Point Cloud: Storage, Loading, and Visualization

Siva Ravada, Mike Horhammer, Baris M. Kazar
Oracle Spatial
Oracle USA, Inc., One Oracle Drive, Nashua, NH 03062, USA

I. Introduction

In recent years laser scanning and terrestrial imaging have enabled the collection of large amounts of LiDAR (Light Detection and Ranging) data. New laser scanning equipment made it easier to acquire 3-D representations of real world entities as high-density point clouds, which may then be generalized to compact forms such as TINs, meshes, surfaces and solids. Increasing LiDAR densities bring new data management challenges for large volumes of point data. Lag time from acquisition to analysis, metadata access and management, fusion with other geospatial and attribute data, multi-user access and security are some of the challenges in managing large scale LiDAR data. Scaling to the sizes of such large point data sets is becoming increasingly impossible for desktop applications. This paper focuses on the high-performance storage, loading and visualization of SDO_PC (Point Cloud) type in 3-D Spatial Engine.

II. Storage model of SDO_PC Type

Oracle Spatial introduces a new data type called SDO_PC to manage very large volumes of data generated by LiDAR scans. The data generate from LiDAR scans is usually called Point Cloud data and SDO_PC type is designed to efficiently manage such point cloud data. Users can store up to $4 \cdot 10^{18}$ points in a single point cloud object in Oracle. Partitioning the points into fixed-size blocks and storing the blocks as rows in a separate block table achieve this scalable storage. Figure 1 shows the storage schematic for the point cloud.

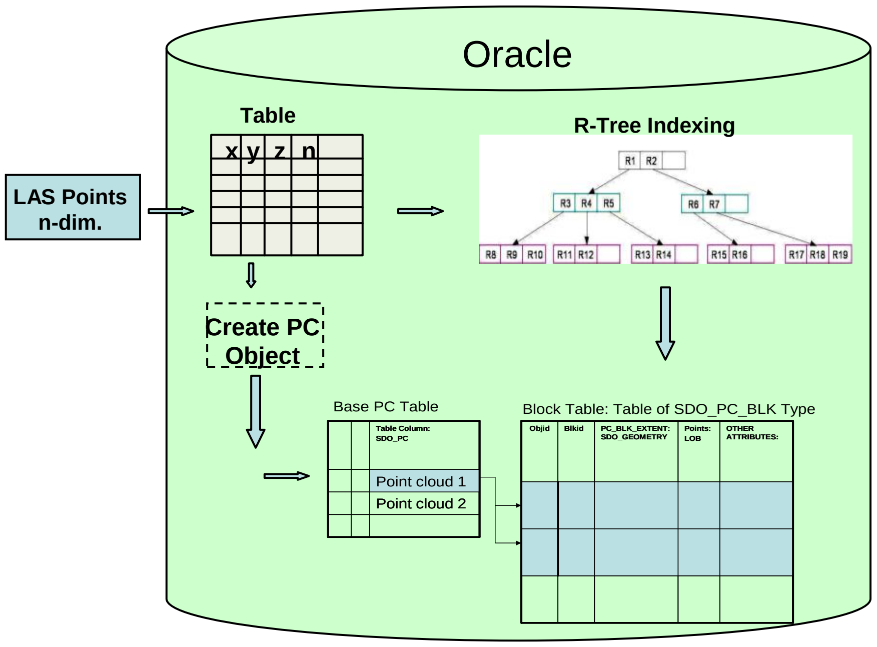

Fig. 1. Storage Model for SDO_PC type

As shown in Figure 1, only the metadata, such as the *block capacity*, the *total number of dimensions* (at least 3: *x*, *y*, *z* and usually more, e.g., color), and the *block table* information is stored in the SDO_PC column. The data points themselves are partitioned into subsets (using spatial partitioning techniques), each subset being stored in the *points* column of the associated block table. Note that the *blk_extent* column stores the extent of each subset and is spatially indexed. This *blk_extent* column allows for efficient query processing in a 2-stage filter processing as discussed in Section III. The *Max_res*, *Min_res* columns allow point clouds to be stored in multiple resolutions. Such multi-resolution point clouds are useful in some applications that need to access them based on proximity to a viewpoint (closer for higher resolution and farther for less detail). This partitioned or blocked storage of a point cloud ensures scalability.

III. Queries on SDO_PC Objects

Oracle provides a clip PL/SQL query on the SDO_PC type. This query takes a *spatial window* (specified as an SDO_GEOMETRY) and identifies all points that have any interaction with the spatial window. In addition to the spatial window, the query also takes a resolution range to identify only those blocks that satisfy the resolution range. This type of query is typically called a resolution-based visibility query and we refer to this as the clip operation. To restrict the scope of the discussion, we refer to the spatial window as the "query" in the rest of the section, ignoring any resolution range portions of the query. As shown earlier in Figure 1, a point cloud is stored as multiple rows in an associated block table, each row containing a *blk_extent* column of type SDO_GEOMETRY to represent the extent of the points in that row. This *blk_extent* is implicitly indexed with an Oracle R-tree index. Using such an index, the CLIP query on an SDO_PC is processed in 2-stages as shown in Figure 2.

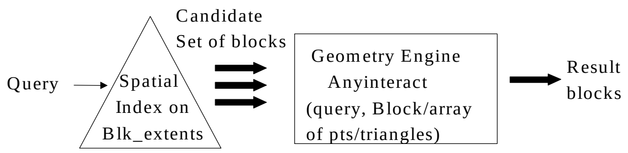
Query
Spatial Index on Blk_extents
Candidate Set of blocks
Geometry Engine Anyinteract (query, Block/array of pts/triangles)
Result blocks

Fig. 2. 2-Stage Query Processing on a Point Cloud.

1. *First-stage*: Using the spatial index on the *blk_extent*, the query identifies all the relevant blocks of a point cloud.
2. *Second-stage*: For each relevant block for the query (identified in first-stage), the associated points are retrieved. Typically the number of such points in a block is of the order of thousands or tens of thousands (typical block capacity). These points are compared with the query geometry using the *anyinteract* function. The intersecting points (and/or triangles) are returned as part of the result of the CLIP routine. The CLIP routine, implemented as a pipelined table function [1], returns one result block row corresponding to each candidate block identified in the first stage in an iterative manner.

Oracle Spatial creates a spatial R-tree index on the *blk_extent* column of the block table. Using such an index, Oracle quickly identifies the blocks that are relevant for a spatial query window. The actual points that are "definitely" relevant for the query window are identified using an "*anyinteract*" query between the query-geometry and the points in the point cloud in the Geometry Engine (secondary filter).

IV. Visualization of SDO_PC Objects

For the purpose of visualization, we use pyramiding, to focus on just a representative subset of points. The entire set of points intersecting with a view frustum may be excessive, and visually redundant.

V. Pyramiding

Originally, the entire point set in a point cloud is spatially clustered and indexed, using the Oracle R-tree. We may consider these "original blocks" a single "level of detail". Any spatial query on the original blocks will return all intersecting points, usually many millions.

To add pyramiding, we generate additional levels of detail $1, \cdots, n$, with level 1 being least-detailed. To avoid repetition, the "pyramid blocks" store each point from the original blocks, exactly once. Thus, the pyramid blocks repeat points from the original blocks, but there is no internal repetition, within the pyramid. A visualization app will only read pyramid blocks of the appropriate level of detail, but never the original blocks.

Generally, any block at level i "summarizes" multiple blocks at level $i+1$, thus covering their spatial extent union. There is, thus, an implicit tree structure, within the pyramid.

The generation of the pyramid can be massively parallel: at any level of detail i, the blocks can be generated, independently from its neighbors, contingent only on its child nodes.

VI. References

[1] Oracle Database 11*g* Release 1 Documentation. http://www.oracle.com/technology/documentation/database.html

[2] S. Shekhar, S. Ravada, G. Turner, D. Chubb, and V. Kumar, Load Balancing in High Performance GIS: Partitioning Polygonal Maps, *Proc. Intl. Symp. on Large Spatial Databases*, Springer Verlag (Lecture Notes in Computer Science), 1995.

[3] S. Shekhar, S. Ravada, V. Kumar, D. Chubb, and G. Turner, Parallelizing a GIS on a Shared-Address Space Architecture, IEEE Computer, December, 1996.

[4] S. Shekhar, S. Ravada, V. Kumar, D. Chubb, and G. Turner, Declustering and Load-Balancing Methods for Parallelizing Spatial Databases, *IEEE Transactions on Knowledge and Data Engineering*, Volume 10, Issue 4, pages 632-655, 1998.

[5] S. Shekhar, B. M. Kazar, D. J. Lilja, Scalable Parallel Approximate Formulations of Multi-Dimensional Spatial Auto-Regression Models for Spatial Data Mining, *24th Army Science Conference*, November 2004, Orlando FL, USA.

[6] B. M. Kazar, S. Shekhar, D. J. Lilja, D. Shires, J. Rogers, M. Celik, A Parallel Formulation of the Spatial Auto-Regression Model, *the International Conference on Geographic Information GIS PLANET*, May 2005, Lisbon, Portugal.

[7] B. M. Kazar, S. Shekhar, D. J. Lilja, Scalable Serial and Parallel Formulations of Multi-Dimensional Spatial Auto-Regression (SAR) Models for Spatial Data Mining, *Workshop on Data Mining: Army Perspectives, Methods, and Applications*, July 2004, Aberdeen MD, USA.

[8] B. M. Kazar, S. Shekhar, D. J. Lilja, Parallel Formulation of Spatial Auto-Regression, *PPoPP2003 Conference*, June 2003, San Diego CA, USA.