Access and update of a triangular irregular network from a quadtree

Antonio Ruiz
Ismael Colomina
Institut Cartogràfic de Catalunya
Balmes, 209-211. 08006-Barcelona
E-mail: toni@icc.es, ismael@icc.es

Lluís Pérez
Universitat Politècnica de Catalunya
Dep. de Llenguatges i Sistemes Informàtics
Secció d'Informàtica Gràfica
Av. Diagonal 647, 08028-Barcelona
E-mail: lpv@lsi.upc.es

Abstract

We present two dynamic structures that jointly allow a fast access and update of a large size Triangular Irregular Network (TIN). These two structures are variations of the Double Connected Edge List (DCEL) and the Bucket Point Region Quadtree (Bucket PR-Quadtree).

1 Introduction

1.1 Motivation and requirements

The Institut Cartogràfic de Catalunya (Cartographic Institute of Catalonia) saw the need to build a database to maintain the Digital Elevation Model (DEM) of Catalonia. The problem has been subdivided in two parts. In a first step the region to model is triangulated and then, a surface model is built on the triangular regions, or, more generally, on arbitrary polygonal regions that are to be pointed at by the triangles. This paper deals with triangular tessellations and the design problems inherent to the procedures of handling the huge amount of data being generated (in the order of three million triangles or 10 Gbytes). Retrieval and update of the blocks of adjacent triangles must be efficient, easy and fast; this includes inserting and deleting points and required edges. The triangulation chosen is the Constrained Delaunay Triangulation (CDT) [?]. Figure 1 shows an example of triangulation using a well known data set [?]. The triangle vertices and the required edges must be selected with geomorphological criteria, to adapt the model to the real surface up to a (previously) given accuracy. Neither these vertices nor the required edges should be conditioned by the data structure chosen to store them.

1.2 The DCEL structure

The Double Connected Edge List (DCEL), is a well-established structure in Computational Geometry [?]. It provides tools for the storage and management of any planar graph, and, in particular, of a TIN. It is a list of oriented edges. For each edge, the DCEL stores two pointers **V1** and **V2** to its vertices, then two pointers **E1** and **E2** to the first neighboring edge that will be found turning around **V1** and **V2** counterclockwise, and optionally, two more pointers **F1** and

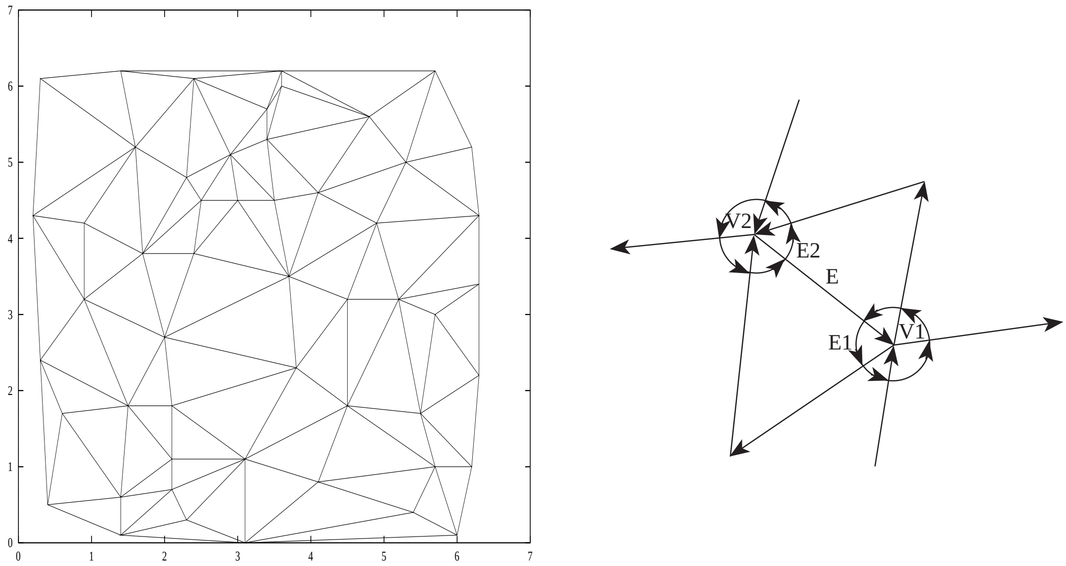

Figure 1: Triangulation of the data in [Dav73] and DCEL structure

F2 to the faces sharing this edge. An additional attribute **req** indicates whether the edge is requiered or not (figure 1).

1.3 Access to triangles.

We can use Lawson's algorithm [?] to locate the triangle containing a given point: The algorithm starts at the current location (meaning the edge under study). If the point lies on the right, pointer **E1** will be followed, if it lies on the left then **E2**. The iterative procedure stops when a turn around the three sides of the same triangle has been completed. Pointers to adjacent edges allow navigation through the structure traveling from one edge to the next in constant time $O(1)$, but, if the algorithm has to reach a distant edge, then (assuming the vertex distribution is roughly uniform) the expected cost is $O(n^{\frac{1}{2}})$ [?], where n is the number of vertices in the TIN, although the worst case time cost is $\Theta(n^2)$. For approximately uniform distributions of points, the DCEL data structure is equivalent to the Guibas & Stolfi **quad-edge** for oriented surfaces [?]. Rotations to the left can be substituted by a proper number of rotations to the right. The increase of computational cost is outweighed by the savings in allocated memory.

Lawson's algorithm [?] is adequate for local searches, when all edges to traverse are kept on the main (fastest) memory. When we deal with large amounts of data, we need some alternative procedure to retrieve –as fast as possible– triangles that are not necessarily adjacent, nor near. We can order edges or, alternatively, overlay a structure allowing access to "distant" edges, without the need to follow a thread of intermediate pointers that, in general, can reside in different sectors ("pages") of secondary (mass) storage (typically, a disk).

2 Previous work: Related Data Structures

2.1 Delaunay Tree

A TIN structure is said to be "dynamic" if it allows the variation of its forming vertices in time, through insertion or deletion.

Boissonat and Teillaud [?] introduced a semi-dynamic structured called Delaunay Tree; it allows insertion but not deletion of vertices. The structure keeps the log of the incremental construction of the triangulation. When a new point is inserted, triangles $\{T_i \mid i = 1, \cdots, n\}$ in conflict with the new point M are not erased. Let $R(M)$ be the region that would be created by the union of those triangles in conflict and $F(M)$ the border of that region. A triangle in conflict, T_i, will have a side F belonging to $F(M)$. Let N be the triangle that shares this side F with T_i. When M is inserted, the new triangles are formed joining the vertices of $F(M)$ with M. Let S be the triangle just formed with a side F. To construct the Delaunay tree, the algorithm builds pointers from T_i and N to S. S is said to be a son of T_i and a stepson of N.

Each triangle can have a son and a list of stepsons for each of its edges. Triangles that belong to the final triangulation are those that have no son (although they may have stepsons).

Devillers et al. [?] present, as a development of [?], an algorithm for the deletion of vertices. The state of the Delaunay tree after deletion is the same as it would be if the point had never been inserted. The tree allows the stochastic insertion of points in time $O(n \log n)$ and the deletion in $O(n \log \log n)$. The space allocated to store the tree is $O(n)$ but the constant of proportionallity is quite large. For each point added, j triangles are destroyed and $j+2$ triangles are created when the triangle lies in the inner region and $j+1$ when it belongs to the convex hull. The number of nodes in the Delaunay tree is slightly less than thrice the number of triangles in the Delaunay triangulation for random insertions [?].

2.2 PMR quadtree

Nelson and Samet [?] introduce the *PMR (Polygonal Map Random) quadtree*. This structure allows the exact storage of edges, without fragmentation at the frontiers of the leaves. The absence of fragmentation is very important because the limitations in the representation of real numbers imply faulty alignments of edges.

This quadtree is a region quadtree (**R-quadtree**) in which each node is subdivided in four leaves of equal size (by partition of each side in each dimension). An edge is assigned to each leave that it crosses through. The **random** adjective in the name of the method derives from the fact that the criterion for subdivi-

Threshold of subdivision	Occupation experimental	theory
1	1.22	1.22
2	1.62	1.22
3	2.14	2.20
4	2.71	2.81
5	3.31	3.46
6	3.90	4.14
7	4.53	4.81
8	5.10	5.51

Table 1: Average occupation of the *PMR- quadtree*, Samet 1990

sion is probabilistic: if the number of edges inside a block goes beyond a given threshold value, then the block is divided into quadrants only once. Since the number of edges ending at the same vertex is unbounded, if that number reaches the threshold value for subdivision of a leaf, the procedure will not be able to reduce through subdivision the number of edges that a leaf contains. Therefore, some leaves might have a occupation density greater than the threshold value for subdivision and, then, the resulting quadtree depends on the insertion order.

A linked list is an appropriate structure to store edges. The average occupation density (number of points/capacity) of the quadtree leaves is low. Nelson and Samet [?] have obtained results of one half the subdivision threshold value for this density (see table 1). It is difficult to get an estimate and the authors introduce several simplificative hypotheses to reach those results. One of them is the assumption that the ratio of nodes with a given occupation density is independent of the size of the node. Discrepancies between theoretical estimates and experimental results increase with the value of the subdivision threshold.

In the study of the behavior of the PR quadtree, Nelson and Samet found some anomalies that they named *aging* and *phasing* [?]. *Aging* is the tendency of older nodes to grow in size, and to show an increase in the occupation density when compared to smaller nodes. *Phasing* occurs when we insert points that lie on a uniform distribution. Nodes tend to become crowded and subdivided with logarithmic periodicity each time the number of points is multiplied by four. Nodes of the same size tend to become full and subdivided simultaneously and this entails severe oscillations in the average occupation density versus the number of points.

3 Modified DCEL structure and its associated quadtree

We present a modified DCEL structure that will be stored in a region quadtree where the insertion rule is as follows:

Each edge is stored in those quadtree leaves that contain any of its vertices. One given edge can be stored in one or two leaves of the quadtree, not necessarily adjacent. The **leaf criterion** leads the decision of whether a leaf must be subdivided or not. As with the **PMR-quadtree**, it is an overflow criterion. A leaf will be subdivided into four equal quadrants when the number of (internal) vertices that it contains goes beyond a given threshold (figure 2). The subdivision after the number of vertices guarantees that this number will never go beyond the threshold. Since the number of edges that end at these vertices is (previously) unknown, the edges must be stored on a variable length list.

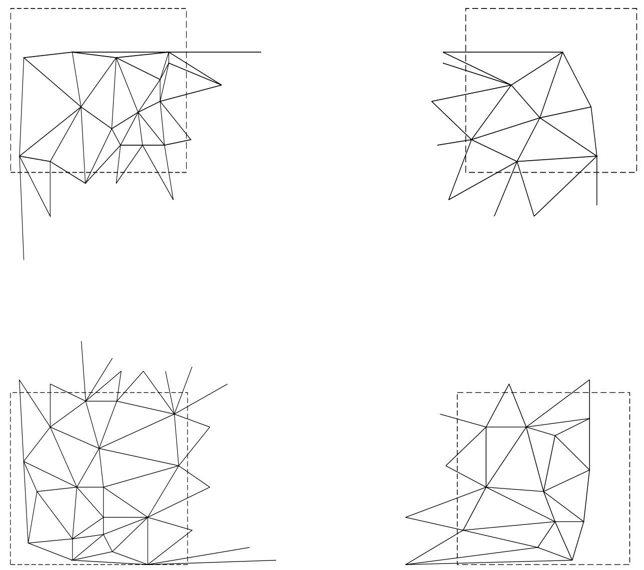

Figure 2: Partition of the triangulation in figure 1 into four leaves

We call *trans-block edge* the edge which vertices fall into two different leaves. That edge appears as duplicate on both leaves. We will then say that a trans-block edge has a homologous edge on another leaf and its attributes will be distributed over its two homologous representations.
We call *foreign vertex* the vertex of a trans-block edge that belongs to the *other* leaf. The number of foreign vertices is, again, not bounded. For efficiency reasons, a leaf of the quadtree will store the list of foreign vertices on a dynamic allocated array. Since the algorithm stores, in each leaf, both vertices of an edge, it can compute, from the coordinates of the foreign vertex, in which leaf of the quadtree lies the homologous edge. The data structure for trans-block edges is slightly different from the structure for ordinary edges. The pointer to the next edge turning around the foreign vertex is substituted by the index of its homologous edge inside the list of edges of the other leaf of the quadtree. The modified DCEL structure is the following:

V1, V2 first and second vertices

E1, E2 next edge turning around V1 (resp. V2) counterclockwise
or index of the homologous edge if X1 (resp. X2) is true

F1, F2 face on the left (resp. right)

req is the edge required?

X1, X2 is V1 (resp. V2) a foreign vertex?

The presence of pointers to the homologous edge and the possibility of computation of the leaf to which that edge belongs (through the coordinates of the foreign vertex) allow the computation of the address of the homologous edge in time $O(p)$, where p is the depth of the quadtree. The maximum value of depth p depends on the minimum distance d between two different sets of c points, where c is the capacity of the leaves. Let a quadtree with n points that covers a square region of side l, and d the minimum distance between two points. The depth p will be inside this interval, where the upper bound is only indicative.

$$\left\lceil \log_4 \frac{n}{c} \right\rceil \leq p \leq \log_4 \left[\frac{2}{\sqrt{3}} \frac{l^2}{cd^2} \right] \tag{1}$$

The minimum distance d is a preestablished minimum value under which we consider two points to be equal and the same. If this minimum distance becomes very small compared to l, then it would be wise to change the leaf criterion to limit the maximum depth p. Leaves will stop subdividing when they reach a preset maximum depth. In our application, capacity c is big, the vertices have an approximately uniform distribution and the average distance between neighboring points is much greater than d. Then, that limit will be reached only with a negligible probability.

We have done a numerical experiment to observe the maximum depth and the average occupation density of the leaves as functions of the number of vertices, and this for several leaf capacities. We have taken a set of real data obtained through photogrammetric restitution and we have inserted them in the quadtree in the same order as they were captured. Figure 3 shows the average occupation of leaves, given as the ratio number of points / capacity. Figure 4 shows the maximum depth reached by the tree. Results for occupation are similar to those shown in table 1. There is no qualitative change in behavior following a change in the capacity of the leaf. There are neither aging nor phasing effects due to non-random insertion. Real data are ordered at capture time and take more time to stabilize than points generated at random.

We can classify this tree as a *bucket point region quadtree* [?] with additional information: a list of edges and a list of foreign vertices.

4 Triangulation update

It is possible to update (insert and delete) incrementally with points and required edges. The algorithms are a combination with some modifications of

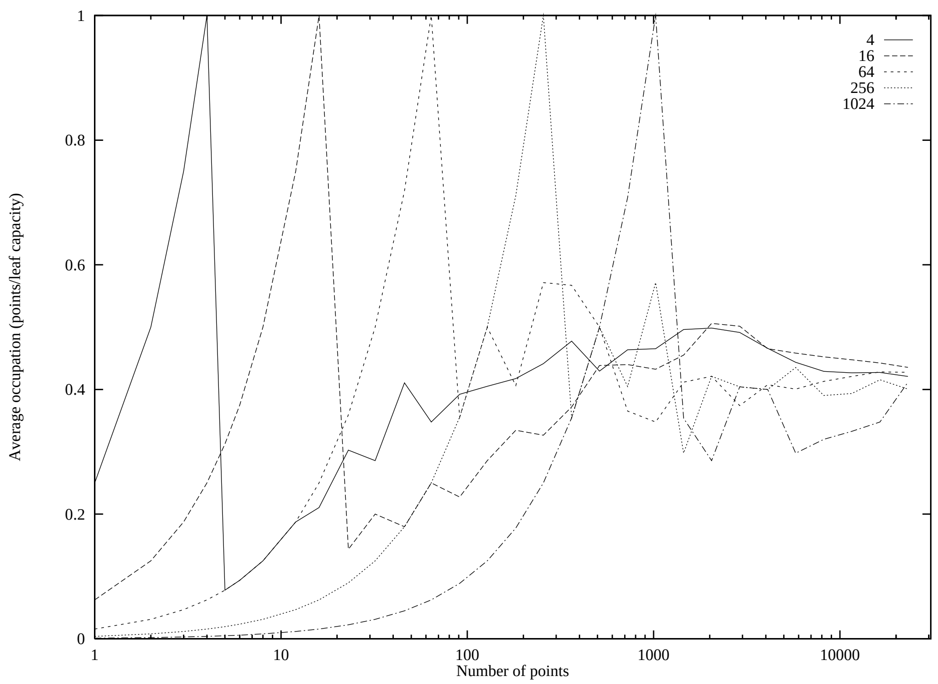

Figure 3: Average occupation versus number of points in the *quadtree* for different leaf capacities: 4, 16, 64, 256 and 1024 points

the algorithms that operate on the *bucket point region quadtree* [?] and the algorithms for the incremental update of a CDT when it is stored on a DCEL [?].

The quadtree structure is altered only by the insertion and deletion of vertices.

To insert a vertex, the procedure starts with a search for the leaf that contains that vertex; the search is performed navigating through the pointers to nodes of the tree from the root. If there is room in that leaf, the vertex is added to the corresponding list. If there is no room, the leaf is subdivided into four leaves (with a relocation of the internal vertices, edges and foreign vertices that it contains). Then the new vertex is actually inserted.

To erase a vertex, the procedure starts (again) with a search for the leaf that contains that vertex. Once the leaf has been found, the search proceeds following the pointers of the DCEL structure as per the Lawson's search algorithm until the "target" vertex is found. Then it is eliminated.

Often, there comes a new insertion after a deletion. That is why it is not wise to group together 4 sister leaves (an operation called **collapsing**), even if the summatory of the four numbers of vertices crosses the threshold (for one node) downwards. Collapse will be actually performed only if all four "sister" leaves are empty.

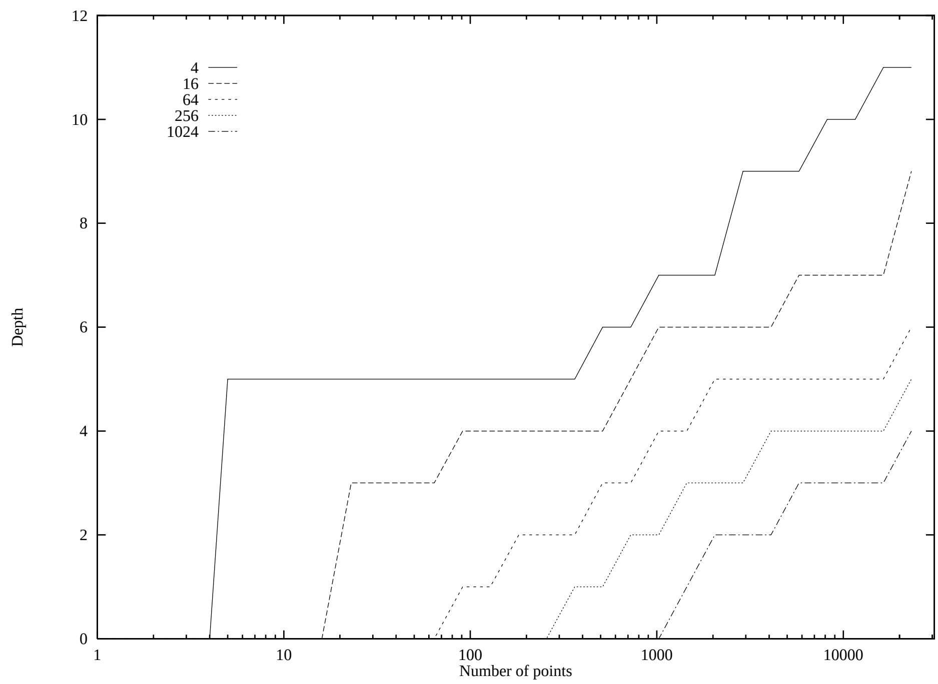

Figure 4: Maximum quadtree depth versus number of points for different leaf capacities: 4, 16, 64, 256 and 1024 points

Neither the insertion nor the deletion of required edges alter the structure of the tree. They only affect the DCEL structure and the implied modification of the lists of edges and of foreign vertices.

In each triangulation update operation, the structure goes through several intermediate states in which some edges and triangles have been erased and there are some closed polygonal regions waiting for triangulation. These intermediate states can be represented both by the DCEL structure and by the structure presented in this paper.

The algorithms for incremental triangulation update operate using to advantage the properties of the DCEL edges (table 1). The structure presented here modifies attributes **E1** or **E2** in the cases where vertices **V1** or **V2** are foreign, and instead of these the index of the homologous edge is stored. It is necessary to modify the algorithms of de Floriani & Puppo [?] that operate on a DCEL to take into account this change. Attributes **E1** and **E2** are pointers used to store the next edge turning from the actual edge **E** counterclockwise around vertices **V1** or **V2** respectively. Let $Rot_1(E)$ and $Rot_2(E)$ be these operations. For the DCEL these operations are straightforward: $Rot_1(E) := E1(E)$ and $Rot_2(E) := E2(E)$. With the proposed modification $Rot_1(E)$ will be:

Both representations of a trans-block edge have the same orientation. The algorithm for $Rot_2(E)$ is similar, just changing the ordinal 1 for 2.

```
if V1 = foreign then
  Return(E1(E1(E)))
else
  Return(E1(E))
endif
```

Figure 5: Operation $Rot_1(E)$

5 Discussion.

Trees are very appropriate structures to store great amounts of data. The Delaunay tree needs a lot of space and does not keep the data ordered, not even partially. If the tree has to be stored in secondary storage, data retrieval for a Delaunay tree following the chain of pointers will imply a great number of page faults. On the other hand, random insertion might imply a very inefficient unbalanced tree, even in the case of the vertices being uniformly distributed. This is due to the fact that partition of the space depends both on the data points and on the order in which they are inserted. If, on the other hand, the procedure keeps a log of the history of the triangulation, then the storage needed is very large. The *PMR-quadtree* does not maintain the DCEL and presents a high redundancy ratio because it replicates each edge in each and every one of the leaves that it (the edge) crosses.

The proposed data structure yields a good partial ordering of the data points. If the vertices list capacity is big enough, the resulting tree will not be very deep and, therefore, full storage in main memory is possible. In this case, retrieval of the data falling on a small rectangular patch can be done with a single access to disk (or secondary mass storage, for that matter). Under these conditions, the quadtree behaves as a directory that holds the address (on the disk) where the data can be found. On the reverse side, though, when the capacity c of the list of vertices increases, the time complexity for the incremental insertion of points into the triangulation also goes up. This is due to the fact that the redistribution of data associated to the subdivision of a leaf is a sequential process with a cost of $O(c)$.

Edges are stored *only* in the leaf or leaves where each of the vertices lie, even when these leaves are not adjacent. In the *PMR-quadtree* the edge must be repeated in *all* the leaves that are crossed by the edge, because it is the tree that holds the topological relationships among edges.

In the data structure presented here, the relationships are maintained by the DCEL. The quadtree yields an adaptive partition of the DCEL in blocks that are approximately rectangular. The quadtree gives a quick approach to the block in which the triangle being searched lies; the quadtree shows a partial ordering of the data points that allows the pagination and segmentation of the triangulation between central memory and secondary storage. This partition is adaptive in the sense that the number of blocks and their size depend on the density of vertices in the corresponding area.

6 Future work

These algorithms are being programmed in Eiffel [?]. The data base holding the data will be O_2 [?].

We have also designed algorithms for simultaneous substitution of blocks of data inside a polygon. This has been done to perform massive update of data and to preserve compatibility with other commercial Terrain Modelling packages.

As stated in the introduction, this structure lies at the foundation of a system for Topographical Surface Modelling called BasAlt [?]. The partition of the surface into triangles is a generic partition. We can associate any surface model to each patch in the tessellation: flat triangle, NURB, bilinear surface, etc. At the beginning, the default surface model will be the planar triangle. The use of Object Oriented technology (Eiffel and O_2) allows the addition of functionalities and progressive refinement of the application.

References

[1] L. De Floriani, B. Falcidieno, and C. Pienovi. A Delaunay-based representation of surfaces defined over arbitrarily-shaped domains. *CVGIP: Graphical Models and Image Processing*, 32:127–140, 1985.

[2] J.C. Davis. *Statistics and data analysis in geology.* John Wiley & Sons, New York, 1973.

[3] F.P. Preparata and M.I. Shamos. *Computational Geometry: An Introduction.* Springer, New York, 1985.

[4] C.L. Lawson. Software for C^1 surface interpolation. In J R Rice, editor, *Mathematical Software III.* Academic Press, New York, 1977.

[5] L.J. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Transactions on Graphics*, 4(2):74–123, 1985.

[6] J. D. Boissonat and M. Teillaud. A hierarchical representation of objects: The Delaunay tree. In *Second ACM Symposium on Computational Geometry in Yorktown Heights*, 1986.

[7] O. Devillers, S. Meiser, and M. Teillaud. Fully dynamic Delaunay triangulation in logarithmic expected time per operation. Technical Report 1349, INRIA, Rocquencourt, France, 1990.

[8] R.C. Nelson and H. Samet. A consistent hierarchical representation for vector data. In *Proc. SIGGRAPH'86 Conference*, pages 197–206, 1986.

[9] H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison Wesley, Reading, MA, USA, 1990.

[10] L. De Floriani and E. Puppo. An on-line algorithm for constrained Delaunay triangulation. *CVGIP: Graphical Models and Image Processing*, 54:290–300, 1992.

[11] B. Meyer. *Object-Oriented Software Construction.* Prentice Hall, New York, 1988.

[12] O_2 Technology, Versailles. France. *The O_2 user manual*, 1995.

[13] A. Ruiz. Basalt: Base de dades altimètriques. Memòria del projecte. Technical Report 000111010, Institut Cartogràfic de Catalunya, Barcelona, 1995.