Visual Database Systems 4 (VDB4), Chapman & Hall, London, 1998, pp. 147-153.

Visualizing and Animating R-trees and Spatial Operations in Spatial Databases on the Worldwide Web[*]

František Brabec and Hanan Samet
Computer Science Department, Center for Automation Research, and Institute for Advanced Computer Studies, University of Maryland
College Park, Maryland 20742, USA, +1-301-405-1755,
{brabec,hjs}@cs.umd.edu

Abstract

The rationale behind the design of a JAVA^{TM} spatial index applet is described that enables users on the worldwide web to experiment with different variants of the R-tree. The applet is part of the VASCO system that contains JAVA^{TM} applets for a large set of hierarchical spatial data structures. The R-trees can be built using different splitting rules and include the R*-tree. The applet enables users to see in an animated manner how a number of basic spatial database search operations are executed for them. The spatial operations are spatial selection (i.e., a window or spatial range query) and a nearest neighbor query that enables ranking spatial objects in the order of their distance from a given query object. The results of the different splitting rules and the algorithms are visualized and animated in a consistent manner using the same primitives and colors so that the differences between the effects of the rules can be easily understood. The applet can be used to monitor the performance of spatial databases that use an R-tree spatial index as well as tune them by observing their behavior for different distributions of data. The applet can be found at `http://www.cs.umd.edu/~hjs/rtrees/index.html`.

Keywords

R-trees, spatial databases, worldwide web, incremental nearest neighbor finding, window queries, ranking

1 INTRODUCTION

The visualization of data in database systems is an important issue (e.g., Spaccapietra & Jain (1995)). Usually the volume of the data is so large that it is impractical to look at it all at once. This has led to the development of data reduction methods

[*]This work was supported in part by the National Science Foundation under Grant IRI-9712715.

that enable users to observe effects such as clustering, etc. (e.g., Keim & Kriegel (1995)). In other cases, users would like to monitor the behavior of the database system when certain key operations are applied to it. In this case, the underlying representation of the data also plays a significant role in what we are observing. In particular, we do not necessarily need to visualize the data as much as we need to visualize the interaction between the representation and the operations. In this paper we focus on the visualization of operations on data in spatial databases. We have chosen an animated approach (see also, for example, Brown & Sedgewick (1984)) that is accessible on the worldwide web. This naturally leads us to also deal with the visualization of the representation of the underlying data as this has a strong impact on the execution efficiency of the operations.

The representation of spatial data is an important issue in spatial databases (Güting 1994, Scholl & Voisard 1997). It involves the selection of an appropriate decomposition of the underlying space that contains the spatial objects as well as an access structure (e.g., a sequential list, array, tree, etc.), known as a *spatial index*, to facilitate finding the objects. Spatial indexes usually provide a quick way to access the objects given a specific location or set of locations. Spatial database systems make use of a number of different space decompositions and access structures. For example, ORACLE (Herring 1996, Oracle Corporation 1996) makes use of a quadtree-block decomposition (e.g., Samet (1990*a*) and Samet (1990*b*)) and a B-tree access structure (Comer 1979). The result is known as *SDO* denoting *Spatial Data Option*. INFORMIX (Stonebraker 1997) makes use of data blades that combine a hierarchy of minimum bounding boxes (often known as an R-tree (Guttman 1984)) with a B-tree access structure. ESRI makes use of a two-level grid (as well as a three-level grid) which is a variant of a grid file (Nievergelt, Hinterberger & Sevcik 1984) that uses a regular decomposition partition. The result is known as *SDE* denoting *Spatial Data Engine*.

Although we have mentioned a number of general representations above, actual implementations make use of variants of these representations. The relationship between these variants is not immediately clear, although a well-trained computer scientist can consult the original references or books (e.g., Samet (1990*a*) and Samet (1990*b*)). Often, this form of consultation requires an effort that is beyond what users and database designers are willing to expend. In addition, it may not provide the intuition that is desired. What is really needed is a hands-on experience with the ability to vary the data, assuming a rather limited volume of data so as to make the task manageable.

In this paper we describe the rationale behind the design of an R-tree JAVA^{TM} (e.g., Arnold & Gosling (1996)) applet that enables users on the worldwide web to experiment with a number of variants of the R-tree and see in an animated manner how a number of basic spatial database search operations are executed for them. The R-tree is based on a hierarchy of aggregations of minimum bounding boxes for the spatial objects (e.g., an R-tree). What differentiates the R-tree variants are the competing methods used for aggregating the bounding boxes and for dealing with the situation that an aggregate is too full (i.e., overflow). Our applet can be used by the database designer to tune the database to enhance its performance by observing its behavior for different distributions of data. This applet is a small part of a much more general system called VASCO (denoting *Visualizing and Animating Spatial Constructs and Operations*) under continuous development that contains JAVA$^{\text{TM}}$ applets for a much larger set of hierarchical spatial data structures including a number of quadtree and k-d tree variants. Due to the limitations of the display, our examples are restricted to a two-dimensional domain where the objects can be points, lines, and rectangles, although our example data objects are restricted to rectangles. Extending the system to handle points and lines is simple as we just use their minimum bounding boxes.

In addition to seeing how the underlying space is decomposed through the ability to insert data in an incremental manner, we also enable users of the applet to delete data so that the effect of any sequence of arbitrary insertion and deletion operations can be understood. Most importantly, users of the applet can see how the R-tree supports some common database search operations. In particular, we show in an animated manner how spatial selection (Aref & Samet 1991) is executed for an arbitrary rectangular search region (also known as a window operation or a spatial range query), and how nearest neighbors are found. The animations are performed in a consistent manner through the use of the same user interface and colors.

Both the spatial selection and the nearest neighbor operations are executed in an incremental manner which means that the data objects that satisfy the query are returned and displayed one-by-one. The nearest neighbor algorithm is particularly noteworthy as when the algorithm is run to completion, it provides a full ranking of the data objects in terms of their distance from the query point (Hjaltason & Samet 1995). This is in contrast to an alternative approach that makes use of a K-nearest neighbor algorithm (Roussopoulos, Kelley & Vincent 1995). In particular, once we have computed the k nearest neighbors, in order to obtain the $k+1^{st}$ neighbor, using the incremental approach (Hjaltason & Samet 1995) we only need to resume the algorithm at the point at which it was right after having obtained the k^{th} neighbor, whereas using the K-nearest neighbor approach (Roussopoulos et al. 1995) we must restart the algorithm to compute the $k+1$ nearest neighbors.

The rest of this paper is organized as follows. Section 2 contains a brief overview of the R-tree spatial data structure as this is the focus of our presentation. Section 3 discusses what we are displaying and how it is being displayed. This is particularly oriented towards presenting the mechanics of the display process and includes a rationale for the format of the applet window. Sections 4 and 5 deal with the mechanics of the actual operation of the applet. In particular, Section 4 explains how the data structures are populated and modified, while Section 5 describes the algorithms that we are visualizing and how they are animated to maximize comprehension. Section 6 contains concluding remarks and directions for future research. For more details about the functionality of the applet and how to use it, see Brabec & Samet (1998).

Visual Database Systems 4 (VDB4), Chapman & Hall, London, 1998, pp. 147-153.

2 OVERVIEW OF R-TREES

There are two principal methods of representing spatial data. The first is based on a decomposition of the underlying space into disjoint blocks so that a subset of the objects are associated with each block. This subset is often defined by placing a bound on the number of objects that can be associated with each block (termed a *stopping condition*). The drawback of this disjoint method is that when the objects have extent (e.g., lines, rectangles, as well as any other non-point objects), then an object may be associated with more than one block. We do not discuss this method further in this paper (for more details, see (Samet 1990*a*, Samet 1990*b*)). The alternative is to use an object hierarchy that aggregates objects into groups based on proximity and then use proximity to further aggregate the groups. The drawback of this method is that it results in a non-disjoint decomposition of space. This means that if a search fails to find an object in one path starting at the root, then it is not necessarily the case that the object will not be found in another path starting at the root.

The R-tree (Guttman 1984) is an object hierarchy which is applicable to arbitrary spatial objects. The hierarchy is formed by aggregating the minimum bounding boxes of the spatial objects and storing the aggregates in a tree structure. The aggregation is based, in part, on proximity of the objects or bounding boxes. The number of objects or bounding boxes that are aggregated in each node is permitted to range between $m \leq \lceil M/2 \rceil$ and M (termed *bucket size parameters*), thereby leading us to use the prefix (m, M) to characterize a particular R-tree. The root node in an R-tree has at least two entries unless it is a leaf node in which case it has just one entry corresponding to the bounding box of an object. In the rest of this section we review the different variants of the R-trees that are supported by our applet.

R-trees can be constructed in either a dynamic or a static manner. The dynamic methods build the R-tree as the objects are encountered while the static methods wait until all the objects have been input before building the tree. The results of the static methods are usually characterized as being *packed* since knowing all of the data in advance permits each R-tree node to be filled to its capacity.

There are two principal methods of determining how to fill each R-tree node. The most natural method is to take the space occupied by the objects into account when deciding which ones to aggregate. An alternative is to order the objects prior to performing the aggregation. However, in this case, once an order has been established, we don't really have a choice as to which objects (or bounding boxes) are being aggregated. The most obvious order, although not particularly interesting or useful, is one that preserves the order in which the objects were initially encountered (i.e., objects in aggregate i have been encountered before those in aggregate $i + 1$). We do not use this technique in our applet.

The more common orders are based on proximity of representative points for the objects (e.g., their centroids). The packed R-tree (Roussopoulos & Leifker 1985), Hilbert packed R-tree (Kamel & Faloutsos 1993), and Morton packed R-tree (e.g., Kamel & Faloutsos (1993)) are examples of these methods. The principal difference between them is that the packed R-tree uses a two-stage ordering process. The first stage orders the centroids of the objects in Peano-Hilbert order. The second stage, fills the leaf nodes by examining the objects in increasing Peano-Hilbert order of their centroids (obtained by the first stage). Each leaf node is filled with the first unprocessed object and its $M-1$ nearest neighbors (in the space in which the objects lie) which have not yet been inserted in other leaf nodes. Thus in the packed R-tree each node is filled with objects according to the proximity of their centroids in the space in which they lie, while in the Hilbert and Morton packed R-trees each node is filled with objects according to the proximity of the positions of their centroids in the ordering (i.e., their positions in the Peano-Hilbert or Morton orders are close).

The dynamic methods differ primarily in the techniques used to split an overflowing node p during insertion. The first goal is to minimize the number of children of node p that must be visited by search operations. This goal is accomplished by minimizing the area common to the children of p (termed *overlap*). The second goal is to reduce the likelihood that each node q is visited by the search. This is accomplished by minimizing the total area spanned by the bounding box of q (termed *coverage*). The goal is to minimize coverage and overlap. These goals are at times contradictory and thus heuristics are often used.

The above dynamic methods range from being quite simple (e.g., exhaustive search (Guttman 1984)) to being fairly complicated (e.g., R*-tree (Beckmann, Kriegel, Schneider & Seeger 1990)). Most just split the overflowing node, while others try to reinsert some of the objects and nodes from the overflowing nodes (Beckmann et al. 1990) thereby striving for better overall behavior (e.g., reduction in coverage and overlap). Methods of intermediate complexity range from taking linear time (e.g., Ang & Tan (1997) and (Guttman 1984)) to taking quadratic time (Guttman 1984).

Other dynamic methods also make use of the ordering applied to the bounding boxes (using their centroids in our examples). These methods are characterized as *non-packed*. The Morton non-packed R-tree (White 1982) and the Hilbert non-packed R-tree (Kamel & Faloutsos 1994) are examples of these methods. In this case, the result is equivalent to a B$^+$-tree (Comer 1979). All update algorithms are B$^+$-tree algorithms and do not make use of the spatial extent of the bounding boxes to determine how to split a node. Thus the goals of minimizing overlap or coverage are not part of the node splitting process although, as we will see, this does not preclude these methods from having good behavior with respect to these goals. For more details on how these splitting methods are implemented in our applet, see Brabec & Samet (1998).

3 MECHANICS OF THE DISPLAY PROCESS

We explain the display process by referring to Figure 1 which shows a sample applet window. The applet window contains a drawing canvas and a control panel. In Figure 1, the drawing canvas is deliberately left empty and is overlaid with a window listing the various splitting methods. The window is implemented as a JAVA$^{\text{TM}}$ check box group plus a close button. The features in the control panel of the drawing canvas, as well as the rationale for their inclusion, are explained in this and the remaining sections. In the interest of conserving space, some of the remaining figures that correspond to examples of an applet window omit the control panel.

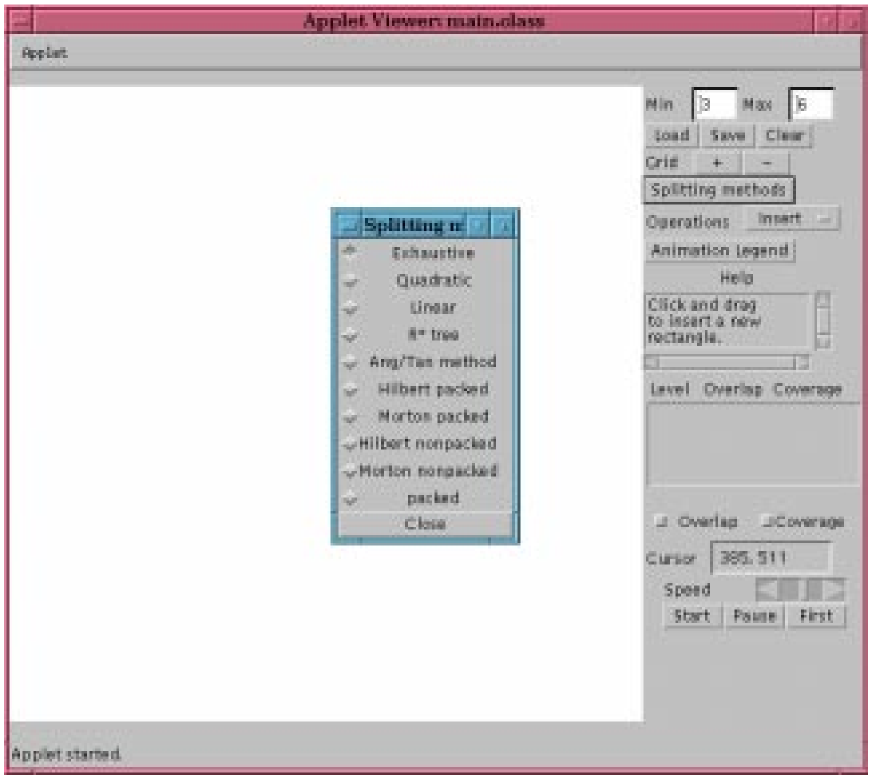


Figure 1: A sample applet window. The drawing canvas is deliberately left empty. It is overlaid with a window containing a check box group listing the various splitting methods and a close button.

Our goal is to treat the various operations in a consistent manner. An important issue is how to display the hierarchy inherent to the representation. One problem is that although the R-tree makes use of a tree-like access structure, it is quite cumbersome to draw. However, for representations based on aggregations of objects such as the R-tree, which consist of a hierarchy of minimum bounding boxes, the intermediate levels of decomposition do provide useful information. In particular, the amount of overlap between different minimum bounding boxes at the same level and the extent of the coverage of the underlying space play an important role in the evaluation of different aggregation and node splitting policies.

In this case, we provide a mechanism under the user's control for viewing the aggregations of bounding boxes at different levels of decomposition one level at a time, as well as combinations of several levels at a time. We also provide information about the coverage and overlap at the different levels in terms of ratios where values of 1 are ideal. The decompositions at the various levels are displayed using a range of different colors so that their interaction can be seen (see Brinkhoff, Kriegel, Schneider & Seeger (1994) for a visualization of an R*-tree and Kang & Li (1995) for a related approach to visualizing a related representation known as an R$^+$-tree (Sellis, Roussopoulos & Faloutsos 1987)). A particular level of aggregation is shown in the same color regardless of its depth in the tree. For example, the result of the first level of aggregation is shown in green regardless of whether there are two or twenty levels of aggregation present. Figures 2a and 2b show the aggregations of bounding boxes at two different levels of decomposition for the same data set using an R*-tree where Figure 2a corresponds to the first level of aggregation while Figure 2b corresponds to the second level of aggregation.

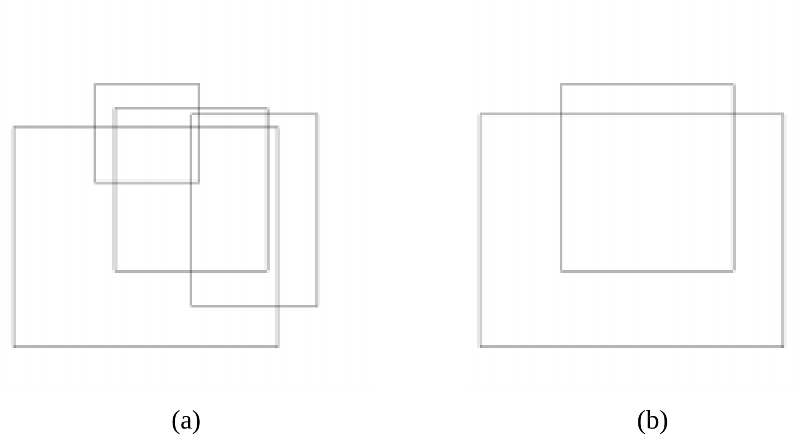


Figure 2: Snapshot of an R-tree applet drawing canvas showing the effect of the (a) first and (b) second levels of aggregation for the same data set when using an R*-tree.

Visible levels are displayed in the control panel by a JAVA^{TM} list element that can also optionally indicate overlap and coverage. We use the term `Visible Levels` to refer to this element. There is one line in the `Visible Levels` list for each level in the R-tree where the first row (i.e., the top row) in `Visible Levels` corresponds to the root of the R-tree. The levels which are to be visible are set by moving the mouse over them and clicking in a toggling manner. When the boundaries of regions at different levels of aggregation coincide, then the level which was most recently selected for display is the one whose color dominates. This is similar to a “back-to-front” painter‘s algorithm (e.g., Foley, van Dam, Feiner & Hughes (1990)) in computer graphics for deciding what is visible. Initially, as the rectangles are inserted into an empty drawing canvas, the data rectangles are displayed in red and no aggregation is shown. In other words, only the deepest level of R-tree is visible initially.

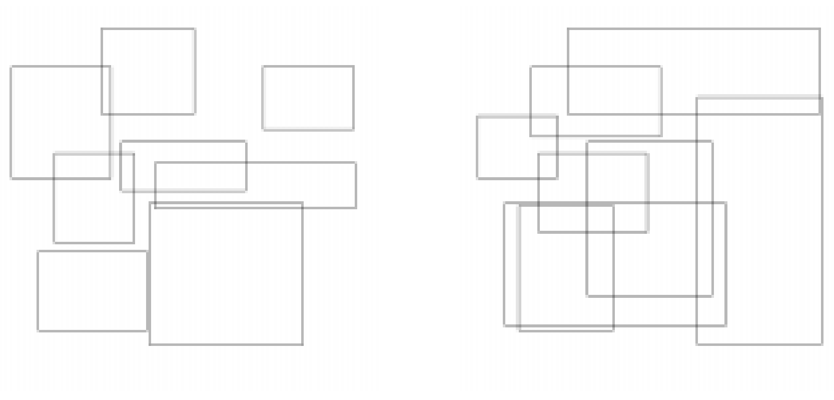
The `Visible Levels` list is also used to display coverage and overlap information when the user deems it appropriate by selecting the `Overlap` and `Coverage` elements which are implemented as JAVA$^{\text{TM}}$ check boxes. The rationale for these check boxes is that for very large data sets overlap and coverage may take time to compute (using plane-sweep methods from computational geometry (Preparata & Shamos 1985)) and thus they do not appear automatically. In particular, we did not feel that users would always wish to wait for its computation as they are switching between the levels. Thus clicking on one or both of these check boxes initiates the computation and display update of these values for all visible levels until these check boxes are clicked again at which time the action is de-selected.

The drawing canvas permits users to see the effects of the different node splitting methods as well as the effects of varying the bucket size parameters m and M which are part of the characterization of the R-tree. Once the splitting method has been selected, objects can be inserted and deleted at will. At any time, users may switch between the different splitting methods as well as change the bucket size parameters. The latter is achieved by implementing the bucket size parameters using a JAVA$^{\text{TM}}$ text field widget.

Since the amount of space on the control panel of the applet window is limited, we do not have enough room to list all of the splitting methods. Thus we make use of a `Splitting Methods` button which, when activated, creates a window that consists of a check box group of actual splitting method choices plus a close button (see Figure 1). The effect is similar to a menu which has been torn off thereby becoming a tear-off menu with radio buttons (as only one splitting method can be chosen). This enables users to switch between different splitting methods by just moving the mouse and selecting the desired rule rather than requiring three mouse actions (i.e., a click, a drag while depressed, and a release) as would be the case if we do not provide a button to create a window and instead use a JAVA$^{\text{TM}}$ choice element.

The ability to easily switch between splitting methods provides a very useful tool for their evaluation. For example, Figures 3a and 3b show the effect of the first level of aggregation for a given data set when using an R*-tree and linear splitting rule, respectively. Notice the superior performance of the R*-tree splitting rule in the sense that it reduces the amount of overlap while the coverage stays about the same.

From an implementation perspective (rather than from a user interface perspective), there are two ways to support switching between different splitting methods. The simplest way is to keep a separate R-tree corresponding to the result of each splitting method. This means that each time we insert or delete a rectangle, all R-trees must be updated. This requires quite a bit of storage and is time consuming especially if we never look at the result of using a particular splitting method. The alternative is to rebuild the R-tree when switching to another splitting method. Unfortunately, this is not so straightforward as some of the splitting methods (i.e., most of the dynamic ones) are sensitive to the order in which the spatial objects were inserted



(a) (b)

Figure 3: Snapshot of an R-tree applet drawing canvas showing the effect of the first level of aggregation for a given data set when using an (a) R*-tree and (b) linear splitting rules.

into the R-tree. This creates a potential problem when we switch between different splitting methods as the tree structure is destroyed. This is resolved by keeping a separate representation R of all objects. R is ordered according to the time t at which the object o was created and thereby inserted into the data structure that was being displayed at t. In addition, when an object o is deleted from the data structure that is being displayed, o is inserted again into R and marked as being deleted. This is important when the tree is rebuilt for the corresponding splitting method. Thus we see that an object may appear more than once in R. Keeping such an ordering ensures predictable and consistent results as we switch between splitting methods.

In order to visualize the methods that make use of an ordering of the objects, (i.e., packed R-tree, Morton packed R-tree, Hilbert packed R-tree, Morton non-packed R-tree, and Hilbert non-packed R-tree), we also display the locations of their centroids using a dot as well as their positions in the ordering. This is useful in enabling the user to see the sensitivity of these methods to the actual location of the centroids of the objects. The locations of the centroids are displayed using a red dot while their positions in the ordering are also displayed in red. The centroids and their positions in the ordering are removed when we switch to a splitting method that does not make use of an ordering (i.e., the exhaustive, linear, quadratic, R*-tree, and Ang/Tan splitting methods).

4 DATA STRUCTURE POPULATION AND MODIFICATION

In order to be able to visualize the decomposition induced by the data structures as well as the associated algorithms, we need a way to populate and modify them which means that we must make it easy to insert and delete spatial objects. Rectangles are inserted by selecting the `Insert` operation in the `Operations` choice widget (the JAVA^{TM} analog of a menu with a selection mechanism analogous to radio-buttons) which places the user in insertion mode. The actual specification of the rectangle is accomplished by depressing the mouse at a particular point, which serves as one corner, and then dragging the mouse (while depressed) to the point which is to serve as the diagonally opposite corner, at which time the mouse is released. As soon as an object is inserted, the visible levels are updated, and likewise when an object is deleted. There are also provisions for clearing the applet, loading the applet with data from a file, merging the data in the applet with data from a file, saving the data in the applet (not the R-tree; just the data objects) to a file, and adding the data in the applet (again, just the data objects) to a file (security considerations notwithstanding).

Rectangles are deleted by selecting the `Delete` operation in the `Operations` choice element which places the user in deletion mode. The actual specification of the rectangle to be deleted is done by positioning the mouse somewhere within the rectangle that we wish to delete. This rectangle is determined by invoking a nearest neighbor algorithm and it is highlighted by flashing its border. The actual deletion occurs when the mouse button is depressed. If the mouse position p is in more than one rectangle (as may be the case when the rectangles overlap), then the containing rectangle r whose boundary is the closest is deleted. Note that r is not necessarily the rectangle t whose boundary is the closest to p. In particular, although t's boundary may be closer to p, we ignore t unless it also contains p. Observe that if p is in a rectangle that is part of a nested set of rectangles, then this rule results in deleting p's minimum enclosing rectangle. If p is not within any rectangle, then the rectangle whose boundary is the closest is deleted.

The insertion and deletion processes are facilitated by the `Cursor` text field element which indicates the position of the mouse (see Figure 1). In addition, a pair of `Grid` control panel buttons labeled '$+$' and '$-$' can be used to impose a successively finer and looser, respectively, grid on the drawing canvas. This is useful when we want to ascertain the exact positions of objects. Initially no grid is present. Each time the '$+$' button is depressed, the level of resolution is doubled. Each time the '$-$' button is depressed, the level of resolution is halved.

5 ALGORITHM VISUALIZATION AND ANIMATION

There are two modes of animation: incremental and continuous. The difference is that in the incremental mode, the animation halts after each object is found that satisfies the query, while in the continuous mode no halts are made, unless of course, the user presses the `Stop` or `Pause` buttons. Regardless of the mode, users can temporarily suspend the animation or stop it.

We have tried to use the same color conventions in our visualization of the animation for distinguishing between the data, query objects, and the intermediate results of the search operations. Although our objects in this applet are rectangles, we have tried to make the presentation as general as possible by framing it in terms of objects unless the topic was related specifically to rectangles. In order to understand the motivation for our choices, we now give a brief explanation of the algorithms that we use to implement the search operations. We first explain the incremental nearest neighbor algorithm which locates the objects in the database in the order of their distance from the query object q (Hjaltason & Samet 1995). Next, we explain the window query algorithm.

The incremental nearest neighbor algorithm makes use of a priority queue where the queue elements are the bounding boxes of the underlying data structure as well as the spatial objects themselves. The priority queue is ordered on the basis of the distance of its elements from the location of the query object which is a point in our implementation. The distance from a query point q to a spatial object o is the distance from q to the nearest point p in o. If q lies in o, then the distance from q to o is zero. In case two spatial objects are at the same distance from the query object, then a natural course of action is to pick one of them at random. This is how the original formulation of the algorithm was implemented (Hjaltason & Samet 1995). This approach was fine as long as the spatial objects do not have a nonzero area (e.g., points and lines).

However, when the spatial objects have a nonzero area (e.g., rectangles), such an approach can result in problems when the spatial objects are not disjoint (i.e., the rectangles overlap or are nested) and q is contained in several objects. In this case, we need a better tie-breaking rule to choose the nearest point. A good suggestion is to use a rule that orders the objects containing q by the minimum distance between a point on their boundary and q. Recall that this is the same rule that was used to determine the nearest containing object when deleting a rectangle spatial object given a particular point which was contained in more than one spatial object. Thus we see that we have modified the original incremental nearest neighbor algorithm to make use of two keys in the ordering:

1. The primary key is the distance from q to the nearest point p in o.
2. If $p = q$ (i.e., o contains p), then the secondary key is the distance from q to the nearest point r on the boundary of the nearest object s that contains q.

In fact, with this modification, we are able to implement the deletion algorithm with the incremental nearest neighbor algorithm.

The incremental nearest neighbor algorithm works in a top-down manner in the sense that as elements are removed from the queue, they are checked to see if they correspond to bounding boxes that are not at the lowest level of the hierarchy (i.e., nonleaf nodes). If this is the case, then their immediate descendants (i.e., the children) are inserted in the queue ordered according to their distance from the query object. Otherwise, the objects that they contain are inserted into the queue ordered according to their distance from the query object. If the element e that has been removed from the queue is a data object, then e is reported as the next nearest neighbor of the query object.

In order to be able to visualize the behavior of the incremental nearest neighbor algorithm, at any instance of time (see Lenhof & Smid (1994) and Murphy & Skiena (1993) for alternative nearest neighbor algorithm animations) we distinguish between the following entities by using different colors.

1. Bounding boxes in the priority queue are denoted by light blue.
2. Objects in the priority queue are denoted by green.
3. Objects that have not yet been processed (i.e., entered explicitly into the queue or output into the ranking) are denoted by red.
4. Objects that have been processed and hence have been ranked are denoted by blue. The numeric position of the object in the ranking is displayed in blue next to the object.
5. Bounding boxes that have been processed (although their objects may still be in the queue) are denoted by gray.
6. Parts of the underlying space which have not been processed as they are outside the space spanned by the bounding boxes of the data objects are denoted by white.
7. The next item in the queue to be processed (could be a bounding box or an object) is denoted by yellow.
8. The query object is denoted by orange.

It is important to note that we distinguish between objects that have yet to be processed and those that are explicitly in the queue. Objects that have not yet been processed are those that have not been enqueued explicitly although some of their containing bounding boxes, say the set C, are in the queue, while the bounding boxes that contain the elements of C have already been dequeued. This distinction enables users to watch the progression of the algorithm.

Observe that if a bounding box b corresponding to a nonleaf node in the R-tree is in the queue, then all of the bounding boxes contained in b are also implicitly in the queue although we do not distinguish between them and b. This means that we really don't have an entity such as a bounding box that is unprocessed in contrast to having objects that have not yet been processed.

An interesting question is why we differentiate between enqueued objects and enqueued bounding boxes by using a different color for them (i.e., green and light blue, respectively). It would appear that we should use the same color for both of these entities as they can both be members of the queue. However, using such a convention would not enable us to distinguish between them when an object o in the queue overlaps a bounding box b in the queue where o is not in b. This is a direct result of the fact that the R-tree results in a non-disjoint decomposition of the underlying space.

Note that we distinguish between the elements in the priority queue, those that have been processed, and the one that is currently being processed (i.e., the next one to be processed). Initially, we only made a distinction between the elements in the priority queue and the ones that have already been processed. However, this made the visualization of the animation very difficult to follow as users could not easily detect what had changed from step to step. In particular, without making the explicit distinction between the elements that have been processed and the element to be processed (or currently being processed) we would have to do so via the process of a mental visual subtraction of 'before' and 'after' states of the animation process. We prefer our approach.

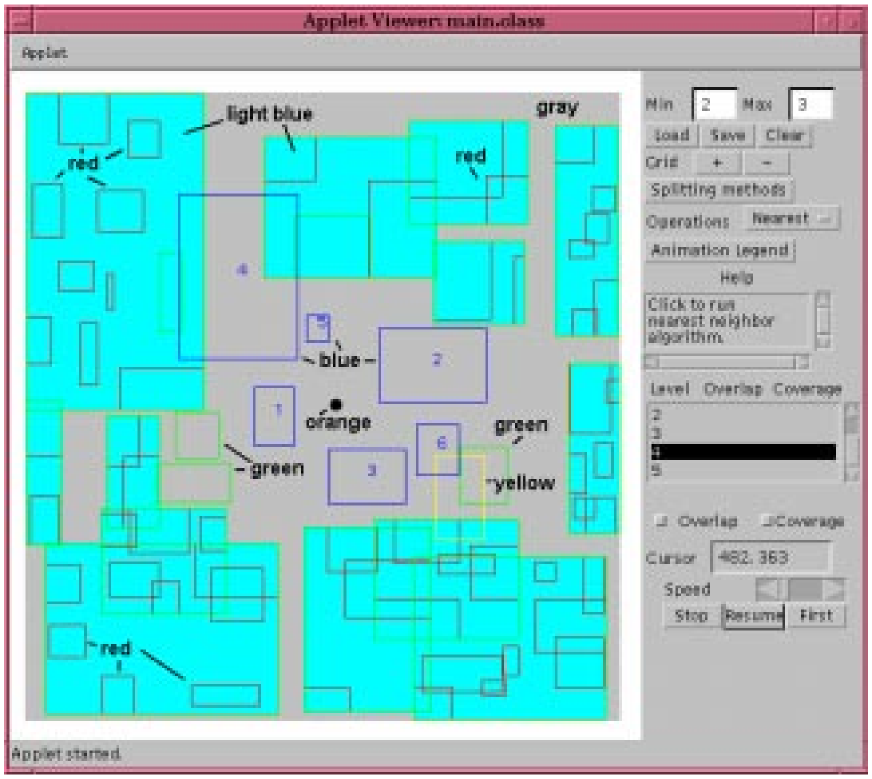
Observe that the white region corresponds to a part of the underlying space that is not contained in any of the bounding boxes. Since the incremental nearest neighbor algorithm visits the entire space spanned by the root node, it would seem that white is not necessary. However, it is used for the part of the underlying space that is outside of the bounding box associated with the root node. Of course, we could dispense with it, but doing so may confuse the user who might be misled to believe that the bounding box associated with the root node spans the entire drawing canvas which is not always the case. Note that if we were trying to visualize a representation that is based on a decomposition of the underlying space into disjoint blocks (e.g., one of the quadtree or k-d tree variants) rather than one that is based on an object hierarchy, then there would never be a need for the white region as the root spans the underlying space completely.

As an example of the execution of the incremental nearest neighbor algorithm, consider Figure 4 which shows an intermediate stage of the algorithm for an R-tree immediately after retrieving the six nearest neighbors. Observe that the manner in which the algorithm proceeds leads to a constantly growing circle centered at the query point where the bounding boxes inside the circle are gray corresponding to the bounding boxes that have been processed, while the area outside the circle is light blue corresponding to bounding boxes in the queue which remain to be processed. Notice the white area which corresponds to the portion of the underlying space that is outside the root of the R-tree.

The window query algorithm is a simple tree traversal that recursively visits the bounding boxes of the representation in a top-down manner checking at each stage if the bounding box b overlaps the query window. If there is no overlap, then exit the level of recursion. Otherwise, check if b is not at the lowest level of the hierarchy (i.e., b is a nonleaf node), in which case the algorithm is applied recursively to the immediate descendants of b. If b is at the lowest level of the hierarchy, then check if the objects contained in b are in the query window and report them as satisfying the query.

Once again, in order to be able to visualize the behavior of the window query algorithm, at any instance of time we distinguish between the following entities by using different colors.

1. Bounding boxes that remain to be processed (i.e., they partially overlap the query range) are denoted by light blue.



Min 2 Max 3

Operations Nearest

Click to run nearest neighbor algorithm.

Cursor 482, 363

Applet started.

Figure 4: R-tree applet showing an intermediate stage in the incremental nearest neighbor algorithm immediately after finding the sixth nearest neighbor. The query object is orange. The six nearest neighbors are blue as is their position in the ranking. All bounding boxes in the priority queue are light blue while the rectangles in the priority queue are green. The current item being processed, in this case a rectangle, is yellow. All rectangles not yet processed or not in the priority queue are red. The area spanned by bounding boxes that have been processed is gray, while the area outside the space spanned by the root node is white.

2. Objects that have not yet been processed but whose smallest enclosing bounding box has been found to be in the query range are denoted by green.
3. Objects that have not yet been processed (in the sense that their smallest containing bounding box has not been tested with respect to being in the query range or has been found to be outside the query range) are denoted by red.

4. Objects that have been processed and that have been found to be in the query range are denoted by blue.
5. Objects that have been explicitly processed and that have been found to be outside the query range are denoted by violet.
6. Bounding boxes that have been processed are denoted by gray (although some of the objects that they contain remain to be processed).
7. Bounding boxes that have not been processed as they are outside the query range are denoted by white.
8. The next item to be processed (could be a bounding box or an object) is denoted by yellow.
9. The query range is denoted by orange.

When the objects have extent (e.g., lines, rectangles, etc.), we need to be a bit more precise as to what is retrieved by the window query. The issue is whether the retrieved object o must be contained in its entirety in the query window w, whether o must enclose w, or whether a point on the boundary of o must intersect a part of the boundary of w (i.e., a partial overlap). For rectangles, we have the following three options:

1. The entire rectangle is in the query window.
2. The entire rectangle contains the query window.
3. At least some part of the rectangle boundary intersects (i.e,, crosses) the boundary of the window.

The applet forces the user to specify which combination of these variants of the window query is to be used by opening a window consisting of three check boxes plus a continue button. We use three check boxes rather than a check box group as the three options are not mutually exclusive in the sense that we would like to invoke combinations of them. The window query returns all rectangles that satisfy any of the selected options. For example, we may wish to return all rectangles that are contained in the query window or whose boundary crosses that of the query window. This is achieved by choosing options 1 and 3 in the above list as we want all rectangles that satisfy either of these options.

6 CONCLUDING REMARKS

We have described a JAVA^{TM} applet to enable users on the worldwide web to visualize different variants of the R-tree and to animate the execution of some common spatial operations on them. The applet can be found at: `http://www.cs.umd.edu/~hjs/rtrees/index.html`. A more detailed description of its capabilities and how to use it can be found in Brabec & Samet (1998). Future work includes adding more operations, data types, and data structures.

It is important to point out that although use of the applet can reveal qualitative differences between the various splitting methods, users must be careful to bear in mind that R-trees are designed for very large data sets and thus M (one of the bucket size parameters) is often as large as several hundred thereby mirroring the effect of a B-tree. On the other hand, by the nature of the size of the applet and the runtime environment which we use, our examples are usually restricted to small values of M and to data sets of small size. Thus any conclusions drawn about the relative efficacy of the different splitting methods from observations should be made with caution.

REFERENCES

Ang, C. H. & Tan, T. C. (1997), New linear node splitting algorithm for R-trees, *in* M. Scholl & A. Voisard, eds, 'Advances in Spatial Databases — Fifth International Symposium, SSD'97', Berlin, Germany, pp. 339–349. (Also Springer-Verlag Lecture Notes in Computer Science 1262).

Aref, W. G. & Samet, H. (1991), Optimization strategies for spatial query processing, *in* G. Lohman, ed., 'Proceedings of the 17th International Conference on Very Large Data Bases', Barcelona, pp. 81–90.

Arnold, K. & Gosling, J. (1996), *The JAVA*$^{\text{TM}}$ *programming language*, Addison-Wesley, Reading, MA.

Beckmann, N., Kriegel, H. P., Schneider, R. & Seeger, B. (1990), The R*-tree: an efficient and robust access method for points and rectangles, *in* 'Proceedings of the ACM SIGMOD Conference', Atlantic City, NJ, pp. 322–331.

Brabec, F. & Samet, H. (1998), The VASCO R-tree JAVA$^{\text{TM}}$ applet, *in* 'Proceedings of the IFIP 2.6 4th Working Conference on Visual Database Systems (VDB-4)', L'Aquila, Italy.

Brinkhoff, T., Kriegel, H. P., Schneider, R. & Seeger, B. (1994), GENESYS: A system for efficient spatial query processing, *in* 'Proceedings of the ACM SIGMOD Conference', Minneapolis, MN, p. 519.

Brown, M. H. & Sedgewick, R. (1984), 'A system for algorithm animation', *Computer Graphics* **18**(3), 177–186. (Also *Proceedings of the SIGGRAPH'84 Conference*, Minneapolis, July 1984).

Comer, D. (1979), 'The ubiquitous B-tree', *ACM Computing Surveys* **11**(2), 121–137.

Foley, J. D., van Dam, A., Feiner, S. K. & Hughes, J. F. (1990), *Computer Graphics: Principles and Practice*, second edn, Addison-Wesley, Reading, MA.

Güting, R. H. (1994), 'An introduction to spatial database systems', *VLDB Journal* **3**(4), 401–444.

Guttman, A. (1984), R-trees: a dynamic index structure for spatial searching, *in* 'Proceedings of the ACM SIGMOD Conference', Boston, MA, pp. 47–57.

Herring, J. R. (1996), Oracle7 spatial data option$^{\text{TM}}$: Advances in relational database technology for spatial data management, Technical report.

Hjaltason, G. R. & Samet, H. (1995), Ranking in spatial databases, *in* M. J. Egenhofer & J. R. Herring, eds, 'Advances in Spatial Databases — Fourth International Symposium, SSD'95', Portland, ME, pp. 83–95. (Also Springer-Verlag Lecture Notes in Computer Science 951).

Kamel, I. & Faloutsos, C. (1993), On packing R-trees, *in* 'Proceedings of the Second International Conference on Information and Knowledge Management', Washington, DC, pp. 490–499.

Kamel, I. & Faloutsos, C. (1994), Hilbert R-tree: An improved R-tree using fractals, *in* J. Bocca, M. Jarke & C. Zaniolo, eds, 'Proceedings of the 20th International Conference on Very Large Data Bases', Santiago, Chile, pp. 500–509.

Kang, M. A. & Li, K. J. (1995), Query processing methods for connectivity search in visual databases using R$^+$-trees, *in* S. Spaccapietra & R. Jain, eds, 'Proceedings of the IFIP 2.6 3rd Working Conference on Visual Database Systems (VDB-3)', Chapman and Hall, London, England, pp. 365–377.

Keim, D. A. & Kriegel, H. P. (1995), Possibilities and limits in visualizing large databases, *in* S. Spaccapietra & R. Jain, eds, 'Proceedings of the IFIP 2.6 3rd Working Conference on Visual Database Systems (VDB-3)', Chapman and Hall, London, England, pp. 177–190.

Lenhof, H. P. & Smid, M. (1994), An animation of a fixed-radius all-nearest-neighbors algorithm, *in* 'Proceedings of the Tenth Annual Symphosium on Computational Geometry', Stony Brook, NY, p. 387.

Murphy, M. & Skiena, S. S. (1993), Ranger: A tool for nearest neighbor search high dimensions, *in* 'Proceedings of the Ninth Annual Symposium on Computational Geometry', San Diego, CA, pp. 403–404.

Nievergelt, J., Hinterberger, H. & Sevcik, K. C. (1984), 'The grid file: an adaptable, symmetric multikey file structure', *ACM Transactions on Database Systems* **9**(1), 38–71.

Oracle Corporation (1996), Advances in relational database technology for spatial data management, Oracle spatial data option technical white paper.

Preparata, F. P. & Shamos, M. I. (1985), *Computational Geometry: An Introduction*, Springer–Verlag, New York.

Roussopoulos, N. & Leifker, D. (1985), Direct spatial search on pictorial databases using packed R-trees, *in* 'Proceedings of the ACM SIGMOD Conference', Austin, TX, pp. 17–31.

Roussopoulos, N., Kelley, S. & Vincent, F. (1995), Nearest neighbor queries, *in* 'Proceedings of the ACM SIGMOD Conference', San Jose, CA, pp. 71–79.

Samet, H. (1990*a*), *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*, Addison-Wesley, Reading, MA.

Samet, H. (1990*b*), *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, Reading, MA.

Scholl, M. & Voisard, A., eds (1997), *Advances in Spatial Databases — Fifth International Symposium, SSD'97*, Springer-Verlag, Berlin, Germany. (Also Springer-Verlag Lecture Notes in Computer Science 1262).

Sellis, T., Roussopoulos, N. & Faloutsos, C. (1987), The R^+–tree: a dynamic index for multi–dimensional objects, *in* P. M. Stocker & W. Kent, eds, 'Proceedings of the 13th International Conference on Very Large Databases (VLDB)', Brighton, England, pp. 71–79. (Also University of Maryland Computer Science TR–1795).

Spaccapietra, S. & Jain, R., eds (1995), *Proceedings of the IFIP 2.6 3rd Working Conference on Visual Database Systems (VDB-3)*, Chapman and Hall, London, England.

Stonebraker, M. (1997), Limitations of spatial simulators for relational DBMSs, Technical report, INFORMIX Software, Inc. (see `http://www.informix.com/informix/corpinfo/zines/whitpprs/wpsplsim.pdf`).

White, M. (1982), N-trees: large ordered indexes for multi-dimensional space, Statistical research division, US Bureau of the Census, Washington, DC.

František Brabec is a Research Assistant at the University of Maryland at College Park where he works in the area of spatial databases. He received an M.S. in Computer Science from the University of Maryland in 1997 and a B.S. and an M.S. in Informatics from Charles University, Prague, Czech Republic in 1993 and 1994, respectively.

Hanan Samet is a Professor of Computer Science at the University of Maryland, College Park. His research group has developed the QUILT system which is a GIS based on hierarchical spatial data structures such as quadtrees and octrees, and the SAND system which integrates spatial and non-spatial data. He received his Ph.D. in Computer Science from Stanford University in 1975. He is the author of the two books *The Design and Analysis of Spatial Data Structures* and *Applications of Spatial Data Structures: Computer Graphics, Image Processing and GIS*, Addison-Wesley, Reading, MA, 1990. He is a Fellow of the ACM, IEEE, and the IAPR (International Association of Pattern Recognition).