Extended abstract for the invited presentation at the IQmulus Workshop on Processing Large Geospatial Data, 8 July 2014, Cardiff, Wales, UK (version 23 May 2014)

Point cloud data management

Peter van Oosterom, Siva Ravada, Mike Horhammer, Oscar Marinez Rubi, Milena Ivanova, Martin Kodde and Theo Tijssen

Abstract

Point cloud data are important sources for 3D geo-information. The point cloud data sets are growing in popularity and in size. Modern *Big Data* acquisition and processing technologies, such as laser scanning from airborne, mobile, or static platforms, dense image matching from photos, multi-beam echo-sounding, or from autotracking seismic data, have the potential to generate point clouds with millions or billions (or even trillions) of 3D points (with in many cases one or more attributes attached). This is especially true with the available and expected repeated scans of same area (the temporal dimension). These point clouds are too massive to be handled efficiently by common geo-ICT infrastructures. At the database level, initial implementations are available in both commercial and open source database products, illustrating the user need for point cloud support; e.g. Oracle spatial's SDO_PC data type and PostgreSQL/PostGIS PCPATCH data type. This new data type should be available in addition to the existing vector and raster data types. Further, a new and specific web-services protocol for point cloud data is investigated, supporting progressive transfer based on multi-resolution. The eScience project investigates solutions in order to better exploit the rich potential of point cloud data. The project partners are: Rijkswaterstaat (RWS), Fugro, Oracle, Netherlands eScience Centre and TU Delft. An inventory of the user requirements has been made using structured interviews with users from different background: government, industry and academia. Based on these requirements a benchmark has been developed to compare various point cloud data management solutions w.r.t. functionality and performance. The main test data set is the second national height map of the Netherlands, AHN2, with 6 to 10 samples for every square meter of the country, resulting in more than 600 billion points with 3 cm accuracy.

1. Introduction: DBMS approach

Why try to manage the large volumes of point cloud data in a DBMS environment? Why not continue further along today's common practice to use a (collection of) file(s) and specific tools (programs, libraries) to select, analyze, manipulate and visualize the point cloud data? Point cloud data shares the sampling nature which is also behind raster data and the same question was raised in this context (and similar answers can be given). Both types of data (both raster and point cloud) are often quite massive, but relatively static. As there is limited requirement for update (or delete, insert) of individual points in a multi-user context, there is so not so much need for DBMS transaction support. It is indeed true, that it is possible to develop specific file based solutions (Rapidlasso's LAZ, ESRI's ZLAS,..) for point cloud 'data management'. However, when data sets will only grow and grow, more and more data management challenges can be expected (and in the end we are re-developing large parts of a DBMS). For example, our test data (AHN2) is stored and distributed in 60.000 LAZ files, which is not really efficient for simple point cloud data selection purposes even with the quite reasonable LAZ tools (as from Rapidlasso). Here a DBMS could provide an easier to use and more scalable alternative.

Further, a specific (ad hoc) file-based solution may be good at one aspect (handling point clouds), but in reality users have a range of different data sets and types: administrative data, vector data, raster data, temporal data, etc. Therefore a standardized and generic DBMS solution would be preferable as users want to combine, vector and point cloud data in their queries; e.g. select points from a point cloud data set, which overlap with a set of 3-meter buffered polygons around buildings owned by certain person. There is an interesting list of spatial functionality for the support of point clouds (at DBMS level); see Annex A.

Given the huge volumes of point cloud data, performance is critical, both with respect to data storage/ transfer size and to speed (for loading and querying). Therefore, it will be a real challenge to make an efficient implementation. However, several techniques/ theories are available and there is no fundamental reason why this can not be done is a DBMS context. The initial implementations such as Oracle spatial's SDO_PC data type and PostgreSQL/PostGIS PCPATCH data type are steps in right direction. It is important to assess the current implementations as fair as possible via testing, or benchmarking and comparing various solutions (Postgres, Oracle, MonetDB and sometime even file based approaches).

2. User expectations/requirements

The input of point cloud data users was obtained via an user questionnaire. A wide range of users, varying from scientists at the TU Delft (and Utrecht University) and government users at RWS to users from Fugro's network (partners and customers), with various research and production tasks/ applications, participated. This approach was applied to obtain a good overview of relevant functionality. In addition, in a group session on 31 may 2013 with staff from Oracle, TU Delft, and NL eScience centre (the author team), the importance of the various requested point cloud query (analysis) functionalities was assessed.

3. Conceptual benchmark

A 'conceptual' benchmark was developed based on the users requirement at a relative high abstraction level (not platform specific) to assess various solutions for point cloud data management. This covers both the functionality richness and the performance (storage space, loading times, and query times). The conceptual benchmark has to be translated into a platform specific benchmark, which can then be executed. The benchmark includes specification of test data and queries (including the type of query geometries when relevant). The structure or specification of conceptual benchmark, covers the following aspects:

X. Storage (platforms: Oracle, MonetDB, PostgreSQL flat table/PC_type, parameters)
A. Datasets (range of different data sizes, and with/without additional attributes)
B. Query return method (count, create table … as select, local frontend, webservices)
C. Query accuracy level (storage blocks, 2D/3D query rectangle, 2D/3D exact geometry)
D. Number of parallel users (1 to 10.000, mainly selections)
E. Query types (functionality based on user requirements, also see Annex A)
F. Manipulations (adding new data, compute attributes, but individual point updates rare)

4. Executable benchmark

Before running a full benchmark, it is good to first execute a smaller functional test (limited data sets and limited queries), called the *mini-benchmark (with 20.000.000 points, 20 million).* The number of tests in the benchmarks becomes very large as many of the described aspects are 'orthogonal', resulting in very large number different tests, even without any repeating of tests (to be more sure about the results/ measurements). Therefore initial mini-benchmark tests are executed to gain experience. The queries of the mini-benchmark include the following 7 types: 1. Small rectangle, axis aligned (2.703 m^2), 2. Large rectangle, axis aligned, (49.506 m^2), 3. Small circle radius 20 m, 4. Large circle at radius 115 m, 5. Simple polygon specified by 9 points, 6. Complex polygon specified by 792 points (including one 1 hole), and 7. Long narrow diagonal rectangle. Below the first query of the mini-benchmark in Oracle:

```
  CREATE TABLE query_res_1 AS
  SELECT * FROM table (sdo_pc_pkg.clip_pc(SDO_PC_object,
  (SELECT geom FROM query_polygons WHERE id = 1), NULL, NULL, NULL, NULL));
```

After these initial tests and gradually up-scaling, it has been decided which tests options (most promising configurations) are part of the *medium-benchmark (with 20.000.000.000 points, 20 billion).* While benchmarking, the used databases (and parameter settings) are analyzed, and when needed improved; e.g. in Oracle a faster blocking approach as been proposed and implemented. The medium-benchmark will also contain more functionality: 1. polyline with buffer query, 2. height query in area (find all points below -1 meter in rectangle), and 3. nearest neighbor query. Also the query geometries of the medium-benchmark are in a different part of the domain. The medium-benchmark is a half-way sanity check before loading and querying the complete AHN2 point cloud data. By assessing various systems (different databases/ configurations, and even different hardware platforms; besides a 'normal' server also an Oracle EXADATA machine), strong and weak points of various approaches will be identified. This will support the work towards an 'optimal' solution. The testing itself is automated up to a large extend, both execution and processing of results, given the huge number of test options. If medium-benchmark works well, then it is time to scale up and run more *full-benchmark*: complete AHN2 data set and more queries/ functionality. In the future also further scalability is tested by replication the AHN2 data: *upscaled-benchmark*; e.g. 20.000.000.000.000 points, 20 trillion, about 30 times the current AHN2 data sets).

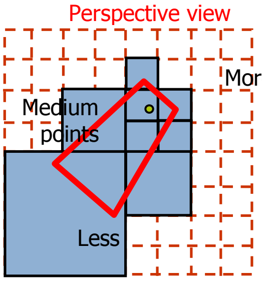
5. Data organization, blocking, clustering, compression

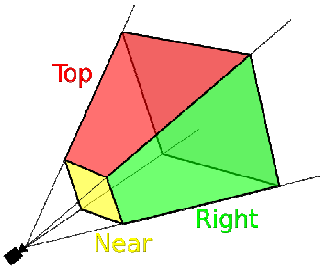
Given the size of point cloud data, techniques have to be applied to work efficiently with these large data sets. Besides using the new PC data types, for reference (and other) purposes also the flat table model is assessed (one point per row). First of all, this can be used as comparison to measure the improved efficiency of a PC data type extended database (amount of storage needed, and query performance). The flat table storage can also be considered as a flexible intermediate stage, where it is relatively easy to organize, sort points for various purposes; e.g. according to their Hilbert-code or Morton-code value (exploiting spatial coherence). After preparing points in this flat table, it is easier to create the more efficient, more compact PC data type representation as a sequence of rows, will form the content of a block. The PC data type representation has several benefits: is spatially coherent, may be used for simple compression (first bits/digits of all coordinates are equal and may be stored at block level), has less tuple overhead, may be used for caching, etc. However, it might also be possible to use the flat model directly (e.g. in case no PC data type is available or in case the database hardware/software can handle very well compression and fast selections on flat table, such as Oracle's EXADATA machine). Part of the design of a good point cloud data management system is specifying the parameters for several options (before actual data loading, and which might impact significantly on data use/query):

1. 2D/3D clustering/indexing (or 4D with attribute, time, intensity)
2. block sizes (100, 1.000, 5.000, 10.000 points per block)
3. compression options (none, medium, high)
4. pyramid options (with/ without duplication between levels, how to select points for higher level, etc.)

6. Vario LoD/data pyramid

Given the amount of data, it is not always needed and efficient to send all points overlapping the query selection region form server to client (too much detail which is not visible anyhow, but which will take data communication and processing time). Therefore, data pyramid solutions have been implemented. For example, a selection of the points is moved to a higher level block combined with selected points form neighbor blocks resulting in a higher level block with larger extend. Typically, at each level the edge size of a block rectangle is doubled and in case of 2D blocking, the 4 lower level blocks are covered by one upper level block (and 1/4th of the bottom level points are repeated at higher level block, or in case of no duplication 1/5th of the data of bottom level block is moved to higher level block). As a result all blocks contain same amount of data. The data pyramid can be used to realize a perspective view query:





The further away from viewer the lesser points selected (i.e. the higher level blocks/points). The drawback of this approach is that there are a discrete number of levels and viewer may notice the difference in density between neighboring blocks of a different level. We will therefore investigate the possibility of vario-scale LoD (and storage techniques). This is work in progress, but we intend to apply lessons from vario-scale research: add one dimension to the geometry (2D data vario-scale represented by 3D geometry) and apply this to point cloud data. This starts by computing the importance value of a point (comparable to LoD level, but now not limited to discrete levels) and use this as the additional imp-height dimension. So, the point cloud data sets either becomes: x,y,imp (z and others are treated as attributes) or x,y,z,imp (and others as attributes). Next compute clustering/indexing of the 3D (or 4D) points using Hilbert or Morton code, sort points, and create blocks and index the blocks with 3D (or 4D) R-tree. This is more or less identical to the normal non-LoD point cloud blocking, but now with one more dimension (imp).

During the use of the vario-scale cloud data, we define perspective the view selections: the view frustum object gets one more dimension (importance): further from the viewer, gradually only the higher imp points are selected. This view frustum, a 3D polyhedral object, can be selected quickly from the blocked and indexed vario-scale point cloud data (either at block level or at exact overlap-level). Visualizations using these selected points do not have the density shocks that discrete LoD based data pyramids have. In case of a moving the view position, the delta point sets, can be selected efficiently (and transferred from server to client): *new points* are selected based on 1. overlap with new vario-scale view frustum and 2. non-overlap with the previous vario-scale frustum (as visualization client already has these points). And for *old points* to be dropped in visualization: 1. overlap with previous vario-scale view frustum and 2. non-overlap with new vario-scale view frustum.

The data pyramid is not only relevant for the data storage (and query), but also relevant in the context of web-services. A specific web-services protocol for point cloud data is investigated, supporting progressive transfer based on multi-scale resolution (or vario-scale resolutions) as point clouds data sets can be massive and users might first want to have a good ‘overview' or users what good perspective views.

7. Possible standardization needed/ useful?

As described, we have been testing various solutions for managing point clouds, all having a different interface (and different possibilities). This is not only inconvenient for benchmarking, but also for normal application development (esp. when combining software components from different origin). More and more actors are involved and provide (partial) solutions for point cloud data processing: data acquisitions, storage/ management, visualization, analysis, etc. There are both commercial sector and open source based solutions at the various levels: Oracle, Esri, Bentley, LAStools, Postgres, MonetDB, and the list is much longer. The end-users, from industry, government, or research, want interoperability and be able to combine point cloud data from various sources with functionality in tools form other vendors/ developers, or even own developed specific purpose applications. When agreed by the community that standardizing point clouds is important, the next questions are related to what functionality need to be standardized and at what level. At least two closely related levels of standardization are to be considered: 1. Database SQL (Structure Query Language) extension for point clouds, and 2. Web Point Cloud Services (WPCS) for progressive transfer of point clouds.

As the focus is on geographic applications, these point cloud standards should fit and be complimentary to existing standards from the Open Geospatial Consortium (OGC) and/or the International Organisation for Standardisation/Technical Committee 211 (ISO TC211). Perhaps to the outside world, the web-services level is most important, but it is not useful to specify web-services functionality which can not be supported by lower data management and selection functionality. Point cloud data resemble both vector (object) and raster (field) data. One could argue that a point cloud could be stored as a standard vector data; e.g. in a Simple Feature Specification (SFS) multipoint in a single row or many rows with a single point. Theoretically true, but the size of the point clouds make them unsuitable to store them. A work-around could offer a pragmatic solution; e.g. store point cloud in several/ many (spatially coherent) multipoint objects (but this is non-trivial to realize and will put a serious work load on the user to realize this grouped multipoint storage). One could also argue that point cloud data is similar to raster data as it is based on sampling. True, but the result in case of point cloud data is not a regular grid, so the available raster data storage solutions are also unsuitable. Therefore, it is proposed to add a *third type of spatial representation* to our geographic information processing systems (and standards): a point cloud data type. Find below an initial list, which is describing the characteristic of a possible point cloud data type:

1. xyz (a lot, specify basic storage of coordinate in a SRS using various base data types: int, float, double, number, or perhaps even varchar..)
2. attributes per point (0 or more; e.g. intensity I, color RGB or classification, or observation point in addition to resulting target point…; this might conceptually correspond to a higher dimensional point),
3. fast access (based on spatial coherence) → some blocking scheme (in 2D, 3D, …)
4. space efficient storage → compression techniques (exploiting spatial cohesion)
5. data pyramid (LoD, multi-scale/vario-scale, perspective) support
6. temporal aspect, options for time per point (costly) or block (less refined)
7. query accuracies (report storage blocks, refines subsets of blocks with/without tolerance value of on 2D, 3D or nD query ranges or geometries)
8. operators, functionality in the following initially proposed categories, see Annex A
    a. loading, specify
    b. selections
    c. analysis I (not assuming 2D surface in 3D space)
    d. conversions (some assuming 2D surface in 3D space)
    e. towards reconstruction
    f. analysis II (some assuming a 2D surface in 3D space)
    g. LoD use/access
    h. updates
9. options to indicate the use of parallel processing for operations as listed in 8

Annex A. Details on functionality in the various categories

a. loading, specify:
   - input format
   - storage blocks based on which dimensions (2, 3, 4,…)
   - data pyramid, block dimensions (level: discrete or continuous)
   - compression option (none, lossless, lossy)
   - spatial clustering (morton, hilbert,…) within and between blocks
   - spatial indexing (rtree, quadtree) within and between blocks
   - validation (more format, e.g. no attributes omitted, than any geometry or topological validation; perhaps outlier detection)?

b. selections:
   - simple 2D range/rectangle filters (of various sizes)
   - selections based on points along a 2D polyline (with buffer)
   - selections of points overlapping a 2D polygon
   - spatial joint with other table; e.g. overlap point with polygons
   - select on address, postal code or on other textual geographic names (gazetteer)
   - selections based on the attributes such as intensity I (RGB, class)
   - spatiotemporal selections (space and time range),
   - combine multiple point clouds (Laser + MBES, classified + unclassified)

c. simple analysis I (not assuming 2D surface in 3D space):
   - (local) density of points
   - k-nearest neighbors
   - temporal differences computations
   - compute min/max/avg/median height (z or attribute value)
   - compute cross profiles (intersect with vertical plane, project 'nearby' points)

d. conversions (some assuming 2D surface in 3D space):
   - uncompressed to compression (lossless, lossy) and back
   - change blocksize (incl. optional data pyramid, multi/vario-scale support)
   - coordinate transformation (e.g. NL RD-NAP → ETRS89)
   - PC to contours,
   - PC to TIN,
   - PC to (height) raster/grid,
   - PC to image (e.g. oblique view on point cloud)
   - also opposite direction, generate point clouds from contours, TIN, raster
   - also higher dimensional objects/spaces

e. towards reconstruction:
   - compute normal vectors of points,
   - slope orientation or steepness computation
   - flat plane detection (and segmentation of point, add identifier attribute)
   - curved surface (cylinder, sphere patches, NURBS) detection
   - compute building (given) polygon height using point cloud (difference inside-outside)

f. advanced analysis (some assume a 2D surface in 3D space):
   - compute area of implied surface (by point cloud)
   - compute volume below surface
   - volume difference between design (3D polyhedral) surface and point could surface
   - detect break line in point cloud surface
   - hill shading relief (generate image)
   - view shed analysis

-

g. LoD use/access
    - multi-resolution/LoD selection (select top 1%, or top 0.01% of points, or key points) or aggregate new points,
    - sort points on relevance/importance (support streaming)
    - selection based on perspective view point cloud density: at view position M points per m^3 and (linear) reducing to 0 points per m^3 at distance d
    - delta selection of query E after moving to next view position (give additional points were densification is needed due to closer view range)

h. updates
    - update point geometry, some small changes of many points,
    - update point geometry, larger changes of few points,
    - update attribute (e.g. classify point)
    - insert new data of same type (0.1%, 1%, 5%, 10%, 25%,…)
    - delete points (0.1%, 1%, 5%, 10%, 25%,…)