Visual Database Systems 4 (VDB4), Chapman & Hall, London, 1998, pp. 123-140.

The VASCO R-tree JAVA™ Applet*

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

Features of the VASCO R-tree JAVA™ applet are described to support its demonstration. This includes an explanation of the different R-tree node splitting methods that are implemented in VASCO and of the functionality of the control panel of the applet. The applet enables users to visualize the different variants of the R-tree as well as observe their behavior for finding the nearest neighbors to a query point and the objects within a given query window in an incremental manner. The applet can be found at `http://www.cs.umd.edu/~hjs/rtrees/index.html`.

Keywords

R-trees, JAVA™ applet, VASCO, spatial databases, worldwide web, incremental nearest neighbor finding, window queries, ranking

1 INTRODUCTION

The representation of spatial data is an important issue in spatial databases as well as in a wide variety of applications including computer graphics, computational geometry, computer vision, pattern recognition, and geographic information systems (GIS) (e.g., Samet (1990*a*) and Samet (1990*b*)). One popular representation is an object hierarchy that aggregates objects into groups based on proximity and then use proximity to further aggregate the groups. The R-tree (Guttman 1984) is an example of such a representation. In this paper we provide information to support a demo of an R-tree JAVA applet which is a part of VASCO, a system for Visualizing and Animating Spatial Constructs and Operations. We show its use in a comparative manner to examine the differences between a number of different implementations of the R-tree. The objects in our demo are rectangles drawn from the two-dimensional plane. The rectangles are aggregated into groups where the minimum bounding boxes of the groups are also aggregated thereby forming the hierarchy. The implementations that we examine differ in the way in which the rectangles and rectangle groups are aggregated, and in the way a group that becomes too full (i.e., overflows) is split.

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

The rest of this paper is organized as follows. Section 2 presents the different R-tree splitting methods that are supported. Section 3 describes the control panel of the R-tree applet. Section 4 outlines the animation modes and how they are realized in the control panel. Section 5 contains concluding remarks. For more details on the design of the R-tree applet, see Brabec & Samet (1998).

2 R-TREE SPLITTING METHODS

The number of objects or bounding boxes that can be stored in each R-tree node ranges between $m \leq \lceil M/2 \rceil$ and M (termed *bucket size parameters*). 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 one to fill each R-tree node to capacity. Below, we briefly describe the methods that we have implemented for building R-trees and dealing with nodes that are too full.

1. Exhaustive Search: If the R-tree node overflows, then this dynamic method tries all possible ways of splitting the node into two new nodes that satisfy the requirements on the minimal and maximal number of children that can be stored in a node. Choose the split which causes bounding boxes of the two new nodes to have the smallest area (Guttman 1984). Thus the goal of this method is to reduce the coverage (i.e., the area spanned by the nodes). This method is quite complex as it tries all possibilities.
2. Quadratic method: This dynamic method examines all the children of the overflowing node and finds the pair of bounding boxes that would waste the most area were they to be inserted in the same node. This is determined by subtracting the sum of the areas of the two bounding boxes from the area of the covering bounding box. These two bounding boxes are placed in separate nodes, say j and k. Next, examine the set of remaining bounding boxes, and the bounding box i whose addition maximizes the difference in coverage between the bounding boxes associated with j and k is added to the node whose coverage is minimized by the addition. This process is reapplied to the remaining bounding boxes (Guttman 1984). This method takes quadratic time.
3. Linear Method: This dynamic method finds the two bounding boxes with the greatest normalized separation along both axes, and splits along this axis. The remaining bounding boxes in the node are assigned to the nodes whose covering bounding box is increased the least by the addition (Guttman 1984). This method takes linear time.
4. R*-tree: The R*-tree (Beckmann, Kriegel, Schneider & Seeger 1990) is a name given to a variant of the R-tree which makes use of the most complex of the node splitting algorithms to realize a dynamic method. The algorithm differs from the

other algorithms as it attempts to reduce both overlap and coverage. In particular, the primary focus is on reducing overlap with ties broken by favoring the splits that reduce the coverage by using the splits that minimize the perimeter of the bounding boxes of the resulting nodes. In addition, when a node a overflows, instead of immediately splitting a, an attempt is made first to see if some of the objects in a could possibly be more suited to being in another node. This is achieved by reinserting a fraction (30% has been found to yield good performance (Beckmann et al. 1990)) of these objects in the tree (termed *forced reinsertion*). The node is only split if it has been found to overflow after reinsertion has taken place. This method is quite complex.

5. Ang/Tan Method: This dynamic method forms four sets, one for each face (i.e., side) of the overflowing node a. Associate each bounding box o in a with the closest face of a in each of the two dimensions. Once the four sets representing four partitions have been constructed (i.e., each bounding box o has been associated with two sets), select the partition that ensures the most even distribution of bounding boxes. In case of a tie, choose the partition with the least overlap. In case of another tie, choose the partition with the least coverage. This method takes linear time (Ang & Tan 1997).

6. Hilbert Non-packed R-tree: This dynamic method is based on ordering the objects on the basis of the Peano-Hilbert number (e.g., Samet (1990*b*)) corresponding to their centroid (Kamel & Faloutsos 1994). The objects are stored in a B^+-tree.

7. Morton Non-packed R-tree: This dynamic method is based on ordering the objects on the basis of the Morton number (e.g., Samet (1990*b*)) corresponding to their centroid (White 1982). The objects are stored in a B^+-tree.

8. Packed R-tree: This static method is based on ordering the objects on the basis of some criterion (Roussopoulos & Leifker 1985). It has two stages. In our implementation, 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. Once an entire level of the packed R-tree has been obtained, the algorithm is reapplied to add nodes at the next level using the same nearest neighbor criterion, terminating when a level contains just one node. The only difference between the ordering that is applied at the levels containing the nonleaf nodes from that used at the level of the leaf nodes is that in the former case we are ordering the bounding boxes while in the latter case we are ordering the actual objects.

9. Hilbert Packed R-tree: This static method is based on ordering the objects on the basis of the Peano-Hilbert number corresponding to their centroid (Kamel & Faloutsos 1993). Once this ordering has been obtained, the leaf nodes are filled to capacity by examining the objects in increasing order. The nodes at the remaining levels are ordered according to the time at which they were created.

10. Morton Packed R-tree: This static method is based on ordering the objects on the basis of the Morton number corresponding to their centroid (e.g., Kamel & Faloutsos (1993)). Once this ordering has been obtained, the leaf nodes are filled to capacity by examining the objects in increasing order. The nodes at the remaining levels are ordered according to the time at which they were created.

3 CONTROL PANEL OF THE DRAWING CANVAS

The control panel of the drawing canvas has a number of components. We describe them below in the order of their appearance, from top to bottom.

- `Min` and `Max`: Editable text fields to set the bucket size parameters m and M for the R-tree.
- `Load`: Button to load a data set of rectangles into the applet and build an R-tree using the current splitting method. The data may come from a file or an editable text area into which data can be typed or cut and pasted. Data can either be added to an existing R-tree or a new R-tree can be created. Since applets are not allowed to write into files, we are running a data server with storage capacity to which the applet can connect and from which it can download previously saved data.
- `Save`: Button to save the rectangles in the R-tree of the applet. The data may be written to a file or an editable text area from which data can be copied via cut and paste. Data can either be added to an existing file or a new file can be created. In case of a file, the file server described above is used.
- `Clear`: Button to delete the entire R-tree currently represented by the applet..
- `Grid`: Two buttons labeled '$+$' and '$-$' that impose a successively finer and looser, respectively, grid on the drawing canvas. Each time '+' is depressed, the resolution is doubled, while each time '-' is depressed, the resolution is halved.
- `Splitting Methods`: Button creating a window consisting of a check box group that indicates the variant of the R-tree being viewed, plus a `Close` button. The window stays open during all subsequent operations until explicitly closed by depressing the `Close` button. The splitting methods are described in Section 2. If the splitting method is based on ordering the objects based on their centroids (i.e., packed R-tree, Morton packed R-tree, Hilbert packed R-tree, Morton non-packed R-tree, and Hilbert non-packed R-tree), then display the locations of their centroids and their positions in the ordering. If the splitting method does not make use of an ordering (i.e., exhaustive, linear, quadratic, R*-tree, and Ang/Tan splitting methods), then remove the centroids and their positions in the ordering.
- `Operations`: Choice selecting the operation mode out of the following options:
  1. `Insert`: Insert a new rectangle by depressing the mouse at a particular point, which serves as one corner, and then drag the mouse (while depressed) to the point which is to serve as the diagonally opposite corner, at which time

release the mouse. If the splitting method is based on ordering the centroids 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), then display the location of the centroid of the rectangle and the positions of all the rectangles in the ordering.

2. `Delete`: Delete an existing rectangle which is the closest one to the position where the mouse is clicked by flashing its border.
3. `Window`: Execute a rectangular range query once the nature of the search has been specified by selecting the desired options. These options are given by three check boxes in a modal window opened when this operation was selected.
   (a) The entire rectangle is in the query window.
   (b) The entire rectangle contains the query window.
   (c) At least some part of the rectangle boundary intersects (i.e., crosses) the boundary of the window.

   Once the options have been chosen, select the `Continue` button in the window, thereby closing the window, and proceed to specify the search range rectangle in the same manner as in `Insert` mode. Since the window is modal, only operations inside the window are allowed until the window has been closed. Return all rectangles that satisfy any of the selected options.
4. `Nearest`: Order the rectangles by their distance from a query point.

- `Animation Legend`: Button to open a new window displaying a legend for the colors used in the current operation mode.
- `Help`: Non-editable text area to explain the current operation mode.
- `Visible Levels`: List in which each line corresponds to one level in the R-tree where the first row 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. Each line can also display coverage and overlap information for the corresponding level when the `Overlap` and `Coverage` check boxes that are situated immediately below the `Visible Levels` list are set.
  1. `Overlap`: Ratio indicating the extent to which bounding boxes have area in common. Smaller overlap reduces the likelihood that more than one child subtree is visited at each level during a search.
  2. `Coverage`: Ratio indicating the extent to which bounding boxes span an area greater than that of the actual data rectangles. Smaller coverage reduces the likelihood that a subtree will be visited during a search when its child subtrees do not overlap the query object.
- `Cursor`: Text field indicating the position of the mouse on the drawing canvas.
- `Speed`: Horizontal scroll bar used to vary the speed of the animation process.

- `Animation Control`: Buttons to control the animation process. Only three buttons are visible at any given instance of time (see Section 4).
  1. `Start`: Start the animation process in continuous mode.
  2. `Stop`: Halt the animation process.
  3. `First`: Start the animation process in incremental mode to find the first object satisfying the search query.
  4. `Next`: Continue the animation process in incremental mode to find the next object satisfying the search query.
  5. `Pause`: Temporarily suspend the animation process.
  6. `Resume`: Resume the animation process in the same animation mode in which it was when it was last suspended.

4 ANIMATION MODES

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 selects the `Pause` or `Stop` buttons. Regardless of the mode, users can temporarily suspend the animation by selecting the `Pause` button which is converted to a `Resume` button for subsequent resumption of the animation. Note that once the `Pause` button has been selected, animation is suspended immediately. In other words, even if we are in incremental mode, then we do not finish the search for the object until the `Resume` button has been selected. The incremental mode is initiated by selecting the `First` button which is converted to a `Next` button for subsequent continuation of the incremental animation. Selecting the `First` button also causes the `Start` button to be converted to a `Stop` button. However, once the `Next` button has been selected for the first time, its semantic action remains one of obtaining the next object that satisfies the query. The continuous mode is initiated by selecting the `Start` button which is converted to a `Stop` button for future halting of the animation. Once the `Stop` button has been selected, all animation halts regardless of the mode and the `Stop` button is converted to a `Start` button while the `Next` button is reset to a `First` button if it was not already set to this value.

5 CONCLUDING REMARKS

We have described the features of the VASCO R-tree JAVA™ applet which enables 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`.
Future work includes adding more operations, data types, and data structures.

Visual Database Systems 4 (VDB4), Chapman & Hall, London, 1998, pp. 123-140.

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).

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), Visualizing and animating R-trees and spatial operations in spatial databases on the worldwide web, *in* Y. Ioannidis & W. Klas, eds, 'Visual Database Systems (VDB4)', Chapman and Hall, London, pp. 123–140. (Also *Proceedings of the IFIP TC2//WG2.6 Working Conference on Visual Database Systems 4 (VDB4)*, L'Aquila, Italy, May 1998)).

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

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.

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.

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.

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,

Visual Database Systems 4 (VDB4), Chapman & Hall, London, 1998, pp. 123-140.

Reading, MA, 1990. He is a Fellow of the ACM, IEEE, and the IAPR (International Association of Pattern Recognition).