A Comparison of Spatial Query Processing Techniques for Native and Parameter Spaces

Jack Orenstein
Object Design, Inc
One New England Executive Park
Burlington, MA 01803

jack@odi com

Abstract
Spatial queries can be evaluated in native space or in a parameter space In the latter case, data objects are transformed into points and query objects are transformed into search regions The requirement for different data and query representations may prevent the use of parameter-space searching in some applications Native-space and parameter-space searching are compared in the context of a z order-based spatial access method. Experimental results show that when there is a single query object, searching in parameter space can be faster than searching in native space, if the data and query objects are large enough, and if sufficient redundancy is used for the query representation The result is, however, less accurate than the native space result When there are multiple query objects, native-space searching is better initially, but as the number of query objects increases, parameter space searching with low redundancy is superior Native-space searching is much more accurate for multiple-object queries

1. Introduction

Most access methods designed for spatial queries against point data operate by partitioning the space The structures differ from one another in the techniques used for partitioning the space, and the mechanisms for locating the partitions Non-point data is more complicated to organize A point can be classified in exactly one partition of the space, but this is not true for non-point objects There are several ways of dealing with this problem, and each has given rise to at a number of recent access methods

The simplest way of dealing with non-point objects is to transform the search problem from its "native" space into an equivalent search problem in a parameter space It is then possible to use any access method for point data to implement the search Many such structures have been proposed for use in a database environment [LOME, LOME89, MERR78, NIEV84, OREN84, TAMM82] One example of this technique is to transform k-dimensional boxes into points in a 2k-dimensional space (Boxes are assumed to be oriented so that each face is perpendicular to one axis of the space) Transformations of this kind require fairly simple spatial objects, so in many cases it is necessary to place the data objects in containers which are then parameterized [HINR85]

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery To copy otherwise or to republish requires a fee and/or specific permission
© 1990 ACM 089791 365 5/90/0005/0343 $1 50

Other techniques for handling non-point data are based on partitioning the native space Access methods that use this idea have to address the problem of how to deal with data objects that span partitions The R-tree [GUTT84] assigns such an object to one of the partitions Interior nodes of the structure represent regions of space containing a union of partitions These regions may overlap, and if a query covers the area common to two regions, then both corresponding subtrees have to be searched

The R+-tree [SELL87] deals with objects spanning partitions differently by storing a copy of the object in each overlapping partition Regions in interior nodes no longer overlap one another, but the tree is larger than the corresponding R-tree, because some objects are stored redundantly The same technique can be applied to the grid file [NIEV84] and EXCELL [TAMM82] to yield structures for non-point data, (see [HORN87] for the grid cell variant, and [MANT83] for the EXCELL variant)

Access methods based on z order [OREN86, OREN88] also use redundancy to deal with non-point objects Z order-based access methods, and the use of redundancy in such structures will be discussed in section 3

None of the non-point access methods mentioned can provide completely correct answers to spatial queries In one way or another, each structure introduces an approximation of the data objects As long as these approximations are conservative, a search acts as a filter, returning all objects satisfying the query, and possibly some others For this reason, spatial queries involve the following two-step strategy

- **Filter step** The spatial index is used to rapidly eliminate objects that could not possibly satisfy the query The result of this step is a set of *candidates* which includes all the results and possibly some false hits

- **Refinement step** Each candidate is examined False hits are detected and eliminated

The filter step involves retrieval from the database, so the time measure of interest is the number of disk accesses The refinement step is typically CPU-intensive, and its running time is dependent on the accuracy of the filter step since each false hit has to be examined For some access methods, accuracy of the filter step can be controlled to some extent, so it is possible to adjust the time spent in each step according to application-specific concerns For example, if the data and query objects are points and boxes respectively, then each candidate can be verified very rapidly But if the data and query objects are more complicated, e g arbitrary polyhedra, then verification is more expensive and it may make sense to spend more time in the filter step to reduce the number of false hits Section 3 3 shows how, for z order-based structures, the accuracy and time of the filter step are related, and how they can be controlled by adjusting the amount of redundancy in object representations

The aim of this paper is to compare native-space and parameter-space techniques for dealing with non-point data The organization of this paper is as follows Section 2 describes the overlap query, and the evaluation of this query in native and parameter spaces Section 3 provides an overview of z order-based access methods, a technique that can be used to implement both native-space and parameter-space searching The experiments to be described were carried out using this technique Section 4 describes the design of the experiments, and section 5 presents results and analysis Section 6 contains a summary and conclusions

2. Data and query object representations

2.1. The overlap query

The problem that drives much research on spatial searching is to find all data objects overlapping, containing, or contained by a given query object This, and other important problems are special cases of the ***overlap query***. Given two sets of spatial objects, D and Q, find pairs (d, q), such that d is in D, q is in Q, and d and q overlap[1] A range query is an overlap query in which D contains points and Q contains a single box In point classification, D contains objects that partition a space, (e g the regions of a map), Q contains a single point, and the goal is to discover which partition contains the point

There are also situations in which both $|D| \geq 1$ and $|Q| \geq 1$ For example, in VLSI applications, one form of validity checking involves spatial constraints that require some minimum separation between objects of certain types In this case, one input set contains objects of one type, and the other contains objects of the other type, "expanded" or "buffered" by the separation specified in the constraint The overlap query reports violations of the constraint

Queries involving proximity can be transformed into overlap queries The nearest-neighbor query finds the data object nearest a given point, q Given an upper bound, r, on the distance between q and its nearest neighbor, a search region can be defined by drawing a circle of radius r around q The nearest-neighbor must fall within the circle A good upper bound is easy to find with any spatial structure

In the following sections, a *one-many* query will denote an overlap query in which $|D| = 1$, and a *many-many* query will denote an overlap query in which $|D| \geq 1$

2.2. Overlap queries in parameter space

A one-many query can be evaluated in parameter space by transforming data objects into points, and query objects into search regions The query is evaluated by locating the points inside the search region Figure 1 shows a one-many query in 1-d space There are four data objects, intervals [1, 2], [3, 3], [5, 5], and [7, 7] Each data object can be transformed into a point in 2-d space by treating the left boundary as the x coordinate, and the right boundary as the y coordinate

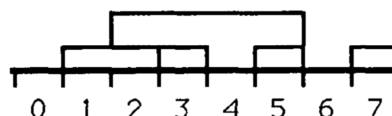
There is a single query object, [2, 5], which overlaps all data objects except for [7, 7] The overlap query locates all intervals [lo, hi] overlapping the query interval [qlo, qhi], i e $lo \leq qhi$ and $hi \geq qlo$ Therefore, the query is transformed into a region in the 2-d parameter space, $0 \leq x \leq 5, 2 \leq y \leq 7$

[1] D and Q are meant to represent "data" and "query" respectively, although, as discussed in section 2 3, the distinction is often artificial

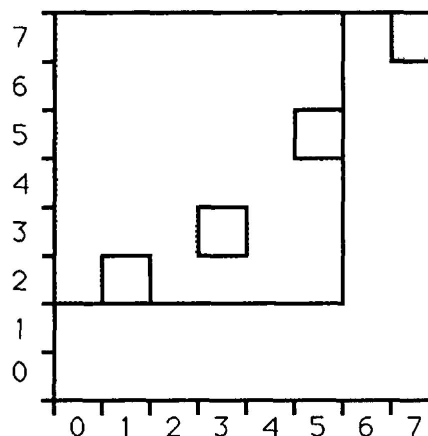
Figure 1

One-many query

Native space



Parameter space



Many-many queries can be transformed to parameter space also One of the input sets, D, is identified as data, and each object is transformed into a point as in one-many queries The other input, Q, is identified as query, and each object is transformed into a region These representations will be referred to as D and Q formats respectively In figure 2, D is the same as in figure 1 Q contains two objects, [0, 3], [3, 6] The response to this query contains four pairs of overlapping objects

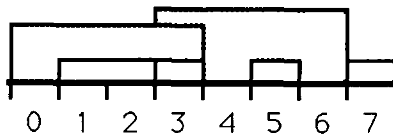
Query object	Data object
[0, 3]	[1, 2]
[0, 3]	[3, 3]
[3, 6]	[3, 3]
[3, 6]	[5, 5]

As in figure 1, a query object is represented by a region in 2-d space The [3, 3] interval, which is covered by both query objects, belongs to both query objects This is emphasized by dark shading in figure 2

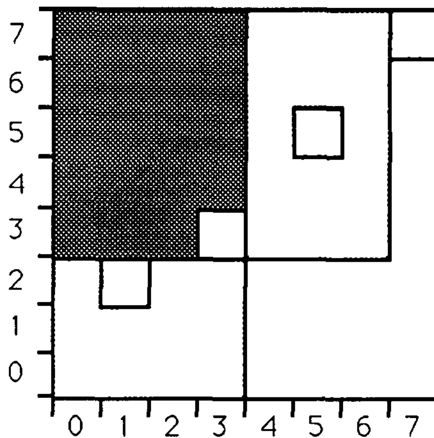
Figure 2

Many-many query

Native space



Parameter space



The terms "data" and "query" are used here to identify different transformations From a user's point of view, the two inputs to an overlap query are symmetric

2.3. One or two representations?

In practice, it is often the case that spatial objects play dual roles, sometimes acting as data, sometimes as query In the query "find all cities within 20 miles of the road I95", the object representing the geographic extent of Boston is retrieved as a data object In the query "find all aircraft within 5 miles of Boston", the same object is used to define a query region In other words, the classification of a spatial object as data or query is not inherent in a spatial object, but instead applies to the role of an object or set of objects within a query

This presents a problem for parameter-space searching where data and query objects have different representations For one-many queries, the problem is not serious Objects can be stored in the D format, and the Q format can be derived as needed However, for many-many queries, the use of different data and query representations is more problematical Either two representations have to be stored and maintained, or it is necessary to compute the second representation before the query can be evaluated

For example, suppose that an application has three sets of spatial objects, A, B, and C, and suppose that overlap queries involving all three pairs, AB, BC, and AC are required In order to run an overlap query, one input must be in D format, and the other must be in Q format Suppose that A and C are stored in D format and that B is stored in Q format Then AB and BC can be computed immediately But the AC overlap query cannot be evaluated until either A or C is made available in Q format This significantly increases the cost of the query

3. Overview of z order-based access methods

In [OREN84] a solution to the range query problem is given The technique used is to transform the problem of finding all the points in a k-d box into an equivalent search problem in 1-d Each data point is transformed into a 1-d interval of size 1 and the query box is transformed into a set of intervals of varying size A data point is contained by the query box iff the corresponding interval falls in one of the query's intervals This approach yields a *family* of data structures for evaluating range queries A member of this family is derived by providing an access method that supports random and sequential access, e g a sorted array, an AVL tree, or a b-tree The search algorithm is expressed in terms of random and sequential accesses to the underlying access method In spite of the generality of this approach, performance is comparable to that of more specialized structures Furthermore, this approach permits all the theory, techniques, and even software, that has been developed for ordinary (one-dimensional) searching problems, to be applied to spatial searching

This approach can be generalized to deal with arbitrary spatial objects and overlap queries [OREN86] Each spatial object in each input set is transformed to a set of 1-d intervals An algorithm called *spatial join* implements the filter step Each resulting candidate is a pair of objects, one from each input, that are likely to overlap Spatial join and filtering algorithms for other spatial problems appear in [OREN88] These algorithms comprise the *geometry filter* (GF) Sections 3 1 and 3 2 provide a brief overview of the spatial join algorithm

3.1. Decomposition: generating the geometry filter's representation

As discussed above, the geometry filter works by transforming a k-d spatial object into a set of 1-d intervals There are many ways to do this Many of the modularity and performance benefits of the geometry filter derive from the particular way in which the geometry filter does this transformation

The "conceptual" representation used by the geometry filter is a grid of fixed resolution The representation for an object is obtained by noting which cells are completely or partially occupied by the object This representation is a conservative approximation Partially occupied cells are included so that the filtering property is retained If such cells were omitted, then the approximation would not be conservative and some positive results would be lost in the output of the filter step

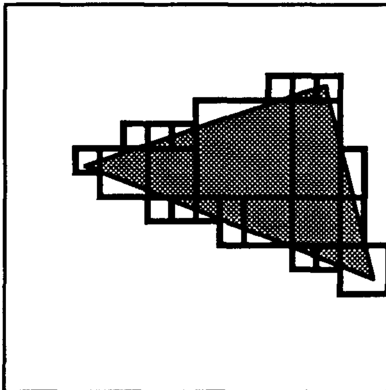
The grid representation can easily be transformed to 1-d, e g , by *listing the occupied cells in row-major order* However, the number of occupied cells depends on the volume of the object As a result, the space and time requirements for algorithms based on this representation will be very high Instead, an encoding of the grid is used The space is recursively partitioned until the resolution of the grid is reached Regions that are entirely contained in the object do not have to be split further The space requirement for this encoding is proportional to the surface area of the object, not the volume, so the space and time requirements are much better

By constraining the partitioning process, a highly compact representation of the spatial object can be obtained A region created by partitioning under these constraints is called an

element Typically, each element can be described by one 32-bit word Each partition is represented by one bit, and the relationship of the element to the partition is described by the value of the bit. (In 2-d space, a 0 means to the left of or below A 1 bit means to the right of or above) The bit-sequence corresponding to an element is called a *z value* Given a z value, the size, shape and position of an element can be reconstructed

Figure 3 shows the encoding of a spatial object achieved by elements This is a more compact representation than the explicit listing of each grid cell, since the storage requirement for each element is the same, regardless of its size

Figure 3



The *decompose* algorithm is responsible for generating the elements corresponding to a spatial object In spite of the constraints on the partitioning process, there are actually a variety of decomposition strategies, as described in section 3 3

The geometry filter's representation for a set of spatial objects is obtained by decomposing each object, associating each z value with the object, and then merging all the (z value, object) pairs into a single 1-d access method, keyed by z value Redundancy occurs when the same object is associated with multiple z values This structure will be referred to as the GF representation or GF sequence

The geometry filter algorithms operate by generating random and sequential accesses against GF sequences

3.2. Spatial join: generating candidates

The spatial join algorithm performs a merge of two GF sequences, searching for situations where an element in one sequence, as represented by its z value, contains an element from the other input This can be determined by checking whether the binary representation of one z value is a prefix of the other When such a pair is found, a candidate, comprising the objects associated with the elements, is generated If the sizes of the input sequences are |D| and |Q|, then the time for the merge is O(|D| + |Q|) disk accesses However, it is often possible to do much better For example, consider a range query The n data points yield a GF sequence of size n The other input set will contain some small number of elements for the query box The merge can be optimized by taking advantage of the fact that the elements of the query box usually occur in clusters For example, all the data points whose z values are less than that of the first query element can be skipped Similarly, the data points between elements of the query box and those data points following the last element of the query can be skipped In general, data elements in the space between query elements can be safely ignored The ability to "skip over" elements that are clearly not of interest is the source of the random access requirement.

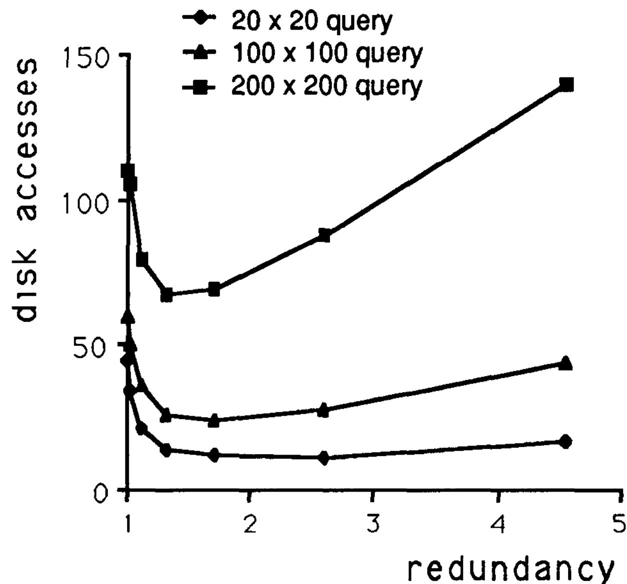
With this optimization, and assuming a certain distribution of data (which is more regular than uniform), it can be shown that the expected performance for range queries is O(fN) where f is the fraction of the space covered by the query, and N is the number of data objects [OREN86]

3.3. Redundancy

The decomposition of a point object yields one single-cell element For non-point objects, decomposition yields multiple elements The *redundancy* of a GF representation is R = (number of elements) / (number of objects), $R \geq 1$ In searching a collection of non-point objects with a box query object, the performance is $O(fNR_d)$ [OREN89a] where R_d is the redundancy of the collection of data objects (As discussed below, this does not hold for low values of R_d) Redundancy of the query representation will be denoted by R_q

In practice, the decomposition algorithm described above generates too many elements R_d increases with the surface area of the objects being decomposed, and the resolution of the space The problem is that the decompose algorithm introduces redundancy in an uncontrolled way In [OREN89a, OREN89b] it was shown that redundancy could be controlled by limiting the number of elements generated by the decomposition, or by limiting the accuracy of the decomposition With either approach, the time required by the filter step, as measured by disk accesses, first *decreases* as R_d increases, and then increases In figure 4, this is shown for 20 x 20, 100 x 100, and 200 x 200 queries in a 1000 x 1000 space containing 5000 boxes of size 30 x 30

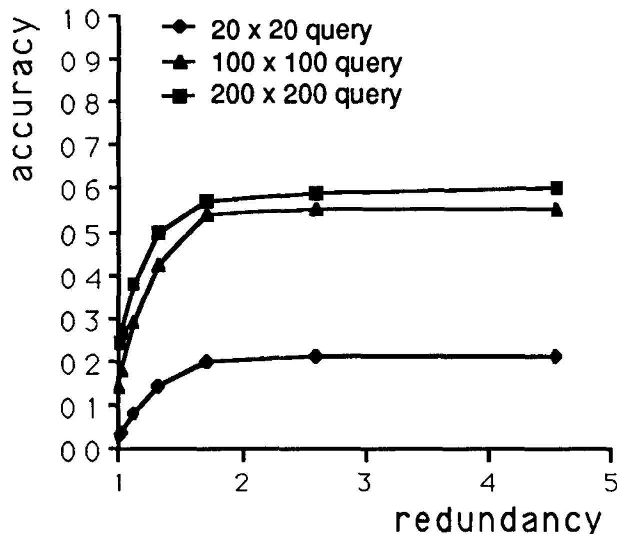
Figure 4



This occurs because, at low redundancy, the decompositions are very poor approximations of the objects, and there are many false hits For higher values of R_d, accuracy of decomposition is better, and the linear increase in f, N, and R_d can be observed

Accuracy of the filter step is defined as the number of items satisfying the query divided by the number of items located by the filter step Accuracy increases monotonically, with the biggest gains appearing at low values of R_d, (see figure 5)

Figure 5



This is fortunate since, for the data sets studied, accuracy is near its maximal value when disk accesses are minimized

The studies cited above examined native space searching only, and focussed on the effect of redundancy applied to data objects In the experiments to be described, redundancy of the query objects is significant also

4. Design of the experiments

The experiments to be described compare the performance of native-space and parameter-space implementations of the overlap query A z order-based access method, the zkd b+-tree, is used in both implementations In native space, redundancy is applied to objects in both D and Q In parameter space, objects in D format are points, so no redundancy is required, but objects in Q format are regions, so redundancy is used As discussed in section 1, we are interested in the disk accesses required by the filter step, and accuracy of the filter step

Time is measured by counting disk accesses If a page is requested while present in a buffer, a disk access is not counted Accuracy is defined as the number of items in the result divided by the number of candidates These definitions are applicable to both one-many and many-many queries

The details of the design are as follows

- Experiments were carried out on 1-d and 2-d native spaces[2]

- Native spaces had 12 bits of resolution, 2^{12} in 1-d, $2^6 \times 2^6$ in 2-d Parameter spaces had 24 bits of resolution, $2^{12} \times 2^{12}$ in 2-d, $2^6 \times 2^6 \times 2^6 \times 2^6$ in 4-d

- The coordinate systems in which data and query objects are described had a resolution of $2^{12} = 4096$ per dimension, e g $2^{12} \times 2^{12}$ in 2-d spaces

[2] Unless stated otherwise, the discussion refers to data and query objects in the native space

- Six data sets were used, 1-d intervals of size 10, 50, and 100 (in a space of size 2^{12}), and 2-d squares of size 50 x 50, 200 x 200, and 500 x 500 (in a space of size $2^{12} \times 2^{12}$) Each data set contains 5000 uniformly distributed objects

- Input sets were stored in zkd b+-trees, with leaf capacity of 20 2-d boxes or 30 1-d intervals Each b+-tree had a private pool of 15 page buffers, managed using LRU replacement

- In native space, data objects were decomposed using the ERROR-BOUND(8) strategy, which was reported to be optimal for similar data sets in [OREN89a]

Two coordinate systems are used in this design The data and query objects are described in a space with resolution of 2^{12} = 4096 in each dimension The geometry filter uses a different space, as described above, and is responsible for transformations between the two spaces The performance of the geometry filter is not significantly affected by the resolution it uses (unless a very coarse resolution is used, in which case accuracy will suffer) The reason for this is that the limits on redundancy, as described in section 3 3, will usually terminate the decomposition process well before the resolution of the space is reached (See [OREN89a] for further discussion of this point)

There are a number of ways to transform intervals and boxes into higher-dimension spaces One technique is to use end-points For example, the 1-d interval [lo, hi] would be represented by the 2-d point (lo, hi) Other parameterizations are obtained by using some fixed position, e g , an end-point or the center, and the extent In these experiments, end-points were used Other transformations would most likely affect quantitative results, but not the trends to be described

The software used in the experiments was developed in C and run on a Commodore Amiga 2000

5. Results and analysis

This section presents the results of the experiments, and proposes explanations for the observations First, the results for one-many queries are described Next, results on many-many queries are presented To the best of my knowledge, there are no other published performance results for many-many queries Finally, results concerning storage requirements, for both one-many and many-many queries are presented

5.1. One-many queries

Figures 6-8 show how disk accesses to D (the collection of data objects) and accuracy for one-many queries in 1-d space vary with query size The graphs show results for data objects of size 10, 50, and 100 (in a space of size 4096 as described in section 4) All graphs show native space results for $R_q \le 8$[3] The results for $R_q \le 12$ were virtually identical, indicating that query redundancy

[3] R_q is given as an inequality because the decomposition algorithm guarantees only that redundancy will not exceed the stated bound The maximum number of elements is generated, and then elements that can be merged without loss of accuracy are merged This results in less redundancy than the maximum permitted

greater than 8 is not needed in native space All graphs also show parameter space results for $R_q \le 16$, 32, 64, 128, and 256 For the disk access results, the curves for $R_q \le 128$ and $R_q \le 256$ are very close, and further increases seem unlikely to produce significantly better results The more-or-less linear increase in disk accesses with both query size is consistent with [FALO89a], which examined a different transformation to parameter space

Figure 6a

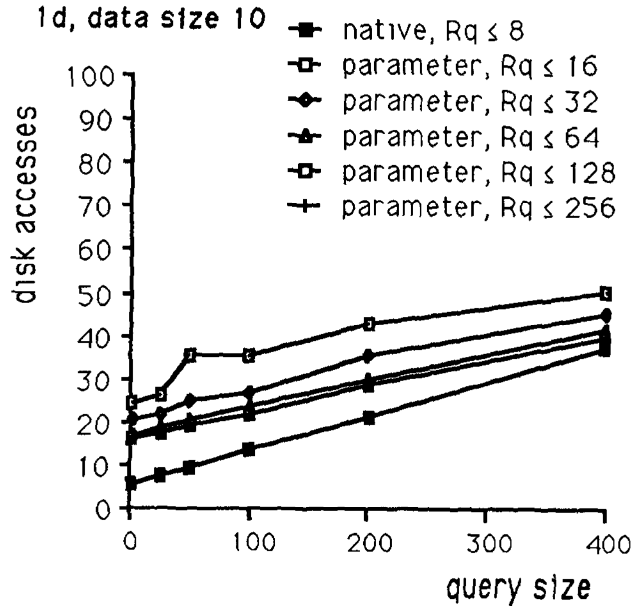
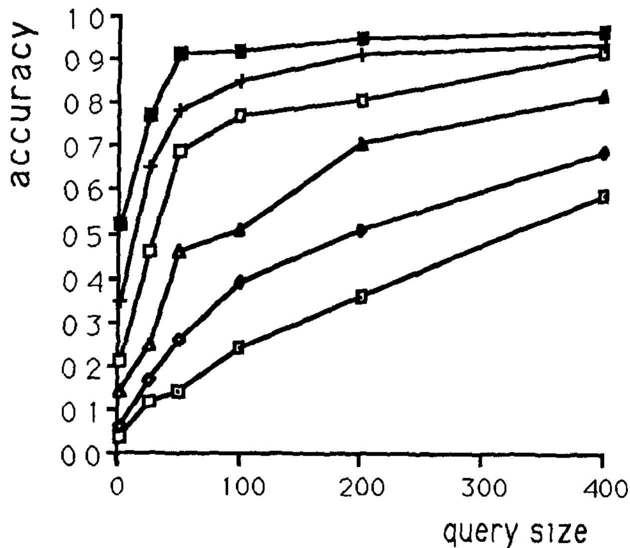


Figure 6b



For data object sizes 50 and 100, the disk access curves for parameter space cross the native space curve The query size at which crossing occurs depends on the amount of query redundancy used in the parameter space For example, in figure 7a, at $R_q \le 16$, the curves cross at query size 270 (approximately), while the crossing is at 120 for $R_q \le 256$ In figure 6a the curves do not cross, but a small extrapolation suggests that they will cross just past query size 400 (the largest query examined)

Figure 7a

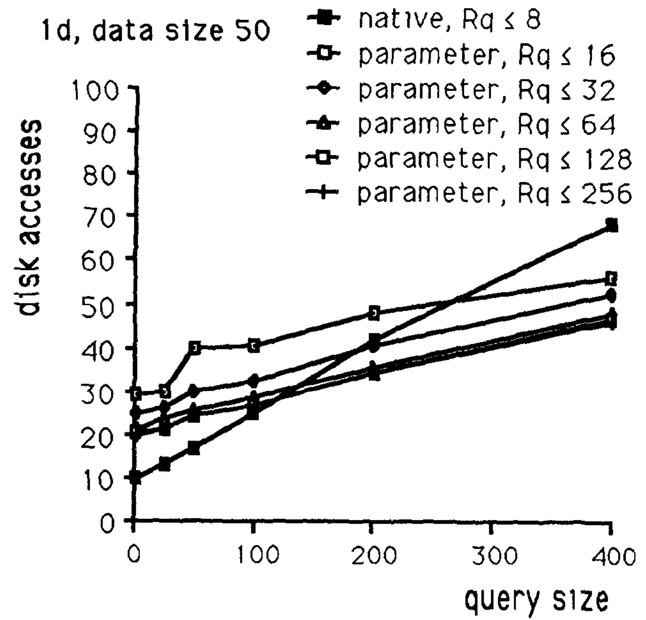
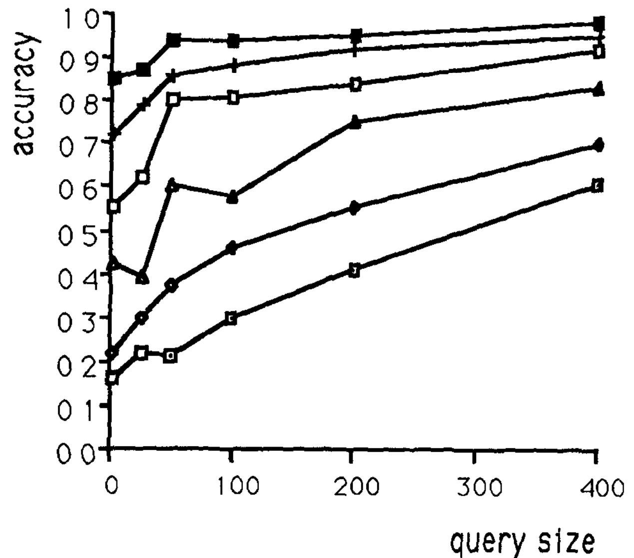


Figure 7b



The general pattern is that small queries are cheaper in native space, but that the rate of increase is faster than for parameter space queries As mentioned above, the number of disk accesses expected is $O(fNR_d)$, and in parameter space $R_d = 1$ It is therefore to be expected that, in the native space, where $R_d \ge 1$, disk accesses increase more rapidly

Figure 8a

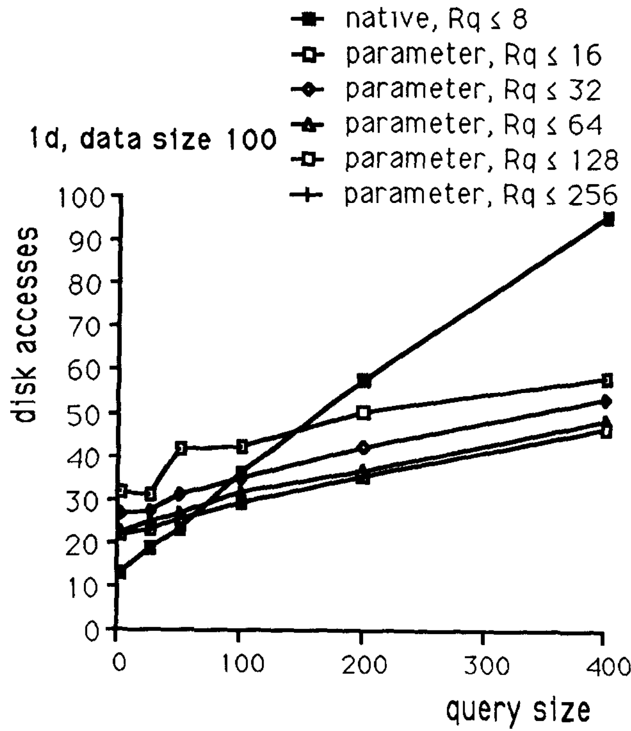
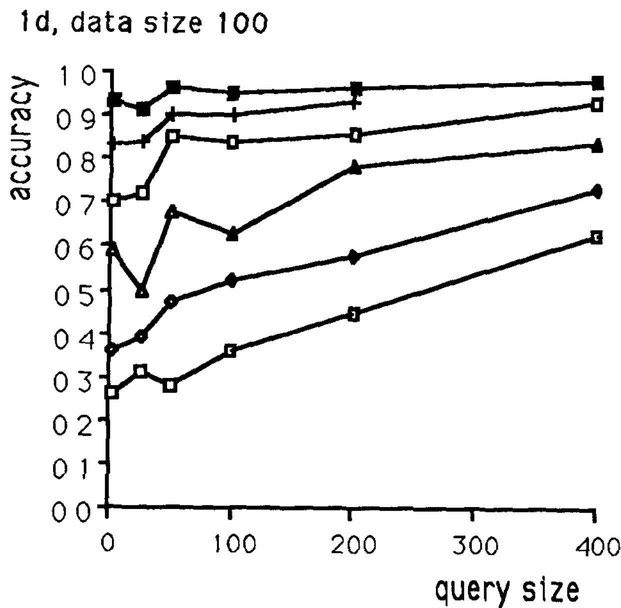


Figure 8b



In terms of accuracy, the native space results are extremely good - consistently above 90% except for the smaller queries with data sizes 10 and 50 The parameter space results cover a wide range Only with the highest query redundancy tested do the parameter space results approach the native space results

Figures 9-10 show the results for 2-d data The data object sizes are 200 x 200 (figure 9) and 500 x 500 (figure 10) 50 x 50 results were obtained and are qualitatively similar to the 200 x 200 results They are not shown due to space limitations For the disk access results, only the data size 500 x 500 graph clearly shows the pattern observed for 1-d data It is possible that the smaller data sizes will show the same crossing at much larger query sizes The accuracy results follow the same pattern as in 1-d, but the parameter space results are not as good, relative to the 1-d results

Figure 9a

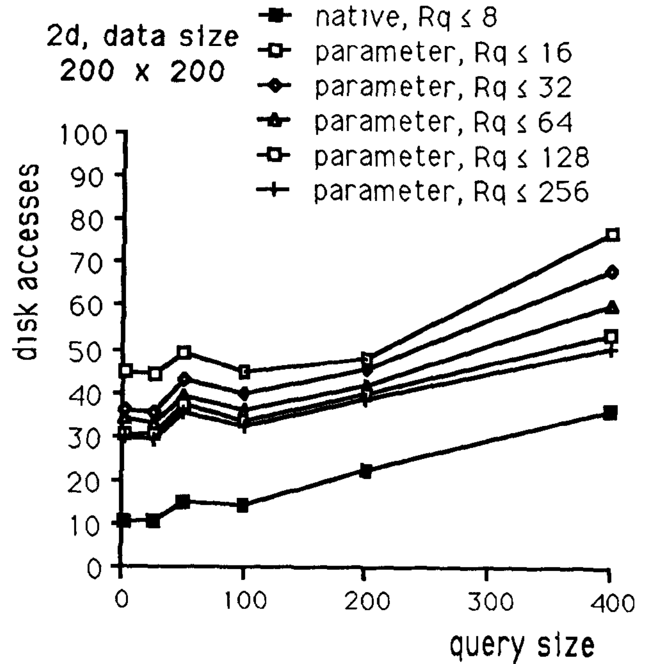


Figure 9b

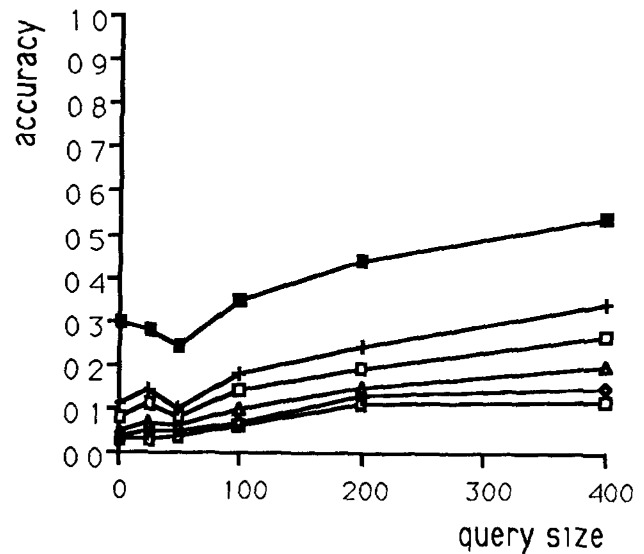


Figure 10a

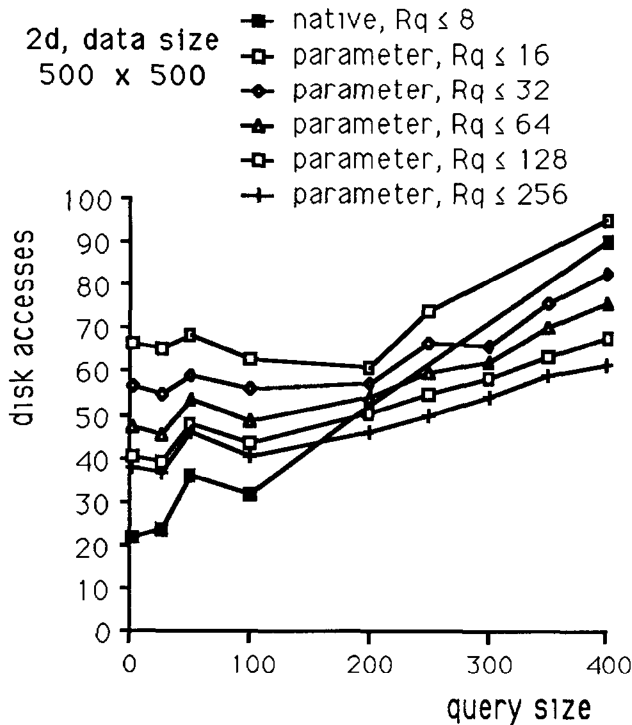
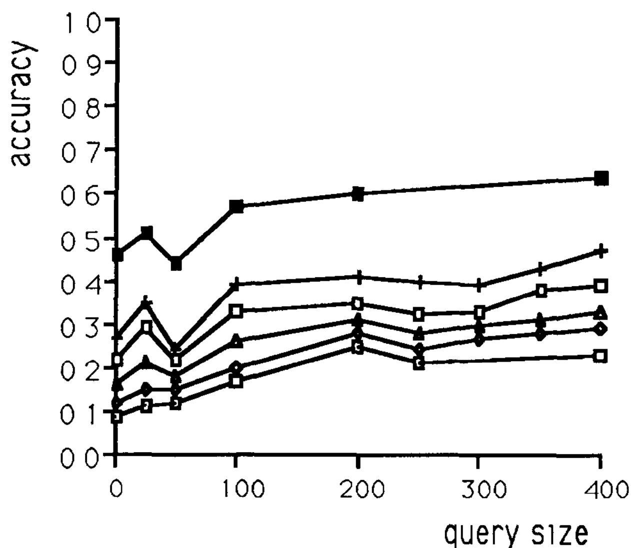


Figure 10b



These results show that neither approach is consistently superior with respect to disk accesses For sufficiently large data and query objects, parameter-space querying provides better results, given enough query redundancy The explanation is as follows increasing data object size leads to an increase in data redundancy in the native space representation As a result, a large query in the native space will retrieve several copies of each data object While the parameter space search provides less accuracy, each object is retrieved only once, whereas in the native space, accuracy is better, but each candidate is retrieved R_d times on the average Lowering data redundancy in the native space is unlikely to reverse the trend, to favor native space, since data object decompositions will be less accurate, resulting in the retrieval of more data objects [OREN89a]

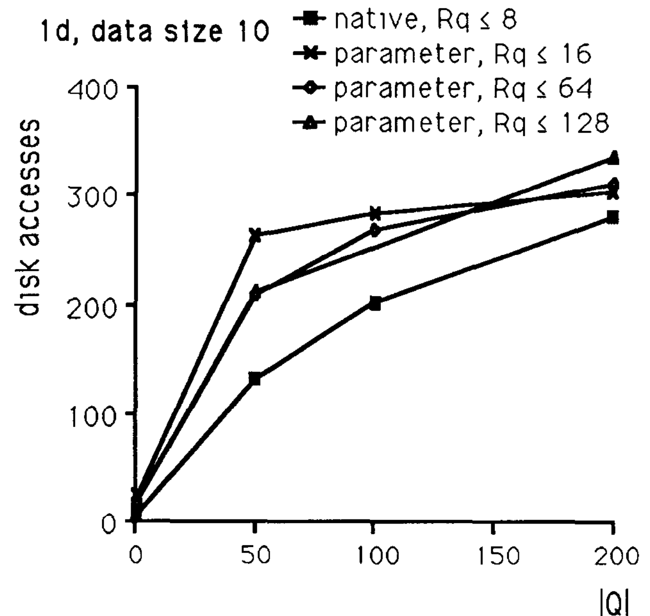
If data objects are sufficiently small, or query objects are sufficiently small, or if the cost of refinement per candidate is sufficiently high, then a native space representation should be used

5.2. Many-many queries

Many-many queries generalize one-many queries by permitting any number of query objects The distinction between "data" and "query" is not clear for a many-many query, but the terms are still useful in describing the two different transformations required for parameter-space searching

The experiments in this section examine disk accesses and accuracy for 1-d as the number of query objects, |Q|, is varied In each experiment, the size of the objects in Q are fixed The results for |Q| = 1 are from the experiments on one-many queries The disk access results reflect accesses to D and to Q Figure 11 shows the results for query objects of size 1

Figure 11a



Native space searching is cheaper for the range of |Q| examined, but it appears as if parameter space searching with $R_q \leq 16$ will be cheaper when there are more than 200 query objects (There are 5000 data objects) Disk accesses increases with |Q|, but the rate of increase drops as |Q| increases The change in rate is sharper for parameter space This phenomenon will be discussed below Accuracy is virtually constant across the range of |Q| examined The parameter-space accuracy results are far worse than the native-space results

Figure 11b

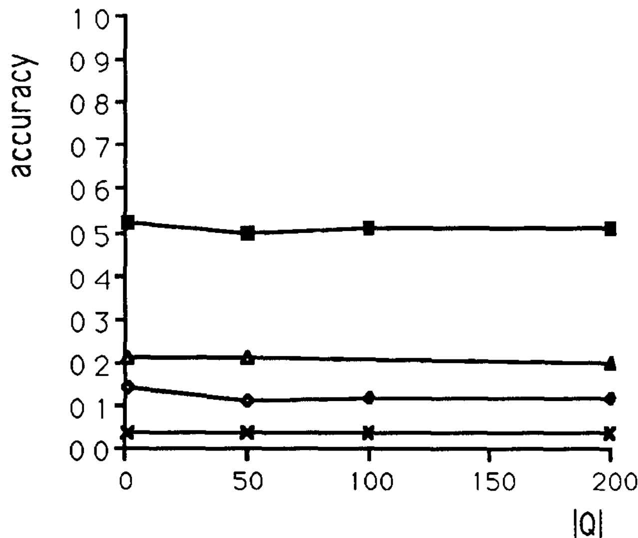


Figure 12a

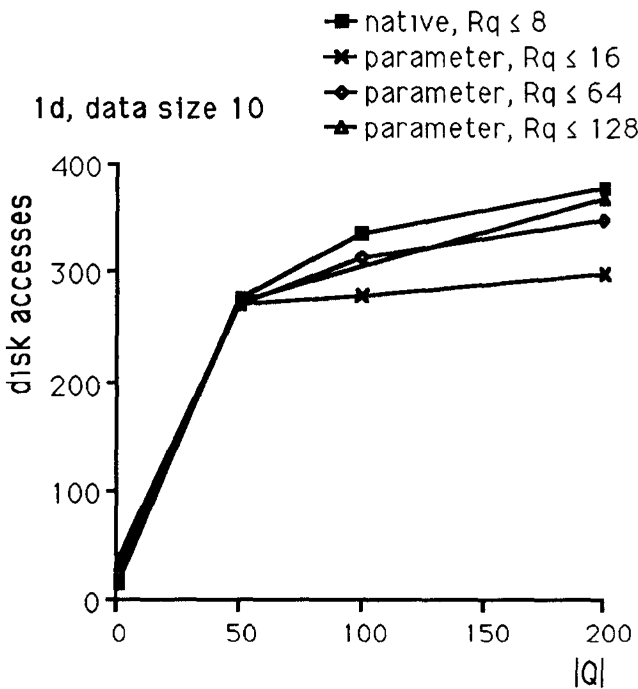
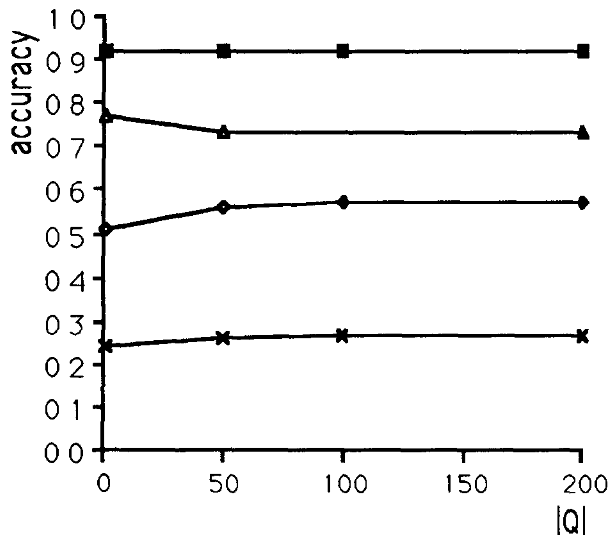


Figure 12 shows the results for query objects of size 100. The accuracy results are again constant, but are higher than in figure 11. The disk access results are indistinguishable until |Q| = 50. For the larger query object size, native space results get worse relative to parameter space results. Past |Q| = 50 the curves separate, and the rate of increase drops (this is more exaggerated than in figure 12). The reason for the "break" at |Q| = 50 is as follows. As |Q| increases from 1, more and more of the space is covered by query objects. In other words, less and less of the space is not covered by any query object. The spatial join algorithm takes advantage of these "gaps" by skipping over parts of D (the other input) that are not covered by any query object. As more query objects are added, fewer opportunities for this optimization occur. Finally, the space is entirely covered by query objects. Starting at |Q| = 50 (or slightly above), nearly all pages of D are being accessed, and further increases are due solely to increases in the space requirement for Q.

In figure 12a the best disk access results are obtained in parameter space with the lowest query redundancy examined, 16. The reason for this follows from the explanation given above. When the entire space is covered by query objects, all the data pages will be accessed regardless of query redundancy (i.e. accuracy), so the only way to save on disk accesses is to reduce the storage requirement of Q. This can be done by reducing query redundancy. As with |Q| = 1, it is necessary to choose between speed and accuracy - the strategy that leads to the best disk access results (for query size 100) also has the worst accuracy.

Figure 12b



The disk access results of figures 11a and 12a support the earlier claim that the time requirement of the spatial join is O(|D| + |Q|) disk accesses. The disk accesses never exceed the total number of pages occupied by D and Q.

It was noticed that in all native-space searches, every page of every query zkd b+-tree was accessed. However, in parameter space, a significant fraction of the pages were untouched. This can be explained as follows. For the data set and transformation used, the points representing data objects lie near the "main diagonal", (the line x = y in 2-d). The distance of a data point from the diagonal is determined by the size of the object. Each search region corresponding to a query object covers a large fraction of the space, including the upper left corner (see figure 2). Elements lying far away from the main diagonal contain no data elements and are rarely accessed. If a page of the Q zkd b+-tree contains only these distant elements, then the page will not be accessed. Other transformations of the same data set (e.g. as in [FALO89a]) would, most likely, show the same phenomenon.

5.3. Storage requirements

The zkd b+-trees for the parameter space data sets showed storage utilization between 68% and 70%. For native space, storage utilization was lower, from 63% to 69%. The storage

utilization decreased as data object size increased The reason for this may be related to the fact that the elements resulting from the decomposition of a single object were inserted in order of ascending z-value Larger objects yield more elements (since data objects were decomposed using the ERROR-BOUND strategy) Storage utilization could be increased by using b*-tree techniques (in which records are sometimes shifted among neighbors to avoid splitting) It is also possible that changing the order of insertion would have some benefit

The storage utilization results for the 2-d experiments resemble the 1-d results For the parameter space experiments, storage utilization was 69%-70% For native space, storage utilization was 69% for the 50 x 50 and 200 x 200 data objects, and 62% for the 500 x 500 objects

While the performance criteria of greatest interest are disk accesses and accuracy, absolute storage requirements cannot be neglected In many of the results presented above, the best results were obtained by native space searching, or by parameter-space searching with very high redundancy, e g $R_q \leq 128$ or 256 Obviously, the space requirement induced by this amount of redundancy is considerable

Data redundancy requirements were more modest For parameter spaces, data objects are transformed to points, so R_d was 1 In 1-d native space, R_d ranged from 1 27 to 3 02, while in 2-d the range was 1 27 to 4 87

6. Summary, conclusion and future work

Applications involving spatial data can be characterized by the kinds of queries that must be supported (one-many or many-many), by the expected sizes of data and query objects, and by the dimensionality of the space The results presented here provide some guidance in choosing spatial query processing strategies based on application characteristics

Parameter space searching is attractive because of its simplicity It can be supported on top of any access method designed for point data For one-many queries, it provides superior performance for the filter step, provided the data and query objects are sufficiently large, and that sufficient query redundancy is used The result of the filter step is likely to be less accurate than what can be obtained using native space searching In many applications, this is a secondary consideration since the refinement step is CPU-bound, unlike the filter step which is disk-bound

For many-many queries, native-space searching is at least as good as parameter-space searching until the space is "saturated" with query objects Then, parameter-space searching is faster, but contrary to the result for one-many queries, less redundancy is favored, especially as the number of query objects (i e , objects in the Q format) increases Accuracy results are extremely poor compared to native space Comparable accuracy is obtained only at very high levels of query redundancy

A drawback of parameter space searching is the requirement for dealing with two representations If an application makes heavy use of many-many queries, then this alone may indicate that a native space representation is appropriate This is due to the expense of maintaining two representations (D and Q), or creating the Q representation when needed

These conclusions are based on experiments carried out using the zkd b+-tree as the underlying spatial access method, and a particular transformation to parameter space Other spatial access methods and other transformations would, most likely, yield different quantitative results, but it would be surprising if the qualitative results differed Techniques that are applicable include different transformations to parameter space [HINR85, FALO89a], and the use of different space-filling curves, which lead to better physical clustering [FALO89b]

There are many ways in which this work can be extended examination of other space-filling curves and transformations, as in [FALO89b], experimentation with more complex spatial objects, and the development of analytical models I am currently working on a variety of issues relating to more complex spatial objects (specifically, polygons), and the implementation and analysis of the GF overlay algorithm [OREN88], essentially an n-input overlap query

References

FALO89a C Faloutsos, W Rego Tri-cell a data structure for spatial objects *Information Systems 14*, 2 (1989)

FALO89b C Faloutsos, S Roseman Fractals for secondary key retrieval Proc ACM PODS, (1989)

GUTT84 A Guttman R-trees a dynamic index structure for spatial searching Proc ACM SIGMOD, (1984)

HINR85 K H Hinrichs The grid file system implementation and case studies of applications Doctoral dissertation, ETH Nr 7734, Swiss Federal Institute of Technology, Zurich, Switzerland, (1985)

HORN87 D Horn, et al Spatial access paths and physical clustering in a low-level geo-database system Technical report, Technical University of Darmstadt, West Germany (1987)

LOME D B Lomet, B Salzberg The hB-tree a multi-attribute indexing method with good guaranteed performance To appear in *ACM Trans on Database Systems*

LOME89 D B Lomet, B Salzberg A robust multi-attribute search structure Proc Data Engineering, (1989)

MANT83 M Mantyla, M Tamminen Localized set operations for solid modeling *ACM Computer Graphics, 17*, 3 (1983) 279-288

MERR78 T H Merrett. Multidimensional paging for efficient database querying Proc Int'l Conference on Management of Data, Milan (1978), 277-290

NIEV84 J Nievergelt, H Hinterberger, K C Sevcik The grid file an adaptable, symmetric multi-key file structure *ACM TODS 9*, 1 (1984), 38-71

OREN84 J A Orenstein, T H Merrett A class of data structures for associative searching Proc 3rd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, (1984), 181-190

OREN86 J A Orenstein Spatial query processing in an object-oriented database system Proc ACM SIGMOD, (1986)

OREN88 J A Orenstein, F A Manola PROBE spatial data modeling and query processing in an image database application *IEEE Trans on Software Eng 14*, 5 (May, 1988) 611-629

OREN89a J A Orenstein Redundancy in spatial databases Proc ACM SIGMOD (1989)

OREN89b J A Orenstein Strategies for optimizing the use of redundancy in spatial databases Proc Symposium on Large Spatial Database (1989)

SELL87 T Sellis, N Roussopoulos, C Faloutsos The R+-tree a dynamic index for multi-dimensional objects Proc VLDB, (1987)

TAMM82 M Tamminen, R Sulonen The EXCELL method for efficient geometric access to data Proc 19th ACM Design Automation Conf (1982), 345-351