The Reactive-tree: A Storage Structure for a Seamless, Scaleless Geographic Database

Peter van Oosterom*

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

Abstract

This paper presents the first fully dynamic and reactive data structure. *Reactive data structures are vector based structures tailored to the efficient storage and retrieval of geometric objects at different levels of detail. Geometric selections can be interleaved by insertions of new objects and by deletions of existing objects. Detail levels are closely related to cartographic map generalization techniques. The proposed data structure supports the following generalization techniques: simplification, aggregation, symbolization, and selection. The core of the reactive data structure is the* Reactive-tree, *a geometric index structure, that also takes care of the selection-part of the generalization. Other aspects of the generalization process are supported by introducing associated structures, e.g. the* Binary Line Generalization-tree *for simplification. The proposed structure forms an important step in the direction of the development of a seamless, scaleless geographic database.*

1 Introduction

The deficiencies of using map sheets in Geographic Information Systems are well-known and have been described by several authors [5, 10]. The obvious answer to these deficiencies is a *seamless* or sheetless database. A seamless database is made possible in an interactive environment by using some form of multi-dimensional indexing, e.g. the R-tree [15] or the KD2B-tree [35]. It turns out that the *integrated* storage of multi-scale (*scaleless*) data in a spatial indexing structure forms the bottleneck in the design of a seamless, scaleless database [14]. A first approach might be to define a discrete number of levels of detail and store them separately each with its own spatial indexing structure. Though fast enough for interactive applications, this solution is not particularly elegant. It introduces redundancy because some

*A part of this work was done while the author was at the Department of Computer Science, University of Leiden, P.O. Box 9512, 2300 RA Leiden, The Netherlands.

a. Scale 1:25,000 b. Scale 1:50,000

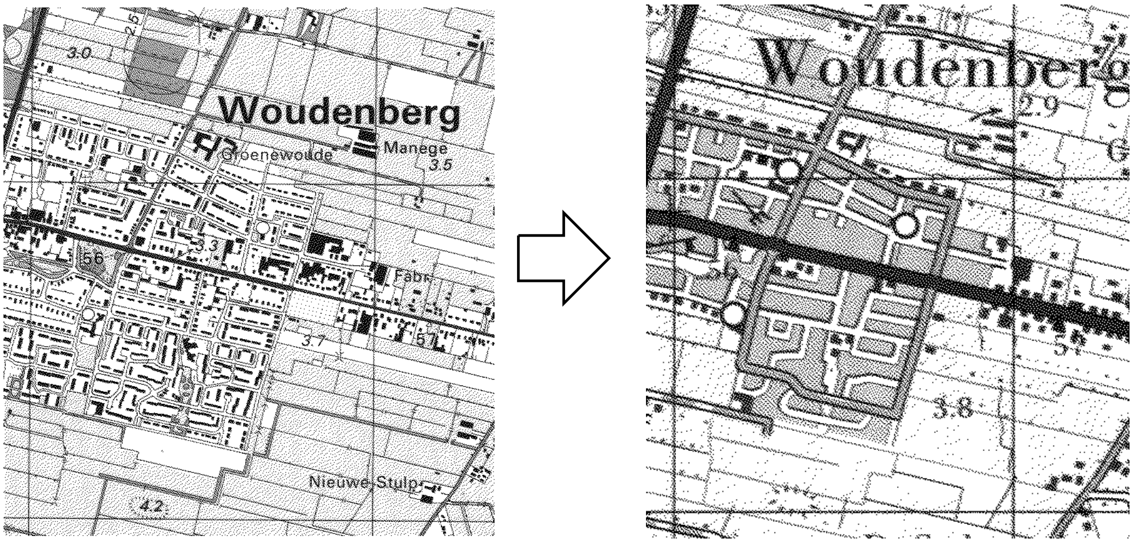

Figure 1: The Map Generalization Process

objects have to be stored at several levels. Apart from the increased memory usage, another drawback is that the data must be kept explicitly consistent. If an object is edited at one level, its "counter part" at the other levels must be updated as well. In order to avoid these problems we should try to design a storage structure that offers both spatial capabilities and multiple detail levels in an integrated manner: a *reactive data structure*. Two spatial data structures, that provide some limited facilities for multiple detail levels, are known: the Field-tree [9, 11] and the reactive BSP-tree [33, 34]. However, these are not fully dynamic.

First, we will discuss some of the fundamental problems associated with detail levels in a multi-scale database. The concept of multiple detail levels can not be defined as sharply as that of spatial searching. It is related to one of the main topics in cartographic research: *map generalization*; that is, to derive small scale maps (large regions) from large scale maps (small regions). Figure 1 illustrates the generalization process by showing the same part of a 1:25,000 map and of an enlarged 1:50,000 map. A number of generalization techniques for geographic entities have been developed and described in the literature [26, 30, 31]:

- simplification (e.g. line generalization);
- combination (aggregate geometrically or thematically);
- symbolization (e.g. from polygon to polyline or point);
- selection (eliminate, delete);
- exaggeration (enlarge); and
- displacement (move).

Unlike spatial searching, which is a pure geometric/topologic problem, map generalization is application dependent. The generalization techniques are categorized into two groups [23, 26]: geometric and conceptual generalization. In *geometric generalization* the basic graphic representation type remains the same, but is, for example, enlarged. This is not the case in *conceptual generalization* in which the

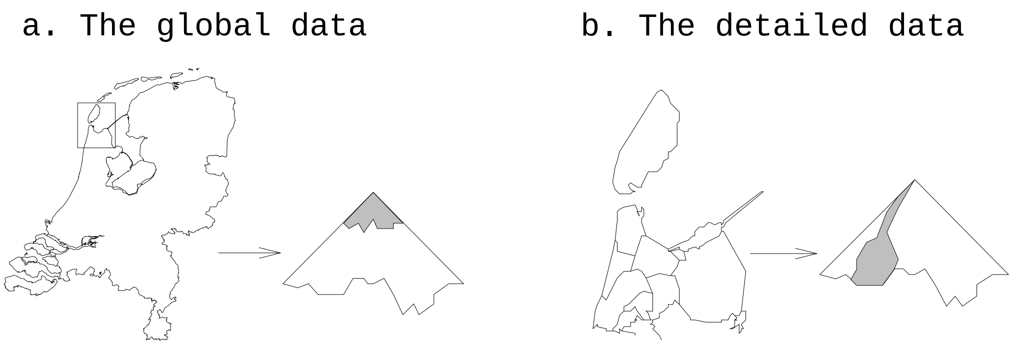

Figure 2: The Place of Global and Detailed Data

representation changes, e.g. change a river from a polygon into a polyline type of representation.

Generalization is a complex process of which some parts, e.g. line generalization [21, 22], are well suited to be performed by a computer and others are more difficult. Nickerson [25] shows that very good results can be achieved with a rule based expert system for generalization of maps that consist of linear features. Shea and McMaster [31] give guidelines for when and how to generalize. Müller [24] also applies a rule based system for selection (or its counterpart: elimination) of geographic entities. Brassel and Weibel [4] present a framework for automated map generalization. Mark [20], Müller [23], and Richardson [29] all state that the nature of the phenomenon must be taken into account during the generalization in addition to the more traditional guidelines, such as: the graphic representation (e.g. number of points used to draw a line) and the map density. This means that it is possible that a different generalization technique is required for a line representing a road than for a line representing a river. It is important to note that the spatial data structure with detail levels, presented in this paper, is only used to store the results of the generalization process.

The *guideline* that important objects must be stored in the higher levels of the tree, is the starting point for the design of the Reactive-tree. This *guideline* was derived during the development of the reactive BSP-tree [33, 34] and is illustrated in Figure 2: the global data are stored in the top levels of the tree (gray area in Figure 2a) and the detailed data of the selected region are stored in the lower levels of the tree (Figure 2b) in nodes which are "quite close" to each other. The Reactive-tree is an index structure, which supports geometric searching at different levels of importance. The properties of the Reactive-tree are described in Section 2, together with a straightforward Search algorithm. Insert and Delete algorithms are given in the subsequent section. Support for the generalization technique *simplification* is provided by representing polygonal or polyline objects by a Binary Line Generalization-tree, see Section 4. Support for the generalization techniques *aggregation* and *symbolization* is discussed in Section 5. In Section 6 the Alternative Reactive-tree is presented, not based on the *guideline* stated above. This paper is concluded with an evaluation of the presented structures.

2 The Properties of the Reactive-tree

In the following subsection, it is argued that *importance values* associated with objects, are required. The two subsequent subsections give an introduction to the Reactive-tree and a formal description of its properties, respectively. The last subsection describes a geometric Search algorithm, which takes the required importance level into account.

2.1 Importance Values

Generalization is, stated simply, the process of creating small scale (coarse) maps out of detailed large scale maps. One aspect of this process is the removal of unimportant and often, but not necessarily, small objects. This can be repeated a number of times, each time resulting in a smaller scale map with fewer objects in a fixed region. Each object is assigned a logical *importance value*, a natural number, in agreement with the smallest scale on which it is still present. Less important objects get low values, more important objects get high values. The use of importance values for the selection of objects was first published by Frank [8].

Which objects are important is depends on the application. In many applications a natural hierarchy is already present. In the case of, for example, a road map these are: highways, major four-lane roads, two-lane roads, undivided roads, and dirt roads. Another example can be found in WDB II [12] where lakes, rivers, and canals are classified into several groups of importance. Typically, the number of levels is between five and ten, depending on the size and type of the geographic data set. In a reasonable distribution the number of objects having a certain importance is one or two orders of magnitude larger than the number of objects at the next higher importance level; a so called, *hierarchical distribution*.

2.2 Introduction to the Reactive-tree

Several existing geometric data structures are suited to be adapted for the inclusion of objects with different importance values, for example the R-tree [15], the Sphere-tree, and the dynamic KD2B-tree [35]. In this paper, the Reactive-tree is based on the R-tree, because the R-tree is the best known structure. However, if orientation insensitivity is important, then one of the other structures mentioned must be used. The Reactive-tree is a multi-way tree in which, normally, each node contains a number of entries. There are two types of entries: *object-entries* and *tree-entries*. The internal nodes may contain both, in contrast to the R-tree. The leaf nodes of the Reactive-tree contain only object-entries. An object-entry has the form

(MBR, imp-value, object-id)

where *MBR* is the minimal bounding rectangle, *imp-value* is a natural number that indicates the importance, and *object-id* contains a reference to the object. A tree-entry has the form

(MBR, imp-value, child-pointer)

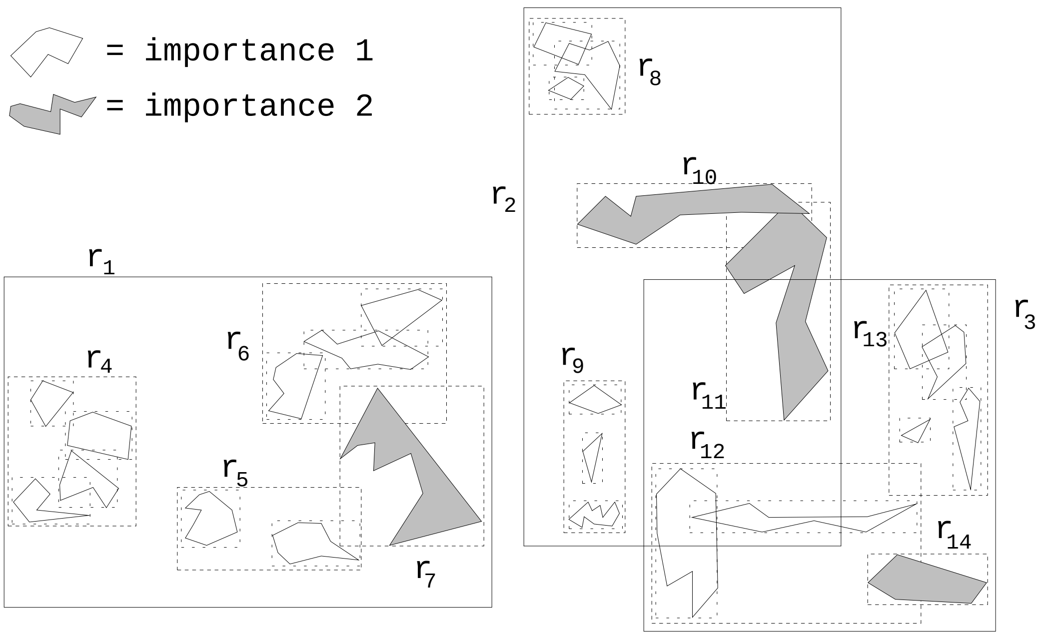

Figure 3: The Scene and the Rectangles of the Reactive-tree

where *child-pointer* contains a reference to a subtree. In this case *MBR* is the minimal bounding rectangle of the whole subtree and *imp-value* is the importance of the child-node incremented by 1. The importance of a node is defined as the importance of its entries. Note that the size of a tree-entry is the same as that of an object-entry. When one bit in the *object-id/child-pointer* is used to discriminate between the two entry types, then there is no physical difference between them in the implementation. Each node of the Reactive-tree corresponds to one disk page. Just as in the R-tree, M indicates the maximum number of entries that will fit in one node, and $m \leq \lceil M/2 \rceil$ is the minimum number of entries. Assume that the page size is 1024, then M is 48 in a realistic implementation.

2.3 Defining Properties

In this subsection the defining properties of the Reactive-tree are presented. The fact that the empty tree satisfies these properties and that the Insert and Delete algorithms given in Section 3 do not destroy them, guarantees that a Reactive-tree always exists. The Reactive-tree satisfies the following properties:

1. For each object-entry *(MBR, imp-value, object-id)*, *MBR* is the smallest axes-parallel rectangle that geometrically contains the represented object of importance *imp-value*.
2. For each tree-entry *(MBR, imp-value, child-pointer)*, *MBR* is the smallest axes-parallel rectangle that geometrically contains all rectangles in the child node and *imp-value* is the importance of the child-node incremented by 1.
3. All the entries contained in nodes on the same level are of equal importance, and more important entries are stored at higher levels.
4. Every node contains between m and M object-entries and/or tree-entries, unless it has no brothers (a *pseudo-root*).

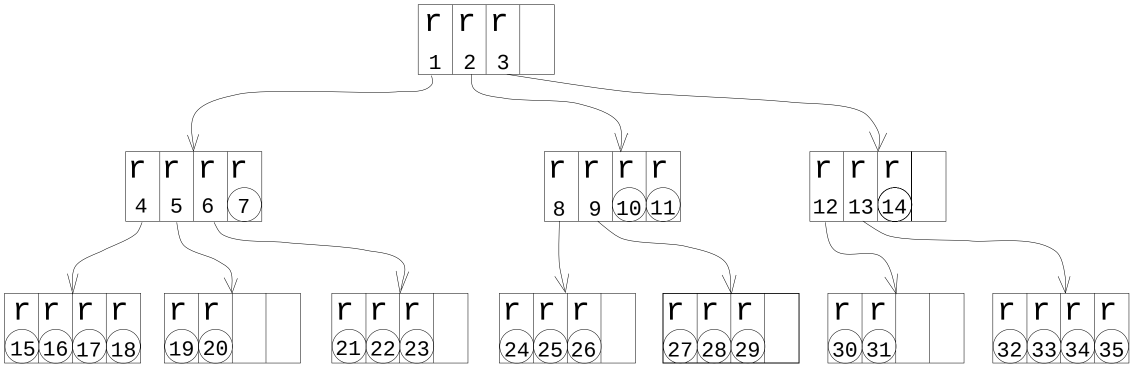

Figure 4: The Reactive-tree

5. The root contains at least 2 entries unless it is a leaf.

It is not difficult to see that the least important object-entries of the whole data set are always contained in leaf nodes on the same level. In contrast to the R-tree, leaf nodes may also occur at higher levels, due to the more complicated balancing criteria which are required by the multiple importance levels; see properties 3, 4, and 5. Further, these properties imply that in an internal node containing both object-entries and tree-entries, the importance of the tree-entries is the same as the importance of the object-entries. Figure 3 shows a scene with objects of two importance levels: objects of importance 1 are drawn in white and the objects of importance 2 are drawn in grey. This figure also shows the corresponding rectangles as used in the Reactive-tree. The object-entries in the Reactive-tree are marked with a circle in Figure 4. The importance of the root node is 3, and the importance of the leaf nodes is 1.

2.4 Geometric Searching with Detail Levels

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 and if the required region is small, then the less important objects must be selected also. The recursive *Search* algorithm to report all object-entries that have at least importance *imp* and whose *MBRs* overlap search region S, is invoked with the root of the Reactive-tree as current node:

1. If the importance of the current node N is less than *imp*, then there are no qualifying records in this node or in one of its subtrees.
2. If the importance of the current node N is greater or equal to *imp*, then report all object-entries in this node that overlap S.
3. If the importance of the current node N is greater than *imp*, then also invoke the Search algorithm for the subtrees that correspond to tree-entries that overlap S.

3 Insert and Delete Entry Algorithms

The Search algorithm is the easy part of the implementation of the Reactive-tree. The hard part is presented by Insert and Delete algorithms that do not destroy the properties of the Reactive-tree. In the implementation presented here, there is exactly one level in the Reactive-tree for each importance value, in the range from min_imp to max_imp, where min_imp and max_imp correspond to the least and to the most important object, respectively. If necessary, there may be one or more tree levels on top of this, which correspond to importance levels max_imp + 1 and higher. Then the top level nodes contain tree-entries only. Assume that $tree\_imp \geq max\_imp$ is the importance of the root of the Reactive-tree, then the height of the tree is $tree\_imp$ +1 - min_imp. The values of min_imp and $tree\_imp$ are stored in global variables. In the algorithms described below, the trivial aspects of maintaining the proper values of these variables are often ignored. Because of the direct relationship between the importance and the level of a node in the Reactive-tree of this implementation, the imp_value may be omitted in both the object-entry and the tree-entry.

3.1 Insert Entry

The Insert algorithm described below does not deal with the special cases: empty tree and the insertion of an entry with importance greater than $tree\_imp$. Solutions for both are easy to implement and set the global variable $tree\_imp$ to the proper value. The *Insert* algorithm to insert a new entry E of importance E_imp in the Reactive-tree:

1. Descend the tree to find the node, that will be called N, by recursively choosing the best tree-entry until a node of importance E_imp or a leaf is reached. The best tree-entry is defined as the entry that requires the smallest enlargement of its MBR to cover E. While moving down the tree, adjust the MBRs of the chosen tree-entries on the path from the root to node N.
2. In the special case that node N is a leaf and the importance N_imp is greater than E_imp, a linear path (with length $N\_imp - E\_imp$) of nodes is created from node N to the new entry. Each node in this path contains only one entry. This is allowed, because these are all pseudo-roots.
3. Insert the (path to) new entry E in node N. If overflow occurs split the node into nodes N and N' and update the parent. In case the parent overflows as well, propagate the node-split upward.
4. If the node-split propagation causes the root to split, increment $tree\_imp$ by 1 and create a new root whose children are the two resulting nodes.

The node splitting in step 3 is analogous to the node splitting in the R-tree. A disadvantage of the Reactive-tree is the possible occurrence of pseudo-roots. These may cause excessive memory usage in case of a "weird" distribution of the number of objects per importance level; e.g. there are more important objects than unimportant objects.

3.2 Delete Entry

An existing object is deleted by the *Delete* algorithm:

1. Find the node N containing the object-entry, using its MBR.
2. Remove the object-entry from node N. If underflow occurs, then the entries of the under-full node have to be saved in a temporary structure and the node N is removed. In case the parent also becomes under-full, repeat this process. It is possible that the node-underflow continues until the root is reached and in that case *tree_imp* is decremented.
3. Adjust the MBRs of all tree-entries on the path from the removed object-entry back to the root.
4. If underflow has occurred, re-insert all saved entries on the proper level in the Reactive-tree by using the Insert algorithm.

There are three types of underflow in the Reactive-tree: the root contains 1 tree-entry only, a pseudo-root contains 0 entries, or one of the other nodes contains $m-1$ entries. The temporary structure may contain object-entries and tree-entries of different importance levels.

4 The Binary Line Generalization-tree

Selection, as supported by the Reactive-tree, can assure that only global and important polylines (or polygons) are selected out of a large-scale geographic data set, when a small-scale map (large regions) has to be displayed. However, without specific measures, these polylines are drawn with too much detail, because all points that define the polyline are used. This detail will be lost on this small-scale due to the limited resolution of the display. Also the drawing will take an unnecessary long period of time. It is better to use fewer points. This can be achieved by the *k-th point algorithm*, which only uses every k-th point of the original polyline for drawing. The first and the last points of a polyline are always used. This is to ensure that the polylines remain connected to each other in the nodes of a topologic data structure [3, 27]. This algorithm can be performed "on the fly" because it is very simple. The k can be adjusted to suit the specified scale. However, this method has some disadvantages:

- The shape of the polyline is not optimally represented. Some of the line characteristics may be lost if the original polylines contain very sharp bends or long straight line segments.
- If two neighboring administrative units are filled, for example, in case of a choropleth, and the k-th point algorithm is applied on the contour, then these polygons may not fit. The contour contains the re-numbered points of several polylines.

Therefore, a better line generalization algorithm has to be used, for instance the *Douglas-Peucker algorithm* [6]. Duda and Hart [7] describe an algorithm similar to the Douglas-Peucker algorithm and call it the "iterative end-point fit" method. Both references date back to 1973. A slightly earlier publication is given by Ramer [28] in 1972. These types of algorithms are time consuming, so it is wise to compute

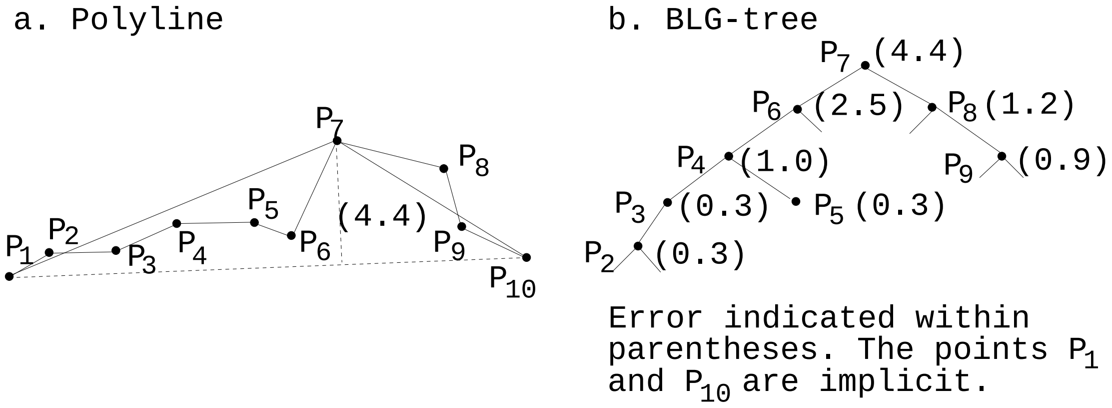

Figure 5: A Polyline and its BLG-tree

the generalization information for each polyline in a pre-processing step. The result is stored in, for instance, a Multi-scale Line Tree [17, 18]. The disadvantages of the Multi-scale Line Tree have already been discussed in [37]: it introduces a discrete number of detail levels and the number of children per node is not fixed. Strip trees [1] and Arc trees [13] are binary trees that represent curves (in a 2D-plane) in a hierarchical manner with increasing accuracy in the lower levels of the tree. These data structures are designed for arbitrary curves and not for simple polylines. Therefore, we introduce a new data structure that combines the good properties of the structures mentioned. We call this the *Binary Line Generalization-tree* (BLG-tree).

The BLG-tree stores the result of the Douglas-Peucker algorithm in a binary tree. The original polyline consists of the points p_1 through p_n The most coarse approximation of this polyline is the line segment $[p_1, p_n]$. The point of the original polyline, that has the largest distance to this line segment, determines the error for this approximation. Assume that this is point p_k with distance d, see Figure 5a. p_k and d are stored in the root of the BLG-tree, which represents the line segment $[p_1, p_n]$. The next approximation is formed by the two line segments $[p_1, p_k]$ and $[p_k, p_n]$. The root of the BLG-tree contains two pointers to the nodes that correspond with these line segments. In the "normal" situation this is a more accurate representation.

The line segments $[p_1, p_k]$ and $[p_k, p_n]$ can be treated in the same manner with respect to their part of the original polyline as the line segment $[p_1, p_n]$ to the whole polyline. Again, the error of the approximation by a line segment can be determined by the point with the largest distance. And again, this point and distance are stored in a node of the tree which represents a line segment. This process is repeated until the error (distance) is 0. If the original polyline does not contain three or more collinear points, the BLG-tree will contain all points of that polyline. It incorporates an exact representation of the original polyline. The BLG-tree is a static structure with respect to inserting, deleting and changing points that define the original polyline. The BLG-tree of the polyline of Figure 5a is shown in Figure 5b. In most cases, the distance values stored in the nodes will become smaller when descending the tree. Unfortunately, this is not always the case, as shown in Figure 6. It is not a monotonically decreasing series of values.

The BLG-tree is used during the display of a polyline or polygon at a certain scale. One can determine the maximum error that is allowed at this scale and the primitive

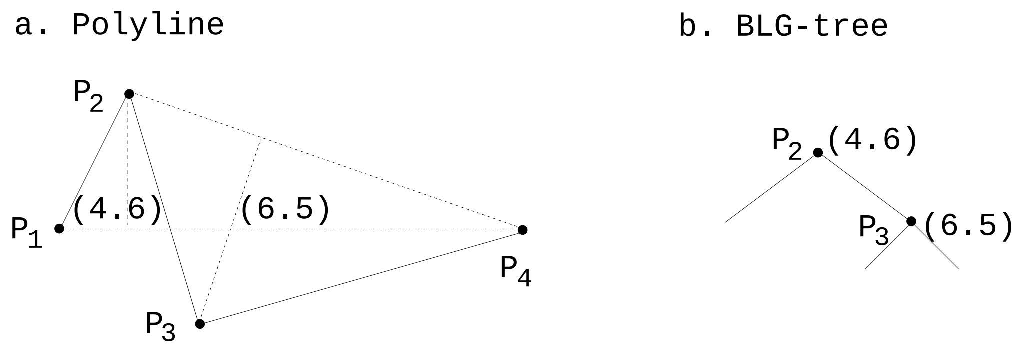

Figure 6: Increasing Error in BLG-tree

is *simplified* and a good graphic representation is obtained. During traversal of the tree, one does not have to go any deeper in the tree once the required accuracy is met. The BLG-tree can also be used for other purposes, for example (further details can be found in [37]):

- Estimating the area of a region enclosed by a number of polylines.
- Estimating the intersection(s) of two polylines. This is a useful operation during the calculation of a map overlay (polygon overlay).

Note that the BLG-tree is most useful for polylines and polygons defined by a large number of points. For a small number of points, "on the fly" execution of the Douglas-Peucker [6] algorithm may be more efficient. For polylines that are somewhere in between, another alternative might be interesting. Assign a value to each point to decide whether the point is used when displaying the polyline at a certain scale. This simple linear structure is probably fast enough for the medium sized polyline.

5 Support for Other Generalization Techniques

The Reactive-tree and the BLG-tree reflect only a part of the map generalization process: selection and simplification. A truly reactive data structure also deals with other aspects of the generalization process. In this section two more aspects are discussed: symbolization, and aggregation. These terms may be confusing in the context of the Reactive-tree, because the tree is usually described "top-down" (starting with the most important objects) and map generalization is usually described "bottom-up" (starting at the most detailed level). The two generalization techniques are incorporated in the reactive data structure by considering objects not as a simple list of coordinates, but as more complex structures. In practice, this can be implemented very well by using an object-oriented programming language, such as Procol [19, 32, 36, 37].

Symbolization changes the basic representation of a geographic entity, for example, a polygon is replaced by a polyline or point on a smaller scale map. Besides the coordinates of the polygon, the object structure contains a second representation in the form of a polyline or point. Associated with each representation is a resolution range which indicates where it is valid. An example of the application of the symbolization technique is a city which is depicted on a small scale map as a dot and

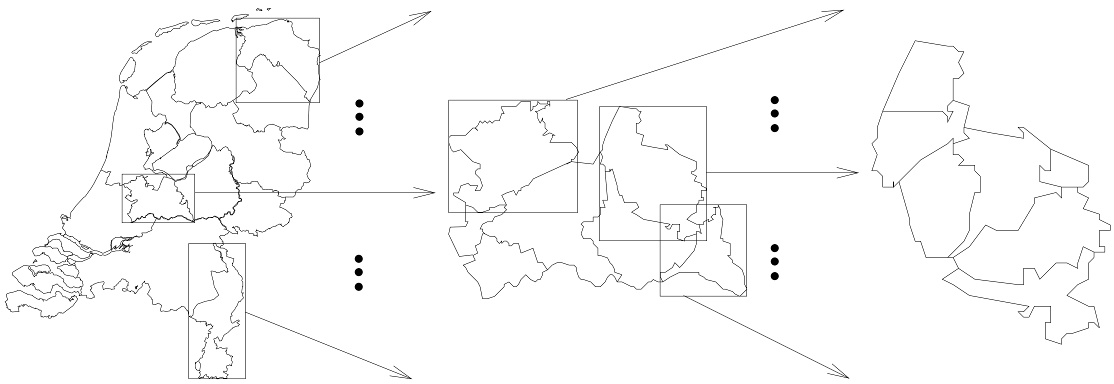

Figure 7: A Large Object is Composed of Several Small Objects

on a large scale map as a polygon.

The last generalization technique included in the reactive data structure is *aggregation*, that is the combination of several small objects into one large object. From the "top-down hierarchical tree" point of view, a large object is composed of several small objects; see Figure 7. The geometric description of the large object and the geometric descriptions of the small objects are all stored, because there is no simple relationship between them. The large object is some kind of "hull" around the small objects, see Figure 7. Usually, a bounding box around the small objects is a sufficient "geometric search structure", because the number of small objects is limited. However, if the number of small objects combined in one large object is quite large, then a R-subtree may be used.

Aggregation is used, for example, in the map of administrative units in The Netherlands [37]. Several municipalities are grouped into one larger economic geographic region (EGR), EGRs are grouped into a nodal region, nodal regions are grouped in a province, and so on. Another approach to this case is to consider the boundaries as starting point of the design, instead of the regions. In that case selection is the appropriate generalization technique and the Reactive-tree can be used without additional structures.

6 An Alternative Reactive-tree

In this section a reactive data structure is presented, which is not based on the guideline that important objects must be stored in the higher levels of the tree. The advantage of the *Alternative Reactive-tree* over the Reactive-tree is that it does not assume a hierarchical distribution of the number of objects over the importance levels.

The 2D Alternative Reactive-tree is based on a 3D R-tree. The 3D MBR of a 2D object with importance *imp* is defined by its 2D MBR and its extents in the third dimension are from *imp* and to *imp*+δ, where δ is a positive real number, so an object corresponds to a block with non-zero contents (except for point objects). Figure 8 depicts the 3D MBRs of a number of 2D objects at two different importance levels. When the parameter δ is chosen very small, e.g. 0.01, the Alternative Reactive-tree tries to group the objects that belong to the same importance level. This can be explained by the fact that there is a heavy penalty on the inclusion of an object with

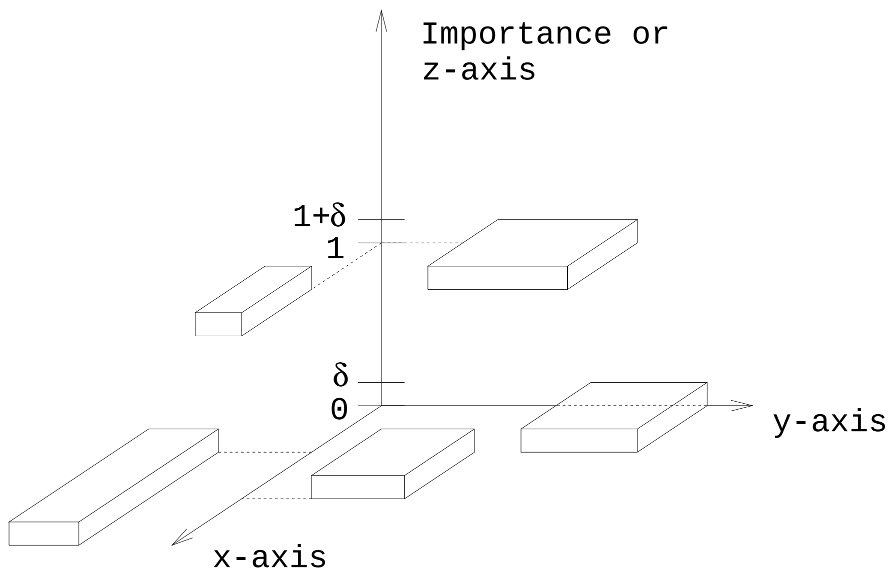

Figure 8: The 3D MBRs of the Alternative Reactive-tree

another importance value, as the volume of the 3D MBR will increase by at least a factor $(1+\delta)/\delta$. The larger δ becomes, the less the penalty, and the more likely it is that objects of different importance are grouped, and the Alternative Reactive-tree behaves more like a normal 2D R-tree. In any case, all objects, important and unimportant, are stored in leaf nodes on the same level.

The Alternative Reactive-tree can be generalized to support objects with general *labels* instead of the hierarchical importance values. This enables queries such as "Select all *capital* cities in region R." The label *capital* is associated with some of the geographic objects, by inserting these entries into the tree. A Geographic object may be associated with more labels by inserting more entries for the same object. In the implementation, label corresponds to a numeric value. By choosing certain values for these labels and for δ, possible coherence between labels may be exploited. This is what is actually done in the 2D Alternative Reactive-tree for hierarchically distributed data.

7 Discussion

This paper described the first fully dynamic and reactive data structure. It was presented as a 2D structure, but 3D and higher dimensional variants are possible. Note that this has nothing to do with the use of a 3D R-tree for the 2D Alternative Reactive-tree. The Reactive-tree and the Alternative Reactive-tree have been implemented in C++ on a Sun 3/60. Two large data sets have been used to test the reactive structures: WDB II [12] and the map of administrative units in The Netherlands. Both performance tests showed the advantage of the *selection* based on importance level and geometric position. Displaying the whole map area at interactive speed was possible, in contrast to the situation where the normal R-tree was used, which also showed a lot of annoying details. The additional structures for the support of simplification, symbolization, and aggregation are currently being implemented. Future performance tests depend on the availability of digital maps with generalization information.

Two other generalization techniques where not discussed: exaggeration and displacement. *Exaggeration* seems easy to include, because it is a simple enlargement of an aspect of the graphic representation of one object, e.g. the line width. However, the enlargement of linear features may cause other features to be covered and they must therefore be *displaced*. Exaggeration and displacement are difficult to handle, because multiple objects have to be considered. An ad hoc solution is to associate an explicit set of tuples *(displacement, map-scale-range)* with each object that has to be displaced and a set of tuples *(enlargement, map-scale-range)* with each object that has to be enlarged. Further research is required in order to develop more elegant solutions.

Very recently, another reactive data structure has been proposed by Becker and Widmayer [2]. The *Priority Rectangle File* (PR-file, based on the R-file [16]) forms the backbone of their structure. A significant common characteristic of the PR-file and the Reactive-tree is that, in general, both store more important objects in higher levels. A few differences of the PR-file, compared to the Reactive-tree, are: objects of equal importance (priority) are not necessarily on the same level, and object-entries and tree-entries can not be stored in the same node.

Finally, other Reactive-trees should be considered which are able to deal efficiently with a non-hierarchical distribution of the number of objects over the importance levels, whilst sticking to the guideline that important objects are to be stored in the higher levels of the tree. This might be realized by changing the properties in such a manner that one tree level is allowed to contain multiple importance levels, but it is not (yet) clear how the Insert and Delete algorithms should be modified. This is subject to further research.

References

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

[2] Bruno Becker and Peter Widmayer. Spatial priority search: An access technique for scaleless maps. Technical report, Institut für Informatik, Universität Freiburg, June 1990.

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

[4] Kurt E. Brassel and Robert Weibel. A review and conceptual framework of automated map generalization. *International Journal of Geographical Information Systems*, 2(3):229–244, 1988.

[5] Nicholas R. Chrisman. Deficiencies of sheets and tiles: Building sheetless databases. *International Journal of Geographical Information Systems*, 4(2):157–167, 1990.

[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] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons, New York, 1973.

[8] A. Frank. Application of DBMS to land information systems. In *Proceedings of the Seventh International Conference on Very Large Data Bases*, pages 448–453, 1981.

[9] André Frank. Storage methods for space related data: The Field-tree. Technical Report Bericht Nr. 71, Eidgenössische Technische Hochschule Zürich, June 1983.

[10] Andrew U. Frank. Requirements for a database management system for a GIS. *Photogrammetric Engineering and Remote Sensing*, 54(11):1557–1564, November 1988.

[11] Andrew U. Frank and Renato Barrera. The Field-tree: A data structure for Geographic Information System. In *Symposium on the Design and Implementation of Large Spatial Databases, Santa Barbara, California*, pages 29–44. Lecture Notes in Computer Science 409, Springer Verlag, July 1989.

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

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

[14] Stephen C. Guptill. Speculations on seamless, scaleless cartographic data bases. In *Auto-Carto 9, Baltimore*, pages 436–443, April 1989.

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

[16] Andreas Hutflesz, Hans-Werner Six, and Peter Widmayer. The R-file: An efficient access structure for proximity queries. In *Proceedings IEEE Sixth International Conference on Data Engineering, Los Angeles, California*, pages 372–379, February 1990.

[17] Christopher B. Jones and Ian M. Abraham. Design considerations for a scale-independent cartographic database. In *Proceedings 2nd International Symposium on Spatial Data Handling, Seattle*, pages 348–398, 1986.

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

[19] Chris Laffra and Peter van Oosterom. Persistent graphical objects. In *Eurographics Workshop on Object Oriented Graphics*, June 1990.

[20] David M. Mark. Conceptual basis for geographic line generalization. In *Auto-Carto 9, Baltimore*, pages 68–77, April 1989.

[21] Robert B. McMaster. Automated line generalization. *Cartographica*, 24(2):74–111, 1987.

[22] J.-C. Müller. Optimum point density and compaction rates for the representation of geographic lines. In *Auto-Carto 8*, pages 221–230, 1987.

[23] Jean-Claude Müller. Rule based generalization: Potentials and impediments. In *4th International Symposium on Spatial Data Handling, Zürich, Switzerland*, pages 317–334, July 1990.

[24] Jean-Claude Müller. Rule based selection for small scale map generalization. Technical report, ITC Enschede, The Netherlands, 1990.

[25] Bradford G. Nickerson. *Automatic Cartographic Generalization For Linear Features*. PhD thesis, Rensselaer Polytechnic Institute, Troy, New York, April 1987.

[26] F.J. Ormeling and M.J. Kraak. *Kartografie: Ontwerp, productie en gebruik van kaarten* (in Dutch). Delftse Universitaire Pers, 1987.

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

[28] Urs Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1:244–256, 1972.

[29] Diane E. Richardson. Database design considerations for rule-based map feature selection. *ITC Journal*, 2:165–171, 1988.

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

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

[32] Jan van den Bos and Chris Laffra. Procol – A parallel object language with protocols. In *OOPSLA '89, New Orleans*, pages 95–102, October 1989.

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

[34] Peter van Oosterom. A modified binary space partitioning tree for Geographic Information Systems. *International Journal of Geographical Information Systems*, 4(2):133–146, 1990.

[35] Peter van Oosterom and Eric Claassen. Orientation insensitive indexing methods for geometric objects. In *4th International Symposium on Spatial Data Handling, Zürich, Switzerland*, pages 1016–1029, July 1990.

[36] Peter van Oosterom and Chris Laffra. Persistent graphical objects in Procol. In *TOOLS '90, Paris*, pages 271–283, June 1990.

[37] 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.