Visualizing and Animating Search Operations on Quadtrees on the Worldwide Web*

František Brabec
Computer Science Department
University of Maryland
College Park, Maryland 20742
brabec@umiacs.umd.edu

Hanan Samet
Computer Science Department
Center for Automation Research
Institute for Advanced Computer Studies
University of Maryland
College Park, Maryland 20742
hjs@cs.umd.edu

December 24, 1999

Abstract

A set of spatial index JAVA™ applets is described that enable users on the worldwide web to experiment with a number of variants of the quadtree spatial data structure for different spatial data types, and, most importantly, enable them to see in an animated manner how a number of basic search operations are executed for them. The spatial data types are points, line segments, and rectangles. The search operations are finding nearest neighbors from an object of arbitrary type and shape, and retrieving all objects that overlap an object of arbitrary type and shape or are within a given distance of an object of arbitrary type and shape. The nearest neighbor and within queries retrieve their results in the order of their distance from the given query object. The representations and algorithms are visualized and animated in a consistent manner using the same primitives so that the differences between the effects of the representations can be easily understood. The applets can be found at `http://www.cs.umd.edu/~hjs/quadtree/index.html`.

Keywords and phrases: quadtrees, worldwide web, incremental nearest neighbor finding, window queries, ranking, animation, visualization

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

In this paper we focus on the visualization of different representations of spatial data and on the use of animation to visualize search operations such as those found in geographical information systems (GIS). We have chosen an approach that is accessible on the worldwide web. Our work is strongly related to that performed in algorithm animation. Much of the research in this field has focussed on investigating issues involved in the development of general purpose systems designed to work for many different types of algorithms (e.g., [5, 4, 6, 16]). As our focus is on the representations of spatial data, systems that are oriented towards geometric computing are of greater interest. However, many of these systems (e.g., [7, 9, 14]) are primarily designed as general systems to facilitate the implementation of geometric algorithms rather than their animation although it is possible to do so using them. More closely related to our work, although with a different focus, are GASP-II [15] which is designed to facilitate the visualization of three-dimensional geometric algorithms in an electronic classroom and Mocha [2] which provide animation of two-dimensional algorithms on the worldwide web. In contrast, we focus on the issues involved in the visualization of a particular family of spatial data structures and a specific application domain. It is important to note that we are not providing a general framework for all algorithms and data representations.

The representation of spatial data is an important issue in computational geometry, geographic information systems (GIS), computer graphics, image processing, pattern recognition, robotics, etc. There are two types of spatial data: locational data and object data. Locational data consists of points while object data consists of spatial objects that have extent (i.e., they occupy space) such as line segments, rectangles, regions, surfaces, volumes, etc. The representation of spatial data involves the selection of an appropriate decomposition of the underlying space that contains the spatial objects as well as the selection of 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.

There are two principal methods of representing spatial data. The first 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. Data structures such as the R-tree [8] and the R*-tree [3] are examples of the use of this method. The alternative is based on a recursive 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., line segments, rectangles, and any other non-point objects), then an object may be associated with more than one block.

In this abstract we present VASCO, a system for Visualizing and Animating Spatial Constructs and Operations. VASCO consists of a set of spatial index JAVA$^{\text{TM}}$ (e.g., [1]) applets that enable users on the worldwide web to experiment with a number of quadtree and R-tree (e.g., [11, 12]) representations for different spatial data types, and, most importantly, enable them to see in an animated manner how a number of basic spatial database search operations are executed for them. Due to the limitations of our displays, our examples are restricted to a two-dimensional domain where the objects can be points, line segments, and rectangles. The decompositions can be regular (i.e., based on a recursive halving or quartering of the underlying space into blocks of equal size) until the stopping condition is satisfied, or non-regular (in which case the blocks can have arbitrary size). A non-regular decomposition is one where the positions of the partitions are based on the data rather than rather than being restricted to fixed positions due to the halving or quartering of the underlying space. The decomposition process can partition all of the axes at once (as is the case for the quadtree), or one axis at a time (e.g., a k-d tree).

Figure 1 shows a sample applet window for quadtree representations of line data. The applet window contains a drawing canvas and a control panel. The drawing canvas shows a PM_1 quadtree [13] representation of a collection of line segments that form a polygonal subdivision. In essence, the underlying space is recursively decomposed into four blocks of equal size until there is just one portion of a line segment in each block, or if there are portions of more than one line segment in a block, then all of the portions must meet at a vertex within the block. Figure 1 also contains a window describing the available data structures with the line applet (see also Figure 2a). This window is implemented as a JAVA^{TM} check box group plus a close button. The window is created when the user activates the `Data Structures` button. Similar control panels and data structure selection windows are available for the point (Figure 2b) and rectangle (Figure 2c) applets.

Although the representations that we have implemented are hierarchical, in most cases only the decomposition at the lowest level of the tree which corresponds to blocks of the underlying space is shown explicitly. This means that the elements of the hierarchy are displayed implicitly as they consist of aggregations of the blocks at the lower level. In particular, the blocks at the remaining levels are visualized by combining the blocks at the lower level. Note that displaying just the lowest level of the hierarchy is adequate for most of the representations as their intermediate levels only correspond to different aggregations of the underlying space rather than the objects contained in the space. Nevertheless, in the animation of the search operations, these blocks are often displayed explicitly (thereby hiding the blocks at the lower levels). Such a situation arises when the algorithms process the elements of the data structure in a top-down manner as is the case when the algorithms are based on a tree traversal or when the algorithms take advantage of the aggregation of the blocks to prune some of the smaller blocks from consideration.

Figure 1 shows the available operations for the line data applet. Most of these operations are available to all of the applets, when applicable. Users can see how the underlying space is decomposed through the ability to insert and delete data in an incremental manner. Moreover, users can also see the effect of the positioning and resizing of the data on the representation through the use of a move operation. Of course, the same effect could be achieved by a sequence of `delete` and `insert` operations. However, the move capability is much more intuitive (and most importantly less tedious) thereby enabling users to get a continuous view of how the data structure changes as the data is moved. In the case of non-point data, users have the ability to move both the object and its constituent elements thereby changing its size. For example, in the case of line segment data, we can move the lines (a `move` operation), as well as change them by modifying their constituent elements (i.e., the vertices via a `move vertex` operation). Similarly, for rectangle data, we can move the rectangles (via a `move` operation), as well as modify them by moving their vertices and edges (via `move vertex` and `move edge` operations). This is preferable to the tedious process of deleting a data item and reinserting it at another position.

Users of the applets can see how the space decompositions support the most common database search operations including finding nearest neighbors from an object of arbitrary type and shape (`nearest` as in Figure 3a), retrieving all objects that overlap an object of arbitrary type and shape (`overlap` as in Figure 3b) or within a given distance of an object of arbitrary type and shape (`within` as in Figure 3c). These operations are executed in an incremental manner which means that the data objects that satisfy the query are returned and displayed one-by-one. When the algorithms that implement the search operations are run to completion, they provide the data objects for the `nearest` and `within` operations in the order of their distance from the query object (i.e., they are ranked in this order). The advantage of the incremental algorithm over traditional methods that compute the

k nearest neighbors (e.g., [10]) is that if instead we want the $k+1$ nearest neighbors, then we don't have to compute all $k+1$ neighbors again. In contrast, using our algorithm, the $k+1^{st}$ neighbor is found by simply continuing the search from the point immediately after having found the k^{th} nearest neighbor.

The algorithms are implemented using a priority queue where the queue elements are the blocks of the underlying data structure as well as the objects themselves. In the case of the `nearest` and `within` operations, the priority queue is ordered on the basis of the distance of its elements from the location of the query object. The algorithm works in a top-down manner in the sense that as elements are removed from the queue, they are checked if they correspond to blocks 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 sons) 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.

The `nearest` and `overlap` operations are visualized by distinguishing, using colors, between the blocks and objects that have been processed and those that remain to be processed by virtue of either being in the queue or not even examined. The speed of the animation can be varied. It can be run in continuous mode, or incrementally in which case it halts each time an object is processed by the algorithms, or each time an answer is found. There is also a capability to review the progress of each algorithm by replaying it backward through the use of the `progress` scrollbar.

The same incremental approach can also be used with the `overlap` operation. The overlap algorithm is a simple tree traversal that visits the blocks of the representation in a top-down manner checking at each stage if the block b overlaps the query window. If there is no overlap, then exit. 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.

The `overlap` operation algorithm is animated in a similar manner to that for the `nearest` and `within` operations. In particular, it also makes use of the concept of blocks to be processed, as well as objects to be processed. It uses a queue to keep track of the blocks and objects whose smallest containing block has a nonempty intersection with the query range. In other words, the queue contains the blocks and objects that are guaranteed to be processed in the future (i.e., they are explicitly checked to see if their intersection with the query range is nonempty). The difference from the priority queue of the `nearest` and `within` operations is that there the elements in the priority queue are ordered by their distance from the query object while there is no required order in the set of blocks or objects to be processed in the `overlap` operation. In other words, once a block b is inserted in the set of blocks to be processed, b can be processed at any time.

The VASCO system can be found at: `http://www.cs.umd.edu/~brabec/quadtree/index.html`. The VASCO system has found use in a number of applications such as spatial database performance evaluation and also as an instructional aid in courses in computational geometry and database design.

References

[1] K. Arnold and J. Gosling. *The JAVA™ Programming Language*. Addison-Wesley, Reading, MA, 1996.

[2] J.E. Baker, I.F. Cruz, G. Liotta, and R. Tamassia. Algorithm animation over the world wide web. In *Proceedings of the International Workshop on Advanced Visual Interface (AVI'96)*, pages 203–212, Gubbio, Italy, May 1996.

[3] N. Beckmann, H. P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *Proceedings of the ACM SIGMOD Conference*, pages 322–331, Atlantic City, NJ, June 1990.

[4] M. Brown. Zeus: a system for algorithm animation and multi-view editting. *Computer Graphics*, 18(3):177–186, May 1992.

[5] M. H. Brown. Exploring algorithms using Balsa-II. *Computer*, 21(5):14–36, May 1988.

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

[7] P. Epstein, J. Kavanagh, A. Knight, J. May, T. Nguyen, and J. R. Sack. A workbench for computational geometry. *Algorithmica*, 11(4):404–428, April 1994.

[8] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proceedings of the ACM SIGMOD Conference*, pages 47–57, Boston, MA, June 1984.

[9] K. Mehlhorn and S. Naher. Leda: a library of efficient data types and algorithms. In *Mathematical Foundations of Computer Science 1989 (MFCS'89)*, A. Kreczmar and G. Mirkowska, eds., pages 88–106, Porabka-Kozubnik, Poland, August 1989. (Also Springer-Verlag Lecture Notes in Computer Science 379).

[10] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proceedings of the ACM SIGMOD Conference*, pages 71–79, San Jose, CA, May 1995.

[11] H. Samet. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*. Addison-Wesley, Reading, MA, 1990.

[12] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, MA, 1990.

[13] H. Samet and R. E. Webber. Storing a collection of polygons using quadtrees. *ACM Transactions on Graphics*, 4(3):182–222, July 1985. (Also *Proceedings of Computer Vision and Pattern Recognition 83*, Washington, DC, June 1983, 127–132; and University of Maryland Computer Science TR–1372).

[14] P. Schorn. The XYZ GeoBench: a programming environment for geometric algorithms. In *Computational Geometry – Methods, Algorithms and Applications*, H. Bieri and H. Noltemeier, eds., pages 187–202, Bern, Switzerland, March 1991. (Also Springer-Verlag Lecture Notes in Computer Science 553).

[15] M. Shneerson and A. Tal. Visualization of geometric algorithms in an electronic classroom. In *Proceedings IEEE Visualization'97*, R. Yagel and H. Hagen, eds., pages 455–458, Phoenix, AZ, October 1997.

[16] J. T. Stasko. Tango: a framework and system for algorithm animation. *Computer*, 23(9):27–39, September 1990.

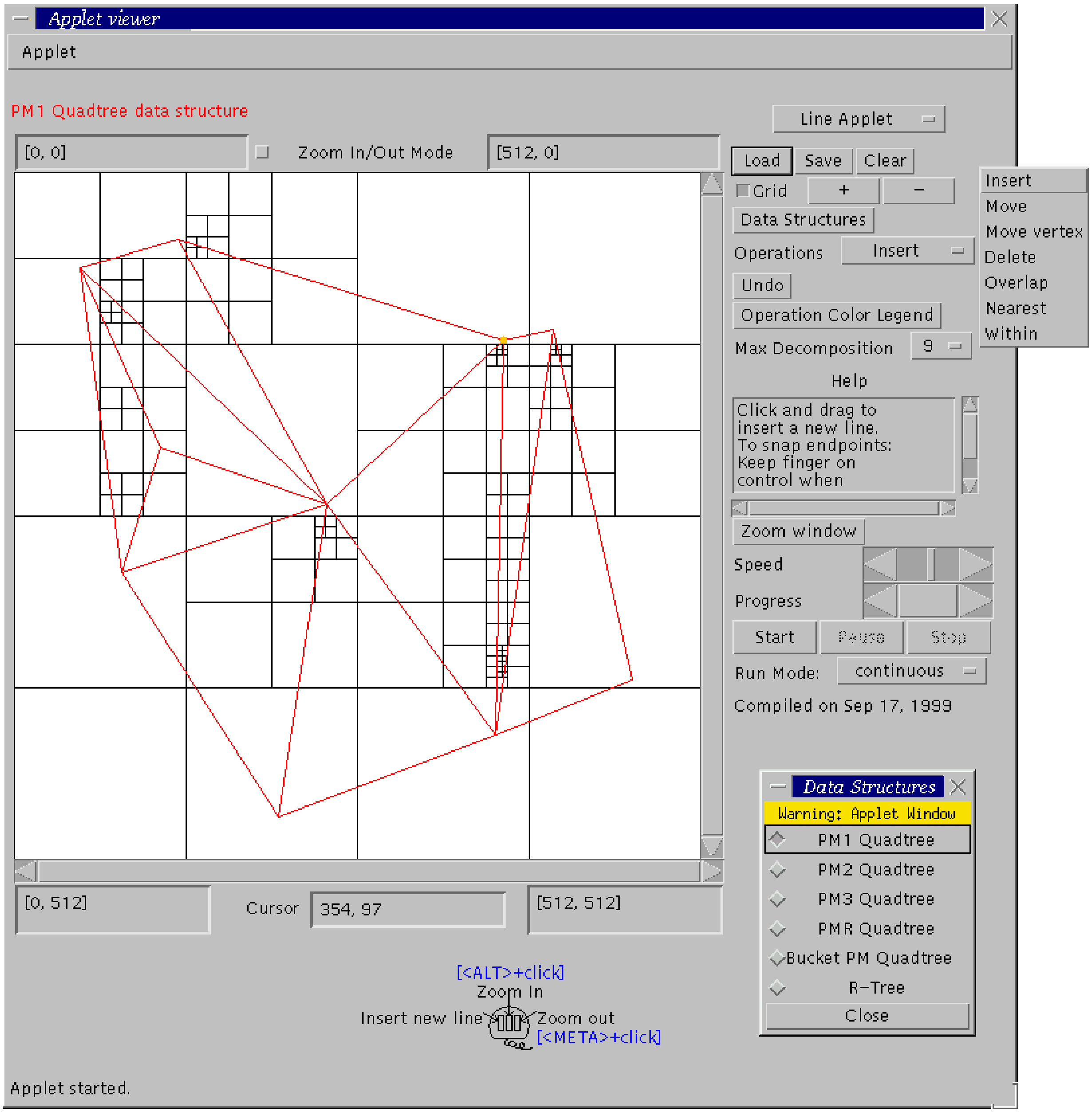
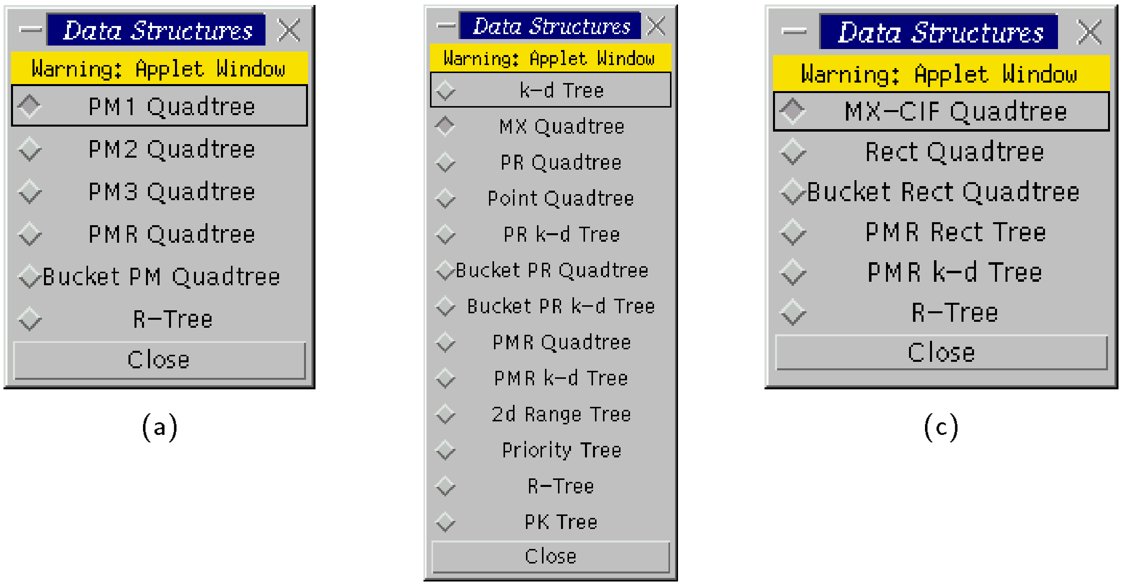
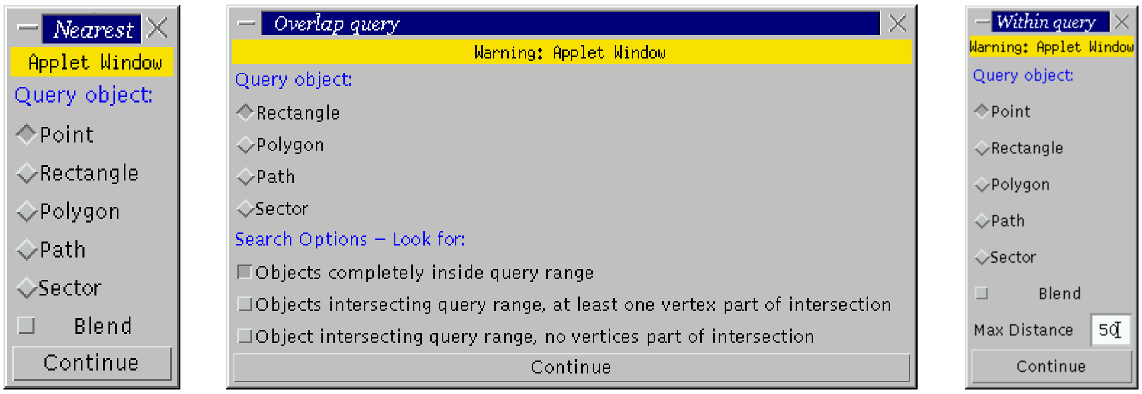


Figure 1: A sample applet window for line segment data. The drawing canvas shows a sample polygonal subdivision represented by a PM_1 quadtree. The set of possible data structures are also shown by a separate window.



(a) (b) (c)

Figure 2: The set of possible data structures for the (a) line segment, (b) point, and (c) rectangle data applets.



(a) (b) (c)

Figure 3: The dialog boxes for the (a) nearest, (b) overlap, and (c) within queries.