A COUNTRYWIDE TIN ELEVATIONS DATABASE

Antonio RUIZ
Institut Cartogràfic de Catalunya
Parc de Montjuïc. 08038 Barcelona, Spain
toni@icc.es

KEY WORDS: Delaunay triangulation, Quadtree, DTM, TIN, Database

ABSTRACT

The regular grid elevations database that the Institut Cartogràfic de Catalunya has since 1987 is insufficient for certain applications. For this reason, we started the development of a new database. It is our aim to build a one piece TIN model with precision enough to generate cartographic quality contours at 1:5000 scale and smaller for the whole country (32000 sq. km). We plan to load approximately 300 million points. The model must be dynamic because it has to support insertion and deletion, the surface model should be refinable, algorithms have to be robust and data integrity must be preserved.
The chosen triangulation is constrained Delaunay triangulation (CDT) and we employ the incremental construction algorithm. Its performance is good when the input data are well distributed and is fully dynamic.
The main difficulty is how to deal with a triangulation in external memory. Common triangulation programs hold all the information in core memory and cannot triangulate very large data sets (some million vertices) because the system degrades very fast due to the page faults. We use a bucket point region quadtree to help point location and to reduce page faults by bucketting.
The surface model is a polyhedral TIN terrain by default but it can be extended in different ways. The use of object-oriented technology means that these extensions can be programmed easily and gives the design great flexibility.

1 INTRODUCTION

Since 1987 the Institut Cartogràfic de Catalunya (ICC) has a Digital Terrain Model (DTM) database with a 15 m regular step for our country (32000 sq km). The main use of this DTM is the rectification of images and the shadowing and hypsometric coloring of maps.

At the beginning the database was loaded with data compiled with a Gestalt Photomapper IV and analytic stereoplotters. At the same time the 1:5000 topographic (MTC 1:5000 v1) maps were produced with contours computed from a TIN model. The TIN was build from profiles and breaklines compiled with analogic and analytic stereoplotters and the DTM database was completed with regular grid data interpolated from the TIN models.

The TIN model of Catalonia from the MTC 1:5000 v1 consists of some 75 million vertices and it is not available on-line. Now we are producing the second version of the 1:5000 topographic map (MTC 1:5000 v2) and we expect from 5 to 6 times more points. In this new version almost all the points and lines from the topographic map that lay on the ground enter to form the TIN model. The contour lines for the MTC 1:5000 v2 are computed from this TIN model.

2 REQUIREMENTS TO A NEW DTM DATABASE

The on-line available grid model is not enough for contouring at large scales or for the orthorectification of large bridges in roadways, buildings, etc. We would rather have the TIN model available on-line but with standard programs is not possible to build a huge TIN in one piece. Commercial TIN programs are limited by the core memory of the computer because they require to have all the information in memory and rely on the standard paging services of the operating system when more memory than available by hardware is needed. For that reason, standard programs can hardly triangulate some million points and they cannot deal with datasets of the size of ours. The most difficult aim of this project is to build such a large TIN model in one piece. The triangulation we have chosen is the constrained Delaunay triangulation (CDT) but this is only a subdivision of the surface domain. We plan to extend the commonly used polyhedric surface model to represent both the digital elevations model (DEM) and the digital surface model (DSM). The DEM must be accurate enough to generate cartographic quality contours at 1:5000 scale for the whole country. The model required for the rectification of images is the DSM. It is the visual surface that follows the top of the buildings and the trees seen from above. Some work has been done to extend the surface model to represent buildings, bridges and to give double z values in water covered regions (bathimetric depth or heigh of the water surface). The design is flexible enough to improve and extend the surface model as needed. The model must be dynamic because it has to support insertion and deletion at any time, algorithms have to be robust and data integrity must be preserved.

3 DESIGN

The TIN model is more complex than the grid model. We have to store on one side the geometric properties and on the other side the combinatorial properties: We need a list of vertices with 3 coordinates each and a list of pointers that describe explicitly how the vertices join to form the edges and how the edges join to form the triangles. Many structures have been used to store triangulations and between them we have chosen the Double Connnected Edges List (DCEL) introduced by Muller and Preparata in 1978 ((Preparata and Shamos, 1985)).

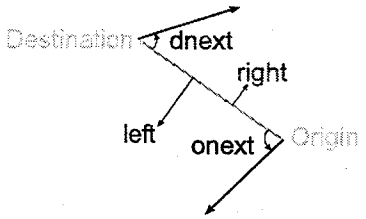

Figure 1: DCEL data structure

Edges in DCEL are oriented, but the orientation is arbitrary. Each edge has six pointers: one to the origin vertex and another to the destination, one pointer to the next edge that is found rotating counterclockwise around the origin and another to the next around the destination. Two more pointers are usually reserved to the left and to the right triangles. The space required to store the triangulation is $|P| = 6|E| \simeq 18|V|$ where we have taken into account the Euler relation and $|P|$ is the number of pointers, $|E|$ the number of edges and $|V|$ the number of vertices.

We can find in constant time which vertices belong to a triangle and which are its neighbouring triangles. With the DCEL it is possible to represent any plannar graph and in particular we can represent intermediate states of the triangulation that appear while updating it. We can also store edge attributes without redundancy.

By default our surface will be the polyhedral surface formed by the flat triangles and we will not store the triangles explicitly. In the general case the triangles will not hold any information and they can be recovered from their boundary edges in constant time. For that reason our pointers to triangles will usually be null pointers but we reserve these pointers for the extension of the polyhedrical surface model as will be shown later in this report.

3.1 Triangulation in external memory

There are many algorithms to construct the contrained Delaunay triangulation. The sweep line algorithm for the CDT (Seidel, 1988) is a generalization of Fortune's algorithm for DT (Fortune, 1987). There is also a divide and conquer algorithm for CDT (Paul Chew, 1989) that generalizes the algorithm by Lee and Schacter (Lee and Schachter, 1980). Those algorithms are optimal within the real RAM model of computation (Preparata and Shamos, 1985) but they are not dynamic. They require to sort all the points prior to triangulate them because they work on ordered sets of points. We prefer for our application the incremental construction algorithm because it is fully dynamic, i.e. we can insert and delete vertices and required edges at any time. This method is $O(n^2)$ in the worst case but the average cost is low with the data distributions found in practice. Melhorn et al. showed in 1990 that randomized insertion of points is $\Theta(n \log n)$ on average with the advantage of being fully dynamic.

The real RAM computational model is not appropriate for our application. One better approximation for our case is the two level memory model (Goodrich et al., 1993). In this model of computation two kinds of memory are considered: one very fast that represents the computer RAM memory and another very slow that represents the disk storage. Operations in the fast memory can be considered as instantaneous ($\sim$ 10 ns) in comparison to the external memory operations ($\sim$ 10 ms) that are a million times slower. We are aware of the optimal external memory algorithm for the computation of the convex hull in 3D and this problem is equivalent to Delaunay triangulation but this algorithm is not dynamic and is very complex. We have not gone further in this direction. Instead we have followed a more heuristic approach.

The TIN data is irregular. In case our data set is very large it will not fit in core memory and when the program follows pointers from one edge to another many page faults can take place. It is not affordable that each pointer access gives raise to one I/O operation. To increase the performance of our algorithms we must minimize the number of I/O operations and reduce page faults. The technique available to do that is bucketing. Points and edges that are close in terrain should be stored also close into disk because most of the operations that are performed on the TIN are local. Usually we are going to recover or to update the information of a region corresponding to a photogrammetric model or to a sheet map. We are not interested in the insertion of points distributed at random anywhere on the domain. If the information for this region spans on a few pages on disk, a small number of I/O operations can be enough and the operation will be faster.

3.2 Point location and bucketting

The optimal point location cost is $O(\log(n))$ in time. If we want to insert n points, for each of them we have to perform a point location and so the cost of triangulation in the real RAM model is $O(n \log(n))$, and this is optimal.

Point location by Lawson navigation or by ray following is $O(n)$ (Lawson, 1977), but with data points homogeneously distributed as they are found in practice the average cost is $\Theta(n^{\frac{1}{2}})$ (Guibas and Stolfi, 1985). One of the reasons for incremental triangulation beeing suboptimal is because point location is suboptimal. The updating is also suboptimal because the number of edges to be swaped due to a point insertion depends on the data distribution and it can be the whole set. On average with the data sets found in practice the number of triangles that need to be updated after the insertion of a point is low. We need to improve point location and one solution to do this is to build a tree and to perform a search on it. If the tree is balanced, point location can be performed in $O(\log(n))$ time.

We have chosen a bucket point-region quadtree (bucket PR-quadtree or BPRQT) to store the vertices of the triangulation. The triangulation is a data oriented partition of the plane while the region quadtree is a regular partition of the plane in which the data only determines the depth of the tree. The quadtree (QT) has the twofold mission of improving point location and of keeping partially sorted the information stored into disk.

The idea of storing a triangulation in a QT is not new. Chen et al. (Chen, 1990) also used a quadtree to store a triangulation but they stored at most one triangle per leaf.

A BPRQT is a quadtree (QT) in which a leaf subdivides into four by halving in each direction (Samet, 1990b). A leaf can contain points up to a maximum or threshold. If we want to insert one more point then the leaf subdivides into four and the contained points are redistributed between the new sibling leaves. Figure 2 shows the triangulation of Davis sample data set (Davis, 1973) in a BPRQT for a threshold of 4 points per leaf. Practical thesholds are larger. We apply a threshold of 400 points/leave.

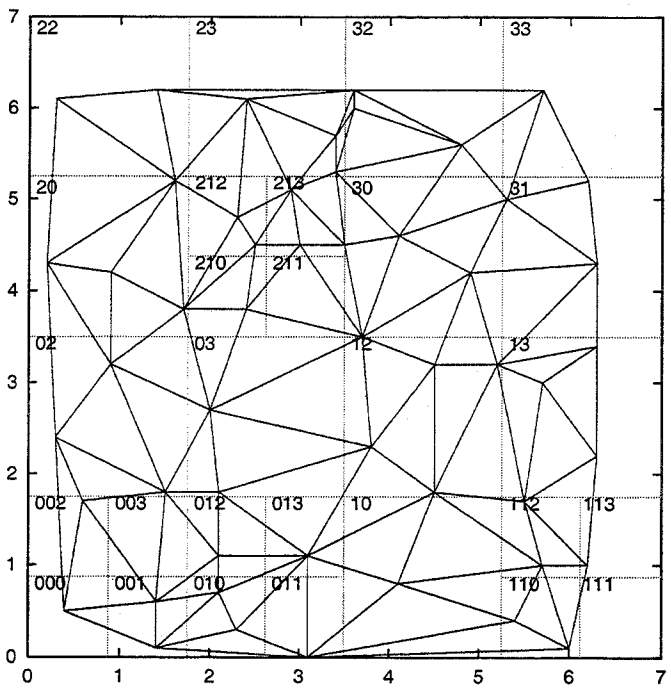

Figure 2: Triangulation of a sample set of points in a bucket PR-quadtree

We have extended the BPRQT in the following way: each leaf contains also a pointer to one interior edge or to an edge close to the center of the leaf. We name this edge the "central edge". Point location is performed in two steps. First we search in the QT the containing leaf. Then the search proceeds as a Lawson navigation following the DCEL pointers starting on the central edge of the leaf.

The algorithms for the management of a BPRQT of points are described in Samet ((Samet, 1990a), (Samet, 1990b)). A maximum depth must be defined because when points are very close there is no warranty to separate them by halving space a few times. Another problem with the PR-quadtree is the low average occupation of the leaves. The leaves are void at the beginning, when a leaf fills it subdivides and the contained vertices are redistributed between 4 siblings. We did a numerical experiment with real data to find the average occupation of leaves. After a few oscillations the average

occupation was close to 0.4 times the leaf capacity (figure 3), independently of the chosen threshold value(Ruiz et al., 1995).

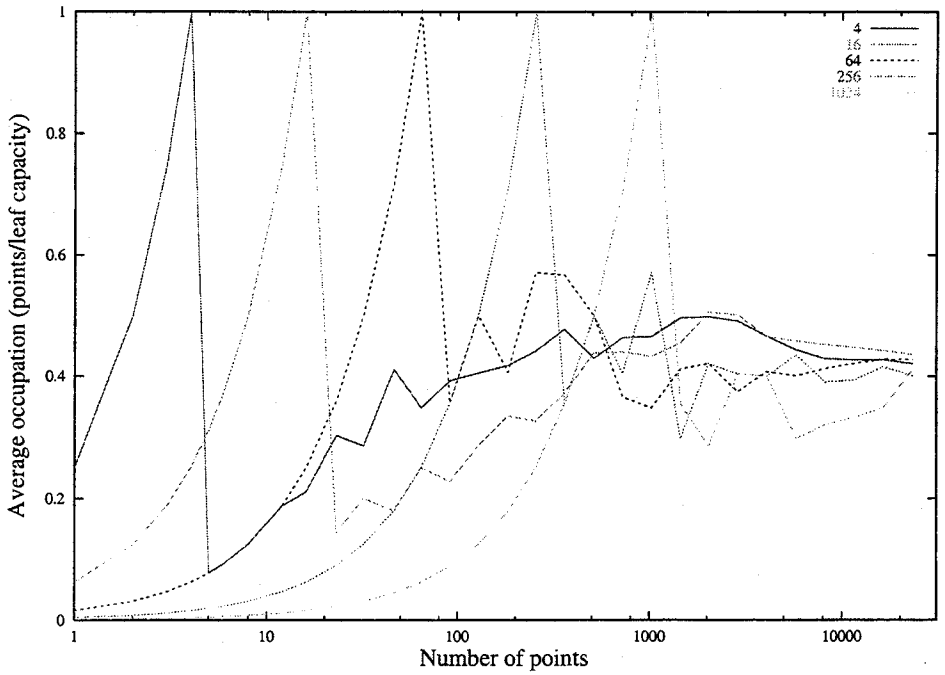

Figure 3: Average occupation versus number of points in the quadtree for different leaf capacities: 4, 16, 64, 256 and 1024 points

We need variable length lists to store the vertices in a leaf. Nelson and Samet (Nelson and Samet, 1986) found that the average occupation would be a half of the threshold value. They described the effects of aging and phasing of the leaves. Aging means that the older leaves use to hold a larger amount of points than younger ones. Phasing appears when points belonging to a uniform data distribution are inserted.

3.3 Algorithms for the CDT computation

Bernal presented algorithms for the incremental construction of the CDT (Bernal, 1988).

The algorithms we have programmed are those published by de Floriani and Puppo for a triangulation represented in DCEL structure ((de Floriani and Puppo, 1992)). We only have changed the criterion to retriangulate the polygons Q that appear when a required edge l is inserted in the triangulation and the crossing edges are removed.

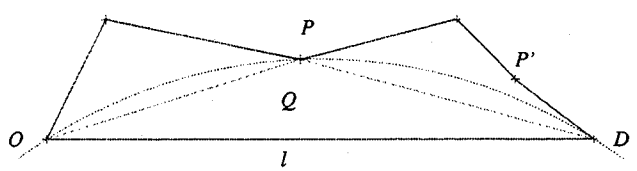

Figure 4: Retriangulation of Q after insertion of required edge l

De Floriani and Puppo chose the vertex closer to l to build the first triangle joining l origin O and destination D. We have changed this criteria and choose the vertex P whose subtended angle $\widehat{OPD}$ is larger. By the definition of CDT the vertex we choose is more appropriate than the vertex closer to l. Our vertex can be further from l but no other vertex P' will lay in the circumcircle of (O, P, D) to this side of the required edge l or otherwise the angle $\widehat{OPD}$ cannot be maximal. The search for the maximum angle can be done without computing trigonometric functions. Instead of searching for $\max(\alpha)$ we look for $\max(f(\alpha))$ which is equivalent in the interval $\alpha \in [0, \pi)$

$$f(\alpha) = \begin{cases} \sin^2 \alpha & \text{if } \cos\alpha \geq 0 \\ 2 - \sin^2 \alpha & \text{otherwise} \end{cases} \quad (1)$$

and $sin\alpha$ is found by vector product computations.

3.4 Robustness

During the computation of the Delaunay triangulation there appear two geometric predicates that we have to evaluate accurately. They are related to two degenerate cases of the points distribution. The first predicate tests the orientation of a point respecting to an oriented edge. One point can be to the left or to the right of an oriented edge or, in the degenerate case, the three points can be inline.

The second predicate is named the incircle test. Given a triangle of the triangulation there is a circle that passes through its three vertices. A point of the plane can be in the interior of the curcumcircle or out of it. In the degenerate case the four points can be cocircular. Both tests can be put in the form of a determinant and the result of the test depends on the sign of this determinant.

$$Orient2d(a,b,c) = \begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{vmatrix} \tag{2}$$

$$Incircle(a,b,c,d) = \begin{vmatrix} a_x & a_y & a_x^2+a_y^2 & 1 \\ b_x & b_y & b_x^2+b_y^2 & 1 \\ c_x & c_y & c_x^2+c_y^2 & 1 \\ d_x & d_y & d_x^2+d_y^2 & 1 \end{vmatrix} \tag{3}$$

We do not need to evaluate these determinants exactly. We only need to be confident on its sign. Only in case the value is close to zero the result of the computation is unreliable. We need to evaluate the error of the computation and increase the number of bits of the computation arithmetic when the absolute value of the determinant is smaller that the computational error. We follow the adaptative precision algorithms by Shewchuck (Shewchuk, 1996a, Shewchuk, 1996b).

3.5 Refinable model and programming

We have programmed in C++. In object oriented (OO) programming a class consists of the data structures and its associated methods. The main classes we have defined correspond to the geometric objects we manipulate: point, vertex, DCEL edge, TIN, PR-quadtree node, face... We distinguish always an active edge and one active side that is to the left or to the right of this active edge. We have methods to change these active objects. There are methods to move the active edge to the next or to the previous edge around the origin or destination vertex of the current active. There are also iterator classes that allow to traverse the triangulation in different ways. One iterator class allows traversing all the edges and vertices interior to a polygon. Another can follow the edges that cross an oriented segment. There is a cursor class to represent one active edge with its active side. We have not a triangle class. A triangle can be identified by the left or the right of one of its boundary edges and can be represented with a cursor. Programming has been done following Meyer's programming by contract paradigm (Meyer, 1997). Most methods have preconditions and some of them have postconditions that have been really helpfull to detect programming bugs.

With object-oriented programming the surface model can be extended and refined easily. The default surface model is the polyhedrical surface formed by flat triangles. Usually the triangles are not stored and the edges hold void pointers to triangular faces. In case a different surface model is required we attach this surface model to the triangular faces. The edge pointers to face are then used and the attached face object provides an alternative height function different from the height that corresponds to the flat triangle. Some faces we have defined are the exterior face that returns unknown height for the region out of the defined domain, the sea surface that returns zero or the batimetric depth depending on a state bit, and the building model that returns a value on the visual DSM or the DEM depending on another state bit.

We can also store efficiently hybrid models consisting of regular grids plus some irregular data (points and lines). This is the kind of data we get with authomatic stereo matching methods. The boundary of the regular grid is triangulated and also the irregular data plus the vertices of the cells in the grid containing the irregular data. In the cells with irregular data the surface model is the polyhedral TIN surface but in the regular grid cells the surface model is the bilinear surface. The triangular faces formed by triangulating the boundaries of the regular grid region hold pointers to the grid surface model. In this way we do not need to triangulate all the grid nodes. This is another example of the flexibility of our design.

The methods corresponding to triangulation algorithms belong to the TIN class and the methods corresponding to the QT belong to the bucket PR-quadtree class. TINQT class is a refinement of the TIN class with the specific methods for a TIN stored in a QT. It redefines point location as a hierarchical search on the PR-quadtree plus a standard Lawson search on the TIN. The bucket PR-quadtree class is also the container class for the vertices. After updating the TIN new vertices are

inserted in the bucket PR-quadtree. Duplicated points and crossing edges are detected during insertion. Vertices closer than a minimum to an existing vertex are not allowed. Vertex and edges have attributes that identify them as topographic objects. For example, we distinguish required edges as belonging to a road, a railway, a river, etc.

The data is stored in the commercial database O_2. All the programming is done as if all the data was in memory. O_2 takes care of recovering data from external storage as needed, maps the database pointers and the C++ program pointers and updates the information on disk at commit time. O_2 provides us with the standard properties of a transactional database: atomicity, consistency, isolation and durability.

4 CURRENT STATE OF DEVELOPMENT AND FUTURE WORK

In order to reduce the insertion overhead the edges are not stored in the QT at this stage of development. During the insertion of the data corresponding to a region, edges are destroyed almost at the same speed as they are created and it is a waste of time to store them in the QT. This has the bad consequence that performance decreases when a lot of data has been inserted on the database because there is no bucketing on the edges. We rather would store the edges in the QT when we close the transaction and no more updatings are expected for the region. Our insertion transactions correspond usually to the insertion of all the vertices of a sheet map.

The development of the software started on an old Unix machine with 64 Mb of RAM. Up to 2.8 million vertices where triangulated and then the performance decreased significantly. Now we are completing the migration of the application to a PC with Windows NT operating system.

One possibility to improve the edges storage is to made use of the clustering facilities that O_2 database provides. We can create a spatial index on the edges to instruct the database manager to keep closer in disk those objects with closer index values. A valid spatial index is the Morton code of the leaf containing the edge origin.

A big source of trouble in computer programming with C or C++ is freeing program memory and garbage collection. We never free program memory directly because we do not know if an object will be needed later again or not. Instead, when we are short of memory we close database and reopen it. This operation frees all the program memory. When one object is requested again, it is recovered from O_2 buffers and recreated in program memory by the database services. All the data that comes from a photogrammetric model or one 1:5000 sheet map can be inserted into the database without closing for freeing memory but to insert different sheet maps, we employ separate runs of the insertion program.

Another weakness of the current state of development is the TIN traversal method. We use process bits to indicate wheather a vertex or an edge have been traversed and a stack to hold the set of edges to be traversed in the iterator classes. This has the bad consequence that any recovery of data from the database is a read and write transaction because the state bits change their values from unprocessed to processed and after the traversal has been completed another traversal is required to change the state bit values again to the unprocessed state. For this reason some transactions are more complex than they should be and multiuser access for reading of the one region is not available. To solve this problem we have to rewrite iterator methods to perform the traversal without process bits and without stacks (Gold and Cormack, 1986, de Berg et al., 1996).

The internal nodes of the quadtree can store generalizations of the surface stored in the leaves. A simple grid model of fixed step interpolated from the TIN can be stored in the internal nodes inmediatly over the leaves that contain the TIN. The internal nodes of lower resolution can hold low pass filtered models of decreasing resolution but this generalized models have to be developed.

5 CONCLUDING REMARKS

We have shown the main difficulties to build a large TIN model and we have presented our proposed solution. We also have shown some extensions to the TIN polyhedrical model that are interesting for terrain modelling.

We have tested a preliminary version of the software with nearly 2.8 million vertices in an old Unix machine. Now we have moved the software to another platform and operating system but more performance improvements are still required until we can say that the system is operational.

REFERENCES

Bernal, J., 1988. On constructing Delaunay triangulations for a set of constrained line segments. Tech Note 1252, National Institute of Standards and Technology. Department of Commerce.

Chen, Z. T., 1990. A quadtree guides fast spatial searches in triangular irregular network (TIN). Proc Int Sym Spatial Data Handling. Zurich pp. 209–215.

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

de Berg, M., van Kreveld, M., van Oostrum, R. and Overmars, M., 1996. Simple traversal of a subdivision without extra storage. Tech. Rep. UU-CS-96-17, Dep. Computer Science. Utrech Univ., Utrech.

de Floriani, L. and Puppo, E., 1992. An on-line algorithm for constrained Delaunay triangulation. CVGIP: Graphical Models and Image Processing 54(3), pp. 290–300.

Fortune, S. J., 1987. A sweepline algorithm for Voronoi diagrams. Algorithmica 2, pp. 153–174.

Gold, C. and Cormack, S., 1986. Spatially ordered networks and topographic reconstructions. In: Proc 2nd Int Sym Spatial Data Handling, pp. 74–85.

Goodrich, M. T., Tsay, J.-J., Vengroff, D. E. and Vitter, J. S., 1993. External-memory computational geometry. In: Proc. IEEE Symp. on Foundations of Comp. Sci., pp. 714–723.

Guibas, L. and Stolfi, J., 1985. Primitives for the manipulation of general subdivisions and the comparison of Voronoi diagrams. ACM Transactions on Graphics 4(2), pp. 74–123.

Lawson, C. L., 1977. Software for C^1 surface interpolation. In: J. Rice (ed.), Mathematical Software III, Vol. 3, Academic Press, New York.

Lee, D. T. and Schachter, B. J., 1980. Two algorithms for constructing a Delaunay triangulation. Int. J. Comp. Inf. Sci. 9(3), pp. 219–242.

Meyer, B., 1997. Object-oriented software construction. 2nd edn, Prentice-Hall, New Jersey.

Nelson, R. C. and Samet, H., 1986. A consistent hierarchical representation for vector data. Computer Graphics 20(4), pp. 197–206.

Paul Chew, L., 1989. Constrained Delaunay triangulation. Algorithmica 4, pp. 97–108.

Preparata, F. and Shamos, M., 1985. Computational Geometry. Springer-Verlag, New York.

Ruiz, A., Colomina, I. and Pérez, L., 1995. Acceso y actualización de una red irregular de triángulos desde un quadtree. In: VI Encuentros de Geometría Computacional, Barcelona.

Samet, H., 1990a. Applications of Spatial Data Structures. Addison-Wesley, Reading, MA.

Samet, H., 1990b. The Design and Analysis of Spatial Data Structures. Addison-Wesley, Reading, MA.

Seidel, R., 1988. Constrained Delaunay triangulations and Voronoi diagrams with obstacles. Technical Report 260 IIG, TU Graz, Austria.

Shewchuk, J. R., 1996a. Robust adaptive floating-point geometric predicates. In: Proc 12th Ann Symp on Computational Geometry, ACM.

Shewchuk, J. R., 1996b. Triangle: engineering a 2D quality mesh generator and Delaunay triangulator. In: First Workshop on applied computational geometry, ACM, Pennsylvania, pp. 124–133.