

Spatial Query Processing for High Resolutions

Hans-Peter Kriegel^{*}, Martin Pfeifle^{*}, Marco Pötke^{**} and Thomas Seidl^{***}
^{*}University of Munich, {kriegel, pfeifle}@dbs.informatik.uni-muenchen.de
^{**}sd&m AG software design & management, marco.poetke@sdm.de
^{***}RWTH Aachen University, seidl@informatik.rwth-aachen.de

Abstract

Modern database applications including computer-aided design (CAD), medical imaging, or molecular biology impose new requirements on spatial query processing. Particular problems arise from the need of high resolutions for very large spatial objects, including cars, space stations, planes and industrial plants, and from the design goal to use general purpose database management systems in order to guarantee industrial-strength. In the past two decades, various stand-alone spatial index structures have been proposed but their integration into fully-fledged database systems is problematic. Most of these approaches are based on decomposition of spatial objects leading to replicating index structures. In contrast to common black-and-white decompositions which suffer from the lack of intermediate solutions, we introduce grey approximations as a new and general concept. We demonstrate the benefits of grey approximations in the context of encoding spatial objects by space filling curves resulting in grey interval sequences. Spatial intersection queries are then processed by a filter and refine architecture which, as an important design goal, can purely be expressed by means of the SQL:1999 standard. Our new High Resolution Indexing (HRI) method can easily be integrated into general purpose DBMSs. The experimental evaluation on real-world test data from car and plane design projects points out that our new concept outperforms competitive techniques that are implementable on top of a standard object-relational DBMS by an order of magnitude with respect to secondary storage space and overall query response time.

1. Introduction

The efficient management of spatially extended objects has become an enabling technology for many novel database applications, including computer aided design (CAD), medical imaging or molecular biology. As a common and successful approach, spatial objects can conservatively be approximated by voxels, i.e. cells of a grid covering the complete data space. An important new requirement for large objects, including cars, planes or space stations, is a high approximation quality which is primarily influenced by the resolution of the grid. Low resolutions result in large approximation errors whereas high resolutions yield a high quality but require high efforts in terms of storage space as well as update and query processing time. By means of space filling curves, each cell of the grid can be encoded by a single

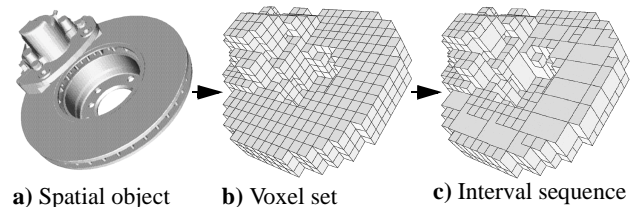


Figure 1: Conversion pipeline from spatial objects to interval sequences

integer and, thus, an extended object is represented by a set of integers. As a principal design goal, space filling curves achieve good spatial clustering properties since cells in close spatial proximity are encoded by contiguous integers. Following [16], adjacent cell values can be grouped together to *Object Interval Sequences* (cf. Figure 1) which are basic datatypes for spatial applications.

By expressing spatial region queries as intersections of interval sequences, vital operations for two-dimensional GIS and environmental information systems [18] can be supported. Efficient and scalable database solutions are also required for two- and three-dimensional CAD applications to cope with rapidly growing amounts of dynamic data and highly concurrent workflows. Such applications include the digital mock-up of vehicles and airplanes [1], haptic simulations in virtual product environments [19] or engineering data management. Furthermore, spatial databases have evolved from highly specialized applications to mainstream business software such as enterprise resource planning systems (ERP) [10].

For commercial use, a seamless and capable integration of temporal and spatial indexing into industrial-strength databases is essential. Fortunately, a lot of traditional database servers have evolved into Object-Relational Database Management Systems (ORDBMS). This means that in addition to the efficient and secure management of data ordered under the relational model, these systems now also provide support for data organized under the object model. Object types and other features, such as large objects (LOBs), external procedures, extensible indexing, user-defined aggregate functions and query optimization, can be used to build powerful, reusable server-based components.

In order to guarantee an efficient evaluation of user-defined predicates, the extensibility services of the ORDBMS offer a conceptual framework to supplement the functional evaluation of user-defined predicates with index-based look-

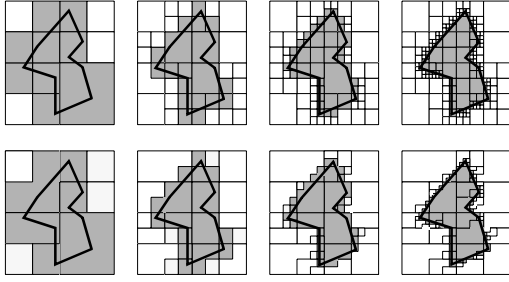


Figure 2: Quadtree tessellation (top) and Z-order interval sequence decomposition (bottom) of a 2D spatial object for various resolutions 4x4, 8x8, 16x16, and 32x32 (left to right).

ups. A wide variety of access methods for one- and multidimensional extended objects has been published so far. For a general overview on external temporal and spatial index structures, we refer the reader to the surveys of Manolopoulos, Theodoridis and Tsotras [20] or Gaede and Günther [8]. An extensive comparison of index structures for one-dimensional intervals has been done by Kriegel, Pötke and Seidl with a particular focus on relational storage and relational query processing, i.e. an implementation mainly based on SQL [15]. Furthermore, multidimensional access methods for objects with a spatial (or temporal) extension can be classified, with respect to inherent data replication, i.e. the need to produce redundancy for spatial objects or their identifiers [16].

Many of the *non-replicating access methods*, e.g. R-trees [2, 9, 14], use simple spatial primitives such as rectilinear hyperrectangles for one-value approximations of extended objects. Although providing the minimal storage complexity, one-value approximations of spatially extended objects often are far too coarse. In many applications, GIS or CAD objects feature a very complex and fine-grained geometry. The rectilinear bounding box of the brake line of a car, for example, would cover the whole bottom of the indexed data space. A non-replicating storage of such data causes region queries to produce too many false hits that have to be eliminated by subsequent filter steps. For such applications, the accuracy can be improved by decomposing the objects.

In the case of *replicating access methods*, e.g. R⁺-tree [25] or the RI-tree [16], the number of the simple spatial primitives used to approximate the objects can become very high, resulting in a storage and query processing overhead. Gaede pointed out that the number of tiles and intervals representing a spatially extended object exponentially depends on the granularity of the grid approximation [7] (cf. Figure 2). Furthermore, the extensive analysis given in [17] and [6] shows that the asymptotic redundancy of an interval-based decomposition is proportional to the surface of the approximated object. Thus, in the case of high resolution huge parts (e.g. wings of an airplane), the number of intervals can become unreasonably high.

A promising way to cope with high resolution spatial data may be found somewhere in between replicating and non-replicating spatial index structures. In this paper, we present our new concept of grouping simple black intervals into *grey intervals* which helps to range between these two extremes.

The remainder of the paper is organized as follows: Section 2 introduces grey intervals and how they can be stored in an ORDBMS. Section 3 shows that spatial query processing based on grey intervals can elegantly be expressed within the SQL-standard. In Section 4, we present the empirical results, which are based on two real-world test data sets of our industrial partners, a German car manufacturer and an American plane producer, dealing with high resolution CAD data. In Section 5, we summarize our work.

2. Grey Intervals

Interval sequences, representing high resolution spatially extended objects, often consist of very short intervals connected by short gaps. Experiments suggest that both gaps and intervals obey an exponential distribution (cf. Section 4). Thus, it seems promising to group them together to longer *grey intervals* in order to improve storage behavior and query response time.

Definition 1 (grey object interval sequences)

Let id be an *object identifier* and $W = \{(l, u) \in \mathbb{N}^2, l \leq u\}$ be the domain of intervals which we call *black intervals* throughout this paper. A black interval (l, u) contains all integers x such that $l \leq x \leq u$. Furthermore, let $b_1 = (l_1, u_1), \dots, b_n = (l_n, u_n) \in W$ be a sequence of intervals with $u_i + 1 < l_{i+1}$ for all $i \in \{1, \dots, n-1\}$. Moreover, let $m \leq n$ and let $i_0, i_1, i_2, \dots, i_m \in \mathbb{N}$ such that $0 = i_0 < i_1 < i_2 < \dots < i_m = n$ holds. Then, we call $O_{grey} = (id, \langle \langle b_{i_0+1}, \dots, b_{i_1} \rangle, \langle b_{i_1+1}, \dots, b_{i_2} \rangle, \dots, \langle b_{i_{m-1}+1}, \dots, b_{i_m} \rangle \rangle)$ a *grey object interval sequence* of cardinality m . If m equals n , we denote O_{grey} also as a *black object interval sequence* O_{black} . We call each of the $j = 1, \dots, m$ groups $\langle b_{i_{j-1}+1}, \dots, b_{i_j} \rangle$ of O_{grey} a *grey interval* I_{grey} . If $i_{j-1} + 1$ equals i_j , we denote I_{grey} also as a *black interval* I_{black} .

Intuitively, a grey interval is a covering of one or more disjoint and nonadjacent black intervals where there is at least a gap of one integer between adjacent intervals. In the next definition, we introduce a few useful operators on *grey intervals*.

Definition 2 (operators on grey intervals)

For any grey interval $I_{grey} = \langle (l_r, u_r), \dots, (l_s, u_s) \rangle$ we define the following operators:

Length: $L(I_{grey}) = u_s - l_r + 1$.

Cardinality: $C(I_{grey}) = s - r + 1$.

Number of Black Cells: $N_b(I_{grey}) = \sum_{i=r}^s (u_i - l_i + 1)$.

Number of White Cells: $N_w(I_{grey}) = L(I_{grey}) - N_b(I_{grey})$.

Density: $D(I_{grey}) = N_b(I_{grey}) / L(I_{grey})$.

Hull: $H(I_{grey}) = (l_r, u_s)$.

Gap: $G(I_{grey}) = \begin{cases} 0 & r = s \\ \max\{l_i - u_{i-1} - 1, i = r + 1, \dots, s\} & \text{else} \end{cases}$.

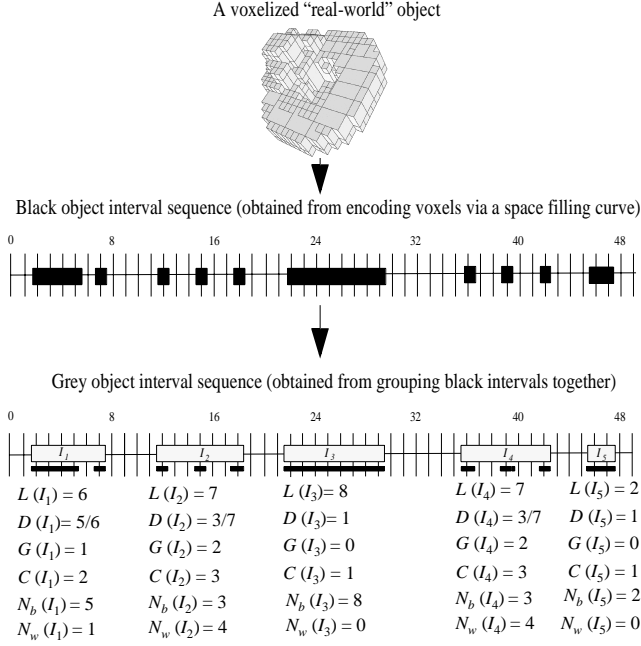


Figure 3: Grey object interval sequence

Figure 3 demonstrates the values of these operators for a sample set of grey intervals.

2.1. Storing Grey Intervals in an ORDBMS

One approach for storing an object id in a database is to map its black object interval sequence $O_{black} = (id, \langle b_1, \dots, b_n \rangle)$ to a set of n rows in an object-relational table *BlackIntervals* ($id, interval$). An index on the attribute *interval* supports efficient query processing. For high resolution spatial data, this approach yields a high number of table and index entries and, consequently, leads to a poor query response behavior.

A key idea of our approach is to store the grey object interval sequence $O_{grey} = (id, \langle \langle b_{i_0+1}, \dots, b_{i_1} \rangle, \langle b_{i_1+1}, \dots, b_{i_2} \rangle, \dots, \langle b_{i_{m-1}+1}, \dots, b_{i_m} \rangle \rangle)$ in a set of m rows in an object-relational table *GreyIntervals* ($id, intervalsequence$). In this case, the black intervals b_r, \dots, b_s of each grey interval $I_{grey} = \langle b_r, \dots, b_s \rangle$ are mapped to the complex attribute *intervalsequence* which consists of the hull $H(I_{grey})$, a *BLOB*, and the *BLOB* type, indicating the structure of the *BLOB*. The two important advantages of this approach are as follows: First, the number of table and index entries can be controlled and reduced dramatically. Secondly, the access to the grey intervals is efficiently supported by established relational access methods for intervals. These access methods have to be created on $H(I_{grey})$.

2.1.1. Storing an Interval Sequence in a BLOB. The detailed black interval sequence b_r, \dots, b_s of a grey interval $I_{grey} = \langle b_r, \dots, b_s \rangle$ can be materialized and stored in a *BLOB* in many different ways. A good materialization should consider two aspects. First, as little as possible secondary storage should be occupied. Secondly, as little as possible time

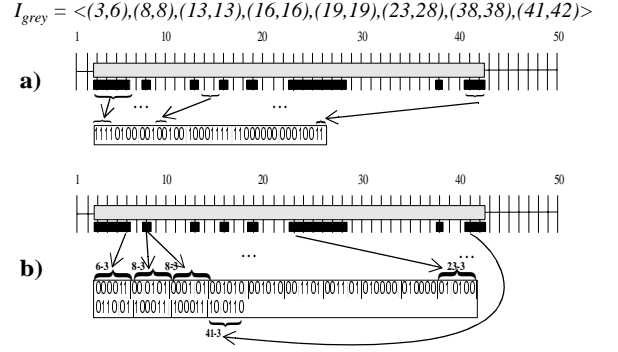


Figure 4: Storing the detailed black interval sequence in a *BLOB*
a) bit-oriented, b) offset-oriented

should be needed for the evaluation of the *BLOB*. These two requirements are in accordance with each other as long as the CPU-costs for the evaluation of the *BLOB* are negligible compared to the I/O-costs. We introduce two alternatives, the *bit-oriented* approach and the *offset-oriented* approach, and show in which case each of them uses less secondary disk space. Finally, we present a *mixed* approach which yields both, a good secondary storage behavior and a good query response time (cf. Section 4).

Bit-Oriented Approach. A very important observation is that a *black object interval sequence* for high resolution spatial objects consists of a large number of very short intervals (e.g. points) which are connected by short gaps. This motivates the bit-oriented approach illustrated in Figure 4a. Each voxel of the data space which is covered by a grey interval $I_{grey} = \langle b_r, \dots, b_s \rangle$ is represented by one bit in its *BLOB*. Thus, the size of the *BLOB* is always equal to $u_s - l_r + 1$ bits, i.e. the space complexity is $O(L(I_{grey}))$. Obviously, this approach works well for short intervals with short gaps.

Offset-Oriented Approach. Unfortunately, the *bit-oriented* approach is extremely bad if a grey interval includes very long black intervals or long gaps. Therefore, we introduce the *offset-oriented* approach (cf. Figure 4b) where we process the boundary values of the black intervals. For a grey interval $I_{grey} = \langle b_r, \dots, b_s \rangle$ with $C(I_{grey})$ black intervals, we store the values $u_r - l_r, l_{r+1} - l_r, u_{r+1} - l_r, \dots, l_{(s-1)} - l_r, u_{(s-1)} - l_r$, and $l_n - l_r$ sequentially in the *BLOB*. Each of these $1 + 2 \cdot (C(I_{grey}) - 2) + 1 = 2 \cdot (C(I_{grey}) - 1) + 1$ values is smaller than $L(I_{grey})$. Thus, only $\lceil \log_2 L(I_{grey}) \rceil$ bits are needed for storing one value (e.g. $\lceil \log_2 (42 - 3) \rceil = 6$ bits in the example of Figure 4). In the offset-oriented approach the size of the *BLOB* does not depend linearly on $L(I_{grey})$, but logarithmically. Furthermore, it depends linearly on the cardinality $C(I_{grey})$ of the grey interval, i.e. we have a space complexity of $O(C(I_{grey}) \cdot \log_2 L(I_{grey}))$.

A final remark: instead of using the *offset-oriented approach*, we could have organized the sequence b_r, \dots, b_s by means of *run-length-coding*. The result is in the same storage space complexity class but the process of accessing the

BLOB at a desired offset is only possible by scanning the BLOB from the beginning, whereas our approach allows bi-partitioning the BLOB and so a logarithmic access time is guaranteed (cf. Section 3).

Mixed approach. Obviously, it is sometimes better to use the *bit-oriented approach* which needs $O(L(I_{grey}))$ bits, and sometimes the *offset-oriented approach* which needs $O(C(I_{grey}) \cdot \log_2 L(I_{grey}))$ bits is better. Fortunately, it can be decided individually which one is preferable for each grey interval, depending on the length and the cardinality of the grey interval.

Theorem 1 (*bit-oriented versus offset-oriented approach*)

Let $I_{grey} = \langle b_r, \dots, b_s \rangle$ be a grey interval. Then, the *bit-oriented approach* needs less secondary disk space compared to the *offset-oriented approach* for storing the sequence b_r, \dots, b_s if the following formula holds

$$L(I_{grey}) < 2 \cdot (C(I_{grey}) - 1) \cdot \lceil \log_2 L(I_{grey}) \rceil.$$

Proof: The *bit-oriented approach* needs $L(I_{grey})$ bits for storing the sequence b_r, \dots, b_s in a BLOB. On the other hand, we have to store $2 \cdot (C(I_{grey}) - 1)$ boundary values of which each needs $\lceil \log_2 L(I_{grey}) \rceil$ bits in the *offset-oriented approach*. Thus, the formula holds.

Based on this theorem, we can decide for each grey interval whether the *offset-oriented* or the *bit-oriented approach* needs less secondary storage.

3. Query Processing

Most ORDBMSs, including Oracle [21, 24], IBM DB2 [5, 12] or Informix IDS/UDO [4, 13], provide extensibility interfaces in order to enable database developers to seamlessly integrate custom object types and predicates within the declarative DDL and DML. These interfaces form a necessary prerequisite for the seamless embedding of spatial objects and the *intersect* predicate into off-the-shelf ORDBMSs. As we represent spatial objects by grey object interval sequences, we first clarify when two of these sequences intersect.

Definition 3 (object intersection)

Let $W_{black} = \{(l, u) \in \mathbb{N}^2, l \leq u\}$ be the domain of black intervals, and let $I^1 = \langle b_1^1, \dots, b_{n_1}^1 \rangle$ and $I^2 = \langle b_1^2, \dots, b_{n_2}^2 \rangle$ be two grey intervals. Furthermore, let $O^1 = (id^1, \langle I_1^1, I_2^1, \dots, I_{m_1}^1 \rangle)$ and $O^2 = (id^2, \langle I_1^2, I_2^2, \dots, I_{m_2}^2 \rangle)$ be two grey object interval sequences. Then, the notions *intersect* and *interlace* are defined in the following way (cf. Figure 5):

1. Two black intervals, $b_1 = (l_1, u_1)$ and $b_2 = (l_2, u_2)$, *intersect* if $l_1 \leq u_2$ and $l_2 \leq u_1$.
- 2a. Two grey intervals I^1 and I^2 *intersect* if for any $i \in \{1, \dots, n_1\}, j \in \{1, \dots, n_2\}$, the black intervals b_i^1 and b_j^2 intersect.
- 2b. Two grey intervals I^1 and I^2 *interlace*, if their hulls, $H(I^1)$ and $H(I^2)$ intersect.
- 3a. Two objects O^1 and O^2 *intersect* if for any $i \in \{1, \dots, m_1\}, j \in \{1, \dots, m_2\}$, the grey intervals I_i^1 and I_j^2 intersect.
- 3b. Two objects O^1 and O^2 *interlace* if for any $i \in \{1, \dots, m_1\}, j \in \{1, \dots, m_2\}$, the grey intervals I_i^1 and I_j^2 interlace.

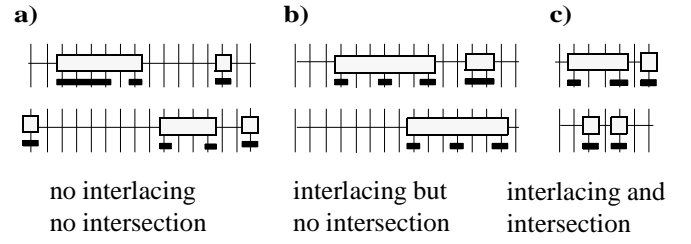


Figure 5: grey object interval sequences
a) non interlacing, b) interlacing but no intersection,
c) intersection

3.1. The *intersect* SQL Statement

As shown in Section 2.3, the *black object interval sequences* can be mapped to an object-relational schema *BlackIntervals*. Following this approach yields a clear SQL-statement for the detection of intersecting spatial objects (cf. Figure 6). In order to efficiently determine all intersecting black intervals, we can use one of the many index structures for intervals, surveyed in [15], and integrate them into the extensible indexing framework of modern ORDBMSs.

```
SELECT DISTINCT db.id
FROM BlackIntervals db, :BlackQueryIntervals q
WHERE intersects (db.interval, q.interval)
```

Figure 6: SQL statement for spatial object intersection, based on black object interval sequences

These index structures also support the evaluation of the *intersect* predicate based on grey intervals. As we defined grey object interval sequences as a conservative approximation of black object interval sequences, we can use the hulls of the grey intervals in a first conservative filter step. Thereby, we can take advantage of the same access methods as used for the detection of intersecting black interval pairs. As shown in Section 2.1, the *grey object interval sequences* can be mapped to an object-relational schema *GreyIntervals*. Following this approach, we can also clearly express the *intersect* predicate on top of the SQL engine (cf. Figure 7).

We use *table* as a nesting function that groups references of interlacing grey query and database interval pairs together. In our implementation, we realized this NF2-operator *table* by a user-defined aggregate function as provided in the SQL:1999 standard. As we want to find out which database objects are intersected by a specific query object, we have to test the interlacing grey intervals for intersection. This test is carried out by a stored procedure *blobintersection*. If we find one intersecting grey database and query interval pair, we can stop investigating other grey interlacing interval pairs belonging to the same database object, and issue the object's *id*. This *skipping principle* is realized by means of the *exists*-clause within the SQL-statement.

```

SELECT candidates.id FROM
(
  SELECT db.id AS id,
         table(pair(db.rowid, q.rowid)) AS ctable
  FROM GreyIntervals db, :GreyQueryIntervals q
  WHERE intersects (hull(db.intervalsequence),
                  hull(q.intervalsequence))    -- interlacing
  GROUP BY db.id
) candidates
WHERE EXISTS
(
  SELECT 1
  FROM GreyIntervals db, :GreyQueryIntervals q,
         candidates.ctable ctable
  WHERE db.rowid = ctable.dbrowid AND
        q.rowid = ctable.qrowid AND
        blobintersection (db.intervalsequence,
                          q.intervalsequence)    -- intersection
)

```

Figure 7: SQL statement for spatial object intersection, based on grey object interval sequences

The approach introduced above actually forms a new spatial access method which can easily be integrated into common extensible indexing frameworks [22].

3.1.1. Stored Procedure blobintersection. In order to decide whether two interlacing grey database and query intervals intersect, the detailed black interval sequences have to be scrutinized. If an intersection is detected, the procedure *blobintersection* can stop testing the interlacing area furthermore. Thus, unnecessary disk accesses are avoided.

If the grey interval $I_{grey} = \langle b_r, \dots, b_s \rangle$ is *bit-oriented* and has to be examined in the area $L_{interlace} = (l_{interlace}, u_{interlace})$, we can find the starting point of the interlacing area $l_{interlace}$ in constant time by using $l_{interlace} - l_r$ as offset. We then have to examine at most the $L_{interlace}$ bits of this area.

If the grey interval $I_{grey} = \langle b_r, \dots, b_s \rangle$ follows the *offset-oriented* approach, the starting point of the interlacing area $l_{interlace}$ of I_{grey} can be found by bipartitioning. We first access the value in the middle of the BLOB and compare it to $l_{interlace}$. If it is smaller, we only have to consider the upper half of the BLOB. If it is higher, we take the lower half. This test can be done in constant time and has to be performed at most $1 + \log_2 C(I_{grey})$ times. Then, we have to access the black intervals in the interlacing area which are consecutively organized.

3.2. Optimizations

For each database object which interlaces the query object, it suffices to find a single intersecting interval pair in order to issue the database *id*. Obviously, it is desirable to detect such intersecting interval pairs as early as possible in order to avoid unnecessary *blobintersection* tests.

In this section, we present two optimizations striking for this goal. First, we introduce a fast second filter step which

tries to determine intersecting intervals without examining the detailed black interval sequence of a grey interval. This test is entirely based on aggregated information of the grey intervals. Secondly, we introduce a probability model which leads to an ordering for the interlacing intervals such that the most promising *blobintersection* tests are carried out first.

In order to put these optimizations into practice, we pass $G(I_{grey})$ and $D(I_{grey})$ (cf. Definition 2) as additional parameters to the user-defined aggregate function *table*. Thus, the following two optimizations can easily be integrated into this user-defined aggregate function. For efficiency reason, we materialize not only $H(I_{grey})$ (cf. Section 2.1), but also $G(I_{grey})$ and $D(I_{grey})$ in the complex attribute *intervalsequence* of the table *GreyIntervals*. If the fast second filter step determines an intersecting pair of intervals, all other interval pairs are deleted so that the resulting table of candidate interval pairs, called *ctable*, consists only of one intersecting interval pair. If this test detects that the intervals cannot intersect, this interval pair is not added to *ctable*. Nevertheless, there might be database objects where for none of the corresponding interval pairs this second filter step determines an intersection. In this case, we sort the interval pairs at the end of our user-defined aggregation function *table* such that the most promising *blobintersection* tests are carried out first.

3.2.1. Fast Grey Test. In the following paragraph, we will discuss how grey and black intervals have to look like so that we can decide whether two interlacing intervals intersect each other or not without accessing their BLOBs.

Two Black Intervals. If two black intervals interlace, they necessarily intersect as well.

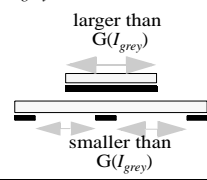
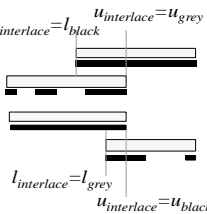
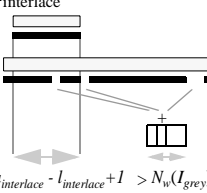
Black and Grey Intervals. In this case, the situation is a little bit more complicated. In almost any cases where a black interval $I_{black} = (l_{black}, u_{black})$ interlaces a grey interval I_{grey} with $H(I_{grey}) = (l_{grey}, u_{grey})$, there is an intersection of I_{black} and I_{grey} as well. If any of the three conditions depicted in Table 1 holds, then I_{black} and I_{grey} intersect.

Two Grey Intervals. Obviously, two grey intervals I_{grey} and I'_{grey} , with $H(I_{grey}) = (l_{grey}, u_{grey})$ and $H(I'_{grey}) = (l'_{grey}, u'_{grey})$, which interlace do not have to intersect. Fortunately, there are two cases where we can assert that they intersect without examining the detailed black interval sequences. The two cases are illustrated in Table 2.

No intersection. There are only two situations, depicted in Table 3, where we can determine that two interlacing intervals do *not* intersect.

3.2.2. Ranking. As shown above, we can pinpoint, based on relatively little information, whether two interlacing intervals intersect or not. Nevertheless, there might be cases where we cannot do this for any interlacing database and query interval pair. In this situation, it is still helpful if we can predict how likely an intersection might be in order to rank this interval pair properly in the set of all interlacing

Table 1: Intersection between an interlacing black and grey interval

condition	explanation
$L(I_{black}) > G(I_{grey})$ 	<p>If the black interval is longer than the maximum gap between two black intervals of the detailed black interval sequence of I_{grey} then the two intervals intersect.</p>
$(l_{interlace} = l_{black} \text{ and } u_{interlace} = u_{grey})$ or $(l_{interlace} = l_{grey} \text{ and } u_{interlace} = u_{black})$ 	<p>If one of the two conditions presented in the box on the left holds, then the black and grey interval intersect. This is due to the fact that the grey intervals end and start with black intervals.</p>
$N_w(I_{grey}) < L_{interlace}$ 	<p>If the number of the white cells $N_w(I_{grey})$ of a grey interval is smaller than the length of the interlacing area, then the grey and the black interval necessarily intersect.</p>

query and database interval pairs belonging to the same database object.

Theorem 2 (Probability for an intersection of interlacing intervals)

Let I_{grey} and I'_{grey} be two grey intervals with densities $d = D(I_{grey})$ and $d' = D(I'_{grey})$ which interlace within an area of length L . Furthermore, let the black and white cells of the two intervals be equally distributed. Then the probability for an intersection is:

$$P = 1 - \frac{\binom{L-d \times L}{d' \times L}}{\binom{L}{d' \times L}}$$

Proof: As we assume that the black cells of both grey intervals are equally distributed, we can conclude that $N = d \times L$ black cells of I_{grey} are included in the interlacing area and likewise $N' = d' \times L$ black cells from I'_{grey} . The number of all different possibilities how N' black cells of I'_{grey} can be placed over the L cells of the interlacing area is $\binom{L}{N'} = \binom{L}{d' \times L}$. Assuming that all N black cells of I_{grey} are already distributed over L , then the number of different possibilities how N'

Table 2: Intersection between two interlacing grey intervals

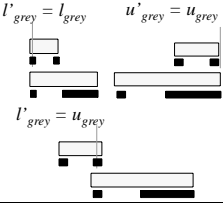
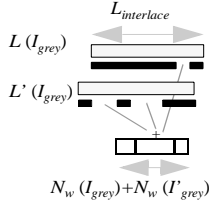
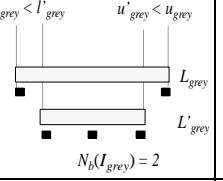
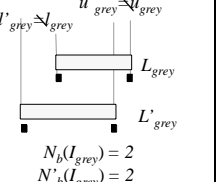
condition	explanation
$u_{grey} = u'_{grey}$ OR $l_{grey} = l'_{grey}$ OR $l_{grey} = u'_{grey}$ 	<p>If one of the three conditions depicted in the box on the left holds, then the two grey intervals intersect (grey intervals start and end with black intervals). This test is similar to the <i>polygon boundary test</i> in [11].</p>
$N_w(I_{grey}) + N_w(I'_{grey}) < L_{interlace}$ 	<p>If the sum of the number of the “white cells” of two grey interlacing intervals is smaller than the length of the interlacing area, then the two intervals necessarily intersect. This is the generalization of the third case of Table 1. This test is similar to the <i>false area test</i> in [3].</p>

Table 3: No intersection between two interlacing grey intervals

condition	explanation
$N_b(I_{grey}) = 2$ and $l_{grey} < l'_{grey}$ and $u_{grey} > u'_{grey}$ 	<p>If I_{grey} consists only of two black cells and I'_{grey} is totally “included” in I_{grey}, then we know that the two intervals cannot intersect each other, although they interlace.</p>
$N_b(I_{grey}) = 2$ $N_b(I'_{grey}) = 2$ and $l_{grey} \neq l'_{grey} \neq u_{grey} \neq u'_{grey}$ 	<p>If both grey intervals consist only of two black cells and, furthermore, have distinct interval bounds, then the two intervals certainly do not intersect.</p>

black cells of I'_{grey} can be placed over the remaining $L-N$ white cells such that no intersection occurs is $\binom{L-N}{N'} = \binom{L-d \times L}{d' \times L}$. Thus, the probability for a *non-intersection* is equal to $\frac{\binom{L-d \times L}{d' \times L}}{\binom{L}{d' \times L}}$, which proves the theorem.

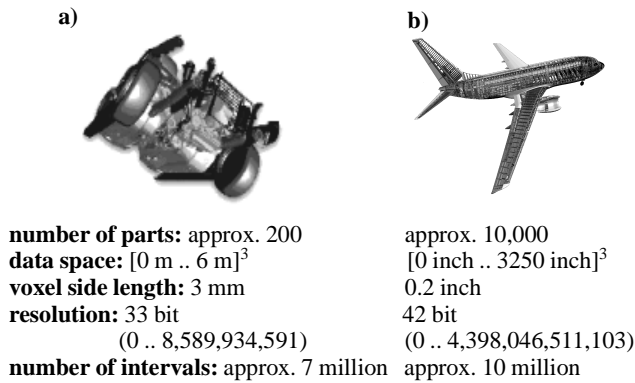


Figure 8: High resolution CAD test data sets
a) CAR, b) PLANE

4. Experimental Evaluation

In this section, we evaluate the performance of the grey object interval sequences based on two test data sets *CAR* and *PLANE* (cf. Figure 8). These test data sets were provided by our industrial partners, a German car manufacturer and an American plane producer, in form of high resolution voxelized three-dimensional CAD parts. In both cases, the Z-curve was used as a space filling curve to enumerate the voxels.

In order to support the first filter step of our new *High Resolution Indexing* method *HRI*, we can take an arbitrary access method for intervals as presented in [15]. Throughout our experiments we have used the relational interval tree for this purpose. Many spatial access methods compete with our *HRI* technique but only few of them may be integrated into standard SQL database servers. Examples include the *Linear Segment Tree*, the *Linear Quadtree (Octree)*, the *Relational R-tree* and the *spatial version of the RI-tree* [16]. Since the latter outperforms competing approaches by factors between 4.6 and 58.3 for query response time [16], we compared the *HRI* method to the optimized spatial version of the *RI-tree* which deals efficiently with black interval sequences.

In order to evaluate the *HRI* method, we have grouped black object interval sequences into different grey object interval sequences depending on a *MAXGAP* parameter. The grouping algorithm tries to minimize the number of grey intervals while not allowing that a maximum gap $G(I_{grey})$ of any grey interval exceeds the *MAXGAP* parameter. Let us note that the *HRI* method and the *RI-tree* coincide if the *MAXGAP* parameter is 0. In this case, we always used the original optimized version of the *RI-tree* and not the *HRI* method.

We have implemented the optimized *RI-tree* and the *HRI* method on top of the Oracle9i Server using PL/SQL for the computational main memory based programming. All experiments were performed on a Pentium III/700 machine with IDE hard drives. The database block cache was set to 500 disk blocks with a block size of 8 KB and was used exclusively by one active session.

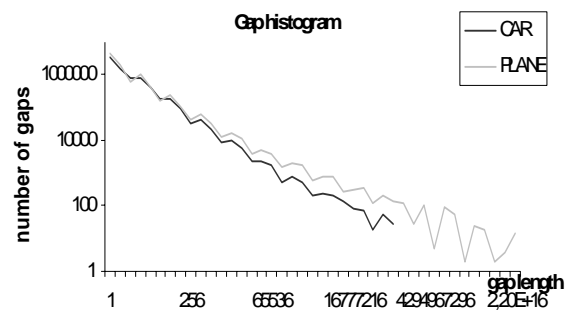


Figure 9: Gap histogram

4.1. Gap and Interval Histograms

As shown by Gaede [7] and Faloutsos et al. [6], the number of intervals generated via a space-filling curve out of a real-world object mainly depends on the surface and the shape of the objects and on the resolution of the underlying grid. Unfortunately, there is nothing mentioned about the distribution of the intervals or the corresponding gap distribution. In [23] it is asserted that the gap histograms show local peaks at gap lengths around 2^{3k} with $k \in \mathbb{N}$ and $k \geq 0$ for 3D data. This behavior is caused by the fact that many gaps represent empty cube-like (3D) regions at the boundary of the spatial objects. Figure 9 supports this assertion. Figure 10 depicts the interval distribution. It can be seen that the bucket which includes most intervals is regularly increasing with increasing *MAXGAP*.

4.1.1. Storage Requirements. Although the storage complexity $O(n/b)$ of the *RI-tree* is optimal, it seems rather wasteful to spend one row in the table *BlackIntervals* of the *RI-tree* for each short interval, especially if these black intervals are connected to each other by short gaps. Figure 11a shows the different storage requirements for the *GreyIntervals* table with respect to the different organization approaches of the *BLOBs*. As you can see, the bit-oriented approach is very bad for high *MAXGAP* values, but it is better than the offset-oriented approach when using small *MAXGAP* values. The mixed approach combines the advantages of both and is, therefore, used throughout the rest of this paper. In Figure 11b, the storage requirements for the index as well as for the complete *GreyIntervals* table are depicted. In the case of small *MAXGAP* parameters, the number of disk

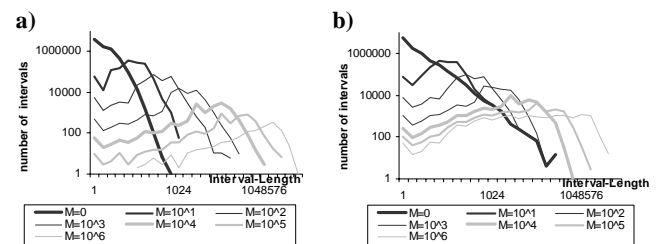


Figure 10: Interval length depending on the *MAXGAP* parameter

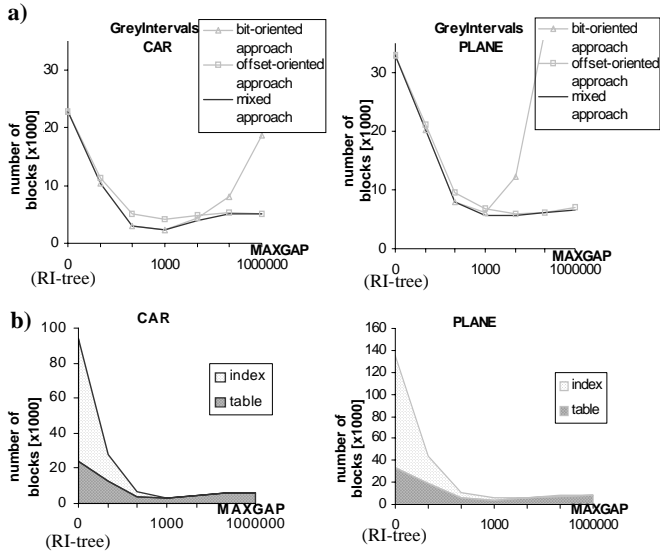


Figure 11: Storage requirements

a) GreyIntervals table , b) GreyIntervals table + index

blocks used by the index dominates the number of disk blocks for the *GreyIntervals* table. With increasing *MAXGAP* parameters the number of disk blocks used by the index dramatically decreases hand in hand with the number of grey intervals, and at high parameter values they yield no significant contribution any more to the overall sum of used disk blocks.

Observation 1

With the HRI method we can improve the storage requirement at least by an order of magnitude.

4.2. Evaluation of the Dynamic Properties of the HRI Method

In this section, we want to turn our attention to the different facets related to the query response behavior of the HRI method where we consider *collision queries*.

All figures presented in this paragraph depict the average result obtained from collision queries where we have taken every physical mechanical part from both test data sets *CAR* and *PLANE* as query objects in order to determine which parts in the associated database are colliding with the query object. We first discuss the overall runtime behavior, and then, investigate the number of tested candidates. Finally, we close with a few general remarks on miscellaneous facets.

Response time. In Figure 12, it is shown in which way the overall response time depends on the *MAXGAP* parameter. As the second filter step does not require any measurable time, it is not visible there. If we use small *MAXGAP* parameters, we still need a lot of time for the first filter step. On the other hand, using large values leads to an expensive BLOB

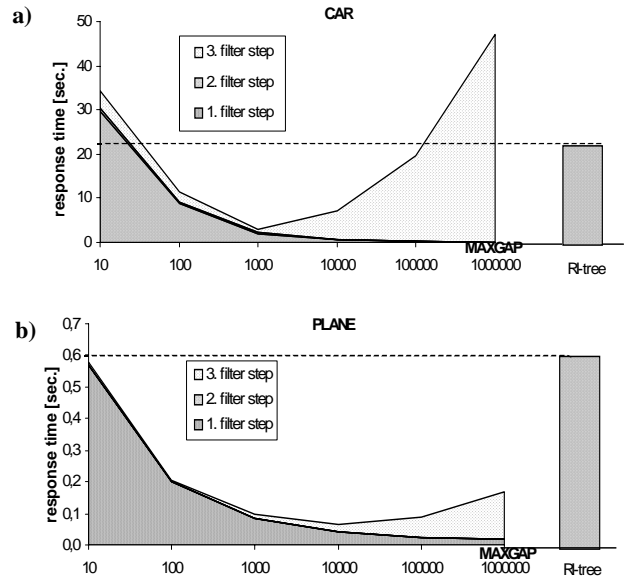


Figure 12: Response time on collision queries

a) CAR, b) PLANE

test. Fortunately, using *MAXGAP* parameters in the middle leads to a good query response time.

Observation 2

With the HRI method we can improve the response time of collision queries by an order of magnitude.

Number of Tested candidates. Figure 13 illustrates the number of interval candidate pairs and the number of the corresponding tests which are actually carried out. In the *second filter step*, the number of the candidate pairs rapidly decreases with increasing *MAXGAP* value although the number of candidate object IDs increases (cf. Figure 14). At low *MAXGAP* values, we have to test only a fractional amount of candidate pairs as the *fast second filter step* works very successfully with this parametrization (cf. Figure 13). Consequently, there is only a relative small number of candidate pairs left for the *blobintersection* tests.

In the *blobintersection step*, the number of both candidate pairs and corresponding tests do not vary as much as in the second step. We can still see that in the case of the best response time on the *CAR* data (i.e. *MAXGAP* equals 1,000), we only have to test 40% of all candidates and, thus, this step benefits as well from the *skipping principle* introduced in Section 3.1.

In Figure 14, it is illustrated that at small *MAXGAP* values the number of the different object IDs resulting from the first filter step is only marginally higher than the number of different IDs in the final result set. Likewise, the number of detected hits in the second filter step is only marginally

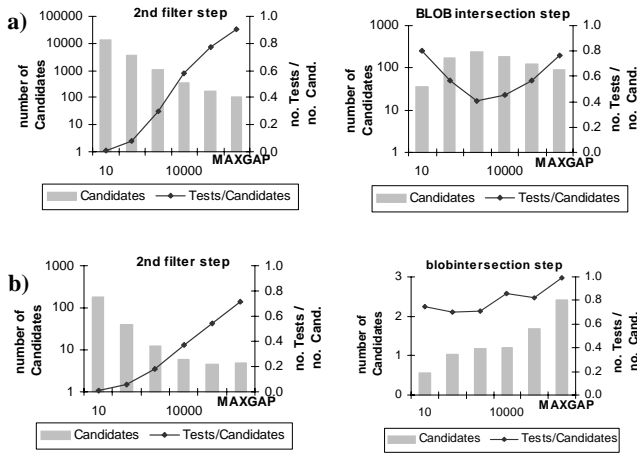


Figure 13: Tested candidate pairs of query and database intervals a) CAR, b) PLANE

smaller. With increasing *MAXGAP* values the two curves diverge.

Miscellaneous. The size of the parts in the *PLANE* data set varies considerably. We have a lot of small parts and only a few very large ones. In the case of the *CAR* data, this peculiarity is far less distinctive. As large query parts produce a large number of query intervals, it is obvious that the size of a part correlates with the response time. In Figure 15a, it is shown that for most parts from the *PLANE* data set the HRI method (*MAXGAP* = 10,000) outperforms the RI-tree ‘only’ by a factor of 2.9 whereas there are some parts for which this

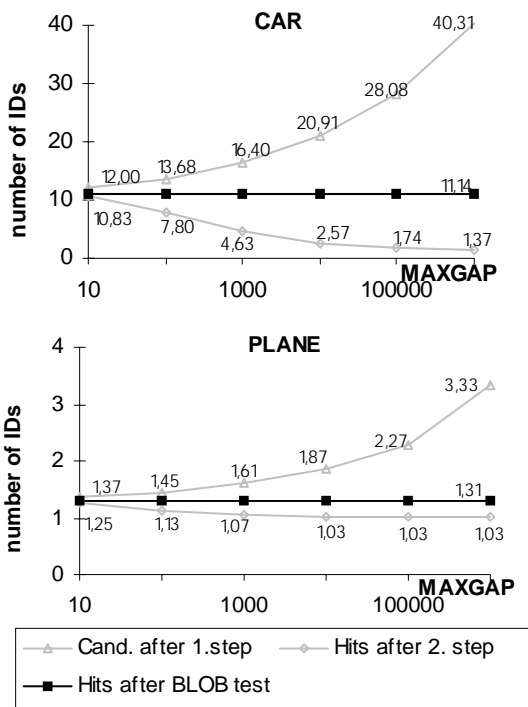


Figure 14: Candidate IDs and result sets

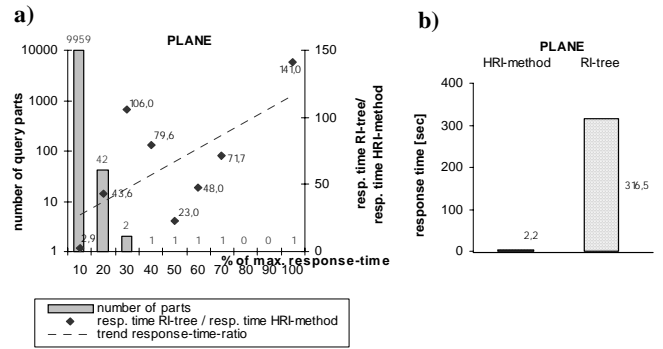


Figure 15: Response time
a) |resp. time RI-tree | / |resp. time HRI method|,
b) maximum response times

factor is higher than 100. In Figure 15b, it is illustrated that we have to wait for more than five minutes for some collision queries when using the RI-tree. On the other hand, using the HRI method yields almost interactive response times for all collision queries.

Observation 3

If we use voluminous high resolution spatial query objects, leading to high response times, we particularly benefit from the HRI method.

An interesting observation is that the minimum of secondary storage requirement (cf. Figure 11) and the minimum of the response time (cf. Figure 12) are found at the same value of the *MAXGAP* parameter, i.e. 1,000 on the *CAR* data and 10,000 on the *PLANE* data. This value of *MAXGAP* coincides with the smallest *MAXGAP* value for which the height of the B^+ -directory is 1 (cf. Figure 16). Smaller *MAXGAP* values lead to more grey intervals and, thus, to a higher B^+ -directory.

5. Conclusion

In this paper, we present a new multi-step architecture for querying high resolution spatial objects on top of standard database systems, called *High Resolution Indexing* (HRI). As the key to HRI we introduce a novel concept of *grey approximations* and apply it to the representation of spatial objects by interval sequences which are obtained from encoding voxels by means of space filling curves. As the

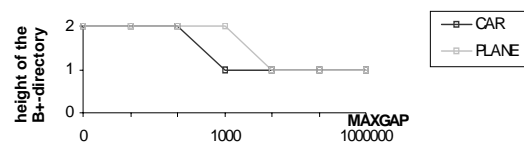


Figure 16: Height of the B^+ -directory of RI-tree

number of intervals which represent high resolution spatial objects can become unreasonably high, we reduce this number by grouping intervals of a sequence into *grey intervals*. Technically, small gaps between consecutive black intervals are closed. In order to guarantee complete and correct answer sets in the query process, three types of information for a grey interval are stored in the database: First, the hull of the grey interval, second, aggregated information including density and maximum gap length and, third, the exact black interval sequence. This information about grey intervals is reflected in the three major steps of the query process as follows:

- In a first filter step, any standard index for interval data is employed in order to determine all interlacing pairs of grey database and query intervals.
- In a fast second step we identify some actually intersecting black intervals by examining aggregated information of the grey intervals. Furthermore, the grey interval pairs are released in an order that reflects the probability whether the exact black interval sequences intersect or not.
- Finally, an expensive *BLOB test* is carried out, scrutinizing the exact black interval sequences.

We have implemented our new access method on top of the Oracle9i Server. The experimental evaluation of the HRI method can be summarized as follows: First, using the HRI method improves the secondary *storage requirement* at least by an order of magnitude. Second, using the HRI method improves the *response time* of collision queries by an order of magnitude.

In our future work we plan to extend the idea of grey approximations to other access methods, for instance the use of grey rectangles in R-trees.

6. References

- [1] Berchtold S., Kriegel H.-P., Pötke M.: *Database Support for Concurrent Digital Mock-Up*. Proc. IFIP Int. Conf. PROLAMAT, Globalization of Manufacturing in the Digital Communications Era of the 21st Century: Innovation, Agility, and the Virtual Enterprise, Kluwer Academic Publishers, 499-509, 1998.
- [2] 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, 322-331, 1990.
- [3] Brinkhoff T., Kriegel H.-P., Schneider R., Seeger B.: *Multi-Step Processing of Spatial Joins*. Proc. ACM SIGMOD Int. Conf. on Management of Data, 197-208, 1994.
- [4] Bliujute R., Saltenis S., Slivinskas G., Jensen C. S.: *Developing a DataBlade for a New Index*. Proc. 15th Int. Conf. on Data Engineering (ICDE), 314-323, 1999.
- [5] Chen W., Chow J.-H., Fuh Y.-C., Grandbois J., Jou M., Mattos N., Tran B., Wang Y.: *High Level Indexing of User-Defined Types*. Proc. 25th Int. Conf. on Very Large Databases (VLDB), 554-564, 1999.
- [6] Faloutsos C., Jagadish H. V., Manolopoulos Y.: *Analysis of the n-Dimensional Quadtree Decomposition for Arbitrary Hyperrectangles*. IEEE TKDE 9(3): 373-383, 1997.
- [7] Gaede V.: *Optimal Redundancy in Spatial Database Systems*. Proc. 4th Int. Symp. on Large Spatial Databases (SSD), LNCS 951: 96-116, 1995.
- [8] Gaede V., Günther O.: *Multidimensional Access Methods*. ACM Computing Surveys 30(2): 170-231, 1998.
- [9] Guttman A.: *R-trees: A Dynamic Index Structure for Spatial Searching*. Proc. ACM SIGMOD Int. Conf. on Management of Data, 47-57, 1984.
- [10] Günther O.: *Looking Both Ways: SSD 1999 ± 10*. Proc. 6th Int. Symp. on Large Spatial Databases (SSD), LNCS 1651: 12-15, 1999.
- [11] Huang Y.-W., Jing N., Rundensteiner E. A.: *A Cost Model for Estimating the Performance of Spatial Joins Using R-trees*. Proc. 9th Int. Conf. on Scientific and Statistical Database Management (SSDBM), 30-38, 1997.
- [12] IBM Corp.: *IBM DB2 Universal Database Application Development Guide, Version 6*. Armonk, NY, 1999.
- [13] Informix Software, Inc.: *DataBlade Developers Kit User's Guide, Version 3.4*. Menlo Park, CA, 1998.
- [14] Kamel I., Faloutsos C.: *Hilbert R-tree: An Improved R-tree Using Fractals*. Proc. ACM SIGMOD Int. Conf. on Management of Data, 500-509, 1994.
- [15] Kriegel H.-P., Pötke M., Seidl T.: *Managing Intervals Efficiently in Object-Relational Databases*. Proc. 26th Int. Conf. on Very Large Databases (VLDB), 407-418, 2000.
- [16] Kriegel H.-P., Pötke M., Seidl T.: *Interval Sequences: An Object-Relational Approach to Manage Spatial and Temporal Data*. Proc. 7th Int. Symposium on Spatial and Temporal Databases (SSTD), LNCS 2121: 481-501, 2001.
- [17] Moon B., Jagadish H. V., Faloutsos C., Saltz J. H.: *Analysis of the Clustering Properties of Hilbert Space-filling Curve*. Tech. Rep. CS-TR-3611, University of Maryland, 1996.
- [18] Medeiros C. B., Pires F.: *Databases for GIS*. ACM SIGMOD Record, 23(1): 107-115, 1994.
- [19] McNeely W. A., Puterbaugh K. D., Troy J. J.: *Six Degree-of-Freedom Haptic Rendering Using Voxel Sampling*. Proc. ACM SIGGRAPH, 401-408, 1999.
- [20] Manolopoulos Y., Theodoridis Y., Tsotras V. J.: *Advanced Database Indexing*. Boston, MA: Kluwer, 2000.
- [21] Oracle Corp.: *Oracle8i Data Cartridge Developer's Guide, Release 2 (8.1.6)*. Redwood Shores, CA, 1999.
- [22] Pfeifle M.: *Object-Relational Management of High-Resolution CAD Databases*. Diploma Thesis, University of Munich, 2001.
- [23] Pötke M.: *Spatial Indexing for Object-Relational Databases*. Ph.D. Thesis, Faculty for Mathematics and Computer Science, University of Munich, 2001.
- [24] Srinivasan J., Murthy R., Sundara S., Agarwal N., DeFazio S.: *Extensible Indexing: A Framework for Integrating Domain-Specific Indexing Schemes into Oracle8i*. Proc. 16th Int. Conf. on Data Engineering (ICDE), 91-100, 2000.
- [25] Sellis T., Roussopoulos N., Faloutsos C.: *The R⁺-Tree: A Dynamic Index for Multi-Dimensional Objects*. Proc. 13th Int. Conf. on Very Large Databases (VLDB), 507-518, 1987.