

## The VMO-Tree: A Dynamic Spatial Data Model for Maps

Weiping Yang and Christopher Gold

Chaire industrielle en géomatique  
Centre de recherche en géomatique, Université Laval  
Québec, Qc, Canada G1K 7P4  
E-mail: weiping@gmt.ulaval.ca  
Tel: (4 18) 656-3652 Fax: (4 18) 656-74 11

**Abstract:** This paper describes an object-oriented approach to model a set of 2D vector data, depicting points, lines, and polygons. These primitive spatial objects are embedded in the Euclidean space with the dynamic Voronoi diagram. A spatial object condensation technique is used to partition the Voronoi diagram into smaller subspaces. Each subspace defines a bounded complex object and constitutes a node of the VMO-Tree (Voronoi based Map Object Tree), a novel data model describing relationships of condensed objects. The VMO-Tree can be constructed by either top-down partition, bottom-up aggregation, or combination of the two. Features of the VMO-Tree include: representing spatial objects in different levels of detail; node containing topologically structured complex objects, some of them child nodes; being able to search vertically between nodes of the tree and to navigate horizontally in each node in a seamless fashion; loading and down-loading spatial objects contained in a node dynamically; complex objects in the tree are independent map sheets and can be operated on separately; each map sheet corresponds to a disk page and has a minimized memory occupation.

## 1. Introduction

Contemporary mapping applications require that the spatial data be handled with a great deal of flexibility. This need can be imposed on all aspects of processing data which might be enumerated as: viewing, storing, manipulating, analyzing, and communicating. Ideally, any interesting data should be organized as an operable unit or object, the change of focus back and forth amongst objects should be smooth and automatic, and the system should allow both outlined and detailed views. From the data structure point of view, these requirements imply that the spatial data be topologically structured in a hierarchical fashion, with different levels of detail. Each node accommodates a region of arbitrary polygonal shape which again contains isolated points, lineal features, and smaller regions of different complexity, and the nodes at each level correspond to disk pages.

The paper describes an object-oriented dynamic data model satisfying the above needs. The model is based on the kinematic Voronoi diagram [Gold 1990; Roos 1991; Gold et al. 1995] which provides fundamental tools to manage and analyze primitive map objects. The complex objects are constructed from the primitive Voronoi diagram using a spatial object condensation technique [Yang and Gold 1995] which effectively partitions the primitive Voronoi diagram into its subsets and vice versa. The spatial object condensation implies partitioning the topological space and preserving the result in the memory model in a compressed fashion. Each subspace, i.e. spatial object, is viewed as a constrained child map sheet and hence inherits properties and methods of its ancestors. Besides, a child map sheet can have children. On top of this is the VMO-Tree, the spatial object manager. Structurally, the VMO-Tree reflects the object class hierarchy. Functionally, it provides a platform to manipulate spatial objects in both single and multiple themes.

A similar attempt towards this scenario of handling spatial data is reported in [Bertolotto and De Floriani, 1994], a HPEG (Hierarchical Plane Euclidean Graph) based planar map model for a

multiresolution representation of a region. The HPEG model recursively decomposes regions into smaller ones. At the same time, region boundaries at one level can be simplified by creating 1-homotopies with some line generation algorithms, where  $\varepsilon$  specifies the radius of a band convoluted from a chain of edges. The major difference between the HPEG model and the VMO-Tree model lies in the underlying geometric and topological data structures. The HPEG data structure is stemmed from DCEL [Preparata and Shamos 1985], an encoding of an edge-oriented planar subdivisions. The Voronoi data structure partitions space based on the adjacent neighbourhood information. Besides, it is not clear how the HPEG can be partitioned into disk pages and retrieved efficiently from them. We believe that the ability to represent maps as manageable objects in both spatial and memory subspaces is a key feature towards persistent programming. The detailed comparison between the two models does not form the focus of this paper.

Section 2 of this paper defines the Voronoi diagram of lines and points, and its dual topological structure, the Delaunay triangulation, over which the computer data structures are designed to represent the Voronoi diagram. Disadvantages of the primitive Voronoi diagram applied to large data sets are pointed out. It is followed by a brief description of the spatial object condensation concepts in Section 3. The major contribution of this paper, the VMO-Tree class structure, is described in Section 4. Section 5 illustrates the visual interface of the VMO-Tree and its applications. Conclusions and future work are given in Section 6.

## **2. The Dynamic Voronoi Diagram**

Given a set of  $n$  ( $n > 1$ ) objects  $S = \{\text{points, line segments}\}$  in the  $R^2$  Euclidean space, the dynamic Voronoi diagram,  $V(s)$ , is constructed in an incremental approach which partitions the plane into

Voronoi regions,  $v$ , such that for two distinct objects  $s_i, s_j \in S$ , the dominance of  $s_i$  over  $s_j$  is defined as the subset of the plane being at least as close to  $s_i$  as to  $s_j$ :

$$D(s_i, s_j) = \{x \in R^2 \mid \delta(x, s_i) \leq \delta(x, s_j)\},$$

for  $\delta$  denoting the Euclidean distance function.  $D(s_i, s_j)$  is a closed half plane bounded by the bisector of  $s_i$  and  $s_j$ , denoted by  $B(s_i, s_j)$ :

$$B(s_i, s_j) = \partial D(s_i, s_j) = \{x \in R^2 \mid \delta(x, s_i) = \delta(x, s_j)\}$$

The Voronoi region  $v(s_i)$  is the portion of the plane lying, in all of the dominances of  $s_i$  over the remaining objects in  $S$ :

$$v(s_i) = \bigcap_{s_j \in S - \{s_i\}} D(s_i, s_j).$$

If two bisectors  $B(p, q)$  and  $B(p, r)$  ( $p, q$ , and  $r \in S$ ) intersect, i.e.

$$B(p, q) \cap B(p, r) \neq \emptyset,$$

the intersection is called a *Voronoi point* and the bisector delimited by two consecutive Voronoi points is called a *Voronoi edge*. Two objects are *neighbours* if they share a common Voronoi edge (possibly extended to infinity). The Voronoi point has an equal distance to at least three objects and is therefore the circumcentre defined by these objects.

If  $S$  is a point set, the region associated with a  $p \in S$  will be the intersection of all half-planes containing  $p$  and delimited by the bisectors between  $p$  and the other members in  $S$ . It follows that the Voronoi regions are (possibly unbounded) convex polygons. This nice property does not hold for a general Voronoi region if  $S$  contains points and lines. The bisector between a point and a line is no longer a straight line but a locus of points bisecting them. While the definition of the dominance  $D(p, q)$  still holds the bound  $B(p, q)$  may become a parabola and the Voronoi region  $v(p)$  is not, in general, a convex polygon. Figure 1 shows the Voronoi diagram (gray) of points and lines (dark).

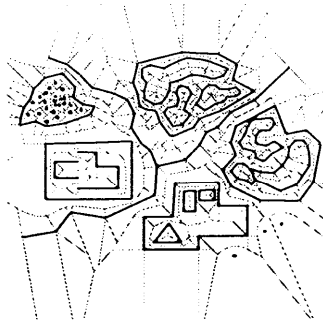


Figure 1 The Voronoi Diagram of Points and Lines

## 2.1 The Delaunay Triangulation

The Voronoi edge separating  $p$  and  $q$  indicates that  $p, q$  are neighbours. A straight line linking  $p, q$  is used to represent this relationship. It follows that for  $e$  Voronoi edges we can have the same number of straight lines. Each Voronoi point corresponds to one triangle formed by three straight lines linking three objects on the circumcircle<sup>1</sup>. The union of these triangles is called the dual structure of the Voronoi diagram, the Delaunay Triangulation  $D(S)$ . Each straight line linking two objects is called a Delaunay edge. For  $S = \{\text{points, line segments}\}$ , the corresponding Delaunay edges between a line and a point can be graphically implemented by linking the point and the middle of the line segment. Figure 2 illustrates a Delaunay triangulation (dotted) for some lines and points (solid).

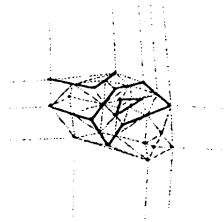


Figure 2. The Delaunay Triangulation of Points and Lines

---

<sup>1</sup> By saying so we assume that  $S$  is in general position, i.e. there are no cocircular cases and not all objects in  $S$  are colinear. These degenerate cases can be handled by some additional rules.

Unlike its counterpart for a point set, the Delaunay triangulation for points and lines is not necessarily optimal in angles. It simply becomes a graph representing adjacency relationships between objects. Nevertheless, the circle tangent to three vertex objects is object-free, i.e. none of the given objects is contained in its interior.

In common practice, the Delaunay triangulation is the actual topological structure used to model its dual Voronoi diagram in the computer. One of the reasons for this is due to the simplicity of the data structure needed to represent the Delaunay triangulation.

## **2.2 The Memory Space Related Problems**

The memory *space* is referred to as the computer memory allocated to store spatial objects and their topological structures. In our case, the topological structure is the Delaunay triangulation. Two problems can be observed in applying the dynamic Voronoi diagram as a fundamental data model for a dynamic GIS, within the framework of incremental construction. First, compared with other geometric and topological data models embedded in most current GISs, the Voronoi diagram consumes more memory space. This observation is obvious because, in addition to creating indices to geometric objects and their coordinates, the structure must maintain the topology for each triangle. For a complex polygonal map composed of hundreds of thousands of short line segments, the size of the Delaunay triangulation could be extremely large. Second, since the Voronoi diagram is generated incrementally, the input of objects does not necessarily follow a particular order. This may result in an instance of the data structure with poor spatial indexing. That is, objects and triangulation that are close in space may be stored far apart in memory, possibly scattered in different data blocks. This makes the ordering of the space according to some pattern impossible. It follows

that no matter how big a map is. all the object and topological data must be loaded in memory in order to ensure the presence of the neighbourhood around any area of interest.

Large memory use with unnecessarily detailed objects lowers the overall performance of the system and hinders operators from concentrating on more important properties of a data set. Managing spatial data in a non-splittable fashion fails to meet the requirements for modern data structures which must be dynamic, hierarchical, with levels of details, and divisible into disk pages [van Oosterom 1993].

If some part of the Voronoi diagram of detailed objects can be taken away from the organizing structure and represented by some compound objects, the number of objects present in memory would be smaller. Again, if the separated compound object preserves its original topological partitions of the space, it can be loaded into memory any time as an independent workspace. The whole space can then be managed seamlessly as if there is no partition. Furthermore, if scattered indices for spatial objects and their local topological relationships in a memory subspace can be aggregated, they can be more efficiently managed and retrieved. These expectations can be achieved by introducing the concept of condensed spatial objects.

### **3. The Spatial Condensation Technique**

The spatial object condensation technique is composed of two steps. The first step partitions the Voronoi diagram into disk pages and the second one rearranges the memory subspaces such that the gaps in memory space are eliminated and the resulting storage of spatial objects and their topology become contiguous.

### 3.1 The Justification of Partitioning Map Data and Their Topological Relationships

The general procedure condensing spatial objects utilizes the properties of the Voronoi diagram of line segments. In applications, the concept of “polygon” is applied to describe aggregate properties of some spatial phenomenon. A polygon is a complex object with a closed boundary within which other types of spatial objects may exist. To simplify the description, we assume that the outer boundary of a complex polygon is a simple closed loop, i.e. every node in the loop is of order 2.

The Jordan Curve Theorem indicates that a simple polygon boundary divides the space into internal and external regions. This implies that the point set of an object space  $X$  can be divided into two subsets  $X_1$  and  $X_2$ . Each subset  $X_i$  has three distinct parts: the interior,  $X_i^\circ$ , the boundary  $\partial X_i$ , and the exterior  $\overline{X_i}$ . The intersection of  $X_1$  and  $X_2$  is the dividing Jordan curve. The Jordan theorem is also true for the topological space of  $X$  mapped with the Voronoi diagram. Intuitively, it is observed that the localized construction of the Voronoi polygons for each  $x \in X_i^\circ$  is *self contained*, i.e., no point  $y \in \overline{X_i}$  needs to be involved. This observation can be proved [Yang and Gold 1995] deductively using the neighbourhood relationships. The ability to separate both object spaces and operations over their topologically mapped spaces ensures that the map data and their topology delimited by the Jordan curve boundaries can be partitioned and maintained individually.

### 3.2 The Partitioning Technique

The difficulty of partitioning a Voronoi diagram lies in its computer implementation. Most topological data structures explicitly encode certain topological relationships among adjacent objects. For example, the DCEL based data structures contain boundary and co-boundary information. Each edge structure encodes polygons on both sides. This is also true for the Delaunay triangulation. In our implementation, indices of adjacent triangles are encoded in each triangle. After

the partition. the memory space of the original objects and topological structures is separated and will be finally dumped onto different disk pages. Some indices pointing to triangles in other memory subspaces will become invalid. This causes a dead end for the traversal of the topological structures in one partitioned object.

The technique [Yang and Gold 1995] used to overcome this problem employs “critical triangles” (Figure 3) which are created and embedded in boundary line segments. It can be devised that the pointers to another subspace be bent such that they point back to the critical triangles belonging to the same subspace (Figure 4). This process transfers dead ends into continuously traversable paths.

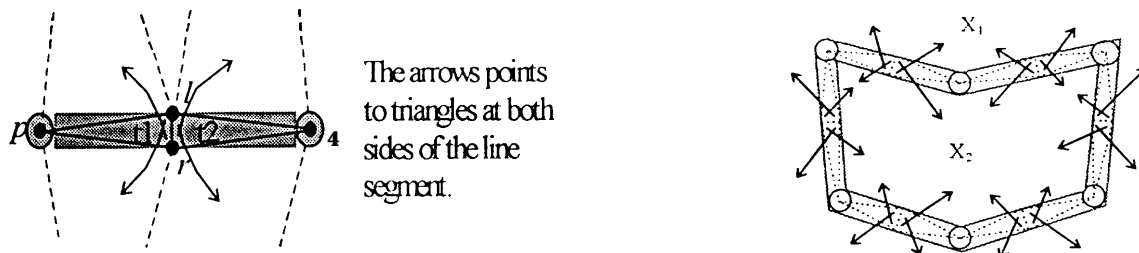


Figure 3 Critical Triangles and Their Pointers

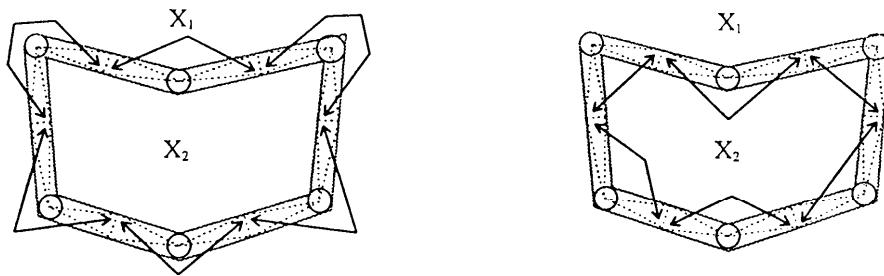


Figure 4 Bending Pointers in Critical Triangles

### 3.3 Partitioning and Compressing the Memory Space

The creation of critical triangles and the modification of boundary conditions effectively make the memory space of the topological structure self contained. The next step of the condensation technique needs to identify objects and their topological relationships in one subspace. Once

identified, they can be written into the newly allocated subspace and later removed from the original subspace. Unfortunately, due to the incremental construction, the memory space occupied by one subspace may not be contiguous. This makes sequential access to objects and topology in one subspace impossible. Some spatial traversal technique has to be used. In our implementation, we use the flood fill algorithm to traverse the topological space in spiral order. For each triangle and object visited, instead of copying its original index to the new memory space, a new consecutive index number is assigned to it. This eliminates gaps in memory space between two spatially adjacent objects (Figure 5).

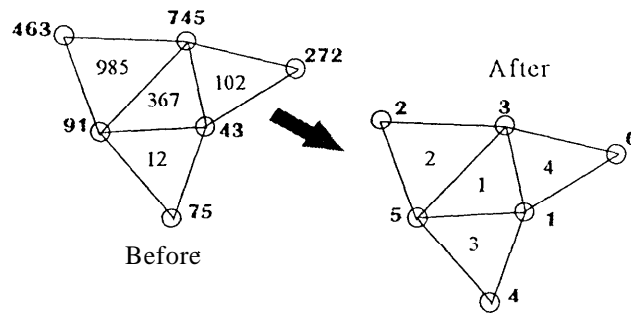


Figure 5 Flood Fill Memory Compressing

### 3.4 The Result of the Condensation Technique

In summary, the condensation technique partitions spatial objects and their topological relationships in a map into subsets for both the data structures and their memory space. For example, five smaller map objects (Figure 6) are partitioned from the map shown in Figure 1. Each partitioned object is a map of the same type as its ancestor. It can be loaded and worked with independently. Even smaller objects can be created from it. As is illustrated in Figure 7, three more objects are created from the object labelled “C”.

The time complexity of the condensation algorithms is composed of two parts. The partition of objects and their topological relationships uses  $O(m \times k)$ , where  $m$  is the number of line segments forming the partitioning boundary and  $k$  is the average number of adjacent triangles around the boundary. Writing and compressing the memory space takes  $O(nt)$  time, where  $nt$  is the number of triangles in the new object which is generally 2 ~ 4 time bigger than the number of more primitive objects contained in the new object.

Although the examples given include only objects with simple boundaries. The method applies to any closed polygons with complex boundaries.

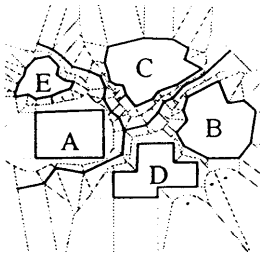


Figure 6 A Partition of the Map in Figure 1

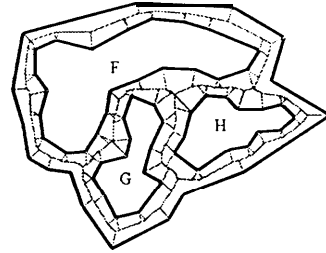


Figure 7 A Partition of Object C in Figure 5

#### 4. The VMO-Tree

The spatial object condensation creates complex polygonal objects. Each object resides in a separate memory space and can be saved as a file in a storage medium. The portion of its memory space in the original map will then be released. This results in a reduction of the memory for the original map. In this section, we will describe the relationship between the new object and the object from which it is created. A new class named VMO-Tree (Voronoi based Map Object Tree) will be introduced. The VMO-Tree is a graph representation of the relationships of map objects. The prime roles of the VMO-Tree are managing complex objects and serving as a platform for transactions upon these objects.

## 4.1 The Data Model for the VMO-Tree Class

The objects created by condensing naturally form a hierarchical relationship where the object from which a partition is made is the parent and the newly partitioned object is the child. As Figures 6 and 7 show, a parent can have a number of children and each child itself can have children. When a complex polygon is condensed, the partitioning boundary is duplicated in both parent and child objects. At the parent level, the area enclosed by the boundary is a simple polygon with all detail suppressed. Topologically, this area is equivalent to the closed disc. As mentioned earlier, the boundary is a simple loop and is a Jordan curve. If we attach a label to the condensed object, the internal side of the boundary will have the same label. At the child level, the detail is expanded and the external side of the boundary has the same label as that of its parent.

In what follows, four aspects of the data model for the VMO-Tree class will be identified. They are namely: components of the class, data structures representing components and their relationships, operations on components, and constraints over operations.

### 4.1.1 The Components of the VMO-Tree Class

**Definition 1:** The *outline image* is the partitioning boundary left on the subspace  $X$  from which the space partitioning occurs. The *object outline* is the partitioning boundary duplicated on the partitioned subspace  $Y$  whose extent is confined in the enclosure of  $Y^\circ$ .

**Definition 2:** The class *map* is a quaternion  $A4 = (P, L, R, A)$ , where  $P$  is a class of points,  $L$  is a class of lines including outline images of condensed objects,  $R$  is a class of regions with closed boundaries, and  $A$  is a class of condensed objects.

Within the four subclasses, the point class is the most primitive one, the other three classes are aggregated from relatively simpler ones. For example, the compositions of  $R$  and  $A$  are  $R = (P, L, R_s)$  with  $R_s$  denoting smaller regions enclosed in  $R$ , and  $A = (P, L, R, A_s)$  with  $A_s$  denoting smaller condensed objects in  $A$ , respectively. Since  $A$  and  $A_4$  have the same composition of subclasses, they belong to the same map class. Therefore, a condensed object is also called a map in a recursive way.

Both  $R$  and  $A$  represent areal objects. They differ in that the components in  $r \in R$  are present in the same map containing  $r$ , while the components in  $a \in A$  are dropped from the memory space for the map. Only the outline image of  $a$  is preserved in the map.

A map may not have all types of objects presented. The following combinations are valid cases: 1)  $M = (P, L, R, \emptyset) = (P, L, R)$ , 2)  $M = (P, \emptyset, \emptyset, \emptyset) = (P)$ , 3)  $M = (P, L, \emptyset, \emptyset) = (P, L)$ , and 4)  $M = (\emptyset, \emptyset, \emptyset, \emptyset) = \emptyset$ . A map with no components is called a *null map* or *null object*.

**Definition 3:** An *object embedding* of a map object is a layout of its composing objects in the  $E^2$  space.

For a single map, the object embedding disallows general intersection between objects, following the concept of the *single valued vector map* [Molenaar 1989]. This ensures the planarity of the map. The object imbedding can be dynamic. That is, the components of the object can be changed. However, any change must be confined to the extent of the object outline.

**Definition 4:** A *topological embedding* of a map is a realization of topological relationships over the object embedding of the map. The topological relationships are specified in the topological data structures.

Note that the topological embedding of a map does not apply to the interior of condensed objects in the map because the object embeddings of these objects are suppressed. However, the

outline image of a condensed object is topologically embedded in the map. The topological embedding is also dynamic, due to the dynamic property of the object embedding.

#### 4.1.2 The VMO-Tree Structure

**Definition 5:** A VMO-Tree is a graph  $G = (N, E)$  with node set  $N$  and edge set  $E$ . The node set  $N$  is of the map class and each node has a label. For any pair  $\langle x, y \rangle \in N$ , an edge  $e \in E$  exists between  $x$  and  $y$  iff the object embedding of  $x$  contains the outline image of  $y$  and  $y$  has both object and topological embeddings, and both of them are confined in its outline duplication. The node  $x$  is called a parent, and the node  $y$  is a child.

The construction of a VMO-Tree can proceed from either top-down, bottom-up, or a combination of both. There are two scenarios when the top-down strategy is used. In either case, the root node is first created. One scenario of the top-down approach condenses a null object for each smaller area such that the whole study region is completely partitioned by the outline images of these null objects. These null objects are added into the VMO-Tree at level 1. Each null object can then be worked on independently and even smaller objects are added in the tree at lower levels (Figure 8).

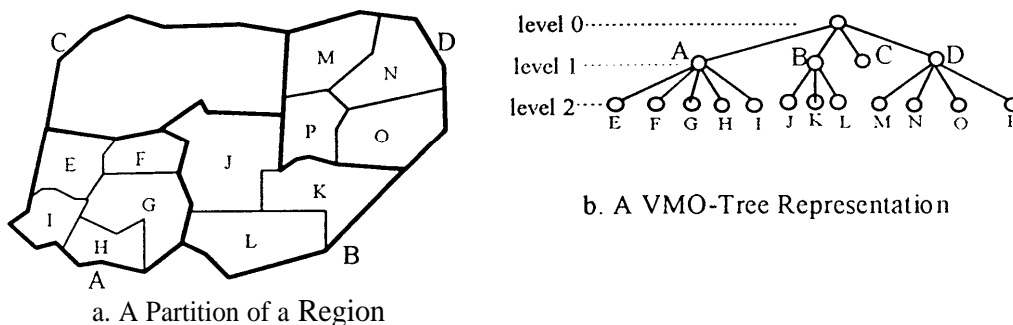


Figure 8 The Top-Down Partition and its VMO-Tree

Another scenario of the top-down construction works with the root map and adds components in it. Some heuristic approach can be taken to aid the decision on where and when a cluster of components can be condensed. Whenever a new object is dynamically condensed, the node representing the object is inserted into the tree.

The bottom-up strategy works in a reverse way where the smallest distinct objects are first composed as maps separately. There are also two ways of constructing the tree. The first one creates the root and inserts into it all the object nodes from the first step. These objects will be at level 1. This results in a flat tree where all children have the root as the parent. Then a generation process takes place which aggregates smaller objects into bigger ones. When an bigger object aggregated from  $m$  smaller objects at level  $i$  is formed, it becomes the parent of the  $m$  objects. Note that the aggregation happens at level  $i-1$ . The new node will therefore be at level  $i$  and the  $m$  children are dropped to level  $i + 1$ . The  $m$  pointers of the original parent at level  $i - 1$  will be released and the tree becomes narrower by  $m - 1$  branches at level  $i$ . The generation process can be performed heuristically.

The second method of bottom-up tree construction generalizes more complex objects without knowledge of the root. When an aggregate object is created, it serves temporarily as a root of a smaller tree. A forest can exist during the whole process. A bigger tree can be composed by combining a number of smaller trees.

From a cartographer's point of view, the top-down approach works from smaller scale, larger sized objects towards larger scale, smaller objects. The deeper down in the VMO-Tree, the more details an object can expose. On the contrary, the bottom-up approach works from larger scale, smaller sized objects towards smaller scale, larger objects. The higher a node is in the VMO-Tree, the more abstract it becomes.

The combined construction repeatedly uses partition and aggregation processes. The description is straight-forward and is omitted from this paper.

Inserting an existing condensed object into the VMO-Tree and aggregating smaller condensed objects into a bigger ones are compound operations each of which takes several steps. The major steps for insertion are to duplicate the outline image of the new object in the proper node map and attach the new object to the tree. For aggregating, the outline of the new object can either be the union of the outlines of the smaller objects, neglecting the overlapping part, or sometimes bigger, depending on the heuristic decision.

#### 4.1.3 Constraints upon the VMO-Tree Construction

The VMO-Tree is a topological structure mapped from the partitioning of another topological space. In order to make VMO-Tree well defined, the constraints guiding partitions must be defined.

Let  $A = \{a_1, a_2, \dots, a_n\}$  be a finite set of condensed objects in one map. These constraints state:

**Constraint 1:** The number of condensed objects in one map corresponds to the number of children under the node representing the map.

**Constraint 2:** For  $a_i, a_j \in A$ , ( $i \neq j$ ),  $a_i^\circ \cap a_j^\circ = \emptyset$ . That is, interior spaces of all the siblings must be disjoint at the same level. Overlapped or partially overlapped outline images are disallowed. The non-empty intersection of distinct member of  $A$  may only happens on the outline images.

**Constraint 3:**  $\bigcup_{a \in A} \text{outline}(a) \subset \text{outline}(A)$ . That is, any outline image of a child will be confined in the outline of the map.

### 3.2 Properties of the VMO-Tree

The VMO-Tree possesses the following properties which are stated as:

**Theorem 1:** The VMO-Tree is acyclic.

Proof: If the VMO-Tree is cyclic, there must be a one way path starting and terminating at the same node. Assume a subset  $(a, b, c) \subset N$  are three distinct nodes and  $a \rightarrow b \rightarrow c \rightarrow a$  is such a path. By the definition of the VMO-Tree, node  $b$  is a child of  $a$  and node  $c$  a child of  $b$ , the outline of  $b$  is relatively smaller than that of  $a$  and the outline of  $c$  relatively smaller than that of  $b$ . From geometric transitivity we get that the outline of  $c$  is smaller than that of  $a$ . However, the piece of the path  $c \rightarrow a$  indicates that the outline of  $a$  is smaller than that of  $c$ . Conflicting.

**Theorem 2:** Given a partition strategy for a topological space  $X$ , the mapping between the partition and the VMO-Tree has one-one correspondence.

Proof The partition strategy determines: 1) the number of nodes in the VMO-Tree; 2) the parent-child relationships between the partitioning and the partitioned objects. Denote the mapping from the partition satisfying the above constraints to the VMO-Tree  $f$ . It is not difficult to see that for each partitioned object  $a$ , a node  $f(a)$  exists in the VMO-Tree. Reversely, for each node  $n$  in the VMO-Tree, there exists a partition in the parent map such that  $f^{-1}(n) = a$ .

### 4.3 Operations over the VMO-Tree

There are five categories of operations over the VMO-Tree: building, set-theoretic, spatial queries, attribute database linkage, and network communications. The first three categories are spatial operations. More formal and detail discussions about some of these operations can be found in Worboys [1992].

The building operations insert and delete nodes from the tree. The building operations also contains basic editing tools which allow modification over components of an object.

The set-theoretic operations work on either single and multiple themes. These include calculating unions, intersections, and completions, with respect to given themes and objects. New instances of the VMO-Tree can be generated out of a set operation. An example of this calculates the intersection of two objects, each from a VMO-Tree instance. A third instance of the VMO-Tree can result.

Spatial queries can be greatly facilitated by the VMO-Tree, which provides a fast search along edges of the tree, and the topological embedding within each spatial object, which enables navigation and searches over components of a map. This ability of search objects vertically and horizontally resembles that of the  $B^+$  tree.

How to store and manage attribute data of spatial objects is still an open question. The majority of GIS implementing a relational model typically adopt a hybrid architecture. Due to the special characteristics of spatial data, the future options consist of two major directions: extending relational systems with specialized functionality for geo-information; or moving to radically new approaches, such as the object-oriented systems [Worboys 1994]. The option of adding specialized functions to relational databases not only can speed up the development, but also complies to the reality that most of the attribute data are maintained by relational databases.

The VMO-Tree manages map objects, some of which may reside in or be accessed from remote sites. The object networking functions have to take care the data security, integrity and concurrency controls, and dynamic communications.

## 5. The Interface for the VMO-Tree Operations

One of the objectives of the spatial data handling system based on the VMO-Tree is that any operations on spatial objects should be made easy. This can be achieved by designing a well defined visual user interface. For example, the VMO-Tree can be represented by a visual component, following the visual programming paradigm. The VMO-Tree component responds to some events triggered by users and the methods built in the VMO-Tree class can be activated by events. Figures 9, 10, 11 illustrate a prototype interface from which some typical operations can be facilitated.

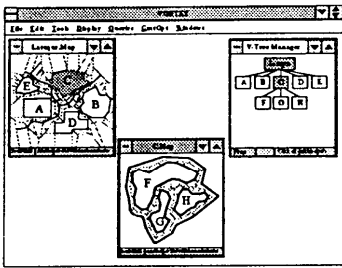


Figure 9 Accessing Objects in the VMO-Tree

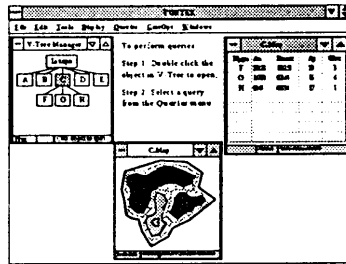


Figure 10 Viewing Attribute Method

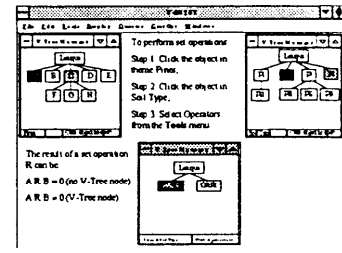


Figure 11 Set-Theoretic Operations through the VMO-Tree

## 6. Conclusions and Future Work

In this paper, an object-oriented spatial data model for managing maps is proposed. The objects are condensed from the primitive Voronoi diagram, each having a separate memory subspace corresponding to disk pages. Condensed objects are managed by an acyclic VMO-Tree which is one-one correspondent to the map partitions. The VMO-Tree can be built by either top-down subdividing, or bottom-up aggregating of map objects, or a combination of the two. The construction constraints ensures that the projected tree is well defined.

The current specifications about the data model are preliminary. The authors are developing detailed descriptions on relationships of spatial components, the operations, the boundary conditions for dynamic modifications, and the visual interface of the data model. The attribute data storage and networking issues will be further studied in the near future.

## References

- Bertolotto, M. and L. De Floriani. Multiresolution Topological Maps. In *Advanced Geographic Data Modelling: Spatial Data Modelling and Query Languages for 2D and 3D Applications*. Eds. M. Molenaar and S. de Hoop. *Publications on Geodesy, New Series* No. 40, Delft, Netherlands, 1994, pp. 179- 190.
- Gold, C.M. Spatial Data Structures: the Extension from One to Two Dimensions, in: L.F. Pau (ed.), *Mapping and Spatial Modelling for Navigation, NA TOASI Series, F, No. 65*, Springer, Berlin, 1990, pp. 11-39.
- Gold, C.M., P. R. Remmele, and T. Roos. Voronoi Diagrams of Line Segments Made Easy. In *Proceedings of the 7th Canadian Conference on Computational Geometry*, August 1995, Québec, Canada, pp. 223-28.
- Molenaar, M. Single Valued Vector Maps - A Concept in Geographic Information Systems. *Geographical Information Systems*, (2) 1, 1989, pp. 18-26.
- Preparata, F. and M. Shamos. *Computational Geometry: An Introduction* Springer-Verlag, New York, 1985.
- Roos, T. Dynamic Voronoi Diagrams. *Ph.D Thesis*, Universität Würzburg, Switzerland, 1991.
- van Oosterom, P. *Reactive Data Structures for Geographic Information Systems*. Oxford University Press, 1993.
- Worboys, M.F. Object-Oriented Approaches to Geo-Reference Information. *Int. J. Geographic Information Systems*, (8)4, 1994, pp. 385-399.
- Worboys, M.F. A Generic Model for Planar Geographic Objects. *Int. J. Geographic Information Systems*, (6)5, 1992, pp. 353-72.
- Yang, W.P. and C.M. Gold. Dynamic Spatial Object Condensation Based on the Voronoi Diagram. In *Proceedings 4th Symposium of LIESMARS*, Wuhan, China, October 1995, pp. 134-145.