6th Int. Symposium on Large Spatial Databases (SSD), 1999, Hong Kong, China

XZ-Ordering: A Space-Filling Curve for Objects with Spatial Extension

Christian Böhm[1], Gerald Klump[1] and Hans-Peter Kriegel[1]

[1] University of Munich, Computer Science Institute, Oettingenstr. 67,
D-80538 Munich, Germany
{boehm,klump,kriegel}@informatik.uni-muenchen.de

Abstract. There is an increasing need to integrate spatial index structures into commercial database management systems. In geographic information systems (GIS), huge amounts of information involving both, spatial and thematic attributes, have to be managed. Whereas relational databases are adequate for handling thematic attributes, they fail to manage spatial information efficiently. In this paper, we point out that neither a hybrid solution using relational databases and a separate spatial index nor the approach of existing object-relational database systems provide a satisfying solution to this problem. Therefore, it is necessary to map the spatial information into the relational model. Promising approaches to this mapping are based on space-filling curves such as Z-ordering or the Hilbert curve. These approaches perform an embedding of the multidimensional space into the one-dimensional space. Unfortunately, the techniques are very sensitive to the suitable choice of an underlying resolution parameter if objects with a spatial extension such as rectangles or polygons are stored. The performance usually deteriorates drastically if the resolution is chosen too high or too low. Therefore, we present a new kind of ordering which allows an arbitrary high resolution without performance degeneration. This robustness is achieved by avoiding object duplication, allowing overlapping Z-elements, by a novel coding scheme for the Z-elements and an optimized algorithm for query processing. The superiority of our technique is shown both, theoretically as well as practically with a comprehensive experimental evaluation.

1. Motivation

Index structures for spatial database systems have been extensively investigated during the last decade. A great variety of index structures and query processing techniques has been proposed [Güt 94, GG 98]. Most techniques are based on hierarchical tree-structures such as the R-tree [Gut 84] and its variants [BKSS 90, SRF 87, BKK 97]. In these approaches, each node corresponds to a page of the background storage and to a region of the data space.

There is an increasing interest in integrating spatial data into commercial database management systems. Geographic information systems (GIS) are data-intensive applications involving both, spatial and thematic attributes. Thematic attributes are usually best represented in the relational model, where powerful and adequate tools for evaluation and management are available. Relational databases, however, fail to manage spatial attributes efficiently. Therefore, it is common to store thematic attributes in a relational database system and spatial attributes outside the database in file-based multidimensional index structures (*hybrid solution*).

The hybrid solution bears various disadvantages. Especially the integrity of data stored in two ways, inside and outside the database system, is difficult to maintain. If an update operation involving both, spatial and thematic attributes fails in the relational database (e.g. due to concurrency conflicts), the corresponding update in the spatial index must be undone to guarantee consistency. Vice versa, if the spatial update fails, the corresponding update to the relational database must be aborted. For this purpose, a distributed commit protocol for heterogeneous database systems must be implemented, a time-consuming task which requires a deep knowledge of the participating systems. The hybrid solution involves further problems. File systems and database systems have usually different approaches for data security, backup and concurrent access. File-based storage does not guarantee physical and logical data independence. Thus, changes in running applications are complicated.

A promising approach to overcome these disadvantages is based on object-relational database systems. Object-relational database systems are relational database systems which can be extended by application-specific data types (called *data cartridges* or *data blades*). The general idea is to define data cartridges for spatial attributes and to manage spatial attributes in the database. For data-intensive GIS applications it is necessary to implement the multidimensional index structures in the database. This requires the access to the block-manager of the database system, which is not granted by most commercial database systems. For instance the current universal servers by ORACLE and INFORMIX do not provide any documentation of a block-oriented interface to the database. Data blades/cartridges are only allowed to access relations via the SQL interface. Thus, current object-relational database systems are not very helpful for our integration problem.

We can summarize that anyway, using current object-relational database systems or pure relational database systems, the only possible way to store spatial attributes inside the database is to map them into the relational model. An early solution for the management of multidimensional data in relations is based on space-filling curves. Space-filling curves map points of a multidimensional space to one-dimensional values. The mapping is distance preserving in the sense that points which are close to each other in the multidimensional space, are *likely* to be close to each other in the one-dimensional space. Although distance-preservation is not strict in this concept, the search for matching objects is usually restricted to a limited area in the embedding space.

The concept of space-filling curves has been extended to handle polygons. This idea is based on the decomposition of the polygons according to the space-filling curve. We will discuss this approach in section 2 and reveal its major disadvantage that it is very sensitive to a suitable choice of the resolution parameter. We will present a new method for applying space-filling curves to spatially extended objects which is not based on decomposition and avoids the associated problems.

For concreteness, we concentrate us in this paper on the implementation of the first filter step for queries with a given query region such as window queries or range queries. Further filter steps and the refinement step are beyond the scope of this paper. The rest of this paper is organized as follows: In section 2, we introduce space-filling curves and review the related work. Section 3 explains the general idea and gives an overview of our solution. The following sections show how operations such as insert, delete and search are handled. In section 5, a comprehensive experimental evaluation of our technique using the relational database management system ORACLE 8 is performed, showing the superiority of our approach over standard query processing techniques and competitive approaches.

2. Z-Ordering

Z-Ordering is based on a recursive decomposition of the data space as it is provided by a space-filling curve [Sag 94, Sam 89] called *Z-ordering* [OM 84], *Peano/Morton Curve* [Mor 66], *Quad Codes* [FB 74] or *Locational Codes* [AS 83].

2.1 Z-Ordering in Point Databases

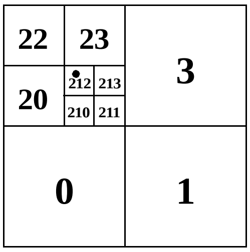


Figure 1: Z-Ordering.

Assume a point taken from the two-dimensional unit square $[0..1]^2$. The algorithm partitions the unit square into 4 quadrants of equal size (we change the description of Z-ordering here slightly to make it more comparable to our approach), which are canonically numbered from 0 to 3 (cf. figure 1). We note the number of the quadrant and partition this quadrant into its four sub-quadrants. This is recursively repeated until a certain basic resolution is reached. The fixed number of recursive iterations is called the resolution level g. Then we stop and use the obtained sequence of g digits (called *quadrant sequence*) as ordering key for the points (we order lexicographically). Each quadrant sequence represents a region of the data space called *element*. For instance, the sequence <00> stands for an element with side length 0.25 touching the lower left corner of the data space. Elements at the basic resolution which are represented by quadrant sequences of length g are called *cells*. If an element e_1 is contained in another element e_2, then the corresponding quadrant sequence $Q(e_2)$ is a prefix of $Q(e_1)$. The longer a quadrant sequence is, the smaller is the corresponding element. In the unit square, the area of an element represented by a sequence of length l is $(1/4)^l$. In a point database, only cells at the basic resolution are used. Therefore, all quadrant sequences have the same lengths and we can interpret the quadrant sequences as numbers represented in the quaternary system (i.e. base 4). Interpreting sequences as numbers facilitates their management in the index and does not change ordering of the points, because the lexicographical order corresponds to the less-equal relation of numbers. The points are managed in an order-preserving one-dimensional index structure such as a B^+-tree.

2.2 Query Processing in Z-Ordering

Assume a window query with a specified window. The data space is decomposed into its four quadrants. Each quadrant is tested for intersection with the query window. If the quadrant does not intersect the query window, nothing has to be done. If the quadrant is completely enclosed in the query window, we have to retrieve all points from the database having the quadrant sequence of this element as a prefix of their keys. If the keys are represented as integer numbers (cf. section 3.2), we have to retrieve an interval of subsequent numbers. All remaining quadrants which are intersected by the window but not completely enclosed in the window (i.e. “real” intersections) are decomposed recursively until the basic resolution is reached.

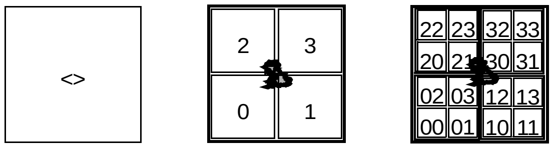


Figure 2: The one-value-representation.

2.3 A Naive Approach for Polygon Databases

To extend the concept of Z-ordering for the management of objects with a spatial extension (e.g. rectangles or polygons), we face the problem that a given polygon intersects with many cells. A naive approach could be to store every cell covered by the object in the database. Obviously, this method causes a huge storage overhead unless the basic grid is very coarse. Therefore, several methods have been proposed to reduce the overhead when using a finer grid.

2.4 One-Value-Approximation

The objects are approximated by the smallest element which encloses the complete object (cf. figure 2). In this case our recursive algorithm for the determination of the quadrant sequence is modified as follows: Partition the current data space into four quadrants. If exactly one quadrant is intersected by the object, proceed recursively with this quadrant. If more than one quadrant is intersected, then stop. Use the quadrant sequence obtained up to that point as the ordering key. This method has the obvious advantage that each object is represented by a single key, not by a set of keys as in our naive approach. But this method yields also several disadvantages. The first disadvantage is that the quadrant sequences in this approach have different lengths, depending on the resolution of the smallest enclosing quadrant. Thus, our simple interpretation as a numerical value is not possible. Keys must be stored as strings with variable length and compared lexicographically, which is less efficient than numerical comparisons. The second problem is that objects may be represented very poorly. For instance any polygon intersecting one of the axis-parallel lines in the middle of the data space (the line $x = 0.5$ and the line $y = 0.5$) can only be approximated by the empty quadrant sequence. If the polygon to be approximated is very large, an approximation by the empty sequence or by very short sequences seems to be justified. For small polygons, the relative approximation error is too large. The relative space overhead of the object approximation is thus unlimited. In fact, objects approximated by the empty quadrant sequence are candidates to every query a user asks. The more objects with short quadrant sequences are stored in the database, the worse is the selectivity of the index.

2.5 Optimized Redundancy

To avoid the unlimited approximation overhead, Orenstein proposes a combination of the naive approach and the one-sequence representation [Ore 89a, Ore 89b]. He adopts the idea of the object decomposition in the naive approach, but does not necessarily decompose the object until the basic resolution is reached. Instead, he proposes two different criteria, called *size-bound* and *error-bound* to control the number of quadrants into which an object is decomposed. Each subobject is stored in the index by using its quadrant sequence, e.g. represented as a string. Although this concept involves object duplication (which is called redundancy by Orenstein), the number of records stored in the index is not directly determined by the grid resolution as in the naive approach. Unlike in the one-sequence approach, it is not necessary to represent small objects by the empty sequence or by very short sequences. According to Orenstein, typically a decomposition into 2-4 parts is sufficient for a satisfactory search performance.

Orenstein's approach alleviates the problems of the two previous approaches, but a duplicate elimination is still required and the keys are sequences with varying length. Orenstein determines an optimal degree of redundancy only experimentally. An analytical solution was proposed by Gaede [Gae 95] who identified the complexity of the stored polygons, described by their perimeter and their fractal dimension, as the main parameters for optimization. A further problem when redundancy is allowed arises in connection with secondary filters in a multi-step environment. Information which can be exploited for fast filtering of false hits, such as additional conservative approximations (e.g. *minimum bounding rectangles MBR*), should not be subject to duplication due to its high storage requirement. To avoid duplication of such information, it must be stored in a separate table which implies additional joins in query processing.

A further consequence of Gaede's analysis [Gae 95] is that the number of intervals which are generated from the query window is proportional to the number of grid cells intersected by the boundary of the query window (i.e. its perimeter). This means that a too fine resolution of the grid leads to a large number of intervals and thus to deteriorated performance behavior when a relational database system is used. The reason is that the intervals must be transferred to and processed by the database server, which is not negligible, if the number of intervals is very high (e.g. in the thousands).

2.6 Alternative Techniques

Several improvements of the Z-ordering concept are well-known (cf. figure 3). Some authors propose the use of different curves such as Gray Codes [Fal 86, Fal 88], the Hilbert Curve [FR 89, Jag 90] or other variations [Kum 94]. Many studies [Oos 90, Jag 90, FR 91] prefer the Hilbert curve among the proposals, due to its best distance preservation properties (also called *spatial clustering properties*). [Klu 98] proposes a great variety of space-filling curves and makes a comprehensive performance study using a relational implementation. As this performance evaluation [Klu 98] does not yield a substantial performance improvement of the Hilbert curve or other space-filling curves over Z-ordering, we use the Peano/Morton curve because it is easier to compute.

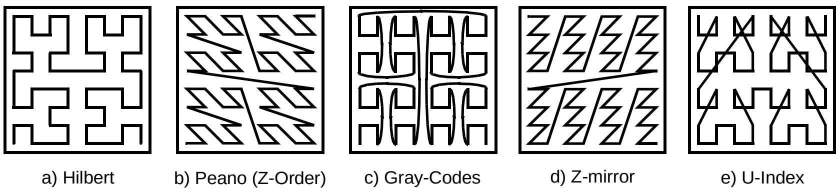


Figure 3: Various Space-Filling Curves.

3. A Space-Filling Curve for Spatially Extended Objects

In contrast to the previous approaches, we propose a solution which avoids the disadvantages of object duplication and variable-length quadrant sequences. Our method is thus completely insensitive against a too fine grid resolution. There is no need to optimize the resolution parameter. It can always be taken as fine as possible, i.e. the full bit precision of the CPU can be exploited. Three ideas are applied to achieve this robustness: The first idea presented in section 3.1 is to incorporate overlap into the concept of elements. We will define the elements such that adjacent elements at the same resolution level l overlap each other up to 50%. This method enables us to store objects without redundancy (i.e. object duplication) and without uncontrolled approximation error. In particular, it is impossible that a very small object must be represented by a very short sequence or even by the empty sequence.

The second idea is to use a sophisticated coding scheme for the quadrant sequences which maps quadrant sequences with varying length into the integer domain in a distance-preserving way. The coding algorithm is presented in section 3.2. The third idea (cf. section 4.3) is an efficient algorithm for interval generation in query processing. The goal of the algorithm is to close small gaps between adjacent intervals if the overhead of processing an additional interval is larger than the cost for overreading the interval gap.

We call our technique which maps polygons into integer-values *extended Z-ordering* or *XZ-ordering*. The integer values forming the keys for search are called *XZ-values*. For each polygon in the database, we store one record which contains its XZ-value and a pointer to the exact geometry representation of the polygon. As we avoid object duplication, further information such as thematic attributes or information for secondary filters (the *MBR* or other conservative and progressive approximations, cf. [BKS 93]) can be stored in the same table.

3.1 Overlapping Cells and Elements

The most important problem of the *one-value representation* is that several objects are approximated very poorly. Every object intersecting with one of the axis-parallel lines $x = 0.5$ and $y = 0.5$ is represented by the empty quadrant sequence which characterizes the element comprising of the complete data space. If the object extension is very small (close to 0), the relative approximation error diverges to infinity.

In fact, every technique which decomposes the space into disjoint cells gets into trouble if an object is located on the boundary between large elements. Therefore, we modify our definition of elements such that overlap among elements on the same resolution level l is allowed.

The easiest way to envisage a definition of overlapping elements is to take the original elements as obtained by Z-ordering and to enlarge the height and width by a factor 2 upwards and to the right, as depicted in figure 4. Then, two adjacent cells overlap each other by 50%. The special advantage is, that this definition contains also small elements for objects intersecting with the middle axis.

Definition 1: Enlarged elements

The lower left corner of an enlarged element corresponds to the lower left corner of Z-ordering. Let s be the quadrant sequence of this lower left corner and let $|s|$ denote its length. The upper right corner is translated such that the height and width of the element is $0.5^{|s|-1}$.

It is even possible to guarantee bounds for a minimal length of the quadrant sequence (and thus of the approximation quality) based on the extension of the object in *x*- and *y*-direction.

Lemma 1. Minimum and Maximum Length of the Quadrant Sequence

The length $|s|$ of the quadrant sequence s for an object with height h and width w is bounded by the following limits:

$$l_1 \leq |s| < l_2 \text{ with } l_1 = \lfloor \log_{0.5}(\max\{w, h\}) \rfloor \text{ and } l_2 = l_1 + 2$$

Proof (Lemma 1)

Without loss of generality, we assume $w \geq h$. We consider the two disjoint space decompositions into elements at the resolution levels l_1 and l_2 and call the arising decomposition grids the l_1-grid and the l_2-grid, respectively. The distances w_1 and w_2 between the grid-lines are equal to the widths of the elements at the corresponding decomposition levels l_1 and l_2.

(1) As the distance w_1 between two lines of the l_1-grid is greater than or equal to *w,*

$$\text{because } w_1 = 0.5^{l_1} = 0.5^{\lfloor \log_{0.5}(w) \rfloor} \geq 0.5^{\log_{0.5}(w)} = w,$$

the object can at most be intersected by one grid line parallel to the *y*-axis and by one grid line parallel to the *x*-axis. If the lower left element among the intersecting elements is enlarged as in definition 1, the object must be completely contained in the enlargement.

(2) As the distance w_2 between two lines of the l_2-grid is smaller than *w*/2,

$$\text{because } w_2 = 0.5^{l_2} = 0.5^{\lfloor \log_{0.5}(w) \rfloor + 2} < 0.5^{\log_{0.5}(w) + 1} = w/2,$$

the object is intersected at least by two *y*-axis parallel lines of the l_2-grid. Therefore, there is no element at the l_2-level which can be enlarged according to definition 1 such that the object is contained.

❑

Lemma 1 can be exploited to provide boundaries for the relative approximation error of objects. As polygons can be arbitrary complex, it is not possible for any approximation technique with restricted complexity (such as MBRs or our technique) to provide error boundaries. However, we can guarantee a maximum relative error for square objects which are not smaller than the basic resolution:

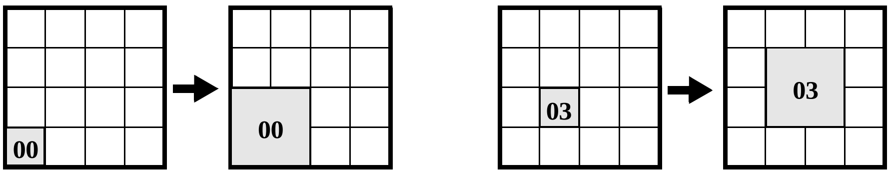


Figure 4: Enlarged Regions in XZ-Ordering.

Lemma 2. Maximum approximation error

The relative approximation error for square objects is limited.

Proof (Lemma 2)

According to lemma 1, the quadrant sequence for a square object of width w has at least the length $l_1 = \lfloor \log_{0.5}(w) \rfloor$. The width of the corresponding cell is

$$w_1 = 0.5^{l_1} = 0.5^{\lfloor \log_{0.5}(w) \rfloor} < 0.5^{\log_{0.5}(w) - 1} = 2 \cdot w .$$

The enlarged element has the area $A = 4 \cdot w_1^2 < 16 \cdot w^2$.

The maximum approximation error is limited by the following value:

$$E_{\text{rel}} = \frac{A - w^2}{w^2} < 15 \quad .$$

❑

Although a relative approximation error of 15 seems rather large, it is an important advantage of our technique over the one-value representation that the approximation error is limited at all. Our experimental evaluation will show that also the average approximation error of our technique is smaller than that of the one-value representation.

3.2 Numbering Quadrant Sequences

We are given a quadrant sequence with a length varying from 0 to the maximum length g determined by the basic resolution. Our problem is to assign numbers to the sequences in an order-preserving way, i.e. the less-equal-order of the numbers must correspond to the lexicographical order of the quadrant sequences. Let the length of the quadrant sequence s be l. The following lemma is used to determine the number of cells and elements contained in the corresponding region $R(s)$:

Lemma 3. Number of cells and elements inside of region $R(s)$

The number of cells of resolution g contained in a region described by the quadrant sequence s with $|s| = l$ is

$$N_{\text{cell}}(l) = 4^{g-l} .$$

The corresponding number of elements (including element $R(s)$ and the cells) is

$$N_{\text{elem}}(l) = \frac{4^{g-l+1} - 1}{3} .$$

Proof (Lemma 3)

(1) There is a total of 4^l elements with length l and a total of 4^g cells in the data space. As both cells and elements of length l cover the data space in a complete and overlap-free way, the area of an element is $(1/4)^l/(1/4)^g = 4^{g-l}$ times larger than the area of a cell.

(2) The number of elements of length i contained in an element of length l corresponds to 4^{i-l}. For obtaining the number of all elements, we have to summarize over all i ranging from l to g:

$$\sum_{l \le i \le g} 4^{g-i} = \frac{4^{g-l+1}-1}{3}$$

❑

For the definition of our numbering scheme, we have to make sure that between the codes of two subsequent strings s_1 and s_2 of length l there are enough numbers for all strings which are ordered between s_1 and s_2. These are exactly the strings having s_1 as prefix, and their number is thus $(4^{g-l}-1)/3$. We therefore multiply each quadrant number q_i in the sequence $s = \langle q_0\, q_1\, ...\, q_i\, ...\, q_{l-1} \rangle$ with $(4^{g-i}-1)/3$.

Definition 2: Sequence Code

The sequence code $C(s)$ of a quadrant sequence $s = \langle q_0\, q_1\, ...\, q_i\, ...\, q_{l-1} \rangle$ corresponds to

$$C(s) = \sum_{0 \le i < l} q_i \cdot \frac{4^{g-i}-1}{3} + 1$$

Lemma 4. Ordering Preservation of the Sequence Code

The less-equal order of sequence codes corresponds to the lexicographical order of the quadrant sequences:

$$s_1 <_{\text{lex}} s_2 \Leftrightarrow C(s_1) < C(s_2)$$

Proof (Lemma 4)

'$\Rightarrow$': Suppose $s_1 <_{\text{lex}} s_2$ with $s_1 = \langle q_0 ... q_{l-1} \rangle$ and $s_2 = \langle p_0 ... p_{m-1} \rangle$. Then, one of the following predicates must be true according to the definition of the lexicographical order:

(1) there exists an i ($0 \le i < \min\{m, l\} - 1$) such that $q_i < p_i$ and $q_j = p_j$ for all $j < i$

(2) $m > l$ and $q_j = p_j$ for all $j < l$.

In case (1), we know that the i-th term of the sum of $C(s_1)$ is at least by $(4^{g-i}-1)/3$ less than the i-th term in the sum of $C(s_2)$. The summands for all $j < i$ are equal. We have to show that the difference D of the sums of all remaining terms is smaller than $N_{\text{elem}}(g - i)$ to guarantee $C(s_1) < C(s_2)$. The difference D is maximal if $q_j = 3$ and $p_j = 0$ for all $j > i$. In this case, we can determine D as follows:

$$D = \sum_{i < j < g} 3 \cdot \frac{4^{g-j}-1}{3} = \sum_{0 < j < g-i} 4^j - 1 = \frac{4^{g-i}-1}{3} - (g-i+1) < \frac{4^{g-i}-1}{3}$$

In case (2), $C(s_2) > C(s_1)$, because the first i summands are equal, and $C(s_2)$ has additional positive summands which are not available in the sum of $C(s_1)$.

'$\Leftarrow$': We can rewrite the condition $s_1 <_{\text{lex}} s_2 \Leftarrow C(s_1) < C(s_2)$ to $s_1 \ge_{\text{lex}} s_2 \Rightarrow C(s_1) \ge C(s_2)$. The proof is then analogue to the '$\Rightarrow$'-direction.

❑

Lemma 5. Minimality of the Sequence Code

There exists no mapping from the set of quadrant sequences to the set of natural numbers which requires a smaller interval of the natural numbers than $C(s)$.

Proof (Lemma 5)

From lemma 3 it follows that there are $N_{\text{elem}}(0) = (4^{g+1} - 1)/3$ different elements. $C(s)$ is maximal if s has the form $<3^g>$. In this case, $C(s)$ evaluates to the term

$$C(s) = \sum_{0 \le i < g} 3 \cdot N_{\text{elem}}(g - i) + 1 = \sum_{0 \le i < g} 3 \cdot \frac{4^{i+1} - 1}{3} + 1 = \left(\sum_{0 \le i \le g} 4^i \right) - 1$$

$$= \frac{4^{g+1} - 1}{3} - 1 = N_{\text{elem}}(0) - 1$$

The coding of the empty string $C(<>) = 0$. Therefore, the $N_{\text{elem}}(0)$ different strings are mapped exactly to the interval $[0, N_{\text{elem}}(0) - 1]$.

❏

From lemma 5, it also follows that C is a surjective mapping onto the set $\{0, ..., N_{\text{elem}}(0) - 1\}$. We note without a formal proof that C is also injective. The quadrant sequence can be reconstructed from the coding in an efficient way, but this is not necessary for our purposes.

4. Query Processing

4.1 Insert and Delete

We know from lemma 1 that the quadrant sequence has the length $l_1 = \lfloor \log_{0.5}(\max\{w, h\}) \rfloor$ or $l_1 + 1$. We can decide this by the following predicate which tests whether the object is intersected by one or two grid lines (x_l denotes the lower boundary of the object in x direction; the same criterion must be applied for y_l):

$$\left\lfloor \frac{x_l}{l_1} + 2 \right\rfloor \cdot l_1 \le x_l + w$$

Once the length l of the quadrant sequence is determined, the corresponding quadrant sequence s is determined for the lower left corner of the bounding box of the object, as described in section 2.1. This sequence s is clipped to the length l and coded according to definition *2*. The obtained value is used as a key for storage and management of the object in a relational index. Our actual algorithm performs the operations of recursive descent into the quadrants and coding according to definition *2* simultaneously without explicitly generating the quadrant sequence. The algorithm runs in $O(l)$ time.

4.2 From Window Queries to Interval Sets

For query processing, we can proceed in a recursive way similar to the algorithm presented in section 2.2. We determine which quadrants are intersected by the query. Those which are not intersected are ignored. If a quadrant is completely contained in the query window, all elements having the corresponding quadrant sequence as prefix are completely contained in the query. Therefore, the interval of the corresponding XZ-values is generated and marked for a later retrieval from the database. If a quadrant is intersected,

```
FOREACH interval i in list {
        WHILE succ (i).lower - i.upper < maxgap {
                i.upper := succ (i).upper ;
                delete succ (i) ;
}       }
```

Figure 5: The simple algorithm for gap closing.

the corresponding XZ-value is determined and marked (it is handled as a one-value interval). Then the algorithm is called recursively for all its sub-quadrants. Finally, we have marked a set of intervals of XZ-values which are to be retrieved from the database. This set is translated into an SQL statement which is transferred to the DBMS.

Problems arise if the resolution of the basic grid is chosen too fine, because in this case, typically many elements are partially intersected. In the average case, the number of generated intervals is in the order $O(2^g)$. It is very costly to transfer and compile such a complex query. To alleviate this problem, one could apply the simple densification algorithm depicted in figure 5 to the set of intervals. As only small gaps between subsequent intervals are closed, it is unlikely that this densification of intervals causes additional disk accesses in query processing. The densification after the interval generation optimizes query compilation cost and related cost factors, but does not change the general complexity of the interval generation. For this purpose, an algorithm must be devised which generates the intervals directly in a way that closes the gaps. This algorithm is described in the subsequent section.

4.3 An Efficient Algorithm for the Interval Generation

This algorithm allows the user to specify the number of intervals n_{int} which are generated. We exploit a general property of XZ-ordering and Z-ordering, which we note here without a formal proof: Let us consider interval sets which come up if we restrict our interval generation to a certain length l ($0 \le l \le q$) of the corresponding quadrant sequences. If l is increased, it is possible that more intervals are generated. The factor by which the number of intervals in the interval set grows when increasing the length l by one, is restricted by 4. All intervals of the larger interval set are contained in an interval of the smaller set. The additional gaps between the intervals in an interval set generated when l is increased by 1, can only be smaller than all gaps which are visible when l is decreased by 1. If we would descend the recursion tree of the algorithm in section 2.2 in a breadth-first fashion and stop if we have found 4 times as many intervals as demanded ($4n_{\text{int}}$), we know that we can densify this set to a set of n_{int} intervals with the largest possible gaps between them.

Instead of the breadth-first traversal, we have implemented the following algorithm: In a first phase, depicted in figure 6, the algorithm performs a depth-first traversal for determining the number of intervals in each recursion level. The clue is, that the recursive descent is avoided, if the corresponding level has reached a number of $4n_{\text{int}}$ intervals, because we are only interested in the first level having more than $4n_{\text{int}}$ intervals. In each level, the algorithm measures the number of transitions in the XZ-order, where the extended elements of the space-filling curve cross the query region. As the algorithm is

```
VAR num_ch: ARRAY [0..g] OF INTEGER ;

PROCEDURE det_num_changes (element, query: REGION; cur_num_ch: INTEGER;
                                   c_depth: INTEGER; VAR inside: BOOLEAN) {
    IF (NOT intersect (element, query) ) {
        IF (inside) {
            inside := FALSE; INCREMENT (num_ch [c_depth]);
    }   }
    ELSE IF (contains (query, element) ) {
        IF (NOT inside) {
            inside := TRUE; INCREMENT (num_ch [c_depth] ) ;
    }   }
    ELSE {
        IF (NOT inside) {
            inside := TRUE; INCREMENT (num_ch [c_depth] ) ;
        }
        IF (current_depth < g AND c_num_ch + num_ch[c_depth] < n_int) {
            FOREACH subquadrant {
                det_num_changes (   subquadrant, query, c_num_ch + num_ch [c_depth],
                                    c_depth + 1, inside) ;
}   }   }   }

FUNCTION suitable_level (query:REGION): INTEGER {
    INITIALIZE num_ch := {0, 0, ..., 0} ;
    det_num_changes (dataspace, query, 0, 0, FALSE) ;
    suitable_level := first i where num_ch[i] >= n_int ;
}
```

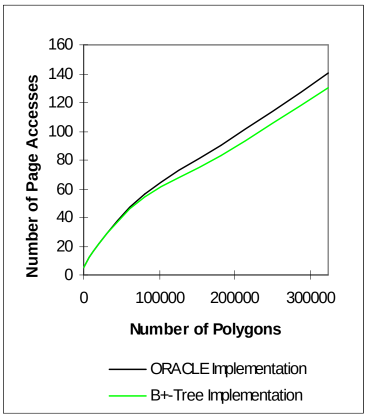
Figure 6: The Algorithm for Determining the Suitable Recursion Level ("Phase 1").

in each level called $4n_{int}$ times maximum, the complexity of the algorithm is bounded by $O(n_{int} \cdot g)$.

In the second phase, we generate the corresponding interval set by a depth-first traversal which is limited to the recursion depth obtained in phase one. Whenever the number of intervals becomes greater than n_{int}, the two neighboring intervals with the smallest gap between them are merged. In a third phase, the upper bounds of the intervals are investigated. It is possible that the upper bounds can be slightly improved (i.e. decreased) by a deeper descent in the recursion tree. All three phases yield a linear complexity in g.

5. Experimental Evaluation

In order to verify our claims that an implementation of spatial index structures does not only provide advantages from a software engineering point of view but also in terms of performance, we actually implemented the XZ-Ordering technique on top of ORACLE-8 and performed a comprehensive experimental evaluation using data from a GIS application. Our database contains 324,000 polygons from a map of the European Union. We also generated smaller data sets from the EU map to investigate the behavior with vary-



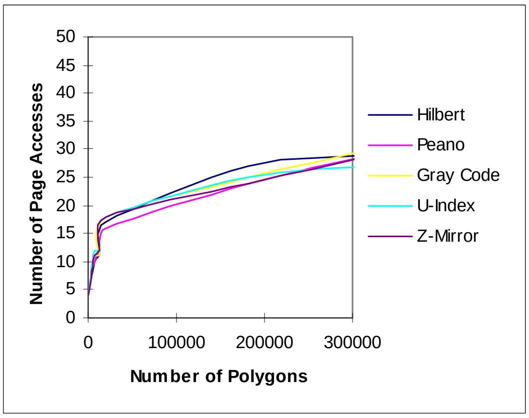
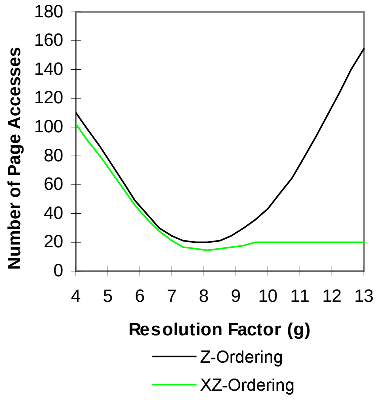


Figure 7: (l.) Comparison between ORACLE and file-based B+-tree
(r.) Comparison between various Space-Filling Curves.

ing database size. Our application was implemented in Embedded SQL/C (dynamic SQL level 3) on HP C-160 workstations. The database server and the client application run on separate machines, connected by a 100MBit Fast Ethernet.

We also had an implementation on top of a file-system based B+-tree which was used for comparison purposes. The reason for this test was to show that our technique is implementable on top of a commercial database system and that the additional overhead induced by the database system is small. In both cases, we varied the database size from 1,000 to 324,000 and executed window queries with a selectivity of 1% (with respect to the number of polygons in the query result). All experiments presented in this section were repeated ten times. The presented results are averages over all trials. The number of page accesses (cf. figure 7, left diagram) is in Oracle only by up to 8% higher than in the file-based B+-tree implementation. We used a comparable node capacity in both implementations. A second experiment depicted on the right side of figure 7 shows that the application of a specific space filling curve has no strong influence on the performance of our technique. Using the same settings, we tested various curves including Z-Ordering (Peano), the Hilbert-curve and Gray-Codes. There was no consequent trend observable which could suggest the superiority of one of the curves. Therefore, we decided to perform all subsequent experiments using the Peano-/Morton-curve, because the implementation is facilitated.

The purpose of the next series of experiments is to show the superiority of our approach over competitive techniques such as Orenstein's Z-Ordering. First, we demonstrate in figure 8 that our technique, in contrast to Z-Ordering, is not subject to a performance deterioration when the grid resolution is chosen too high. We constructed several indexes with varying resolution parameter g for both techniques, XZ-Ordering and Z-Ordering, where we applied the size-bound decomposition strategy resulting in a redundancy of 4. In this experiment, we stored 81,000 polygons in the database and retrieved 1% of them in window queries. Z-Ordering has a clear minimum at a resolution of $g = 8$ with a satisfying query performance. When the resolution is slightly increased or decreased, the query performance deteriorates. For instance, if g is chosen to 10, the num-



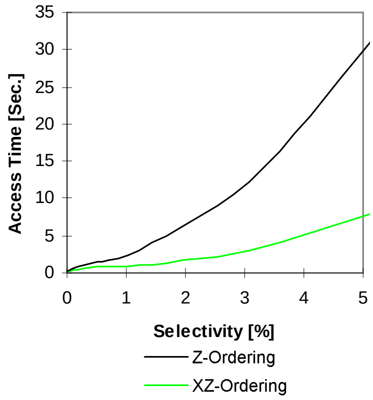
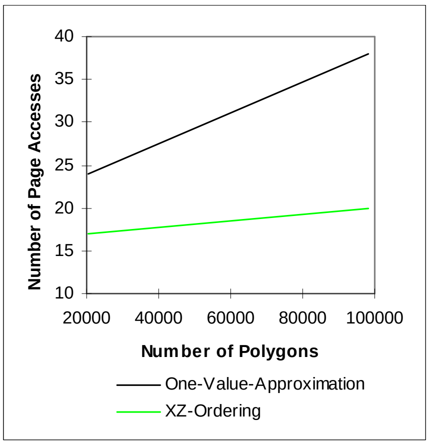


Figure 8: (l.) Z-Ordering is sensitive to the resolution
(r.) Performance comparison with varying selectivity.

ber of page accesses is by 80% higher than at the optimum. At the maximum resolution factor $g = 13$, the number of page accesses was by a factor more than 6.4 higher than the optimum. In contrast, our technique shows a different behavior. If the grid resolution parameter g is too small, the performance is similar to the performance of Z-ordering. A too coarse grid leads obviously to a bad index selectivity, because many objects are mapped to the same Z-values or XZ-values, respectively. Both techniques have the same point of optimum. Beyond this point, XZ-Ordering yields a constant number of page accesses. Therefore, it is possible to avoid the optimization of this parameter which is difficult and depends on dynamically changing information such as the fractal dimension of the stored objects and their number. This trend is maintained and even intensified if the selectivity of the query is increased, as depicted on the right side of figure 8. Here, the number of polygons was 324,000 and the resolution was fixed to $g = 10$.



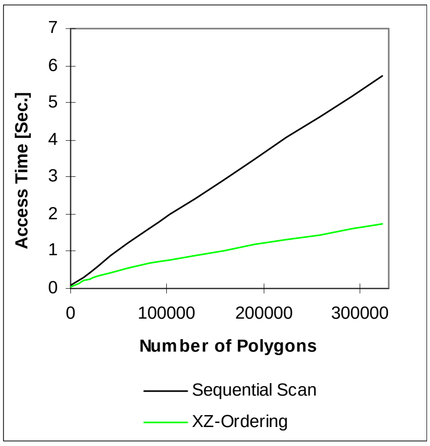


Figure 9: (l.) Comparison with the One-Value-Approximation
(r.) Comparison with the Sequential Scan.

In a further series of experiments, we compared our technique with the One-Value-Approximation (cf. section 2.4) and with the sequential scan of the data set. The remaining test parameters correspond with the previous experiments. Both competitive techniques are clearly outperformed as depicted in figure 9. The One-Value-Approximation yields up to 90% more page accesses. The sequential scan needs up to 326% as much processing time as XZ-Ordering.

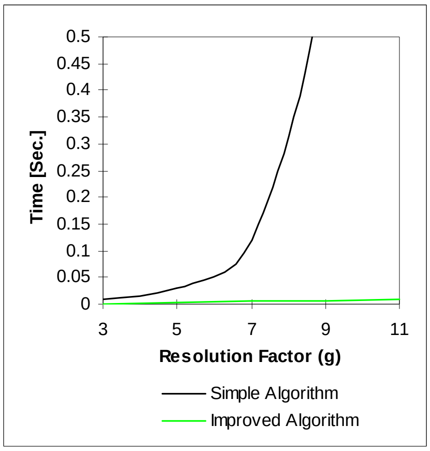


Figure 10: Improved interval generation.

In a last experiment, we determined the influence of the query processing algorithm presented in section 4.3. We varied again the resolution parameter g from 3 to 11 and generated interval sets of 20 intervals according to a window query with side length 0.03 (the unit square is the data space). We measured the CPU time which is required for the generation of the intervals and of the corresponding string for the dynamic SQL statement. We compared our algorithm with the simple algorithm (cf. section 2.2) extended by the gap closing algorithm (cf. figure 5). The results are presented in figure 10. The simple algorithm has an exponential complexity with respect to the resolution, whereas the improved algorithm is linear. For the finest resolution (g = 11), the improved algorithm is by the factor 286 faster than the simple algorithm.

6. Conclusion

In this paper, we have proposed XZ-Ordering, a new technique for a one-dimensional embedding of extended spatial objects such as rectangles or polygons. In contrast to previous approaches which require the optimization of the resolution parameter in order to become efficient, our technique is insensitive against a fine resolution of the underlying grid. Therefore, the resolution parameter is only restricted by hardware constants (the number of bits for an integer value) and can be chosen as fine as possible. This robustness is achieved by applying three basic concepts. Object decomposition is avoided by the concept of overlapping elements. A sophisticated order-preserving coding into the integer domain facilitates the management of the search keys for the DBMS. A new query processing algorithm is also insensitive to the grid resolution.

The superiority of our approach was shown both, theoretically as well as practically. We implemented the XZ-Ordering technique on top of the relational database system ORACLE-8. Our technique outperformed competitive techniques based on space-filling curves as well as standard query processing techniques.

References

[AS 83] Abel D.J., Smith J.L.: *'A Data Structure and Algorithm Based on a Linear Key for a Rectangle Retrieval Problem'*, Computer Vision 24, pp. 1-13.

[BKK 97] Berchtold S., Keim D., Kriegel H.-P.: '*Using Extended Feature Objects for Partial Similarity Retrieval*', VLDB Journal Vol. 6, No. 4, pp. 333-348, 1997.

[BKS 93] Brinkhoff T., Kriegel H.-P., Schneider R.: '*Comparison of Approximations of Complex Objects Used for Approximation-based Query Processing in Spatial Database Systems'*, ICDE 1993, pp. 40-49.

[BKSS 90] Beckmann N., Kriegel H.-P., Schneider R., Seeger B.: *'The R*-tree: An Efficient and Robust Access Method for Points and Rectangles',* Proc. ACM SIGMOD Int. Conf. on Management of Data, Atlantic City, NJ, 1990, pp. 322-331.

[Gae 95] Gaede V.: *'Optimal Redundancy in Spatial Database Systems'*, Proc. 4th Int. Symposium on Advances in Spatial Databases, SSD'95, Portland, Maine, USA, 1995, Lecture Notes in Computer Science Vol. 951, pp. 96-116.

[GG 98] Gaede V., Günther O.: '*Multidimensional Access Methods'*, ACM Computing Surveys, Vol. 30, No. 2, 1998, pp. 170-231.

[Güt 94] Güting R. H.: *'An Introduction to Spatial Database Systems'*. VLDB Journal , Vol. 3, No. 4, 1994.

[Gut 84] Guttman A.: *'R-trees: A Dynamic Index Structure for Spatial Searching',* Proc. ACM SIGMOD Int. Conf. on Management of Data, Boston, MA, 1984, pp. 47-57.

[Hil 91] Hilbert D.: *'Über die stetige Abbildung einer Linie auf ein Flächenstück'*, Math. Annln., Vol. 38, 1891, pp. 459-460.

[Klu 98] Klump G.: *'Development, Implementation and Evaluation of Strategies for Geometric Query Processing Under Oracle 8'* (in German), master thesis, University of Munich.

[Kum 94] Kumar A.: *'A Study of Spatial Clustering techniques'*, DEXA 1994, pp. 57-71.

[Fal 86] Faloutsos C.: *'Multiattribute Hashing Using Gray Codes'*, Proc. ACM SIGMOD Int. Conf. on Management of Data, Washington D.C., 1986, pp. 227-238.

[Fal 88] Faloutsos C.: *'Gray Codes for Partial Match and Range Queries'*, IEEE Transactions on Software Engineering (TSE) , Vol. 14, No. 10, 1988, pp. 1381-1393.

[FB 74] Finkel R, Bentley J.L. *'Quad Trees: A Data Structure for Retrieval of Composite Keys'*, Acta Informatica , Vol. 4, No. 1, 1974, pp. 1-9.

[FR 91] Faloutsos C., Rong Y.: *'DOT: A Spatial Access Method Using Fractals'*, Proc. 7th Int. Conf. on Data Engineering, Kobe, Japan, 1991, pp. 152-159.

[FR 89] Faloutsos C., Roseman S.: *'Fractals for Secondary Key Retrieval'*, Proc. 8th ACM PODS, Philadelphia, PA, 1989, pp. 247-252.

[Jag 90] Jagadish H. V.: *'Linear Clustering of Objects with Multiple Atributes'*, Proc. ACM SIGMOD Int. Conf. on Managment of Data, Atlantic City, NJ, 1990, pp. 332-342.

[Mor 66] Morton G.: *'A Computer Oriented Geodetic Data BAse and a New Technique in File Sequencing'*, IBM Ltd., 1966.

[Oos 90] Oosterom P.: *'Reactive Data Structures for Geographic Iinformation Systems'*. Ph.D. thesis, University of Leiden, The Netherlands.

[Ore 89a] Orenstein J. A.: *'Redundancy in Spatial Databases'*, Proc. ACM SIGMOD Int. Conf. on Management of Data, Portland, OR, 1989, pp. 294-305.

[Ore 89b] Orenstein J. A.: *'Strategies for Optimizing the Use of Redundancy in Spatial Databases'*, Proc. 1st Symposium on Large Spatial Databases, Santa Barbara, CA, pp. 115-134.

[OM 84] Orenstein J. A., Merrett T. H.: *'A Class of Data Structures for Associative Searching'*, Proc. 3rd ACM PODS, Waterloo, Ontario, Canada, 1984, pp. 181-190.

[Sag 94] Sagan H.: *'Space-Filling Curves'*, Berlin/Heidelberg/New York: Springer-Verlag, 1994.

[Sam 89] Samet H.: *'The design and analysis of spatial data structures'*. Reading, MA: Addison-Wesley.

[SRF 87] Sellis T. K., Roussopoulos N., Faloutsos C.: *'The R+-Tree: A Dynamic Index for Multi-Dimensional Objects'*, Proc. 13th Int. Conf. on Very Large Data Bases, Brighton, England, 1987, pp. 507-518.