Storage and analysis of massive TINs in a DBMS

Hugo Ledoux

April 8, 2010

Abstract

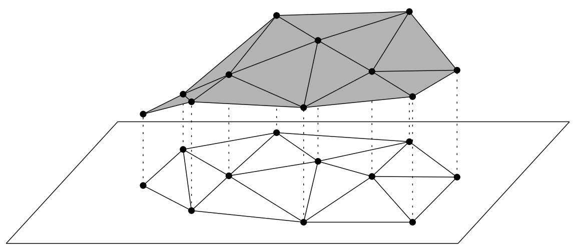
Advances in technologies to collect elevation data are far superior to the advances to store and process the resulting point clouds. This paper investigates the use of a DBMS to store and manage not only the points coming from massive point-based datasets, but also a TIN of these points. TINs are used as a support structure to implement processing and manipulation operators. I discuss in the paper why storing efficiently a TIN in a DBMS is a complex task, and I propose a new solution. It does not store explicitly triangles, as only the *star* of each point is stored. The details of the data structure are discussed and compared with other solutions currently available.

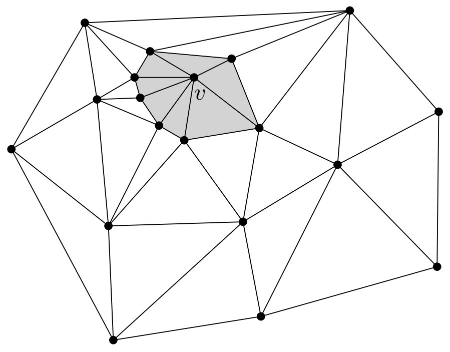
1 Introduction

New technologies such as airborne altimetric LiDAR (**Li**ght **D**etection and **R**anging) or multi-beam echosounders permit us to collect millions—and even billions—of elevation points (samples) for a given area, and that very quickly and with great accuracy. One example of the use of that technology in the Netherlands is the efforts by Rijkswaterstaat to build an accurate digital terrain model (DTM) of the whole country containing as much as 10 samples per m^2 (the AHN^2 dataset, `www.ahn.nl`). Such datasets have attracted a lot of attention because of all their possible applications, e.g. flood modelling, monitoring of dikes, forest mapping, generation of 3D city models, etc.

The main problems with massive point cloud datasets is that while they provide us with unprecedented precision, computers have great many problems dealing with very large datasets that exceed the capacity of their main memory. The result is that they cannot efficiently display all the information, let alone *process* them. Examples of point cloud processes useful for many applications are: derivation of slope/aspect, conversion to a grid format, control of double points, calculations of area/volumes, viewshed analysis, creation of simplified DTM, extraction of bassins, etc. Advances in technologies to collect data are basically superior to our ability to process data.

LiDAR or multi-beam datasets are formed by scattered points in 3D space, which are—in several cases—the samples of a surface that can be projected on the horizontal plane (a so called "2.5D surface"), see Figure 1a. This is of course not always the case but for several situations 2.5D is enough and it is therefore worth developing specific tools to handle this case since they can be highly optimised. While simply storing millions of unconnected points in a DBMS is no problem, the processing of LiDAR datasets needs more: we must be able to reconstruct the surface represented by


(a)



(b)

Figure 1: **(a)** A 2D surface obtained from a set of points in the plane having a height as an attribute. **(b)** A triangulation of a set of points in the plane. The vertex v has 7 neighbours.

the points, and we must also be able to access this surface to manipulate it (e.g. adding/removing new samples), and also to derive values from it. The surface gives us the spatial relationships between unconnected points in 3D space, which is required if processing on the surface is wanted. The best way to reconstruct such a surface is arguably with a triangulated irregular network (TIN) respecting the Delaunay criterion (Wang et al., 2001). As shown in Figure 1b, a TIN subdivides the space covered by the points into non-overlapping triangles, and these can be used to reconstruct the surface and answer neighbourhood queries ("which points are close to x and surround it?"). In Figure 1b, the vertex x has for instance 7 well-defined neighbours.

This paper investigates the use of a DBMS to store and manage not only the points coming from a very large LiDAR dataset, but also the TIN of these points. DBMSs are investigated since they are arguably the best tool to store and manage very large datasets (of any kind). The paper briefly discusses in Sections 2 and 3 current solutions for storing and processing point clouds, and then proposes in Section 4 an alternative which could have several benefits. This proposed solution goes beyond the usual "store points and edges and triangles" described in Section 2, and use recent advances in the compression of graphs.

2 Current solutions for storing and processing point clouds

Many companies offer solutions for the storage and processing of LiDAR data, and the problem is also being tackled in the academia. What follow is a short overview of the most interesting solutions, and does not claim to be a thorough review of all the possibilities. These were choosen because they are, as far as I know, the most used by practitioners and the tools that offer the most functionalities to handle massive point clouds.

Commercial software

The company **Terrasolid**[1] offers several useful tools for the processing of LiDAR data. The main problem here is that, as is the case with many other tools, the size of your computer's main memory decides how many points can be processed. Basically, once the memory is full, transfer of data between the disk and the memory starts, and if this happens too much, the computations might simply stop (called *trashing*). The first solution that Terrasolid and others provide is *tiling* a big dataset into smaller parts that all fit into memory, and working on one given part at a time. While this solution is viable in some cases, for any processing that goes across a "seam" this can be problematic, e.g. calculation of areas/volumes of features, simplification often need a global view of the dataset, flow modelling, etc. The second solution is *thinning*, which means that when reading the input file, the algorithm will only read every 500th point for example. While it permits the user to visualise and process the data, it somehow destroys the idea of working with high-resolution altimetry data in the first place.

Since version 9.2, **ArcGIS** also provides a new type for the storage of very large TINs. In a nutshell, the *Terrain* type stores all the points in a DBMS, but the triangles are not explicitly stored. The user must select so-called "vertical indexes" so that a hierarchy of TINs is created. For each level in the hierarchy, ArcGIS selects representative points to form the TIN (the method used is clear, it appears that they are randomly selected). Because each TIN is small in size when compared to the original dataset, one can use the usual processing for TINs as available in ArcGIS (see Peng et al. (2004) for more details). However, this solution suffers from the same problem as Terrasolid, that is if the TIN created is larger than the memory there is no guarantee that the processing operations will terminate. For instance, if one wants to create a high-resolution grid from a TIN with all the input points, then if the TIN size (points + triangles + topological links) is too big then it will simply not work.

Oracle Spatial, since version 11g, also provides new types for the storage of LiDAR datasets: a *Point Cloud* type and a TIN type, which are similar. For building a Point Cloud, first the input points are "bucketed" into cells containing a given maximum of points (let us say 2500), then a spatial index (a R-tree) is constructed, and finally each point within a cell is also indexed (for TINs, the triangle are simply defined by a reference to 3 vertices, and are also indexed within one cell). Preliminary tests at GDMC/TUDelft have shown that many intermediate results are stored to build the spatial indexes, and these use a lot of memory (Tijssen, 2009). Also, the TIN type does not store the topological relationships between triangles (simply the 3 vertices), which means that these types are probably more targeted towards the storage of very large datasets, and that the manipulation of the surface is limited (to the best of my knowledge Oracle Spatial does not offer any functions to manipulate or analyse the surface at this moment).

Academia

To deal with massive datasets, one can also design external memory algorithms (Vitter, 2001). These basically use disks to store temporarily files that do not fit in memory, and instead of using the mechanism of the operating system, design explicit rules for the swapping between the disk

[1] `www.terrasolid.fi`

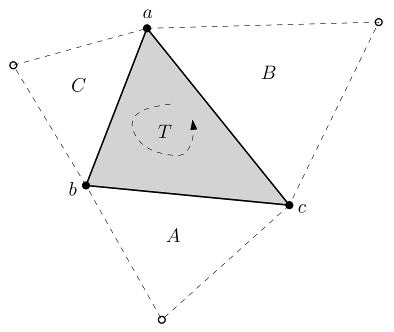


Figure 2: With the triangle-based data structure, the triangle T has 3 pointers to its 3 vertices a, b and c; and to its 3 neighbouring triangles A, B and C. The pair vertex-triangle is organised in such a way that for instance a is 'opposite' to A.

and the memory. These are an alternative to using a DBMS, and have been used for a few LiDAR processing problems, see for instance Agarwal et al. (2006) and Arge et al. (2006). The drawback is that the design of such algorithms is rather complex, and for different problems different solutions have to be designed.

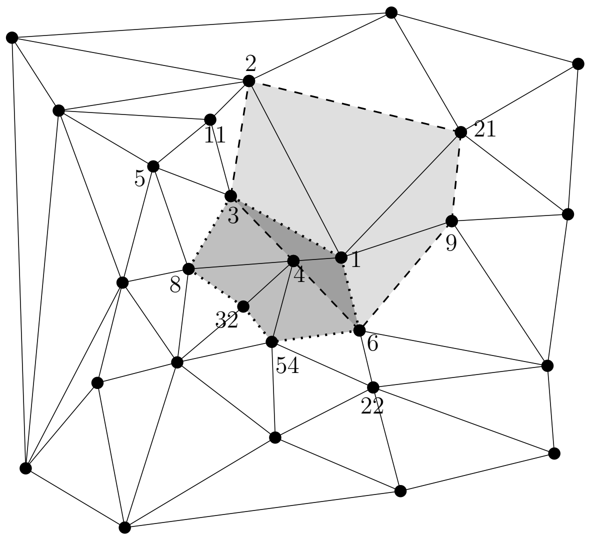
An alternative solution is the *streaming* approach of Isenburg et al. (2006a,b), which mixes ideas from external memory algorithms with different ways to keep the memory footprint very low. The input points are processed in a certain order, and removed from memory when they are not needed anymore. The idea was applied to create a billion-triangle TIN, and they succeeded in under one hour with a conventional laptop (Isenburg et al., 2006b), which improved by a factor of 12 the fastest existing, that of Agarwal et al. (2005). The streaming ideas are very useful for certain *local* problems (e.g. interpolation), but unfortunately cannot be used (or it would be extremely challenging) for *global* processes such as simplification or flow modelling.

3 Storing TINs in a DBMS

The simplest way to store a TIN in a DBMS is to define a new datatype *Triangle* which contains either the coordinates or the IDs of three points in the point cloud (a polygon could also be used). As mentioned previously, that permits us to store the triangle, but topological relationships between the triangles have to be computed (an expensive operation) if analysis of the surface is wished, and it required spatially indexing massive amount of triangles.

An alternative is to implement in a DBMS the triangle-based data structure used by most triangulation libraries, e.g. that of CGAL (Boissonnat et al., 2002). As shown in Figure 2, it considers the triangle its atom and stores each triangle with three pointers to its vertices and three pointers to its adjacent triangles. The vertices and edges of each triangle τ must be oriented consistently (e.g. counterclockwise) if we want to manipulate or perform some operations on a triangulation. This structure stores topological relationships between triangles, but increases the storage space required and at least two tables (points + triangles) have to be maintained when the structure is updated.

More complex data structures for planar subdivisions could be also used (e.g. the DCEL (Muller



$$\boxed{star(1) = 2, 3, 4, 6, 9, 21}$$

$$\boxed{star(4) = 54, 6, 1, 3, 8, 32}$$

Figure 3: A triangulation with the stars of vertices 1 (light grey with dashed lines) and 4 (darker grey with dotted lines) shown.

and Preparata, 1978), the *half-edge* (Mäntylä, 1988)) or the *Generalized Maps* (G-Maps) (Bertrand et al., 1993; Lienhardt, 1994)), but they are very verbose (several primitives are stored and these are linked by several pointers) and would most likely not be optimal in a DBMS.

4 A star-based data structure

The approach I propose in this paper to store and process massive point clouds is storing the whole TIN in a DBMS (the points, the triangles and their topological relationships), and rely on the DBMS for memory management. Since the whole TIN is available (and does not need to be recomputed), one can readily query it and manipulate it. The processing operations would then be built over the database without designing special memory management methods. This solution will most likely be a bit slower than solutions working with main memory, but once the data structure is working and efficient, the benefits of a DBMS should compensate for the lack of speed.

The data structure proposed uses recent advances in the compression of graphs, particularly the "star-based" structure of Blandford et al. (2005), which was developed for main memory. It does not store triangles explicitly, instead the *star* of every vertex is stored. As shown in Figure 3, the star of a given vertex v, denoted star(v), consists of all the triangles that contain v; it forms a star-shaped polygon. The star of a vertex can be simply represented as an ordered list of the IDs of the vertices on the star; a triangle is formed by the centre of the star plus 2 consecutive vertices in the star list. If the star of every vertex in a dataset is stored (thus each star overlaps several other stars), then we can implicitly store not only the triangles of the TIN, but also all their topological relationships. Adjacency between triangles are stored, but also incidence relationships between vertices and edges and triangles (so one can for instance navigate counterclockwise through all the triangles incident to a given vertex). Observe that each triangle is present in exactly 3 stars and each edge in 2 stars

ID	x	y	z	star
1	3.21	5.23	2.11	2–44–55–61–23
2	5.19	29.01	4.55	7–98–111–233–222
3	22.43	15.99	8.19	99–101–73–23
...	...	...	...	...
5674	221.19	15.23	37.81	309–802–793–1111

Figure 4: One example of a table to store the star of each point.

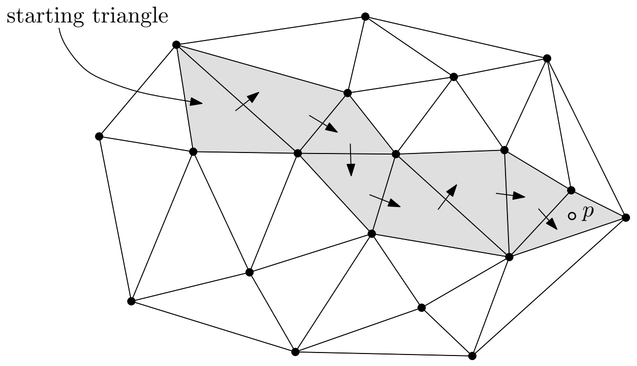


Figure 5: The WALK algorithm for a triangulation.

(in Figure 3 the edge (1,3) is represented in the stars of vertices 4 and 2, in the opposite direction). The data structure is therefore also akin to the the *half-edge* (Mäntylä, 1988) where each edge is stored twice, one for each direction.

As shown in Figure 4, implementing this data structure in a DBMS is straightforward: only one table with 5 attributes ($id - x - y - z - star$) is needed; $star$ is an array of IDs. Perhaps the main advantage of such a structure is that the use of a separate spatial index is not necessary. Indeed, we can rely on the topological relationships between the triangles to perform typical queries such as: (i) given a location (x, y), which triangle contains it?; (ii) range queries: return all the triangles contained in a rectangle; (iii) obtain the triangles adjacent to a given one; etc. Functions have to be implemented inside the DBMS (server-side programming), and only a standard index on the ID column is needed because several SQL queries will be have to be made. An example of such a function is the *walking* in the triangulation (see Figure 5 for an example) where adjacency relationships are used to navigate inside a TIN (see Mücke et al. (1999) for a detailed explanation). Using SQL queries to perform such a walk is usually problematic, but with server-side functions this is possible.

5 Discussion

The proposed star-based data structure is significantly different from what is usually available in a DBMS (i.e. each element (point and/or triangle) is stored independently and a spatial index is used for queries). It permits us to bypass the use of a spatial index at the cost of possible slower answers to queries. However, it is space efficient compared to other solutions described in this paper and the topological relationships between the triangles are explicitly stored, which permits us to process and manipulate the structure. With solutions where the triangles are stored independently, the reconstruction of these relationships is necessary each time an operation needs to be performed.

Furthermore, the structure permits dynamic updates (addition and removal of triangles and/or vertices are possible, with local updates (Blandford et al., 2005)), and these could be implemented directly in the DBMS. The GISt group at the Delft University of Technologies is currently working on implementing and optimising this structure, with basic functions such as interpolation in TINs, slope/aspect derivation and conversion to grids.

The solution proposed is valid not for 2.5D models but also for any boundary representations that can be triangulation (also for closed volumes), and the ideas are readily extensible to higher dimensions. As a consequence, the idea of storing stars of edges in 3D would permit us to efficiently store tetrahedra, and would offer an alternative to the structure of Penninga (2008).

References

P. K. Agarwal, L. Arge, and K. Yi. I/O-efficient construction of constrained Delaunay triangulation. In *Proceedings 13th European Symposium on Algorithms*, pages 355–366, 2005.

P. K. Agarwal, L. Arge, and A. Danner. From point cloud to grid DEM: A scalable approach. In A. Reidl, W. Kainz, and G. Elmes, editors, *Progress in Spatial Data Handling—12th International Symposium on Spatial Data Handling*. Springer, 2006.

L. Arge, A. Danner, H. Haverkort, and N. Zeh. I/O-efficient hierarchical watershed decomposition of grid terrain models. In A. Reidl, W. Kainz, and G. Elmes, editors, *Progress in Spatial Data Handling—12th International Symposium on Spatial Data Handling*, pages 825–844. Springer-Verlag, 2006.

Y. Bertrand, J. F. Dufourd, J. Françon, and Pascal Lienhardt. Algebraic specification and development in geometric modeling. In *Proceedings TAPSOFT'93*, volume 668 of *Lecture Notes in Computer Science*, pages 75–89, Orsay, France, 1993.

Daniel K. Blandford, Guy E. Blelloch, David E. Cardoze, and Clemens Kadow. Compact representations of simplicial meshes in two and three dimensions. *International Journal of Computational Geometry and Applications*, 15(1):3–24, 2005.

Jean-Daniel Boissonnat, Olivier Devillers, Sylvain Pion, Monique Teillaud, and Mariette Yvinec. Triangulations in CGAL. *Computational Geometry—Theory and Applications*, 22:5–19, 2002.

Martin Isenburg, Yuanxin Liu, Jonathan Richard Shewchuk, and Jack Snoeyink. Streaming computation of Delaunay triangulations. *ACM Transactions on Graphics*, 25(3):1049–1056, 2006a.

Martin Isenburg, Yuanxin Liu, Jonathan Richard Shewchuk, Jack Snoeyink, and Tim Thirion. Generating raster DEM from mass points via TIN streaming. In *Geographic Information Science—GIScience 2006*, volume 4197 of *Lecture Notes in Computer Science*, pages 186–198, Münster, Germany, 2006b.

Pascal Lienhardt. N-dimensional generalized combinatorial maps and cellular quasi-manifolds. *International Journal of Computational Geometry and Applications*, 4(3):275–324, 1994.

Martti Mäntylä. *An introduction to solid modeling.* Computer Science Press, New York, USA, 1988.

Ernst P. Mücke, Isaac Saias, and Binhai Zhu. Fast randomized point location without preprocessing in two- and three-dimensional Delaunay triangulations. *Computational Geometry—Theory and Applications*, 12:63–83, 1999.

D. E. Muller and Franco P. Preparata. Finding the intersection of two convex polyhedra. *Theoretical Computer Science*, 7:217–236, 1978.

Wanning Peng, Dragan Petrovic, and Clayton Crawford. Handling large terrain data in GIS. In *ISPRS 2004—XXth Congress*, volume IV, pages 281–286, Istanbul, Turkey, 2004.

Friso Penninga. *3D Topography: A Simplicial Complex-based Solution in a Spatial DBMS.* PhD thesis, Delft University of Technology, Delft, the Netherlands, 2008.

Theo Tijssen. Puntenwolken in Orable RDBMS (in dutch). Internal report, GDMC, Delft University of Technology, 2009.

Jeffrey Scott Vitter. External memory algorithms and data structures: dealing with massive data. *ACM Computing Surveys*, 33(2):209–271, 2001.

Kai Wang, Chor-pang Lo, George A. Brook, and Hamid R. Arabnia. Comparison of existing triangulation methods for regularly and irregularly spaced height fields. *International Journal of Geographical Information Science*, 15(8):743–762, 2001.