Benchmarking and improving point cloud data management in MonetDB

Oscar Martinez-Rubi[1], Peter van Oosterom[2], Romulo Gonçalves[1], Theo Tijssen[2], Milena Ivanova[1], Martin L. Kersten[3], Foteini Alvanaki[3]
[1]Netherlands eScience Center, Amsterdam, The Netherlands
[2]Delft University of Technology
[3]CWI, Amsterdam, The Netherlands

Abstract

The popularity, availability and sizes of point cloud data sets are increasing, thus raising interesting data management and processing challenges. Various software solutions are available for the management of point cloud data. A benchmark for point cloud data management systems was defined and it was executed for several solutions. In this paper we focus on the solutions based on the column-store MonetDB, the generic out-of-the-box approach is compared with two alternative approaches that exploit the spatial coherence of the data to improve the data access and to minimize the storage requirements.

1 Introduction

With the recent advances in point cloud data acquisition technologies the increasing sizes of the produced data sets require efficient point cloud data management systems. Most of the existing management solutions present limits in the amount of data that can be handled efficiently in terms of data loading or preparation times, storage requirements and data retrieval response times. Therefore, concerted action is needed in order to improve the existing systems or develop new options to handle the massive point clouds that are becoming available. Scalable data structures and efficient data management strategies should be explored.

Specific file-based approaches such as the scriptable tools provided by Rapidlasso LAStools [7] are very widely used. Some database systems such as Oracle and PostgreSQL already offer point cloud support in the form of extensions and specific data types. Concretely, the SDO_PC and SDO_PC_BLK data types in Oracle Spatial and Graph [5] and the PCPATCH data type in PostgreSQL-PostGIS [13] implement a storage model based on the grouping of spatially co-located points in blocks. An alternative storage model for databases is the straight-forward flat table model where each point is stored in a row of the table. Oracle experts recommend this storage model when using Oracle Exadata systems [6] for point cloud data management. In the column-store MonetDB [3], initial tests with a flat table model implementation were performed [12] revealing that column stores offer promising routes for point cloud data. More recently, the MonetDB developers have started to improve the spatial support of the column-store with special focus on the point cloud data.

The interaction between point clouds and Geographic Information System (GIS) was not properly explored. An inventory of the data management user requirements was compiled using structured interviews with point cloud data users from different background: government, industry and academia [15]. These user requirements were used to define a conceptual benchmark. An executable benchmark was also developed and executed for different point cloud data management systems using Oracle, PostgreSQL, LAStools and MonetDB [16]. Among

other things, we identified the limitations of using regular flat tables and default indexing methods for the storage of point cloud data.

In this paper we present alternative management solutions for MonetDB exploiting the spatial coherence of the data using space filling curves. Some of the used concepts have been already introduced and tested in PostgreSQL [16] with satisfactory results. Here we re-use and extend these concepts and we use the developed benchmark platform to compare the new alternatives approaches with the out-of-the-box approach that was previously used.

The remainder of the paper is as follows. MonetDB is introduced in Section 2. Section 3 gives an overview of the benchmark for point cloud data management systems. The used hardware and software are detailed in Section 4. In Section 5 we describe the tested solutions, all based on MonetDB. The results of the execution of the benchmark are presented in Section 6. Finally, Section 7 contains the conclusions extracted from this test and Section 8 presents the future work.

2 MonetDB

MonetDB is an in-memory column-stored database system developed by the Database Architectures group at the Centrum Wiskunde and Informatica (CWI) in Amsterdam, the Netherlands. MonetDB is open-source and it is being successfully used in various fields [9, 11] and it has a modern architecture with interesting features such as (a.) vertical fragmentation of the storage model, i.e. a column-store database kernel, (b.) CPU-tuned and automatically parallelized query execution architecture, (c.) a novel indexing strategy called Imprints [14], that creates index structures on-the-fly when the columns are queried, (d.) completely modifiable and extendable system thanks to the liberal open-source license, (e.) run time query optimization, (f.) a modular software architecture, (g.) a basis on the SQL 2003 standard with full support for foreign keys, joins, views, triggers and stored procedures, and (h.) being a fully ACID compliant system that supports a rich spectrum of programming interfaces (JDBC, ODBC, PHP, Python, RoR, C/C++, Perl).

2.1 Spatial column-store

An initial SQL interface to the Simple Feature Specification of the Open Geospatial Consortium(OGC) [10] was already available in previous releases of MonetDB with a limited support for the objects and functions defined in the specification. Recently an improvement and extension of the MonetDB spatial module has been initiated with a special focus on the point cloud data. The authors team is cooperating with the activities of the MonetDB developers and this has resulted in improvements for the point cloud data support such as a binary loader of point cloud data. The loader takes as input LAS files, or compressed LAS files (LAZ)[8], and for each property it generates a new file that is the binary dump of a C-array containing the values of the property for all points. Then, the generated files are appended to each column of the flat table using the bulk loading operator COPY BINARY [4].

3 Benchmark description

The executable benchmark uses different subsets of the Actueel Hoogtebestand Nederland 2 (AHN2) [1], a point cloud data set with over 640 billion points of the surface of the Netherlands.

The benchmark has different stages with increasing tested functionalities and data sets sizes: (a.) The mini-benchmark stage is used to gain experience with the different systems using a small data set with 20 million points, a small set of functionalities are tested. (b.) In the medium benchmark stage four subsets with various sizes are used, from 20 million points to 23 billion points. A larger set of functionalities is used with 20 different queries. The goal of this benchmark stage is to analyze the performance of the most wanted functionalities

according to the user requirements as well as observing possible scaling issues. (c.) The full-benchmark stage tests even more queries or functionalities with the 640 billion points of the full AHN2 data set. (d.) In the future, scalability will be further tested in the upscaled-benchmark with a data set size of 20 trillion points.

Some executions of the benchmark have already been done using up to the full-benchmark stage [16]. For the results presented in this paper we use the medium-benchmark stage.

4 Hardware and Software

For the tests described in this document we have used a server with the following details: HP DL380p Gen8 server with 2 x 8-core Intel Xeon processors, E5-2690 at 2.9 GHz, 128 GB of main memory, and a RHEL 6 operating system. The disk storage which is directly attached consists on a 400 GB SSD, 5 TB SAS 15K rpm in RAID 5 configuration (internal), and 2 x 41 TB SATA 7200 rpm in RAID 5 configuration (in Yotta disk cabinet).

Regarding the software, we used a modified version of the default branch of MonetDB downloaded on the 11th December 2014. The modifications include the definition and implementation of new functions within the spatial module that are not yet included in the default branch as well as functions related to the approaches using the space filling curves. However, all these features will be included in the April 2015 MonetDB release.

5 MonetDB point cloud data management systems

In this work we compare three approaches using MonetDB for the management of point cloud data. First we describe the out-of-the-box approach or regular flat table approach followed by the Morton-based flat table approaches, the Morton-added approach and the Morton-replacedXY approach.

5.1 Out-of-the-box approach

The straight-forward approach for managing point clouds in the MonetDB system is based on storing the points in a regular flat table, one entry per point and let the system automatically take care of the indexing by using the Imprints index structures when required. This approach with an ASCII loader was used in the initial point cloud tests of MonetDB where we compared it with the PostgreSQL point cloud approach [12]. It has also been used in the more recent benchmark executions where it was compared with other approaches using PostgreSQL, Oracle and LAStools [16]. However, for the experiments of this paper we used the binary loader which is a lot faster then the ASCII loader.

For the queries we used a two-step approach based on doing a preselection of points in the bounding box of the query region that are later refined to obtain only the points in the query region. Note that the second filtering is not required for rectangular regions. In the case of circles the refinement is done with the distance method, i.e. $(x - x_c)^2 + (y - y_c)^2 < r$ while the *contains* method is used for other types of regions (polygons).

In the previous tests and benchmark executions it was revealed that, even though the query response times in the flat tables approaches were, in general, faster for small data sets than the blocks-based approaches in PostgreSQL and Oracle, they presented significant scaling issues. We observed that the scale factor for the flat tables was in the order of 10 to 3, thus a data set with ten times more points had three times slower query response times.

5.2 Morton-based approaches

Space filling curves are already being used for efficient access to spatial data in some solutions such as LAStools and Oracle Spatial and Graph. LAStools exploits the potential of spatial filling curves in point cloud data in a very efficient manner in its tools *lassort*, *lasindex*, *lasmerge* and *lasclip*. On the other hand, Oracle Spatial and

Graph also uses some space filling curves concepts in one of the blocking methods available in the point cloud module. In previous work [16] we demonstrated that the space filling curves can be used to improve the query response times of the flat table approaches. In that case we used PostgreSQL and we observed that, by using a Morton-based flat table, we could get scalable query response times, i.e the query response times only depended on the size and complexity of the query region and not on the data set size.

5.2.1 Morton-added approach

In this approach we compute for each point its position in a Morton curve filling the two-dimensional data set domain, this value which is stored in a 64 bits unsigned integer is usually referred to as Morton code. Figure 1 (left) shows examples of Morton code computations. Note that the X and Y values have to be offsetted to be positive values when computing the Morton code.

In the database we load the x, y, z and the Morton code in a staging table. Next we create the final table with a CTAS (Create Table As Select) statement with an *ORDER BY* expression to sort the points by their Morton code values. This enables a very interesting feature of MonetDB, if a table has been created from a select statement with an *ORDER BY* expression, the queries that are related to the ordered dimension are solved by using a binary search algorithm (similar to the functioning of a B-Tree index). Hence, this feature allows to have the data and the index structure in one unique entity. In addition the data becomes spatially coherent, near points in space are also closely stored, further improving the data access speed by exploiting data locality.

The SQL statement to create the final table ordered by the Morton code is:

```
CREATE TABLE final_table_name AS
SELECT * FROM staging_table_name ORDER BY k WITH DATA
```

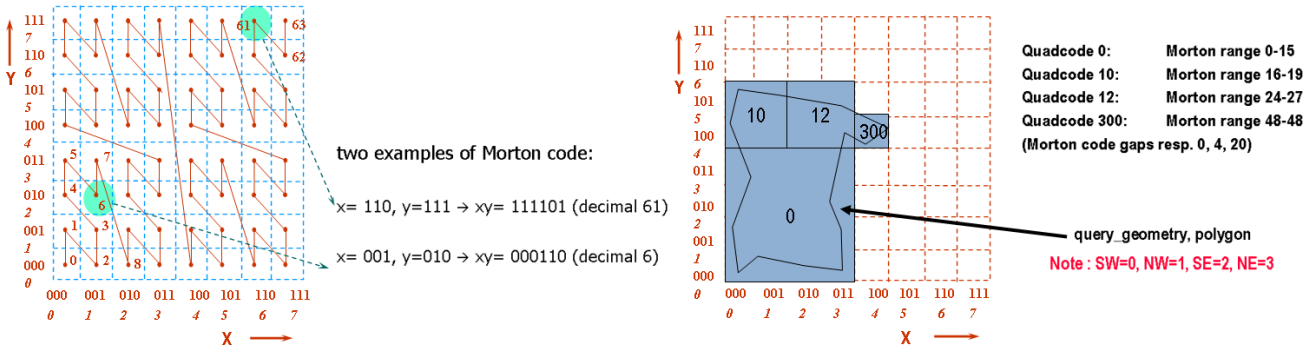

Figure 1: Left: Morton code computation of single cell by bitwise interleaving x and y; Right: Ranges of Morton codes for the query polygon (note the Quad-Tree relationship)

Regarding the queries, using the intrinsic relationship between the Morton space filling curve and the Quad-Tree structure the queries to X and Y can be converted to a list of Morton code ranges. However, this initially requires the query geometry to be decomposed in Quad-Tree cells. This conversion is currently done out-of-the-core with a Python script. An example of the relationship between the Morton curve and the Quad-Tree is illustrated in Figure 1 (right). For example, the actual query statement for the query region #1 of our benchmark is:

```
CREATE TABLE query_results_1 AS SELECT x,y,z FROM (SELECT * FROM ahn_flat WHERE (
(m between 8860467200 and 8862564351) OR (m between 8864661504 and 8875147263) OR
(m between 8877244416 and 8879341567) OR (m between 8883535872 and 8887730175) OR
(m between 8891924480 and 8902410239) OR (m between 8904507392 and 8906604543) OR
(m between 8908701696 and 8912895999) OR (m between 8917090304 and 8919187455))) a
 WHERE (x between 85670.0 and 85721.0) and (y between 446416.0 and 446469.0) WITH DATA
```

The points in the Quad-Tree cells overlapping the query region (given by the Morton ranges) are preselected, similarly to the preselection done using the bounding box in the regular flat table approach. The preselected points are later filtered out with (a.) exact X and Y range selection for rectangles like in the example for query region #1, (b.) distance method for circles, or (c.) the *contains* method for other polygons.

5.2.2 Morton-replacedXY approach

The Morton-added approach described in Section 5.2.1 stores for each point the X, Y, Z coordinates and the Morton code. However, if the space filling curve is completely defined for the full resolution of the entire domain, then the X and Y values can be recovered from the Morton code, which in this case we call replacing code. As previously stated we use a 64 bits unsigned integer for the code. Ideally, we have 32 bits for each X and Y which means we can cover over 460 million km^2 with centimeter resolution (the Earth surface is 510 million km^2). In practice, we only allow 31 bits for X and Y because unsigned integers are not supported in MonetDB but we still need to guarantee that they are positive numbers producing a positive code, otherwise we would have strange effects on the produced curve. Therefore, the actual coverage is 115 million km^2 which is still more than enough to cover all the used data sets as well as the whole AHN2.

In the Morton-replacedXY approach we only store Z and the replacing Morton code, the X and Y coordinates are decoded when required. As in the previous case the data is ordered using the replacing Morton code and the queries to X and Y need to be converted. The querying procedure is similar to the Morton-added approach, the only difference is that the X and Y values are decoded after the preselection of points in the Quad-Tree cells is done. The advantage is that the separated X and Y columns do no need to be accessed.

6 Benchmark execution

The medium-benchmark stage was executed with the three described MonetDB approaches. Table 1 contains the times and storage requirements of the loading of four data sets with 20 million points, 210 million points, 2 billion points and 23 billion points. The time is split in three steps: (*Init*) The MonetDB instance is created, (*Load*) the data is loaded with the binary loader, and (*Close*) the index is created for the out-of-the-box approach or the data is sorted for the Morton-based approaches.

Approach	Data set [points]	Time[s]				Size[MB]		Points/s
		Total	Init	Load	Close	Total	Index	
Out-of-the-box	20165862	5.25	0.92	2.78	1.55	472	10	3841117
Morton-added	20165862	11.98	1.08	4.08	6.82	615	0	1683294
Morton-replacedXY	20165862	9.17	1.04	2.76	5.37	308	0	2199113
Out-of-the-box	210631597	19.58	1.05	14.45	4.08	4868	47	10757487
Morton-added	210631597	80.75	1.22	17.36	62.17	6428	0	2608441
Morton-replacedXY	210631597	59.88	0.95	11.80	47.13	3214	0	3517562
Out-of-the-box	2201135689	205.20	0.98	174.46	29.76	50579	199	10726782
Morton-added	2201135689	1483.45	1.20	239.45	1242.80	67173	0	1483795
Morton-replacedXY	2201135689	732.57	1.35	151.58	579.64	33586	0	3004676
Out-of-the-box	23090482455	3921.07	1.53	3440.20	479.34	529180	680	5888822
Morton-added	23090482455	55542.70	1.28	4603.42	50938.00	704666	0	415725
Morton-replacedXY	23090482455	38973.80	1.31	2532.34	36440.20	352333	0	592461

Table 1: Times and sizes of the data loading procedure for the different approaches and data sets.

The fastest entire data loading procedure (*Init*, *Load* and *Close* steps) was obtained in the out-of-the-box approach while the least storage was required by the Morton-replacedXY approach. In fact, the fastest data loading using the binary loader (*Load* step) was also in the Morton-replacedXY approach because only two columns are required. However, the binary loader had a drop of performance for the largest dataset in all the

approaches due the size of each column (170 GB) being larger than the available memory (128 GB). Improved loading procedures are being investigated to address this performance drop. The procedure used in the Morton-based approaches when sorting the data (*Close* step) was worryingly slow with responses that were not either linear (*O(n)*) or linearithmic (*O(n log n)*). On the other hand, the Imprints indexing used in the out-of-the-box approach offered an almost linear growing. Compared to the out-of-the-box approach the Morton-replacedXY approach saved around 30% storage while the Morton-added approach required an additional 30% storage.

The twenty queries of the medium-benchmark were executed. In Figure 2 we show a sample of the response times of four queries for the different data sets and approaches.

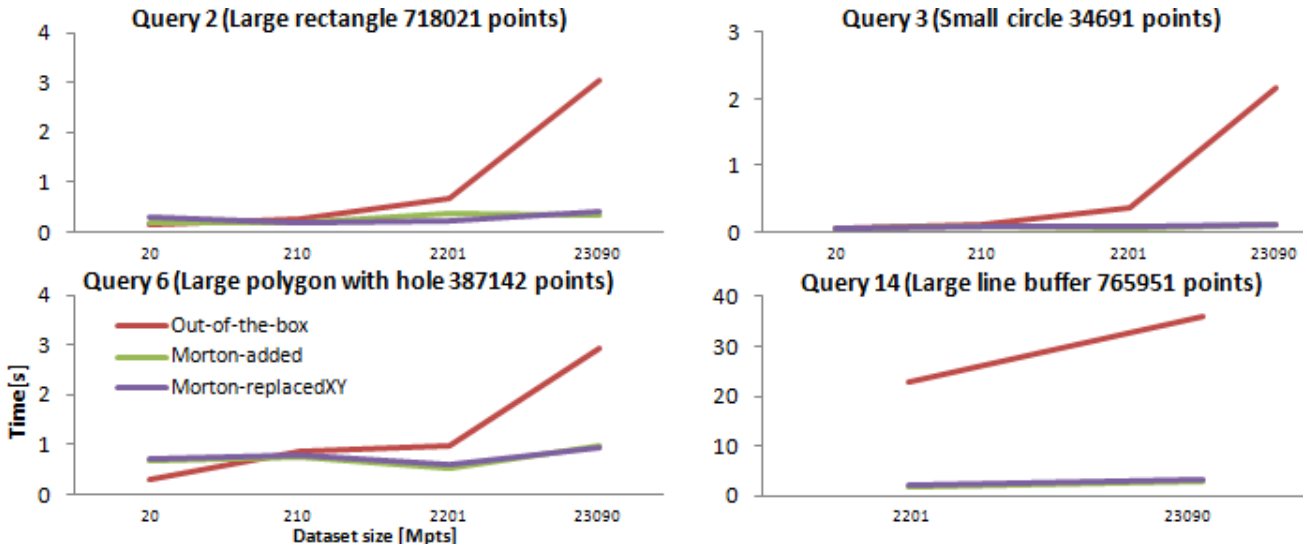

Figure 2: Response times for the various approaches with different data set sizes for four of the twenty queries of the medium-benchmark. Note that query 14 was not executed in the smaller data sets because the query region was not overlapping their spatial extent.

Due to the latest updates in the spatial module of MonetDB the response times have decreased compared to the results obtained in previous tests, specially in the queries using the *contains* method. Comparing the three approaches we observe that the scaling was much better for the Morton-based approaches. For the largest data set the Morton-based approaches offered typically 20 times faster times. The two Morton-based approaches had a similar query performance even though the Morton-replacedXY needs to decode the X and Y values which makes it slightly slower even though this is hardly noticeable in the graphs. However, in both cases the response times still increase a little with growing input data sizes. In the previous test with PostgreSQL the query times with the Morton-based approach were more constant; i.e. no dependent on the input size, but only of the size and complexity of the query region.

7 Conclusion

In this paper we have presented the comparison of three different approaches for point cloud data management using MonetDB. The developed binary loader improves enormously the loading times when compared to the ASCII loader but it still presents a drop in the performance for large point clouds which becomes noticeable when the size of each column is larger than the available memory. The sorting procedure used in the alternative approaches based on the Morton space filling curves is not as scalable as the Imprints indexing used in out-of-the-box approach. The Morton-replacedXY approach decreases the storage requirements around 30% when compared to the out-of-the-box approach. The recent developments in MonetDB spatial support have already

improved the querying times of the out-of-the-box approach when compared to previous tests [12]. However, it still presents important scaling effects in the query response times. The Morton-based approaches offer better scalability in the query response times. The obtained results show that column-stores such as MonetDB can get scalable and faster spatial data access and decreased storage requirements when using space filling curves to enhance the point cloud data management. However, in order to have a scalable MonetDB solution for both loading and querying the indexing, the sorting and the storage structures used for the point cloud data still need to be further improved.

8 Future work

Improved solutions for point cloud data management will be developed for MonetDB that will combine the ideas presented in this paper with other developments currently being researched within the spatial module of MonetDB. The binary loader will be improved to address the observed drop of performance for large datasets.

Hilbert curves are an interesting alternative to Morton curves that should be tested, they are better at preserving locality but they are more difficult to be implemented. Either Morton or Hilbert curves can as well be used to build 3D codes with X, Y and Z. These codes from higher dimensional curves can also be used as replacing codes for the coordinates, thus improving the storage requirements as seen in the 2D case. However, at this point we encounter challenges in the bit encoding of such replacing Morton or Hilbert codes as 64 bits are not enough to fit full resolution codes for significantly large spatial domains. In addition, difficulties arise when porting the query algorithm to 3D. For example, querying a 2D region in a domain covered by a 3D space filling curve means a whole column within the Oct-Tree domain with much more Oct-Tree cells or ranges of Morton codes.

An important feature which is currently missing in many of the existing point cloud management solutions and must be added in the new solutions is the Level of Detail (LoD). For example, being able to select a certain percentage of the points in the query region. Implementations using 4D space filling curves with the LoD as forth dimension with replacing codes should be explored for MonetDB. Easier 3D implementations with X, Y and LoD in the replacing 3D code and Z as an additional column could be a starting point.

In the Morton-based approaches the acquisition of the overlapping Quad-Tree cells with the query region done in the querying is currently performed out-of-the-core in a Python script. Porting this procedure inside MonetDB will improve the query performance as well as provide a more user-friendly solution.

For all the new solutions, even more functionalities and larger datasets will be tested with the full-benchmark stage and the upscaled-benchmark stage.

Acknowledgments

The authors would like to thank the CWI database group and the rest of members of the massive point cloud project (see project website [2]. The research conducted in this project is supported in part by the Netherlands eScience Center under project code: 027.012.101. The MonetDB developments in point cloud data management are partly within the strategic partnership between the Netherlands eScience Center and the CWI database architectures group under project code: 027.013.703.

References

[1] Actueel Hoogtebestand Nederland (AHN). `http://www.ahn.nl/`.

[2] Massive point clouds for eSciences. `http://www.pointclouds.nl/`.

[3] MonetDB. `https://www.monetdb.org`.

[4] MonetDB binary bulk load. https://www.monetdb.org/Documentation/Cookbooks/SQLrecipes/BinaryBulkLoad.

[5] Oracle Database Online Documentation 12c Release 1 (12.1): Spatial and Graph Developer's Guide / SDO_PC_PKG Package (Point Clouds). http://docs.oracle.com/database/121/SPATL/sdo_pc_pkg_ref.htm.

[6] Oracle Exadata Database Machine X4-2. https://www.oracle.com/engineered-systems/exadata/database-machine-x4-2/index.html.

[7] Rapidlasso GmbH. http://rapidlasso.com/.

[8] rapidlasso GmbH LASzip - free and lossless LiDAR compression. http://www.laszip.org/.

[9] P. Boncz, S. Manegold, and M. Kersten. Optimizing main-Memory join on modern hardware. *TKDE*, 2002.

[10] J. R. Herring. OpenGIS Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture . http://portal.opengeospatial.org/files/?artifact_id=25355, May 2011.

[11] S. Manegold, M. Kersten, and P. Boncz. Database Architecture Evolution: Mammals Flourished Long Before Dinosaurs Became Extinct. VLDB, 2009.

[12] O. Martinez-Rubi, M. Kersten, R. Goncalves, and M. Ivanova. A column-store meets the point clouds. FOSS4GEurope, 2014.

[13] P. Ramsey. A PostgreSQL extension for storing point cloud (LIDAR) data. https://github.com/pramsey/pointcloud.

[14] L. Sidirourgos and M. Kersten. Column Imprints: A Secondary Index Structure. SIGMOD, 2013.

[15] P. M. Suijker, I. Alkemade, M. P. Kodde, and A. E. Nonhebel. User requirements Massive Point Clouds for eSciences (WP1). Technical report, Delft University of Technology, April 2014.

[16] P. van Oosterom, O. Martinez-Rubi, M. Ivanova, M. Horhammer, D. Geringer, S. Ravada, T. Tijssen, M. Kodde, and R. Goncalves. Massive point cloud data management: design, implementation and execution of a point cloud benchmark. *Computer Graphics*, 2015. (Manuscript accepted for publication).