ELSEVIER Theoretical Computer Science 181 (1997) 3–15

Theoretical Computer Science

Space-filling curves and their use in the design of geometric data structures

Tetsuo Asano[a], Desh Ranjan[b], Thomas Roos[c,*], Emo Welzl[d], Peter Widmayer[c]

[a] *Department of Applied Electronics, Osaka Electro-Communication University, 18-8 Hatsu-cho, Neyagawa 572, Japan*
[b] *Department of Computer Science, New Mexico State University, Las Cruces, NM, USA*
[c] *Theoretische Informatik, ETH Zentrum, 8092 Zürich, Switzerland*
[d] *Institut für Informatik, Freie Universität Berlin, Takustrasse 9, 14195 Berlin, Germany*

Abstract

We are given a two-dimensional square grid of size $N \times N$, where $N := 2^n$ and $n \geqslant 0$. A space filling curve (SFC) is a numbering of the cells of this grid with numbers from $c+1$ to $c+N^2$, for some $c \geqslant 0$. We call a SFC recursive (RSFC) if it can be recursively divided into four square RSFCs of equal size.

We prove several useful and interesting combinatorial properties of recursive and general SFCs. For an optimality criterion that is important in the design of geometric data structures, we propose a RSFC that is optimal in the worst case and outperforms the previously known RSFCs.

1. Introduction

Data structures for maintaining sets of multidimensional points on external storage play an important role in many non-standard database systems where query performance is still a major bottleneck. In contrast to the one-dimensional case, the situation in higher dimension is far more complex, since there is no obvious total order on the points that serves all purposes.

The most frequent type of queries on these multidimensional point sets are range queries: Given the set of points and a multidimensional query interval, report all points lying in the interval. The query intervals (or aligned rectangles, as they are often called) may have arbitrary size, aspect ratio, and position. An obvious possibility for a multidimensional data structure is to map multidimensional points to one dimension, and to maintain the resulting one-dimensional points in one of the well-known data structures (such as, for instance, B^+-trees).

* Corresponding author. Email: roos@inf.ethz.ch.

The mapping should be such that the query corresponding to the image of a multidimensional range in the given, original space can be supported by a data structure in image space. The ideal mapping (that does not exist) would map each multidimensional interval into a one-dimensional interval containing precisely the images of the corresponding points. With the goal of approximating this ideal in some way, several mappings have been proposed. The most prominent ones include the *z-order* [13, 15–17], the *Gray code* [6, 7], and *Hilbert's curve* [8, 12]. The z-order is also known as *Morton encoding* [14], *bit interleaving* [19], *quad code* [9] or *locational code* [2].

The various embeddings of multidimensional space into one dimension have mostly been studied experimentally; there are, however, theoretical studies with a different focus [18]. It seems that z-order and Hilbert's curve are the best known numbering schemes in general [1, 12, 20]. They typically map a multidimensional interval into quite a large number of one-dimensional intervals. In order to achieve a better range query efficiency, it may be worthwhile to map a multidimensional interval into *as few as possible* one-dimensional intervals. The reason is that the time for moving the read-write-head of a disk to the proper track, the *seek time*, is typically much higher than *latency* or *transfer* time spent in waiting for the proper block to pass by the head and in transferring the block between internal and external storage, and that for each one-dimensional interval of blocks, only one disk seek operation is needed. As a consequence, data structures that aim at minimizing the number of seek operations (and not the number of block accesses) have been proposed [10, 11].

In this paper, we consider the problem of finding a mapping from multidimensional space into one dimension in such a way that the number of seek operations for the image of a range query is low. To keep the number of disk seek operations low, we tolerate the transfer of a number of unnecessary blocks, when a range query is answered. We put, however, a bound on this number: For each seek operation, the number of unnecessary blocks that are transferred to avoid extra seeks is not allowed to exceed the number of necessary block transfers by more than a constant factor. In our study, we investigate the proximity preservation properties of such mappings, defined to suit the above purpose.

In the next section, we lay the foundations for our results, including the cost model. Section 3 shows that when the size of the query range is fixed in advance, a space-filling curve can be defined that answers each query with no more than two disk seek operations. Section 4 looks at recursively defined space-filling curves and square queries of varying sizes; it concludes that three disk seek operations are necessary and sufficient in this case.

2. Foundations

We are given a number $N = 2^n$, $n \geqslant 0$, and a two-dimensional square grid $\langle 1 \ldots N, 1 \ldots N \rangle$ of size $N \times N$ with N^2 cells. A *numbering* $\mathscr{P}$ of this grid is a one-to-one mapping

$$\mathscr{P}: N \times N \to \{1,\dots,N^2\}.$$

The *distance* between two cells (i,j) and (i',j') in the numbering $\mathscr{P}$ is defined to be $|\mathscr{P}(i,j) - \mathscr{P}(i',j')|$. We can display the numbering by connecting any two cells (i,j) and (i',j') via an edge iff they have consecutive numbers in a numbering $\mathscr{P}$. This results in a graph in which each node represents a cell and the distance between nodes (u,v) and (u',v') is the same as the distance in the $\mathscr{P}$ numbering. So, one can think of these numberings as curves that cover all cells of a two-dimensional array. This motivates the term *space-filling curves* (*SFC*) that we often use instead of *numbering*.

We call a SFC $\mathscr{P}$ with numbers $c+1,\dots,c+N^2$, for some $c \geqslant 0$, *recursive* (RSFC) if $N=1$ or $\mathscr{P}$ can be divided into four square RSFCs $\mathscr{P}_0,\dots,\mathscr{P}_3$ of equal size in the following way. There exists a permutation $\pi: \{0,\dots,3\} \to \{0,\dots,3\}$ such that $\mathscr{P}_{\pi(i)}$ is a RSFC containing the numbers

$$\left\{c + \frac{iN^2}{4} + 1, \dots, c + \frac{(i+1)N^2}{4}\right\}$$

for $i = 0,\dots,3$.

RSFCs have several advantages over SFCs in general; e.g., they allow a compact representation of all numbers without storing the entire SFC explicitly. In fact, the number $\mathscr{P}(i,j)$ assigned to a specific cell (i,j) on the grid can be determined in $O(\log N)$ time. As a matter of fact, all well-known RSFCs, as e.g. the Hilbert curve, the z-curve, or the Gray code are generated by very regular permutation rules, as we shall see in the following.

A SFC $\mathscr{P}$ is called *closed* if

$$\|\mathscr{P}^{-1}(i+1) - \mathscr{P}^{-1}(i)\|_\infty = 1 \quad \text{for all } i = c+1,\dots,c+N^2-1,$$

where $\| \ \|_\infty$ denotes the maximum norm (Chebyshev distance L_∞).

For each cell of the grid, we define the *k-neighborhood*, $k \in \{0,\dots,N-1\}$, as the set of the at most $(2k+1)^2$ cells that can be reached from the current cell in a L_∞-distance of at most k. As a consequence, in a closed SFC, any two cells that are adjacent on the curve lie in the 1-neighborhood of each other. For any given k, a cell is called a *k-interior* cell iff its k-neighborhood does not contain boundary cells of the grid.

The construction of RSFCs can be visualized easily by means of a grammar. A variable of the grammar represents a cell. We start with a variable that initially represents the area of the entire $N \times N$ grid as one cell. The grammar is an E0L-type (extended zero-sided Lindenmayer [22]), that is, we force rewriting to take place simultaneously at every cell of the partition, and each syntactic variable is a terminal symbol of the grammar at the same time.

Hence, we can view the rewriting process as going through a number of iterations, where each iteration rewrites all cells. In the i-th iteration, $i = 0,\dots,n-1$ (recall that $N = 2^n$), we get a partition of the area of the $N \times N$ grid into $2^i \cdot 2^i$ blocks, each of size $2^{n-i} \times 2^{n-i}$. In each iteration, each block is replaced by four blocks

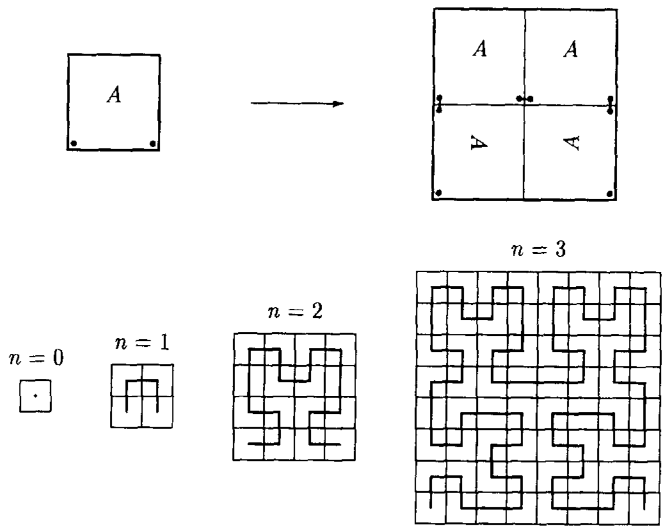

Fig. 1. The Hilbert production and the Hilbert curves for $n = 0, \ldots, 3$.

of size $2^{n-i-1} \times 2^{n-i-1}$ according to an applicable production of the grammar. Our grammar allows to rotate blocks (displayed by a rotation of the corresponding symbols ⋖, ∀, ⋗).

Two dots denote the entry and exit points of the curve into a cell. In a cell that is not refined further, both dots coincide at the center of the cell; in each refinement, the dots lie at the corresponding corner cells of the refined partition. Finally, lines between dots on the right-hand side of the productions represent subsequent blocks in the numbering. Figs. 1, 2 and 3 display the productions and the first few iterations of the Hilbert curve, the z-curve, and the Gray code, respectively. Notice that a single refinement rule is sufficient for each of them.

2.1. The Model

Recall that one important application of SFCs is their use as a data structure for embedding multidimensional points into one dimension while preserving spatial proximity to the largest possible extent. Data structures for points partition the data space into cells. All points that lie in one cell of the space partition are stored on one disk block. Because the disk block is the atomic entity for disk accesses, one might say that locally, i.e. within a cell, this preserves proximity in space also on disk. The predominant reason for aiming at proximity preservation is the efficiency of range queries. A range query asks for all points in the data structure that lie inside a given query range.

Spatial data structures for external memory process a range query in two steps. The first step retrieves a superset of the response from disk, by reading those blocks whose cells intersect the query range. The second step examines the retrieved points and partitions them into two subsets, the points that lie in the query range (hits)

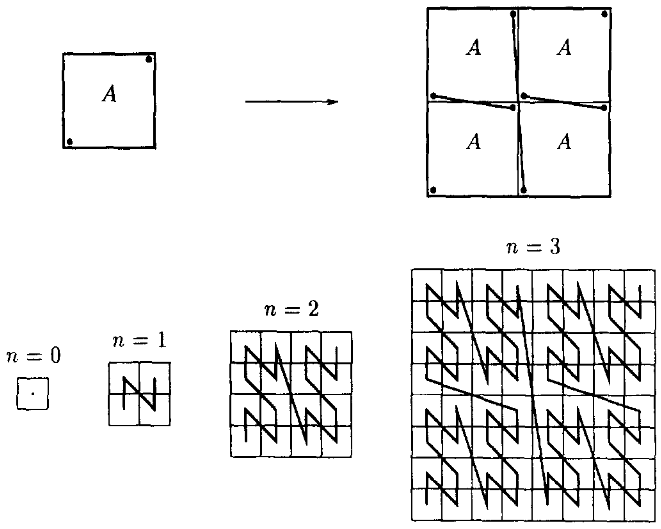

Fig. 2. The z-curve production and the z-curves for $n = 0,\ldots,3$.

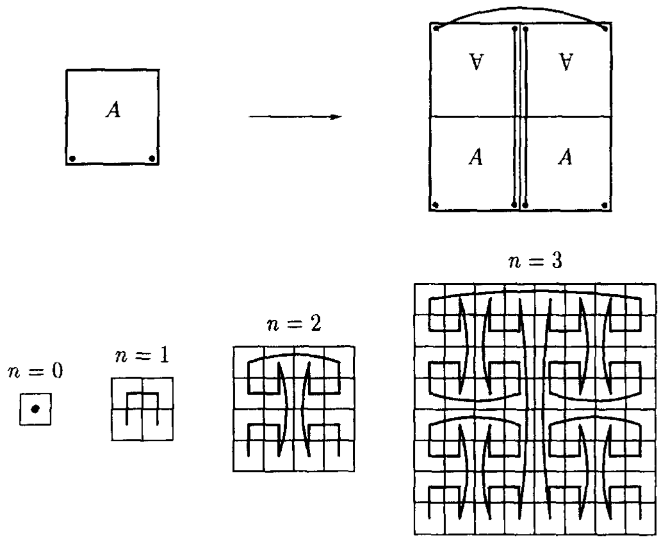

Fig. 3. The Gray code production and the Gray code curves for $n = 0,\ldots,3$.

and those that do not (false drops). Our interest is solely in the first step, because disk accesses are far more costly than internal memory operations.

The time needed to answer a range query does not only depend on the number of disk blocks retrieved, but also – and even more so – on their physical location on the disk. A few blocks in consecutive locations can be accessed with almost the same cost as one block, because the disk arm positioning time (disk seek time) exceeds the time for a block read operation by far. Because a rectangular query range is typically much larger than one cell, we are interested in an ordering of the cells that extends the local proximity preservation beyond disk blocks. A cell number in the ordering will then be used directly as the physical disk block address. Since there is no proximity preserving total ordering of points into two (or more) dimensions, we aim at a best possible mapping of two-dimensional points to one dimension. The mapping should preserve proximity in the sense that neighboring cells should ideally correspond to neighboring blocks on disk. For simplicity, let us view a disk as a linear sequence of blocks. In the ideal case, any two-dimensional rectangular query region should be mapped by the SFC to one disk block interval. However, this is not possible, as we will see in the next section.

Let us restrict our view to range queries for which the union Q of all cells intersecting the query square is a square. Let k^2 be the number of cells of the $k \times k$ square Q. Since we aim at keeping the number of disk seek operations low, we decide to allow the redundant transfer of a constant proportion of all blocks queried. More precisely, for some fixed constant C, up to Ck^2 blocks may be transferred in response to a query. The reason here is that it is faster to let a few blocks pass by the disk head than to perform another disk seek operation. In the following sections, we prove lower and upper bounds on the number of seek operations, and we design optimal classes of SFCs that minimize this number.

3. Fixed-size queries

The question of preserving proximity of SFCs has been studied by Asano et al. [4]. They answered the question of how many neighbors of each cell can be drawn in the SFC within a given constant distance d in the numbering.

Lemma 1 (Asano et al. [4]). *For any given d, there exist N and k such that for all numberings of the $N \times N$ grid, there is a k-interior cell such that at least $(2k-1)(k+1)$ of the k-neighbors are at a distance greater than d from the cell.*

For any given N and k, there exist $d \in \Theta(k^2)$, independent of N, and a numbering such that each k-interior cell contains at least $k(2k+3)+1$ out of its $(2k+1)^2$ k-neighbors within distance d of the cell's number in the numbering.

A *snake curve* of height k is defined by partitioning the $N \times N$ grid into horizontal stripes of height k (for simplicity let k divide N). Each of these stripes is covered by a snake-like curve of the type depicted in Fig. 4. Now, let Q be any $k \times k$ window. It is an easy exercise to show that a snake curve of height k (if k is odd) or $k+1$ (if k is even) results in two seek operations in the worst case.

On the other hand, two seek operations are already optimal, because for any numbering, there is a path of length at most $N-1$ on the grid that connects the cells with the numbers 1 and N^2, and therefore there always exist two directly neighboring cells whose numbers differ by at least $N+1$. Now, if we query these two cells with a 2×2 window, we obviously need at least two seek operations in our model of computation. We state this result as follows.

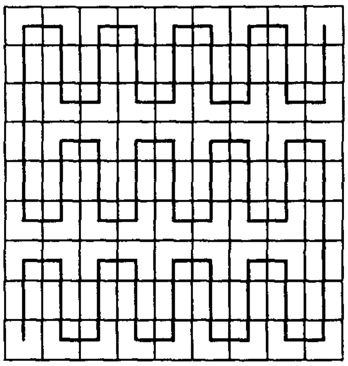
Fig. 4. The snake curve.

Theorem 2. *For fixed k, there exists a closed SFC such that each square query region requires only* 2 *seek operations in the worst case. This is optimal.*

4. Variable-size queries

In this section, we define a closed RSFC that outperforms all known RSFCs with respect to the number of seek operations required in the worst case. We use an E0L-type grammar to describe the new class. In addition, we give a lower bound of three seek operations for RSFCs, thereby proving that the new RSFC is optimal. In contrast, all previously known RSFCs require at least four seek operations in the worst case. To prove the lower bound for RSFCs, we consider a partitioning step with an $N \times N$ grid which is divided into four parts, each of size $N/2 \times N/2$. There always exist two parts that share a common boundary, with their numbers having a distance of $\Theta(N^2)$; the distance between two parts is defined as the smallest of the absolute values of the difference between two numbers, one from each part.

We inspect the boundaries at which two of these distant parts touch (see Fig. 5). For the sake of contradiction, assume that there is an RSFC and a constant C such that for all square queries it is possible to access all relevant information with at most two seek operations (one above and one below the separating line), thereby reading at most Ck^2 blocks.

Look at a 2×2 query window that is cut by the separating line (see Fig. 5). The two cells below the separating line lie within a distance of $4C$. By placing the 2×2 window at each possible position where the separating line cuts through it, we conclude by transitivity that all $N/2$ cells in the entire row below the line have a pairwise distance of at most $4C(N/2 - 1) = 2CN - 4C$.

Next, look at all windows of size $(k+1) \times (k+1)$, with one row above the line and k rows below. By placing the window at each such position, it covers a total of $kN/2$ cells below the line. Within each position of the $k \times (k+1)$ window below the line,

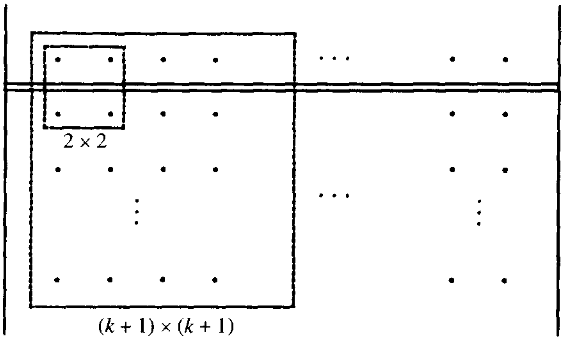

Fig. 5. Illustrating the proof of the lower bound.

the maximum pairwise distance of two cell numbers is $C(k+1)^2$. Since the row below the separating line belongs to the area covered by the $(k+1)\times(k+1)$ windows, the cell numbers in the $k \times N/2$ stripe can deviate from those in the top row by at most $C(k+1)^2$ upwards and downwards in the numbering. Hence, by transitivity we get that the maximum distance between any two numbers in the $k \times N/2$ stripe is at most $2CN - 4C + 2C(k+1)^2$. But, since $2CN - 4C + 2C(k+1)^2 < k\ N/2$ for sufficiently large k and N, we get a contradiction. Thus, *three* seek operations are necessary in the worst case.

Next, we outline a class of RSFCs that achieve this bound. However, before that, we prove that every RSFC achieves already an upper bound of *four* seek operations. This can be seen by the following observation. For each $k \times k$ query window Q, we select the minimal i with $k \leqslant 2^i$ and consider the partitioning of the $2^n \times 2^n$ grid into blocks of size $2^i \times 2^i$. Obviously, Q is covered by at most 4 blocks of size $2^i \times 2^i$. Thus, four seek operations with at most

$$4 \cdot 2^i \cdot 2^i < 2^{2\log k+4} = 16k^2$$

grid cells in total suffice to read all elements of Q. This proves that C can be selected as 16.

To improve the number of seek operations, we design a RSFC with the following property. For any point in which four blocks of size $2^{n-i} \times 2^{n-i}$ at the i-th level of the refinement meet, we guarantee two of the four blocks to be direct neighbors in the numbering of the RSFC. This brings the number of seek operations down to *three* in the worst case.

We can prove that the presented class of RSFCs (see Figs. 6 and 7) has the desired property by induction on the level of the refinement.

Theorem 3. *For variable-size square query regions, there exists a recursive and closed RSFC such that each query requires only* 3 *seek operations in the worst case. This is optimal for RSFCs.*

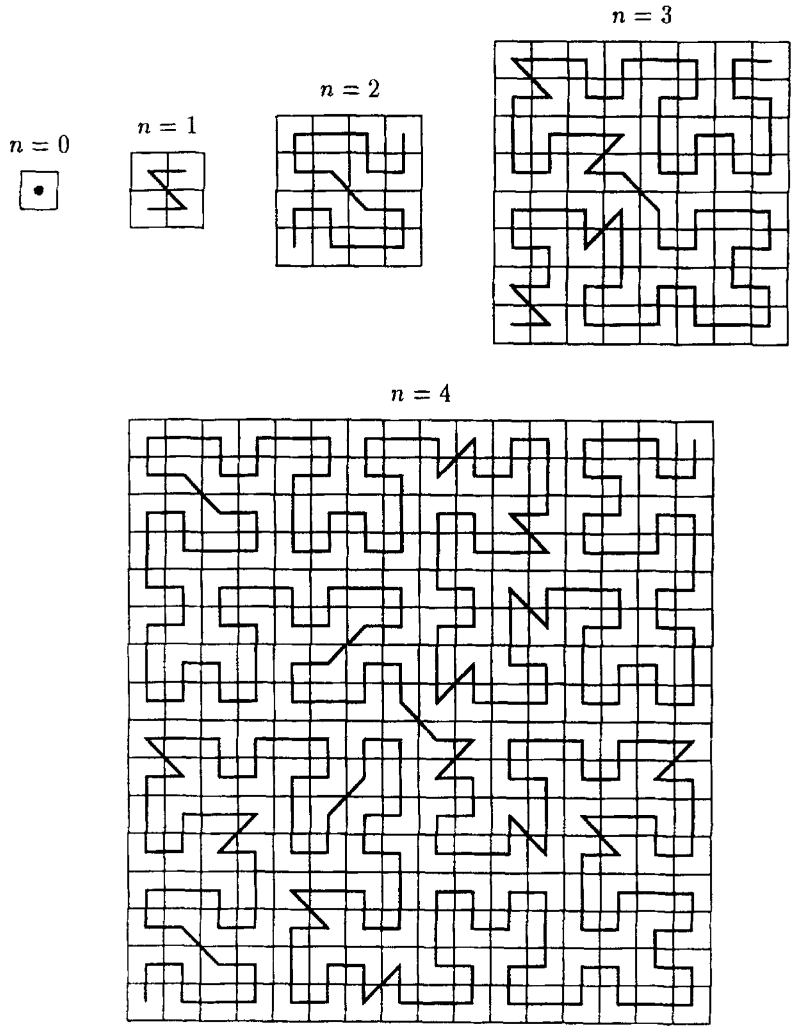

Fig. 6. Our query optimal RSFC starting with variable $A1$.

Proof. To prove the theorem, we present a RSFC that has the desired properties. Our RSFC is given by the eight productions in Fig. 7. Even though there is an inherent symmetry in these productions that makes it possible to describe the RSFC with only four productions and a *flip*-operation, we chose to describe it with this larger, but more explicit and more readily understood set of productions.

First of all, the application of the productions $\mathscr{R} := \{A1, A2, B1, \ldots, D2\}$ to any start variable defines a closed RSFC. Recall that the i-th level of refinement of the RSFC, $i = 0, \ldots, n$, decomposes the RSFC into blocks of size $2^{n-i} \times 2^{n-i}$ thus generating $(2^i - 1)^2$ points where four of these blocks meet.

For convenience, we call such a point a *vertex* and assign it to a unique level, namely the lowest level where it appears. A vertex is called a *transition vertex* if in

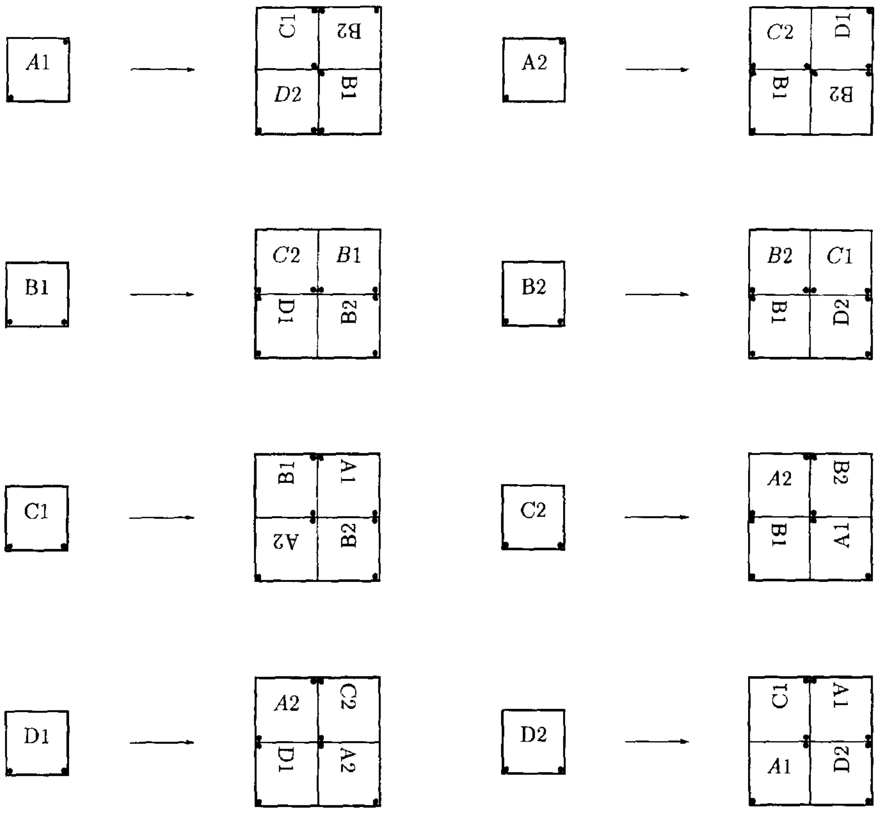

Fig. 7. The productions of our query optimal RSFC.

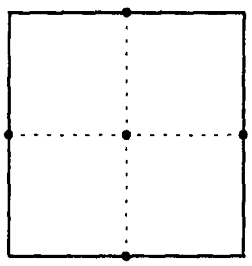

Fig. 8. The feasible placements of a vertex with respect to the previous level of refinement.

its level and all levels with higher index, at least two of the four blocks adjacent to the vertex are direct neighbors in the RSFC.

At first, it is easy to see that all vertices of level $i = 1$ and $i = 2$ are transition vertices for any start variable. Now, any vertex v of level $i \geqslant 3$ lies either in the center or on the boundary of some block B of level $i - 1$ as shown in Fig. 8.

In the first case, v is a transition vertex, because all productions in $\mathscr{R}$ generate central points which are transition vertices. In the second case, v lies on the boundary of B. In

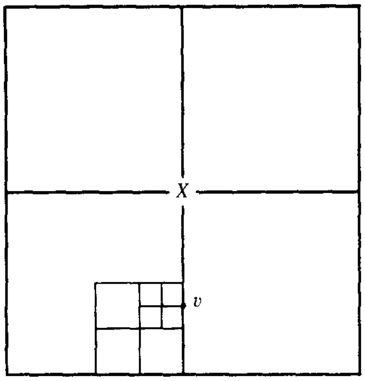

Fig. 9. Illustrating the construction.

fact, v may also lie on the boundary of two neighboring blocks of lower levels. Thus, we consider the lowest level $j \leqslant i-1$ that has this property. Therefore, v lies on the common boundary of two neighboring blocks of level j which are both contained in a block of level $j-1$ which is represented by a variable $X \in \mathscr{R}$ (cf. Fig. 9).

Therefore, we have reduced our problem to the one of proving that for any start variable of $\mathscr{R}$, all vertices in all subsequent levels that lie on one of the two bisecting lines of the *first* level with respect to X are transition vertices. This can be shown by looking at the common boundary of each pair of adjacent cells that occur in some derivation, starting from an arbitrary cell. For instance, within the first production, we see that the lower boundary of $C1$ touches the right boundary of $B2$. It is, however, not sufficient to merely consider the set of boundary pairs occurring directly in the productions, because the repeated application of the productions creates new boundary pairs. To see this, consider the application of the productions for $\rotatebox{90}{$C1$}$ and $\rotatebox{180}{$B2$}$ to the right-hand side of the production for $A1$. This yields the boundary pairs $\rotatebox{90}{$A2$}|\rotatebox{180}{$C1$}$ and $\rotatebox{180}{$B2$}|\rotatebox{90}{$D2$}$. Both of these pairs are new, because they do not appear anywhere within the set of productions. An exhaustive application of the productions, however, shows that already after three derivations a fixpoint is reached in the set of pairs of touching boundaries.

We will now prove that all vertices are transition vertices by looking in more detail at the touching boundaries. Some of the vertices are recognized as transition vertices by looking only on one side of a region boundary that results from a derivation. Fig. 10 shows all patterns of transition vertices that appear on one side of a region boundary after up to three derivations. The transition vertices are shown as dots in Fig. 10; their connections on one side of the boundary are shown as line segments. The first pattern is marked with $A1$, $\underline{\text{I}V}$, ..., indicating that it appears after three derivations of the upper boundary of $A1$, the lower boundary of $A1$, etc.

There are only four different boundary characteristics, named (a)–(d) in Fig. 10, for all productions and all orientations of cells. We finally show that boundaries touch only

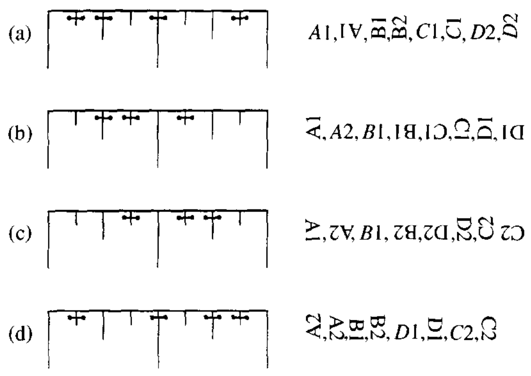

Fig. 10. The four different boundary characteristics.

in a way that guarantees for each vertex on each side of the boundary to be a transition vertex. The only pairs of boundary characteristics that are generated by the productions are the pairs ((a), (b)) and ((c), (d)). These are exactly the pairs of boundary characteristics that make each vertex on a boundary a transition vertex. This proves that all productions of $\mathscr{R}$ generate only transition vertices, thus concluding the proof. □

5. Final remarks

The proposed RSFC guarantees an upper bound of three disk seek operations for a square range query in the worst case, under the assumption that a constant proportion of extra disk blocks may be accessed whenever this helps to reduce the number of seek operations. This result is optimal under the proposed condition for all RSFCs. It does not imply superiority of the RSFC over other RSFCs, such as Hilbert's curve, in all practical situations; it is rather an asymptotic worst case result. In addition, it does not imply optimality with respect to non-square range queries: Whenever the aspect ratio of query rectangles is fixed and known beforehand, one should try to use a specifically adapted RSFC. Our results also do not imply optimality for other spatial operations, such as ray shooting, nearest-neighbor queries, and the like. Nevertheless, we are optimistic that our RSFC should give good results in practice for those spatial operations that refer to proximity in space, whenever not much is known beforehand about the query population to be expected.

Acknowledgements

Special thanks go to Edouard Bugnion [5] for implementing a workbench for RSFC and experimentally verifying the claimed properties of the new RSFC. We are also grateful to Peter Remmele and Roger Wattenhofer [21] for carefully reading the manuscript and for helpful discussions and comments.

The authors acknowledge the partial support of the Swiss National Science Foundation SNF and of the ETH Zürich. Part of this work was done while Desh Ranjan and Emo Welzl were visiting ETH Zürich.

References

[1] D.J. Abel and D.M. Mark, A comparative analysis of some two-dimensional orderings, *Internat. J. Geographical Inform. Systems* **4** (1990) 21–31.

[2] D.J. Abel and J.L. Smith, A data structure and algorithm based on a linear key for a rectangle retrieval problem, *Computer Vision Graphics Image Processing* **24** (1983) 1–13.

[3] T. Asano, A. Hasegawa, D. Ranjan and T. Roos, Optimal and approximate digital halftoning algorithms and their experimental evaluation, Extended Abstract, in: *Proc. Asian Conf. on Computer Vision ACCV'93*, Osaka, 1993.

[4] T. Asano, D. Ranjan and T. Roos, Digital halftoning algorithms based on optimization criteria and their experimental evaluation, *Trans. IEICE on Fundamentals Electronics, Communications, and Computer Sciences*, to appear.

[5] E. Bugnion, T. Roos and P. Widmayer, Indexing points with space filling curves, private communication, 1994.

[6] C. Faloutsos, Multiattribute hashing using Gray codes, in: *Proc. ACM SIGMOD Internat. Conf. on the Management of Data*, Washington, DC (1985) 227–238.

[7] C. Faloutsos, Gray codes for partial match and range queries, *IEEE Trans. Software Eng.* **14** (1988) 1381–1393.

[8] C. Faloutsos and S. Roseman, Fractals for secondary key retrieval, in: *Proc. 8th ACM SIGACT/SIGMOD Symp. on Principles of Database Systems* (1989) 247–252.

[9] R.A. Finkel and J.L. Bentley, Quad trees: a data structure for retrieval on composite keys, *Acta Inform.* **4** (1974) 1–9.

[10] A. Hutflesz, H.-W. Six and P. Widmayer, Globally order preserving multidimensional linear hashing, in: *Proc. 4th Internat. Conf. on Data Engineering*, Los Angeles (1988) 572–579.

[11] A. Hutflesz, P. Widmayer and C. Zimmermann, Global order makes spatial access faster, *Internat. Workshop on Database Management Systems for Geographical Applications*, ESPRIT Basic Research Series Proc. (Springer, Berlin, 1992) 161–176.

[12] H.V. Jagadish, Linear clustering of objects with multiple attributes, in: *Proc. ACM SIGMOD Internat. Conf. on the Management of Data*, Atlantic City, NJ (1990) 332–342.

[13] F. Manola and J.A. Orenstein, Toward a general spatial data model for an object-oriented DBMS, in: *Proc. 12th Internat. Conf. on Very Large Data Bases*, Kyoto (1986) 328–335.

[14] G.M. Morton, A computer oriented geodetic data base and a new technique in file sequencing, IBM, Ottawa, Canada, 1966.

[15] J.A. Orenstein, Redundancy in spatial databases, in: *Proc. ACM SIGMOD Internat. Conf. on the Management of Data*, Portland (1989) 294–305.

[16] J.A. Orenstein, A comparison of spatial query processing techniques for native and parameter spaces, in: *Proc. ACM SIGMOD Internat. Conf. on the Management of Data*, Atlantic City, NJ (1990) 343–352.

[17] J.A. Orenstein and T.H. Merrett, A class of data structures for associative searching, in: *Proc. 3rd ACM SIGACT/SIGMOD Symp. on Principles of Database Systems*, Waterloo (1984) 181–190.

[18] H. Sagan, *Space Filling Curves* (Springer, Berlin, 1994).

[19] H. Tropf and H. Herzog, Multidimensional range search in dynamically balanced trees, *Angewandte Informatik* **2** (1981) 71–77.

[20] P. van Oosterom, Reactive data structures for geographic information systems, Dissertation (Proefschrift), Rijksuniversiteit Leiden, 1990.

[21] R. Wattenhofer, Raumfüllende Kurven für den Entwurf von Räumlichen Zugriffsstrukturen, Diploma Thesis, ETH Zürich, Switzerland, 1995.

[22] D. Wood, *Theory of Computation* (Harper & Row, New York, 1987).