Spatial Priority Search: An Access Technique for Scaleless Maps

Bruno Becker
Institut für Informatik
Universität Freiburg
Rheinstraße 10–12
D–7800 Freiburg

Hans-Werner Six
Praktische Informatik III
FernUniversität Hagen
Postfach 940
D–5800 Hagen

Peter Widmayer
Institut für Informatik
Universität Freiburg
Rheinstraße 10–12
D–7800 Freiburg

Abstract

In geographic information systems, an important goal is the maintenance of seamless, scaleless maps. The amount of detail desired on a map decreases with decreasing scale. Cartographic techniques called *generalization* define the representations of geographic objects, depending on the scale. While generalization as a whole is considered an art, simple automatic generalization techniques exist for simple geometric objects. For polygonal lines and polygons, simplification techniques assign priorities to points. A map at a desired scale is then obtained by ignoring all points of sufficiently low priority. This implies that a geometric object appears on a map only if its priority is high enough, and also that an object is represented only by those of its defining points that have sufficiently high priority. The efficiency of retrieving a map of some area at a certain scale ideally should only depend on the amount of data retrieved.

In this paper, we present algorithms and a fully adaptive data structure that efficiently supports insertions, deletions, updates and geometric queries at an arbitrary location for an arbitrary scale. Our data structure adapts to dynamically changing data in such a way that objects outside the query area or of irrelevant priority do not influence the efficiency of the query substantially. The structure is of broader interest, because it supports general proximity queries with priority threshold for nonzero size geometric objects.

This work has been supported by DFG grants Si 374/1 and Wi 810/2

Furthermore, we show how to partition polygon corner points of different priorities such that they can be stored in the structure with practically no redundancy, and polygons for any priority threshold can be reassembled easily. A performance evaluation of our implementation with topographic data reveals that a performance gain of a high factor can be achieved with the structure under realistic assumptions.

1 Introduction

One of the major purposes of geographic database systems is computer support for storing and retrieving computerized maps. Map data contain geometric and non-geometric parts. For instance, the geometric part may be polygons described by line segments, where line segments are given by their endpoints. Any such point corresponds to a point in the real world coordinate system; the line segment connecting two points is an abstraction of the real world curve connecting these two points. Depending on the scale in which a map is viewed, the display size of line segments varies. For a small scale, it may be undesirable for a viewer to see all endpoints of line segments stored in the database. This is due primarily to the desire of the viewer to see relevant information only; too many details may hinder rather than help. For instance, Figure 1(a) shows the city boundaries of Berlin, drawn to scale 1 : 20000, and Figure 1(c) shows the Berlin city boundaries with scale 1 : 50000, drawn with fewer line segment endpoints than Figure 1(a).

Line generalization techniques [3, 11, 14, 18] have been developed for the systematic adaptation of the number of points that represent a polygonal line segment to the desired scale. Generalization in full generality, i.e., the systematic adaptation to a given scale of all the features to be displayed for a map, employs line generalization within ranges of geometric scaling, but in addition exhibits semantic leaps at certain changes of scale [6, 10, 12, 13, 16, 19]. An object as a whole appears on a map only if it is large enough for the desired scale. Line generalization allows for the automatic derivation of Figure 1(c) from Figure 1(a) by selecting some of the line segment endpoints in Figure 1(a) as more relevant than others. Figure 1(b) shows the effect of applying a line generalization to Figure 1(a); Figure 1(c) is just Figure 1(b), drawn at smaller scale. As the result of applying any of these automatic line generalization methods, with each of the line segment end points a priority value – the *generalization index* – indicating the importance of the point is associated. A map at a desired scale is obtained by ignoring all points of low enough priority.

Hence, different data have to be stored for the same map at different scales. As a consequence, usually geographers maintain several maps of an area at several fixed scales. Since a data point will then be stored for more than one map on average, this entails a potentially high degree of redundancy and storage space consumption. In addition, inconsistencies, e.g. arising from updating a map at one scale only, must be carefully avoided.

(a) (b) (c)

Figure 1

Consider the alternative of storing a map at the largest available scale only. Then a map at any desired scale may be generated by applying a line generalization method to the data. This is certainly more flexible than the previous approach: For instance, when two maps of different themes (e.g. city boundaries and land use polygons) need to be integrated to form a third map, and the two given maps are at different scales, it is possible to first downscale the larger scale map to the scale of the smaller scale map, and then integrate both at the same scale. In order not to apply line generalization methods for each rescaling operation, it may be beneficial to compute a generalization index for each line segment endpoint once and for all, and store it with the point itself in the database.

The problem with this approach is a potentially low efficiency, whenever a rescaling operation is combined with a proximity query. For instance, rescaling all objects in a rectangular subrange of a map translates into a retrieval operation on the geographic data base, where all points in the query range with generalization index below the threshold value corresponding to the desired scale must be retrieved. The importance of proximity queries, like e.g. range queries, in geographic databases is well known; access structures for supporting proximity queries have been proposed in the literature [2, 4, 5, 7, 9, 17]. The ubiquity of rescaling operations comes from the fact that in our model geographic data are stored at the largest available scale, which may rarely be the scale desired by the user for some operation. An access structure that simultaneously takes into account proximity queries and priority limit, the reactive tree, has been proposed [20] a short time after a preliminary version of this paper [1]. The reactive tree, however, only supports a fixed set of priority values, and it allocates objects according to these values in fixed tree levels, one or more for each value. As a consequence, the reactive tree cannot react to dynamic changes, and it cannot adapt access paths appropriately to the set of objects to be stored. In this paper, we propose a fully adaptive access structure for proximity queries with priority limit, and we show that it is especially well suited for geographic databases.

2 Separate priority access

To make priority queries efficient, we propose not to store geometric objects, like polygonal line segments (polylines) or polygons, as atomic entities, but rather to partition the set of their defining points according to the priority. Then a geometric access structure may be applied to geometrically cluster points of the same priority. A representation of a polyline at a large scale can be obtained from the representation of the polyline at a smaller scale by throwing in additional data points, according to the order of the points on the polyline at the largest scale. The two sequences of line segment endpoints, one for the smaller scale representation, the other for the additional data points, can be merged correctly, if for each point its position in the sequence of all data points for the polygonal line segment, i.e., the position in the polyline for the largest scale, is known. Therefore, let us associate this position with each point. Assuming that the generalization index is a positive integer varying between 1 for points present at the smallest scale to some maximum for points present only at the largest scale, a map at a scale corresponding to generalization index i can be generated by merging the point sequences for generalization indices 1 to i.

Since a range query returns all parts of objects intersecting the query range, it is necessary to keep track of line segments themselves, and not only their endpoints [17]. For many applications, it makes sense to store spatially extended objects according to their bounding boxes. Then the answer to a range query consists of those objects whose bounding boxes intersect the query range. Since bounding box information consumes some storage space, and objects should be cut into manageable pieces [15], let us give a fixed number of data points of a polyline with the same generalization index one common bounding box. Now, to correctly answer a range query in the way just described, it is not enough to define the bounding box of a sequence of points with generalization index i as the smallest rectangle enclosing the points with generalization indices 1 to i in the sequence. In Figure 2(a), where x_j denotes a point with generalization index j and the objects contain 2 points, a range query with query range r and generalization index 2 needs to report line segment $a_1a_2 \cap r$, but r does not intersect the bounding box of a_1b_1 and hence the coordinates of point a_1 are not found. We therefore define a bounding box as the smallest rectangle enclosing all data points in the considered sequence, even those with higher generalization index (see Fig. 2(b)). If the considered sequence with generalization index i is not the whole polyline, the bounding box has to be extended such that it encloses the next point on the polyline with generalization index at most i at both ends. In the example of Fig. 2(a), the next points of a_2b_2 are c_2 and a_1 and the corresponding bounding box encloses both points. Since priority numbers are defined by line generalization methods, this will in general not lead to much larger bounding boxes.

(a) (b)

Figure 2

In a first approach, we propose to store each set of bounding boxes with associated priority in a separate geometric access structure for spatially extended objects, e.g. the R-file [9]. This clearly supports queries within each priority very well; it can therefore be used to answer proximity queries with priority limit i by querying all R-files for priority numbers at most i and merging the results. Our experiments (see Section 5) show the efficiency of this method. Nevertheless, at least for queries with small query regions, the directory searching overhead in more than one access structure may be substantial. This is especially undesirable, since the query region is exactly the same for all priorities. We therefore propose to integrate all single priority structures into one common access structure that simultaneously clusters objects according to their priorities and their geometry in a fully adaptive way. Our access structure, the *Priority Rectangle File (PR-file)* avoids repeated geometric selection. This is a particulary useful feature for *zoom-in* queries, where the user narrows the query region and thereby increases the level of detail, starting from a query and detail level selected before. The additional effort for zooming in is very small for our integrated structure, but consumes an extra directory search from the root for separate structures.

3 Integrated priority access: the PR-file

Let us now present the basic idea of integrating geometry and priority; details of operations will be described more formally in the next section. To be precise, a geometric object g is characterized by a bounding box b, a priority p and further attributes a; we write $g = (b,\ p,\ a)$ and denote the components of g by $g.b$, $g.p$ and $g.a$. To keep the presentation simple, we restrict ourselves to two-dimensional geometry; the extension to more dimensions is straightforward. Then, a bounding box is a rectangle in the plane whose sides are parallel to the Cartesian coordinate axes; it is given as $b = (\mathit{left},\ \mathit{right},\ \mathit{bottom},\ \mathit{top})$. A priority number is a positive integer, where 1 denotes the highest priority. We deliberately assume thata geometric object g can be uniquely identified by the pair $(g.b,\ g.p)$; this is only a minor technicality that facilitates the subsequent discussion.

The PR-file is designed to support proximity queries on sets of rectangles with priority limits. As an example proximity query, we describe the range query operation in detail; other proximity queries, such as nearest neighbour or zoom-in and zoom-out, are carried out correspondingly. In more detail, the PR-file structure supports the following operations for a set G of geometric objects stored in a PR-file *PR(G)*:

- *priority range query* $(r,\ p,\ PR(G))$:
  find and return each object in G with priority number at most p whose bounding box intersects r; i.e., return $\{g \mid g \in G,\ g.b \cap r \neq \emptyset,\ g.p \leq p\}$;
- *search* $(r,\ p,\ PR(G))$:
  find and return the (unique) object g in G with $g.b = r$ and $g.p = p$;
- *insert* $(g,\ PR(G))$:
  insert object g into $PR(G)$, resulting in $PR(G \cup \{g\})$;
- *delete* $(r,\ p,\ PR(G))$:
  delete g from $PR(G)$, where $g.b = r$ and $g.p = p$, resulting in $PR\ (G \setminus \{g\})$.

In case the geometric objects are created by line generalization, we expect the number of stored objects to increase substantially with increasing priority number. The PR-file is designed to avoid multiple directory overhead by mixing references to blocks with different priority numbers in a carefully controlled way. In one single directory, references to blocks are organized dynamically in such a way that, simultaneously, references with smaller priority number are found earlier in the search, and also references are spatially clustered. Since these two goals conflict, we propose a mechanism that balances the two goals so as to provide high priority range query efficiency. We base the directory structure and the scheme for covering the data space with cells on the R-file [9], a good structure for range queries; certainly other choices, like R-trees [7] or R*-trees [2], are also possible. Let us motivate some of the design decisions and illustrate some of the basic features — by far not all — at the example of the city boundaries of Berlin (see Figure 1). Figure 3(a) shows the rectangles created from the Berlin boundary by forming objects of (close to) 6 consecutive points on a polygonal line and enclosing them in their bounding boxes, ignoring generalization indices. In Figure 3(b) and (c), objects have been built of (close to) 6 points of generalization index 1 and 2, respectively.

Now let us show in detail what happens when the rectangles of Figure 3(b) and (c) are inserted into an initially empty PR-file, in the order indicated in the figure by the rectangle identifiers. Assume that a data bucket can store at most 2 geometric objects, and a directory block stores at most 3 references, to bring out the effects fairly clearly for a small example.

The empty PR-file consists of a directory block A_0 with no reference. After rectangles 1 and 2 have been inserted, there is one directory block A_0 with a reference to a data bucket A_1 storing rectangles 1 and 2. Insertion of rectangle 3 leads to the creation of a new data bucket A_2 with priority number 2, since objects of different priorities are not stored in a common bucket. The reason is that queries with priority limit 1 should not retrieve objects of priority number 2, and that further these objects should not influence the organization of the directory for objects of priority number 1. References to blocks with different priorities, however, are stored in a common directory block, if references to blocks with higher priority number do not influence those with lower number. That is, references to blocks with higher priority number tend to be placed in blocks higher up in the directory tree, if free storage space is available. For the moment, this simply means that a reference to A_2 is entered in A_0. The geometric subspace of the data space directing operations from A_0 to A_1 and A_2, i.e., the cells of A_1 and A_2, is in both cases the data space itself (see Figure 8(a) in the Appendix).

(a) (b) (c)

Figure 3

Next, rectangle 4 is inserted to A_2. Upon insertion of rectangle 5, A_2 becomes overfull and is split in a balanced way into A_2 and B_2, as proposed by the R-file mechanism [9], see Figure 8(b). Rectangle 6 leads to a split of bucket A_1 into A_1 and B_1. Since four references cannot be stored in A_0, this in turn leads to a split of A_0, a directory split. Directory splits are the operations that balance priority and geometry in the directory. In our case, a directory block B_0 for the references to buckets of priority number 2 is created, these references are moved from A_0 to B_0, and a reference to B_0 is entered in A_0 (see Figure 8(c)).

Rectangle 7 is entered into B_1. Rectangle 8 with priority number 2 is inserted by following in A_0 the reference to the block with the smallest cell enclosing rectangle 8, that does not refer to a data bucket of smaller priority number than 2, and among all those the cell with highest priority number that is at most 2. That is, the search for the bucket for rectangle 8 proceeds from A_0 to B_0, and then to A_2, where rectangle 8 can be stored. Similarly, rectangle 9 is found to belong into A_2, making A_2 overfull. This leads to a split of A_2 into A_2 and its right half cell C_2, and to a reference to C_2 in B_0. Rectangle 10 is found to belong into bucket C_2, leading to a split of C_2 into C_2 and its lower half D_2. Now B_0 is overfull, since it needs to store four references, and is split into B_0 and C_0 (see Figure 8(d)). Hence, the restructuring due to the insertion of rectangle 10 takes place totally within the part of the directory for priority number 2.

Insertion of rectangle 11 with priority 1 makes B_1 overfull, resulting in a split of B_1 into B_1 and its lower half C_1. A_0 becomes overfull and is split into A_0 and D_0. Here, it is important to note that even though A_0 contains a reference to a block of higher priority number, the split of A_0 is balanced only with respect to the references to blocks of priority number 1. This serves to avoid any influence of higher number references on the organization of the directory at A_0. It would be incorrect to stop the restructuring at this point, since a search for A_0 a rectangle with priority number 2 in the right half of the data space would follow the reference to D_0, while the rectangle would actually be stored currently in a bucket reached through B_0 (see Figure 8(e)). Hence, we restructure the directory such that the rectangles in question are reached through D_0, by moving to D_0 the reference to the largest cell of priority number 2 enclosed in (or equal to) the cell of D_0 (see Figure 8(f)). Finally, rectangle 12 is correctly found to belong to bucket C_2. Its insertion leads to a split of C_2 into C_2 and E_2, both referenced from C_0.

4 Geometric operations with priority limits

Let us now define more precisely the PR-file storage structure and operations, in line with the basic idea described in the previous section. Out of several access structures that have been proposed and studied in the literature for maintaining a set of rectangles in the plane on secondary storage (see e.g. [2, 7, 9, 17]), the R-file [9] appears to perform well for geometric data; it unites several generally desirable properties and is still conceptually simple and easy to implement. Since spatial clustering of rectangles and simultaneously a priority ordering are needed to efficiently support priority range queries, we incorporate the underlying idea of the R-file into our proposed structure, the PR-file. If all rectangles to be stored have the same priority, the PR-file coincides with the R-file.

To simplify the discussion, but without compromising the generality of the approach, let us for the time being assume that all blocks (data buckets) and object descriptions are of fixed size, and several objects fit into a block. With each block B, a rectangular subspace of the data space, a *cell* $B.C$ is associated. For each object g stored in block B, rectangle $g.b$ is enclosed by (or equal to) $B.C$. Any two objects stored in the same data bucket B have the same priority number; the block is said to have this priority number, denoted by $B.p$. To be precise, an object is stored in the unique data bucket B with $B.p = g.p$ whose cell $B.C$ is the smallest one (among all present cells with priority $g.p$) enclosing $g.b$.

For the decision of whether a block B may be relevant for an operation, it is enough to know the geometry of its cell $B.C$ and its priority $B.p$. Hence, cell descriptions consisting of the cell geometry, the priority number and the reference to the block are to be maintained on secondary storage. Since the operations for cells resemble those on objects, and cells are special geometric objects, we maintain them similary, but with simultaneous attention to cell geometries and priorities. This leads to a hierarchy of blocks, the *PR-file directory tree*. For efficiency reasons, we keep the directory small, in a way similar to the R-file [9] and the BANG-file [4, 5], but unlike the R-tree [7] or the R*-tree [2]. To this end, each cell is created from the data space by repeated halving along the coordinate axes. For any two cells with the same priority number, either they don't intersect, or one is contained in the other; i.e., cells are *nested*. Dynamically, this is achieved by splitting the cell of an overfull block — the possible result of an insertion — into itself and a nested subcell. The latter is chosen so as to divide the number of entries between both new cells as evenly as possible — a *balancing* split.

Whereas an entry in a data bucket is just the description of a geometric object g, an entry in a directory block provides the necessary information to direct each operation to further blocks. A directory entry $d = (c, ref, p, bb)$ describes a cell boundary c, the reference *ref* to the block B detailing the cell, the priority number p of B, and the *bounding box* bb of all rectangles $g.b$, where g is an object stored in a block reached by following *ref* in the directory, i.e., g is stored in B or in a descendant of B. The cell boundary can be expressed by few bits, by applying any of several schemes for numbering recursively partitioned regions (see e.g. [4, 5, 8, 9]). Like the R-file and the BANG-file, the PR-file has a linearly growing directory. The priority number of a directory block B is defined as the minimum of $d.p$, taken over all entries d in B. That is, $B.p$ is the smallest priority number of any object stored in a data bucket reached through B. This information serves to quickly execute the priority range query: the query terminates locally as soon as the block priority number is higher than the query priority number. The bounding box bb serves to locally terminate the query whenever no more intersection of the query range with geometric objects reached through the current block is possible.

Block references in a directory block are not mixed arbitrarily. Instead, they are restricted to reflect the simultaneous consideration of spatial clustering and priority ordering, in the following way. Any directory block B with priority number $B.p$ contains references

- to directory blocks B' with $B.p = B'.p$ only, or
- to data buckets B' with $B.p \leq B'.p$ only, or
- to directory blocks B'' with $B''.p = i$ for some $i > B.p$, and to data buckets B' with $B.p \leq B'.p < i$.

Then, on any path in the directory tree from the root to a data bucket, blocks occur in order of increasing priority number. Furthermore, for any object g, the data bucket storing g is reached on the path from the root by following the reference $d.ref$ with highest priority number $d.p \leq g.p$ among those to the smallest cell $d.c$ enclosing $g.b$ within each block, disregarding the references to data buckets d' with priority number $d'.p < g.p$. Note that within a directory block B, the cells associated with different entries may be the same, provided that the priorities differ. This gives us the complete freedom of covering cell $B.C$ in any desired way for any priority. Certainly, for any priority p present at some entry of B, cell $B.C$ is covered by referenced cells of priority p. Let $B(d.ref)$ denote the block referenced by $d.ref$.

Given this structural PR-file invariant, let us now describe the operations of the PR-file that preserve this invariant in more detail. We express the operations in a very loose, Pascal oriented notation, to make them rather precise and intuitively understandable at the same time. The search for a geometric object with bounding box r and priority number p in a PR-file for a set G of objects whose root block is B is carried out by calling *search* $(r,\ p,\ B)$, where *search* is defined as follows:

search (*r: rectangle; p: priority number;*
    B: PR-file block)*: geometric object;*
  $B' :=$ *search block* $(r,\ p,\ B)$;
  if B' is a data bucket $\{\ g.p = p\ \}$ and there is an
    object g in B' with $g.b = r$ **then**
    return g
  else return "object not found"
end *search.*

Here, *search block* is defined as follows:

search block (*r: rectangle; p: priority number;*
    B: PR-fileblock)*: PR-file block;*
  if B is a directory block and there is an entry d in B
    with largest $d.p$ such that $d.p \leq p$ among those for
    which $d.c$ is the smallest cell enclosing r,
    disregarding entries which refer to data buckets B'
    with $B'.p < p$ **then**
    return *search block* $(r,\ p,\ B(d.ref))$
  else return B
end *search block.*

The most interesting operation, the priority range query in the PR-file with root block B for all geometric objects whose bounding box intersects the query range r and whose priority number is at most p, is carried out by calling *priority range query* $(r,\ p,\ B)$, where *priority range query* is defined as follows:

priority range query (*r: rectangle; p: priority number;*
    B: PR-file block)*:* **set of** *geometric object;*
  if B is a data bucket **then** $\{\ g.p \leq p\ \}$
    report all objects g in B with $g.b \cap r \neq \emptyset$
  else for each entry d in B with $d.p \leq p$ whose cell,
    restricted by $d.bb$, intersects r **do**
    report *priority range query* $(r,\ p,\ B(d.ref))$
end *priority range query.*

Here, **report** collects the overall answer to the query and returns it. Of course, in some cases the answer to a priority range query is just a filtering step; additional computation may be needed to decide whether a retrieved geometric object, e.g., a polygonal line enclosed in a bounding box, actually intersects the query range. This computation, however, is of no concern for the design of the PR-file.

To clarify the dynamics of the PR-file structure, let us explain in detail the insertion of an object. The empty PR-File is a root directory block B with no entries. A geometric object g is inserted into a PR-file with root block B by calling *insert* $(g,\ B)$, where *insert* is defined as follows:

insert (*g: geometric object; B: PR-file block*);
  $B' :=$ *search block* $(g.b,\ g.p,\ B)$;
  if B' is a data bucket **then**
    $\{\ g.p = B'.p$, so add g to the contents of $B'\ \}$
    $B' := B' \cup \{g\}$;
    if B' is overfull **then**
      $\{$ split B' into B' and B'' with $B''.C \subset B'.C\ \}$
      bucket split $(B',\ B'')$;
      $\{$ add a reference to B'' into the father of $B'\ \}$
      refer (*father* (B'), B'')
  else $\{$each of the blocks referenced from a directory
    block on the path from B to B' has priority
    number less than $g.p\ \}$
    create a new data bucket B'', and let $B''.p := g.p$
    and $B''.C := B'.C$;
    $B'' := \{g\}$;
    refer $(B',\ B'')$
end *insert.*

Since operation *search block* accesses the directory blocks on the path from B to B', any reasonable implementation should not require extra external storage accesses to retrieve the father of B' in the directory tree, denoted by *father* (B'). Note that the creation of a new bucket B'' with just one entry is not as bad as it might seem: it only happens if no object with priority number $g.p$ is enclosed in the cell of the directory block from which B'' is referenced. Further insertions of objects with priority $g.p$ will be able to utilize B'' instead of creating a new bucket.

The bucket split of B' into two buckets B' and B'' is carried out similar to the split proposed in [4, 5, 9], as follows. The cell $B'.C$ of B' remains unchanged, and the cell $B''.C$ of a new bucket B'' is created from $B'.C$ by repeated halving in alternating directions. The process stops with the cell $B''.C$ for which the number of objects g stored in B' before splitting whose bounding boxes are enclosed in $B''.C$ differs the least from the number of those whose bounding boxes are not; for details, see [9]. Especially, [9] explains how to treat the case that all rectangles intersect some selected split line, and hence a split, as simple as the proposed one, does not succeed. The priority number of the new bucket $B''.p$ is defined as $B'.p$.

A reference to block B' is added to the references in directory block B as follows:

refer (B, B': *PR-file block*);
enter in B an entry referencing B';
if B is overfull **then**
directory split (B)
end *refer.*

The directory split operation lies at the very heart of the dynamic adaptation of the PR-file. It is designed to support the dynamic distribution of block references with different priorities on different levels of the directory tree. Especially, references with high priority numbers should not increase the height of the directory for the lower priority numbers. To this end, references to be redistributed by a directory split may not only cause the directory split problem to propagate towards the root in the directory tree, but may also be distributed to descendants of the directory block to be split, thereby sometimes even avoiding the split in its entirety. In detail, the directory split operates as follows:

directory split (B: *PR-file block*);
if $d.p = B.p$ for each entry d in B **then**
block split (B, B');
try priority merge (B);
try priority merge (B');
add reference (B, B')
else if $d.p > B.p$ for exactly one entry d **then**
{ the cell of B is the same as the cell of $B(d.ref)$ }
let B'' denote the block $B(d.ref)$;
block split (B, B') disregarding d;
add reference (B, B');
if B'' is a directory block **then**
{ move to B' the entries d' to be referenced by B' }
pull up (B', B'');
try priority merge (B);
try priority merge (B');
else { B'' is a data bucket }
create a new data bucket B''', and let
$B'''.C = B'.C$ and $B'''.p = B''.p$;
move to B''' all geometric objects g in B'' with
$g.b$ enclosed by $B'''.C$;
enter in B' an entry referencing B'''
else { $d.p > B.p$ for more than one entry d in B }
$d :=$ the (unique) entry in B with largest cell $d.c$
among all entries with largest priority number in B;
{ $d.c = B.C$ }
$p' := d'.p$, where $d'.p$ is largest among all entries
in B except d;
if $d.ref$ refers to a data bucket **then**
{ all $d.ref$ in B refer to data buckets }
create new directory block B', and let
$B'.C := B.C$ and $B'.p := p'$;
move all entries d' in B with $d'.p \geq p'$
from B to B';
enter in B an entry referencing B'
else { $d.ref$ refers to a directory block }
move all entries d' in B except d with $d'.p = p'$
from B to $B(d.ref)$;
if $d.p > p'$ **then**
{ d is the only directory block reference in B }
$d.p := p'$
end *directory split.*

A *block split* of B into blocks B and B', initiated within a directory split, creates a cell $B'.C$ such that the references of B are distributed most evenly between B and B' (see also the split operation for data buckets and the directory split in [9]). Such a split may disregard some of the references of B for determining $B'.C$; in this case, the disregarded references are inserted into B or B' after the split, as appropriate. If moving entries into some block B' makes B' overfull, then B' is split, like in an insert operation. We *pull up* entries from B'' — and for rare cases, similar to the forced split in the BANG-file [4, 5] or the R-file [9], from descendants on one path from B'' towards a leaf of the directory tree — as follows:

pull up (B, B': *PR-file block*);
move all entries d of B' with $d.c$ enclosed in $B.C$
from B' to B;
if $B.C$ is not yet covered and B' contains a reference
$d'.ref$ to a directory block such that $d'.c$ is the
smallest cell in B' enclosing $B.C$ **then**
pull up (B, $B(d'.ref)$)
end *pull up.*

After a geometric split, we try to avoid sparse directory blocks by priority merging as follows:

try priority merge (B: *PR-file block*);
if $d'.p > B.p$ for exactly one entry d' in B and
$d'.ref$ refers to a directory block B' **then**
if all references in B' fit into B **then**
{ priority merge possible }
remove $d'.ref$ from B;
move all entries from B' to B;
dispose (B');
try priority merge (B)
else *try priority merge* (B')
end *try prioritymerge.*

Dispose (B) returns the storage space for block B to the file handling system, i.e., frees the space for B for reuse. Adding a reference to a block created by a split into the directory tree is just a slight extension of the *refer* operation:

add reference (B, B': *PR-file block*);
 if *father* (B) exists **then**
  refer (*father* (B), B')
 else create new root directory block B'';
  refer (B'', B);
  refer (B'', B')
end *add reference*.

5 Performance evaluation

In this section, we compare the performance of the PR-file with that of a single R-file for objects of all priorities, and with a collection of different R-files, one for each priority. To this end we have implemented the R-file and the PR-file in Lightspeed Pascal on a Macintosh IIcx. Both implementations support insertions, exact match and range queries. We have evaluated the performance of the three methods for real world data from a geographic database representing polygonal boundaries of the communities of Baden-Württemberg, a German province. We have generalized the data for six different scales, by applying a well established line simplification technique [3]; an object is present in the map of a certain scale only if justified by its geometric size. The set of 36520 data objects is partitioned into subsets according to priorities 1 through 6 of 391 objects, 2264 objects, 3399 objects, 4630 objects, 9285 objects and 16551 objects, respectively. The distribution of object numbers over the priorities conforms with cartographer's experience, formulated e.g. in the 'radical law'[19].

The size of data buckets is chosen such that 8 objects fit in one bucket. To observe the behavior of sizable directory trees, we chose fairly small directory blocks, with room for 24 references per block. During a sequence of insertions into the initially empty PR-file, we counted external storage accesses, the number of buckets, and the number of directory blocks. To make the access structure efficiency clear, we only used internal storage space for one bucket and one directory block; hence, no buffering of blocks was possible.

This performance evaluation revealed that bucket space utilization was almost the same, namely between 69% and 70% for all three methods; directory block utilization was best for the single R-file (69%) and worst for the PR-file (63.7%). Since there are far more buckets than directory blocks, total space utilization was best for the single R-file (69.6%) and worst for the PR-file (68.8%), although with a negligible difference. The dynamic adaption of the access structure to the changing set of objects is cheapest for the collection of R-files and almost as cheap for the single R-file, but considerably more expensive for the PR-file, namely 5.66, 5.82 and 8.07 external accesses (read and write) per inserted object, respectively.

In return, the PR-file supports proximity queries with a given priority limit, such as range queries and zoom queries. We measured the number of external read operations to answer range queries and zoom-in queries with square ranges of different areas and different priority limits. Instead of carrying out a limited number of queries at random positions, we statistically calculated the average number of accesses over all possible query range positions in the entire data space, for all possible priority limits.

Figure 4 shows the number of external accesses (on the ordinate) for four different range query areas (as dashed horizontal lines for a single R-file, dotted curves for the collection of R-files, and solid curves for the PR-file), varying from 0.01% to 0.5% of the data space area, and for all six priority limits (on the abscissa). Figure 5 shows the same for three more range query areas, from 1% to 4%; the curve for the collection of R-files almost coincides with that of the PR-file and is therefore omitted. Any range query with area greater than 0.1% that limits the level of detail at all can be answered far faster in the PR-file than in the single R-file. Only for very small range size or without any priority limit, the single R-file outperforms the PR-file slightly.

It turns out that data bucket regions are almost identical for the collection of R-files and the PR-file. This is not surprising, because the PR-file associates objects with the same bucket as the respective R-file in the collection, except for the rare case (less than 0.1% of all cases in our experiments) of a PR-file forced split. Since therefore the number of data bucket accesses is almost the same, the difference lies in the directory block accesses.

Figure 4

Figure 5

Figures 6 and 7 show for the collection of R-files (dotted) and the PR-file (solid) the additional number of directory block accesses for our seven different range query areas needed to relax the priority limit by one level to the value at the abscissa, where the query range is unchanged. For instance, a range query of size 0.5% needs roughly four additional directory block accesses to get from priority limit four to priority limit five in a PR-file, whereas it needs between six and seven in a collection of R-files. That is, the PR-file clearly outperforms the collection of R-files in all cases except priority limit one.

Figure 6

Figure 7

6 Conclusion

We have proposed an access technique for geometric databases that efficiently supports proximity queries (such as range, nearest neighbor, and zoom queries) with priority limit. The proposed structure, the PR-file, is suitable for storing maps independent of a scale, and extracting an arbitrary area of a map at any desired scale with the appropriate amount of detail. As a prerequisite, we have shown how geometric objects can be partitioned and reassembled according to priorities of defining points. Retrieval efficiency in the PR-file depends mainly on the number of geometric objects present on the extracted map, not on those outside the map area or on those that are in the map area, but are irrelevant because of the limited detail in the map. This is true even for incremental retrieval, as in zoom operations, where either the level of detail or the area of the query increases. The PR-file is the first fully adaptive access structure that supports the combination of range and priority queries for non-zero size geometric objects efficiently. It appears to be quite a promising first step: Our performance evaluation with geographic data shows that the PR-file speeds up priority range queries by a high factor and not just a few percent, as compared with a structure for non-zero size geometric objects that does not take priorities into account. We expect that PR-file variants can be tuned to specific applications; e.g., the partitioning strategy for geometric objects according to priorities and the PR-file structure have been defined independent of each other in this paper, but could certainly take advantage of each other in a PR-file for cartography.

Acknowledgement

We wish to thank Thomas Ohler for valuable discussions and suggestions, Oliver Günther for helpful comments, and Trudi Halboth and Gabriele Reich for typesetting the paper and assisting with the figures.

References

[1] B. Becker, P. Widmayer: Spatial priority search: an access technique for scaleless maps, technical report No. 24, Institut für Informatik, Universität Freiburg, 1990.

[2] N. Beckmann, H.P. Kriegel, R. Schneider, B. Seeger: The R*-tree: An Efficient and Robust Access Method for Points and Rectangles, ACM SIGMOD International Conference on the Management of Data, 1990, 322–331.

[3] D.H. Douglas, T.K. Peucker: Algorithms for the reduction of the number of points required to represent a digitized line or its caricature, The Canadian Cartographer, Vol. 10, No. 2, 1973, 112–122.

[4] M. Freeston: The BANG file: a new kind of grid file, Proc. ACM SIGMOD International Conference on the Management of Data, 1987, 260–269.

[5] M. Freeston: Advances in the design of the BANG file, Proceedings 3rd International Conference Foundations of Data Organization, Paris, 1989, 322–338.

[6] S.C. Guptill: Speculations on seamless, scaleless cartographic data bases, Proc. 9^{th} International Symposium on Computer-Assisted Cartography, 1989.

[7] A. Guttman: R-trees: A dynamic index structure for spatial searching, Proc. ACM SIGMOD International Conference on the Management of Data, 1984, 47–57.

[8] A. Hutflesz, H.-W. Six, P. Widmayer: Globally order preserving multidimensional linear hashing, Proc. IEEE 4^{th} International Conference on Data Engineering, 1988, 572–579.

[9] A. Hutflesz, H.-W. Six, P. Widmayer: The R-file: An efficient access structure for proximity queries, Proc. IEEE 6^{th} International Conference on Data Engineering, 1990, 372–379.

[10] D.M. Mark: Conceptual basis for geographic line generalization, Proc. 9^{th} International Symposium on Computer-Assisted Cartography, 1989, 68–77.

[11] R.B. McMaster: The integration of simplification and smoothing algorithms in line generalization, Cartographica, Vol. 26, 1989, 101–121.

[12] J.-C. Muller: Rule based generalization: Potentials and impediments, 4^{th} International Symposium on Spatial Data Handling, 1990, 317–334.

[13] W.-D. Rase: Tools for geometric data acquisition and maintenance, NATO Advanced Research Workshop "Mapping and Spatial Modelling for Navigation", Denmark, August 1989.

[14] J. Robergé: A data reduction algorithm for planar curves, Computer Vision, Graphics, and Image Processing, Vol. 29, 1985, 168–195.

[15] H.-J. Schek, W. Waterfeld: A Database Kernel System for Geoscientific Applications, Proc. 2^{nd} International Symposium on Spatial Data Handling, Seattle, 1986, 273–288.

[16] K.S. Shea, R.B. McMaster: Cartographic generalization in a digital environment: When and how to generalize, Proc. 9^{th} International Symposium on Computer-Assisted Cartography, 1989, 56–67.

[17] H.-W. Six, P. Widmayer: Spatial searching in geometric databases, Proc. IEEE 4^{th} International Conference on Data Engineering, 1988, 496–503.

[18] K. Thapa: Data compression and critical points detection using normalized symmetric scattered matrix, Proc. 9^{th} International Symposium on Computer-Assisted Cartography, 1989, 78–89.

[19] F. Töpfer, W. Pillewizer: The principles of selection, a means of cartographic generalisation, Cartographic Journal, Vol. 3, No. 1, 1966, 10–16.

[20] P. v. Oosterom: The Reactive-tree: A Storage Structure for a Seamless, Scaleless Geographic Database, submitted for publication.

Appendix