(Revised) Paper for GISDATA Specialist Meeting on Generalization,

Compiègne, France, 15–19 December 1993.

The GAP-tree, an approach to "On-the-Fly" Map Generalization of an Area Partitioning

Peter van Oosterom
TNO Physics and Electronics Laboratory,
P.O. Box 96864, 2509 JG The Hague, The Netherlands,
Email: oosterom@fel.tno.nl.

This paper presents solutions for "on-the-fly" map generalization for a volatile display of geographic data at an arbitrary scale without storing multiple representations. Only one, possibly extremely large, geographic data set is used together with a number of data structures, which enable fast interactive selection and simplification. Special attention is payed to area partitioning as used in, for example, a thematic choropleth map. In order to avoid gaps in the map when omitting less important area features on a small-scale map, a new data structure is designed: the GAP-tree. Together with two other data structures, the Reactive-tree and the BLG-tree, the GAP-tree enables fast generalization of an area partitioning. The proposed structures have been implemented in the Postgres DBMS environment, which together with the geographic frontend GEO++ forms an extensible GIS. Performance measurements on the 60 Mb DLMS DFAD test data set indicate that the original response times of about 2-6 minutes (on a Sun SPARCstation II) can be reduced to response times of about 6-9 seconds over a large range of scales.

1 Introduction

The concept of *on-the-fly* map generalization is very different from the implementation approaches described in the paper of Müller et al.: *batch* and *interactive* generalization [13]. The term batch generalization is used for the process in which a computer gets an input data set and returns an output data set using algorithms, rules, or constraints [11] without the intervention of humans. This in contrast to interactive generalization ("amplified intelligence") in which the user interacts with the computer to produce a generalized map.

On-the-fly map generalization does not produce a second data set, as this would introduce *redundant* data. It tries to create a temporary generalization, e.g., to be displayed on the screen, from one detailed geographic database. The quick responses, required by the interactive users of a GIS, demand the application of specific data structures, because otherwise the generalization would be too slow for reasonable data sets. Besides being suited for map generalization, these data structures must also provide spatial properties; e.g., it must be possible to find all objects within a specified region efficiently. The name of these types of data structures is *reactive data structures* [20, 21, 22].

As will be discussed in Section 2, an area partitioning posseses some special problems, when being generalized. In order to avoid gaps when not selecting small area features, a special structure is proposed: the GAP-tree. Section 3 describes two other reactive data structures, which will be used in combination with the new GAP-tree: the Reactive-tree and the BLG-tree. The implementation and test results are given in Section 4, where both visual and numerical results are shown. Finally, conclusions and future work are summarized in Section 5.

2 The GAP-tree

This section first describes an area partitioning as a map basis, and the problems encountered when generalizing this type of map. In Subsection 2.2 the key to the solution is introduced: the area partitioning hierarchy. The creation of the structure that supports this hierarchy, the GAP-tree, is outlined in Subsection 2.3. Operations on the GAP-tree are described in the last subsection.

2.1 Problems with an Area Partitioning

An area partitioning[1] is a very common structure used as a basis for many maps; e.g. choropleths. Two problems occur when using the following generalization techniques:

1. *Simplification:* Applying line generalization to the boundaries of the area features might result in ugly maps because two neighboring area features may now have overlaps and/or gaps. A solution for this problem is to use a topological data structure and apply line generalization to the common edges.

2. *Selection:* Leaving out an area will produce a map with a hole which is of course unacceptable. No obvious solution exists for this problem.

A solution for the second (and also for the first) problem is presented. It is based on a *novel* generalization approach called the *Area Partitioning Hierarchy*, which can be implemented efficiently in a tree structure.

[1]In an area partitioning each point in the 2D domain belongs to exactly one of the areas (polygons). That is, there are no overlaps or gaps.

2.2 Area Partitioning Hierarchy

The gap introduced by leaving out one area feature must be filled again. The best results will be obtained by filling the gap with neighboring features. This can be easy in case of so called "islands": the gap will be filled with the surrounding area, but it may be more difficult in other situations.

A basic notion is the *importance* of an area feature a, which is a function of its size and type (or role): $I(a) = f(\text{size}, \text{type})$. The size could be measured by its area $A(a)$ or perimeter $P(a)$. The weight of the type of the feature depends on the application. For example, a city area may be more important than a grassland area in a given application. This could be described with the following importance function: $I(a) = A(a) * \text{weight\_factor}(a)$.

By taking both the spatial relationships and the importance of an area feature into account, an *area partitioning hierarchy* is created. This hierarchy is used to decide which area is removed and also which other area will fill the gap of the removed feature.

2.3 Building the GAP-tree

The polygonal area partitioning is usually stored in a topological data structure with nodes, edges, and faces. The process, described below, for producing the area partitioning hierarchy assumes such a topological data structure [3, 4, 12, 14]; see Figure 1. The topological elements have the following attributes and relationships:

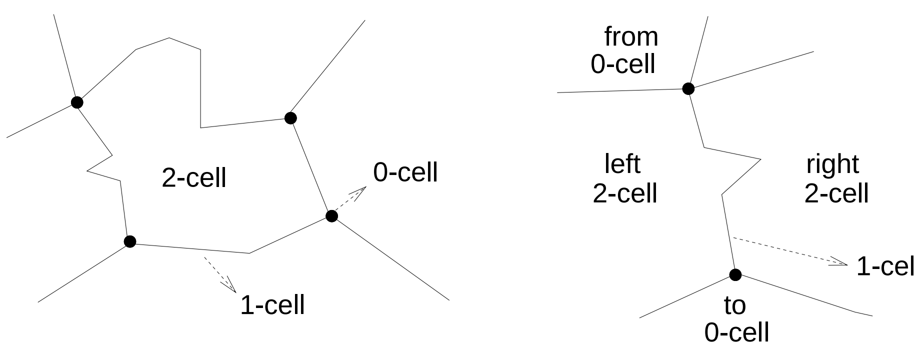

Figure 1: The topological data structure

- a *node* (or 0-cell) contains its point and a list of references to edges sorted on the angle;
- an *edge* (or 1-cell) contains its polyline, length and references to the left and to the right face;
- a *face* (or 2-cell) contains its weight_factor, area, and a list of sorted and signed references to edges forming the outer boundary and possibly inner boundaries;

Note that this topological data structure contains some redundant information because this enables more efficient processing later on. After the topological data structure has been created, the following steps will produce a structure, called the *Generalized Area Partitioning* (GAP)-tree, which stores the required hierarchy:

1. For each face in the topological data structure an "unconnected empty node in the GAP-tree" is created;

2. Remove the least important area feature a, i.e., with the lowest importance $I(a)$, from the topological data structure;

3. Use a topological data structure to find the neighbors of a and and determine for every neighbor b the length of the common boundary $L(a, b)$

4. Fill the gap by selecting the neighbor b with the highest value of the collapse function: $\text{Collapse}(a, b) = f(L(a, b), \text{CompatibleTypes}(a, b), \text{weight\_factor}(b))$. The function CompatibleTypes(a, b) determines how close the two feature types of a and b are in the feature classification hierarchy associated with the data set [1, 4, 5]. For example, the feature types "tundra" and "trees" are closer together than feature types "tundra" and "industry" in DLMS DFAD [4];

5. Store the polygon and other attributes of face a in its node in the GAP-tree and make a link in the tree from parent b to child a;

6. Adjust the topological data structure, importance value $I(b)$, and the length of common boundaries $L(b, c)$ for every neighbor c of the adjusted face b to the new collapsed situation.

Repeat the described steps 2–6 until all features in the topological data structure are at the required importance level (for a certain display operation). This procedure is quite expensive and probably too slow for large data sets to be performed on-the-fly. Therefore, the hierarchy is pre-computed and stored in the GAP-tree. The 2–6 steps are now repeated until only one huge area feature is left, because we can not know what the required importance level will be during the interactive use in a GIS. The last area feature will form the root of the GAP-tree. Further, a priority queue may be used to find out efficiently which face a has the lowest importance value $I(a)$ in step 2 of the procedure.

Figure 2.a shows a scene with a land-use map in the form of an area partitioning. In Figure 2.b the GAP-tree, as computed by the procedure described above, is displayed. Note that a few attributes are shown in Figure 2.b: polygon, area, and perimeter of the final feature in the GAP-tree. The polygon is a real self-contained polygon with coordinates, and it is not a list of references. It is important to realize that this data structure is not redundant with respect to storing the common boundaries between area features. The only exception to this is the situation where a child has a common edge with another child or its parent; see the thick edges in Figure 2.a. More statistical information has to be obtained on how frequent this situation occurs in real data sets. As can be seen in Figure 2.b, the GAP-tree is a multi-way tree and not a binary tree. Some visual results of the on-the-fly generalization techniques with real data are displayed in Section 4. The

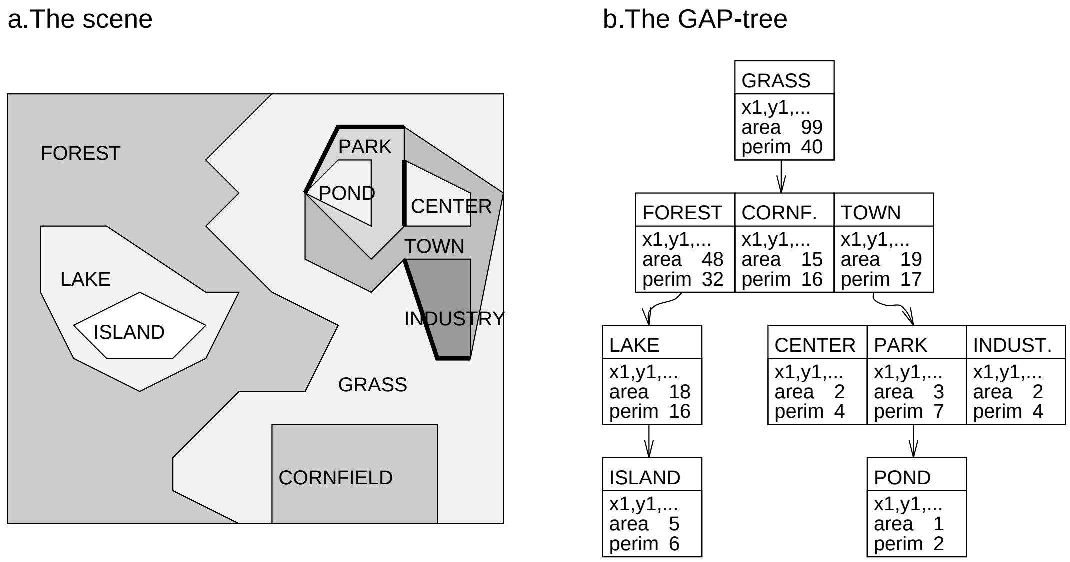

Figure 2: The scene and the associated GAP-tree

additional operations using the GAP-tree, not necessarily related to visualization, are described in the next subsection.

2.4 Operations on the GAP-tree

As a feature in the GAP-tree contains the total generalized area, the actual area A of the polygon has to be corrected for the area of its children with the following formula:

$$A(\text{actual}) = A(\text{parent}) - \sum_{\text{child} \in \text{childeren}} A(\text{child})$$

It is important to realize that only one level down the tree has to be visited for this operation and not the whole subtree below the parent node. In a similar way the perimeter P of a polygon can be computed, with the only difference that the perimeter of the children have to be added to the perimeter of the parent. This results in the formula:

$$P(\text{actual}) = P(\text{parent}) + \sum_{\text{child} \in \text{childeren}} P(\text{child})$$

This formula for perimeter only works if the children have no edges in common with each other or with the parent. Often the boundaries of the areas in the GAP-tree are indeed non-redundant. This also enables the use of the BLG-tree for simplification of important area features at small-scale maps without producing overlaps or gaps between features. The use of the BLG-tree has a very positive effect on the response times of small-scale maps; see Section 4.

A possible implementation difficulty is the fact that the GAP-tree is a multi-way tree. Therefore, a simple linear version has been derived from the GAP-tree by putting the features in a list based on their level in the tree. The top level feature in the tree will be the first element of this list, the second level features will follow, and so on. For example, the linear list for the scene in Figure 2 is: GRASS, FOREST, CORNFIELD, TOWN, LAKE, CENTER, PARK, INDUSTRY, ISLAND, POND. When the polygons are displayed in this order, a good map can be produced without the GAP-tree. However, it is very difficult to compute the actual area without the GAP-tree.

As mentioned before, the on-the-fly generalization has been developed within the Postgres DBMS environment. Postgres is an extensible relational system. A relation does not guarantee any order among its elements. A good display can be obtained by sorting on the sequence number in the list or on the area of the feature.

3 Reactive Data Structures

There are several techniques used during the map generalization process: simplification, selection, exaggeration, displacement, symbolization and aggregation [11, 16]. A reactive data structure [15, 19] is defined as a *geometric* data structure with *detail levels*, intended to support the sessions of a user working in an interactive mode. It enables the information system to *react* promptly to user actions. This section gives a short description of tools for on-the-fly simplification and selection: the *BLG-tree* and the *Reactive-tree* in the Subsections 3.1 and 3.2. More details can be found in [20, 21, 24]. Note that for building the Reactive-tree, the pre-classification of the importance of a feature by a specialist is required.

Implementation of the reactive data structures in the GIS called GEO++ has proven to be very effective [23], especially for point, line, and individual area features: map displays based on very large data sets ($>$ 100 Mb) can be generated within a few seconds. Note that exaggeration is used implicitly when a line (or point) feature in a small-scale map is still displayed on the screen, because the minimum linewidth of one pixel at this scale is already an exaggeration of the thin line features in reality. There is still no support for the other generalization techniques: combination, symbolization, and displacement [17]. In [23] it is indicated how the reactive data structures are implemented in Postgres DBMS environment. There is a big implementation difference between the BLG-tree and the Reactive-tree, because the BLG-tree is implemented as a part of an abstract data type, while the Reactive-tree is a complete new access method. The latter is far more difficult to add, because an access method interacts with many (undocumented) parts of the Postgres DBMS.

3.1 BLG-tree

The Binary Line Generalization tree (BLG-tree) [24] is a data structure used for line generalization. Only important objects (represented by polylines) need to be displayed on a small-scale map, but without a generalization structure, these polylines are drawn with too much detail. A few well known structures for supporting line generalization are the *Strip tree* [2], the *Arc tree* [8], and the *Multi-Scale Line Tree* [10]. In [24] another data structure is described, the BLG-tree, which is suited for polylines, is continuous in detail level, and can be implemented with a simple binary tree.

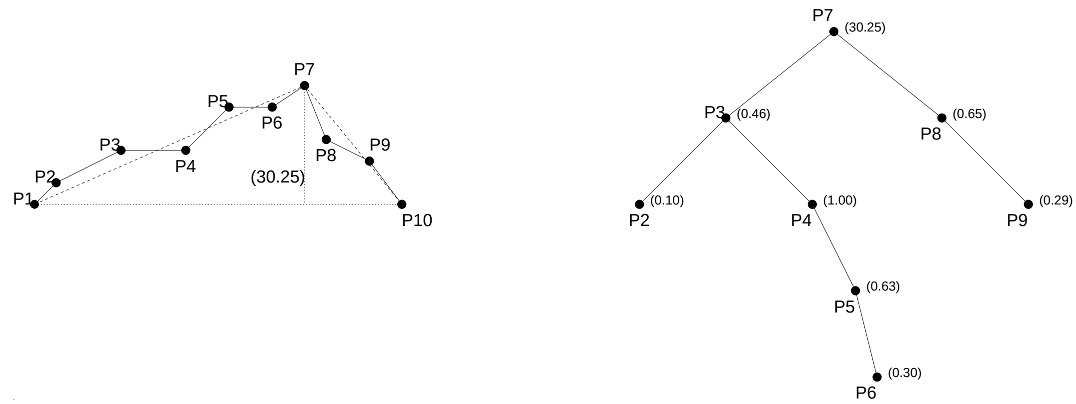

Figure 3: Binary Line Generalization tree

The algorithm to create a BLG-tree is based on the Douglas-Peucker algorithm [6]. Figure 3 illustrates the BLG-tree creation procedure and the resulting binary tree. This tree is stored for every polyline in order to avoid the expensive execution of the Douglas-Peucker algorithm, each time a line generalization is needed. The BLG-tree can also be used for polygons by applying the same techniques on the boundary of the polygon.

3.2 Reactive-tree

The Reactive-tree [20, 21] is based on the R-tree [9] spatial access method, and has similar properties. The main differences with the R-tree are that the internal nodes of a Reactive-tree can contain both *tree entries* and *object entries*, and the leaf nodes can occur at higher tree levels. The motivation behind this is that it must be possible to store the more important features in the higher levels of the tree; see Figure 4.

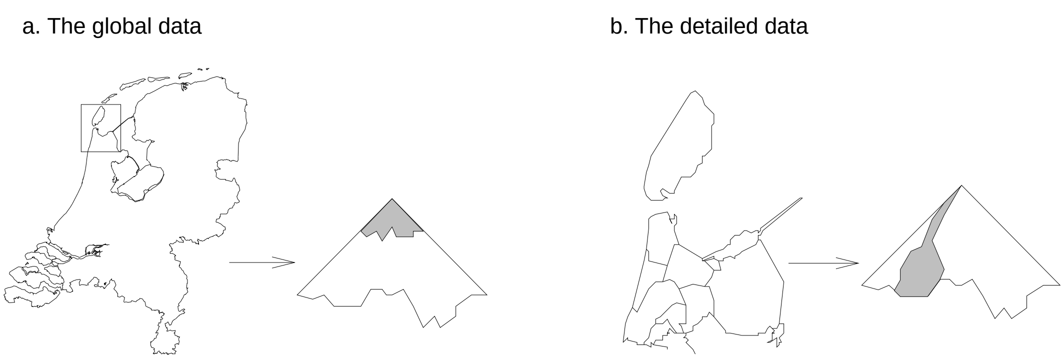

Figure 4: The Scene and the Rectangles of the Reactive-tree

The further one zooms in, the more tree levels must be addressed. Roughly stated, during map generation based on a selection from the Reactive-tree, one should try to choose the required importance value such that a constant number of objects will be selected. This means that if the required region is large only the more important objects should be selected (accessing the top levels in the tree) and if the required region is small, then the less important objects must also be selected (accessing the nodes in a certain region of the tree). If the Reactive-tree is built well, then the total number of accesses is more or less the same. Therefore, the interaction time is also constant.

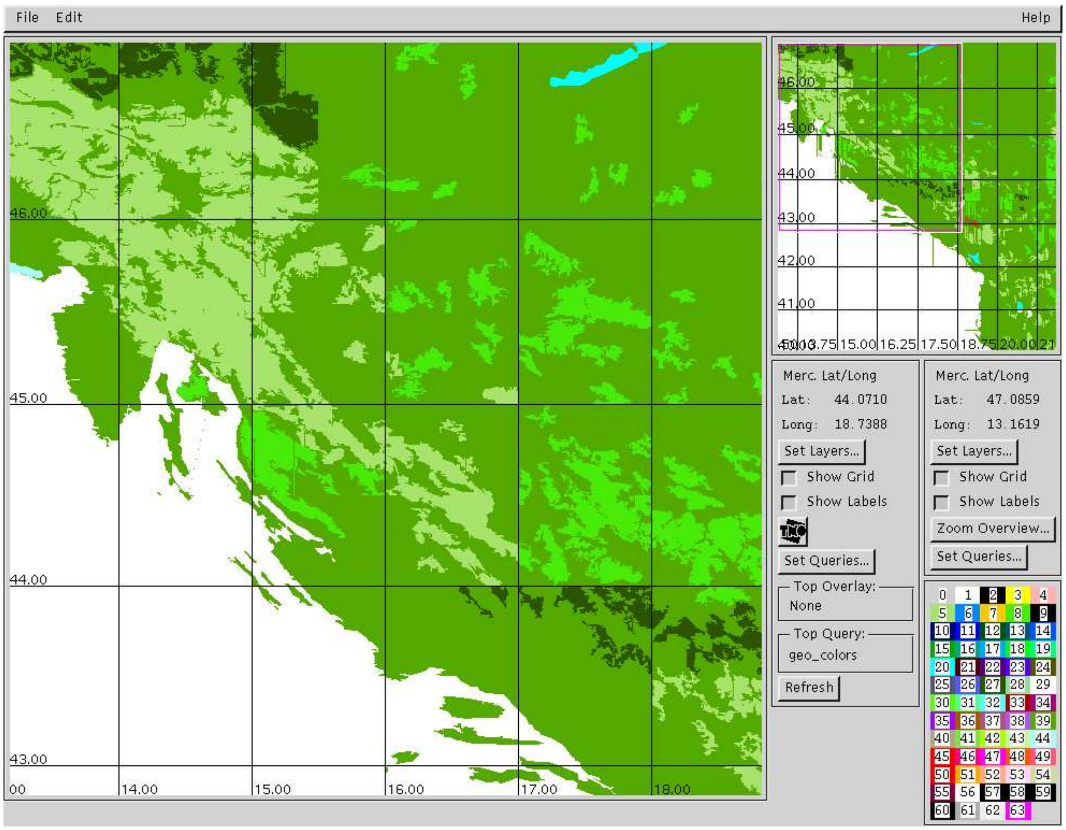

Figure 5: Using the GAP/Reactive-tree and BLG-tree (coarse map)

4 Test Results

In this section the results of the performance tests of the combined use of the GAP-tree, the Reactive-tree, and the BLG-tree are presented. The DLMS DFAD[2] data [5] of the former Republic of Yugoslavia is used. Only the area features are used in order to evaluate the effectiveness of the GAP-tree. Subsection 4.1 shows how the Reactive-tree, BLG-tree, and GAP-tree are used in practice. Subsection 4.2 presents the benchmarks when using these data structures. The tests are performed using the GIS environment which consists of the open DBMS Postgres [18] and the geographic frontend GEO++ [25]

[2]DLMS (Digital LandMass) DFAD (Digital Feature Analyses Data) Level 1 has a data density which can be compared to 1:200,000 scale map

on a Sun SPARCstation II (32Mb main memory) under SunOS 4.1.2. The test results of on-the-fly generalization of line features using the Reactive-tree and the BLG-tree can be found in [23]: querying World Databank II [7] (about 38,000 line features with total size 110 Mb) with response times below 5 seconds.

4.1 Example

In this subsection an example of the use of the Reactive-tree, BLG-tree, and GAP-tree will be given. The Reactive-tree access method can be defined before any tuples are added to the relation, or one can first insert all tuples and decide later that a Reactive-tree is needed. The following two Postquel queries show the definition of the user table `AreaFeature` and the definition of a Reactive-tree index on this table using the `Reactive2_ops` operator class.

```
create AreaFeature (Height=int2, Idcode=int2, Tree=int2,
   Roof=int2, shape=POLYGON2, reactive = REACTIVE2)\g

define index af_index on AreaFeature
   using ReactiveTree (reactive Reactive2_ops)\g
```

The GEO++ system automatically generates Postquel queries with the proper values for the BLG-tree and the Reactive-tree depending on the current scale. For the map in Figure 5 the following query is generated:

```
retrieve (blg_pgn2= Blg2Pgn(AreaFeature.shape,"0.01"::float4))
  where AreaFeature.reactive && "(13,40,23,47,2)"::REACTIVE2
  sort by AreaFeature.oid
```

The Postgres query optimizer automatically selects the Reactive-tree access method when evaluating this query. The BLG-tree is used by specifying the function `Blg2Pgn` in the target list of the query. The linearized GAP-tree is reflected by the *sort by* clause of the query.

4.2 Benchmark

The visual results of the on-the-fly generalization techniques can be seen in Figures 5, 6, and 7. All maps, including the overview in the upper right, are generated by the same query with only different sized retrieved regions. DLMS DFAD can be regarded as land-use data with over 100 different area classifications, such as: lake, water, trees, sand, swamp, tundra, snow/ice, industry, commercial, recreational, residential, etc. A few notes with respect to the visualization:

1. Many colors on the screen are lost in the gray scales of the printer.
2. As the emphasis is on the GAP-tree, the line and points features have been omitted, resulting in an incomplete map.

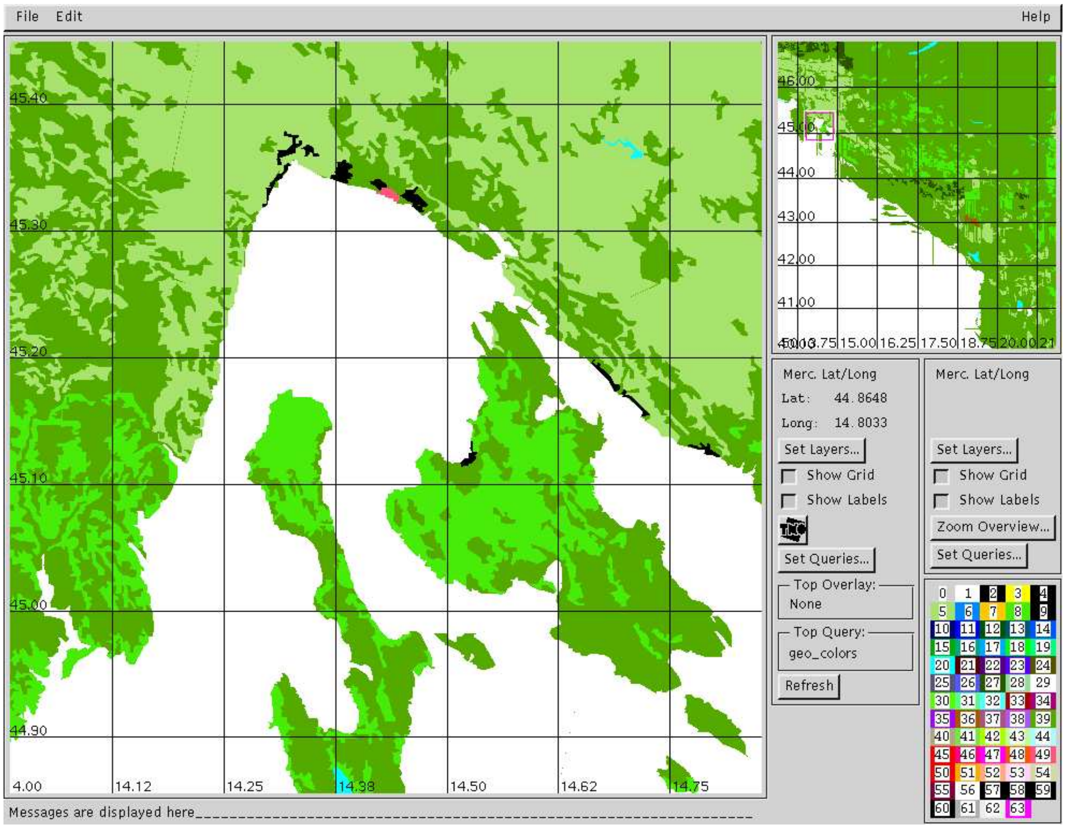

Figure 6: Using the GAP/Reactive-tree and BLG-tree (middle map)

3. In the upper right corner of each figure, an overview of the region is shown without the mainland of Italy.

4. Though the DLMS DFAD data is stored in a seamless database, it has been digitized on a map sheet base. During this process similar features have been classified differently; see Figure 5 near the 44 degrees meridian.

The DLMS DFAD test database contains 70.272 area features (requiring about 60Mb) which are given an importance value ranging from 1 to 5, at each level the more detailed level contains about one order of magnitude more features. Queries which selected tuples at five different area sizes were executed. The area sizes varied from 1.6×1.6 degrees to 0.1×0.1 degrees latitude/longitude. The Reactive-tree uses importance levels to reduce the number of selected objects at the more coarse level. Actually, the information density (i.e., the number of displayed features) should remain equal under the varying scales. The use of the GAP-tree will make sure that the map does not contain gaps when omitting the less important area features. The R-tree has no mechanism to select on importance level, that is why the R-tree retrieves much more objects in the larger areas. The following cases were tested: no index structure, an R-tree index, an R-tree index and a BLG-tree, and finally a GAP/Reactive-tree and a BLG-tree. In all cases, ten random area queries

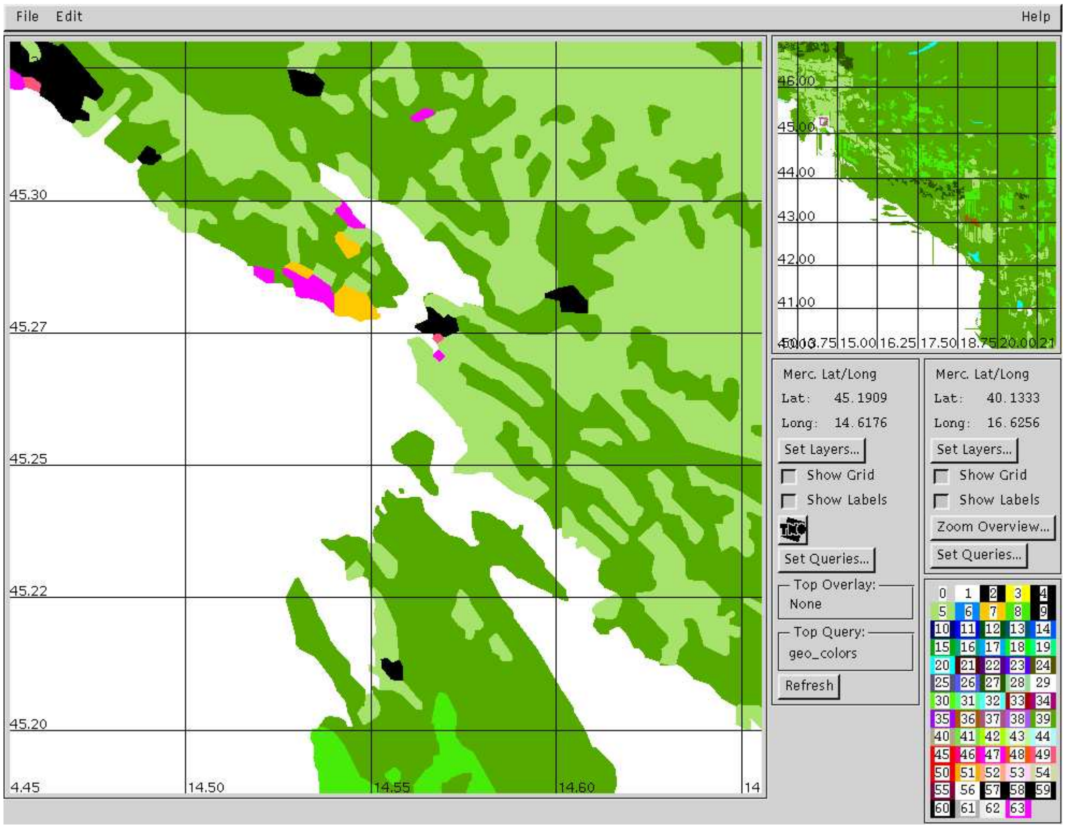

Figure 7: Using the GAP/Reactive-tree and BLG-tree (detail map)

are executed. The presented response time is in seconds and is the average time over the ten random generated queries.

The data was retrieved over a network file system by Postgres. The special test program is a simple Postgres frontend application. The response time was measured with the Unix `time` command. The test program needs a parameter which indicates the size of the search area. This size is used to calculate which importance values are to be retrieved.

The difference in response time between the GAP/Reactive-tree and the R-tree is decreasing when moving to smaller areas, since the difference in the number of returned objects is narrowing. Table 1 shows the average number of returned objects for each method. In a similar way, the BLG-tree is more effective when used in a large area search. In that case, a lot of polygon points can be omitted.

All in all, the GAP/Reactive-tree and BLG-tree combination gives very good improvements. The process of selection and generalization results in a better map without too much detail. The selection could be done with the R-tree, but in that case a user should change the queries on every scale, in order to select only the desired objects. Also, the important objects would then be located at the leaf level in the tree just as the other objects, which would slow down the small-scale queries. Figure 8 shows the response times

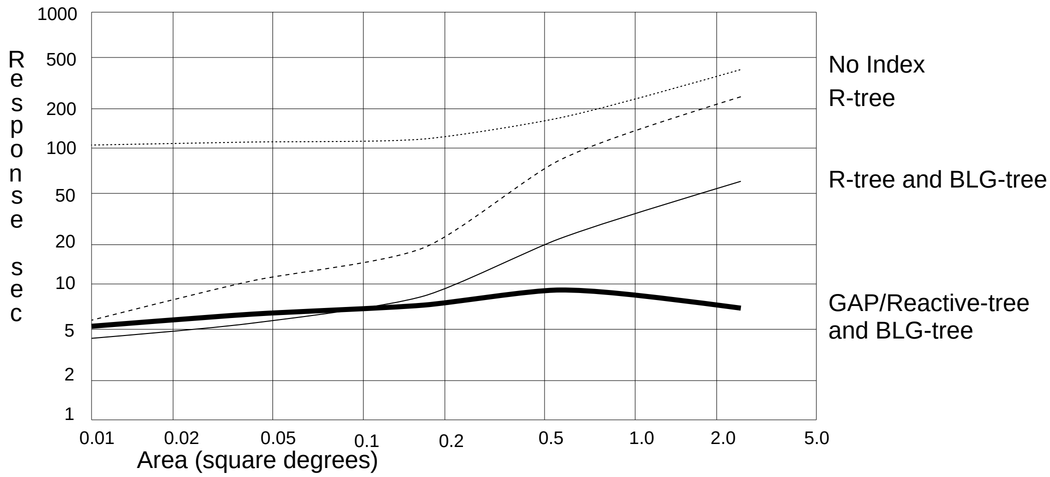

Figure 8: The average response time of the index structures

Table 1: Area sizes and the average number of returned objects

Area (in square degrees)	0.01	0.04	0.16	0.64	2.56
No index	25.9	64.7	198.0	723.6	2262.3
R-tree	25.9	64.7	198.0	723.6	2262.3
R-tree & BLG-tree	25.9	64.7	198.0	723.6	2262.3
GAP/Reactive-tree & BLG-tree	25.9	52.8	43.6	55.8	48.7

of the different methods, while Table 2 shows the same information in tabular form.

If geometrically close objects are guaranteed to be stored close together on disk, some extra speed may be gained. The Reactive-tree does not give such guarantee. It depends on the inserting order whether two geometrically close objects are also stored close on disk. *Clustering* can be used to improve this storage aspect. In Postgres, a simple kind of clustering can be very easily simulated by retrieving the objects using a rectangle that contains all the objects. In fact, an access method scan is performed. When the objects are stored in a new relation in the retrieved order, geometrically close objects are stored close together on disk. This reduces the number of disk pages to be fetched for spatial queries and therfore the results will be returned faster.

Table 2: Area sizes and the average response time (including sort)

Area (in square degrees)	0.01	0.04	0.16	0.64	2.56
No index	113.8	117.7	127.1	183.4	378.7
R-tree	6.6	10.2	19.0	80.5	285.8
R-tree & BLG-tree	4.3	5.7	8.1	21.2	61.2
GAP/Reactive-tree & BLG-tree	5.4	6.9	7.5	9.7	7.3

5 Conclusions

After the Reactive-tree and BLG-tree, the GAP-tree forms a new important step towards the realization of an interactive multi-scale GIS. It is now possible to interactively browse through "area partitioned" geographic data sets. In a database with more than 70.000 features it is now possible to get map displays at any required scale in about 6–9 seconds. In the near feature, more tests with datasets are planned; e.g., with the Topographic base map of The Netherlands. An interesting open question is how to assign importance values automatically to features when building the Reactive-tree for a new data set.

Further research is also required to determine how the GAP-tree can be maintained efficiently under edit operations. Additional further research topics are: the use of (dynamic) clustering techniques, and the design and implementation of other generalization techniques to support: combination, symbolization, and displacement.

Acknowledgments

I would like to thank the Postgres Research Group (University of California at Berkeley) for making their system available. Special thanks to Tom Vijlbrief for helping me with the GEO++ issues and to Vincent Schenkelaars for performing the DLMS benchmarks. Many valuable comments and suggestions on a preliminary version of this paper were made by Marcel van Hekken and Paul Strooper.

References

[1] Arbeitsgemeinschaft der Vermessungsverwaltungen der Länder der Bundesrepublik Deutschland (AdV). *Amtliches Topographic-Kartographisches Informationssystem (ATKIS)*, 1988. (in German).

[2] Dana H. Ballard. Strip trees: A hierarchical representation for curves. *Communications of the ACM*, 24(5):310–321, May 1981.

[3] Gerard Boudriault. Topology in the TIGER file. In *Auto-Carto 8*, pages 258–269, 1987.

[4] DGIWG. DIGEST – digital geographic information – exchange standards – edition 1.1. Technical report, Defence Mapping Agency, USA, Digital Geographic Information Working Group, October 1992.

[5] DMA. Product specifications for digital feature analysis data (DFAD): Level 1 and level 2. Technical report, Defense Mapping Agency, Aerospace Center, St Louis, Mo., April 1986.

[6] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of points required to represent a digitized line or its caricature. *Canadian Cartographer*, 10:112–122, 1973.

[7] Alexander J. Gorny and Russ Carter. World Data Bank II: General users guide. Technical report, US Central Intelligence Agency, January 1987.

[8] Oliver Günther. *Efficient Structures for Geometric Data Management*. Number 337 in Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1988.

[9] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. *ACM SIGMOD*, 13:47–57, 1984.

[10] Christopher B. Jones and Ian M. Abraham. Line generalization in a global cartographic database. *Cartographica*, 24(3):32–45, 1987.

[11] J. P. Lagrange, A. Ruas, and L. Bender. Survey on generalization. Technical report, IGN, April 1993.

[12] Martien Molenaar. Single valued vector maps: A concept in Geographic Information Systems. *Geo-Informationssysteme*, 2(1):18–26, 1989.

[13] J. C. Muller, R. Weibel, J. P. Lagrange, and F. Salge. Generalization: state of the art and issues. Technical report, European Science Foundation, GISDATA Task Force on Generalization, 1993.

[14] Thomas K. Peucker and Nicholas Chrisman. Cartographic data structures. *American Cartographer*, 2(1):55–69, 1975.

[15] Projectgroep. IDECAP interactief pictorieel informatiesysteem voor demografische en planologische toepasingen: Een verkennend en vergelijkend onderzoek. Technical Report Publicatiereeks 1982/2, Stichting Studiecentrum voor Vastgoedinformatie te Delft, 1982.

[16] A. H. Robinson, R. D. Sale, J. L. Morrison, and P. C. Muehrcke. *Elements of Cartography*. John Wiley, New York, 5th edition, 1984.

[17] K. Stuart Shea and Robert B. McMaster. Cartographic generalization in a digital environment: When and how to generalize. In *Auto-Carto 9*, pages 56–67, April 1989.

[18] Michael Stonebraker, Lawrence A. Rowe, and Michael Hirohama. The implementation of Postgres. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):125–142, March 1990.

[19] J. van den Bos, M. van Naelten, and W. Teunissen. IDECAP interactive pictorial information system for demographic and environmental planning applications. *Computer Graphics Forum*, 3:91–102, 1984.

[20] Peter van Oosterom. A reactive data structure for Geographic Information Systems. In *Auto-Carto 9*, pages 665–674, April 1989.

[21] Peter van Oosterom. The Reactive-Tree: A storage structure for a seamless, scaleless geographic database. In *Auto-Carto 10*, pages 393–407, March 1991.

[22] Peter van Oosterom. *Reactive Data Structures for Geographic Information Systems.* Oxford University Press, Oxford, 1993.

[23] Peter van Oosterom and Vincent Schenkelaars. Design and implementation of a multi-scale GIS. In *Proceedings EGIS'93: Fourth European Conference on Geographical Information Systems*, pages 712–722. EGIS Foundation, March 1993.

[24] Peter van Oosterom and Jan van den Bos. An object-oriented approach to the design of Geographic Information Systems. *Computers & Graphics*, 13(4):409–418, 1989.

[25] Tom Vijlbrief and Peter van Oosterom. The GEO system: An extensible GIS. In *Proceedings of the 5th International Symposium on Spatial Data Handling, Charleston, South Carolina*, pages 40–50, Columbus, OH, August 1992. International Geographical Union IGU.