Efficient Spatial Query Processing in Geographic Database Systems

Hans-Peter Kriegel, Thomas Brinkhoff, Ralf Schneider

Institute for Computer Science, University of Munich
Leopoldstr. 11 B, W-8000 München 40, Germany
e-mail: {kriegel,brink,ralf}@dbs.informatik.uni-muenchen.de

1 Introduction

The management of spatial data in *geographic information systems (GISs)* gained increasing importance during the last decade. Due to the high complexity of objects and queries and also due to extremely large data volumes, *geographic database systems* impose stringent requirements on their storage and access architecture with respect to efficient spatial query processing.

Geographic database systems are used in very different application environments. Therefore, it is not possible to find a compact set of operations fulfilling all requirements of geographic applications. But as described in [BHKS 93], *spatial selections* are of great importance within the set of spatial queries and operations. They do not only represent an own query class, but also serve as a very important basis for the operations such as the nearest neighbor query and the spatial join. Therefore, an efficient implementation of spatial selections is an important requirement for good overall performance of the complete geographic database system. The two main representatives of spatial selections are the point and the window query (see figure 1):

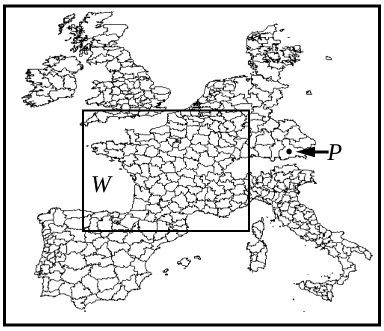

Figure 1: Examples of spatial selections

- *Point query:* Given a query point *P* and a set of objects *M*, the point query yields all the objects of *M* geometrically containing P .
- *Window query:* Given a rectilinear rectangle *W* and a set of objects *M*, the window query yields all the objects of *M* sharing points with *W*.

For the efficient processing of spatial queries, we present a *multi-step-procedure* (see figure 5). The main goal of our *spatial query processor* is to reduce expensive steps by preprocessing operations in the preceding steps which reduce the number of objects investigated in an expensive step. In figure 5 expensive steps are marked with a "$"-symbol.

A spatial selection is abstractly executed as a sequence of steps: First, we scale down the search space by *spatial indexing*. A *spatial access method (SAM)* organizes pages containing sets of entries that consist of a geometric key, an object identifier (ID) and a reference to the exact object geometry. Due to the arbitrary complexity of real geographic objects, it is not advisable to build up an index using the complete geometric description of the objects as a key. Instead, approximations of the objects are used. Thus, the spatial access method is not able to yield the exact result of a query. However, because of the high selectivity of spatial queries a high number of objects is filtered out.

Pages containing results are transferred into main memory. If a page region fulfills the query condition (e.g. the whole page region is contained in the query window), all entries of the page correspond to correct answers of the query *(hits)*. Otherwise, a page consists of entries which may fulfill the query. Therefore, we must inspect these candidates using a *geometric filter* which tests the geometric key (or further approximations of the object geometry) against the query condition. As a result, we obtain three classes of objects: *hits* fulfilling the query, *false hits* not fulfilling the query and *candidates* possibly fulfilling the query.

Now, we have to distinguish two cases: In the first case, we are only interested in the object identifiers. Then, the hits are a subset of the set of answers (in figure 5 case 1 is indicated by a striped arrow). Only the candidates have to be transferred into main memory for further processing. In the other case, we require the complete object geometry as an answer to the query. Therefore, we need a *transfer of the exact geometry* of hits *and* candidates into main memory. The transfer of exact geometry may be very expensive because the exact representation of an object can be large (compare e.g. [BHKS 93]) and additionally for large window queries, large numbers of objects have to be transferred. In window queries the transferred objects are spatially adjacent. A physically contiguous storage of spatially adjacent objects is necessary to support a fast set-oriented access by the I/O-transfer unit.

The other expensive step is *processing the exact geometry* of an object. After filtering we have to investi-

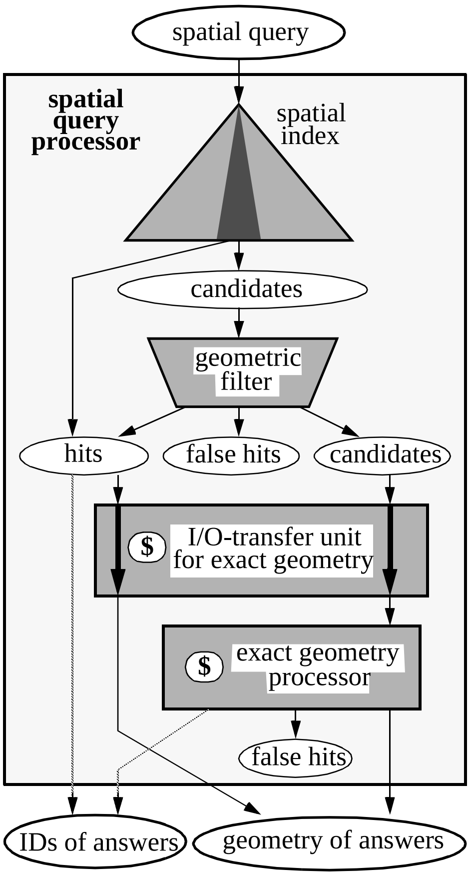

Figure 2: Spatial query processor

gate the remaining candidates. Using complex computational geometry algorithms, it is finally decided whether a candidate fulfills the query or not.

In order to reduce query time, we have to reduce the cost for the expensive operations and to refine preceding steps. In this paper, we present several techniques realizing these goals. First, we shortly discuss spatial access methods to scale down the search space efficiently. In section 3, geometric filtering techniques using several approximations are proposed. Afterwards, the efficient transfer of the exact geometry is discussed. The exact geometry processing supported by decomposition is topic of section 5. The paper concludes with a presentation of a concrete spatial query processor and suggestions for future work. We would like to emphasize that all presented techniques have been implemented and tested with real cartography data.

2 Scaling down the search space

Considering spatial selections in more detail, it turns out that generally a small and locally restricted part of the complete search space has to be investigated. For an efficient scaling down of the search space, it is essential to use *spatial access methods (SAMs)* because high volumes of data have to be organized. Access methods as a part of the internal level of a database system are used to organize a dynamic set of objects on secondary storage. One-dimensional access methods like B-trees or linear hashing are not suitable for geographic database systems. For these systems, we have to provide data structures which organize the spatial objects with respect to their location and extension in the data space. Because of the arbitrary complexity of spatial objects, access methods for simpler two-dimensional objects like minimum bounding boxes are widely discussed in the literature, e.g. the grid file, the quadtree, the buddy tree, the R-tree, and the R*-tree. Samet provides an excellent survey [Sam 90] of almost all of these methods.

Simply stated, SAMs are based on point access methods using one of three techniques [SK 88]: *Clipping* partitions the data space into disjoint regions. The objects are associated with each of the regions they intersect and thus one object is stored in each of the corresponding blocks. In general, the technique of clipping may degrade query performance substantially since the number of objects (copies) to be stored increases which in turn increases the number of regions and thereby increasing the number of copies - a vicious circle. The *transformation technique* views an object as a point in some parameter space. Since transformations do not preserve the spatial neighborhood of objects in the original space, and since the distribution of parameter points is extremely skewed, the query efficiency tends to be quite low. The third technique is *overlapping regions*. In this technique each object is assigned to exactly one region. However, there may exist several regions potentially containing the searched object.

Performance comparisons (e.g. [KSSS 89], [HS 92]) demonstrate that the well-known SAMs do not significantly differ in performance.We favor the R*-tree [BKSS 90], an improved variant of the well-known R-tree [Gut 84], because it is a simple, robust, and efficient spatial access method. This has been demonstrated in tests [BKSS 90] and in a comparison with other access methods [HS 92]. The R*-tree uses the technique of overlapping regions and demonstrates that it is possible to organize spatial objects such that the overlap of the regions in the directory is extremely small. From our point of view, further research in SAMs will improve the overall-performance of query processing only marginally. Therefore, additional concepts have to be integrated into a geographic database system for improving its query performance.

3 Geometric filtering by approximations

As described above, spatial objects are organized and accessed by SAMs using geometric keys which maintain the most important features of the objects (position and extension). The smallest aligned rectangle enclosing an object, the *minimum bounding box (BB)*, is the most popular geometric key. Using the BB the complexity of an object is reduced to four parameters. Inspecting the BBs of the candidates against the spatial query condition, we obtain three classes of objects: *hits* fulfilling the query, *false hits* not fulfilling the query and remaining *candidates* possibly fulfilling the query (see figure 5).

Using BBs provides a fast but inaccurate filter for the response set. The larger the area of the BB differs from the area of the original object, the more inaccurate is the geometric filtering, i.e. the candidate set includes a lot of false hits and the number of identified hits is very small.

In order to get expressive and realistic results on the quality of the approximation when BBs are used, we investigated simple polygons with holes of various real maps. To be as general as possible, we used maps from different sources with different resolutions. The data files contain natural objects such as islands and lakes as well as administrative areas such as counties. Figure 3 depicts the analysed maps.

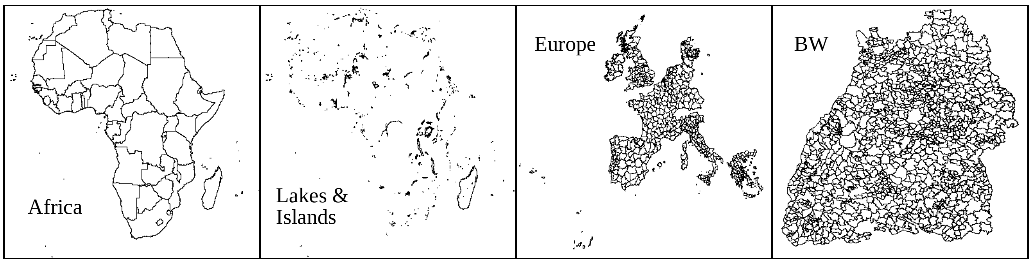

Figure 3: The analysed maps

In the literature, several alternatives are proposed to measure the quality of approximations. In our application we are interested in measuring the accuracy of the geometric filter. The accuracy of the filter step is maximized by minimizing the deviation of the approximation from the original object. This deviation is measured by the false area of the approximation normalized to the area of the approximated object. Table 1 shows impressively that real cartography objects are only roughly approximated by BBs.

	min	max	μ	σ/μ
Europe	0.25	21.14	0.93	0.21
BW	0.20	5.01	0.93	0.18
Lakes	0.21	22.11	0.97	0.32
Africa	0.34	5.64	0.89	0.23

Table 1: False area of the BB-approximation

- μ is the average of false area
- σ/μ is the standard variation normalized to μ
- *min* and *max* denotes the minimum and the maximum deviation of the area of the BB from the area of the corresponding polygon occurring in the map, respectively.

This investigation was the starting point to look for other approximations which have a better approximation quality than the BB. Approximations of objects which are used for geometric filtering should be simple to provide a fast filter (*simplicity criterion*) and they should have a high approximation quality (*quality criterion*) to reduce the number of false hits and to identify as many final answers as possible. In our point of view only convex, straight line bordered approximations which can be represented by a small number of parameters are suitable candidates. Therefore, we tested the *rotated bounding box (RBB)*, the *convex hull (CH)*, the minimal enclosing convex *4-corner (4-C)* and the *5-corner (5-C)*.

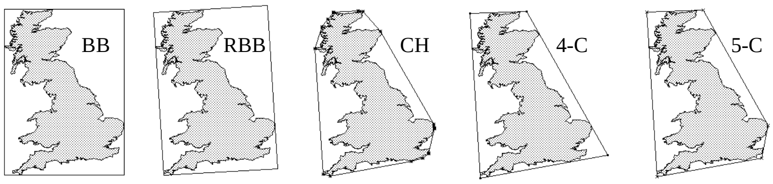

Figure 4: Different tested approximations

In [BKS 93a] we presented a detailed empirical investigation. The measured approximation qualities were almost independent of the various tested maps. Let us emphasize that this result is very interesting because we used maps from different sources with different resolutions (see figure 3). A clear order of rank turned out. Naturally, the convex hull has the best approximation quality (24 % false area), followed by the 5-corner (33 %), the 4-corner (44 %) and the rotated box (62 %). Compared to the minimum bounding box (93 %), clear gains of approximation quality were obtained. Obviously, the more parameters are used for the representation of an approximation, the better is the approximation quality. The results show that the approximation quality of the 5-corner is nearly the same as that of the convex hull. However, the storage requirement of convex hulls varies extremely and is on the average much higher than the storage requirement of the other approximations. The RBB reduces the false area by 31 percentage points compared to the BB, although only one additional parameter is used. The 5-corner needs 6 additional parameters compared to the BB paying off in 60 percentage gain over the average false area of the BB. Summarizing, we can state that the 5-corner yields the best trade-off between additional storage requirement and improvement of approximation quality.

In order to integrate the 5-corner in geometric filtering of spatial query processing two different ways can be followed. First, the common BB remains the geometric key and the 5-corner is additionally stored in the data pages of the SAM. The advantage of this approach is obvious. The approximation quality is improved and all the known SAMs based on BBs can be used. Second, we use the 5-corner as the geometric key and the BB is not stored anymore. Doing this we save storage, but SAMs using the transformation technique are not suitable.

In [BKS 93a] we have shown that the 5-corner approximation can efficiently be organized in the R*-tree, a spatial access method originally designed for bounding boxes. The simplicity and robustness of the spatial access method is preserved, because in the directory simple bounding boxes are organized and only in the data blocks more complex approximations are stored. Obviously, in the filter step approximations other than the minimum bounding box need more block accesses when traversing the SAM because of their higher storage requirement and their higher extension in x- and y-direction. However, the reduced number of false hits due to higher approximation quality results in a substantial gain in the geometry processor by avoiding time intensive computational geometry algorithms. This gain clearly exceeds the slightly higher access costs.

4 Transfer of the geometry into main memory

Most SAMs proposed up to now accommodate object approximations or a small number spatial objects in their data pages. However, the pages storing the geometry of spatially adjacent objects are distributed arbitrarily over the secondary storage. Typical window queries require the contents of many pages to be retrieved from the database. This is much more expensive for arbitrarily distributed pages than for physically contiguous pages because the search time of a page on disk is much higher than the transfer time and because physically contiguous pages can be transferred by one set-oriented disk access into main memory.

Basically, we distinguish three models for storing large sets of spatial objects:

1. *Storing the exact object geometry outside of the spatial access method*, e.g. in a sequential file. The main advantage of this scheme is the large number of approximations stored together in one data page. A fundamental drawback is the fact that the clustering just refers to the object approximations and not to the objects themselves. Consequently, when processing window queries, each access to an exact object geometry needs additional expensive search time.
2. *Storing the exact object geometry inside the data pages of the spatial access method*. Thus, spatial neighborhood is physically preserved at the level of exact object geometry. Objects within one data page are transferred into main memory just using one disk access. An essential drawback of this approach is the low number of objects fitting into one page. As a consequence, adjacent objects are often stored in different pages and clustering is restricted by the page size which is about 4 kbyte.
3. Our new approach, *the scene organization*, associates the geometry of spatially adjacent objects to sets of physically contiguous pages. These sets are derived from a slightly modified R*-tree and are called *scenes*. A scene is described by a minimum bounding box. The geometric keys (and additional approximations) are organized in R*-trees as before. The scene organization allows dynamic changes of the database, supports large range queries as well as small queries, assures that a maximum scene size is not exceeded, and strives for a stable average scene size and a high storage utilization. A detailed algorithmic description of the scene organization is given in [BKS 93b]. A window query proceeds as follows: All scenes intersecting the query window are determined. If the degree of overlap between the scene and the query window is smaller than a heuristically determined query threshold, the window query is processed using the mechanism depicted in figure 5. Otherwise, the scene is completely transferred into main memory using the fast set-I/O. Unfortunately, a scene may contain a number of false hits unnecessarily transferred into main memory. However, a relatively small number of false hits does not affect performance considerably, since the time needed for searching a page exceeds drastically the time for transferring a page. After transferring the scene into main memory, the query works as usual without the transfer step after filtering. The sequence of steps is depicted in figure 5.

In order to evaluate the scene organization, we carried out a detailed empirical performance comparison of the three models using real test data. The database consists of 119,151 objects with an average size of 956 bytes. The page capacity was 4 kbyte and we tested point queries and 5 series of window queries with an area of the query window between 0.0625% and 16% of the data space. We weighted the cost for searching a page on disk by a factor of 10 to the relative cost of a page transfer.

As expected with increasing scene size the search cost decreases and the transfer cost increases. The optimal scene size which leads to minimum access cost depends on the size of the queries. The larger the queries, the larger is the optimal scene size. However, this dependency is not very strong. In our tests there is a factor of 256 in the size of the query windows, but only a factor of 8 in the resulting optimal scene sizes. Additionally, the graphs for the cost functions are rather flat in the proximity of their minimum. Thus, a maximum scene size of 192 kbyte is almost optimal for all our test series.

We also investigated the performance improvement of the scene organization in comparison to the other two approaches. To compare the clustering of our scene organization with the other models, the access cost for point queries is a suitable measure. In the scene organization the cost of point queries is very close to the cost of the two other models. That means that the modifications of the R*-tree have almost no effect on its space partitioning. In table 2, a comparison of the access cost of the three models is presented by describing the speed up factor for query processing using the model 2 and 3 in comparison to model 1.

size of query windows (in % of data space):	0.0625%	0.25%	1%	4%	16%
model 2: geometry inside of the data pages	2.3	2.5	2.6	2.8	2.9
model 3: scene organization	6.5	11.7	18.6	25.4	30.6

Table 2: Speed up factors for window queries in comparison to model 1 (geometry outside of the data pages)

Storing the exact object geometry inside the data pages (model 2) speeds up query processing by a factor of 2.3 to 2.9 in comparison to model 1 which stores the exact geometry outside of the data pages. The scene organization is the clear winner of the performance comparison. Even the processing of small window queries is performed considerably faster by this organization model. For small queries, we have a speed up factor of about 6.5 (in comparison to model 1) which is increasing to the value of 30.6 for large queries.

5 Exact investigation of the geometry

Geometric filtering is based on object approximation and therefore determines a set of *candidate objects* that may fulfil the query condition. The geometry processor tests whether a candidate object actually fulfils the query condition or not. This step is very time consuming and dominates spatial indexing and geometric filtering in many applications (see [BZ 91]). Algorithms from the area of computational geometry are proposed to overcome this time bottleneck. Different specialized data structures and techniques, such as plane sweep or divide-and-conquer, are used to design efficient algorithms for the different spatial queries and operations.

Due to the complexity of the objects on the one hand and the selectivity of spatial queries on the other hand, it is useful to *decompose the objects* into simpler components because the decomposition substitutes complex computational geometry algorithms by multiple calls of simple and fast algorithms. The success of such processing depends on the ability to narrow down quickly the set of components that are affected by the spatial queries and operations. The performance comparisons in [KHS 91] demonstrate that decomposition techniques outperform the traditional undecomposed representation up to one order of magnitude for high selectivity queries.

In [SK 91] we proposed a new representation of a polygonal object, called the *TR*-tree* that efficiently supports various types of spatial queries and operations. The TR*-tree is a dynamic data structure that is persistently stored on secondary storage and is completely loaded to main memory for spatial query processing.

In our approach, we efficiently support the average case of various types of queries and operations in real applications where only one single representation of the objects is used. We decompose in a preprocessing step the polygonal objects into a minimum set of disjoint trapezoids using the plane-sweep algorithm proposed by Asano & Asano [AA 83]. However, we cannot define a complete spatial order on the set of trapezoids that are generated by this decomposition process. Thus binary search on these trapezoids is not possible. Therefore, we propose to use the R*-tree for the spatial search. Due to its tree structure, the R*-tree permits logarithmic searching in the average case but due to the overlap within its directory the search is not restricted to one path and thus logarithmic search time cannot be guaranteed in the worst case. The R*-tree was designed as a spatial access method for secondary storage. In order to speed up the queries and operations mentioned above, we developed the TR*-tree, a variant of the R*-tree, designed to minimize the main memory operations and to store the trapezoids of the decomposed objects as complete objects without approximating them by minimum bounding boxes.

The performance of the TR*-tree cannot be analytically proven because the TR*-tree is a data structure that uses heuristic optimization strategies. Therefore, we documented the performance of the TR*-tree in an experimental analysis investigating synthetically generated data as well as real cartography data in a systematic framework. For example, a point query on objects with 2,500 vertices is answered performing 20 point-in-BB-tests and 3 point-in-trapezoid-tests. For a detailed performance analysis the interested reader is referred to [SK 91].

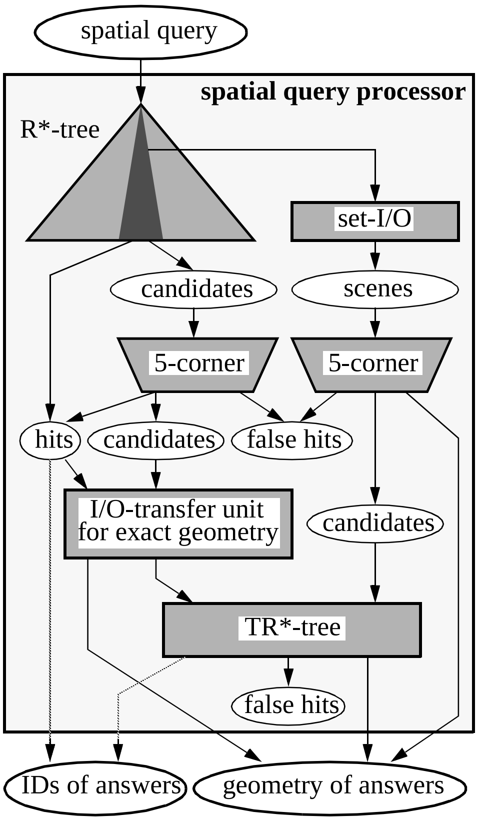

Figure 5: Spatial query processor

6 Summary and future work

In the previous sections, we presented a number of techniques for reducing query time. Figure 5 depicts our spatial query processor consisting of several building blocks using these techniques. The R*-tree is used as a simple, robust, and efficient spatial index. Our performance comparison of approximations yields that the 5-corner is the best trade-off between additional storage requirement and improvement of approximation quality. Therefore, it is employed as a geometric filter. In order to reduce the I/O-cost for window queries retrieving the object geometry, we integrated the scene organization. If the degree of overlap between a scene and a query window is smaller than the query threshold, the complete scene is transferred into main memory using a fast set-I/O device. Otherwise, the R*-tree is completely traversed and the object geometry of candidates (and if necessary for hits) is transferred after filtering. The exact geometry processor is realized by TR*-trees organizing decomposed objects.

In our future work, we want to apply the presented techniques for efficient *spatial join* processing. A first approach is presented in [BKS 93c]. Within this framework, it is necessary to investigate *progressive approximations* describing the kernel of an object and to compare the performance of the TR*-tree to computational geometry techniques.The application of the presented techniques to *3D-objects* is another interesting area of research activities in the future. For example, *bio-computing* is an important field of application for 3D-spatial objects. The first step in this direction is the development and implementation of 3D-approximation and decomposition techniques.

References

[AA 83] Asano Ta., Asano Te.: *'Minimum Partition of Polygonal Regions into Trapezoids'*, Proc. 24th IEEE Annual Symp. on Foundations of Computer Science, 1983, pp. 233-241.

[BHKS 93] Brinkhoff T., Horn H., Kriegel H.-P., Schneider R.: *'A Storage and Access Architecture for Efficient Query Processing in Spatial Database Systems'*, Proc. 3rd Symp. on Large Spatial Databases, Singapore, 1993.

[BKS 93a] Brinkhoff T., Kriegel H.-P., Schneider R.: *'Comparison of Approximations of Complex Objects used for Approximation-based Query Processing in Spatial Database Systems'*, Proc. 9th Int. Conf. on Data Engineering, Vienna, Austria, 1993, pp. 40-49.

[BKS 93b] Brinkhoff T., Kriegel H.-P., Schneider R.: *'Scene Organization: A Technique for Global Clustering in Spatial Database Systems'*, 1993, submitted for publication.

[BKS 93c] Brinkhoff T., Kriegel H.-P., Seeger B.: *'Efficient Processing of Spatial Joins Using R-trees'*, Proc. ACM/ SIGMOD Int. Conf. on Management of Data, Washington D.C., 1993.

[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, N.J., 1990, pp. 322-331.

[BZ 91] Benson D., Zick G.: *'Symbolic and Spatial Database for Structural Biology'*, Proc. OOPSLA, 1991, pp. 329-339.

[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.

[HS 92] Hoel E.G., Samet H.: *'A Qualitative Comparison Study of Data Structures for Large Line Segment Databases'*, Proc. ACM SIGMOD Int. Conf. on Management of Data, San Diego, CA., 1992, pp 205-214.

[KHS 91] Kriegel H.-P., Horn H., Schiwietz M.: *'The Performance of Object Decomposition Techniques for Spatial Query Processing'*, Proc. 2nd Symp. on Large Spatial Databases, Zürich, Switzerland, 1991, in: Lecture Notes in Computer Science, Vol. 525, Springer, 1991, pp. 257-276.

[KSSS 89] Kriegel H.-P., Schiwietz M., Schneider R., Seeger B.: *'Performance Comparison of Point and Spatial Access Methods'*, Proc. 1st Symp. on the Design and Implementation of Large Spatial Databases, Santa Barbara, CA., 1989, in: Lecture Notes in Computer Science, Vol. 409, Springer, 1990, pp. 89-114.

[Sam 90] Samet H.: *'The Design and Analysis of Spatial Data Structures'*, Addison-Wesley, 1990.

[SK 88] Seeger B., Kriegel H.-P.: *'Techniques for Design and Implementation of Efficient Spatial Access Methods'*, Proc. 14th Int. Conf. on Very Large Databases, Los Angeles, CA., 1988, pp. 360-371.

[SK 91] Schneider R., Kriegel H.-P.: *'The TR*-tree: A New Representation of Polygonal Objects Supporting Spatial Queries and Operations'*, Proc. 7th Workshop on Computational Geometry, Bern, Switzerland, 1991, in: Lecture Notes in Computer Science, Vol. 553, Springer, 1991, pp. 249-264.