TU Delft

Delft University of Technology

Comparative analysis of data structures for storing massive tins in a DBMS

Kavisha, Kavisha; Ledoux, Hugo; Stoter, Jantien

DOI
10.5194/isprs-archives-XLI-B2-123-2016

Publication date
2016

Document Version
Final published version

Published in
International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences

Citation (APA)
Kavisha, K., Ledoux, H., & Stoter, J. (2016). Comparative analysis of data structures for storing massive tins in a DBMS. In *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* (Vol. 41-B2, pp. 123-130). ISPRS. https://doi.org/10.5194/isprs-archives-XLI-B2-123-2016

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.
For technical reasons the number of authors shown on this cover page is limited to a maximum of 10.

COMPARATIVE ANALYSIS OF DATA STRUCTURES FOR STORING MASSIVE TINS IN A DBMS

K. Kumar[a]*, H. Ledoux[a], J. Stoter[a]

[a] 3D Geoinformation, Delft University of Technology, Delft, The Netherlands - (k.kavisha, h.ledoux, j.e.stoter)@tudelft.nl

Commission II, WG II/2

KEY WORDS: Massive, point clouds, TINs, DBMS

ABSTRACT:

Point cloud data are an important source for 3D geoinformation. Modern day 3D data acquisition and processing techniques such as airborne laser scanning and multi-beam echosounding generate billions of 3D points for simply an area of few square kilometers. With the size of the point clouds exceeding the billion mark for even a small area, there is a need for their efficient storage and management. These point clouds are sometimes associated with attributes and constraints as well. Storing billions of 3D points is currently possible which is confirmed by the initial implementations in Oracle Spatial SDO_PC and the PostgreSQL Point Cloud extension. But to be able to analyse and extract useful information from point clouds, we need more than just points i.e. we require the surface defined by these points in space. There are different ways to represent surfaces in GIS including grids, TINs, boundary representations, etc. In this study, we investigate the database solutions for the storage and management of massive TINs. The classical (face and edge based) and compact (star based) data structures are discussed at length with reference to their structure, advantages and limitations in handling massive triangulations and are compared with the current solution of PostGIS Simple Feature. The main test dataset is the TIN generated from third national elevation model of the Netherlands (AHN3) with a point density of over 10 points/m^2. PostgreSQL/PostGIS DBMS is used for storing the generated TIN. The data structures are tested with the generated TIN models to account for their geometry, topology, storage, indexing, and loading time in a database. Our study is useful in identifying what are the limitations of the existing data structures for storing massive TINs and what is required to optimise these structures for managing massive triangulations in a database.

1. INTRODUCTION

Airborne LiDAR technology is nowadays extensively used to collect 3D information in the form of dense point clouds, quickly and with great accuracy. *AHN (Actueel Hoogtebestand Nederland)*, the national elevation dataset of the Netherlands, contains billions of points with a point density of 6-10 points/m^2. With the size of such point clouds exceeding the billion mark even for the smaller areas, difficulties arise in storing and managing these datasets. The existing DBMS solutions such as the Oracle Spatial SDO_PC package (Ravada et al., 2009) and the PostgreSQL Point Cloud extension (Ramsey, 2013) provide only for the storage of point clouds i.e. unconnected points that are samples of the surface representing the Earth. However, in order to analyse and extract useful information like slope/aspect, viewshed analysis, watershed modelling, delineation of basins, etc. from a point cloud, we require more than just points: we must be able to reconstruct and manipulate the surface defined by these points. Surface defines the spatial relationships between the disjoint points in 3D space. The most common method for representing a surface in GIS is the construction of a 2.5D structure such as TIN (Triangulated Irregular Network) (Peucker et al., 1978). Practitioners often think of grids as the simplest way to represent a surface. In fact, It may be the best model to depict the real world scenarios, but this possibility depends on the potential of the grids rather than their current use (Fisher, 1997). In this study, we investigate the database solutions for the storage and management of massive TINs. Here, the term *"massive"* refers to extremely large size which is difficult to manipulate and manage with main memory. The more complex representations of TINs take into account the attributes and constraints associated with the connecting elements. The local density of the points in a TIN can be adjusted in accordance with the variations in the height of original terrain e.g. TINs naturally represent areas of detailed relief with denser triangulation than areas with a smoother relief (Kumler, 1994). While the enormous size of the point clouds is already an issue, storing the derived TIN significantly increases the storage requirements since the number of triangles generated is about twice the number of points taken into account for triangulation (De Berg et al., 2000). AHN2 point cloud dataset has over 640 billion points. The number of triangles generated during triangulation is thus around 1.3 trillion.

Storing massive TINs in the database is much more complicated than storing point clouds since storage of TIN not only requires storing its geometry but the topology as well. The requirement is to reduce the redundancy of explicit representations while still supporting mesh traversal and modifications to dynamically maintain the mesh in the event of local updates like add/delete a vertex/triangle, edge flipping, etc. Essentially, the ideal characteristics of storing TINs in a database include low storage costs for geometry, topology and other attributes, simplicity, easy modifications/updates, faster access to the adjacency information and scalability. Solutions that handle the issue of massive point clouds such as *tiling* are less viable in the case of TINs. When the dataset is too large to be processed within the available main memory, it is split into smaller parts (called *tiles*). The processing is done incrementally on the generated tiles. For the analysis of TINs, the main memory of the computer plays a key role in deciding the maximum size of the dataset that can be processed (Isenburg et al., 2006). If the dataset exceeds the memory size, then the swapping of data between the disk and the main memory starts, thereby increasing the processing time. This process of *tiling* cannot be extended to TINs without breaking its topology (Fig. 1) unless there is addition of explicit points on the tile borders (called *Steiner points*) causing changes in TIN topology (Fig. 2).

*Corresponding author

Oracle SDO_TIN uses a slightly different approach when it comes to tiling the dataset. The point cloud is partitioned into multiple blocks/tiles and each block is uniquely identified with a Block ID. For triangles spanning more than one tile/block in the TIN, the vertex is identified by a pair of Block ID *(bigint)* and Point ID *(bigint)* i.e. for each vertex, references to two IDs (Block ID and Point ID) are required to be maintained. This results in a storage cost of 128 bits for referencing a single vertex, which is very high. To avoid the problems with the storage of TIN topology, one solution can be to store only the 3D points and generating the TIN surface when required. But this can be memory and time expensive operation where there is a change in geometry. For example insertion/deletion of a point will require recomputing the entire TIN structure. In the event of completely storing the TIN (along with the points, triangles and topology) in a database, one can fully depend on DBMS for its management. The entire TIN is available in the database and need not to be recomputed, it can be queried and analysed easily. DBMS solutions offer several advantages over the conventional systems: reduced data access and storage costs, security, multi-user access, scalability, rich query language, etc. (Elmasri and Navathe, 2014).

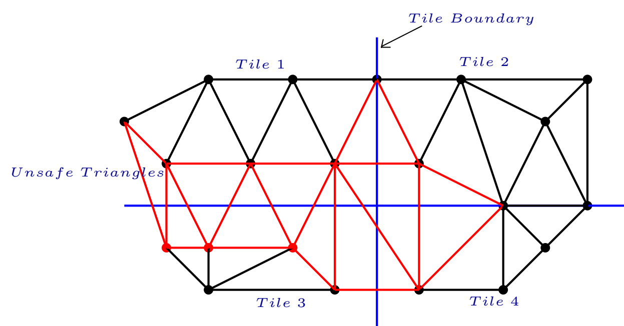

Figure 1: Tiling of TIN

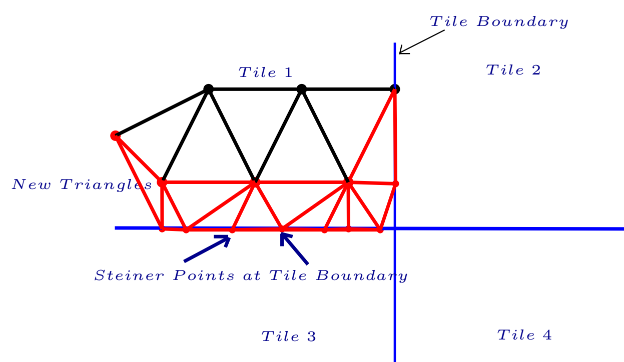

Figure 2: Addition of Steiner Points

There exists several data structures for representing TINs in memory but only a little has been done so far for their database implementations. We provide here a comparative analysis of the existing TIN data structures. The classical (face and edge based) and compact (star based) data structures are discussed at length with reference to their structure, advantages and limitations in handling massive triangulations. Our aim is to compress these structures while maintaining the topological replationships. In this paper, we explain how to implement these structures in a database. These data structures are tested with a few real world datasets and we report on their geometry, topology, storage, indexing, and loading time in a database. The study is useful in deriving what are the limitations of the existing data structures in storing massive TINs and what is required to optimise these structures for managing massive triangulations in a database.

2. RELATED WORK

2.1 TIN Structure

A TIN is a network of non-overlapping triangles formed by the interconnection of irregularly spaced points (Kumler, 1994). They are mostly constructed with Delaunay triangulation but this is not always the case. For a given set of points, different triangulations can be constructed (Rippa, 1990, Dyn et al., 1990). A Delaunay triangulation (DT) of a point set S (Fig. 3) is defined as a triangulation of S such that no point in S lies inside the circumcircle of any other triangle in the triangulation (De Berg et al., 2000). DT maximizes the minimum angle among all possible triangulations to avoid long and skinny triangles.

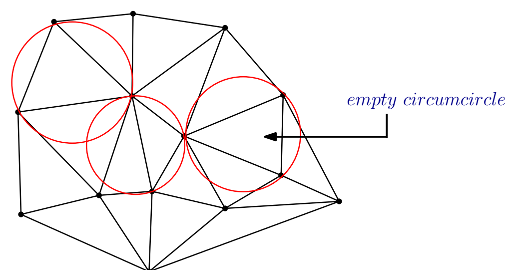

Figure 3: Delaunay Triangulation of a set of points

But everytime the usual inputs for TIN generation are not merely a set of vertices. If the point set is associated to some constraints (segments, polygons, etc.) then a Constrained Delaunay triangulation (CDT) can be constructed. A CDT is similar to DT but every input segment appears as an edge of the triangulation (Shewchuk, 1996). CDT requires that the vertices of every triangle are visible to each other i.e. no input segment (or a part of it) lies between its any two vertices and the circumcircle of each triangle contains no vertices that are visible from the interior of the triangle (Shewchuk, 1996). Fig. 4 and 5 show an example of CDT.

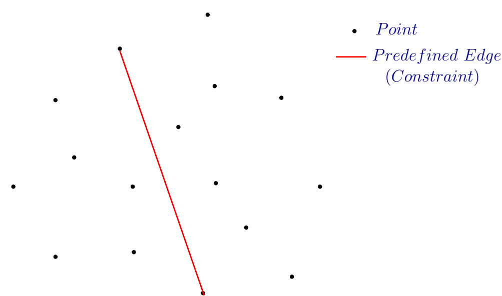

Figure 4: Point set and constrained edge for CDT

2.2 TIN Representations

Several data structures have been proposed for the representation and storage of TINs in memory and it is not surprising to see that they exhibit redundancy and store a lot of information for providing direct retrieval of the adjacency relationships. In this paper, we follow an assumption that the size of the convex hull is relatively small as compared to the total number of vertices in TIN, so there are roughly $2n$ triangles and $3n$ edges in the TIN (De Berg et al., 2000). The term "references" in the text relates to the integer/long integer IDs associated with the vertices/edges/triangles.

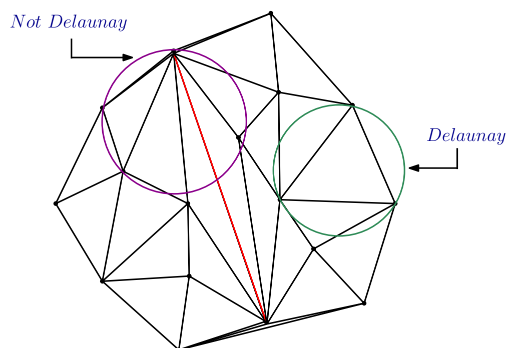

Figure 5: Constrained Delaunay Triangulation of point set and edge

Most of the TIN data structures stores triangulations as an array of vertices or as references to the vertices forming the triangles, thereby capturing the mesh geometry. *OGC Simple Feature* (OGC, 2011) is a classical example of such data structures and is supported by almost all the spatial databases. It stores each triangle *abc* as a linear ring of vertex coordinates with last vertex = first vertex as depicted in Fig. 6. The size of the database increases considerably with the repeated storage of vertex information for each triangle and creation of complex *GiST* spatial index on the triangle geoemetry. The average degree of a vertex in a 2D Delaunay triangulation is 6, given that the vertices follow Poisson distribution (Okabe et al., 2009). This suggests that on an average each vertex is stored *6+6/3 = 8* times. In general terms, each vertex is stored $(t + t/3)$ times in the database with Simple Feature, where t is the number of incident triangles on a vertex. Here the operand $t/3$ suggests the fact that given vertex can be the first vertex (and thereby the last vertex) in $t/3$ number of incident triangles. It does not store explicitly the adjacency relationships between the triangles, which is helpful for the traversal of the triangular mesh.

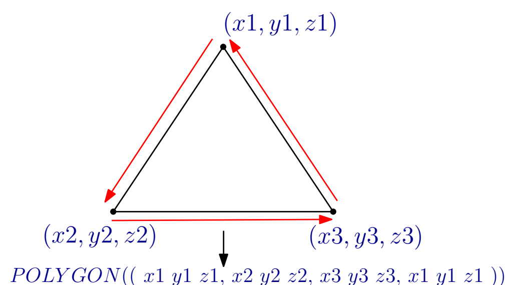

Figure 6: OGC Simple Feature

Similarly, what we refer here as the *Triangle* data structure stores a triangulation as references to the IDs of three vertices forming the triangle (Fig. 7). The coordinates are not repeated here as in the *Simple Feature*. *Oracle Spatial SDO_TIN* (Ravada et al., 2009) uses Triangle for storing TINs. The vertices are stored as *SDO_PC type* and the triangles are stored as references to their three vertices. A notable point of this approach is that the indexing is performed at the block level and not at the vertex or triangle level. The limitation of this approach is that it lacks information about the adjacency and incidence relationships between the triangles and the vertices are not stored explicitly. Also, it suffers from the problem of redundancy, when the triangles span more than one block (Fig. 1). Maintaining the topology of the TIN is by far the most expensive operation.

A more convenient approach is to store the triangles along with their adjacency information (here we refer to it as *Triangle+*).

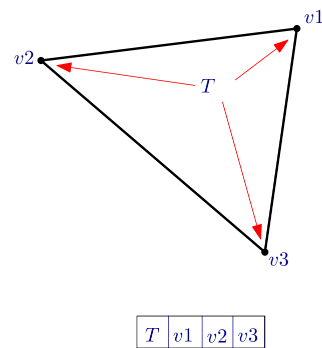

Figure 7: Triangle

CGAL (Computational Geometry Algorithms Library) 2D triangulations (Boissonnat et al., 2002) and Shewchuk's Triangle (Shewchuk, 1996) use this data structure. Apart from storing references to the 3 bounding vertices v_1, v_2, v_3, it also stores references to the 3 adjacent triangles $T1, T2, T3$ for maintaining the topology as depicted in Fig. 8. Thus, it requires *2*(3+3) = 12rpv* (references per vertex) or *3+3 = 6rpt* (references per triangle) for representing the TIN models. The storage requirements are increased with the presence of adjacency relationships.

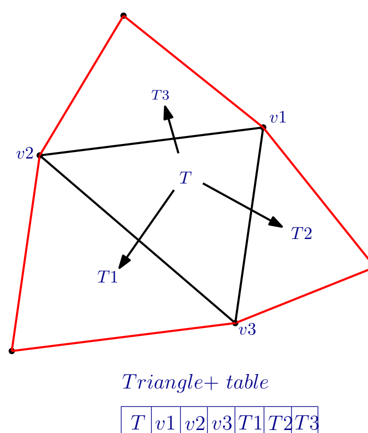

Figure 8: Triangle+

Likewise, the edge based data structures namely, the *Half-edge* (Mäntylä, 1987) or *DCEL (Doubly Connected Edge List)* (Muller and Preparata, 1978) represents each edge as a composition of two half-edges (counter clockwise and opposite) in a slightly non-intuitive fashion (Fig. 9). Apart from storing the adjacency relationships (i.e. references to previous edge $p(e)$, next edge $n(e)$ and the incident face $f(e)$), DCEL maintains references to the incident half edges e and e' and to the bounding vertex v (De Berg et al., 2000). While the number of edges e in a triangulation is about thrice the number of vertices n, this results in $3e + n$ references for the mesh with $19rpv$ or *6rpt* . Other variations like the *Winged-edge* (Baumgart, 1975) stores eight references per edge of the triangle. The storage requirements of these data structures is high which is accompanied by the large size of their spatial index.

Apart from the classical representations, several compact data structures for the triangular meshes have been described to reduce the number of references required and the memory requirements while still ensuring an efficient implementation in terms of run time and mesh traversal operations. Blandford et al. (2005) describes a compressed, pointer less, *star* based data structure for compactly representing the triangular meshes in two and three

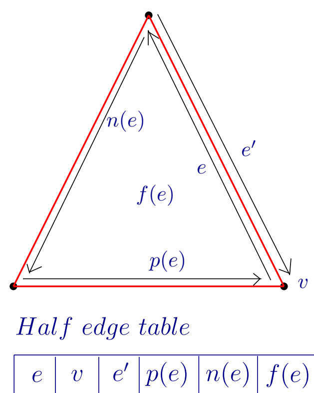

Figure 9: DCEL

dimensions. Each element in the star of a vertex is encoded as the difference of its label value from that of the original vertex. These differences are represented with 4-bit codes therefore, the data structure uses comparatively much less memory than the traditional representations i.e. by a factor of 5 (Blandford et al., 2005). However, this compression scheme is not reasonable while dealing with massive datasets in the DBMS, for e.g. to perform point/range queries, the data needs to be decoded which is not time efficient. For such cases, the uncompressed version of the star data structure *pgTIN* (Ledoux, 2015) to store TINs in a database is of relevance. The star of a bounding vertex $star(v)$ is represented as an ordered list (counter-clockwise) of vertex labels $v_1, v_2, v_3, .., v_i$ forming the link (Fig. 10). The triangles are computed on-the-fly. Each incident triangle at the bounding vertex v is represented by v and the two consecutive vertices v_i in the list. On an average, each vertex in the star stores reference to 6 neighboring vertices, which results in a storage cost of *6rpv*. The main advantage of the star data structure is that it does not require creation of complex spatial index like giST. The indexing is done at vertex level with a simple B-tree.

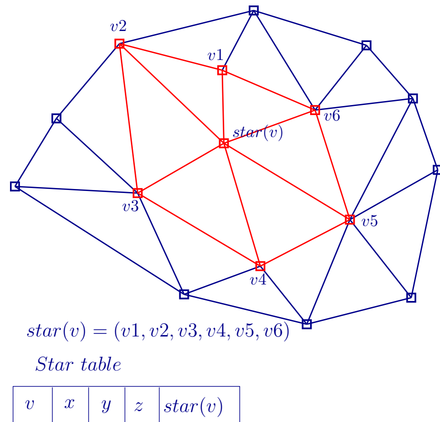

Figure 10: Star of a vertex

For practical use, we attempt to create a data structure which can be built on the fly, simultaneously with mesh generation/processing. Table 1 highlights the comparison between the existing data structures for the triangular meshes.

3. TINS IN DBMS

Traditionally, DBMS were used for handling admistrative and other voluminous data but now they have evolved to integrate the spatial component. At present, database systems like Oracle, PostGIS, IBM DB2, Ingres, Informix, MySQL, etc. provide support for spatial data types, spatial indexing and other extended functionalities. Storage and analysis of spatial data can be done with SQL queries. Most of these database systems offer 2D data types (generally point, line, and polygon). Only two database implementations for TIN type are seen so far, namely *Oracle SDO_TIN* and *OGC Simple Feature Access model* in PostGIS. *OGC Simple Feature Access model* (OGC, 2011) specifies the geometrical model to be followed for representing geographical objects. It describes triangle as a polygon with 3 non-collinear and distinct points and not having any interior rings or holes. A TIN is described as a contiguous collection of triangles (a polyhedral surface) which share common edges. In 3D, with the inclusion of z-values (height), they are refered as TriangleZ and TINZ respectively. At present, the support for storing TriangleZ and TINZ in the database is provided by PostGIS v2. The WKT (Well Known Text) for representing TriangleZ and TINZ is:

TRIANGLEZ ((0 0 0, 0 1 0, 1 1 0, 0 0 0)) and

TINZ (((0 0 0, 0 0 1, 0 1 0, 0 0 0)), ((0 0 0, 0 1 0, 1 1 0, 0 0 0))).

In Simple Feature, the geometry is stored with a unique ID, bounding box, and WKB (Well Known Binary) representation of OGC Geometry. Fig. 11 describes the WKB format for OGC Simple Feature type.

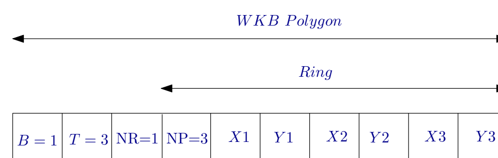

Figure 11: WKB of OGC Simple Feature

Here B represents the Byte order (big endian = 0 and little endian = 1), T represents the WKB Geometry type, NR denotes number of rings in the geometry, NP denotes number of points in the geometry and X, Y, Z are the coordinate values of the points.

Fig. 12 describes the structure of Triangle and TIN type in PostGIS. PostGIS TIN type stores only the geometry of TIN and not the topology.

Oracle SDO_TIN makes use of *Triangle* data structure for storing triangulations (Fig. 14). It is an extension of *Oracle SDO_PC* type with almost same structure and storage scheme but has an additional column to store the TIN objects. The *SDO_PC* partitions the point cloud into multiple blocks which are uniquely identified by their block IDs and then spatially indexed (Finnegan and Smith, 2010). The *SDO_TIN* object stores TIN metadata and a reference to the TIN Block table *(SDO_TIN_BLK)*. Fig. 13 explains the structure of *SDO_TIN*. The TIN is divided into blocks/buckets and is stored in the TIN Block table with points and triangles as BLOBs (Binary Large OBjects). A point BLOB consists of an array of points of size d * 8-bytes (d = dimensions, 2 in our case) along with Bucket/Block ID and Point ID *(BLK_ID, PT_ID)* of each point. Each triangle BLOB comprises of an array of triangles, wherein each triangle has 3 vertices and each vertex is refered by the pair of Bucket/Block ID and Point ID *(BLK_ID, PT_ID)*. Each row of the *SDO_TIN_BLK* table can be considered itself as a complete and connected bucket. Moreover, this explicit storage of block ID with each point solves the issue of dividing TIN with bucketing. (Ravada et al., 2009).

The *Oracle SDO_TIN* does not store the topology of TIN. This can be supplemented with the triangle adjacency information as

Data Structure	Type	rpt	Topology	Features	Limitations
PostGIS SF	Face based	4	X	Face as linear ring of vertices	Vertex information redundancy, No topology, Large spatial index
Oracle Triangle	Face based	3	X	Triangle as reference to 3 vertices only, Bucketing of vertices, Block level indexing	No topology
Triangle+	Face based	6	✓	Triangle as reference to 3 vertices only, Explicit topology storage	Triangle level indexing
DCEL	Edge based	6	✓	Triangles as reference to the half edges	Edge information redundancy, Large spatial index
Star	Vertex based	3	✓	Triangles as ordered list of vertex labels and are computed on the fly, Vertex adjacency information, More compact, No spatial indexing but simple B-tree indexing at vertex level	No dynamic updates

Table 1: Existing TIN data structures

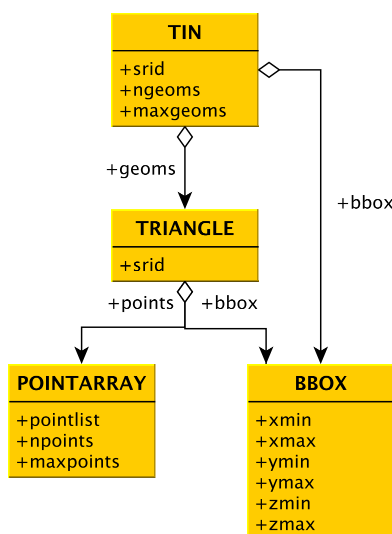

Figure 12: Triangle and TIN type in PostGIS

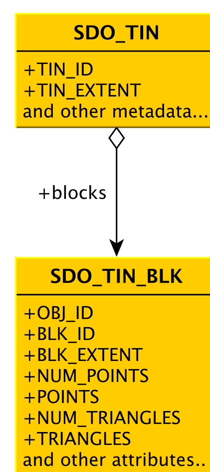

Figure 13: TIN in Oracle

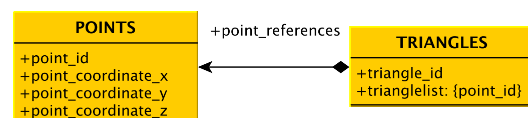

Figure 14: Storing Triangle

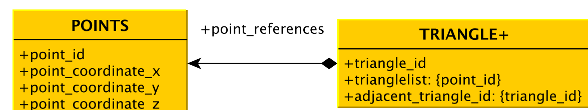

Figure 15: Triangle+ UML

seen in the *Triangle+* data structure in section 2. Storing triangulation requires a table where each row contains the IDs of the three vertices forming the triangle. Adding the adjacency information requires creation of another table with triangle ID and IDs of three neighbouring triangles sharing common edges with it. Fig. 15 shows the structure of *Triangle+* approach in the database. Using this approach, only *6rpt* are required to store a single triangle with topology in the database.

Among other implementations, the *pgTIN* extension for PostgreSQL (Ledoux, 2015) stores TIN as uncompressed stars. It only makes use of a vertex table with each row containing a vertex ID, vertex coordinates and an array of references to the vertices in the star of the vertex as explained in section 2. Fig. 16 describes the structure of stars in a database. Triangles are computed on-the-fly using the explicitly stored topology. Thus, only *3rpt* are required to represent a triangle. However, each vertex in the table is indexed, although using a binary tree, the index so

generated is large.

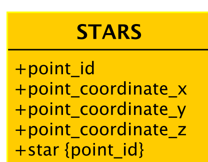

Figure 16: Stars UML

Fig. 17 depicts the row structure of the above discussed data structures in PostgreSQL.

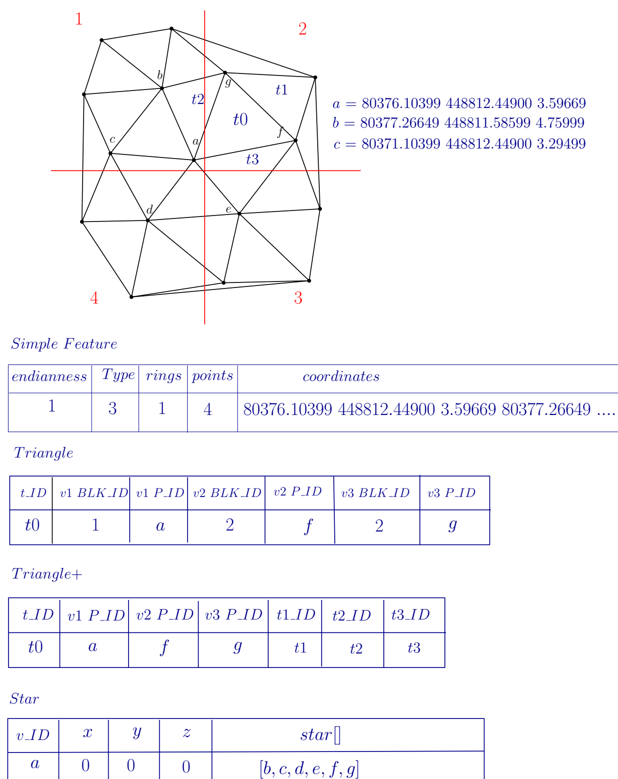

Figure 17: Row structure in SF, Triangle, Triangle+ and Star

4. IMPLEMENTATION

4.1 Construction of TIN

Two filtered subsets and an AHN3 Tile# 37EN/1 (PDOK, 2015) of the massive AHN3 point cloud dataset are utilised as inputs for the construction of TIN. The TIN is generated based on the concept of spatial streaming given by Isenburg (2006). The mesh construction pipeline makes use of *spfinalize* and *spdelaunay2d* modules of the blast extension of Lastools. These modules exploit the spatial coherence of the massive point clouds to compute a streaming triangulation. (Isenburg et al., 2006). The TIN is created and stored without any attributes and constraints. The details of the input datasets are given in Table 2. It lists the number of points in each subset, the number of triangle generated in triangulation, the points forming the convex hull of the subset and the average and maximum number of edges incident on a vertex i.e. the average and maximum degree of a vertex in the triangulation.

4.2 Testing

The PostgresSQL/PostGIS DBMS is used for storing the TIN. For the storage of vertex IDs and triangle IDs, 64-bit large range integers i.e. *bigint* because 32-bit integers limit the dataset to $2^{32}/2$ (i.e. around 2 billion) IDs. The coordinates *(x,y,z)* of the vertices are stored in three separate columns as 64-bit *double precision*. For the storage of *star* of a vertex, variable length integer *array* is used because the link of a vertex can contain 2 to an infinte number of vertices. To store the triangles, the triangle ID and the IDs (64-bit integers) of its three vertices are stored in three separate columns. The adjacency relationsips for the *Triangle+* are stored in a similar manner. For the storage of attributes or constraints associated with the triangles, a new column with proper data type must be created. The python implementations of the data structures are used to populate the PostgreSQL/PostGIS database using the *.smb outputs of *spdelaunay2d*. The testing is done on Intel(R) Xeon(R) CPU E5-2650 v3 (2.30GHz) running Ubuntu (Linux), Python 2.7.6 and PostgreSQL v9.3.11.

To compare these data structures, the TIN is stored:
1. in the *triangle/triangle+/star* based representations with *B-tree* index.
2. in the *PostGIS Simple Feature* with *GiST* index.
Table 3 enlists the time taken for the construction and loading of TIN and spatial index of datasets for different data structures. The storage space used in PostgresSQL is given in table 4.

5. RESULTS AND DISCUSSION

From Table 3, it is observed that it takes significantly large amount of time to populate and index the TIN datasets with PostGIS SF i.e. ~15 hours for a TIN generated from one tile of AHN3. The minimal time is taken by the Star data structure i.e ~4 hours, followed by the Triangle. This is around 4 times less than that of the SF. This can be owed to the construction of complex PostGIS GiST index which took around 14 times more time than indexing stars with a B-tree.

From Table 4, it is clear that GiST is larger than B-tree as it takes the maximum disk space i.e. ~202 GB for storing a TIN generated from a single tile of AHN3. Stars take the least space of about 77 GB for the same. Storing TINs in PostGIS SF is memory expensive because of two reasons: First, the triangles are almost twice in number than the points and with vertex information redundancy there is a significant wastage of storage space of (number of triangles * (t+t/3) * 8) bytes. Second, the bounding box of every triangle is explicitly stored in PostGIS. The memory requiement statistics for the data structures in the database can be seen as:
1. *SF* - (number of triangles * 4 * number of dimensions * 8) bytes
2. *Triangle* - ((number of triangles * 4 * 4) + (number of vertices * 3 * 8) bytes
3. *Triangle+* - ((number of triangles * 7 * 4)+ (number of vertices * 3 * 8)) bytes
4. *Star* - (number of vertices * 4 * number of elements in each link * 4) bytes

Subset	Points	Triangles	Convex Hull	Avg Degree	Max Degree
AHN3 S1	40 506 390	81 012 707	29	5.99(6)	321
AHN3 S2	220 506 390	442 022 687	1168	5.99(6)	238
AHN3 Tile	508 564 458	1 117 129 823	657	5.99(6)	418

Table 2: AHN3 Inputs

Data Structure	AHN3 S1			AHN3 S2			Tile		
	Construction	Indexing	Total	Construction	Indexing	Total	Construction	Indexing	Total
SF	24.93 min	24.72 min	49.65 min	174.5 min	183.8 min	358.3 min	422.8 min	513.6 min	936 min
Triangle	23.8 min	1.3 min	25.1 min	118.5 min	11.8 min	130.3 min	287.4 min	34 min	321 min
Triangle+	44.2 min	1.4 min	45.6 min	202.9 min	11.8 min	214.7 min	613.2 min	34.6 min	647.8 min
Star	13.3 min	2.8 min	16.1 min	73.2 min	15.6 min	88.8 min	212 min	35.8 min	247.8 min

Table 3: Runtime for the pipeline

Data Structure	AHN3 S1			AHN3 S2			Tile		
	Table	Index	Total	Table	Index	Total	Table	Index	Total
SF	13 GB	4.2 GB	17.2 GB	61.4 GB	25.8 GB	87.2 GB	143.5 GB	58.2 GB	201.7 GB
Triangle	5.7 GB	1.7 GB	7.4 GB	22.9 GB	13.3 GB	36.2 GB	53.8 GB	29.5 GB	83.3 GB
Triangle+	9.2 GB	1.7 GB	10.9 GB	48.6 GB	13.3 GB	61.9 GB	134.2 GB	29.5 GB	163.7 GB
Star	4.3 GB	868 MB	5.1 GB	26.4 GB	4.3 GB	30.7 GB	63.2 GB	13.8 GB	77 GB

Table 4: Storage size in Database

6. CONCLUSION

Storage and analysis of massive TINs can be tackled in different ways. One way is to store them in file based systems either as point clouds (*.laz, *.xyz, etc.) or as a collection of triangles (*.obj, *.smb, etc.). But with increasing size of the point clouds and derived TINs, the storage size and requirements are also increased. Therefore, we have investigated in this paper the data structures which can be implemented as DBMS solutions for the storage of TINs. From the tests conducted, it can be concluded that *PostGIS SF* can be overlooked for the storing TINs as far as memory usage, loading and query times, indexing and topology is concerned. Although storing triangles as references to their vertices is quick, it suffers from lack of topological information when it comes to TIN traversal and updates. Other representations like *DCEL* seems to be excessive in terms of adding bookkeeping concerns for object references which require a dependency over robust memory management subsystem to work efficiently with massive triangulations. These classical structures are not useful if the mesh contains a large number of unconnected triangles since the neighbouring information depends on the connectivity of triangles. A compact yet fully connected data structure is the requirement for storage efficient representation of TINs. The requirement for storage efficient representations for triangular meshes has resulted in a number of compression methods and light weight data structures like *SQuad* (Gurung et al., 2011a), *Grouper* (Luffel et al., 2014), *Laced Ring (LR)* (Gurung et al., 2011b), *Zipper* (Gurung et al., 2013), and *Tripod* (Snoeyink and Speckmann, 1999). These data structures just stores a set of core links from which other links can be inferred. In the testing results, *Triangle+* and *Star* seems to be promising representations for the storage and management of TINs. The future research holds for rigourous testing with the compact representations and comparing the analysis results with that of the *Triangle+* and *Star*. Data structures are also to be implemented with tiling the dataset into non-massive tiles/blocks to see how efficiently the topological relations across the tiles can be inferred, thereby making it possible to traverse across the tiles. The data structures (with and without tiling) are to be tested for their practical performance by implementing spatial analysis functions and triggering spatial queries on the stored data.

ACKNOWLEDGEMENTS

This research is supported by the Dutch Technology Foundation STW, which is part of the Netherlands Organisation for Scientific Research (NWO), and which is partly funded by the Ministry of Economic Affairs.

REFERENCES

Baumgart, B. G., 1975. A polyhedron representation for computer vision. In: Proceedings of the May 19-22, 1975, national computer conference and exposition, ACM, pp. 589–596.

Blandford, D. K., Blelloch, G. E., Cardoze, D. E. and Kadow, C., 2005. Compact representations of simplicial meshes in two and three dimensions. International Journal of Computational Geometry and Applications 15(1), pp. 3–24.

Boissonnat, J.-D., Devillers, O., Pion, S., Teillaud, M. and Yvinec, M., 2002. Triangulations in CGAL. Computational Geometry—Theory and Applications 22, pp. 5–19.

De Berg, M., Van Kreveld, M., Overmars, M. and Schwarzkopf, O. C., 2000. Computational geometry. Springer.

Dyn, N., Levin, D. and Rippa, S., 1990. Data dependent triangulations for piecewise linear interpolation. IMA journal of numerical analysis 10(1), pp. 137–154.

Elmasri, R. and Navathe, S. B., 2014. Fundamentals of database systems. Pearson.

Finnegan, D. C. and Smith, M., 2010. Managing LiDAR topography using Oracle and open source geospatial software. In: Proceedings GeoWeb 2010, Vancouver, Canada.

Fisher, P., 1997. The pixel: a snare and a delusion. International Journal of Remote Sensing 18(3), pp. 679–685.

Gurung, T., Laney, D., Lindstrom, P. and Rossignac, J., 2011a. SQuad: Compact representation for triangle meshes. In: Computer Graphics Forum, Vol. 30number 2, Wiley Online Library, pp. 355–364.

Gurung, T., Luffel, M., Lindstrom, P. and Rossignac, J., 2011b. LR: compact connectivity representation for triangle meshes. Vol. 30Number 4, ACM.

Gurung, T., Luffel, M., Lindstrom, P. and Rossignac, J., 2013. Zipper: A compact connectivity data structure for triangle meshes. Computer-Aided Design 45(2), pp. 262–269.

Isenburg, M., Liu, Y., Shewchuk, J. and Snoeyink, J., 2006. Streaming computation of Delaunay triangulations. In: ACM Transactions on Graphics (TOG), Vol. 25number 3, ACM, pp. 1049–1056.

Kumler, M., 1994. An intensive comparison of triangulated irregular networks (TINs) and digital elevation models (DEMs). Cartographica 31(2), pp. 1–99.

Ledoux, H., 2015. Storing and analysing massive TINs in a DBMS with a star-based data structure. Technical Report 2015.01, 3D geoinformation, Delft University of Technology, Delft, the Netherlands.

Luffel, M., Gurung, T., Lindstrom, P. and Rossignac, J., 2014. Grouper: A compact, streamable triangle mesh data structure. Visualization and Computer Graphics, IEEE Transactions on 20(1), pp. 84–98.

Mäntylä, M., 1987. An Introduction to solid modeling. Computer Science Press, Inc., New York, NY, USA.

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

OGC, 2011. Opengis® implementation specification for geographic information-simple feature access-part 1: Common architecture. OGC document version 06-103r4.

Okabe, A., Boots, B., Sugihara, K. and Chiu, S. N., 2009. Spatial tessellations: concepts and applications of Voronoi diagrams. Vol. 501, John Wiley & Sons.

PDOK, 2015. AHN3 downloads. `https://www.pdok.nl/nl/ahn3-downloads/`. (Last accessed: April 10, 2016).

Peucker, T. K., Fowler, R. J., Little, J. J. and Mark, D. M., 1978. The triangulated irregular network. In: Proceedings Digital Terrain Models Symposium, American Society of Photogrammetry, pp. 516–532.

Ramsey, P., 2013. LIDAR in PostgreSQL with Pointcloud. `http://s3.cleverelephant.ca/foss4gna2013-pointcloud.pdf`. (Last accessed: April 10, 2016).

Ravada, S., Kazar, B. M. and Kothuri, R., 2009. Query processing in 3d spatial databases: Experiences with oracle spatial 11g. In: 3D Geo-Information Sciences, Springer, pp. 153–173.

Rippa, S., 1990. Minimal roughness property of the Delaunay triangulation. Computer Aided Geometric Design 7(6), pp. 489–497.

Shewchuk, J. R., 1996. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. Applied Computational Geometry: Towards Geometric Engineering 1148, pp. 203–222.

Snoeyink, J. and Speckmann, B., 1999. Tripod: a minimalist data structure for embedded triangulations. Computational Graph Theory and Combinatorics.