Efficient Processing of Large Spatial Queries Using Interior Approximations

Ravi K. Kothuri and Siva Ravada

Spatial Technologies, NEDC
Oracle Corporation, Nashua NH 03062
{Ravi.Kothuri,Siva.Ravada}@oracle.com

Abstract. Spatial data in CAD/CAM and geographic information systems involve arbitrarily-shaped 2- and 3-dimensional geometries. Queries on such complex geometry data involve identification of data geometries that interact with a specified query geometry. Since geometry-geometry comparisons are expensive due to the large sizes of the data geometries, spatial engines avoid unnecessary comparisons by first comparing the MBRs and filtering out irrelevant geometries. If the query geometry is large compared to the data geometries, this filtering technique may not be effective in improving the performance. In this paper, we describe how to reduce geometry-geometry comparisons by first filtering using the interior approximations of geometries (in addition to and after comparing the exteriors, i.e., the MBRs). We implemented this technique as part of the R-tree indexes in *Oracle Spatial* and observed that the query performance improves by more than 50% (or a factor of 2) for most queries on real spatial datasets.

1 Introduction

With the proliferation of spatial, CAD/CAM and multi-media databases in the past decade and the success of knowledge discovery in such domain-specific and inter-domain applications, scalable database servers for such non-traditional data have become a necessary and vital backbone in business applications. Most database vendors have aptly responded to this need to manage multi-domain data efficiently. Oracle launched the Spatial Cartridge for spatial, and Visual Information Retrieval (VIR) for image data; Informix introduced *data-blade* technology, and IBM developed the *spatial extender* for spatial data and the *QBIC* [15] image retrieval engine for image data. These products employ specialized techniques for processing queries on domain-based data. In this paper, we describe and study one such technique for the efficient processing of spatial queries.

Spatial data occurring in GIS, CAD/CAM applications consists of geometric data that include simple primitive elements such as lines, curves, polygons (with and without holes), and compound elements that are made up of a combination of the primitive elements. Oracle Spatial models its spatial data using an *sdo_geometry* data type which conforms to the OGC (Open GIS Consortium) standard. Oracle Spatial supports two types of spatial indexes for indexing spatial data: an R-tree [11,19,2,3,13,10], and a Quadtree [18] index. These indexes

C.S. Jensen et al. (Eds.): SSTD 2001, LNCS 2121, pp. 404–421, 2001.
© Springer-Verlag Berlin Heidelberg 2001

are implemented using the extensible indexing framework of Oracle and incorporate and enhance some of the best proposals from existing spatial indexing research. Spatial data in Oracle can be queried in one of the following ways.

- window queries with different “interaction” criteria [8,7]
  - intersection: identify data geometries that intersect a specified query geometry
  - inside: identify data geometries that are “completely inside” the query geometry
  - coveredby: identify data geometries that “touch” on at least one border and are inside the query geometry otherwise
  - contains: reverse of inside
  - covers: reverse of coveredby
  - touch: identify geometries that only “touch” the query geometry but disjoint otherwise
  - equal: identify geometries that are exactly the same as the query geometry
- within-distance (or epsilon [14]) queries: identify geometries that are within a specified distance from the query geometry
- nearest-neighbor queries: identify the k nearest neighbors [12,17] for a specified query geometry.

To efficiently process the above queries on complex spatial data, most vendor and research engines including Oracle Spatial use a 2-stage query filtering model as shown in Figure 1. In the first stage *exterior* approximations for data geometries such as minimum bounding rectangles (MBRs) and convex hulls [4,5,6], or quadtree tiles [18], which completely enclose the data geometries, are used. This first stage, usually referred to as the *primary filter*, typically involves a spatial index. Candidate geometries that may satisfy a given query criterion are identified in the primary filter stage with the help of (the exterior approximations in) the spatial index. In the second stage, referred to as the *secondary* filter, the identified candidate geometries are compared with the query geometry and the exact result set is determined and returned to the user.

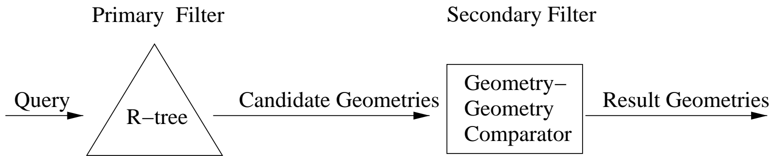

Fig. 1. Query Processing in Oracle Spatial.

In this paper, we study the performance of large spatial queries in the above 2-stage filtering model and propose new techniques to bypass the expensive secondary filter. In Section 2, we motivate the discussion and examine and analyze the bottlenecks in query performance. In Section 3, we propose a general

methodology for improving query performance. In Section 4, we propose specific techniques for improving query performance when the query window is a convex polygon. In Section 5, we address concave query windows which are more common in geographic datasets. Together, these techniques improve the query performance by upto 50% for most queries. In the final section, we conclude the paper with pointers to further research.

2 Analyzing the Performance of Large Queries

For most spatial datasets, the data geometries typically have hundreds or thousands of vertices and are arbitrarily complex. Secondary filter computation for such geometries takes a long time as opposed to the primary filter. Figure 2(a) illustrates this by comparing the time taken for primary and secondary filters in Oracle Spatial. The data consists of 230K polygons representing the US census blocks. The queries correspond to an approximate geometry that represents a circle of 0.25, 0.5, 1, 2, 5, 10, 25, 50, or 100 mile radius on the surface of earth. Since arcs and circles are not easily representable on the surface of earth the circle queries are densified to regular convex polygons in geodetic and non-geodetic domains. The center of the query is randomly-generated using locations of business centers across the United States. Note that such queries, where the query area is larger than those of the spatial features, are quite common in most GIS applications and spatial analysis queries. The x-axis shows the radius in miles from the query center and the y-axis plots the response time for each filter. The figure illustrates that the secondary-filter time is at least twice that of the filter time and dominates the overall computation time. This holds for all radii for the query circle.

The high cost for secondary-filter is due to two reasons: (1) the loading cost for geometries, or in other words, the cost of the table accesses that fetch candidate geometries, and (2) the comparison cost, which is the cost of comparing complex data geometries with the query geometry. For point datasets, the loading cost dominates and for polygon datasets, both costs contribute significantly. Since there is no easy way to reduce these costs, in this paper we examine alternate ways to improve performance: specifically by avoiding secondary filter comparisons whenever possible. In this context, we revisit the experiment of Figure 2(a). Interestingly, the number of geometries eliminated in the secondary filter are quite small compared to the total number retrieved. Figure 2(b) shows that in almost all the cases the difference in primary and secondary filter results is less than 10%. Besides, the figure also indicates that as the query radius increases a substantial number of the results are completely inside the query. From this, it could be inferred that whenever the query window is large compared to data sizes, checking for containment in the query may be a useful pruning strategy for bypassing the secondary filter, i.e., if a data geometry is completely inside a query geometry, then it could be accepted without passing it to the expensive secondary filter.

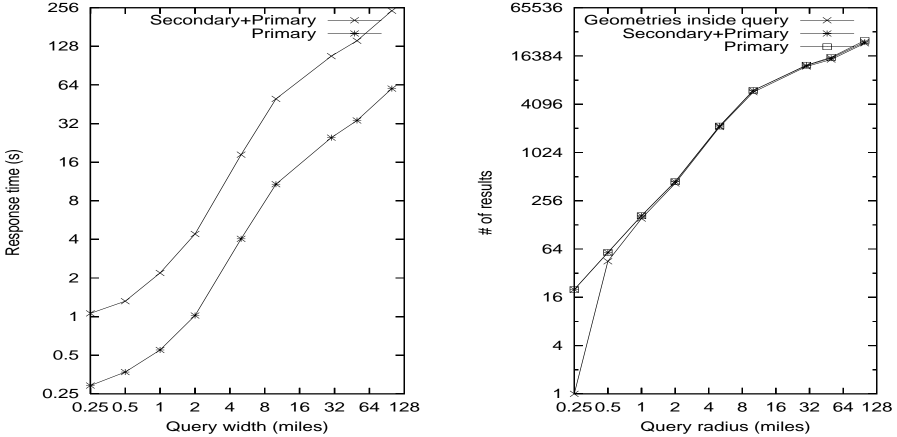

Fig. 2. Comparison of *primary* and *primary+secondary* filter queries on 230K-polygon USBG dataset: (a) Response times plotted: Secondary filter time is more than 2 times that of primary filter. (b) Secondary and Primary Filter results are plotted:The third curve shows the number of geometries inside the query.

Based on this insight, we propose to use *interior approximations* for query geometries to speed up the overall query time in Oracle Spatial. We primarily focus on R-tree indexes although the enhancements can also be made to quadtrees. We describe how to use the interior approximations to bypass secondary-filter comparisons. We then present novel techniques to compute the interior approximations for *convex* and *concave* query geometries. We incorporate these techniques in the Oracle-version of R-trees in Oracle Spatial. In experiments on real data sets using Oracle Spatial, we observe that our approach achieves upto a factor of 2 improvement in the execution speed of most queries.

Note that Badaway et al. [1] use interior tiles in quadtrees to speed up intersection queries. Our work generalizes the concept to non-intersection-type queries and examines its application in the context of R-trees. Unlike the work of Badaway et al. [1], our approach does not store any additional information to be maintained in the index. Instead, we compute the interior approximations only for query geometries and at query time. This strategy ensures no structural changes need to be made to the indexes of Oracle Spatial server. Since the interior is computed once per query, any additional overhead is amortized over the number of geometries that are retrieved. Experiments with this approach suggest that for some queries the time spent in the secondary filter reduces to less than 10% of the overall time (and to 16% of the original secondary-filter overhead). These results indicate that our approach requires little or no structural changes to a commercial database server like Oracle but obtains the meat of the performance enhancements from using interior approximations in a practical scenario. In the next section, we describe the general methodology for using interior approximations of a query to speed up query processing in Oracle Spatial.

3 Intermediate Filter in Oracle Spatial

Since comparison of two geometries (query geometry and a candidate data geometry) in the secondary filter is expensive, in this section we introduce an intermediate filter in *Oracle Spatial* to reduce the number of such expensive comparisons. This intermediate filter uses interior approximations of geometries to bypass the expensive secondary filter *whenever the data geometry is inside the query geometry*. Figure 3 illustrates this concept.

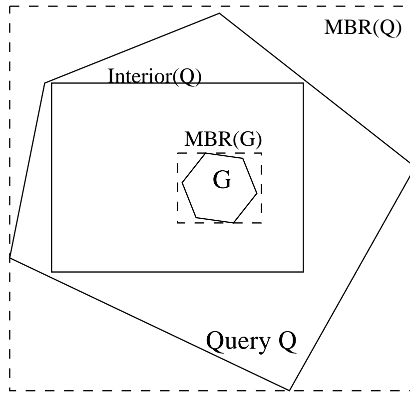

Fig. 3. Interior approximations for geometries.

Consider the query geometry Q and data geometry G of Figure 3. We assume the query is finding all geometries that intersect the query geometry (window query with intersection-type of interaction). The exterior approximations, which are MBRs for *Oracle Spatial* R-trees [16], are shown as dashed boxes and the interior rectangles are shown as solid boxes for the geometries. The primary filter determines that query Q may interact with geometry G using their exterior approximations, the MBRs, which happen to intersect each other. Since the exterior of the geometry G is *inside* the interior of the query Q, it follows that the geometry satisfies the *intersection* criterion of the query. This eliminates the need for the expensive loading and comparison of the geometry and query in the secondary filter. As illustrated in this example, intermediate filtering using interior approximations could enable some query-satisfying geometries to *bypass* the secondary filter. The two-stage filtering process of Figure 1 is augmented using such an intermediate filter as shown in Figure 4.

Next, we describe in detail how *Oracle Spatial* uses interior approximations in its intermediate filter. We then describe how and when the engine computes interior approximations. In the next section, we examine performance benefits from adopting this approach.

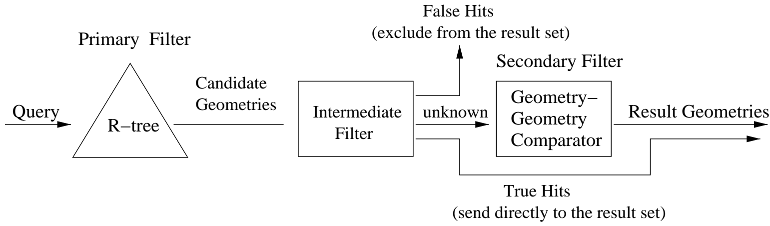

Fig. 4. Augmented primary Filter in Oracle Spatial.

3.1 Using Interior Approximations in Intermediate Filter

The R-tree index uses the MBRs of query and candidate data geometries to determine whether or not they interact. If they do interact, then the query and the geometry are passed on to the intermediate filter. The intermediate filter takes the query q, and a candidate geometry g and returns *true* if the geometry is to be included in the result set, *false* if not, and *unknown* if it cannot determine the relationship, in which case the query-candidate pair is passed to the secondary-filter to determine the exact relationship. These paths from the intermediate filter are shown with *unknown, true, false* labels in Figure 4.

Next, we describe the operational behavior of the intermediate filter for different queries. For the sake of notation, let $I(g)$ denote the interior approximations for a geometry g, and $E(g)$ denote the exterior approximation (in our case, the MBR) for the geometry g.

Window Queries

Intersection (g intersects q), inside (g inside q):

- If (I(q) contains E(g)) then return *true*
- Otherwise return *unknown*

Here, the interior approximation of the query is used to eliminate unnecessary comparisons in secondary filter as in the example of Figure 3. If the above test evaluates to *true*, then the geometry is directly included in the result set bypassing the comparisons in secondary filter. These are one of the most-frequently used queries in practice. As such, this optimization impacts a large gamut of user queries.

Contains (q inside g):

- If (E(q) is not inside E(g)) then return *false*
- Otherwise, return *unknown* (i.e., process using secondary filter)

Note that instead of passing directly to the secondary filter, the interior of the geometry, if exists, could be checked against the exterior of the query. However, in this case the primary filter would already test for the MBR of query being inside the MBR of the candidate geometry before invoking any such comparison. In most spatial applications, the number of such geometries satisfying the primary-filter test would be quite few given the small sizes

of data geometries (compared to the sizes of the query geometries). The improvements would be limited in this case.

Other types of interaction (covers, coveredby, touch,..):

- If (I(q) contains E(g)) then return *false*
- Otherwise, return *unknown*

Here, the interior approximation of the query is used to eliminate "false hits" that may otherwise propagate to the secondary filter.

Within-distance query can be implemented as a window query (query window enlarged by the specified distance) checking for intersection with data geometries.

Nearest-Neighbor Queries. In addition to regular window queries, nearest-neighbor algorithms too can benefit from the use of interior approximations. Nearest-neighbor algorithms could use the interior of the query to obtain a more *precise* estimation of the distances between query and a candidate geometry MBR and use these distances in query pruning and reducing actual-distance calculations. We briefly describe this process below.

Nearest neighbor queries retrieve a specified number, say k, of the data geometries that are closest to a query geometry. Nearest neighbor queries are evaluated by progressively refining the search area from a sphere of infinite radius to a sphere of distance-to-kth-neighbor radius. In this context, the actual minimum distance $d(q, g)$ between a query geometry and a candidate geometry g satisfies the following relation:

$$d(mbr(q), mbr(g)) \leq d(q, g) \leq d(I(q), I(g)) \leq D(I(q), E(g))$$

Here $d(a, b)$ denotes the minimum distance between objects a and b and $d(I(q), I(g))$ denotes the minimum of the minimum distances between every pair of rectangles q_i and g_i where $q_i \in I(q)$ and $g_i \in I(g)$. The distance $D(I(q), E(g))$ denotes the minimum of the maximum distance from any point in the interior of $I(q)$ to $E(g)$. This distance serves as an upper bound on the distance from query to data geometry.

The minimum distance between the MBRs, $(d(mbr(q), mbr(g))$, could be used to prune away all geometries farther than the current k-nearest-neighbor radius (see [12,17] for details). If a geometry falls within the current k-nn radius, then current systems pass it to the secondary filter to determine the actual distance and see if it needs to be included in the k-nn result set. Using the interior approximations for the query geometry, one could get a good upper bound on the actual distance, which can be used to determine whether or not to include the geometry in the result set. The actual distance has to be computed only when the interior-mbr distance is greater than the k-nn search radius, or when it is greater than the distance to the (k-1) neighbor (i.e., the geometry is being considered for the kth neighbor spot and for setting the k-nn radius). This strategy may eliminate expensive computation of the "actual distance $d(q, g)$" from query geometry to a candidate data geometry for candidate geometries that fall inside the k-nn radius.

In the rest of the paper, we primarily focus on window queries although the techniques can be easily extended to nearest-neighbor and other queries as described above. In the next two sections, we describe how to compute the interior for convex and concave geometries and present experimental results to illustrate the usefulness of the proposed techniques.

4 Interior Approximations for Convex Query Windows

Several algorithms exist in computational geometry literature for the fast computation of a maximally inscribed rectangle for a convex polygon. We use the approach of [9] to obtain a maximally inscribed rectangle for a convex polygon. A maximal interior rectangle with 3 points on the polygon boundary, is computed as follows: Divide the polygon boundary into 4 zones: south-west, north-west, north-east and south-east using the extreme vertices in x and y-dimensions as demarcators. In each zone, perform a binary search on the vertices in that zone to identify the (vertex or) edge where the interior rectangles pivoted at either of its vertices is maximum. (An interior rectangle pivoted at a vertex is computed by drawing x-parallel and y-parallel lines to the polygon boundary in other zones). Perform a binary search on the points on the edge to identify the maximum interior rectangle for that zone. Repeat this process for the other zones and obtain the interior rectangle that has maximum area. Likewise, interior rectangles with 2 vertices on the polygon boundary are computed. The authors of [9] show that the maximal interior rectangle is obtained by choosing the maximum area rectangle of the two.

To further maximize the interior area captured for a convex polygon, our approach is to divide a convex polygon into several pieces using the x or the y dimension (whichever has the maximum span) and find the maximally inscribed rectangle for each of these pieces. Our experiments on real data show that 4 pieces obtain a good trade-off between interior computation time and the area captured. Next, we describe a theorem which will be useful in improving the effectiveness of these interior approximations in reducing secondary-filter comparisons.

Theorem 1. If the vertices of a candidate geometry G are inside a convex polygon geometry Q, then the candidate geometry G is inside the convex polygon geometry Q.

Proof Sketch: Since the geometry Q is convex, all points inside are to the right of its edges when we traversed in a counter-clockwise order. If the edge (a, b) of geometry G has a segment which is not inside the convex geometry Q, then either a or b are not inside, which is not true, or an edge of Q intersects edge (a, b). This again implies that either vertex a or b is not to the right of an edge of Q (i.e., not inside in Q) which is a contradiction.

Corollary 1. If the vertices of the MBR of a geometry G (convex or otherwise) are inside a convex polygon geometry Q then the geometry G is inside the convex polygon Q.

The above corollary allows us to use all the interior approximations of a convex geometry *together* in determining if a polygon is inside. This rule, referred to as *combined-interior-acceptance rule* could be used to accept candidate geometries whenever their geometries are not inside any single interior MBR for a query geometry (i.e., the rules of the previous section fail). In such a case, we determine if the four corners of the MBR of a candidate geometry are inside *any* of the interior MBRs. To determine the usefulness of this rule, we ran the following experiment. We considered the USBG dataset which has 230K polygon geometries. We picked 100 random counties from the US counties dataset and used the convex hulls of these counties to query against the US Blockgroup dataset. The average of the results for these queries are shown in Table 1. We note that on an average 20% of the results are eliminated due to the *combined-interior-acceptance* rule. The query time improves by more than 35%, although the number of additional geometries eliminated were only 20%. This is because the geometries that are eliminated are larger in comparison to other geometries and their elimination from secondary filter comparison improved the performance drastically.

Table 1. Comparison of interior optimization with and without combined-interior-acceptance rule for convex polygons: The total number of results is 561.

	Combined-Interior-Acceptance	**Interior**
response time	3.06s	4.21s
# of secondary filter comparisons eliminated	475	380

4.1 Experiments

In this subsection, we describe experimental results that support the use of interior approximations for query geometries in spatial processing. We used two datasets: the US Blockgroup dataset consisting of around 230K arbitrarily-shaped polygon geometries, and the US Business Area dataset consisting of around 10M data points. For each of these datasets, we identified some of the most-densely populated areas and randomly generated query center points within the area. For instance, for the ABI dataset we used the New York Manhattan center as one query center. Using such query centers, we generated a query window of different radii from 0.25 miles width to 100 miles width. For each of these datasets, we compared the query response time for each query window (i.e., each query radius) with and without an intermediate filter. This intermediate filter computed the interior-rectangle for query geometries (or data geometries in case of a contains query) and applies the *combined-interior-acceptance* rule and other rules of Sections 3.1 and 4 whenever applicable. The primary filter is an Oracle version of an R-tree index constructed using a fanout of 35. We conducted the experiments on a Sun Ultra-1 166MHz machine with 256MB memory. We implemented the R-tree and the intermediate and secondary filters on top of Oracle server running a version of Release 9.0.1.

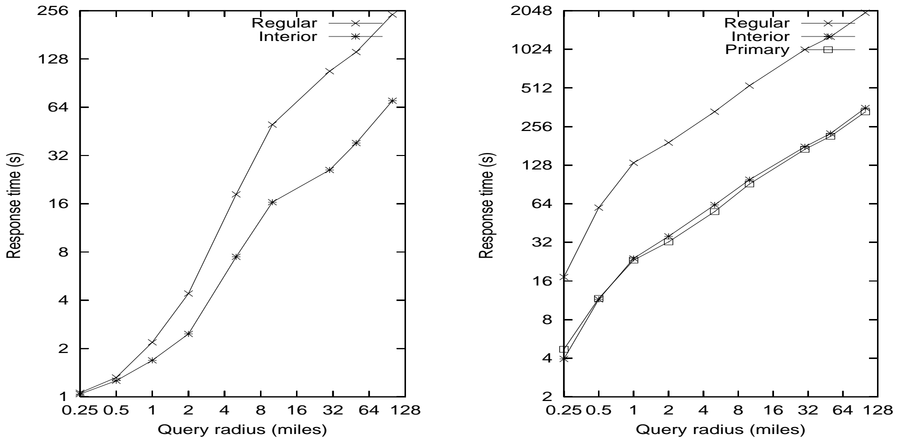

Fig. 5. Intersection query performance for (a) USBG dataset: *Interior* curve shows query times using for secondary+intermediate+primary filters, *Regular* curve shows times using secondary+primary filters, and *Primary* curve shows times for primary filter; *Interior* curve as good as *Primary* curve and (b) ABI dataset: *Interior* curve is nearly twice as fast as *Regular* and matches the performance of primary filter as almost all data are accepted using the interior optimization.

Figure 5 shows the results of comparison for the average query time with and without the intermediate filter that uses interior-rectangles for the query geometries. The queries identify all geometries that intersect the query windows. In the figures, the query window radius (in miles) is plotted along the x-axis and the query response (in milliseconds) is plotted along the y-axis. Note that both scales are logarithmic. Figure 5(a) shows the results for the USBG dataset. Three time curves are plotted: the time for the primary filter (*Primary* curve), the time for primary+secondary filter (*Regular* curve), and the time for primary+intermediate+secondary filter (*Interior* curve). The last one reports the total time for a query when processed using interior rectangles. We observe that using interior rectangle approximations improves query response times by around 25% for a query radius of 1 mile and 50% (or a factor of 2) for a query radius of 2 miles. At a radius of 2 miles, a query on the USBG dataset returned around 350 geometries. The performance gain improves as query windows become larger. For instance, at larger radii of 10-100 miles, the performance improves by nearly 70% (in other words, by a factor of 3). This is because as the query window becomes large, more and more candidate geometries fall inside the query interior and are straight away included in the result set bypassing the secondary-filter. This is verified by the fact that *Interior* curve is quite close to the *Primary* curve which implies the time is spent in secondary-filter is less than 10% of the overall query time (which is less than 16% of the original secondary-filter overhead).

Similar performance enhancements are also obtained for the ABI dataset as shown in Figure 5(b). Notice that for the ABI dataset, even the smallest size

query shows an improvement unlike the USBG dataset. This is because ABI dataset contains point data and potential candidates can be accepted right away using the interior rectangles for even the smallest-size query. This excludes the need for a secondary-filter comparison where the points (geometries) are loaded again from the database table and re-compared with the query. Note also that the overall performance matches that of the primary filter as most of the result data fall inside the interior rectangles and are therefore accepted right away.

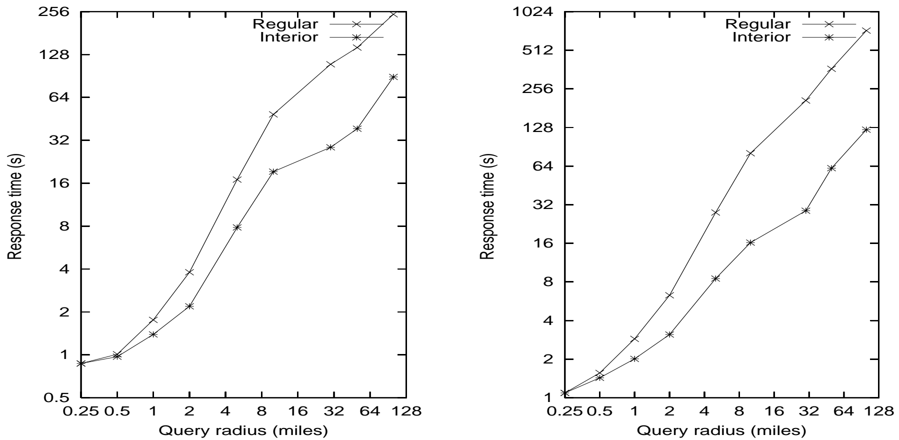

Fig. 6. Performance for USBG dataset: (a) Inside queries: Interior rectangles are used for returning satisfying geometries (b) Touch queries: Interior rectangles are used for eliminating "false hits".

Next, we present the results for inside-type window queries. The results for the USBG dataset are plotted in Figure 6(a) and indicate gains similar to those for intersection queries. In these queries, the interior approximations are used to identify candidate geometries that satisfy the query and to include them in result set thereby bypassing the secondary filter. The interior approximations are also useful in eliminating "false hits" for other-type of interactions such as touch, covers, or coveredby. Figure 6(b) shows the results for touch queries. We observe that the performance gains for 2 mile radius query is around 50% (a factor of 2), and for 10-100 mile radii it is around 84% (a factor of 6). Similar results are also obtained for other masks and for the ABI dataset.

5 Interior Approximations for Concave Queries

The above technique for computing the interior approximation could be extended to concave polygons as follows: divide the concave query window into convex pieces at *concave* vertices and compute interior rectangles for each of them. This approach has the following drawbacks: (1) the number of concave vertices for

most practical concave query windows is quite large: of the order of hundreds. This might result in a large number of interior rectangles – requiring a lot of memory and possibly a spatial-searching mechanism for quick comparison with candidate MBRs. (2) Since the interiors of different pieces of the concave polygon need not be contiguous any more, the combined-acceptance-rule could not be applied. This means the query interior is treated as fragmented and it is not possible to directly accept a data MBR that spans multiple pieces but is still completely inside the combined interiors. To avoid these drawbacks, we propose an alternate technique for finding the interior approximations for concave query windows. The idea is to tile the concave geometry and identify tiles that are interior to it. Candidate MBRs are also tiled and the tiles for the candidate MBR are searched among the interior tiles for the concave query window.

5.1 Computing the Interior

We illustrate the process of computing the interior of a concave query using the simple concave query polygon of Figure 7.

1. Compute the MBR of the concave window.
2. Using the MBR as the tiling domain, tile the concave geometry as in the case of a quadtree. So, if level-1 tiling is chosen, then the MBR is divided into 4 quadrants, if level-2 tiling is chosen each level-1 quadrant is subdivided into 4 sub-quadrants and so on.

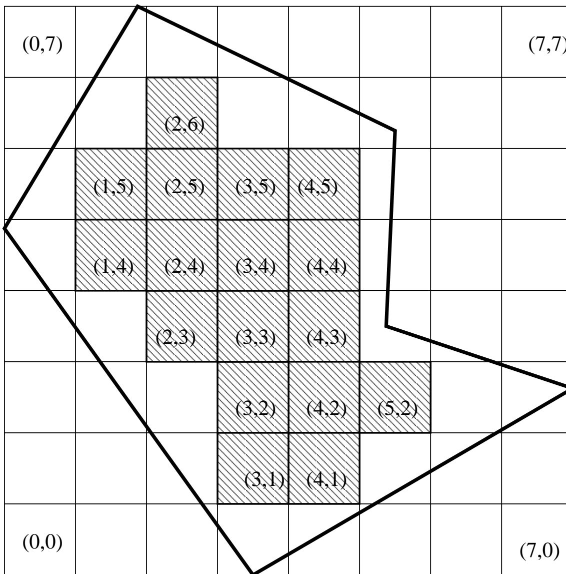

Fig. 7. Interior tiling for concave query polygon: Query is shown in thick lines. Interior tiles are shown as dashed boxes and are identified by their x- and y-positions with respect to the lower-left corner of MBR of the polygon.

3. Each tile is identified by *(xcode, ycode)*, where *xcode* and *ycode* refer to the tile coordinates along the x- and y-axes respectively. Specifically, in Figure 7 the tile at the lower-left corner has (xcode, ycode) values as (0,0) and the tile at the lower-right corner has (7, 0).
4. Among the tiles that cover the concave window, identify the tiles that are interior. In Figure 7, the interior tiles are shown as shaded boxes.
5. Store the interior tiles twice: once in an X-ordered array and a second time in a Y-ordered array. The X-ordered array orders the tiles first by the xcode value of the tile and if these match then using the ycode values. The Y-ordered array orders the tiles first using ycode values and then using xcode values for tiles with matching ycodes. For illustration purposes, the X-ordered array (X-array for simplicity) for Figure 7 has the interior tiles in the following order:

$$\{(1,4),(1,5),(2,3),(2,4),\ldots,(5,2)\}$$

5.2 Using the Concave Interior in Intermediate Filter

Whenever the query MBR intersects the data MBR, the data MBR is compared with the interior tiles of the concave geometry. If the data MBR is inside or matches the interior, then the data geometry is right away included or excluded (based on the interaction criterion: see section 3 for details) in the result set bypassing the secondary filter. Otherwise, the data geometry is passed on to the secondary filter. Next, we describe how to determine whether a data MBR is *inside* the interior of the concave query window.

Consider two data MBRs A and B that happen to intersect the query MBR of Figure 8. To determine whether a candidate MBR is *inside* the query interior, we search in the interior tile arrays of the concave query as follows:

1. Identify the tile corresponding to each corner of the candidate MBR. Let these be the tiles (x0, y0), (x1, y0), (x1, y1), and (x0, y1). For instance, the data MBR A has corners in the tiles (2,3), (4,3), (4,5) and (2,5). Likewise, the data MBR B has corners at (4,0), (5,0), (5,2) and (4,2).
2. Next, we determine if *all* the 4 tile borders of the data MBR corresponding to the 4 sides of the MBR are *inside* the interior of the query. If they are, then the data MBR is *inside* the query window. For example in Figure 8, for MBR A all the tile borders are inside the interior tiles of the query whereas for MBR B only one is inside and the other 3 span non-interior tiles.
   Each border of the MBR is searched in the tile array as follows.
   – If the border is parallel to the x-axis (i.e., y-code values are same for both endpoints), then search in the Y-ordered tile array, otherwise search in the X-ordered tile array. The border (i.e., interval in tile domain) is used to perform a range search in the appropriate interior (X-ordered or Y-ordered) tile array. If the number of tiles retrieved is equal to the length (i.e., difference in x-codes) of the border, then the border is said to be inside; otherwise not. For example, the border joining tiles (2,3) and

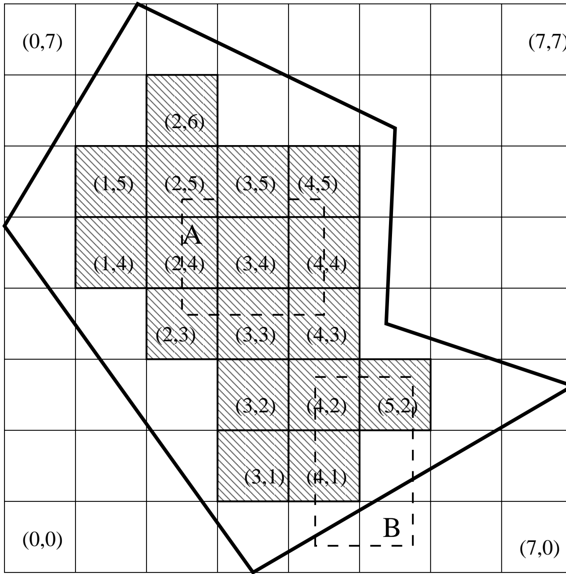

Fig. 8. Query processing using interior tiles: Candidate MBRs A and B are shown as dashed rectangles. MBR A is inside interior tiles of the concave polygon, whereas MBR B is not.

(4,3) is *inside* the query interior as all tiles between (2,3) and (4,3) are also interior whereas the border joining tiles (4,0) and (5,0) of MBR B is not inside as the tile (5,0) is not an interior tile of the query. Note that each such border search involves $2 * \log n$ time where n is the number of interior tiles for the query window.

Since binary searches are performed, the above searching takes at most $O(\log n)$ time if there are at most n tiles. As will be seen in our experiments, this time is quite negligible for most queries.

5.3 Choosing the Tiling Level

The next issue that needs to be addressed is how to choose an appropriate tiling level for the query geometry. The amount of interior that is captured increases with the tiling level and contributes directly to reduction in query time. At tiling level 0 or 1, there will not be any interior tiles for any query window since all relevant tiles for the query would intersect the query polygon border. At tilling level 2, there are at most 4 interior tiles out of a total of 16 tiles; at level 3, there are at most 36 interior tiles out of a total of 64 tiles (56% interior); at level 4 there are at most 196 interior tiles out of a total of 256 tiles (78% interior). In general, assuming the window to be a simple MBR for the best case, at level l, there will be $4 * (2^l - 1)$ tiles that form the border (not interior) out of a total of 4^l tiles.

This indicates that for higher tiling levels, the amount of interior that could be captured will be more and this may lead to higher gains in query performance (due to elimination of secondary-filter comparisons). However, as the tiling level increases, the cost of tiling the window to the specified level is an additional overhead to the query time and may offset any gains from the interior-based query gains. In what follows, we describe some experiments to identify the best tiling level and the resulting performance gains from this strategy.

5.4 Experiments

First, we examine how query performance varies with the tiling level. For this purpose, we experiment with both datasets: 230K USBG dataset of polygons, and 10M ABI dataset of points. As concave query windows, we chose 100 randomly selected counties from the set of 3200 counties in the United States. Figure 9 shows the results for different tiling levels. The tiling level is plotted along the x-axis and the query response time along the y-axis. For small tiling levels (2 or 3), the number of interior tiles is also small and hence the performance improvements will be negligible. As the tiling level increases from 3 to 4, or 4 to 5 as in Figure 9(a), the number of interior tiles but the tiling time also increases and may offset any gains in response time due to acceptance of interior candidates.

Due to the conflicting effects of increasing or decreasing the tiling level on the number of interiors and the tiling time, tiling levels 3 and 4 achieve good performance for one of the two datasets and perform poorly for the other. This is illustrated in Figure 9(a) and (b) using USBG and ABI datasets. Tiling level 3 achieves best performance for USBG data and performs relatively poorly for

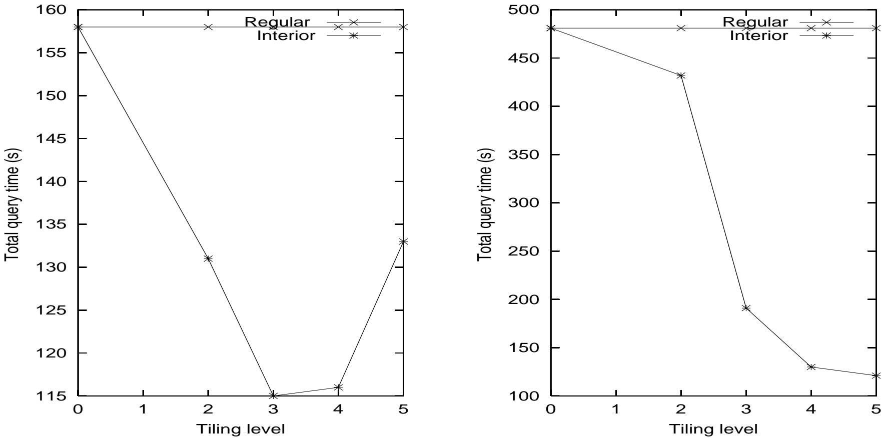

Fig. 9. Anyinteract-Query performance variation with tiling level for (a) USBG dataset: Level-4 interior tiling obtains 25% improvement over regular approach and (b) ABI dataset: level-4 interior tiling obtains 75% improvement over regular approach.

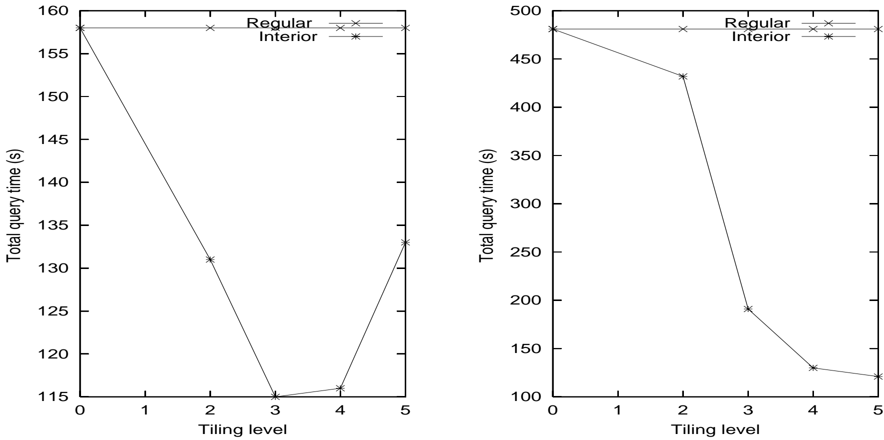

Fig. 10. Touch-Query performance variation with tiling level for (a) USBG dataset: Level-4 interior tiling obtains 30% improvement over regular approach and (b) ABI dataset: level-4 interior tiling obtains 75% improvement over regular approach.

ABI data, whereas tiling level 5 achieves best results for ABI data and performs poorly for USBG data. In contrast, tiling level of 4 achieves nearly the best performance for both the point and polygon datasets: for the USBG dataset, this level of tiling obtains 25% improvement, and for the ABI dataset, it obtains around 75% improvement in query response time. For the ABI dataset, tiling level 4 obtains nearly 75% improvement in performance, as nearly 2000 out of 2500 results are accepted using the interior optimization.

The results for a "touch"-type of query interaction for the two datasets are shown in Figure 10. We observe that once again tiling level 4 obtains consistently good performance gain: 30% for USBG dataset and nearly 75% for ABI dataset. Similar results are also obtained for other interaction-type queries. From these results, we conclude that a tiling level of 4 achieves good performance gains in all cases. In all subsequent experiments, we work with a tiling level of 4.

Next, we examine the improvements for more rugged query windows such as state boundaries. Tables 2 and 3 present the improvements from interior approximations when the state of "Maryland" is used as the query window for the USBG dataset and the ABI dataset respectively. We observe that the query performance improves by 25-30% in both cases.

6 Conclusions

In this paper, we examined the performance of large spatial queries that frequently occur in spatial databases. Most of the time spent in the query goes in secondary-filter comparisons and such expensive comparisons can be reduced by using interior approximations for query geometries. We proposed efficient and effective techniques for computing and using interior approximations for both

Table 2. Performance gains from interior optimization for USBG dataset using Maryland state as the query window: Improvement is around 25%.

Query interaction	**Concave Interior response time (s)**	**Regular response time(s)**
Anyinteract	78	92
Inside	102	132
Touch	530	722

Table 3. Performance gains from interior optimization for ABI dataset using Maryland state as the query window: Improvement is around 30%.

Query interaction	**Concave Interior response time (s)**	**Regular response time(s)**
Anyinteract	450	600
Inside	413	612
Touch	390	580

convex and concave query windows. Experiments with real datasets inside Oracle server showed that using such approximations, query performance can be significantly improved. For convex query windows, query times can be reduced by 85% for point data and by a factor of 60% for polygon data. For concave query windows, query times are reduced by 75% for point data and by 30% for polygon data. In future, we plan to extend these performance-improving techniques to other types of queries like nearest-neighbor and to other spatial indexes like quadtrees. In addition, we would also like to investigate alternate mechanisms for efficiently finding the interior of concave geometries.

References

1. W. M. Badaway and W. Aref. On local heuristics to speed up polygon-polygon intersection tests. In *Proceedings of ACM GIS International Conference*, pages 97–102, 1999.
2. N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R* tree: An efficient and robust access method for points and rectangles. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 322–331, 1990.
3. S. Berchtold, D. A. Keim, and H. P. Kreigel. The X-tree: An index structure for high dimensional data. *Procȯf the Int. Conf. on Very Large Data Bases*, 1996.
4. T. Brinkhoff, H. Horn, H. P. Kriegel, and R. Schneider. A storage and access architecture for efficient query processing in spatial database systems. In *Symposium on Large Spatial Databases (SSD'93), LNCS 692*, 1993.
5. T. Brinkhoff, H. P. Kriegel, and R. Schneider. Comparison of approximations of complex objects in spatial database systems. In *Proc. Int. Conf. on Data Engineering*, pages 40–49, 1993.

6. T. Brinkhoff, H. P. Kriegel, and B. Seeger. Efficient processing of spatial joins using R-trees. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 237–246, 1994.
7. M. J. Egenhofer. Reasoning aobout binary topological relations. In *Symposium on Spatial Databases*, pages 271–289, 1991.
8. M. J. Egenhofer, A. U. Frank, and J. P. Jackson. A topological data model for spatial databases. In *Symposium on Spatial Databases (SSD)*, pages 271–289, 1989.
9. P. Fischer and K. U. Hoffgen. Computing a maximum axis-aligned rectangle in a convex polygon. In *Information Processing Letters, 51*, pages 189–194, 1994.
10. V. Gaede and O. Gunther. Multidimensional access methods. *ACM Computing Surveys*, 30(2), 1998.
11. A. Guttman. R-trees: A dynamic index structure for spatial searching. *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 47–57, 1984.
12. G. Hjaltson and H. Samet. Ranking in spatial databases. In *Symposium on Spatial Databases (SSD)*, 1995.
13. S. T. Leutenegger, M. A. Lopez, and J. M. Edgington. STR: A simple and efficient algorithm for R-tree packing. In *Proc. Int. Conf. on Data Engineering*, 1997.
14. King-Ip Lin, H. V. Jagdish, and C. Faloutsos. The TV-tree: An index structure for high-dimensional data. *VLDB Journal*, 3:517–542, 1994.
15. W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasman, D. Petkovic, and P. Yanker. The QBIC project: Querying images by content using color, texture and shape. In *Proc. of the SPIE Conf. 1908 on Storage and Retrieval for Image and Video Databases*, volume 1908, pages 173–187, February 1993.
16. K. V. Ravikanth, S. Ravada, J. Sharma, and J. Banerjee. Indexing medium-dimensionality data in oracle. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 1999.
17. N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 71–79, May 1995.
18. H. Samet. *The design and analysis of spatial data structures.* Addison-Wesley Publishing Co., 1989.
19. T. Sellis, N. Roussopoulos, and C. Faloutsos. The r$^+$-tree: A dynamic index for multi-dimensional objects. *Procöf the Int. Conf. on Very Large Data Bases*, 13:507–518, 1988.