Point cloud data management

13-11-2014

Peter van Oosterom, joint work with Oscar Martinez Rubi (NLeSc), Theo Tijssen (TUD), Martin Kodde (Fugro), Mike Horhammer (Oracle) and Milena Ivanova (NLeSc)

National eScience Symposium - Enabling Scientific Breakthroughs
Stadsschouwburg Almere, 6 November 2014
Content overview

0. Background
1. Conceptual benchmark
2. Executable benchmark
3. Data organization
4. Conclusion
2 years NL eScience Point cloud project

- **TU Delft:**
  1. GIS technology
  2. TU Delft, Library, contact with research & education users, dissemination & disclosure of point cloud data
  3. 3TU.Datacentrum, Long-term provision of ICT-infra
  4. TU Delft Shared Service Center ICT, storage facilities

- **NL eScience Center,** designing and building ICT infrastructure

- Oracle spatial, New England Development Centre (USA), improving existing software
- Rijkswaterstaat, data owner (and in-house applications)
- Fugro, point cloud data producer
- in practice also: CWI, MonetDB group
User requirements

• report user requirements, based on structured interviews conducted last year with
  • Government community: RWS (Ministry)
  • Commercial community: Fugro (company)
  • Scientific community: TU Delft Library

• report at MPC public website http://pointclouds.nl

• basis for conceptual benchmark, with tests for functionality, classified by importance (based on user requirements and Oracle experience)
Applications, often related to the environment

• examples:
  • flood modeling,
  • dike monitoring,
  • forest mapping,
  • generation of 3D city models, etc.

• it is expected that AHN3 will feature an even higher point density (as already in use at some today; e.g. Rotterdam)

• because of a lack of (processing) tools, most of these datasets are not being used to their full potential (e.g. first convert 0.5m grid or 5m grid, the data is losing potentially significant detail)
  → sitting on a gold mine, but not exploiting it!
Approach

- develop infrastructure for the storage, the management, ... of massive point clouds (note: no object reconstruction)

- support range of hardware platforms: normal/ department servers (HP), cloud-based solution (MS Azure), EXADATA (Oracle)

- scalable solution: if data sets becomes 100 times larger and/or if we get 1000 times more users (queries), it should be possible to configure based on same architecture

- generic, i.e. also support other (geo-)data and standards based, if non-existent, then propose new standard to ISO (TC211/OGC): Web Point Cloud Service (WPCS)

- also standardization at SQL level (SQL/SFS, SQL/raster, SQL/PC)?
Why a DBMS approach?

- today’s common practice: specific file format (LAS, LAZ, ZLAS,...) with specific tools (libraries) for that format

- point clouds are a bit similar to raster data: sampling nature, huge volumes, relatively static

- specific files are sub-optimal data management:
  - multi-user (access and some update)
  - scalability (not nice to process 60,000 AHN2 files)
  - integrate data (types: vector, raster, admin)

- ‘work around’ could be developed, but that’s building own DBMS
- no reason why point cloud can not be supported efficient in DBMS
- perhaps ‘mix’ of both: use file (or GPU) format for the PC blocks
Content overview

0. Background
1. Conceptual benchmark
2. Executable benchmark
3. Data organization
4. Conclusion
Benchmark organization

- **mini-benchmark**, small subset of data
  (20 million = 20,000,000) + limited functionality
  → get experience with benchmarking, platforms
  → first setting for tuning parameters: block size, compression.

- **medium-benchmark**, larger subset
  (20 billion = 20,000,000,000) + more functionality
  → more serious testing, first feeling for scalability
  → more and different types of queries (e.g. nearest neighbour)

- **full-benchmark**, full AHN2 data set
  (640 billion = 640,000,000,000) + yet more functionality
  → LoD (multi-scale), multi-user test

- **scaled-up benchmark**, replicated data set
  (20 trillion = 20,000,000,000,000) → stress test
Test data: AHN2 (subsets)

<table>
<thead>
<tr>
<th>Name</th>
<th>Points</th>
<th>LAS files</th>
<th>Disk size [GB]</th>
<th>Area [km²]</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>20M</td>
<td>20,165,862</td>
<td>1</td>
<td>0.4</td>
<td>1.25</td>
<td>TU Delft campus</td>
</tr>
<tr>
<td>210M</td>
<td>210,631,597</td>
<td>16</td>
<td>4.0</td>
<td>11.25</td>
<td>Major part of Delft city</td>
</tr>
<tr>
<td>2201M</td>
<td>2,201,135,689</td>
<td>153</td>
<td>42.0</td>
<td>125</td>
<td>City of Delft and surroundings</td>
</tr>
<tr>
<td>23090M</td>
<td>23,090,482,455</td>
<td>1,492</td>
<td>440.4</td>
<td>2,000</td>
<td>Major part of Zuid-Holland province</td>
</tr>
<tr>
<td>639478M</td>
<td>639,478,217,460</td>
<td>60,185</td>
<td>11,644.4</td>
<td>40,000</td>
<td>The Netherlands</td>
</tr>
</tbody>
</table>
E. SQL Query types/functionality

1. simple range/rectangle filters (of various sizes) → 10
2. selections based on points along a linear route (with buffer) → 8
3. selections of points overlapping a 2D polygon → 9
4. selections based on the attributes such as intensity I (/RGB) → 8
5. multi-resolution/LoD selection (select top x%) → 8, compute imp
6. sort points on relevance/importance (support streaming) → 7
7. slope orientation or steepness computation → 3
8. compute normal vector of selected points → 4
9. convert point cloud to TIN representation → 5
10. convert point cloud to Grid (DEM) → 6
11. convert point cloud to contours → 4
12. k-nearest neighbor selection (approx or exact) → 8
13. selection based on point cloud density → 2
14. spatial join with other table; e.g. 100 building polygons → 9
15. spatiotemporal selection queries (specify space+time range) → 8
16. temporal differences computations and selection → 6
17. compute min/max/avg/median height in 2D/3D area → 8
E. SQL Query types/functionality

18. hill shading relief (image based on point cloud/DEM/TIN) → 5
19. view shed analysis (directly on point cloud with fat points) → 5
20. flat plane detection (and segmentation point, add plane_id) → 5
21. curved surface detection (cylinder, sphere patches, freeform) → 4
22. compute area of implied surface (by point cloud) → 3
23. compute volume below surface → 5
24. select on address/postal code/geographic names (gazetteer) → 7
25. coordinate transformation RD-NAP - ETRS89 → 7
26. compute building height using point cloud (diff in/outside) → 8
27. compute cross profiles (intersect with vertical plane) → 8
28. combine multiple point clouds (Laser+MBES) → 6
29. volume difference between design (3D polyhedral) surface and point could → 4
30. detect break line point cloud surface → 1
31. selection based on perspective view point cloud density → 7
32. delta selection of query 31, moving to new position → 6
HP DL380p Gen8

‘Normal’ server hardware configuration:

- **HP DL380p Gen8 server**
  1. 2 x 8-core Intel Xeon processors (32 threads), E5-2690 at 2.9 GHz
  2. 128 GB main memory (DDR3)
  3. RHEL 6.5 operating system

- **Disk storage – direct attached**
  1. 400 GB SSD (internal)
  2. 6 TB SAS 15K rpm in RAID 5 configuration (internal)
  3. 2 x 41 TB SATA 7200 rpm in RAID-5 configuration (external in 4U rack 'Yotta-III' box, 24 disks)
Exadata X4-2: Oracle SUN hardware for Oracle database software

- database Grid: multiple Intel cores, computations
  Eight, quarter, half, full rack with resp. 24, 48, 96, 192 cores
- storage Servers: multiple Intel cores, massive parallel smart scans
  (predicate filtering, less data transfer, better performance)
- hybrid columnar compression (HCC): query and archive modes
Content overview

0. Background
1. Conceptual benchmark
2. Executable benchmark
3. Data organization
4. Conclusion
First executable mini-benchmark

- load small AHN2 dataset (one of the 60,000 LAS files) in:
  1. Oracle PointCloud
  2. Oracle flat (1 x,y,x attribute per row, btree index on x,y)
  3. PostgreSQL PointCloud
  4. PostgreSQL flat (1 2D point + z attribute per row, spatial index)
  5. MonetDB flat (1 x,y,x attribute per row, no index)
  6. LASTools (file, no database, tools from rapidlasso, Martin Isenburg)

- no compression, PC block size 5000, one thread, xyz only
- input 20,165,862 XYZ points (LAS 385 Mb, LAZ 37Mb)
Oracle 12c PointCloud (SDO_PC)

- point cloud metadata in SDO_PC object
- point cloud data in SDO_PC_BLK object (block in BLOB)
- loading: text file X,Y,Z,... using bulk loader (from LAS files) and use function SDO_PC_PKG.INIT and SDO_PC_PKG.CREATE_PC procedure (time consuming)
- block size 5000 points
- various compression options (initially not used)

- no white areas
- overlapping scans
- 4037 blocks:
  - 4021 with 5000 points
  - some with 4982-4999 points
  - some others with 2501-2502 points
PostgreSQL PointCloud

- use PointCloud extension by Paul Ramsey
  [github.com/pramsey/pointcloud](https://github.com/pramsey/pointcloud)
- also PostGIS extension (query)
- loading LAS(Z) with PDAL pcipeline
- block size 5000 points
- spatial GIST index for the blocks

- white areas
- overlapping scans
- 4034 blocks:
  - 3930 blocks with 4999 points
  - 104 blocks with 4998
MonetDB

- MonetDB: open source column-oriented DBMS developed by Centrum Wiskunde & Informatica (CWI), the Netherlands

- MonetDB/GIS: OGC simple feature extension to MonetDB/SQL

- MonetDB has plans for support point cloud data (and nD array’s)

- for comparing with Oracle and PostgreSQL only simple rectangle and circle queries Q1-Q4 (without conversion to spatial)

- no need to specify index (will be created invisibly when needed by first query...)
LASTools

- programming API LASlib (with LASzip DLL) that implements reading and writing LiDAR points from/to ASPRS LAS format ([http://lastools.org/](http://lastools.org/) or [http://rapidlasso.com/](http://rapidlasso.com/))
- LAStools: collection of tools for processing LAS or LAZ files; e.g. lassort.exe (z-orders), lasclip.exe (clip with polygon), lasthin.exe (thinning), las2tin.exe (triangulate into TIN), las2dem.exe (rasterizes into DEM), las2iso.exe (contouring), lasview.exe (OpenGL viewer), lasindex.exe (index for speed-up),...

- command: `lasindex [LAS File path]`  
  create LAX file per LAS file with spatial indexing info
- some tools only work in Windows, for Linux Wine ([http://www.winehq.org](http://www.winehq.org))
- note: file base solution, inefficient for large number of files; AHN2 data sets consists of over 60.000 LAZ (and LAX) files
Query geometries (mini-benchmark)

1. small rectangle, axis aligned, 51 x 53 m
2. large rectangle, axis aligned, 222 x 223 m
3. small circle at (85365 446594), radius 20 m*
4. large circle at (85759 447028), radius 115 m*
5. simple polygon, 9 points
6. complex polygon, 792 points, 1 hole
7. long narrow diagonal rectangle

(*) PostGIS not fully support CURVEPOLYGON
SQL Query syntax (geometry 1)

• PostgreSQL PointCloud: `CREATE TABLE query_res_1 AS SELECT PC_Explode(PC_Intersection(pa, geom))::geometry FROM patches pa, query_polygons WHERE pc_intersects(pa, geom) AND query_polygons.id = 1;`
  note, actually points have been converted to separate x,y,z values

• Oracle PointCloud: `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));`
  note SDO_PC_PKG.CLIP_PC function will return SDO_PC_BLK objects, actually have been converted via geometry (multipoint) with SDO_PC_PKG.TO_GEOMETRY function to separate x,y,z values

• LASTools: `lasclip.exe [LAZ File] -poly query1.shp -verbose -o query1.laz`
PC Block size and compression

- block size: 300, 500, 1000, 3000 and 5000 points
- compression:
  - Oracle PC: none, medium and high
  - PostGIS PC: none, dimensional

conclusions (most the same for PostGIS, Oracle):
- Compression about factor 2 to 3 (not as good as LAZ/ZLAS: 10)
- Load time and storage size are linear to size datasets
- Query time not much different: data size / compression (max 10%)
- Oracle medium and high compression score equal
- Oracle load gets slow for small block size 300-500

see graphs next slides: PostGIS (Oracle very similar)
More data

- 20M: 20165862 points
  - 20 LAS files / 1 LAS file
  - 385 MB
  - 1 km x 1.25 km
- 210M: 210631597 points
  - 16 LAS files
  - 4018 MB
  - 3 km x 3.75 km
- 2201M: 2201135689 points
  - 153 LAS files
  - 41984 MB
  - 10 km x 12.5 km
- 23090M: 23090482455 points
  - 1492 LAS files / 12 LAS files
  - 440420 MB
  - 40 km x 50 km
  - 1/30 AHN2
From **mini-** to **medium**-benchmark: load (index) times and sizes

p=Postgres, o=Oracle, m=MonetDB, l=Lastools
f=flat model, b=blocked model
20, 210, 2201, 23090M = million of points

<table>
<thead>
<tr>
<th>Approach</th>
<th>Total</th>
<th>Init.</th>
<th>Load</th>
<th>Close</th>
<th>Size[MB]</th>
<th>Points</th>
<th>Points/s</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pf20M</td>
<td>44.07</td>
<td>2.00</td>
<td>13.86</td>
<td>28.21</td>
<td>1558.71</td>
<td>20,165,862</td>
<td>457,587</td>
</tr>
<tr>
<td>pf210M</td>
<td>771.63</td>
<td>2.09</td>
<td>71.44</td>
<td>698.10</td>
<td>16249.24</td>
<td>210,631,597</td>
<td>272,970</td>
</tr>
<tr>
<td>pf2201M</td>
<td>9897.37</td>
<td>1.15</td>
<td>722.91</td>
<td>9173.31</td>
<td>169776.13</td>
<td>2,201,135,689</td>
<td>222,396</td>
</tr>
<tr>
<td>pf23090M</td>
<td>95014.05</td>
<td>3.64</td>
<td>8745.90</td>
<td>86264.51</td>
<td>1780963.91</td>
<td>23,090,482,455</td>
<td>243,022</td>
</tr>
<tr>
<td>of20M</td>
<td>128.43</td>
<td>0.51</td>
<td>123.92</td>
<td>4.00</td>
<td>879.50</td>
<td>20165862</td>
<td>157018</td>
</tr>
<tr>
<td>of210M</td>
<td>275.72</td>
<td>0.32</td>
<td>230.01</td>
<td>45.39</td>
<td>9124.50</td>
<td>210,631,597</td>
<td>763,933</td>
</tr>
<tr>
<td>of2201M</td>
<td>1805.68</td>
<td>0.31</td>
<td>1228.81</td>
<td>576.56</td>
<td>95825.00</td>
<td>2,201,135,689</td>
<td>1,219,007</td>
</tr>
<tr>
<td>of23090M</td>
<td>15825.53</td>
<td>0.14</td>
<td>9226.37</td>
<td>6599.02</td>
<td>997062.50</td>
<td>23,090,482,455</td>
<td>1,459,065</td>
</tr>
<tr>
<td>mf20M</td>
<td>7.15</td>
<td>1.14</td>
<td>3.70</td>
<td>2.31</td>
<td>475.78</td>
<td>20,165,862</td>
<td>2,820,400</td>
</tr>
<tr>
<td>mf210M</td>
<td>29.15</td>
<td>1.13</td>
<td>16.63</td>
<td>11.39</td>
<td>4888.05</td>
<td>210,631,597</td>
<td>7,225,784</td>
</tr>
<tr>
<td>mf2201M</td>
<td>304.14</td>
<td>4.91</td>
<td>198.76</td>
<td>100.47</td>
<td>50661.30</td>
<td>2,201,135,689</td>
<td>7,237,245</td>
</tr>
<tr>
<td>mf23090M</td>
<td>8490.90</td>
<td>0.93</td>
<td>5466.99</td>
<td>3022.98</td>
<td>529448.70</td>
<td>23,090,482,455</td>
<td>2,719,439</td>
</tr>
<tr>
<td>pb20M</td>
<td>153.41</td>
<td>22.03</td>
<td>130.53</td>
<td>0.85</td>
<td>101.77</td>
<td>20,165,862</td>
<td>131,451</td>
</tr>
<tr>
<td>pb210M</td>
<td>129.14</td>
<td>0.81</td>
<td>121.00</td>
<td>7.33</td>
<td>1009.13</td>
<td>210,631,597</td>
<td>1,631,033</td>
</tr>
<tr>
<td>pb2201M</td>
<td>754.22</td>
<td>0.86</td>
<td>687.49</td>
<td>65.87</td>
<td>10245.61</td>
<td>2,201,135,689</td>
<td>2,918,427</td>
</tr>
<tr>
<td>pb23090M</td>
<td>12263.05</td>
<td>0.87</td>
<td>7450.10</td>
<td>4812.08</td>
<td>106781.48</td>
<td>23,090,482,455</td>
<td>1,882,931</td>
</tr>
<tr>
<td>ob20M</td>
<td>296.22</td>
<td>0.35</td>
<td>228.17</td>
<td>67.70</td>
<td>226.50</td>
<td>20,165,862</td>
<td>68,077</td>
</tr>
<tr>
<td>ob210M</td>
<td>1246.87</td>
<td>0.25</td>
<td>557.70</td>
<td>688.92</td>
<td>2244.50</td>
<td>210,631,597</td>
<td>168,928</td>
</tr>
<tr>
<td>ob2201M</td>
<td>16737.02</td>
<td>0.86</td>
<td>7613.39</td>
<td>9122.77</td>
<td>21220.50</td>
<td>2,201,135,723</td>
<td>131,513</td>
</tr>
<tr>
<td>ob23090M</td>
<td>192612.07</td>
<td>0.31</td>
<td>96148.06</td>
<td>96463.70</td>
<td>220085.50</td>
<td>23,090,482,953</td>
<td>119,881</td>
</tr>
<tr>
<td>lt20M</td>
<td>9.49</td>
<td>0.03</td>
<td>9.46</td>
<td>0.00</td>
<td>384.65</td>
<td>20,165,862</td>
<td>2,124,959</td>
</tr>
<tr>
<td>lt210M</td>
<td>30.07</td>
<td>0.02</td>
<td>28.68</td>
<td>1.37</td>
<td>4021.37</td>
<td>210,631,597</td>
<td>7,004,709</td>
</tr>
<tr>
<td>lt2201M</td>
<td>218.06</td>
<td>0.02</td>
<td>216.18</td>
<td>1.86</td>
<td>41992.78</td>
<td>2,201,135,689</td>
<td>10,094,174</td>
</tr>
<tr>
<td>lt23090M</td>
<td>2129.30</td>
<td>0.03</td>
<td>2116.64</td>
<td>12.63</td>
<td>44084.48</td>
<td>23,090,482,455</td>
<td>10,844,166</td>
</tr>
</tbody>
</table>
Queries: returned points + times
(note flat model: increasing times)

- Scalability flat model: an issue

<table>
<thead>
<tr>
<th>Approach</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>pf20M</td>
<td>74947</td>
<td>718131</td>
<td>34637</td>
<td>562919</td>
<td>182792</td>
<td>387134</td>
<td>45805</td>
<td>0.35</td>
<td>2.25</td>
<td>0.24</td>
<td>1.90</td>
<td>1.18</td>
<td>1.72</td>
<td>0.84</td>
</tr>
<tr>
<td>pf210M</td>
<td>74947</td>
<td>718131</td>
<td>34637</td>
<td>562919</td>
<td>182792</td>
<td>387135</td>
<td>45805</td>
<td>0.42</td>
<td>2.50</td>
<td>0.27</td>
<td>2.26</td>
<td>0.91</td>
<td>1.65</td>
<td>1.34</td>
</tr>
<tr>
<td>pf220M</td>
<td>74947</td>
<td>718131</td>
<td>34637</td>
<td>562919</td>
<td>182792</td>
<td>387135</td>
<td>45805</td>
<td>4.92</td>
<td>19.03</td>
<td>2.90</td>
<td>18.28</td>
<td>9.37</td>
<td>17.71</td>
<td>10.16</td>
</tr>
<tr>
<td>pf23090M</td>
<td>74947</td>
<td>718131</td>
<td>34637</td>
<td>562919</td>
<td>182792</td>
<td>387134</td>
<td>45805</td>
<td>5.17</td>
<td>18.02</td>
<td>3.32</td>
<td>17.75</td>
<td>9.42</td>
<td>15.46</td>
<td>13.50</td>
</tr>
<tr>
<td>of20M</td>
<td>74872</td>
<td>718021</td>
<td>34691</td>
<td>563037</td>
<td>182861</td>
<td>387145</td>
<td>45813</td>
<td>0.24</td>
<td>0.37</td>
<td>0.28</td>
<td>1.85</td>
<td>0.75</td>
<td>1.32</td>
<td>1.32</td>
</tr>
<tr>
<td>of210M</td>
<td>74872</td>
<td>718021</td>
<td>34691</td>
<td>563037</td>
<td>182861</td>
<td>387145</td>
<td>45813</td>
<td>0.45</td>
<td>0.58</td>
<td>0.52</td>
<td>1.27</td>
<td>1.12</td>
<td>1.47</td>
<td>1.79</td>
</tr>
<tr>
<td>of220M</td>
<td>74872</td>
<td>718021</td>
<td>34691</td>
<td>563037</td>
<td>182861</td>
<td>387145</td>
<td>45813</td>
<td>1.47</td>
<td>3.87</td>
<td>1.29</td>
<td>4.26</td>
<td>4.38</td>
<td>6.60</td>
<td>5.24</td>
</tr>
<tr>
<td>of23090M</td>
<td>74872</td>
<td>718021</td>
<td>34691</td>
<td>563037</td>
<td>182861</td>
<td>387145</td>
<td>45813</td>
<td>1.25</td>
<td>18.20</td>
<td>2.34</td>
<td>22.75</td>
<td>6.99</td>
<td>27.18</td>
<td>635.06</td>
</tr>
<tr>
<td>mf20M</td>
<td>74872</td>
<td>718021</td>
<td>34691</td>
<td>563037</td>
<td>182861</td>
<td>387134</td>
<td>45813</td>
<td>0.06</td>
<td>0.13</td>
<td>0.06</td>
<td>0.20</td>
<td>9.96</td>
<td>187.16</td>
<td>38.71</td>
</tr>
<tr>
<td>mf210M</td>
<td>74872</td>
<td>718021</td>
<td>34691</td>
<td>563037</td>
<td>182861</td>
<td>387135</td>
<td>45813</td>
<td>0.13</td>
<td>0.26</td>
<td>0.15</td>
<td>0.28</td>
<td>9.95</td>
<td>185.65</td>
<td>38.56</td>
</tr>
<tr>
<td>mf220M</td>
<td>74872</td>
<td>718021</td>
<td>34691</td>
<td>563037</td>
<td>182861</td>
<td>387135</td>
<td>45813</td>
<td>0.64</td>
<td>0.90</td>
<td>0.64</td>
<td>0.77</td>
<td>10.37</td>
<td>186.38</td>
<td>39.17</td>
</tr>
<tr>
<td>mf23090M</td>
<td>74872</td>
<td>718021</td>
<td>34691</td>
<td>563037</td>
<td>182861</td>
<td>387134</td>
<td>45813</td>
<td>7.21</td>
<td>16.74</td>
<td>9.70</td>
<td>9.88</td>
<td>17.94</td>
<td>198.51</td>
<td>43.96</td>
</tr>
<tr>
<td>pb20M</td>
<td>74947</td>
<td>718131</td>
<td>34697</td>
<td>563108</td>
<td>182930</td>
<td>387142</td>
<td>45821</td>
<td>0.32</td>
<td>2.14</td>
<td>0.20</td>
<td>1.69</td>
<td>0.61</td>
<td>1.72</td>
<td>0.41</td>
</tr>
<tr>
<td>pb210M</td>
<td>74947</td>
<td>718131</td>
<td>34697</td>
<td>563108</td>
<td>182930</td>
<td>387142</td>
<td>45821</td>
<td>0.32</td>
<td>2.15</td>
<td>0.20</td>
<td>1.65</td>
<td>0.64</td>
<td>1.62</td>
<td>0.46</td>
</tr>
<tr>
<td>pb220M</td>
<td>74947</td>
<td>718131</td>
<td>34697</td>
<td>563108</td>
<td>182930</td>
<td>387142</td>
<td>45821</td>
<td>0.31</td>
<td>2.19</td>
<td>0.21</td>
<td>1.67</td>
<td>0.67</td>
<td>1.63</td>
<td>0.41</td>
</tr>
<tr>
<td>pb23090M</td>
<td>74947</td>
<td>718131</td>
<td>34697</td>
<td>563108</td>
<td>182930</td>
<td>387142</td>
<td>45821</td>
<td>0.32</td>
<td>2.19</td>
<td>0.21</td>
<td>1.68</td>
<td>0.68</td>
<td>1.68</td>
<td>0.44</td>
</tr>
<tr>
<td>ob20M</td>
<td>74947</td>
<td>718131</td>
<td>34697</td>
<td>563110</td>
<td>182930</td>
<td>387145</td>
<td>45821</td>
<td>0.41</td>
<td>1.38</td>
<td>0.34</td>
<td>1.21</td>
<td>0.62</td>
<td>1.38</td>
<td>0.53</td>
</tr>
<tr>
<td>ob210M</td>
<td>74947</td>
<td>718131</td>
<td>34697</td>
<td>563110</td>
<td>182930</td>
<td>387145</td>
<td>45821</td>
<td>0.38</td>
<td>1.28</td>
<td>0.36</td>
<td>1.22</td>
<td>0.62</td>
<td>1.29</td>
<td>0.54</td>
</tr>
<tr>
<td>ob220M</td>
<td>74947</td>
<td>718131</td>
<td>34697</td>
<td>563110</td>
<td>182930</td>
<td>387145</td>
<td>45821</td>
<td>0.39</td>
<td>1.36</td>
<td>0.36</td>
<td>1.23</td>
<td>0.60</td>
<td>1.33</td>
<td>0.50</td>
</tr>
<tr>
<td>ob23090M</td>
<td>74947</td>
<td>718131</td>
<td>34697</td>
<td>563110</td>
<td>182930</td>
<td>387145</td>
<td>45821</td>
<td>0.40</td>
<td>1.30</td>
<td>0.34</td>
<td>1.21</td>
<td>0.60</td>
<td>1.40</td>
<td>0.53</td>
</tr>
<tr>
<td>lt20M</td>
<td>74840</td>
<td>717931</td>
<td>34695</td>
<td>563049</td>
<td>182849</td>
<td>460068</td>
<td>45834</td>
<td>0.04</td>
<td>0.12</td>
<td>0.03</td>
<td>0.09</td>
<td>0.66</td>
<td>1.48</td>
<td>0.51</td>
</tr>
<tr>
<td>lt210M</td>
<td>74840</td>
<td>717931</td>
<td>34695</td>
<td>563049</td>
<td>182849</td>
<td>460068</td>
<td>45834</td>
<td>0.06</td>
<td>0.14</td>
<td>0.05</td>
<td>0.14</td>
<td>0.67</td>
<td>1.51</td>
<td>0.51</td>
</tr>
<tr>
<td>lt220M</td>
<td>74840</td>
<td>717931</td>
<td>34695</td>
<td>563049</td>
<td>182849</td>
<td>460068</td>
<td>45834</td>
<td>0.06</td>
<td>0.12</td>
<td>0.05</td>
<td>0.14</td>
<td>0.70</td>
<td>1.51</td>
<td>0.51</td>
</tr>
<tr>
<td>lt23090M</td>
<td>74840</td>
<td>717931</td>
<td>34695</td>
<td>563049</td>
<td>182849</td>
<td>460068</td>
<td>45834</td>
<td>0.05</td>
<td>0.15</td>
<td>0.05</td>
<td>0.14</td>
<td>0.68</td>
<td>1.50</td>
<td>0.52</td>
</tr>
</tbody>
</table>
First Exadata test with AHN2 medium-benchmark

- Oracle SUN hardware uniquely engineered to work together with Oracle database software: ‘DBMS counterpart’ of GPU for graphics

- X4-2 Half Rack Exadata was shortly available (96 cores, 4 TB memory, 300 TB disk)

- Scripts prepared by Theo/Oscar, adapted and executed by Dan Geringer (Oracle)

- 11 LAS files loaded via CSV into Oracle (flat table) on Exadata (one LAS file was corrupt after transfer)

- No indices needed (and also no tuning done yet...)
EXADATA Oracle loading compared to HP Proliant DL380p, 2*8 cores

<table>
<thead>
<tr>
<th>Database</th>
<th>load (min)</th>
<th>storage (Gb)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ora flat</td>
<td>393</td>
<td>508</td>
</tr>
<tr>
<td>PG flat</td>
<td>1565</td>
<td>1780</td>
</tr>
<tr>
<td>Monet flat</td>
<td>110</td>
<td>528</td>
</tr>
<tr>
<td>Ora PC</td>
<td>2958</td>
<td>220 (fastest query)</td>
</tr>
<tr>
<td>PG PC</td>
<td>274</td>
<td>106</td>
</tr>
<tr>
<td>LAS files</td>
<td>11</td>
<td>440 (LAZ 44)</td>
</tr>
<tr>
<td>exa No Compr</td>
<td>8</td>
<td>525</td>
</tr>
<tr>
<td>exa Query Low</td>
<td>10</td>
<td>213</td>
</tr>
<tr>
<td>exa Query High</td>
<td>15</td>
<td>92 ←</td>
</tr>
<tr>
<td>exa Arch Low</td>
<td>19</td>
<td>91</td>
</tr>
<tr>
<td>exa Arch High</td>
<td>29</td>
<td>57</td>
</tr>
<tr>
<td>Query</td>
<td>exaQH (sec)</td>
<td>OraPC (sec)</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>1</td>
<td>0.18</td>
<td>0.74</td>
</tr>
<tr>
<td>2</td>
<td>0.35</td>
<td>5.25</td>
</tr>
<tr>
<td>3</td>
<td>0.49</td>
<td>0.55</td>
</tr>
<tr>
<td>4</td>
<td>0.72</td>
<td>9.66</td>
</tr>
<tr>
<td>5</td>
<td>0.66</td>
<td>1.59</td>
</tr>
<tr>
<td>6</td>
<td>0.67</td>
<td>12.67</td>
</tr>
<tr>
<td>7</td>
<td>0.46</td>
<td>0.99</td>
</tr>
<tr>
<td>8</td>
<td>2.44</td>
<td>25.41</td>
</tr>
<tr>
<td>9</td>
<td>1.43</td>
<td>18.62</td>
</tr>
<tr>
<td>10</td>
<td>0.16</td>
<td>0.59</td>
</tr>
<tr>
<td>11</td>
<td>0.16</td>
<td>0.69</td>
</tr>
<tr>
<td>12</td>
<td>0.30</td>
<td>21.28</td>
</tr>
<tr>
<td>13</td>
<td>52.09</td>
<td>762.93</td>
</tr>
<tr>
<td>14</td>
<td>46.66</td>
<td>77.94</td>
</tr>
<tr>
<td>15</td>
<td>1.56</td>
<td>26.76</td>
</tr>
</tbody>
</table>
## Full AHN2 benchmark: loading

<table>
<thead>
<tr>
<th>system</th>
<th>Total load time [hours]</th>
<th>Total size [TB]</th>
<th>#points</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAStools unlic.</td>
<td>22:54</td>
<td>12.181</td>
<td>638,609,393,087</td>
</tr>
<tr>
<td>LAStools lic</td>
<td>16:47</td>
<td>11.617</td>
<td>638,609,393,101</td>
</tr>
<tr>
<td>LAStools lic LAZ</td>
<td>15:48</td>
<td>0.973</td>
<td>638,609,393,101</td>
</tr>
<tr>
<td>Oracle Exadata</td>
<td>4:39</td>
<td>2.240</td>
<td>639,478,217,460</td>
</tr>
<tr>
<td>MonetDB</td>
<td>17:21</td>
<td>15.00</td>
<td>639,478,217,460</td>
</tr>
</tbody>
</table>
Content overview

0. Background
1. Conceptual benchmark
2. Executable benchmark
3. Data organization
4. Conclusion
Flat models do not seem scalable

- PC data type based approaches have near constant query response times (irrespective of data set size)

- Flat table based models seem to have a non-constant query time (rule: 10 times more data \(\rightarrow\) response 2-3 times slower again)

- Solution: better spatial data organization (also for flat tables).
Data organization

• how can a flat table be organized efficiently?

• how can the point cloud blocks be created efficiently? (with no assumption on data organization in input)

• answer: spatial clustering/coherence, e.g. quadtree/octree (as obtained by Morton or Hilbert space filling curves)
Some Space Filling Curves

space filling curve used for block/cell creation
ordering or numbering of cells in kD into 1D using bijective mapping

default of flat model

Row (first \( y \), then \( x \))

Hilbert

Peano
Construction of Morton Curve

- Morton or Peano or N-order (or Z-order)
  - recursively replace each vertex of basic curve with the previous order curve
  - alternative: bitwise interleaving
- also works in 3D (or nD)
3D Morton curve

illustrations from http://asgerhoedt.dk

2x2x2  4x4x4  8x8x8
Use Morton code

- two options, discussed/implemented so far:
  1. flat table model create b-tree index on Morton code
  2. walk the curve create point cloud blocks

- better flat table model:
  - not use the default heap-table, but an indexed organized table
    (issue with duplicate values → CTAS distinct)
  - no separate index structure needed → more compact, faster

- perhaps best (and also to be tested):
  - not x, y, z attributes, but just high-res Hilbert / Morton code
    (as x, y, z coordinates can be obtained from code)
Overlap_Codes (query_geometry, domain) → Morton_code_ranges

- Based on concepts of Region Quadtree & Quadcodes
- Works for any type of query geometry (point, polyline, polygon)
- Also works in 3D (Octree) and higher dimensions

- Quadcode 0: Morton range 0-15
- Quadcode 10: Morton range 16-19
- Quadcode 12: Morton range 24-27
- Quadcode 300: Morton range 48-48

(Morton code gaps resp. 0, 4, 20)

Note: SW=0, NW=1, SE=2, NE=3
Quadcells / ranges and queries

CREATE TABLE query_results_1 AS (
SELECT * FROM
(SELECT x,y,z FROM ahn_flat WHERE
(hm_code between 1341720113446912 and 1341720117641215) OR
(hm_code between 1341720126029824 and 1341720134418431) OR
(hm_code between 1341720310579200 and 1341720314773503) OR
(hm_code between 1341720474157056 and 1341720478351359) OR
(hm_code between 1341720482545664 and 1341720503517183) OR
(hm_code between 1341720671289344 and 1341720675483647) OR
(hm_code between 1341720679677952 and 1341720683872255)) a
WHERE (x between 85670.0 and 85721.0)
and (y between 446416.0 and 446469.0))
Use of Morton codes/ranges (PostgreSQL flat model example)

<table>
<thead>
<tr>
<th></th>
<th>Q1</th>
<th>Q4</th>
<th>Q7</th>
</tr>
</thead>
<tbody>
<tr>
<td>rect</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>circle</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>line</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20M</td>
<td>0.16</td>
<td>0.85</td>
<td>2.32</td>
</tr>
<tr>
<td>210M</td>
<td>0.38</td>
<td>1.80</td>
<td>3.65</td>
</tr>
<tr>
<td>2201M</td>
<td>0.93</td>
<td>4.18</td>
<td>7.11</td>
</tr>
<tr>
<td>23090M</td>
<td>3.14</td>
<td>14.54</td>
<td>21.44</td>
</tr>
</tbody>
</table>

(response in seconds (of hot/second query first query exact same pattern, but 3-10 times slower both for normal flat model and for Morton flat model))

<table>
<thead>
<tr>
<th></th>
<th>Q1</th>
<th>Q4</th>
<th>Q7</th>
</tr>
</thead>
<tbody>
<tr>
<td>rect</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>circle</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>line</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20M</td>
<td>0.15</td>
<td>0.56</td>
<td>0.82</td>
</tr>
<tr>
<td>210M</td>
<td>0.15</td>
<td>0.56</td>
<td>0.42</td>
</tr>
<tr>
<td>2201M</td>
<td>0.13</td>
<td>0.64</td>
<td>0.41</td>
</tr>
<tr>
<td>23090M</td>
<td>0.15</td>
<td>0.70</td>
<td>0.60</td>
</tr>
</tbody>
</table>
Content overview

0. Background
1. Conceptual benchmark
2. Executable benchmark
3. Data organization
4. Conclusion
Summary

- very innovative and risky project
- no solutions available today (big players active; e.g. Google with street view also collects point clouds, but has not be able to serve these data to users)
- intermediate results: significant steps forward (explicit requirements, benchmark, improved products,...)
- direct contact with developers: Oracle, but also MonetDB, PostgreSQL/PostGIS, lastools,...
- standardization: discussions started (ISO, OGC)
- concepts developed for Multi-/vario-scale point clouds (LoD’s, data pyramid)
- parallel query algorithms
Next Phase of project

- full and scaled-up benchmarking
- web-based viewer (WebGL, LoD-tiles, Fugro prototype)
- model for operational service (for University users)
- ambitious project plan, further increased:
  - MonetDB
  - Lastools (and Esri’s ZLAS format)
  - Patty project
  - Via Apia project
- more data management platforms (optional):
  - SpatialHadoop
  - MS Azure data intensive cloud (announced last week)/MS SQL server
  - GeolinQ (layered solution with bathymetric/hydrographics roots
- more data?
  - Cyclomedia images / areal photographs
  - very high density, prediction 35 trillion points for NL
  - more attributes (r,g,b) $\rightarrow$ 100 times more data than full AHN2
Future topics (beyond project)

- possible topics:
  - different types of hardware/software solutions for point cloud data management (e.g. SpatialHadoop, or lastools/Esri format tools)
  - next to multiple-LoD's (data pyramid), explore true vario-scale LoD's
  - advanced functionality (outside our current scope): surface/volume reconstruction, temporal difference queries, etc.
  - higher dimensional point clouds, storing, structuring point clouds as 4D, 5D, 6D, etc points (instead of 3D point with a number of attributes), explore advantages and disadvantages

- partners (Fugro, RWS or Oracle) most likely interested
- also interest form others (Cyclomedia, MonetDB)
Data pyramid (LoD/multi-scale)

- besides spatial clustering (blocks, space filling curves: Hilbert/Morton), another required technique to obtain sufficient performance is using **data pyramids** (Level of Detail/ Multi-scale)

- option is after every 4 points in cell move 5th point to parent cell (for 2D organization and every 9th point in case of 3D), recursively bottom-up filling the cell/blocks

- results in data pyramid (depending on input data distribution, some areas may reach higher levels than others → discrete number of levels (multi-scale)
Large number of points in response

- overview queries just want subset
- organize in data pyramid

2D schematic view, data blocks.... stretched over domain

- every next higher level, density $2^k$ times less ($2D \rightarrow 4$, $3D \rightarrow 8$)
Data pyramid/multi-scale

- SLC (Morton, Hilbert code) like approach will give compact and efficient storage
- allows fast spatial searching including LoD selection

- the further away from viewer the lesser points selected (i.e. the higher level blocks/points)

- drawbacks:
  1. discrete number of levels
  2. bottom-up filling, unbalanced top
  3. point random assigned to level

Perspective view query

More points
Medium points
Less points
Data pyramid alternatives

- not random points, but more characteristic points move up (more important), some analysis needed; e.g.:
  1. compute local data density → more dense less important
  2. compute local surface shape → more flat less important
  3. other criteria, data collection/application dependent (intensity)

  (combine into) one imp_value of point → better than random

- not bottom-up, but top-down population, make sure that top levels are always filled across complete domain (lower levels may not be completely filled)
Further improvements ... beyond discrete levels

- might result in artefacts when looking at perspective view image (possible 'see' blocks of different levels)

- also not optimal within block (near viewer perhaps not enough points, further from viewer perhaps too much points)

- would a true vario-scale option possible?

→ Vario-scale geo-info research at TU Delft
Vario-scale with polygonal data
Vario-scale for point cloud data

- lesson from vario-scale research: add one dimension to the geometry (2D data vario-scale represented by 3D geometry)
- apply this to point cloud data
  1. compute the imp value
  2. add this as dimension, either
     x,y,imp (z and others attributes) or
     x,y,z,imp (and others as attributes)
  3. Cluster/index the 3D or 4D point
     (Morton code can also be used in 4D)
  4. Define perspective view selections,
     view frustum with one more dimension:
     the further, the higher imp’s
Perspective view

Select upper blue tetrahedron (view_frust) from prism-part of vario-scale $x,y,imp$ point cloud data cube.
Normal view frustum selection and streaming based on importance

- View frustum selection

```sql
select hm_code
from pc_demo
where overlaps (point(hm_code), view_frust)
```

- Ordered on importance for streaming

```sql
select hm_code
from pc_demo
where overlaps (point(hm_code), view_frust)
order by imp desc;
```

(or distance from tilted plane)
Delta queries for moving and zoom in/out

- select and send new points:
  \[ \text{point in new}_{-}\text{frust and point not in old}_{-}\text{frust} \]

- find and drop old points:
  \[ \text{point in old}_{-}\text{frust and not in new}_{-}\text{frust} \]

- note this works form both
  1. changing view position \(x,y,z)\)
  2. zooming in or out (‘view from above’, imp-dimension)

- optional to work at point or block granularity
  (in selection and server-client communication)
Standardization discussion @ Oracle
Standardization of point clouds?

- ISO/OGC spatial data:
  - at abstract/generic level, 2 types of spatial representations: features and coverages
  - at next level (ADT level), 2 types: vector and raster, but perhaps points clouds should be added
  - at implementation/encoding level, many different formats (for all three data types)

- nD point cloud:
  - points in nD space and not per se limited to x,y,z (n ordinates of point which may also have m attributes)
  - make fit in future ISO 19107 (as ISO 19107 is under revision).
  - note: nD point clouds are very generic; e.g. also cover moving object point data: x,y,z,t (id) series.
Characteristics of possible standard point cloud data type

1. **xyz** (a lot, use SRS, various base data types: int, float, double,..)
2. attributes per point (e.g. intensity I, color RGB or classification, or imp or observation point-target point or...)
   → correspond conceptually to a higher dimensional point
3. fast access (spatial cohesion) → blocking scheme (in 2D, 3D, ...)
4. space efficient storage → compression (exploit spatial cohesion)
5. data pyramid (LoD, multi-scale/vario-scale, perspective) support
6. temporal aspect: time per point (costly) or block (less refined)
7. query accuracies (blocks, refines subsets blocks with/without tolerance value of on 2D, 3D or nD query ranges or geometries)
8. operators/functionality (next slides)
9. options to indicate use of parallel processing
8. Operators/functionality

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

(grouping of functionalities from user requirements)
8a. 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)?
8b. 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)

(and so on for the other categories of operators/functionality)
Standardization actions

• participate in ISO 19107 (spatial schema) revision-team
  → make sure nD point clouds are covered

• within OGC make proposal for point cloud DWG

• probably focus on webservices level
  → more support/ partners expected
Webservices

• there is a lot of overlap between WMS, WFS and WCS...

• proposed OGC point cloud DWG should explore if WCS is good start for point cloud services:
  • If so, then analyse if it needs extension
  • If not good starting point, consider a specific WPCS, web point cloud service standards (and perhaps further increase the overlapping family of WMS, WFS, WCS,...)