Efficient Visualization of Graphical Objects

PAULA FREDERICK, MARCELO GATTASS, MAURICIO RIGUETTE MEDIANO

TeCGraf – Grupo de Tecnologia em Computação Gráfica, Departamento de Informática, PUC-Rio,
Rua Marquês de São Vicente, 225, 22453-900 Rio de Janeiro, RJ, Brasil
paula,gattass,mediano@tecgraf.puc-rio.br

Abstract. The use of appropriate data structures for storing sets of graphical objects can lead to great performance improvements in the visualization of large figures, such as maps and CAD drawings. We present here a study on the performance of persistent data structures composed by an R-tree and V-trees, for storing and efficiently retrieving 2D graphical objects. Results are shown to demonstrate the efficiency of the proposed solutions when applied to large maps.
Keywords: GIS, Metafile, Access Methods.

1 Introduction

Over the last decade, systems which use huge quantities of spatial data have been increasingly gaining importance. Examples of such systems can be found in geographic information systems (GIS), CAD/CAM, and others. One of the requirements of those systems is to provide efficient visualization of their spatial data, enabling the users to interactively visualize all or part of the data, in all scales.

In computer graphics the conventional method for drawing spatial objects is to treat them as a simple collection of graphical objects stored either in metafiles, linked lists, or acyclic trees. The visualization process traverses the file or data structure and for each object the program applies the appropriate geometric transformation, clips it with the visualization window, and renders it into the visualization surface. However, as data size and complexity grow, this process becomes very slow, and new techniques are required for dealing with this large data volumes. The following issues have to be considered in the visualization of large spatial data.

- *Memory usage* - because of the data size, it is very likely that the whole data will not fit into the computer's main memory. Thus the need for a secondary storage memory, which implies in much larger access time. Consequently it is important to have a good memory management subsystem in order to minimize the I/O communication.

- *Scene clipping* - the user will often visualize only part of the data. Thus we can decrease visualization time by only retrieving the objects that are inside the visualization area. Objects that are outside this area should be left untouched in the secondary storage memory.

- *Level of detail* - when the data visualization scale is large, the details of the scaled data which are smaller than the current resolution will not be drawn into the visualization surface. Therefore, the simplification of the graphic objects considering the current drawing resolution leads to performance gains.

Areas where database technologies play an important role, such as GIS and CAD, have long recognized the importance of the first two points. In computer graphics, these points are becoming more relevant, as one wishes to render complex scenes such as terrains, cities, building interiors, molecules, and biological structures as quickly as possible [17, 18].

The third point is treated as cartographic simplification in digital cartography, and as level of detail in computer graphics (see Paul Heckbert's SIGGRAPH '97 Lecture Note [6] for a good review on the subject).

In this paper we initially restrict our focus to two-dimensional graphical vector objects composed by graphical primitives such as polygonal lines, closed polygons, text, and point objects. Two-dimensional objects are often complex either because they are composed of many graphical primitives or because their primitive has too many points. In the first case, complexity can be treated with a hierarchical structure based on the bounding box of the primitives, and in the second case it can be treated by some line simplification scheme. Thus, with the use of an adequate data structure for storing the spatial data, we can expect to obtain considerable performance gains. In this paper we describe the use of a persistent data structure composed by an R-tree [5, 15, 2] and V-trees [11], for storing and efficiently retrieving 2D spatial data for their visualization.

Spatial objects are either two-dimensional or three-dimensional and can be represented by either vector or raster graphical objects. In this paper we have only considered two-dimensional graphical vector objects; raster and three-dimensional objects are left for a future work. Our primary goal when developing the techniques pre-

sented here was to visualize large maps. The method proposed is, however, directly applicable to any kind of two-dimensional figure.

To test our ideas we have developed the Drops (*Drawing Objects on Persistent Systems*) system, which stores a general 2D vector figure in a file, and afterwards retrieves portions of data for visualization. Tests were performed comparing our scheme with the use of a conventional CGM (ISO Computer Graphics Metafile [8]) driver. Results are shown to support some conclusions.

This paper is structured as follows. The next section describes how the graphical objects are represented. In the third section we provide some concepts about spatial access methods. In the fourth section we present the persistent storage subsystem used by Drops. Section five describes the methods to store and retrieve data. Finally, we describe the experiments done with geographic data, the results and the conclusions.

2 Graphical Primitives

Two-dimensional graphical objects are represented by a set of primitives, which are like the geometric alphabet of figures. Here these primitives are markers (points), lines, polylines (polygonal lines), filled polygons, rectangles, arcs, filled sectors, vector texts, and scalable raster texts. These are the basic graphical primitives of any 2D graphical library, such as Windows GDI or XLib.

In the geometric sense, two-dimensional graphical objects are simpler than 3D objects, but in the description of their appearance usually the opposite occurs. Thick polygonal lines have joint style, and texts have many typographic attributes.

As usually the attributes of a drawing do not vary as much as the geometric descriptions of the primitives do, and the graphical system has basically two approaches to deal with this: (1) current values, and (2) tables (sometimes called objects). In the current value approach the graphical system stores a set of values for each aspect of the primitives, and these primitives are only defined in geometric terms. The color, style, and other appearance attributes are taken from the current value in the set. Attribute functions are used to change these values for the forthcoming primitives.

In the table approach a given appearance of a type of primitive is stored as a record in a table. There are tables for each type of attributes: pen tables (line attributes), brush tables (interior style of regions), and text tables. Each primitive has a pointer for its attribute in the corresponding table.

The current value approach is simpler to use and yields a smaller description of the figure, once the appearance of many primitives will be only implicitly defined. It depends, however, on the order in which the primitive occurs in the figure description's traversal. This constitutes a problem for spatially indexed graphical objects, as we will discuss later in this paper.

The line attributes used here are line color, line style, line width, and write mode (replace or xor). The primitives affected by this set of attributes are line, polyline, and arc. The area attributes are: foreground color, background color, write mode, opacity, and interior style. Depending on the interior style the primitive may be filled with a stipple or a texture. Primitives box, polygon, and sector have this set of attributes. Finally, the text attributes describe text characteristics. These attributes are: foreground color, background color, write mode, opacity, typeface, style, and size. As pointed out before, two-dimensional objects can have a much more detailed description of their appearance, but these attributes are enough to the objectives of this paper and can yield figures as complex as the one shown in Figure 1.

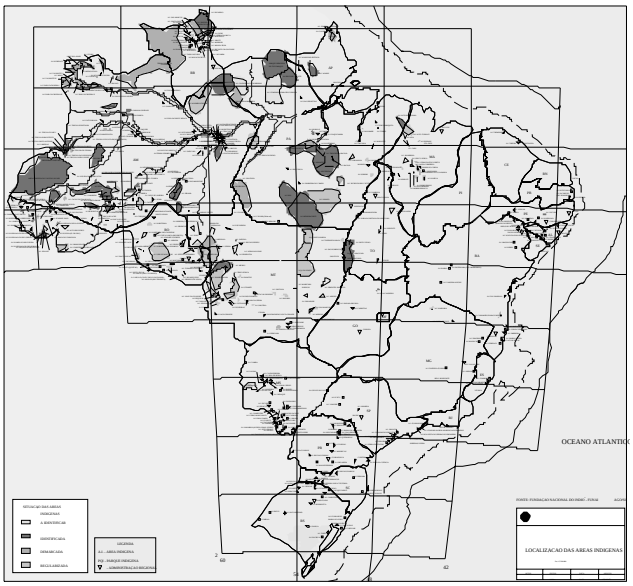

Figure 1: Indian Areas in Brazil

3 Spatial Access Methods

Several works in the field of spatial access methods were proposed over the last years: Z-order [13] transforms points from n-dimensional space into points in one-dimensional space, subsequently indexing them using B-trees; R-trees [5, 15, 2] and Grid-files [12] index points and rectangles, bringing each spatial object closer to its bounding box, and, when the bounding box is not enough to solve a spatial operation, recovering the object's complete geometry.

Two access methods were proposed to index the complete geometry of spatial objects: TR-tree [14, 3], and the V-tree family [11]. The TR-tree indexes a region by means

of a set of trapezoids stored in an R-tree, in which the region's holes are already represented. The V-tree family indexes polygonal lines forming the borders of spatial objects, optimizing geometrical and multiresolution operations.

Strip-tree [1] and Arc-tree [4] are binary trees which store the geometry of polygonal lines, and optimize geometrical and multiresolution operations. The V-trees can be seen as variants of Strip-tree and Arc-tree adequate for storing and manipulating large amounts of vector data in a persistent environment.

4 Persistence

We have used the persistent-object storage subsystem OPS [10]. This system provides an extensible and configurable model through which it is possible to model, implement, and combine different database technologies to provide new capabilities or to obtain performance improvements.

OPS is based on four abstractions: persistent object, object data area, object manager, and media. The OPS subsystem is used to allocate persistent memory, through the media and the object manager, to check in and check out graphic primitives, attribute tables, and R-tree and V-tree nodes. It also manages a buffer layer that maintains in primary memory the recently used object data areas.

5 Drops

The Drops persistent-object manager is the OPS subsystem. It is responsible for handling all the data access made to the secondary storage media. The purpose of its utilization is to minimize of the I/O communication, archived by the memory usage management.

The proposed data structure used by Drops for storing large sets of 2D graphical objects is formed by a composition of a few spatial access methods. Its primary structure is the R-tree [5]. It has the function of spatially indexing the whole object set, enabling the user to retrieve only the objects whose bounding boxes intersect or are inside a given rectangular area. Given the static nature of the drawings we use here a variant of the R-tree called Hilbert Packed R-tree [9].

The remaining access methods, which also form the data structure, are used just for storing complex graphical objects. They store these objects in a multiresolution scheme in order to enable the retrieval of a simplified version of the object. There is a specific access method for each kind of object. In the case of vector objects, only the polylines and the filled polygons are usually complex. When these objects types are complex, Drops stores them in V-trees [11], which index their geometry.

Multiresolution can also be provided by R-trees. However, the R-tree multiresolution is a bit different: rather than approximating the object itself it just approximates an entire subset of objects by its bounding box. In order to add this functionality into the R-tree, we have added a color field to each entry of the tree nodes with the purpose of storing the appearance description of the object subset contained in the entry's sub-tree. This additional field forms together with the entry's bounding box a filled rectangle which is the simplified scheme of the corresponding subset. During the region query, the sub-trees whose bounding boxes fit inside a small given region (for example, a pixel) are not visited. For the appearance description we used just the color which is an essential attribute, but the inclusion of more attributes is trivial. Also for simplicity, we pick the color of one of the objects in the subset to fill the rectangle.

For the visualization process Drops enables the two multiresolution methods to be turned on and off. Notice that, if the V-trees multiresolution is never turned on, the use of V-trees for storing the data will be completely unnecessary.

Since Drops does not store the objects in a sequential manner, the original order of the objects is lost when the data is converted from a sequential format. However, in the cases in which object overlapping occurs, the order in which the objects are drawn affects the resulting image, as is shown in Figure 2.

There are two strategies listed in the litarature to solve the object overlap problem. The first is based on the R^+-tree [15] and uses clipping to avoid this problem. The second strategy is based on multilayer techniques [16, 7]. Drops provides two solutions to this problem: object ordering, and layers.

In the first case, each object receives a counter value indicating its original order. This counter is stored together with the object pointer. When the data is read, first only the object counters and pointers are retrieved. Then they are sorted, according to their original order, with the *Quick Sort* algorithm. Afterwards the entire object descriptions are retrieved.

The second solution consists on the use of several layers for storing the objects. The data set is divided into layers of disjoint objects, thus when the layer order is kept the objects will be drawn correctly. Drops treats this concept by using one R-tree for each layer, however there are some access methods which already treats these concepts. The main advantage of this solution is that there is no need to sort a huge amount of data. However, the application will have to correctly separate the data into a small number of layers, which sometimes can be impossible.

Concerning the object attributes it is important to notice that the current value approach cannot be used by Drops because it does not store the objects sequentially. In order to save space, we use the table approach to store a reference to the attributes instead of storing the attributes themselves

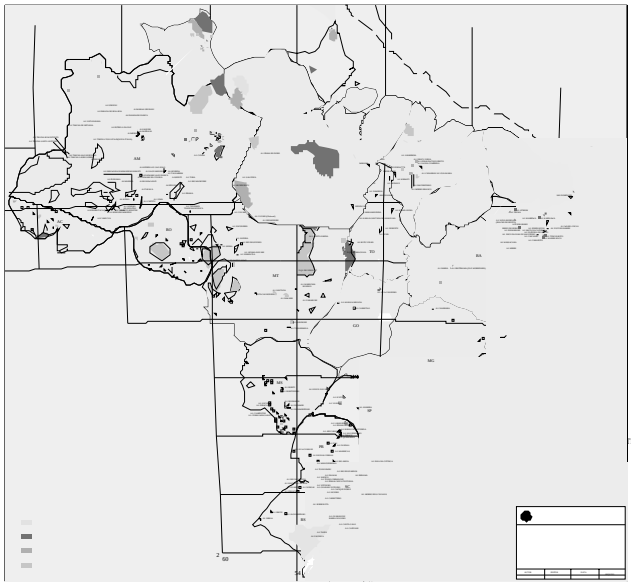

Figure 2: This is the same map from Figure 1, but the primitives were drawn in the order in which they were retrieved from the R-tree.

with the objects' geometry. Furthermore, we implemented an algorithm based on hash files to store uniquely each attribute value in the attribute table. That is, the same set of attributes will never be stored more than once in the file, thus yielding small attribute tables.

6 Experiments

For testing the performance of our method we have designed two sets of experiments. The first one aims at testing the clipping capability; To archieve that, we zoom in the data several times. The second set seeks to test the multiresolution feature of our approach.

For all the experiments we had to define which part of the figure (window) would be shown in the visualization surface. To avoid window system interference in our measurements the visualization surface chosen was a file in the CGM format. For comparison with conventional vector file format we have also tested the use of the CGM file to produce the same picture. That is, we read and write the same CGM file.

6.1 Data

The experiments were performed with four different cartographic data, which are illustrated in Figures 1, 3, 4, and 5. In Table 1 we list some characteristics of these data. The order of the tree-shaped data structures used by Drops for storing the experimental data sets were of $(20, 50)$ for the R-trees and of $(5, 10)$ for the V-trees.

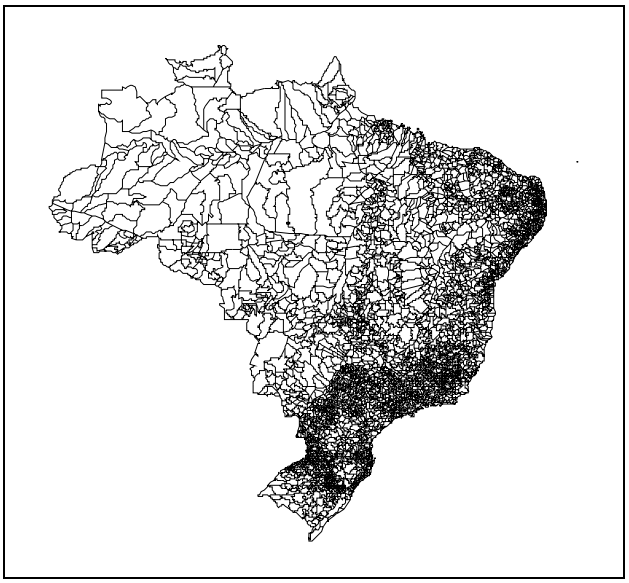

Figure 3: Municipal Districts of Brazil

Looking at Table 1 we can make some important observations. The last three data sets, which are the Municipal Districts, the Asian Continent, and the Madureira Buildings, are large in size. However, in terms of diversity of attribute and primitive types they are very simple, being composed just of black polylines or polygons. On the other hand, the map in Figure 1 is small, but it has many types of attributes and primitives. It is also important to notice that the Madureira Buildings data set has a very small average of vertices per polygon.

As far as the size of the files is concerned, note that in the CGM file format used here the point coordinates are stored in a short integer type that uses only 2 bytes. Our format stores the same data in a 32-bit floating point, thus allowing greater precision.

From Table 1, we can observe that the size of our format is more than twice as large as the CGM file. The main reason for this difference is the extra memory required for storing the R-tree and V-tree data structures. Note, however, that our scheme uses secondary memory, and the main focus here is to obtain fast visualization time.

6.2 Results

Zoom in these experiments was performed by incrementally scaling the data by a factor of 1.1, and displaying them on a viewing area of dimensions 700×500. This way, the part of the data that is visible becomes smaller at each step. Each experiment was computed two times, one ordering and the other without ordering the graphical primitives. In Figures 6, 7, 8, and 9 we plot the elapsed time versus the percentage of the viewing area being displayed.

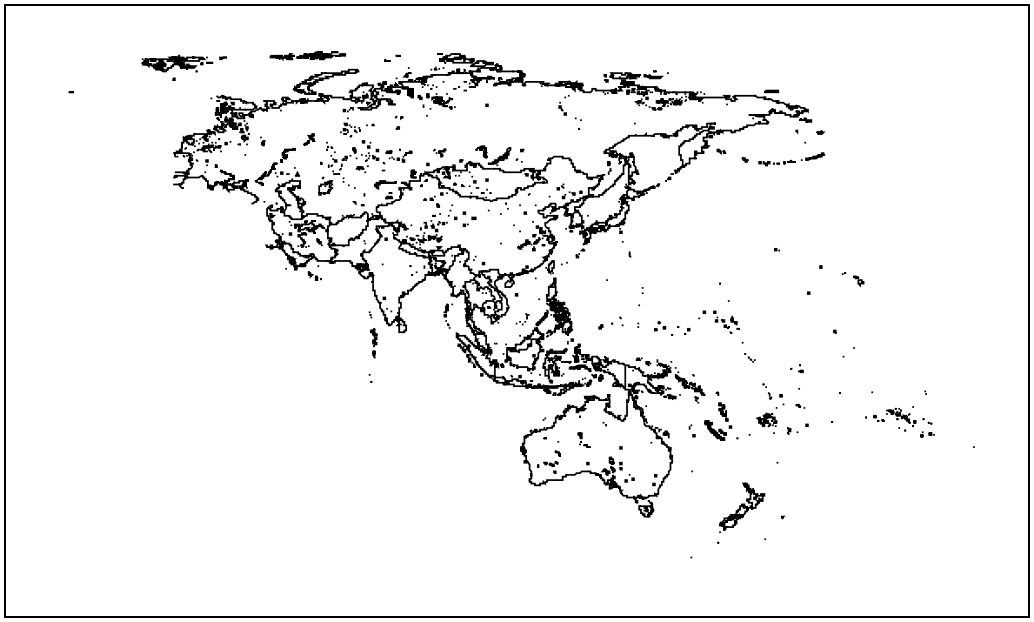

Figure 4: Asia

(a) Complete Data

(b) Zoom over a small area of the data

Figure 5: Madureira Buildings

Data	Number of Objects	Average of Vertices per Polygonal	Number of Attributes	Size (Kb)		
				CGM	Drops *without V-trees*	Drops *with V-trees*
Indian Areas	2243	111	31	829	1400	2060
Municipal Districts	16596	43	1	3025	6220	10884
Asia	9533	108	1	4174	8412	12840
Madureira Buildings	63062	8	1	2858	6264	9452

Table 1: Data Characteristics

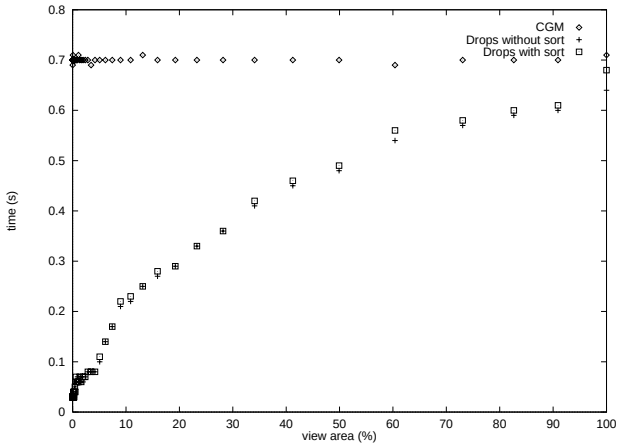

Figure 6: Result of the scaling experiments on the Indian Areas Map.

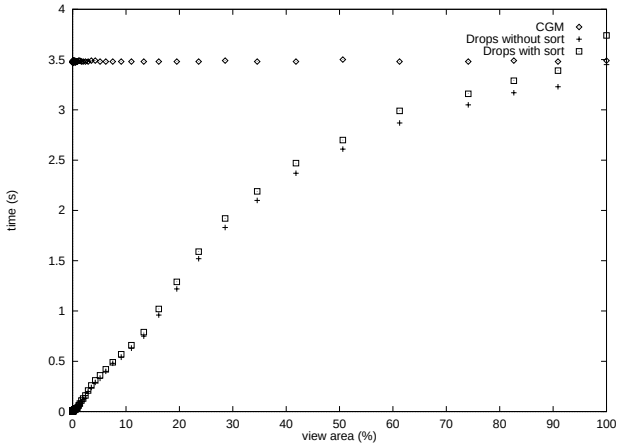

Figure 7: Result of the scaling experiments on the Municipal Districts Map.

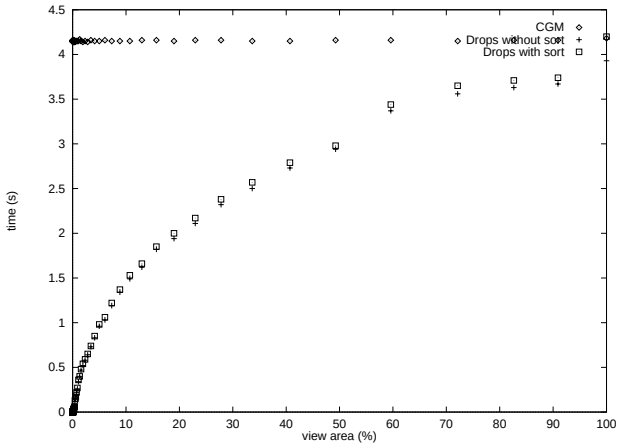

Figure 8: Result of the scaling experiments on the Asia Map.

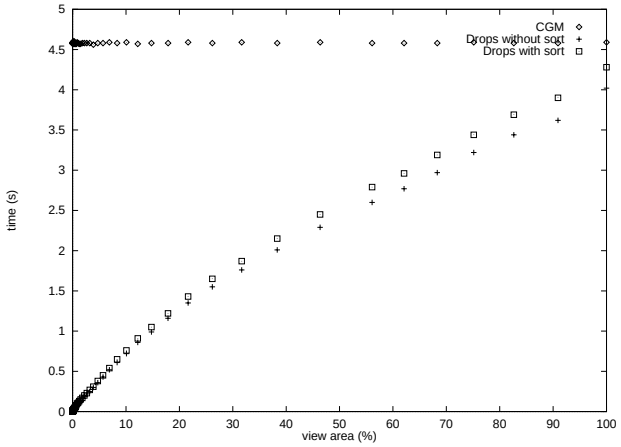

Figure 9: Result of the scaling experiments on the Madureira Buildings.

Through the graphs shown in Figures 6, 7, 8, and 9 we notice that the ordering of the primitives, required to maintain the original overlap order of the objects, does not significantly affect the drawing's efficiency. As the window size decreases (greater zoom) the use of our strategy becomes more relevant. Furthermore, when drawing the entire figure, the Drops system does not worsen the display time. The visualization time of the CGM file remains constant regardless of the fact that just a small part of the data is visible. It is also important to notice that the size of the data did not considerably affect the results.

The second set of experiments aims at testing the proposed multiresolution scheme. In all examples shown below we have drawn a window containing all the data. Just like in the previous examples, we have also used here a CGM file for our display surface to avoid window system interference. Multi-resolution, however, has more meaning if the display surface has a pixel size. To determine the pixel size in the example shown below we have considered a canvas with a resolution of 700×500.

Figures 10, 11, 12, and 13 show the elapse time versus the tolerance of the multiresolution scheme. This tolerance is represented in the upper horizontal axis as the number of pixels in the 700×500 resolution. That is, Drops draws the pixels with the color assigned when the bounding box becomes smaller than the tolerance. This is true for both the R-tree and the V-tree. The lower horizontal axis shows a percentage of the tolerance window relative to the total drawing area.

From the graphs shown in Figures 10, 11, 12, and 13, we can observe significant performance improvements even for small tolerances (1 pixel). However, as the tolerances increase the gains in performance do not increase proportionally.

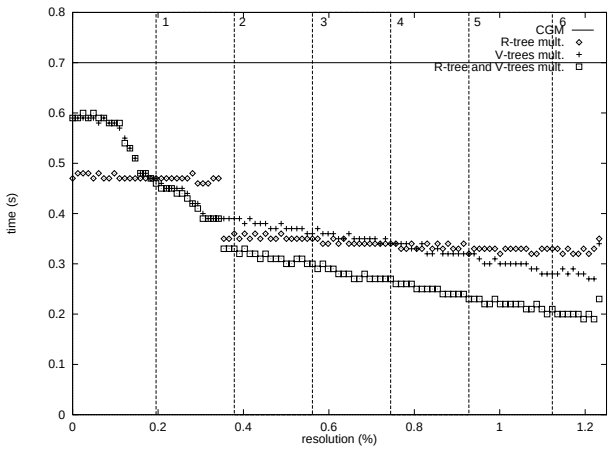

Figure 10: Result of the multiresolution experiments on the Indian Areas Map.

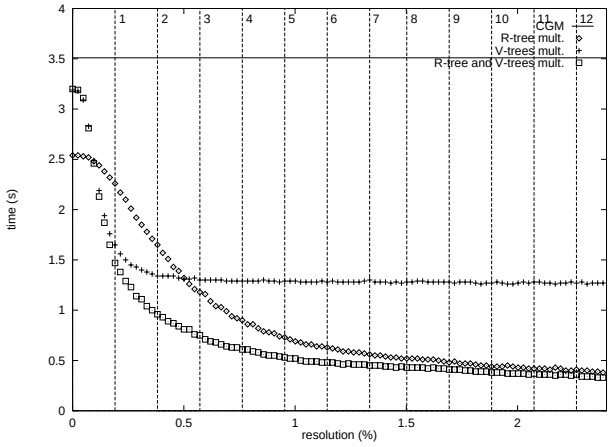

Figure 11: Result of the multiresolution experiments on the Municipal Districts Map.

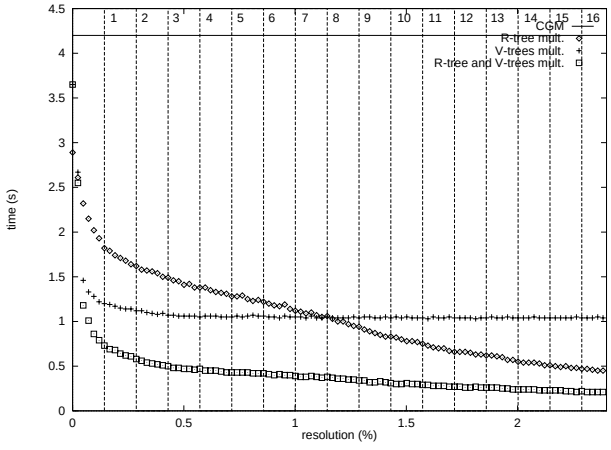

Figure 12: Result of the multiresolution experiments on the Asia Map.

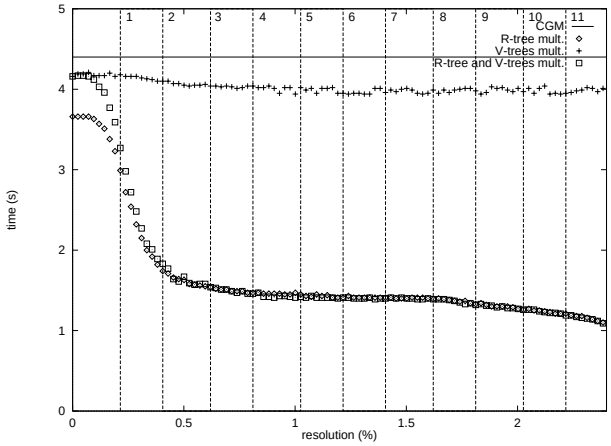

Figure 13: Result of the multiresolution experiments on the Madureira Buildings.

Finally, Figures 14, 15, 16, and 17 show the combined effect of multiresolution and zoom in the data sets. Note that, as expected, for large windows the multiresolution prevails, and, conversely, for small windows (large zooms) the clipping becomes more relevant.

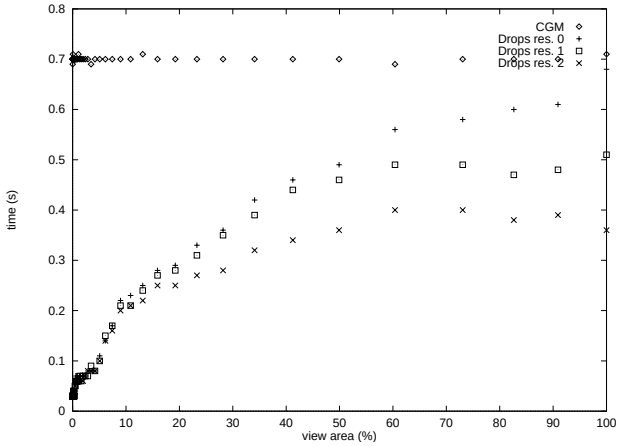

Figure 14: Combined effects of zoom and multiresolution for the Indian Areas Map.

7 Conclusions

The results presented in the previous sections allow us some general conclusions.

First, the proposed method yields an efficient visualization scheme for large and complex spatial data such as maps. The fact that it uses secondary memory suggests the use of this strategy for figures in CD-ROMs.

The main advantage of using the persistent storage subsystem to manage memory usage is the ability to retrieve data from the secondary memory only on demand.

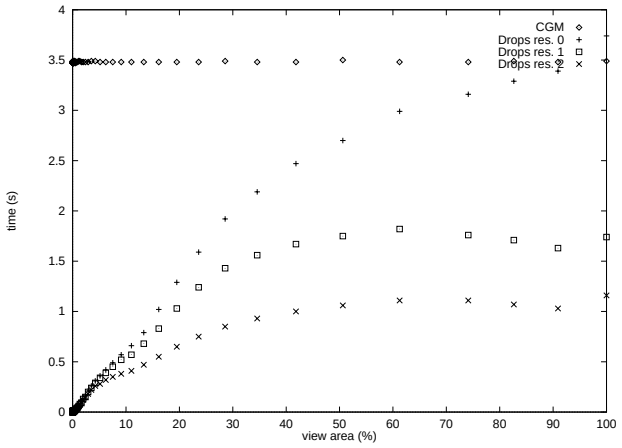

Figure 15: Combined effects of zoom and multiresolution for the Municipal Districts Map.

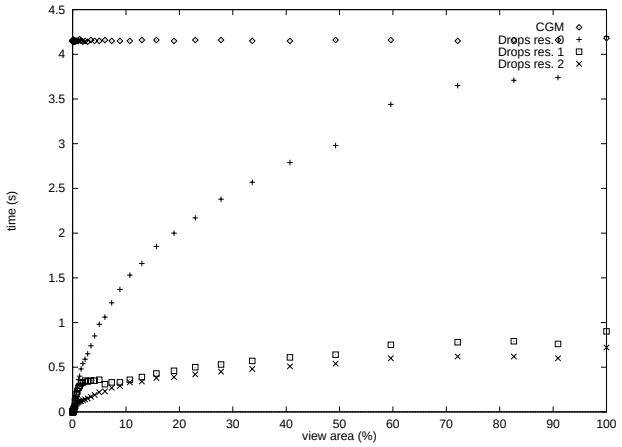

Figure 16: Combined effects of zoom and multiresolution for the Asia Map.

This way the applications do not get extremely slow when the data does not fit into the main memory and the operating system starts to swap memory to disk.

Second, the use of the Packed R-tree for indexing spatial data by their bounding boxes with the heuristics presented in [9] yields a good hierarchical structure for visualization. Groups of objects that are close together become adjacent in the R-tree. If they are outside the visualization window they are not even retrieved from the secondary memory, thus saving time that, otherwise, would be spent clipping them.

Acknowledgements

We thank Newton C. Sanches for his contribution. This work was developed in TeCGraf/PUC-Rio and was partially funded by CAPES, by means of fellowships, and by the PROTEM/CC-GEOTEC project. TeCGraf is a Laboratory mainly funded by PETROBRAS.

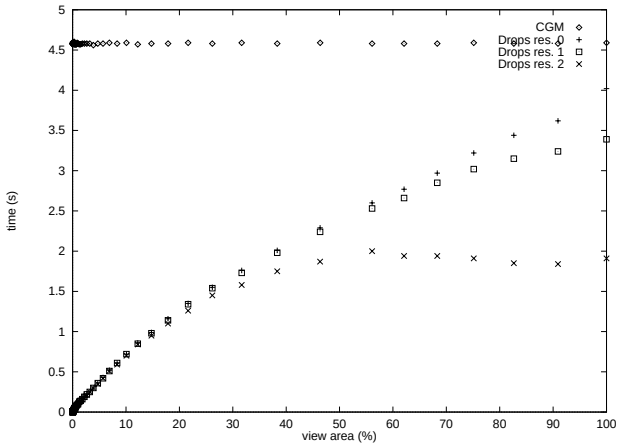

Figure 17: Combined effects of zoom and multiresolution for the Madureira Buildings.

References

[1] D. H. Ballard. Strip trees: Hierarchical representation for curves. *Communications of the ACM*, 24(5):310–321, 1981.

[2] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 322–332, May 1990.

[3] Thomas Brinkhoff, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. Multi-step processing of spatial joins. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 197–208, May 1994.

[4] O. Günther and E. Wong. The arc tree: an approximation scheme to represent arbitrarily curved shapes. *Computer Vision, Graphics and Image Processing*, 51(3):313–337, 1990.

[5] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the ACM SIGMOD Conference on Data Engineering*, pages 47–56, 1984.

[6] Paul Heckbert. Multiresolution surface modeling. In *SIGGRAPH '97, Course Notes n. 25*, 1997.

[7] A. Hutflesz, H. Six, and P. Widmayer. The r-file: An efficient access structure for proximity queries. In *Proceedings of the Sixth IEEE International Conference on Data Engeneering*, pages 372–379, 1990.

[8] ISO/IEC 8632:1992. *Information Technology - Computer Graphics - Metafile for the Storage and Transfer of Picture Description Information*, 1992. Part 1 - Functional Specification (ISO8632-1), Part 2 - Character Encoding (ISO8632-2), Part 3 - Binary Encoding (ISO8632-3), Part 4 - Clear Text Encoding (ISO8632-4).

[9] I. Kamel and C. Faloutsos. On packing R-trees. In *Proceedings of the 2nd International Conference on Information and Knowledges Management (CIKM-93)*, pages 490–499, November 1993.

[10] Maurício Riguette Mediano. OPS: Um subsitema extensível e configurável de armazenamento de objetos persistentes. Technical Report MCC37/97, Departamento de Informática, Pontifícia Universidade Católica, Rio de Janeiro, 1997.

[11] Maurício Riguette Mediano, Marco Antônio Casanova, and Marcelo Dreux. V-tree - a storage method for long vector data. In *Proceedings of the 20th VLDB Conference*, 1994.

[12] J. Nievergelt, H. Hinterberguer, and K. C. Sevick. The grid file: An adaptable, symmetric multikey file structure. *ACM Transactions on Database Systems*, 9(1):38–71, March 1984.

[13] J.A. Orenstein. Spatial query processing in an object-oriented database system. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 326–336, May 1986.

[14] Ralf Schneider and Hans-Peter Kriegel. The TR*-tree: A New Representation of Polygonal Objects Supporting Spatial Queries and Operations. In *Proceedings of 7th Workshop on Computational Geometry*, pages 507–518, 1991.

[15] Timos Sellis, Nick Roussopoulos, and Christos Faloutsos. The R$^+$-Tree: A Dynamic Index for Multi-Dimensional Objects. In *Proceedings of the 13th VLDB Conference*, pages 507–518, September 1987.

[16] H. Six and P. Widmayer. Spatial searching in geometric databases. In *Proceedings of the Fourth IEEE International Conference on Data Engeneering*, pages 496–503, 1988.

[17] Seth J. Teller. Visibility preprocessing for interactive walkthroughs. *Computer Graphics*, 25(4):61–69, July 1991.

[18] Seth Jared Teller. *Visibility Computations in Densely Occluded Polyhedral Environments*. PhD thesis, Computer Science Department University of California at Berkeley, 1992.