To appear in *International Encyclopedia of Geography*, D. Richardson, ed., Wiley and Sons, Oxford, UK, 2016.

Sorting Spatial Data*

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 www.cs.umd.edu/~hjs

February 25, 2015

Word count: 5542

Abstract

Sorting the underlying spatial data enables the efficient execution of operations on it in geographic information systems (GIS). The chapter differentiates the discussion on the basis of the type of the spatial data which in this case falls into raster and vector. The raster data is sorted via the use of a number of different space-filling curves which are mappings from two dimensions to one dimension. The motivation comes from the fact that the data is extremely voluminous and thus all of it cannot be fit into memory at once. Therefore, external storage must be used. The sorting is applied in order to enable efficient execution of operations. In the case of vector data, two methods are discussed. The first is based on an object hierarchy and distinguishes between occupied and unoccupied space, while the second sorts the data with respect to the space that is occupied and in essence sorts the regions comprising the underlying space so that the number of spatial objects that they contain is within the same range. In both cases of the vector data, the sorting is implicit, whereas in the case of the raster data, the sorting is explicit.

1 Introduction

Geographic information systems (GIS) process both spatial and attribute data (also known as feature data). Their performance is greatly dependent on the representation of the data and the extent to which they can integrate it. Ideally, this integration is done through the use of database management systems which are designed to deal with attribute data and hence must be modified to also handle spatial data. One of the main issues for spatial data is the volume of the data, which affects how it is stored and accessed, and what operations will be applied to it.

There are two types of operations. The first type of operations process the data in its entirety as is the case with operations such as map overlay and connected component labeling in the case of raster images. At times, the operations are applied to several data sets in which case they should be applied in such a way that the same locations in the two datasets can be accessed at the same time. This means that the order in which the data is processed is of importance and this is usually achieved by sorting the data.

*This work was supported in part by the National Science Foundation under Grants IIS-10-18475, IIS-12-19023, and IIS-13-20791.

The second type of operations process only a small subset of the data as is the case when we access the data at random or the portion of the data that is processed satisfies some predicate which can involve both its spatial and/or attribute components. An example is when browsing through the data looking for cities with population greater than one million within 10 miles of rivers (e.g., Brabec and Samet 2007, Esperanca and Samet 2002, and Samet et al. 2003). The latter type of processing is important as its efficient execution (other than using brute force to examine every data item) brings into play the need to incorporate notions of sorting as its use is a prerequisite for its efficient retrieval.

In this chapter we discuss the use of sorting to enable the efficient execution of operations on spatial data. Our discussion focuses on the type of the spatial data which in this case falls into raster and vector. Section 2 reviews the application of sorting to raster data where the main motivation is the fact that the data is extremely voluminous and thus all of it cannot be fit into memory at once. Therefore, external storage must be used. In this case the motivation for the sorting is to enable efficient execution of operations. Section 3 reviews the application of sorting to vector data although, as we point out, the methods that we describe are also applicable to raster data. Concluding remarks are drawn in Section 4.

2 Raster Data

The natural representation for raster data (i.e., two-dimensional images) is as an array of pixels where the array serves as an access structure to the raster image. The problem is that even for moderately-sized images, the storage requirements are high and thus the array does not always fit into memory. Hence it is stored on disk in some order with the aid of a mapping from the multidimensional space (two in this case) to a one-dimensional space (i.e., $Z \times Z$ to Z) which reflects the order in which the array elements (i.e., the pixels here) are stored. There are many ways of ordering the pixels that make up a raster image. The objective of the ordering is to provide a systematic way of processing all of the pixels with the property that every pixel in the image will be processed (i.e., none are missed). Any ordering that we devise must satisfy this property as otherwise it is not useful. Such orderings are termed *space-filling curves* (Sagan 1994).

Some of the most important orderings for a two-dimensional space are illustrated in Figure 1 for an 8×8 portion of the space and are described briefly below. Of course, orderings can also be devised for data of three dimensions and higher, but examples involving them are beyond the scope of our discussion. Choosing among the different orderings is not easy because each one has its advantages and disadvantages. Below, we review a few of their desirable properties and indicate which of the orderings satisfy them, and which do not. The mapping from the higher-dimensional space to the integers should be relatively simple and likewise for the inverse mapping. This is the case for all but the Peano-Hilbert order (Figure 1(d)). For the Morton order (Figure 1(c)), the mapping is obtained by interleaving the binary representations of the coordinate values of the pixel in the grid. The result of the mapping (i.e., the number associated with each pixel) is known as its *Morton number*. The Gray order (Figure 1(g)) is obtained by applying a Gray code to the result of bit interleaving, and the double Gray order (Figure 1(h)) is obtained by applying a Gray code to the result of bit interleaving the Gray code of the binary representation of the coordinate values. The U order (Figure 1(i)) is obtained in a similar manner to the Morton order, except for an intermediate application of $d-1$ "exclusive or" ($\oplus$) operations on the binary representation of selected combinations of the coordinate values prior to the

application of bit interleaving. Thus, the difference in cost between the Morton order and the U order in d dimensions is just the performance of additional $d-1$ "exclusive or" operations. This is in contrast to the Peano-Hilbert order, where the mapping and inverse mapping processes are considerably more complex.

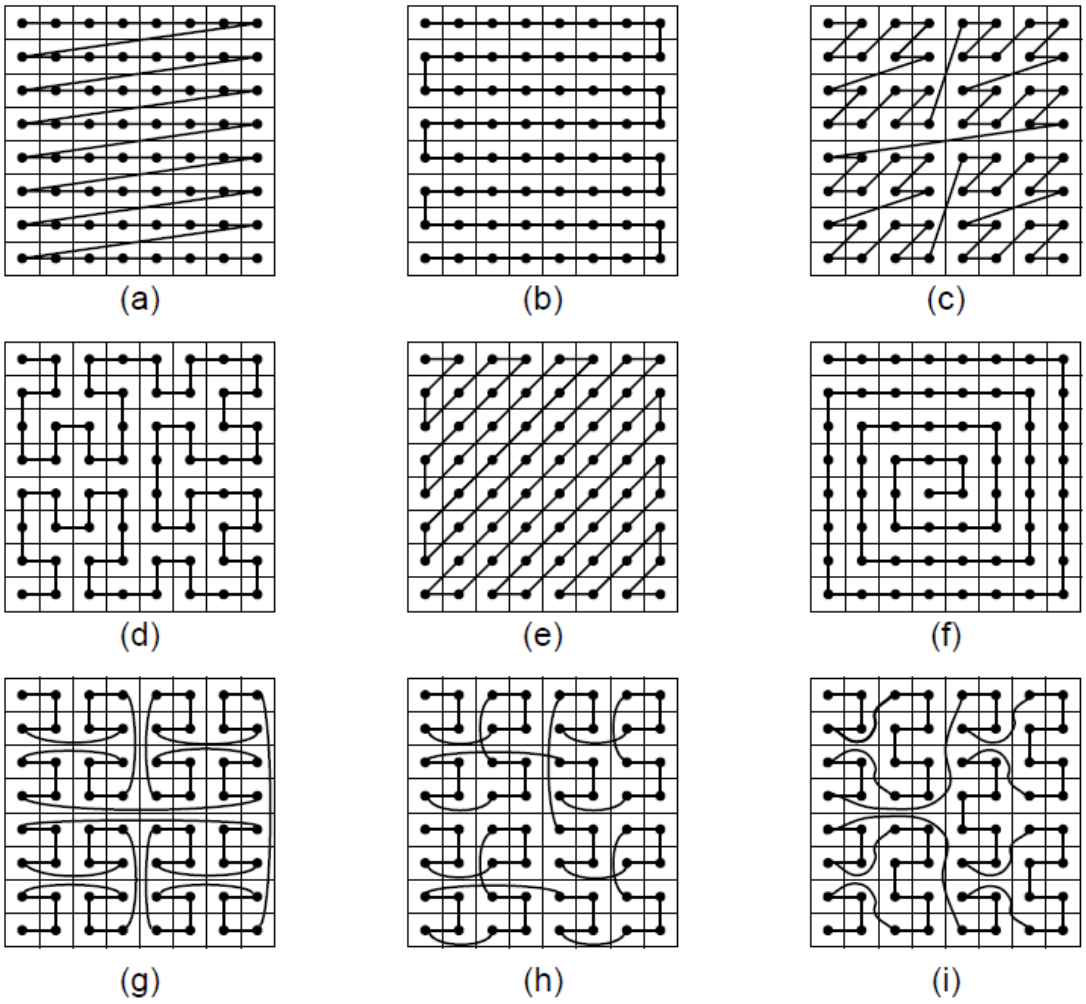

Figure 1: The result of applying several different space-ordering methods to an 8×8 collection of cells whose first element is in the upper-left corner: (a) row order, (b) row-prime order, (c) Morton order, (d) Peano-Hilbert order, (e) Cantor-diagonal order, (f) spiral order, (g) Gray order, (h) double Gray order, and (i) U order.

The ordering should be stable. This means that the relative ordering of the individual locations is preserved when the resolution is doubled (e.g., when the size of the two-dimensional space in which the pixels are embedded grows from 8×8 to 16×16) or halved, assuming that the origin stays the same. The Morton, U, Gray, and double Gray orders are stable, while the row (Figure 1(a)), row-prime (Figure 1(b)), Cantor- diagonal (Figure 1(e)), and spiral (Figure 1(f)) orders are not stable. The Peano-Hilbert order is also not stable, as can be seen by its definition. In particular, in two dimensions, the Peano-Hilbert order of

resolution $i + 1$ (i.e., a $2^i \times 2^i$ image) is constructed by taking the Peano-Hilbert curve of resolution i and rotating the NW, NE, SE, and SW quadrants by 90 degrees clockwise, 0 degrees, 0 degrees, and 90 degrees counterclock-wise, respectively. For example, Figure 2(a–c), gives the Peano-Hilbert curves of resolutions 1, 2, and 3, respectively.

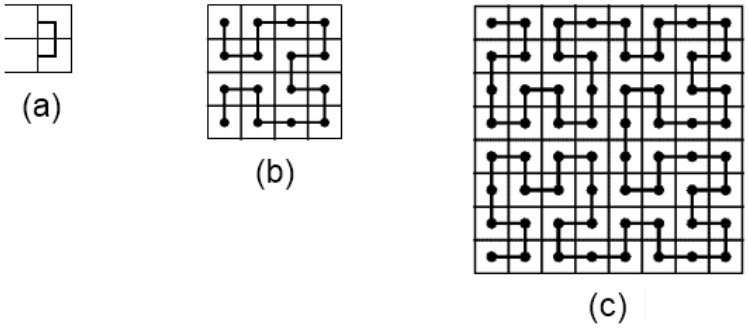

Figure 2: Peano-Hilbert curves of resolution (a) 1, (b) 2, and (c) 3.

Two pixels that are adjacent in the sense of sharing an edge or a side (also known as 4-adjacent) in space are neighbors along the curve defined by the ordering and vice versa. This property is impossible to satisfy for all pixels at all space sizes. However, for the row-prime, Peano-Hilbert, and spiral orders, every element is a 4-adjacent neighbor of the previous element in the sequence, while this is not the case for the other orders. This means that the row-prime, Peano-Hilbert, and spiral orders have a slightly higher degree of locality than the other orders.

The process of retrieving the neighbors of a pixel should be simple. Finally, the ordering should be *admissible* (Dillencourt et al. 1992)—at each pixel position in the ordering, at least one 4-adjacent neighbor in each of the lateral directions (i.e., horizontal and vertical) must have already been encountered. This is useful in several algorithms (e.g., connected component labeling). The row and Morton orders are admissible, while the Peano-Hilbert, U, Gray, and double Gray orders are not admissible. The row-prime, Cantor-diagonal, and spiral orders are admissible only if we permit the direction of the 4-adjacent neighbors to vary from position to position along the curve. For example, for the row-prime order, at positions on odd rows, the previously encountered 4-adjacent neighbors are the western and northern neighbors, while at positions on even rows, they are the eastern and northern neighbors.

The row order (Figure 1(a)) is of special interest to us because its mapping function is the one most frequently used by the multidimensional array which is the most common access structure for a raster image. Assuming that the first entry in the linear structure that is accessed by the mapping has an index value of 0, location (a,b) in an 8×8 collection of pixels is mapped to index value $8b + a$. An alternative ordering that is also used in the multidimensional array is known as *column order* where the difference from row order lies in the order in which the various dimensions are scanned. In the column order, the y coordinate value (row number)

varies most rapidly; in the row order, the x coordinate value (column number) varies most rapidly. The one to choose is arbitrary, although the row order is preferred as it yields a lexicographic ordering when array references are of the form $T[i, j]$, corresponding to an element in row i and column j of array T (i.e., at pixel address$(x,y) = (j,i)$).

The Morton order has a long history, having been first mentioned in 1890 by Peano, and is often used. It is also known as a *Z order* and as an N order. The Peano-Hilbert order was first mentioned soon afterwards by Hilbert number of researchers.

The U order is a variant of the Morton order but also resembles the Peano-Hilbert order. The primitive shape is a "U," which is the same as that of the Peano-Hilbert order. However, unlike the Peano-Hilbert order, and like the Morton order, the ordering is applied recursively with no rotation, thereby enabling it to be stable. The U order has a slight advantage over the Morton order in that more of the pixels that are adjacent (i.e., in the sense of a $(d - 1)$-dimensional adjacency) along the curve are also neighbors in space. This is directly reflected in a lower average distance between two successive positions in the order, which can also be shown to be lower than that of the Gray and double Gray orders. However, the price of this is that, like the Peano-Hilbert order, the U order is also not admissible. Nevertheless, like the Morton order, the process of retrieving the neighbors of a location in space is simple when the space is ordered according to the U order.

At times, the number of pixels may be so large that application of statistical data compression techniques from coding theory may be worthwhile. However, we are interested in methods that exhibit progressiveness. Therefore, as results of processing the data are obtained, we can see the partial results of the operation rather than having to wait until the operation is complete. The result is that we limit ourselves to the situation where many of the pixels have the same data values in which case we try to reduce the required storage by grouping them into blocks of identically-valued constituent pixels. This is especially the case for binary images or other images where adjacent pixels often have the same value. For example, consider a crop map where each crop type is assigned a different numeric code or color.

There are two ways to group identically-valued pixels into blocks. The first aggregates consecutive identically-valued pixels into one-dimensional blocks and termed a *runlength encoding* (Samet 2006). This is particularly useful in the case of the row ordering (Figure 1(a)) where the elements of the ordering are pairs of the form (a, b) where a is the value representing color/type and b is the length of the block.

The second approach applies a two-dimensional aggregation into blocks where the sizes of the blocks are constrained to be powers of 2 and are located at specific positions. In particular, assuming an origin at the upper-left corner of the image, the coordinate values of the upper-left corner of each block (e.g., (i, j) in two dimensions) of size $2^s \times 2^s$ satisfy the property that $i \bmod 2^s = 0$ and $j \bmod 2^s = 0$. The resulting block decomposition is known as a *region quadtree* [13], which also serves as the basic data structure in the QUILT (Shaffer et al. 1990) GIS and spatial library. Such a block decomposition is particularly useful in the case of the Morton, Peano-Hilbert, Gray, double Gray, and U orders (Figure 1). In this case, again, the elements of the ordering are pairs of the form (a, b) where a is the color/type and b is the side length of the square block.

The result of the ordering is an effective linearization of the data. The orderings are adequate when our operations require that all elements in the dataset (i.e., image) be accessed from start to end. However, some operations require access to particular elements of the array (known as *random access*) and therefore performing a

sequential scan from the start is very inefficient. In this case an access structure is useful and can be the one-dimensional array corresponding to the result of the mapping from 2d to 1d. However, this is more complex once we start grouping identically-valued blocks and keeping them in the ordering as the image elements corresponding to the elements in the ordering are no longer the same size. Thus we resort to a tree-like access structure where associated with each element of the ordering is a number corresponding to its position in the ordering and this is the number used to access the tree. The tree can be a binary search tree, B-tree, etc. For example, in the case of the Morton ordering this number is formed concatenating the result of interleaving the binary representation of the *x* and *y* coordinate values of the pixel in the upper-left corner of the block (assuming an origin in the upper-left corner of the image) and the binary representation of the base 2 logarithm of the block's side length. Traversing the tree in the order NW, NE, SW, and SE yields an ordering of the blocks in increasing order.

3 Vector Data

The application of sorting to vector data has its roots in visualization applications in computer graphics. The earliest examples are the hidden-line and hidden-surface elimination algorithms due to Warnock that recursively decompose the picture area into rectangles that are successively smaller while searching it for areas that are sufficiently simple to be displayed. The determination of what part of the picture area is hidden or not is equivalent to sorting the picture area with respect to the position of the viewer. The concept of "sorting" is used in two ways in the above hidden viewing algorithms. The first is the conventional one of ordering the visible part of the picture area with respect to the viewer. However, a different interpretation of "sorting" is given when we recursively decompose the picture area into rectangles stopping when their complexity is reduced. In this case, we are using the dictionary definition of "sorting" which is one of "putting in a certain place or rank according to kind, class, or nature". Notice the absence of "ordering" in the definition.

Notwithstanding the above definition, sorting usually implies the existence of an ordering. Orderings are fine for one-dimensional data. However, they do not exist in two dimensions and higher. For example, suppose we sort all of the cities in the US by their distance from Dallas. This is fine for finding the closest city to Dallas, say with population greater than 200,000. However, we cannot use the same ordering to find the closest city to Miami, say with population greater than 200,000, without resorting the cities. In contrast, once we sort with respect to a reference point in one dimension (e.g., people by their height), we can find the nearest value for any other reference point; there is no need to resort the data.

The problem is that for two dimensions and higher, the notion of an ordering does not exist unless a dominance relation holds—that is, a point $a = \{a_i | 1 \leq i \leq d\}$ is said to dominate a point $b = \{b_i | 1 \leq i \leq d\}$ if $a_i \leq b_i, 1 \leq i \leq d$. Thus the only way to ensure the existence of an ordering is to linearize the data as can be done, for example, using a space-filling curve as described in Section 2. The problem with such an approach is that the ordering is explicit. Instead, what is needed is an implicit ordering so that we don't have to resort the data when, for example in our sample query, the reference point for the query changes (e.g., from Dallas to Miami). Such an ordering is a natural byproduct when we sort objects by spatial occupancy, and is the subject of the rest of this section.

The indexing methods that are based on sorting the spatial objects by spatial occupancy essentially decompose the underlying space from which the data is drawn into regions called *buckets* in the spirit of classical hashing methods with the difference that the spatial indexing methods preserve order. In other words, objects in close proximity should be placed in the same bucket or at least in buckets that are close to each other in the sense of the order in which they would be accessed (i.e., retrieved from secondary storage in case of a false hit, etc.).

There are two principal ways of implicitly sorting spatial data. The first distinguishes occupied from unoccupied space, while the second sorts the regions so that the number of spatial objects that they contain is within the same range.

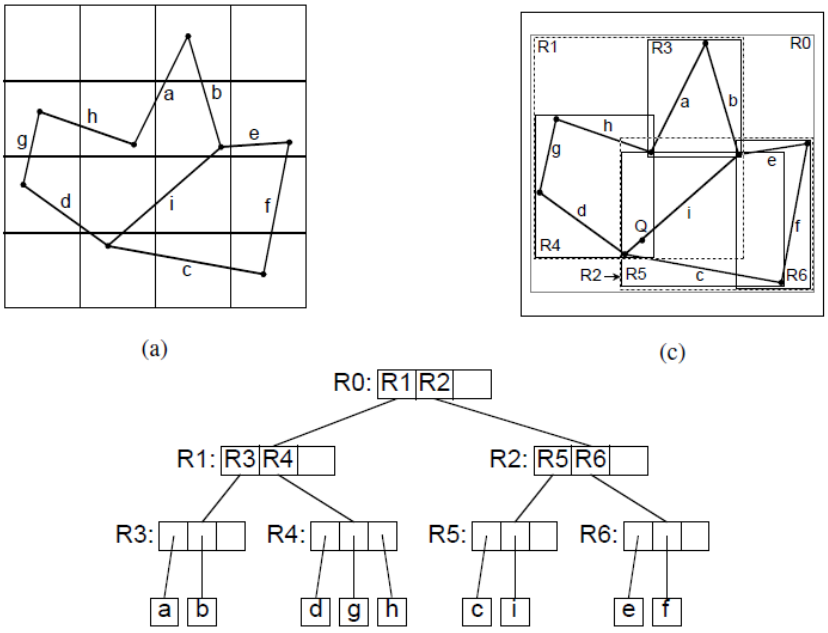

Figure 3: (a) Example collection of straight line segments embedded in a 4×4 grid, (b) the object hierarchy for the R-tree corresponding to the objects in (a), and (c) the spatial extent of the minimum bounding rectangles corresponding to the object hierarchy in (b). Notice that the leaf nodes in (c) also store bounding rectangles although this is only shown for the nonleaf nodes.

The first method makes use of an object hierarchy that initially aggregates objects into groups based on their spatial proximity and then uses proximity to further aggregate the groups thereby forming a hierarchy. Note that the resulting object hierarchy is not unique as it depends on how the objects were aggregated to form the hierarchy. Queries are facilitated by also associating a minimum bounding box of an appropriate shape (e.g., hyper rectangle, sphere) with each object and group of objects as this enables a quick way to test if a point can possibly lie within the area spanned by the object or group of objects. Negative answers mean that no further processing is required for the object or group, while a positive

answer means that further tests must be performed. Thus the minimum bounding box serves to sort the space according to the occupation status. Data structures such as the R-tree (Guttman et al. 1984) and the R*-tree (Beckmann et al. 1990) illustrate the use of this method.

As an example of an R-tree, consider the collection of straight line segment objects given in Figure 3(a) shown embedded in a 4×4 grid. Figure 3(b) is an example of the object hierarchy induced by an R-tree for this collection. Figure 3(c) shows the spatial extent of the bounding rectangles of the nodes in Figure 3(a), with heavy lines denoting the bounding rectangles corresponding to the leaf nodes, and broken lines denoting the bounding rectangles corresponding to the subtrees rooted at the nonleaf nodes.

The drawback of the object hierarchy approach is that the resulting hierarchy of bounding boxes leads to a non-disjoint decomposition of the underlying space. This means that if a an object is not found in one search path starting at the root, then it does not mean that the object will not be found in another search path starting at the root. This is the case in Figure 3(c) when we search for the line segment object that contains Q. In particular, we first visit nodes R1 and R4 unsuccessfully, and thus need to visit nodes R2 and R5 in order to find the correct line segment object i.

The second method is based on a recursive decomposition of the underlying space into disjoint blocks so that a subset of the objects associated with each block satisfies some predetermined criterion. There are several ways to proceed. The first is to simply redefine the decomposition and aggregation associated with the object hierarchy method so that the minimum bounding rectangles are decomposed into disjoint rectangles. This implicitly partitions the underlying objects that they bound. In this case, the partition of the underlying space is heavily dependent on the data and is said to be at arbitrary positions. The k-d-B-tree (Robinson 1981) and the R+-tree (Sellis et al. 1997) are examples of such an approach.

The second way is to partition the underlying space at fixed positions so that all resulting cells are of uniform size, which is the case when using the uniform grid, also the standard indexing method for maps. Figure 1(a) is an example of a 4×4 uniform grid in which a collection of straight line segments has been embedded. The drawback of the uniform grid is the possibility of a large number of empty or sparsely-filled cells when the objects are not uniformly distributed. This is overcome by using a variable resolution representation such as one of the quadtree variants (e.g., Samet 2006) where the subset of the objects that are associated with the blocks are defined by placing an upper bound on the number of objects that can be associated with each block (termed a *stopping condition* for the recursive decomposition process).

The PM_1 quadtree (Samet and Webber 1985, see also the related PMR quadtree Nelson and Samet 1987) is an example of a variable resolution representation for a collection of straight line segment objects such as the polygonal subdivision given in Figure 3(a). In this case, the stopping condition of its decomposition rule stipulates that partitioning occurs as long as a block contains more than one line segment unless the line segments are all incident at the same vertex which is also in the same block (e.g., Figure 4). A similar representation has been devised for three-dimensional images (e.g., Ayala et al. 1985) where no block contains more than one face, edge, or vertex unless the faces all meet at the same vertex or are adjacent to the same edge all in the same block. In this example, the PM_1 quadtree represents the polygonal subdivision by its constituent objects which are the line segments of arbitrary orientation. When the line segments are constrained to be orthogonal to the coordinate axes, the line quadtree (Samet and Webber 1984) uses a decomposition rule in terms of both the boundary and interior of the subdivision so that decomposition into blocks takes place until no boundary element

passes through the interiors of the blocks.

In the case of regions, the quadtree decomposition rule halts the decomposition once the data in the block is uniform. It has been used primarily in two and three dimensions where they are known as octrees (Meagher 1982), as well as for surface data (Sivan and Samet 1992).

In the case of point data, quadtrees enable the execution of many queries without having to perform explicit sorts of the data. They are especially useful for determining the nearest object to a particular location (i.e, a "pick" operation in computer graphics). In this case, the underlying space is recursively decomposed into four congruent square blocks until each block contains no more than a predetermined number of points (a variant of a bucket PR quadtree, Samet 2006). The advantage of the implicit sorting of the underlying space into the blocks is that the position of the subdivision lines vis-a-vis the position of the query object can be used to prune certain blocks from further consideration when executing the operation. Note that the same pruning properties also hold when the blocks are not congruent as in the case of the point quadtree (Finkel and Bentley 1989, Samet 1980). Quadtrees and their variants are to be distinguished from pyramids (e.g., Aref and Samet 1990, Tanimoto and Pavlidis 1975) which are multiresolution data structures.

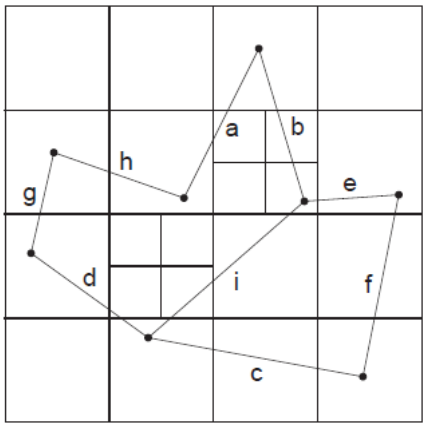

Figure 4: PM_1 quadtree for the collection of straight line segment objects of Figure 3(a).

The principal drawback of the 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. This means that queries such as those that seek the length of all objects in a particular spatial region will have to remove duplicate objects before reporting the total length. Nevertheless, methods have been developed to avoid these duplicates by making use of the geometry of the type of the data that is being represented (e.g., Aref and Samet 1992) to avoid multiple reporting of the same object. Another approach known as the MX-CIF represents each object just once by associating it with its minimum enclosing quadtree block. The loose quadtree (Kedem 1982) and the cover fieldtree (Frank and Barrera 1989) take this approach further by expanding the size of the minimum enclosing quadtree blocks so that the size of the expanded minimum enclosing quadtree block is not completely independent of the size of the object (Samet et al. 2013).

Note that the result of constraining the positions of the partitions means that there is a limit on the possible

sizes of the resulting cells (e.g., a power of 2 in the case of a quadtree variant). However, this means that the underlying representation is good for operations between two different data sets (e.g., a spatial join, Hoel and Samet 1995) often implemented as top-down tree traversals (Samet 1985), as their representations are in registration (i.e., it is easy to correlate occupied and unoccupied space in the two data sets, which is not easy when the positions of the partitions are not constrained as is the case with methods rooted in representations based on an object hierarchy even though the resulting decomposition of the underlying space is disjoint).

4 Concluding Remarks

An overview of the utility of sorting spatial data for use in geographic information systems has been provided. A distinction was made between the application to raster data and vector data. This is important as in the case of raster data, the sorting was based on ordering the actual data, while in the case of vector data, the sorting was more in terms of the underlying space in which the data is embedded. Hence we characterized the sorting as being explicit in the former and implicit in the latter. We pointed out that the sorting was useful in finding nearest objects. In this case, the distance was measured in terms of "as the crow flies". However, these methods are also useful in finding neighbors in road networks where the distance is defined along the network (Sankaranarayanan and Samet 2010). The sorting has also been used in a distributed peer-to-peer setting (Tanin et al. 2005)

Cross-References

Data structure, raster; Data structure, vector; Indexing; Spatial databases

Further Reading:

1. For an early work on spatial data structures, see H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, MA, 1990.

2. For more on executing operations using quadtrees and octrees, see H. Samet. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*. Addison-Wesley, Reading, MA, 1990.

3. For more about the different raster orderings as well as a comprehensive discussion of spatial data structures, see the discussion in (Samet 2006).

4. See http://donar.umiacs.umd.edu/quadtree/index.html for JAVA applets that illustrate a variety of spatial data structures including many of those describe here.

References

Aref, W. G. and Samet. 1990. Efficient processing of window queries in the pyramid data structure. In *Proceedings of the 9th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 265–272, Nashville, TN, April 1990.

Aref, W. G. and H. Samet. 1992. Uniquely reporting spatial objects: yet another operation for comparing spatial data structures. In *Proceedings of the 5th International Symposium on Spatial Data Handling*, pages 178–189, Charleston, SC, August 1992.

Ayala, D., P. Brunet, R. Juan, and I. Navazo. 1985. Object representation by means of nonminimal division quadtrees and octrees. *ACM Transactions on Graphics*, 4(1):41–59, January 1985.

Beckmann, N., H.-P. Kriegel, R. Schneider, and B. Seeger. 1990. 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.

Brabec, F. and H. Samet. 2007. Client-based spatial browsing on the world wide web. *IEEE Internet Computing*, 11(1):52–59, January/February 2007.

Dillencourt, M. B., H. Samet, and M. Tamminen. 1992. A general approach to connected-component labeling for arbitrary image representations. *Journal of the ACM*, 39(2):253–280, April 1992. Also see Corrigenda, *Journal of the ACM*, 39(4):985–986, October 1992.

Esperança, C. and H. Samet. 2002. Experience with SAND/Tcl: a scripting tool for spatial databases. *Journal of Visual Languages and Computing*, 13(2):229–255, April 2002.

Finkel, R. A. and J. L. Bentley. 1974. Quad trees: a data structure for retrieval on composite keys. *Acta Informatica*, 4(1):1–9

Frank, A. U. and R. Barrera. 1989. The Fieldtree: a data structure for geographic information systems. In *Design and Implementation of Large Spatial Databases—1st Symposium, SSD'89*, vol. 409 of Springer-Verlag Lecture Notes in Computer Science, pages 29–44, Santa Barbara, CA, July 1989.

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

Hoel, E. G. and H. Samet. 1995. Benchmarking spatial join operations with spatial output. In *Proceedings of the 21st International Conference on Very Large Data Bases (VLDB)*, pages 606–618, Zurich, Switzerland, September 1995.

Kedem, G. 1982. The quad-CIF tree: a data structure for hierarchical on-line algorithms. In *Proceedings of the 19th Design Automation Conference*, pages 352–357, Las Vegas, NV, June 1982.

Klinger, A. 1971. Patterns and search statistics. In *Optimizing Methods in Statistics*, pages 303–337. Academic Press, New York, 1971.

Meagher, D. 1982 Geometric modeling using octree encoding. *Computer Graphics and Image Processing*, 19(2):129–147, June 1982.

Nelson, R. C. and H. Samet. 1987. A population analysis for hierarchical data structures. In *Proceedings of the ACM SIGMOD Conference*, pages 270–277, San Francisco, May 1987.

Robinson, J. T. 1981. The K-D-B-tree: a search structure for large multidimensional dynamic indexes. In *Proceedings of the ACM SIGMOD Conference*, pages 10–18, Ann Arbor, MI, April 1981 Sagan, H. 1980 *Space-Filling Curves*. Springer-Verlag, New York, 1994.

Samet, H. 1980. Deletion in two-dimensional quad trees. *Communications of the ACM*, 23(12):703–710, December 1980.

Samet, H. 1985 A top-down quadtree traversal algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7(1):94–98, January 1985.

Samet, H. 2006. *Foundations of Multidimensional and Metric Data Structures*. Morgan-Kaufmann, San Francisco, 2006. (Translated to Chinese ISBN 978-7-302-22784-7).

Samet, H., H. Alborzi, F. Brabec, C. Esperança, G. R. Hjaltason, F. Morgan, and E. Tanin. 2003. Use of the SAND spatial browser for digital government applications. *Communications of the ACM*, 46(1):63–66, January 2003.

Samet, H., J. Sankaranarayanan, and M. Auerbach. 2013. Indexing methods for moving object databases: Games and other applications. In *Proceedings of the ACM SIGMOD Conference*, pages 169–180, New York, June 2013.

Samet, H. and R. E. Webber. 1984. On encoding boundaries with quadtrees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(3):365–369, May 1984.

Samet, H. and R. E. Webber. 1985. Storing a collection of polygons using quadtrees. *ACM Transactions on Graphics*, 4(3):182–222, July 1985.

Sankaranarayanan, J. and H. Samet. 2010. Query processing using distance oracles for spatial networks. *IEEE Transactions on Knowledge and Data Engineering*, 22(8):1158–1175, August 2010.

Sellis, T., N. Roussopoulos, and C. Faloutsos. 1987. The R^+-tree: a dynamic index for multi-dimensional objects. In *Proceedings of the 13th International Conference on Very Large Databases (VLDB)*, pages 71–79, Brighton, United Kingdom, September 1987.

Shaffer, C. A., H. Samet, and R. C. Nelson. 1990. QUILT: a geographic information system based on quadtrees. *International Journal of Geographical Information Systems*, 4(2):103–131, April–June 1990.

Sivan, R. and H. Samet. 1992. Algorithms for constructing quadtree surface maps. In *Proceedings of the 5th International Symposium on Spatial Data Handling*, vol. 1, pages 361–370, Charleston, SC, August 1992.

Tanimoto. S. L. and T. Pavlidis. 1975. A hierarchical data structure for picture processing. *Computer Graphics and Image Processing*, 4(2):104–119, June 1975.

Tanin, E. 2005. A. Harwood, and H. Samet. A distributed quadtree index for peer-to-peer settings. In *Proceedings of the 21st IEEE International Conference on Data Engineering*, pages 254–255, Tokyo, Japan, April 2005.

Keywords: sorting, spatial indexing, space-filling curves.