance preserving mappings apply to square grids; in DOT, the second mapping has to operate on a triangular grid (see middle drawing of Figure 3.1). Therefore we modify the original Hilbert and Peano curves, by ignoring the grid-points below the diagonal and by joining the remaining trails of the curve in ascending **x**-value order. We designed and implemented such mappings for triangular grids. By convention, these triangular mappings will be named after their square counterparts, with the prefix "tri-".

A brief introduction to the traditional space-filling curves is necessary: In a general, k-dimensional space, a space-filling curve starts with path on a k-dimensional grid of side 2. The path visits every point in the grid exactly once without crossing itself. This basic curve is said to be of order 1. To derive a curve of order n, each vertex of the basic curve is replaced by the curve of order $n-1$, which may be appropriately rotated and/or reflected. Figures 3.2 and 3.3 respectively show the traditional Peano and Hilbert curves of order 1, 2 and 3 in the 2-dimensional space.

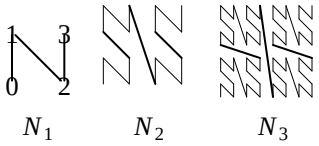

Figure 3.2. Peano curves of order 1, 2, and 3.

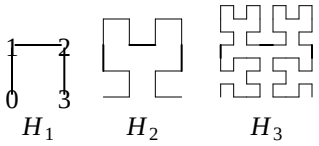

Figure 3.3. Hilbert curves of order 1, 2, and 3.

Figures 3.4, 3.5 and 3.6 show the trails for the the **tri-row**, **tri-Peano** and **tri-Hilbert** curves of order 2 (4×4 grids), as well as for their square counterparts. Note that the tri-column mapping will have the same performance with the tri-row one, if the queries are uniformly distributed.

The calculation of the Peano-value of a point is trivial, by shuffling the binary representations of its coordinates. The calculation of the Hilbert-value of a point in a 2- or higher dimensionality space is more complicated, because the reflection and rotation of the basic pattern

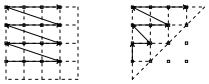

Figure 3.4.
Row and tri-row ordering for
a 4×4 grid.

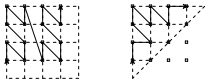

Figure 3.5.
Peano and tri-Peano ordering for
a 4×4 grid.

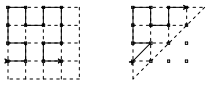

Figure 3.6.
Hilbert and tri-Hilbert ordering for
a 4×4 grid.

has to be accounted for. The algorithm is discussed by Bially [2], where he uses a finite state automaton approach. In a technical report [5] we describe the algorithms that transform a pair of (x,y) coordinates to the corresponding tri-Peano and tri-Hilbert value, as well as the inverse, for the 2-dimensional space.

The algorithm that find the transformation of a query in the final space deserves some discussion. For example, in the setting of Figure 3.1, the (point) query $q_{start}=q_{end}=0.25$ is transformed to the range query $[0,0.25]\times[0.25,1]$ in the intermediate space, which becomes the range [0,6] in the final space. In general, the goal is to find which ranges of the final space contain qualifying elements. The query has to undergo two transformations:

1. In the intermediate space, the query corresponds to the rectangular area in Figure 2.1.
2. In the final space, the query becomes a set of s line segments, say, $\{[l_1, h_1], \cdots, [l_s, h_s]\}$. The resulting set of range queries in the final space can be easily answered by the underlying primary key access method.

The idea of the second step is that the intermediate space is recursively divided and each of the pieces is checked against the query region [5] Each recursive call returns a set of ranges of the final space; the union of these sets (after collapsing consecutive ranges to one) is the result.

All the examples we have presented are for 2-dimensional (intermediate) spaces. This is done just for the purpose of illustration; the proposed space filling curves can be generalized for higher dimensionality (intermediate) spaces. Thus, the DOT method can be applied to any dimensions of initial space.

4. Performance Results

The setting we have in mind is as follows: Consider a set of k-dimensional rectangles; each is assigned a value (x-value) according to the proposed DOT method; the records of the rectangles are stored in a B^+-tree, using the x-values as primary keys. Note that not all possible x-values have to be present; only the existing points of the final space are stored. In this setting, the best measure for the response time is the number of disk accesses that the average range query requires.

The only disadvantage of this measure is that it depends not only on the mappings used, but also on the capacity of the disk pages, the number and distributions of rectangles etc.. A measure that depends only on the mapping is the number of *clusters* that the average query retrieves. For a given second transformation "x", a **cluster** is defined to be a group of points with consecutive "x"-values. The proposed measure is a good indication of how good a clustering a distance-preserving mapping can achieve: If a range query retrieves few clusters, then it is likely to require few disk accesses on the actual file. For example, the query of Figure 3.1 retrieves only one cluster (namely, the one with endpoints [0,6]).

We have ran two types of experiments, using an 1-dimensional initial space (i.e., the rectangles are line segments), with a 2-dimensional intermediate space. First, we examined the average number of clusters that alternative space-filling curves result into, by exhaustive enumeration or analysis. Then, we measured the number of disk accesses under several input data distributions, for the R-trees and for alternative designs of the DOT method.

For the tri-row ordering, we have derived the number of clusters analytically: If N is the size of the side of the grid, the average number of clusters for all possible range queries is given by the formula

$$C_{all} = \frac{\sum_{i=0}^{N-1}\sum_{j=i}^{N-1}(N-j-1)+N}{N(N+1)/2} \qquad (1)$$

The average number of clusters for all possible point queries is:

$$C_0 = (N-1)/2+1/N \qquad (2)$$

Table 4.1 shows the average number t of clusters for all

possible range queries for the tri-row ordering, the tri-Peano curve and the tri-Hilbert curve. Table 4.2 shows the same measure for point queries only. In both cases, the tri-Hilbert curve is the best, retrieving approximately half as many clusters as its competitors.

Order *n*	tri-row	tri-Hilbert	tri-Peano
1	1.00	1.00	1.00
2	1.40	1.20	1.30
3	2.56	1.78	2.33
4	5.12	3.06	4.79
5	10.39	5.70	10.00
6	21.03	11.02	20.59

Table 4.1
Average number of clusters for
all possible range queries

Order *n*	tri-row	tri-Hilbert	tri-Peano
1	1.00	1.00	1.00
2	1.75	1.50	1.75
3	3.63	2.50	3.63
4	7.56	4.50	7.56
5	15.53	8.50	15.53
6	31.52	16.50	31.52

Table 4.2
Average number of clusters for
all possible point queries

Having the above strong indication that the tri-Hilbert curve should perform well, we also run more realistic, simulation experiments to measure the actual number of disk accesses. We used a pseudo-random number generator and we created line segments in the initial space, such that their transformations form three types of distribution in the intermediate space:

1. The first type is the "strip" distribution (abbreviation S.) shown in Figure 4.1(a). It corresponds to the case where there are many, small rectangles; then, the transformed points in the intermediate space are located in a narrow strip above the diagonal. It seems that this distribution appears often in practical applications [17].
2. The second distribution is called "double-strip"(abbreviation D.S.); this is the case where there are many small rectangles with a few large rectangles within certain sizes (see Figure 4.1(b)). We feel this is a practical distribution, too: For example, in a VLSI design, there are many small gates and a few long busses.
3. The third distribution is "strip plus uniform"(abbreviation S+U); it corresponds to the case where there are many small rectangles with a few large rectangles in uniform random sizes (Figure 4.1(c)).

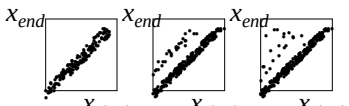

Figure 4.1
Three types of data distribution

Using each of our triangular space-filling curves, we mapped these points into one-dimensional points and stored them in a B^+ tree (one B^+ tree for each triangular mapping with each data set). The leaf and internal node capacity of the B^+ tree with page size 512 are 25 and 63. The leaf and internal node capacity of the B^+ tree with page size 1024 are 46 and 127. A 128 X 128 grid is used for each data set with 1000 line segments. A 256 X 256 grid is used for each data set with 5000 line segments. A 512 X 512 grid is used for each data set with 10000 line segments.

To compare our methods with R-tree, we also stored the line segments into a R-tree. We chose a R-tree that uses linear split algorithm with nodes at least half full, because the search performance of a R-tree is not very sensitive to the choice of the split algorithm [7]. For the experiment, the root of the B^+ tree and the R-tree were assumed to be permanently stored in main memory. Tables 4.3 and 4.4 show the average number of pages touched per qualifying record, for all possible point queries in the initial space (which has N distanct points) with page size 512 bytes and 1024 bytes, respectively. The results are calculated by the formula

$$t = \frac{\sum_{i=1}^{N} (\text{number of pages read})}{\sum_{i=1}^{N} (\text{number of qualifying records})}$$

In all cases, the tri-Hilbert curve behaves better than its competitors, confirming our intuition and the preliminary experiments of Tables 4.1 and 4.2. In our experiments, the DOT method using the tri-Hilbert curve achieved consistently better results over the R-tree (up to more than 50% savings) and over the tri-Peano-based DOT version. The tri-Peano curve is better than the tri-row curve.

N	data distri.	R-tree	B^+ tree		
			tri-H	tri-P	tri-R
1000	S	0.183	0.133	0.182	0.522
1000	D.S	0.128	0.087	0.115	0.247
1000	S+U	0.156	0.092	0.111	0.247
5000	S	0.125	0.106	0.142	0.880
5000	D.S	0.103	0.074	0.091	0.357
5000	S+U	0.101	0.077	0.094	0.351
10000	S	0.121	0.100	0.130	1.315
10000	D.S	0.099	0.065	0.074	0.381
10000	S+U	0.103	0.069	0.079	0.358

Table 4.3
Pages touched per qualifying record,
for all possible point queries
with page size 512 bytes.

N	data distr.	R-tree	B^+ tree		
			tri-H	tri-P	tri-R
1000	S	0.140	0.085	0.108	0.211
1000	D.S	0.121	0.061	0.075	0.135
1000	S+U	0.121	0.055	0.078	0.137
5000	S	0.084	0.049	0.064	0.257
5000	D.S	0.070	0.035	0.042	0.114
5000	S+U	0.078	0.036	0.043	0.110
10000	S	0.072	0.058	0.075	0.501
10000	D.S	0.058	0.041	0.047	0.216
10000	S+U	0.060	0.044	0.053	0.223

Table 4.4
Pages touched per qualifying record,
for all possible point queries
with page size 1024 bytes.

5. Conclusions - Future research.

The main contribution of this work is the proposal of the DOT method, which maps k-dimensional rectangles to 1-dimensional points. Thus, any primary key access method can be used to handle spatial data; the only functions that are needed are (a) the function that maps a k-d rectangle to an 1-d point and (b) the function that maps a k-d query to a set of 1-d ranges. These functions are a few hundred lines of C code long. The rest of the complexity (insertion, deletions, locking, concurrency etc.) is handled by the underlying primary-key data structure, such as a B-tree. The additional advantages of using a DOT on a B-tree are that

(a) the space utilization is bounded

(b) the B-tree will adapt itself automatically to skewed distributions of data.

(c) the B-tree code is readily available in every DBMS

Thus, the method can be easily integrated in any relational DBMS, enhancing it with the ability to handle spatial objects.

We have mainly focused on rectangular queries; however, queries of arbitrary shape can be handled easily, by dividing the shape into rectangle (e.g, in a quadtree fashion).

Additional, minor contributions are:

1) the experimentation among the space filling curves, which pinpoints that the Hilbert and tri-Hilbert curves are the most promising methods.

2) the implementation of the DOT method (with all its alternative designs) on a B^+-tree; this shows the applicability of the method on a real DBMS.

3) The simulation comparison of the DOT method against the R-trees, which showed that the proposed method consistently outperforms the R-trees.

Future work includes:

1) Comparison of the DOT method with other spatial access methods. such as the R^+-trees [3], [19], the z-ordering method [12] e.t.c.

2) Analytical study of the space filling curves to find the one with the best clustering properties (or to prove that the Hilbert curve is the best).

Acknowledgements: The authors would like to thank Jiang-Hsing Chu for providing the B^+-tree package, as well as for his help with the experiments.

References

1. Ballard, D. and C. Brown, *Computer Vision*, Prentice Hall, 1982.

2. Bially, T., ‘‘Space-Filling Curves: Their Generation and Their Application to Bandwidth Reduction,’’ *IEEE Trans. on Information Theory*, vol. IT-15, no. 6, pp. 658-664, Nov. 1969.

3. Faloutsos, C., T. Sellis, and N. Roussopoulos, ‘‘Analysis of Object Oriented Spatial Access Methods,’’ *Proc. ACM SIGMOD* , pp. 426-439, San Francisco, CA, May 27-29, 1987. also available as SRC-TR-87-30, UMIACS-TR-86-27, CS-TR-1781.

4. Faloutsos, C. and S. Roseman, ‘‘Fractals for Secondary Key Retrieval,’’ *Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of*

Database Systems (PODS), pp. 247-252, Philadelphia, PA, March 29-31, 1989. also available as UMIACS-TR-89-47 and CS-TR-2242

5. Faloutsos, C. and Y. Rong, ‘‘Spatial Access Methods Using Fractals: Algorithms and Performance Evaluation.,’’ UMIACS-TR-89-31, CS-TR-2214, Univ. of Maryland, Dept. of Computer Science, College Park, Feb. 1989.
6. Gunther, O., ‘‘The Cell Tree: An Index for Geometric Data,’’ Memorandum No. UCB/ERL M86/89, Univ. of California, Berkeley, Dec. 1986.
7. Guttman, A., ‘‘R-Trees: A Dynamic Index Structure for Spatial Searching,’’ *Proc. ACM SIGMOD*, pp. 47-57, Boston, Mass, June 1984.
8. Guttman, A., ‘‘New Features for Relational Database Systems to Support CAD Applications,’’ PhD Thesis, University of California, Berkeley, June 1984.
9. Hinrichs, K. and J. Nievergelt, ‘‘The Grid File: A Data Structure to Support Proximity Queries on Spatial Objects,’’ *Proc. of the WG’83 (Intern. Workshop on Graph Theoretic Concepts in Computer Science)*, pp. 100-113, Trauner Verlag, Linz, Austria, 1983.
10. Mandelbrot, B., *Fractal Geometry of Nature*, W.H. Freeman, New York, 1977.
11. Orenstein, J., ‘‘Spatial Query Processing in an Object-Oriented Database System,’’ *Proc. ACM SIGMOD*, pp. 326-336, Washington D.C., May 1986.
12. Orenstein, J.A. and F.A. Manola, ‘‘PROBE Spatial Data Modeling and Query Processing in an Image Database Application,’’ *IEEE Trans. on Software Engineering*, vol. 14, no. 5, pp. 611-629, May 1988.
13. Orenstein, J.A., ‘‘A Comparison of Spatial Query Processing Techniques for Native and Parameter Spaces,’’ *Proc. of ACM SIGMOD conf.*, pp. 343-352, Atlantic City, New Jersey, 1990.
14. Ousterhout, J. K., G. T. Hamachi, R. N. Mayo, W. S. Scott, and G. S. Taylor, ‘‘Magic: A VLSI Layout System,’’ *21st Design Automation Conference*, pp. 152 - 159, Alburquerque, NM, June 1984.
15. Peano, G., ‘‘Sur une courbe qui remplit toute une aire plaine,’’ *Mathematische Annalen*, vol. 36, pp. 157-160, 1890.
16. Roussopoulos, N. and D. Leifker, ‘‘Direct Spatial Search on Pictorial Databases Using Packed R-Trees,’’ *Proc. ACM SIGMOD*, Austin, Texas, May 1985.
17. Samet, H., ‘‘The Quadtree and Related Hierarchical Data Structures,’’ *ACM Computer Surveys*, vol. 16, no. 2, pp. 187-260, June 1984.
18. Samet, H., ‘‘Hierarchical Representations of Collections of Small Rectangles,’’ *ACM Computing Surveys*, vol. 20, no. 4, pp. 271-309, Dec. 1988.
19. Sellis, T., N. Roussopoulos, and C. Faloutsos, ‘‘The R+ Tree: A Dynamic Index for Multi-Dimensional Objects,’’ *Proc. 13th International Conference on VLDB*, pp. 507-518, England,, Sept. 1987. also available as SRC-TR-87-32, UMIACS-TR-87-3, CS-TR-1795
20. Stonebraker, M., T. Sellis, and E. Hanson, ‘‘Rule Indexing Implementations in Database Systems,’’ *Proceedings of the First International Conference on Expert Database Systems*, Charleston, SC, April 1986.
21. White, M., *N-trees: Large ordered Indexes for Multi-dimensional Space*, Application Mathematics Research Staff, Statistical Research Division, U.S. Bureau of the Census, Dec. 1981.