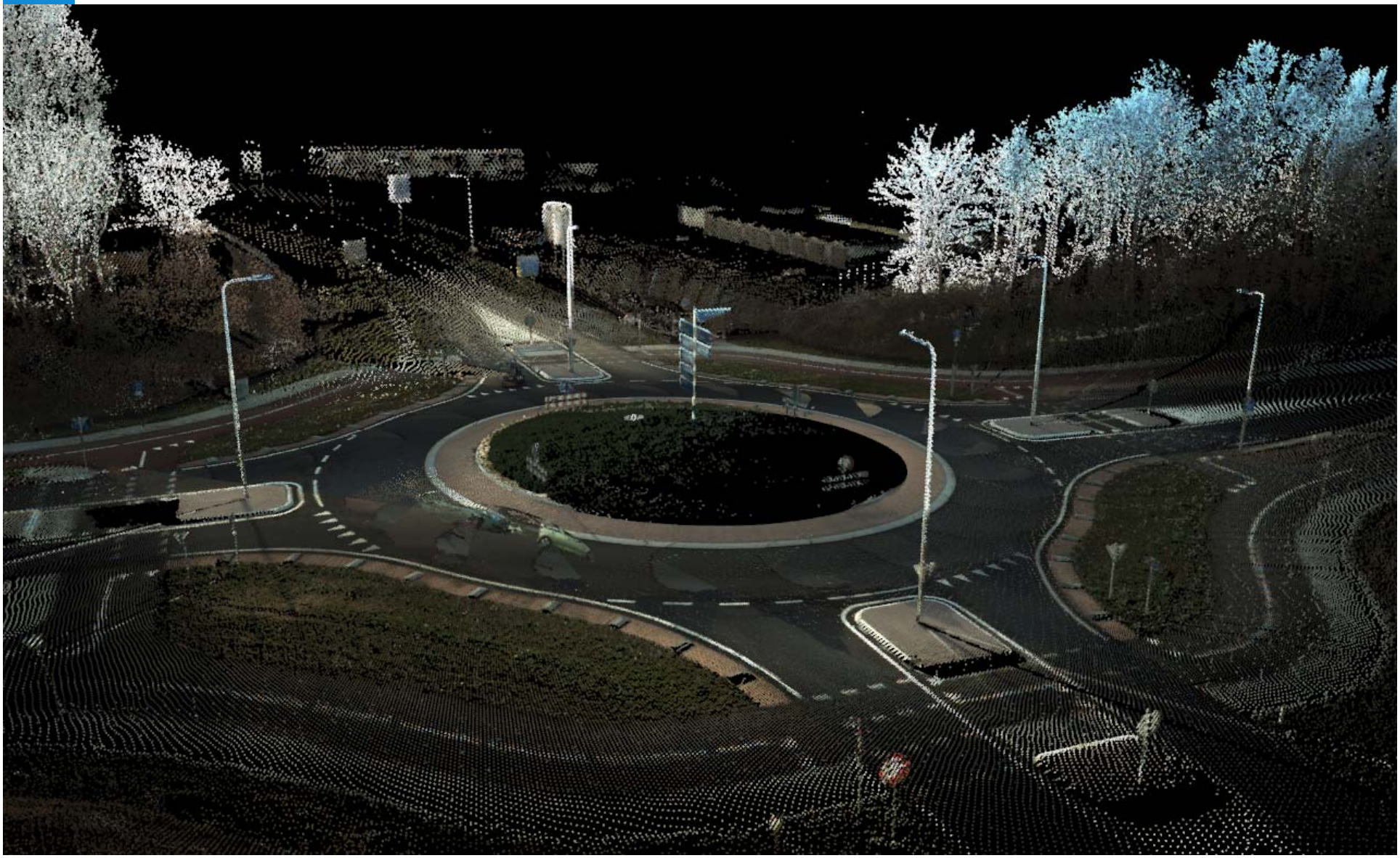
Point cloud data management

2-11-2014

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

Data Science Symposium, Friday, 31 October 2014



Content overview

0. Background
1. Conceptual benchmark
2. Executable benchmark
3. Data organization
4. Possible standardization
5. Conclusion

ORACLE SPATIAL

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)
- 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

Innovations needed

1. parallel ICT architecture solutions with well-balanced configurations (including HW, OS, DBMS)
2. new core support for point cloud data types in the existing spatial DBMS-besides existing vector and raster data types
3. specific web-services point cloud protocol (supporting streaming, progressive transfer based on Level of Detail (LoD)/multi-resolution representation in form of a data pyramid)
4. coherent point cloud blocks/pages based on spatial clustering & indexing of the blocks and potentially within the blocks
5. point cloud compression techniques (storage & transfer; e.g. compact 'local block' coordinates, binary encoding)
6. point cloud caching strategy at client (supporting panning and zooming of selected data)
7. exploit the GPUs (at client side) given the spatial nature of most GIS related computations
8. integrate & fine tuning above parts within the overall system

The 9 WPs of the project

1. Compile specifications/ requirements & Create Benchmarks (data, queries, etc.)
2. Design web-based interface, protocol between server and clients (WPCS supporting streaming data)
3. Create facility, server side (hardware: storage, processing, communication)
4. DBMS extension (with point cloud storage and analysis functionality)
5. Design database schema, load data, tune (spatial clustering/indexing)
6. Develop client side (pose queries, visualize results)
7. Execute Benchmark (incl. scaled up versions of data set)
8. Investigate operational model for facility after the project
9. Project management

Content overview

0. Background
1. Conceptual benchmark
2. Executable benchmark
3. Data organization
4. Possible standardization
5. Conclusion

Conceptual benchmark, specifies

platform independent specification of benchmark

A. datasets
   AHN2: 1.000.000 – 640.000.000.000, other with XYZRGB,...
B. query returns method
   count, create table X as select, local front-end, webservice,...
C. query accuracy level
   2D rectangle, 3D rectangle, 2D geometry, 3D geometry,...
D. number of parallel users
   1 – 10.000
E. query types (functionality)
   over 30 types classified on importance: from 0=low to 10=high
F. manipulations (updates)
   adding complete sets, deleting/adding individuals, changing

Benchmark organization

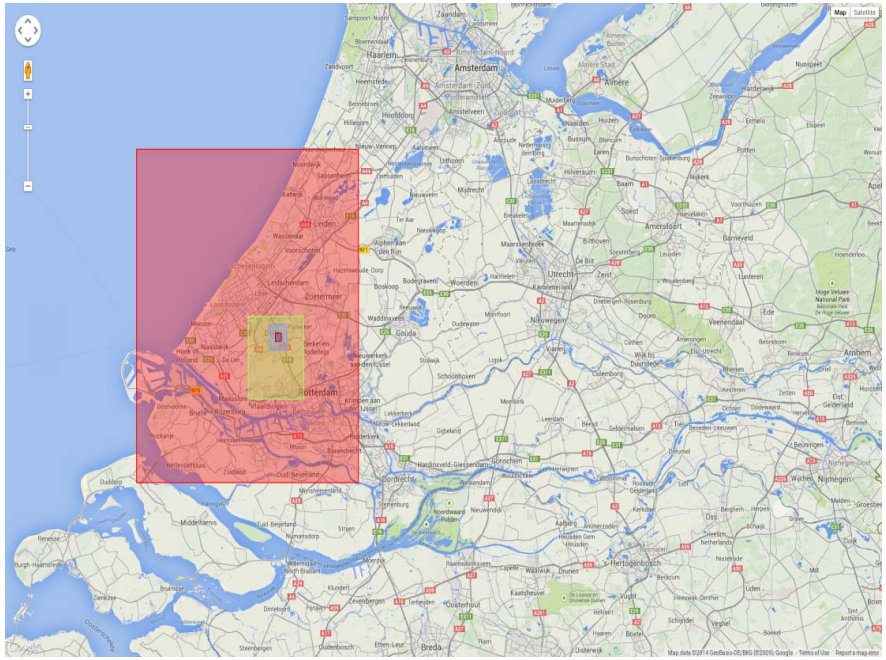
- mini-benchmark, small subset of data
  (20 million = 20.000.000) + limited functionality
  →get experience with benchmarking, platforms
  →first setting for tuning parameters: block size, compression.

- 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)



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

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

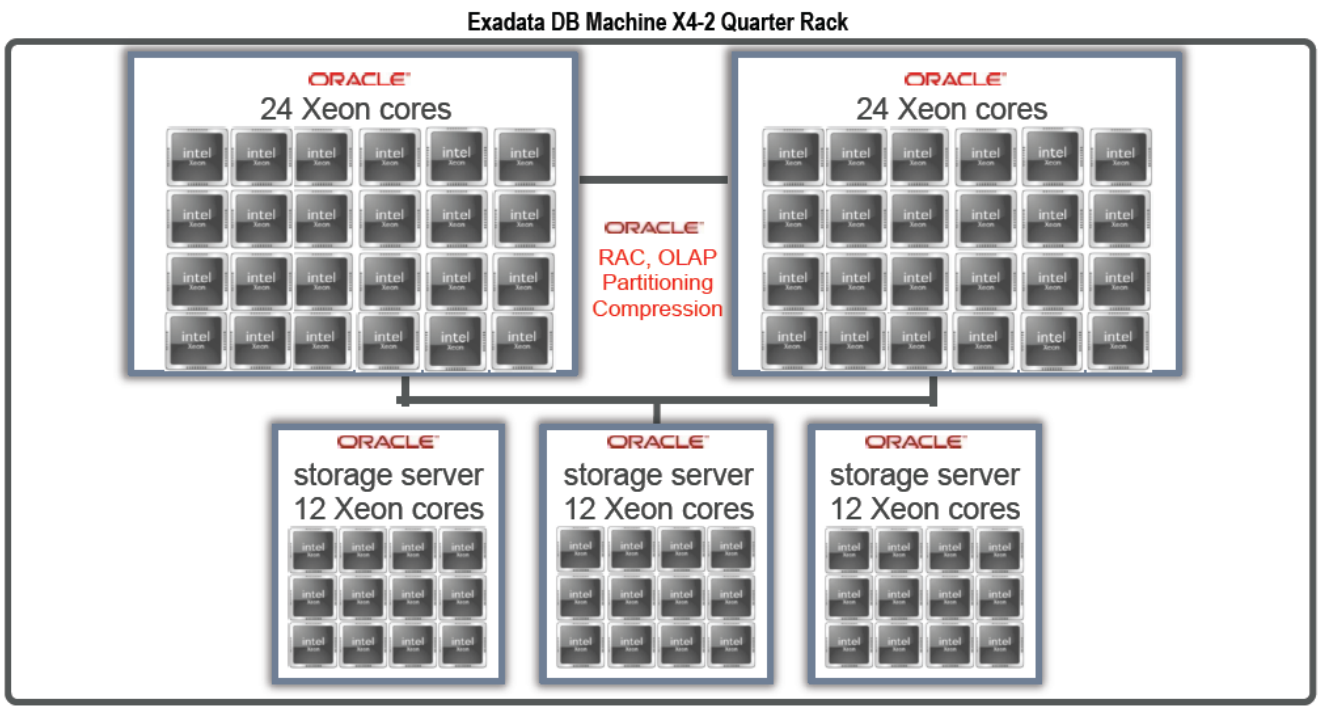
'Normal' server hardware configuration:

- 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

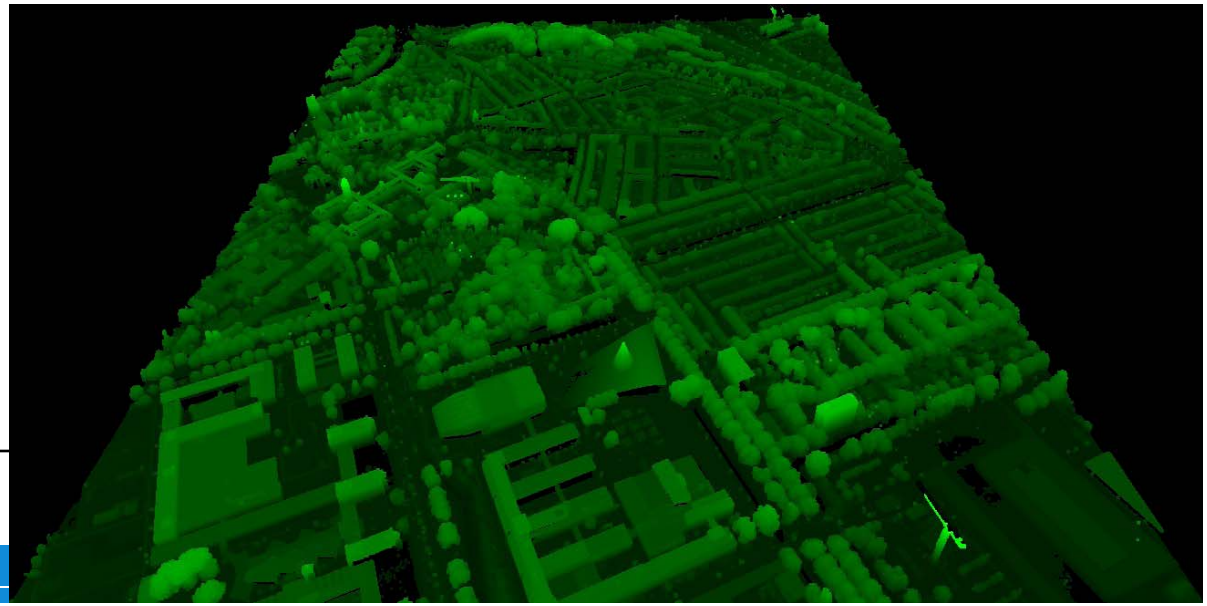


Content overview

0. Background
1. Conceptual benchmark
2. Executable benchmark
3. Data organization
4. Possible standardization
5. Conclusion

First executable mini-benchmark

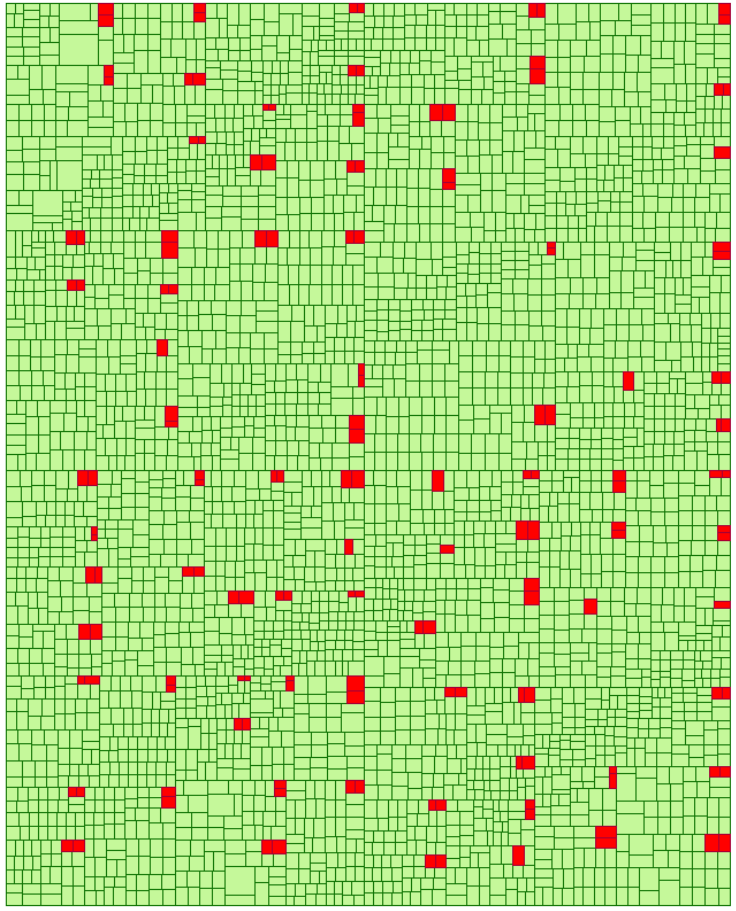
- load small AHN2 dataset (one of the 60.000 LAS files) in:
  1. Oracle PointCloud
  2. Oracle flat (1 x,y,x attribute per row, btree index on x,y)
  3. PostgreSQL PointCloud
  4. PostgreSQL flat (1 2D point + z attribute per row, spatial index)
  5. MonetDB flat (1 x,y,x attribute per row, no index)
  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)

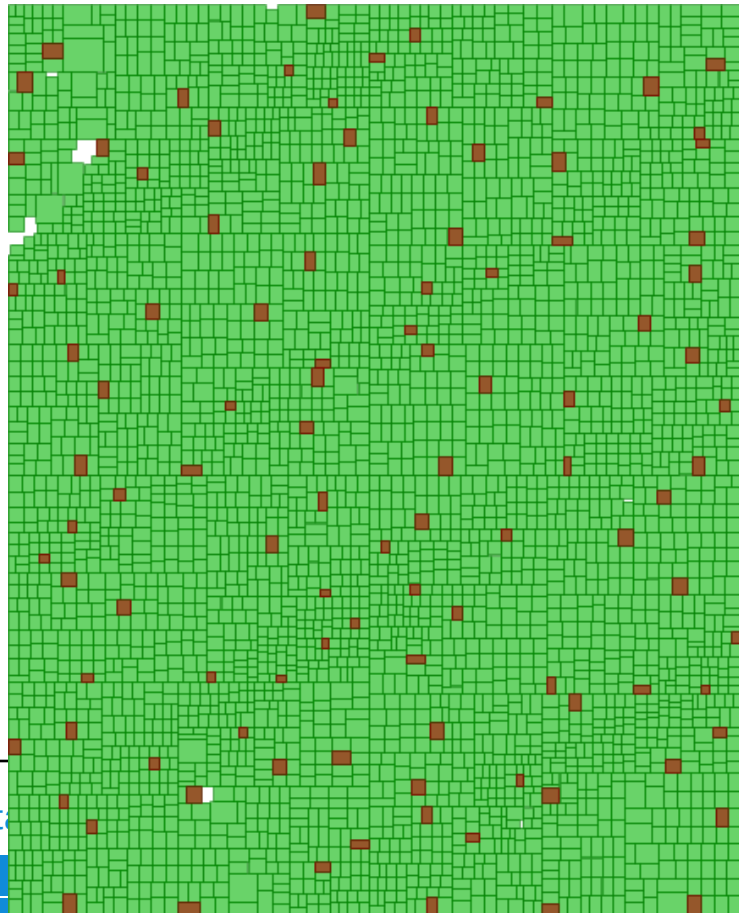
- no white areas
- overlapping scans
- 4037 blocks:
  - 4021 with 5000 points
  - some with 4982-4999 points
  - some others with 2501-2502 points



PostgreSQL PointCloud

- use PointCloud extension by Paul Ramsey
  https://github.com/pramsey/pointcloud
- also PostGIS extension (query)
- loading LAS(Z) with PDAL pcpipeline
- 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

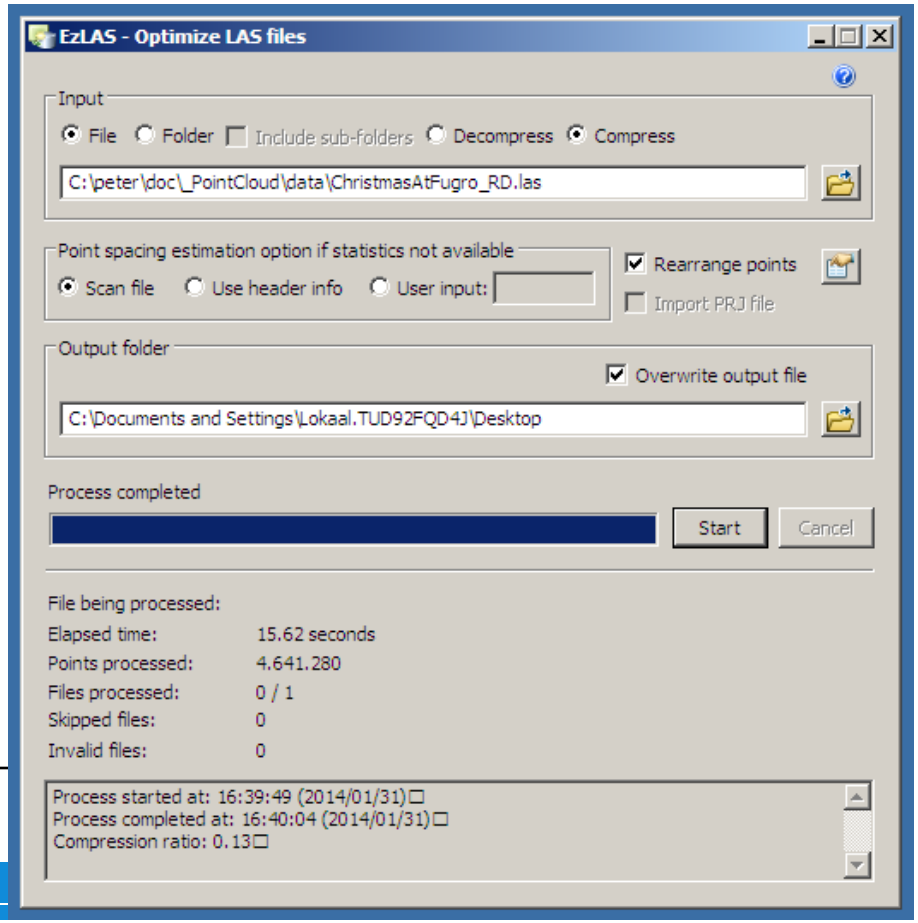
- programming API LASlib (with LASzip DLL) that implements reading and writing LiDAR points from/to ASPRS LAS format (http://lastools.org/ or http://rapidlasso.com/)
- LAStools: collection of tools for processing LAS or LAZ files; e.g. lassort.exe (z-orders), lasclip.exe (clip with polygon), lasthin.exe (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)
- note: file base solution, inefficient for large number of files; AHN2 data sets consists of over 60.000 LAZ (and LAX) files

rapidlasso

Esri's new LiDAR file format: ZLAS

- 6 January 2014, Esri LAS Optimizer/Compressor into ZLAS format http://www.arcgis.com/home/item.html?id=787794cdbd384261bc9bf99a860a374f
- standalone executable, not require ArcGIS
- same executable EzLAS.exe for compression and decompression



- compression a bit disappointing: from 385 Mb to 42 Mb, fact 9, compared to LAZ 36 Mb, fact 10
- perhaps the 'use' performance is better (in Esri tools)...

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**

Analysis from mini- to medium-benchmark

- Compression levels
- Different block sizes
- Larger datasets
- MonetDB, no point cloud type yet → flat table (not in initial plan)
- Oracle load bottle neck → patch developed for faster loading

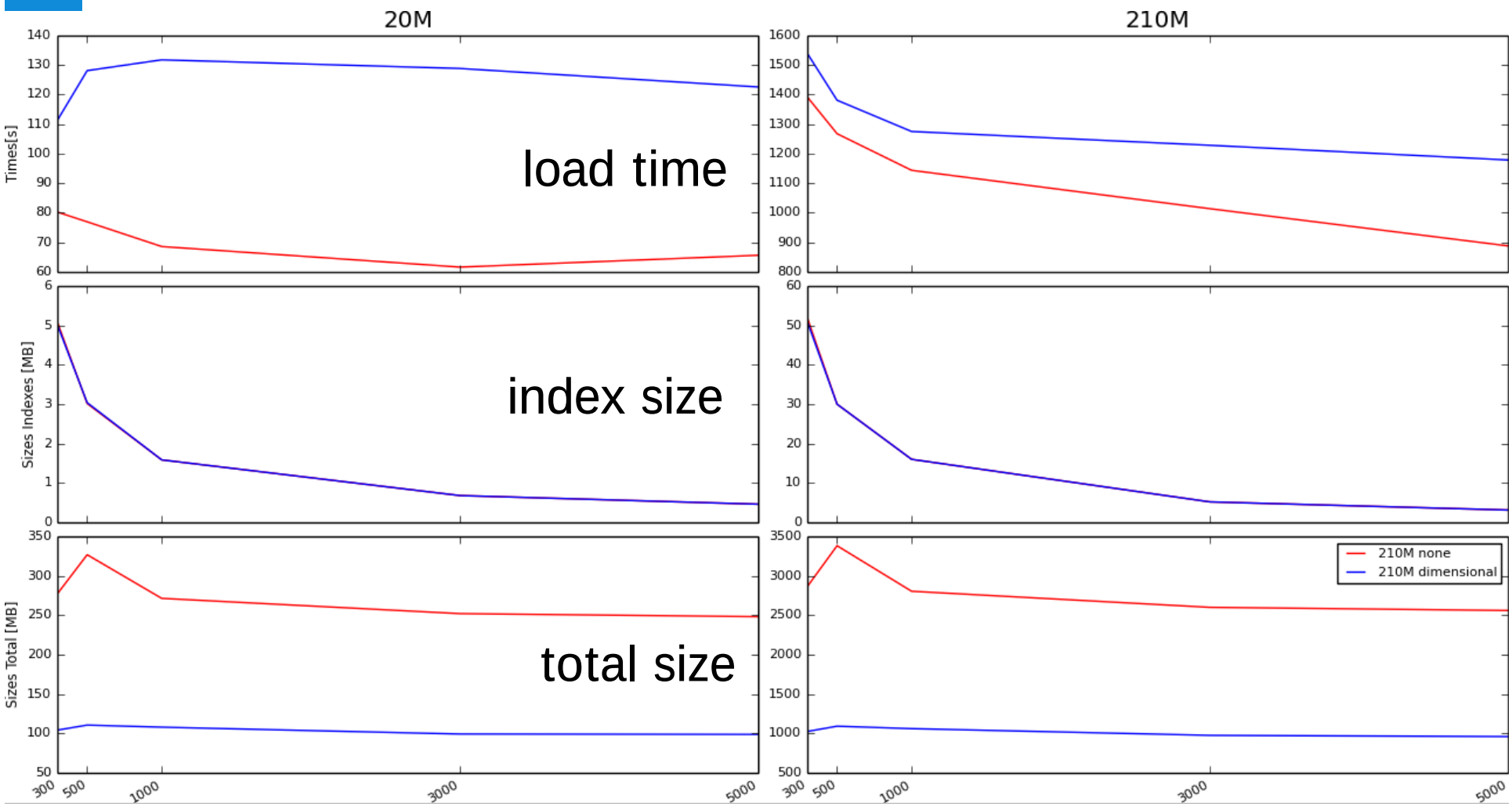
PC Block size and compression

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

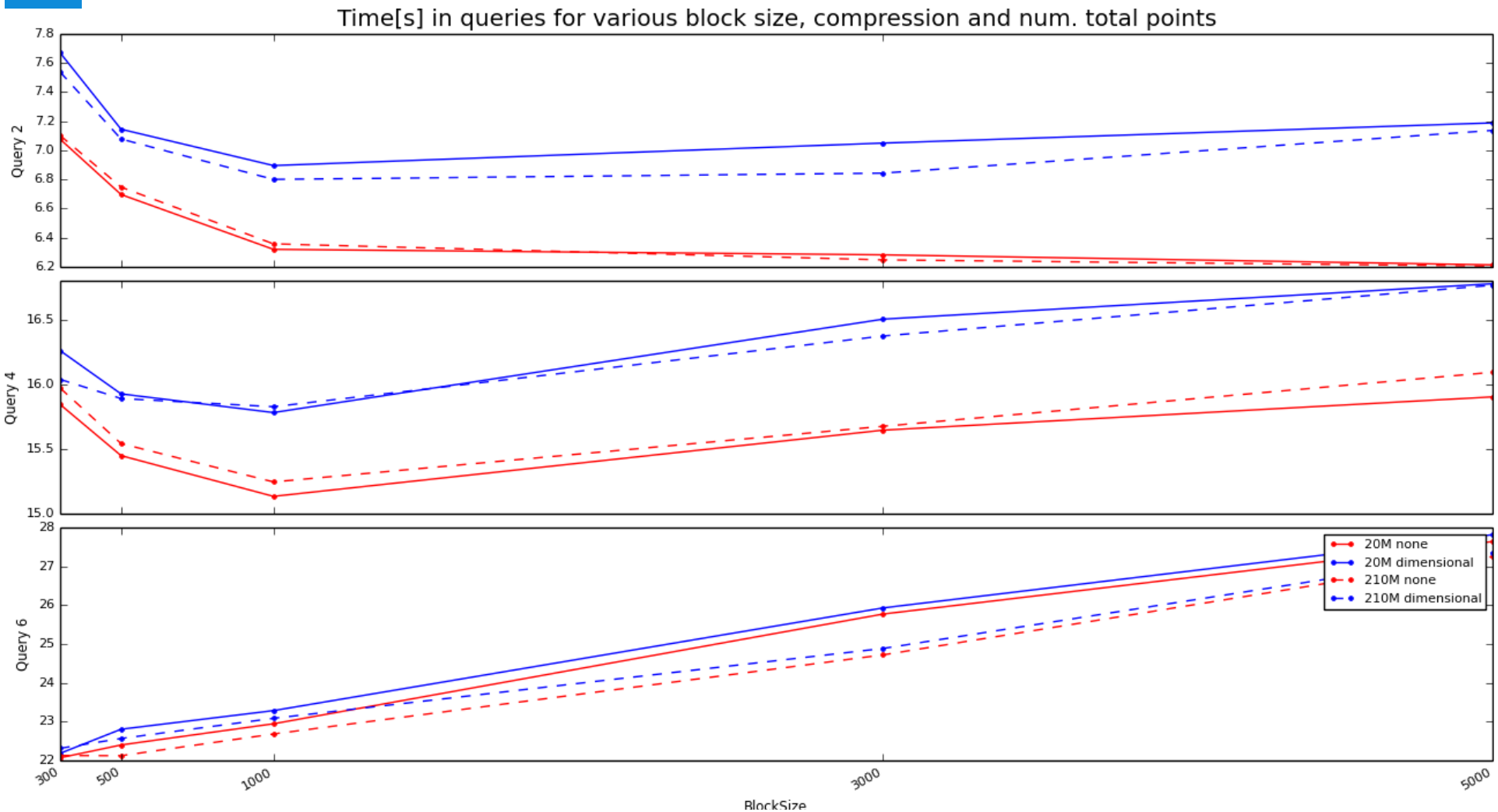
- conclusions (most the same for PostGIS, Oracle):
  - Compression about factor 2 to 3 (not as good as LAZ/ZLAS: 10)
  - Load time and storage size are linear to size datasets
  - Query time not much different: data size / compression (max 10%)
  - Oracle medium and high compression score equal
  - Oracle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

- see graphs next slides: PostGIS (Oracle very similar)

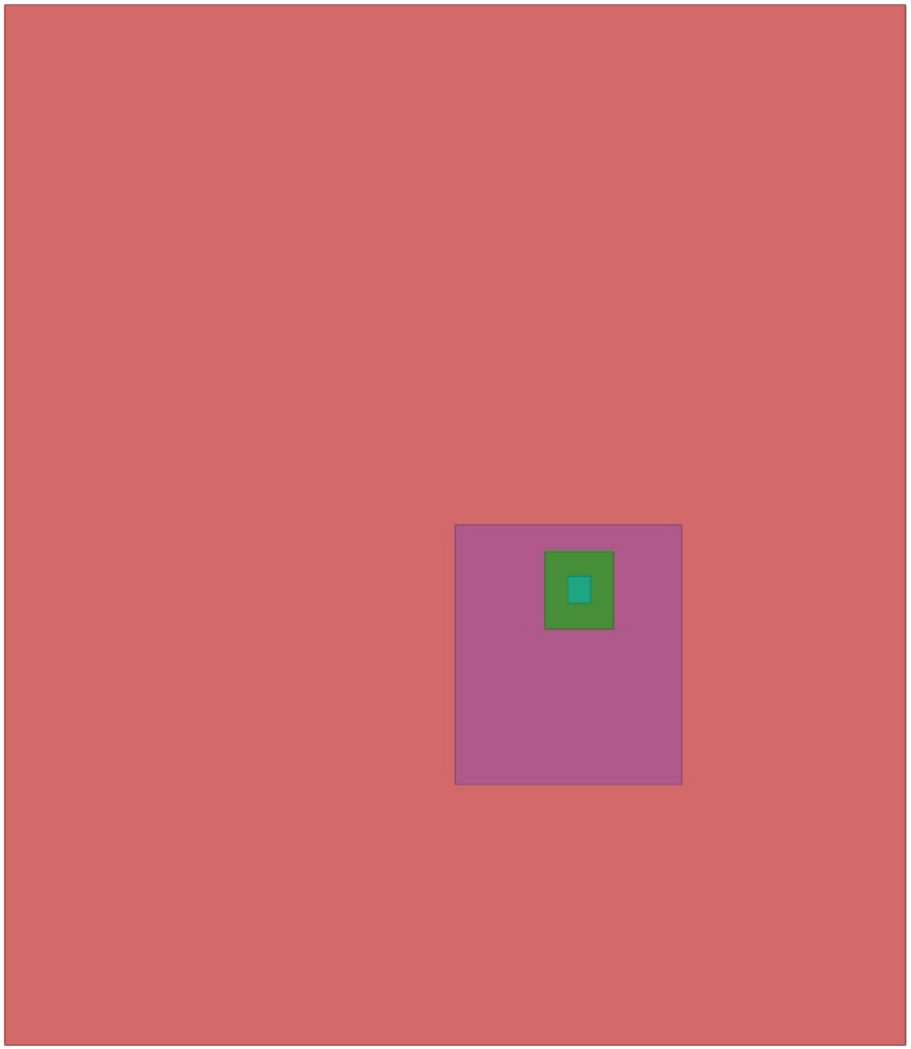
PostGIS loading, blocks 300 – 5000 (blue line is with compression)



PostGIS querying, blocks 300 – 5000 blue=compressed, solid=20M, small



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

Approach	Time[s]				Size[MB]		Points	Points/s
	Total	Init.	Load	Close	Total	Index		
pf20M	44.07	2.00	13.86	28.21	1558.71	551.83	20,165,862	457,587
pf210M	771.63	2.09	71.44	698.10	16249.24	5762.20	210,631,597	272,970
pf2201M	9897.37	1.15	722.91	9173.31	169776.13	60214.46	2,201,135,689	222,396
pf23090M	95014.05	3.64	8745.90	86264.51	1780963.91	631663.71	23,090,482,455	243,022
of20M	128.43	0.51	123.92	4.00	879.50	453.85	20165862	157018
of210M	275.72	0.32	230.01	45.39	9124.50	4739.94	210,6315,97	763,933
of2201M	1805.68	0.31	1228.81	576.56	95825.00	49533.74	2,201,135,689	1,219,007
of23090M	15825.53	0.14	9226.37	6599.02	997062.50	519621.48	23,090,482,455	1,459,065
mf20M	7.15	1.14	3.70	2.31	475.78	14.09	20,165,862	2,820,400
mf210M	29.15	1.13	16.63	11.39	4888.05	66.95	210,631,597	7,225,784
mf2201M	304.14	4.91	198.76	100.47	50661.30	281.17	2,201,135,689	7,237,245
mf23090M	8490.90	0.93	5466.99	3022.98	529448.70	949.37	23,090,482,455	2,719,439
pb20M	153.41	22.03	130.53	0.85	101.77	0.69	20,165,862	131,451
pb210M	129.14	0.81	121.00	7.33	1009.13	5.18	210,631,597	1,631,033
pb2201M	754.22	0.86	687.49	65.87	10245.61	53.05	2,201,135,689	2,918,427
pb23090M	12263.05	0.87	7450.10	4812.08	106781.48	552.77	23,090,482,455	1,882,931
ob20M	296.22	0.35	228.17	67.70	226.50	0.20	20,165,862	68,077
ob210M	1246.87	0.25	557.70	688.92	2244.50	1.44	210,631,597	168,928
ob2201M	16737.02	0.86	7613.39	9122.77	21220.50	13.25	2,201,135,723	131,513
ob23090M	192612.07	0.31	96148.06	96463.70	220085.50	165.55	23,090,482,953	119,881
lt20M	9.49	0.03	9.46	0.00	384.65	0.02	20,165,862	2,124,959
lt210M	30.07	0.02	28.68	1.37	4021.37	3.88	210,631,597	7,004,709
lt2201M	218.06	0.02	216.18	1.86	41992.78	9.40	2,201,135,689	10,094,174
lt23090M	2129.30	0.03	2116.64	12.63	440484.48	68.04	23,090,482,455	10,844,166
oe-nc	508.71				537600.00	0.00	21,090,482,455	41,458,753
oe-ql	619.14				218504.00	0.00	21,090,482,455	34,064,157
oe-qh	928.84				94240.00	0.00	21,090,482,455	22,706,269
oe-al	1149.55				93992.00	0.00	21,090,482,455	18,346,729
oe-ah	1767.53				59096.00	0.00	21,090,482,455	11,932,177

Queries: returned points + times (note flat model: increasing times)

Approach	Number of points							Time[s]						
	1	2	3	4	5	6	7	1	2	3	4	5	6	7
pf20M	74947	718131	34637	562919	182792	387134	45805	0.35	2.25	0.24	1.90	1.18	1.72	0.84
pf210M	74947	718131	34637	562919	182792	387135	45805	0.42	2.50	0.27	2.26	0.91	1.65	1.34
pf2201M	74947	718131	34637	562919	182792	387135	45805	4.92	19.03	2.90	18.28	9.37	17.71	10.16
pf23090M	74947	718131	34637	562919	182792	387134	45805	5.17	18.02	3.32	17.75	9.42	15.46	13.50
of20M	74872	718021	34691	563037	182861	387145	45813	0.24	0.37	0.28	1.85	0.75	1.32	1.32
of210M	74872	718021	34691	563037	182861	387145	45813	0.45	0.58	0.52	1.27	1.12	1.47	1.79
of2201M	74872	718021	34691	563037	182861	387145	45813	1.47	3.87	1.29	4.26	4.38	6.60	5.24
of23090M	74872	718021	34691	563037	182861	387145	45813	1.25	18.20	2.34	22.75	6.99	27.18	635.06
mf20M	74872	718021	34691	563037	182861	387134	45813	0.06	0.13	0.06	0.20	9.96	187.16	38.71
mf210M	74872	718021	34691	563037	182861	387135	45813	0.13	0.26	0.15	0.28	9.95	185.65	38.56
mf2201M	74872	718021	34691	563037	182861	387135	45813	0.64	0.90	0.64	0.77	10.37	186.38	39.17
mf23090M	74872	718021	34691	563037	182861	387134	45813	7.21	16.74	9.70	9.88	17.94	198.51	43.96
pb20M	74947	718131	34697	563108	182930	387142	45821	0.32	2.14	0.20	1.69	0.61	1.72	0.41
pb210M	74947	718131	34697	563108	182930	387142	45821	0.32	2.15	0.20	1.65	0.64	1.62	0.46
pb2201M	74947	718131	34697	563108	182930	387142	45821	0.31	2.19	0.21	1.67	0.67	1.63	0.41
pb23090M	74947	718131	34697	563108	182930	387142	45821	0.32	2.19	0.21	1.68	0.68	1.68	0.44
ob20M	74947	718131	34697	563110	182930	387145	45821	0.41	1.38	0.34	1.21	0.62	1.38	0.53
ob210M	74947	718131	34697	563110	182930	387145	45821	0.38	1.28	0.36	1.22	0.62	1.29	0.54
ob2201M	74947	718131	34697	563110	182930	387145	45821	0.39	1.36	0.36	1.23	0.60	1.33	0.50
ob23090M	74947	718131	34697	563110	182930	387145	45821	0.40	1.30	0.34	1.21	0.60	1.40	0.53
lt20M	74840	717931	34695	563049	182849	460068	45834	0.04	0.12	0.03	0.09	0.66	1.48	0.51
lt210M	74840	717931	34695	563049	182849	460068	45834	0.06	0.14	0.05	0.14	0.67	1.51	0.51
lt2201M	74840	717931	34695	563049	182849	460068	45834	0.06	0.12	0.05	0.14	0.70	1.51	0.51
lt23090M	74840	717931	34695	563049	182849	460068	45834	0.05	0.15	0.05	0.14	0.68	1.50	0.52
oe-qh	40368	369352	19105	290456	132307	173927	9559	0.18	0.35	0.59	0.72	0.66	0.67	0.46

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...)

Loading compared to alternatives on HP Proliant DL380p, 2*8 cores

Database	load (min)	storage (Gb)	
Ora flat	393	508	
PG flat	1565	1780	
Monet flat	110	528	
Ora PC	2958	220	(fastest query)
PG PC	274	106	
LAS files	11	440	(LAZ 44)
exa No Compr	8	525	
exa Query Low	10	213	
exa Query High	15	92	←
exa Arch Low	19	91	
exa Arch High	29	57	

Querying

Query	exaQH sec	OraPC sec	exaQH #	OraPC #
1	0.18	0.74	40368	74872
2	0.35	5.25	369352	718021
3	0.49	0.55	19105	34667
4	0.72	9.66	290456	563016
5	0.66	1.59	132307	182861
6	0.67	12.67	173927	387145
7	0.46	0.99	9559	45813
8	2.44	25.41	2273169	2273179
9	1.43	18.62	620582	620585
10	0.16	0.59	94	2433
11	0.16	0.69	591	591
12	0.30	21.28	342938	342938
13	52.09	762.93	793206	897926
14	46.66	77.94	765811	765844
15	1.56	26.76	2897425	3992023
16	0	0.06	0	0
17	2.21	19.53	2201058	2201060

Full AHN2 benchmark: loading

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

Full AHN2 benchmark: querying

	#points				Time[s]							
QUERY	LAStools unlic.	LAStools	Exadata	MonetDB	LAStools unlic. LAS DB	LAStools unlic. LAS No DB	LAStools LAS DB	LAStools LAS No DB	LAStools LAZ DB	LAStools LAZ No DB	Exadata	MonetDB
01_S_RECT	74861	74863	74863	74855	0,07	0,79	0,06	0,77	0,2	0,84	0,48	108,99
02_L_RECT	718057	718070	718070	718132	0,16	0,85	0,16	0,78	0,49	1,09	0,79	111,33
03_S_CIRC	34700	34699	34675	34699	0,07	0,78	0,06	0,7	0,27	0,89	1,22	109,36
04_L_CIRC	563119	563105	563082	563105	0,16	0,85	0,12	0,78	0,44	1,15	1,69	112,9
10_S_RECT_MAXZ	2413	1734	2434	2434	0,08	0,82	0,08	0,69	0,43	1,02	0,4	112,43
11_S_RECT_MINZ	591	591	591	591	0,05	0,84	0,05	0,72	0,13	0,73	0,44	111,31
12_L_RECT_MINZ	343168	343168	343171	343171	0,26	1	0,26	0,98	1,32	1,95	0,6	114,98
16_XL_RECT_EMPTY	0	0	0	0	0,04	0,75	0,04	0,66	0,03	0,69	0	113,51
18_NN_1000				1000								112,9
19_NN_5000				5000								107,94
20_NN_1000_river				0								109,2
05_S_SIMP_POLY	182871	182875	182875	182875	0,7	31,05	0,39	28,2	0,63	29,46	1,43	110,42
06_L_COMP_POLY_HOLE	460140	460158	387201	387193	1,52	32	1,57	30,2	1,97	30,31	1,29	112,17
07_DIAG_RECT	45831	45813	45815	45815	0,55	31,43	0,55	28,76	0,99	29,23	1,71	112,14
08_XL_POLYGON_2_HOLES	2365925	2365896	2273469	2273380	3,72	34,72	3,87	33,88	5,67	35,88	2,86	120,93
09_S_POLYLINE_BUFFER	620568	620527	620719	620529	2,34	33,44	2,27	32,4	3,94	33,43	1,58	118,71
15_XL_RECT	3992330	3992377	3992290	3992525	0,49	1,37	0,57	1,23	1,94	2,63	2,23	120,18
17_XL_CIRC	2203066	2203055	2201280	2203055	0,32	1,16	0,32	1,01	1,1	1,71	2,51	114,31
21_L_NARROW_RECT	382395	382335	382335	386278	2,23	4,43	2,38	4,11	16,94	18,81	0,95	109,2
27_STREET_DIAG_RECT	27443	27453	27459	27456	1,61	914,44	0,25	29,18	0,35	29,06	1,23	107,92
13_L_POLYLINE_BUFFER	897042	896927	897359	897126	412,29	804,29	266,13	958,88	317,91	1039,09	23,34	723,58
14_DIAG_POLYLINE_BUFFER	765989	766039	766029	766029	102,19	419,78	104,09	428,51	151,45	600,93	15,05	559,73
22_L_NARROW_DIAG_RECT	12148049	12147863	121478	022246.9	154,97	1115,2	141,95	1800,41	284,58	2478,47	113,37	2246,9
23_XL_NARROW_DIAG_RECT	691422551	6,91E+08	6,91E+08	151.16	379,2	886,35	318,93	1051,76	552,01	1154,7	330,85	151,16
24_L_NARROW_DIAG_RECT_2	2468239	2468268	2468367	2468367	251,13	4106,54	294,99	13454	551,13	16510,9	393,21	6308,53
25_PROV_DIAG_RECT		2,03E+09	3,53E+14	12228.76							1193,1	12228,76
26_MUNI_RECT	212416249 7	2,12E+09	2,12E+09	622.73	417,28	1127,56	214,98	405,26	536,1	560,95	25,79	622,73
28_VAALS	809097723	8,09E+08	8,67E+08	1469.47	1635,3	1883,19	2240,1	2525,98	2426,43	2718,26	67,67	1469,47
29_MONTFERLAND	150966261 5	1,51E+09	1,51E+09	5577.69	3579,73	5746,61	4809,9	6419,05	3855,93	6396,54	120,02	5577,69
30_WESTERVELD		2,9E+09	1,34E+14	27683.62							3569,54	27683,62

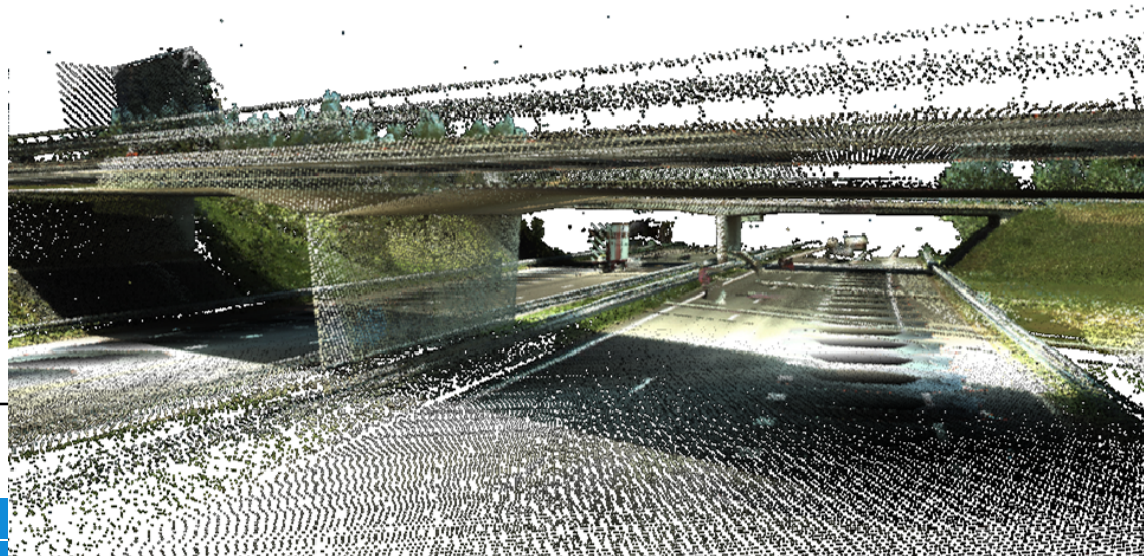
Content overview

0. Background
1. Conceptual benchmark
2. Executable benchmark
3. Data organization
4. Possible standardization
5. Conclusion

Rijkswaterstaat
Ministerie van Infrastructuur en Milieu

Flat models do not seam scalable

- PC data type based approaches have near constant query response times (irrespective of data set size)
- flat table based models seam to have a non-constant query time (rule: 10 times more data → 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)

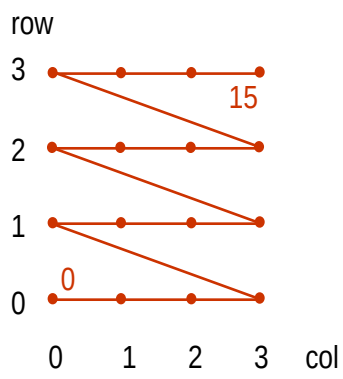
Some Space Filling Curves

Space filling curve used for block/cell creation

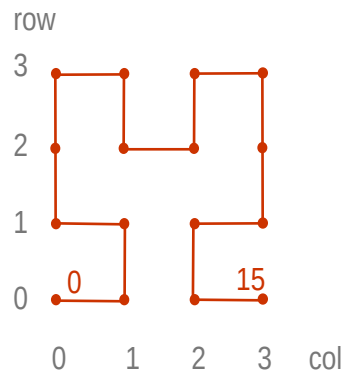
ordering or numbering of cells in kD into 1D using bi-jective mapping

Default of flat model

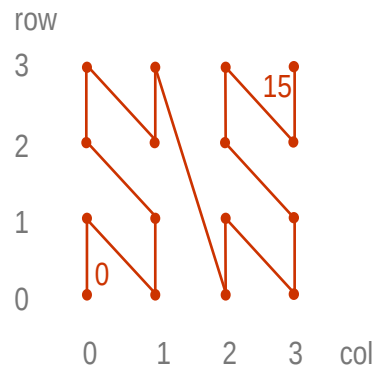
Row (first y, then x)



Hilbert

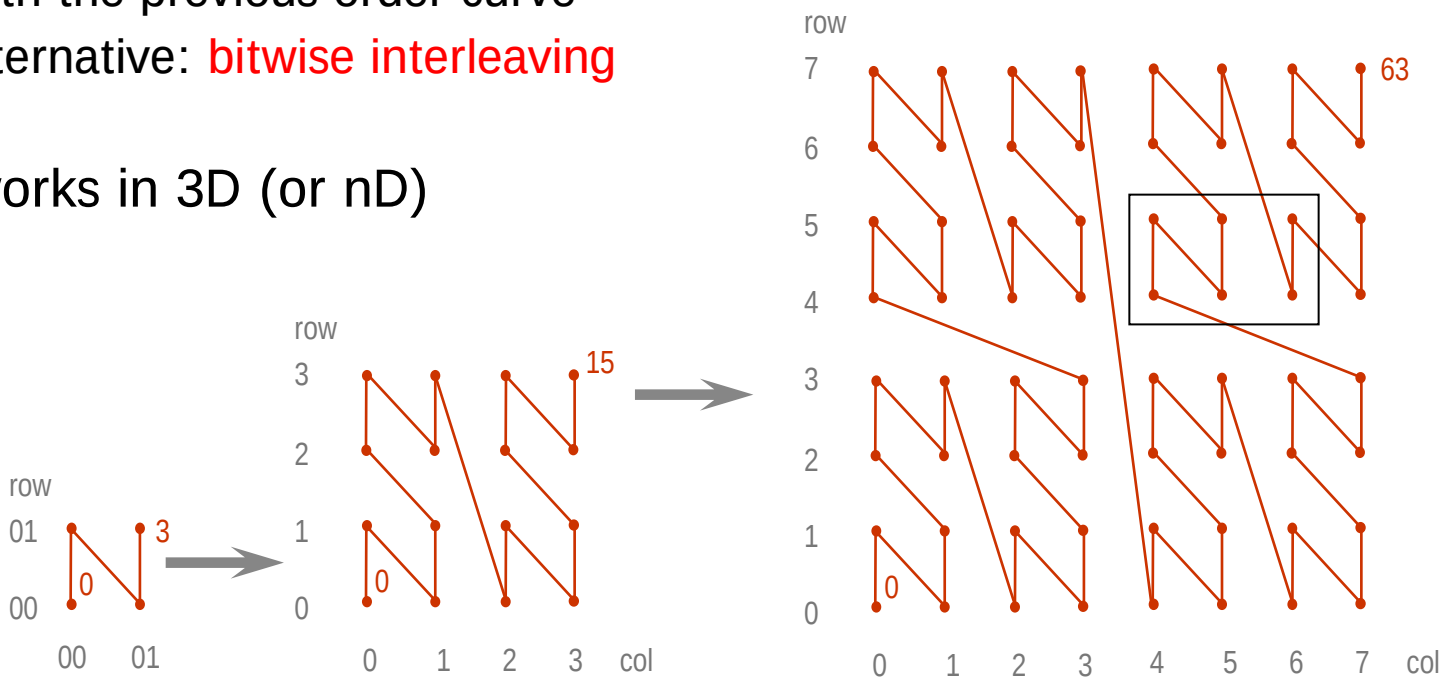


Peano



Construction of Morton Curve

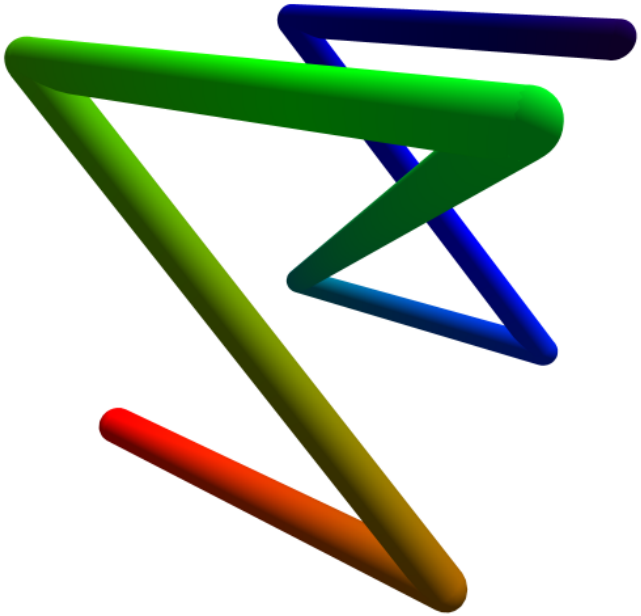
- Morton or Peano or N-order (or Z-order)
  - recursively replace each vertex of basic curve with the previous order curve
  - alternative: bitwise interleaving
- also works in 3D (or nD)



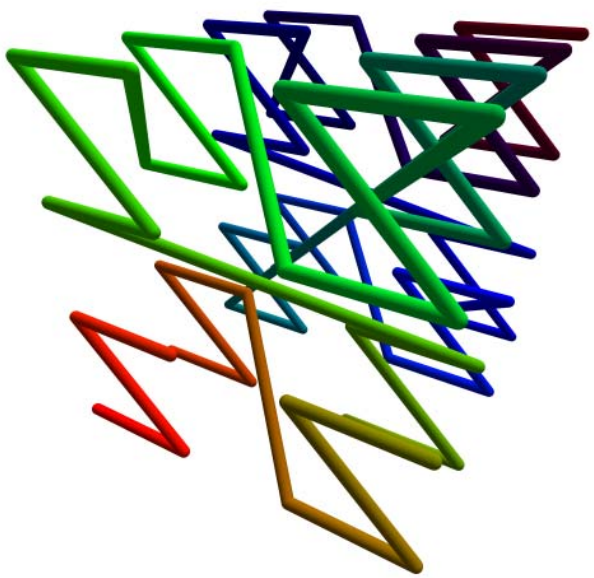
3D Morton curve

illustrations from http://asgerhoedt.dk

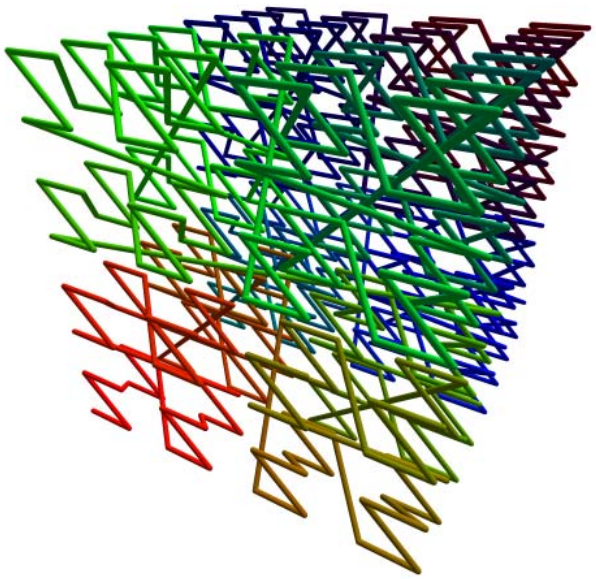
2x2x2



4x4x4

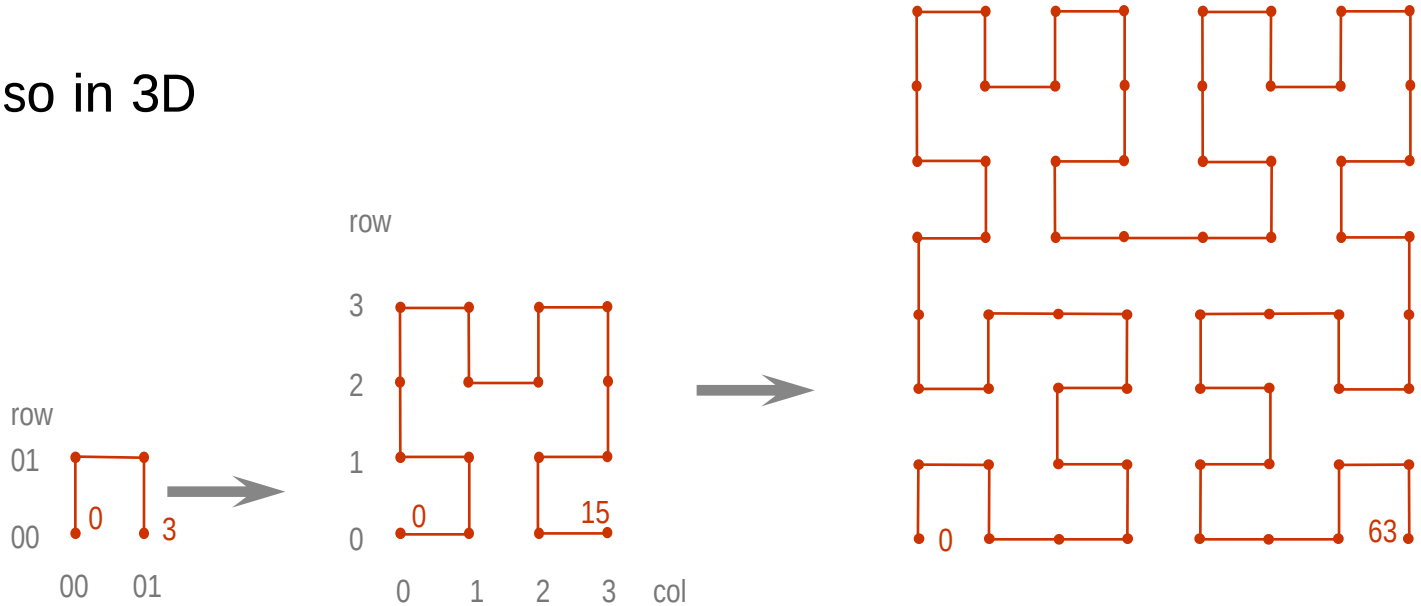


8x8x8



Construction of Hilbert Curve

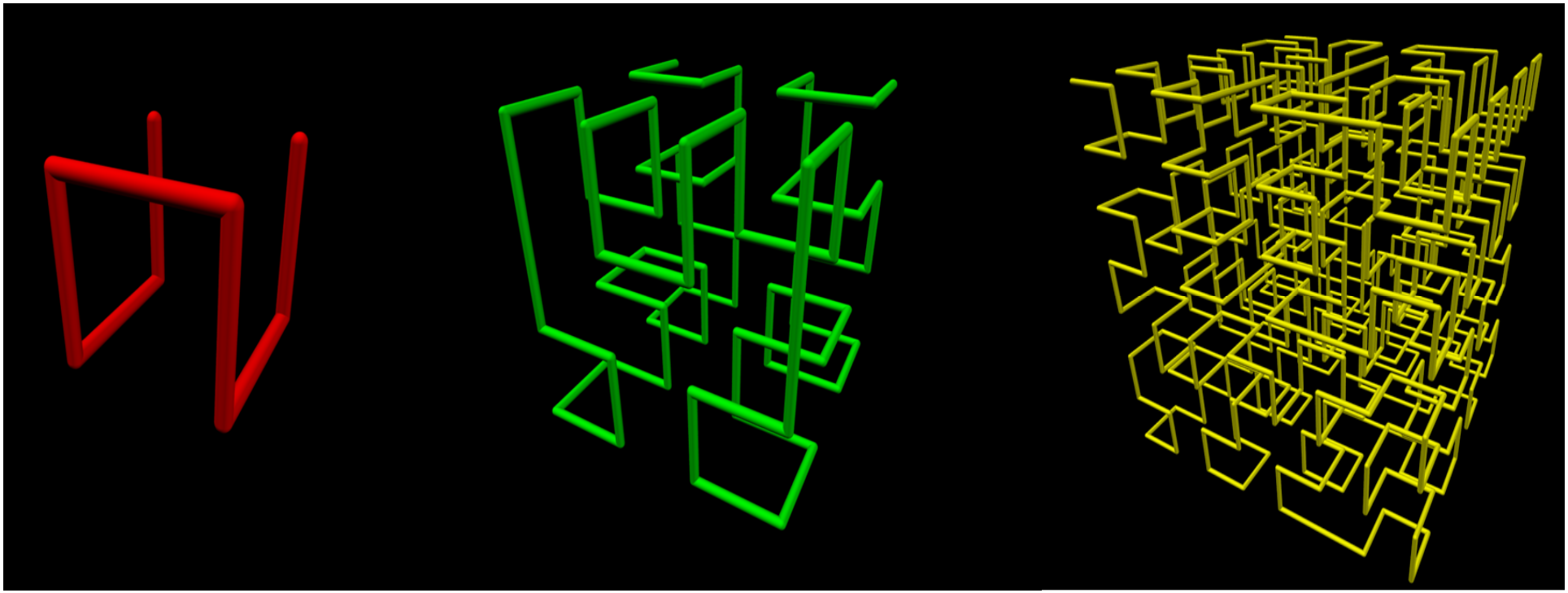
- Hilbert
  - rotate previous order curve
    at vertex 0 (-) and vertex 3 (+)
- also in 3D



3D Hilbert curve

illustrations from Wikimedia Commons

2x2x2 4x4x4 8x8x8

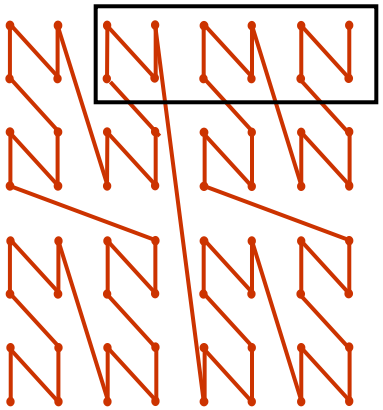


Average number of clusters for all possible range queries

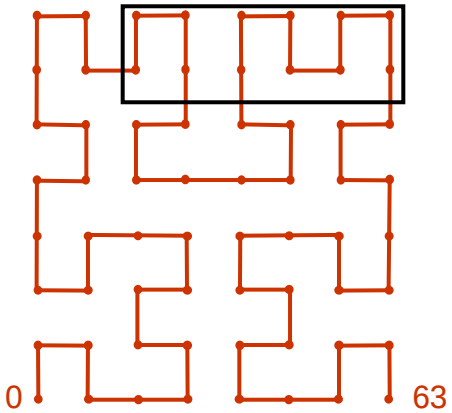
- Faloutsos and Roseman, 1989

N*N GRID	HILBERT	PEANO	I
2*2	1.11	1.22	.11
4*4	1.64	2.16	.52
8*8	2.93	4.41	1.48
16*16	5.60	9.29	3.69

- N=8, number of Clusters for a given range query:



Peano (3 ranges)



Hilbert (2 ranges)

Use Hilbert/Morton code

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

- better flat table model (tested with Oracle):
  - 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)

SQL DDL for index organized table

- Oracle:

```
CREATE TABLE PC_demo (hm_code NUMBER PRIMARY KEY)
    ORGANIZATION INDEX;
```

- PostgreSQL, pseudo solution, not dynamic (better available?):

```
CREATE TABLE PC_demo (hm_code BIGINT PRIMARY KEY);
CLUSTER pc_demo ON pc_demo_pkey;
```

- Oracle Exadata and MonetDB, no index/cluster statements

Morton code technique outline

A. define functions for given square/cubic/... nD domain:

1. Compute_Code(point, domain) → Code; (for storage)
2. Overlap_Codes(query_geometry, domain) → Ranges; (for query)

B. add Morton Code during bulk load or afterwards

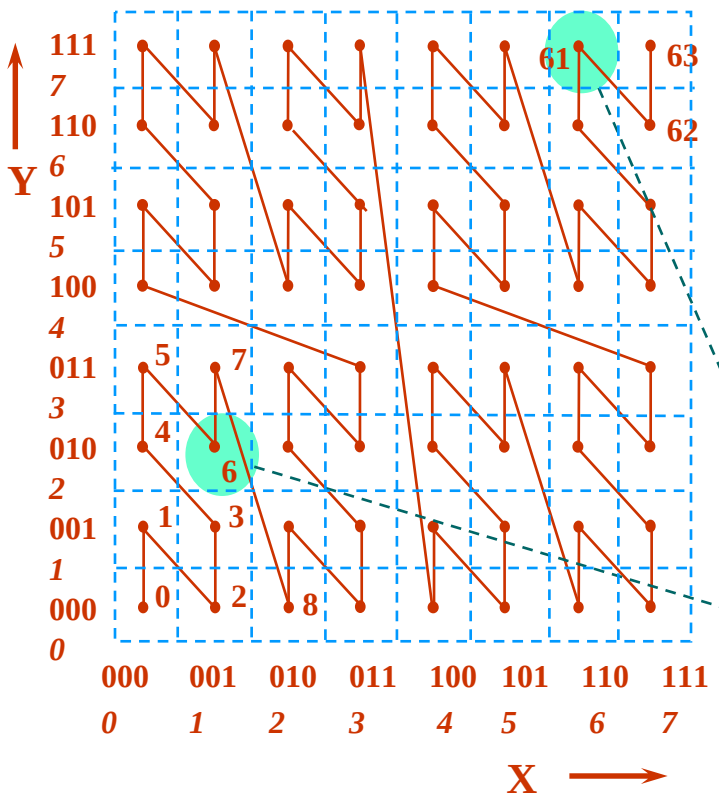
- or even replace point coordinates

C. modify table from default heap to b-tree on Code

Morton code (corresponds to Quadtree in 2D, Octree in 3D, ...)

Compute_Code (point, domain) → Morton_code / Peano key / Z-order

- bitwise interleaving x-y coordinates
- also works in higher dimensions (nD)



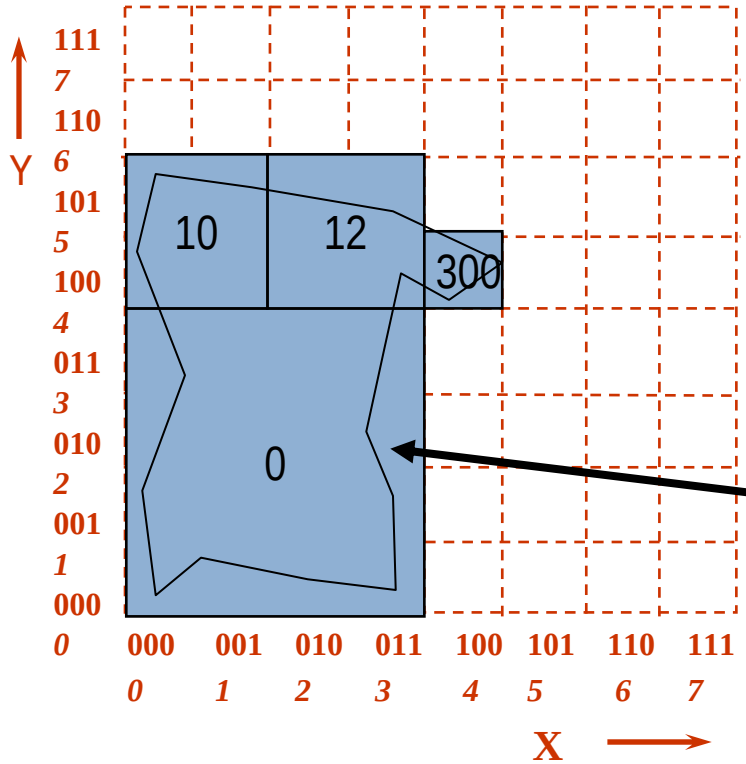
two examples of Morton code:

x= 110, y=111 → xy= 111101 (decimal 61)

x= 001, y=010 → xy= 000110 (decimal 6)

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)

query_geometry, polygon

Note : SW=0, NW=1, SE=2, NE=3

Overlap_Codes(), recursive function Pseudo code

```
Overlap_Codes(query_geometry,domain,parent_quad) def
for quad = 0 to 3 do
  quad_domain = Split(domain, quad);
  curr_quad_code = parent_quad+quad;
  case Relation(query_geometry, quad_domain) is
    quad_covered:  write Range(curr_quad_code);
    quad_partly:   Overlap_Codes(query_geometry,
                     quad_domain, curr_quad_code);
    quad_disjoint: done;
```

notes:
- number of quads 2^k (for 2D: 4, for 3D: 8, etc.)
- quad_covered with resolution tolerance
- Range() translates quadcode to Morton range: start-end
- above algorithm writes ranges in sorted order (eg linked list)

Create ranges & post process (glue)

```
Overlap_codes(the_query, the_domain, '');
Glue_ranges(max_ranges);
```

Overlap_codes() creates the sorted ranges (in linked list).
result can be large number of ranges, not pleasant for DBMS
query optimizer gets query with many ranges in where-clause

reduce the number of ranges to 'max_ranges' with Glue_ranges()
(which also adds unwanted codes)

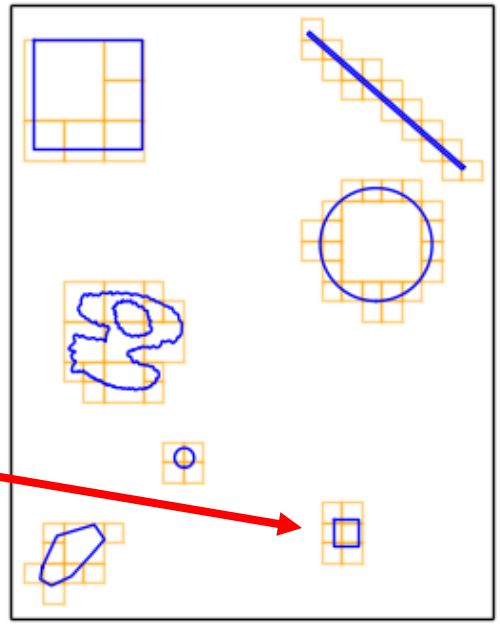
```
Glue_ranges(max_ranges) def
  Num_ranges = Count_ranges();
  Remove_smallest_gaps(num_ranges – max_ranges);
```

notes: - gaps size between two ranges may be 0 (no codes added)
- efficient to create gap histogram by Count_ranges()

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))
```

Query 1 (small rectangle)



Use of Morton codes/ranges (PostgreSQL flat model example)

	Q1 rect	Q4 circle	Q7 line
20M	0.16	0.85	2.32
210M	0.38	1.80	3.65
2201M	0.93	4.18	7.11
23090M	3.14	14.54	21.44

with Morton	Q1 rect	Q4 circle	Q7 line
20M	0.15	0.56	0.82
210M	0.15	0.56	0.42
2201M	0.13	0.64	0.41
23090M	0.15	0.70	0.60

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)

Content overview

0. Background
1. Conceptual benchmark
2. Executable benchmark
3. Data organization
4. Possible standardization
5. Conclusion

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.

Standardization

Standardization actions

- John Herring (ISO TC211, Oracle) asked Peter to join ISO 19107 project edit-team → agreed (John will propose Peter as technical expert on behalf of OGC to ISO 19107 project team)
- within OGC make proposal for point cloud DWG (to be co-chaired by John and Peter)
- better not try to standardize point clouds at database level (not much support/ partners expected), but rather focus on webservices level (more support/ partners expected)

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)

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,...)
- Fugro started developing web-based visualization, based on their desktop experience FugroViewer → Informatics thesis student (Floris Langeraert)

Content overview

0. Background
1. Conceptual benchmark
2. Executable benchmark
3. Data organization
4. Possible standardization
5. Conclusion

Conclusion

- 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,...

- Excellent team, world leading partners

Next Phase of project

- Ambitious project plan, increased by unplanned activities:
  - MonetDB
  - Lastools (and Esri's ZLAS format)
  - Patty project
  - Via Apia project

- More data?
  - Cyclomedia images / areal photographs
  - Very high density, prediction 35 trillion points for NL
  - More attributes (r,g,b)
  - 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)