2013

Octree-based Indexing for 3D Point Clouds within an Oracle Spatial DBMS

Bianca Schoen-Phelan
Technological University Dublin, bianca.schoenphelan@tudublin.ie

Abu Saleh Mohammad Mosa
University College Dublin

Debra Laefer
University College Dublin

See next page for additional authors

Recommended Citation

Schoen-Phelan, B., Mosa, A., Laefer, D. & Bertolotto, M. (2013).Octree-based Indexing for 3D Point Clouds within an Oracle Spatial DBMS.*Computers & Geosciences*, 51, 430-438. doi:10.1016/j.cageo.2012.08.021

Octree-based Indexing for 3D Point Clouds within an Oracle Spatial DBMS

Bianca Schön[a], Abu Saleh Mohammad Mosa[b], Debra F. Laefer[c,1], Michela Bertolotto[b]

[a] National Centre for Geocomputation, National University of Ireland Maynooth, Ireland

[b] School of Computer Science & Informatics, University College Dublin, Belfield, Dublin 4, Ireland

[c] School of Architecture, Landscape & Civil Engineering, University College Dublin, Belfield, Dublin 4, Ireland

Abstract

A large proportion of today's digital data has a spatial component. The storage and effective management of such data poses particular challenges. Nowhere is this truer than with Light Detection and Ranging (LiDAR), where datasets of even small geographic areas may contain several hundred millions points. Currently available spatial information systems do not provide suitable support for 3D data. As a consequence, while a specific system can be used to store the data, another has to be used to process it. Typically several software applications are used to analyze and process LiDAR data, requiring multiple format transformations. Our work aims at providing a more cost-effective way for managing LiDAR data that allows for the storage and manipulation of these data within a single system. We achieve this by exploiting current Spatial Database Management technology. In order to provide an efficient solution, suitable 3D indexing mechanisms are essential. In this paper we describe the implementation of an octree index for 3D LiDAR data atop Oracle Spatial 11g and present a comprehensive evaluation that shows its efficient performance as compared to the native Oracle R-tree index.

1. Introduction

[1] Corresponding Author, Email: Debra.Laefer@ucd.ie, Tel.: +353-1-716-3226, Fax: +353-1-716-3297

The proliferation of point cloud data from both terrestrial and aerial laser scanning coupled with large-scale efforts to provide remote sensing data for web-based, multiple-user access have become strong dual motivators to reconsider current storage strategies for 3D point cloud data. Point cloud datasets provide significant management challenges because of their size. As an example, typical low-density aerial scanning generates 30-50 point/m^2 (for the State of North Carolina a 2001 flood plane mapping generated approximately 5.6 billion points for the entire dataset). More recent flyovers report significantly higher densities of up to 225/m^2 resulting in 225 million points for a single square kilometer (Hinks et al., 2009).

Currently available spatial information systems do not provide suitable support for 3D data. This means that while a specific system can be used to store the data, another has to be used to process it. Typically several software applications are used to analyze and process LiDAR data, requiring multiple format transformations with consequent loss of accuracy. Our work aims at providing a more efficient and cost-effective way for managing LiDAR data that allows for the storage and manipulation of these data within a single system. Our solution exploits current Spatial Database Management (SDBMS) technology and its extensibility capabilities that allow developers to implement additional functionality within such technology. Indeed, while extensive support for 2D data has been made available by several DBMS vendors (including Postgres and Oracle), very limited support is currently provided for 3D data handling. In particular, in order to achieve efficiency, suitable 3D indexing mechanisms are essential. Oracle is currently the only SDBMS that provides native 3D spatial data types and the implementation of a 3D index. However, such an index is based on R-trees and is not adequately efficient for 3D point cloud data.

This paper shows how octree-based indexing can greatly facilitate the storage and indexing of 3D point cloud data within an SDBMS. We present an implementation of the octree index

47 (Laefer et al., 2009) atop Oracle Spatial 11g and show that it outperforms the native R-tree index
48 provided by Oracle.

49 This paper is structured as follows: Section 2 discusses current SDBMS support for 3D data,
50 including indexing approaches. Section 3 describes the implementation of an octree index within
51 an Oracle Spatial 11g database. Section 4 presents an evaluation of the octree index that com-
52 pares its performance with that of the Oracle R-tree. Section 5 provides a critical discussion of
53 the results presented in this paper and an outlook for future efforts in this area of research.

54 ## 2. Background and Technologies

55 This section outlines the various technologies and approaches that are currently used in order to
56 index 3D point cloud data. First, support of SDBMSs is investigated, which is followed by an
57 overview of research approaches in this area. Then, the approach implemented within this paper
58 is explained. An evaluation of an experiment assessing this approach is presented in section 4.

59 ### 2.1. SDBMS support for 3D Point Cloud Data

60 Today, many of the benefits of high-resolution LiDAR remain relatively unexploited as the data
61 cannot be efficiently managed in a traditional Geographical Information System (GIS), because
62 of the current inability of GISs to fully support 3D objects. For example, GIS systems are not
63 designed to support Finite Element Meshes, which are often the intended end product for a Li-
64 DAR scan. A desirable alternative would not be file-based (due to the very large size of datasets)
65 and would overcome the need for multiple programs with repeated import and export transac-
66 tions (to eliminate the potential loss of LiDAR's accuracy through format conversions). The pro-
67 posed solution in this paper integrates all required functionality within a SDBMS.

A Database Management System (DBMS) controls the organization, storage, management and retrieval of all data that is kept in a database. A DBMS ensures that data inconsistencies and data redundancies are significantly reduced compared to storing information in a file system. A DBMS also facilitates data integrity, as well as multi-user control on shared data. Initially, traditional DBMSs did not support storage and querying of spatial data (i.e. data with a spatial component). Later, an integrated approach was developed to store the spatial extent (together with the attribute data) directly into the database (in the same table). Current SDBMSs, such as Oracle Spatial or PostGIS are based on the extensibility of Relational Database Management System (RDBMS), i.e. the ability to add new types and operations. This technology allows for the management of all data within the same engine. Additionally, data retrieval and manipulation are facilitated through Structured Query Language (SQL).

For a spatial system to be fully 3D, it must support 3D data types, such as point, line, surface, and volume in 3D Euclidean space. Three-dimensional data types are based on a 3D geometric data model (i.e. vector and/or raster data with underlying geometry and topology). A 3D spatial system must also offer operations and functions embedded into its query language that can operate with its 3D data types (Bruenig & Zlatanova, 2004). Until recently, SDBMSs have not provided support for 3D data management; an extensive review of this is available elsewhere (Schön et al., 2009a). However, with Oracle Spatial's release of 11g, 3D point clouds can be stored in an in-built data type. Previously Oracle Spatial relied heavily on SDO_GEOMETRY. Now SDO_PC is the main data type employed for the storage of 3D point cloud data. SDO_PC is designed for the storage of multi-dimensional point clouds. There is no upper bound on the number of points in an SDO_PC object. A set of points are grouped and stored as the BLOB object in a row. However, the current version of Oracle Spatial offers only a limited amount of placeholders

for the storage of information alongside locational attributes, as only nine attributes can be stored together in one element within SDO_PC (Murray, 2009). Another disadvantage is that Oracle Spatial does not yet offer functionality to update SDO_PC objects. Consequently, the SDO_GEOMETRY data type remains highly useful for the storage of any geometry type, including 3D data points. Particularly considering 3D point clouds, it is desirable to store the locational information together with its attribute information (e.g. color, intensity) in the same table, as semantic information oftentimes directs feature recognition processes, which are typically applied onto the data at a later stage in the work flow.

Indexes are employed in order to avoid traversing a complete table when performing spatial queries. Thus, indexes are used to organize the space and the objects within this space. Given the sheer size of aerial LiDAR datasets, efficient indexing mechanisms are essential. Spatial indexing techniques evolved in the mid-1980s, with Guttman's R-tree (Guttman, 1984) being one of the most popular and enduring indexing techniques. Indexing techniques are discussed in the next section.

2.2. Indexing of 3D Point Cloud Data

Stanzione and Johnson (2007) argue that a tree structure is inherently more efficient due to its binding with the internal data storage structure. In this spirit, various tree structures have been explored for indexing vast point cloud data. One approach is based on combining R-trees with an importance value (Oosterom, 1990), which is called V-reactive tree (Li, 2001). The V-reactive tree is an R-tree structure in 4D, which is optimized for 3D visualization. However, to date, the structure has not been tested for realistically large point cloud datasets. Hua et al. (2008) proposed a hybrid approach for visualization combining an octree with a k-d tree (Bentley, 1975) by building a local k-d tree at each octree level node. This approach has only been evaluated for

visualization speed of 3D point clouds for up to 100,000 points (Hua, 2008). De Floriani et al. (2008) extended quadtree indexes to work with TIN structures and argued that their mechanism could be generalized to support Tetrahedral Irregular Networks (TENs) on an octree basis in order to support true 3D functionality.

Hierarchical space division based structures (e.g. octree) are critical for 3D surface representations and queries as they are purely volume based. Combined approaches, such as the Volume-Surface tree (V-S tree) aim to avoid a strong imbalance with regards to clustering of points by applying a 3D octree on a global level and a 2D quadtree on a local level (Boubekeur, 2006). However, this method has a tendency to collapse in cases where the surface is not smooth, which results in pure octree indexing (Velizhev & Shapovalov, 2008). Another interesting approach for indexing LiDAR is based on the Hilbert Space-Filing curve (Wang & Shan, 2005). Space-Filing curves (Sagan, 1994) preserve spatial proximity at local level and map points in n-dimensional space into linear order. This approach has been implemented in the MySQL and the Microsoft Access Database for evaluation purpose and tested on 1.4 million LiDAR points from a terrestrial scan of a bridge structure (Wang & Shan, 2005). Currently, Microsoft Access does not provide any spatial support. While MySQL Spatial offers rudimentary spatial support by providing spatial data types, functions and a spatial index. However, due to the limited amount of spatial functions, this database is best used for simple retrieval by bounding box operations.

Presently, Oracle Spatial provides an R-tree based spatial index and a deprecated two-dimensional (2D) quadtree (Murray, 2003). The 2D R-tree is based on minimum bounding rectangles (MBRs), and the 3D extension consists of minimum bounding boxes (MBBs). How to implement a bounding box on a dense point cloud is non-trivial, and sometimes inefficiencies develop due to the overlap of sibling nodes and the uneven size of nodes (Zhu, 2007). An alter-

native approach is to map spatial objects onto one-dimensional space to enable the use of a standard index, such as a B-tree (Bayer, 1971).

In Oracle Spatial, only an R-tree index can be created in 3D on geometry columns; the R-tree implementation is the successor for HHCode in Oracle Spatial (Murray, 2009). However, it only supports one operator in 3D, called SDO_FILTER (Kothuri, 2007, p. 272). This operator performs only a primary filter operation, which identifies all rows of a table where Minimum Bounding Rectangles (MBRs) of the column geometry intersect with the MBRs of the query geometry (Kothuri, 2007, p. 269). As such, this operator returns the superset of results for all spatial queries and cannot be used for a particular spatial query. All other presently implemented operators work only for two-dimensional (2D) geometries. **THIS PARAGRAPH WAS IN THE PREVIOUS SECTION BUT IT DID NOT FIT SO I MOVED IT HERE BUT IT IS NOT LINKED WITH WHAT COMES EARLIER. DO WE NEED IT TO JUSTIFY HOW WE DID THE EXPERIMENTS? If not I would remove it.**

Personally, I think that we can delete this paragraph. It is mainly a response to one of the reviewer's comments regarding HHCode. I don't think it is important and it doesn't really fit anywhere anyway.

PostgreSQL supports the Generalized Search Tree (GiST) index (Geo-Consortium, 2007) which is a "template data structure for abstract data types" that offers more robust support for spatial indexing.

Several strategies have been developed for efficient indexing of multi-dimensional data. Although there is limited vendor support for these, and true 3D index creation is still an ongoing research problem (Schön, 2009b). In most cases, indexes only support two-dimensionality with

159 simple 3D extensions (Arens, 2005). An octree offers an alternative, but currently no SDBMS
160 support octree indexing and to the best of the authors' knowledge no meaningful benchmarks
161 have been provided thus far on this approach. This is where this paper seeks to make a signifi-
162 cant contribution.

163 **2.3. Octree Indexing for Spatial 3D Point Cloud Data**

164 An octree structure offers distinct advantages over the frequently implemented R-tree in particu-
165 lar regarding the indexing of LiDAR datasets. Octrees can index point geometries directly, as
166 opposed to the R-trees that solely rely on bounding boxes. Furthermore, octrees generate dis-
167 jointed, non-overlapping tree nodes, whereas R-tree bounding boxes are often overlapping,
168 which reduces query efficiency in the case of the R-tree. Moreover, storing the logical tree struc-
169 ture into a SDBMS is complex. The tree structure can be stored in a table where each node of the
170 tree structure corresponds to a row in the table. In that specific case, one column is needed to
171 store node identifiers (nodeID) and another to store the list of node identifiers (nodeIDs) as
172 pointers to the child nodes. The node identifier of the root node can be stored in a table, called
173 the metadata table for that index. Oracle Spatial's R-tree index implementation stores the tree
174 structure in a table and selects a node using an internal SQL statement, as each node is visited
175 (Kothuri, 2002). Thus, query operations involve the processing of many recursive SQL state-
176 ments, which consequently increases query processing times (Kothuri, 2002). The octree, on the
177 other hand, can be used to divide the entire space according to a specified tiling level. In this
178 case, only the tiling level needs to be stored as the tree structure can be rebuilt during the query
179 processing; details are described subsequently.

180 A further advantage of the octree lies in its support for optimized 3D point cloud visualization
181 (Koo and Shin, 2005). Rendering of 3D point clouds is computationally expensive and rendering

from an SDBMS causes further delays due to I/O operations on the DBMS. However, an octree can be utilized to filter visible points for rendering according to a specific view frustum, instead of rendering all points in the dataset at once. Nonetheless, the selection of an appropriate spatial index depends on many factors, such as data distribution and data type. Octrees provide a more specific approach applicable to all 3D point cloud object types. The following section outlines how an octree index can be implemented in Oracle Spatial.

3. Design of an Octree Index atop Oracle Spatial 11g

Oracle's Extensibility Framework requires that a data cartridge be implemented, in order to provide a new index structure (Belden, 2008). Data cartridges are re-usable server based components, which utilize object types and features such as large objects (LOBs), external procedures, extensible indexing, and query optimization. Oracle's extensible indexing framework defines a set of interface methods. These must be implemented in an object type, which is called indextype (Belden, 2008). An indextype is an object that specifies the routines that manage a domain (application-specific) index. It has two major component sets: (1) methods that implement the index's behavior and (2) operators that the index supports.

This paper presents a new data cartridge implemented in Oracle's extensible indexing framework that enables octree indexing and is in the following sections referred to as OCTREEINDEX. To facilitate the analysis of 3D point clouds, a window query operator OT_CLIP_3D was also implemented, which performs a window query on a given 3D point geometry stored in an Oracle SDO_GEOMETRY data type. Spatial metadata information is stored in the USER_SDO_GEOM_METADATA view provided by Oracle Spatial (Kothuri, 2007, p. 45). The following presents the index and related window query operator implementation.

3.1. Implementation of the Octree Index

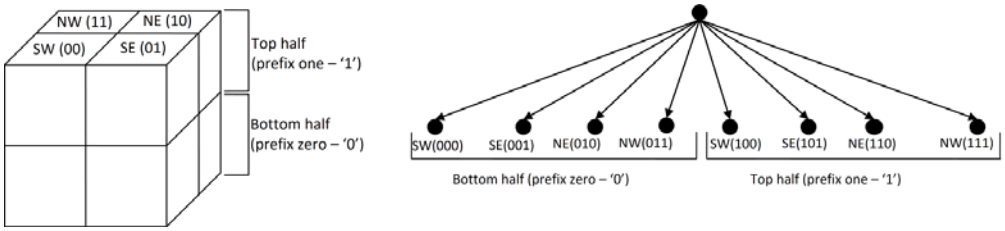
An octree's structure dictates that each internal node contains exactly eight child nodes regardless of its many variants (Samet, 2006). This paper uses a region octree, where the space is decomposed into cubic blocks (or cells) through recursion, until a block is homogeneous. The approach is oriented to interior-based representations for 3D region data, which permits further aggregation of identically valued cells.

MICHELA: add a sentence here about providing a compromise – the trick is to find a suitable tiling level.

Comment [bs1]: I am not sure how this is meant

By definition, an octree can result in an unbalanced hierarchical tree when the data distribution is not uniform. However, with regards to its implementation this harbors a distinct challenge, as it would require storage of its logical tree structure in a SDBMS for reconstruction of the tree structure during query processing. This could introduce inefficient query processing due to the issuance of several internal recursive SQL select statements generated during each node visitation. The approach presented in this paper resolves this issue by constructing a balanced tree structure up to a fixed tiling level; an example is provided in the implemented approach and is described further below. In this case, only the tiling level information (as opposed to the whole tree) needs to be stored for tree reconstruction. The selection of an appropriate tiling level for a specific dataset is a decisive factor, which involves the dataset's area and size. As such, this is a drawback of this approach, as experimentation with different levels is needed in order to optimize performance for a specific dataset. Particulars of this problem are further illustrated in section 4. The user can specify the tiling level through the parameter OCTREE_LEVEL during index creation. Each cell is associated with a unique code, which is herein referred to as the cell code. The cell code is obtained by using z-ordering (i.e. Morton encoding) of all cells at the spec-

ified level (Morton, 1966). Fig. 1 illustrates the decomposition of space on the example of a 2D quadtree, as this is easier to illustrate graphically. The octree functions analogous in 3D. Fig. 1(a) illustrates the 3D space decomposition and fig. 1(b) illustrates the cell code generation. All cells in the bottom half are assigned with the prefix '0' – zero, and all cells in the top half are assigned with prefix '1' – one. The cells are marked south-west (SW), south-east (SE), north-east (NE) and north-west (NW) and associated codes are 00, 01, 10 and 11 consecutively. The associated cell code with each point is identified by traversing the octree structure from root node to leave node. For example, using B to represent the bottom half and T to designate the top half, at tiling level 5, the code for the path BNW (011) – TSW (100) – TNE (110) – BSE (001) – BSW (000) is 011100110001000. Here, it only follows the tree path where the cell associated to a node in the path contains the point. The ROWID of the point and the associated cell code are stored in an index storage table. The metadata (e.g. tiling level, index name, index owner, max level, min level, etc.) for the entire index are stored as a row in a table called index metadata table.



(a) 3D space decomposition. (b) Cell code generation.

Figure 1. Quadtree Sectors.

The 3D query processing using this implementation is illustrated in fig. 2. To generate the result set for a spatial query, the octree index is used as the primary filter to find the area of interest or candidate geometries for this query. Figure 3 illustrates the use of a primary and secondary filter during the query process. The area of interest is the sum of the cells of the octree that interact spatially (e.g. intersect, touch, inside, covered by) with the query geometry, as established by the

primary filter. These cells are identified by the cell code, and candidate geometries are identified by the associated cell code from the index storage table. These candidate geometries are passed through the intermediate filter and divided into two sets. Cells inside or covered by the query geometry are identified as an exact match. The points associated with these cells are sent directly to the result set. The remaining cells (those that intersect or touch the query window) are identified as unknown and passed through the secondary filter. The secondary filter is a spatial function, which corresponds to the spatial query.

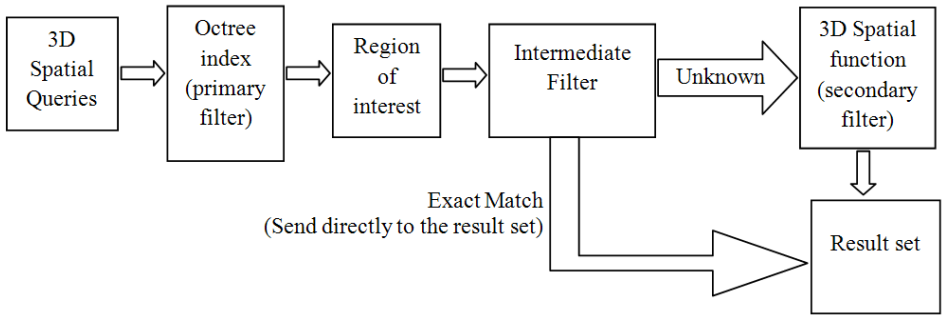


Figure 2. Query Processing Steps

The Oracle Extensibility Framework requires that a new index must implement a certain interface and related methods. The name of the interface is ODCIIndex (Belden, 2008). Associated methods are categorized into four classes: (1) index definition methods, (2) index maintenance methods, (3) index scan methods, and (4) index metadata method. Table 1 summarizes these methods with implementation details described henceforth.

Table 1. ODCIIndex Interface Methods

Category	**Method Name**	**Invoked by**

Definition Methods	ODCIIndexCreate()	"CREATE INDEX" statement
	ODCIIndexDrop()	"DROP INDEX" statement
	ODCIIndexAlter()	"ALTER INDEX" statement
Maintenance Methods	ODCIIndexInsert()	"INSERT INTO" statement on the base table, which involves the indexed column.
	ODCIIndexUpdate()	"UPDATE" statement on the base table, which involves the indexed column.
	ODCIIndexDelete()	"DELETE FROM" statement on the base table, which involves the indexed column.
Scan Methods	ODCIIndexStart()	At the beginning of an index-scan.
	ODCIIndexFetch()	In order to fetch the row identifiers those satisfies the operator predicate.
	ODCIIndexClose()	At the end of the index-scan In order to perform cleanup.
Metadata methods	ODCIIndexGetMetadata()	In order to write implementation-specific metadata into the export dump file using "Export" utility.

An implementation type is required to create the indextype OCTREEINDEX and must contain the implementation of the ODCIIndex interface methods. An object type known as the implementation type and named OCTREE_IM is defined to implement the ODCIIndex interface methods (Belden, 2008). It contains the signature and return type of the interface methods.

The body of OCTREE_IM contains the implementation of these, which can be implemented using PL/SQL, C, C++ or Java. In this implementation, only the ODCIGetInterfaces method is implemented in PL/SQL, while others are implemented as Java callouts, which resides in a Java class. A previously available Java API was exploited (Kothuri, 2007, p. 223). It enables applications written in Java to access and process geometry objects managed in Oracle database with Oracle Spatial. OCTREE_IM contains only the implementation of the method ODCIGetInterfaces, while others are implemented in a Java class entitled OctreeIndex. The mapping of the interface methods to the Java methods is defined in OCTREE_IM.

The process of index creation is outlined below. Other methods implemented for the prototype, as explained in Table 1, are created accordingly. Fig. 3 illustrates index creation process, and fig. 4 illustrates the requisite steps.

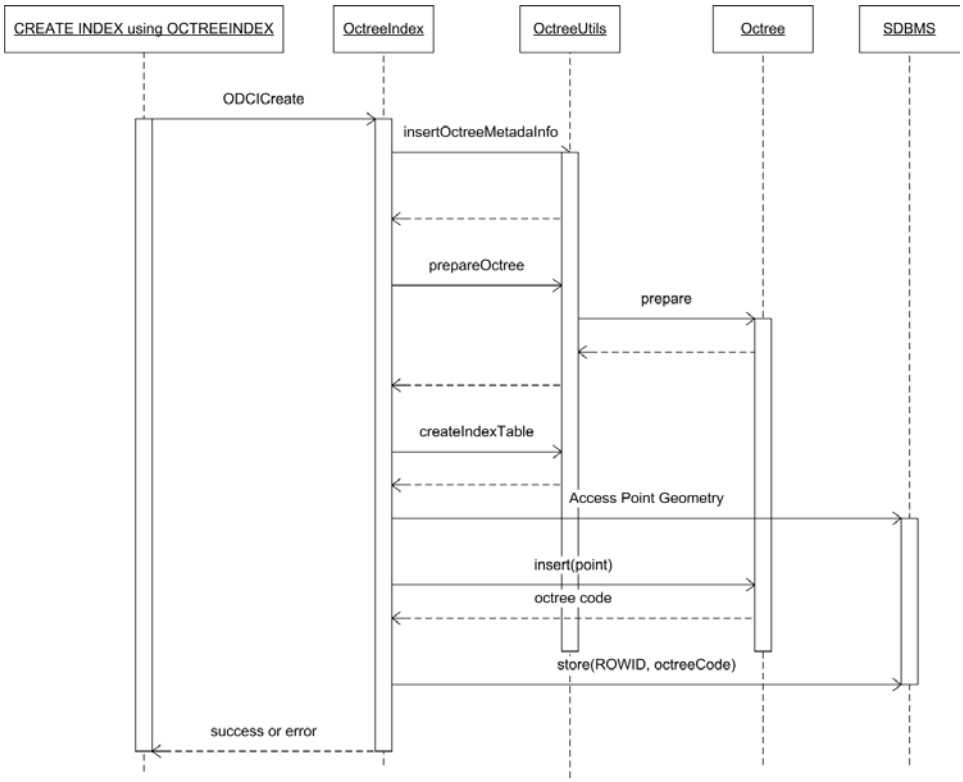


Figure 3. Index Creation Sequence

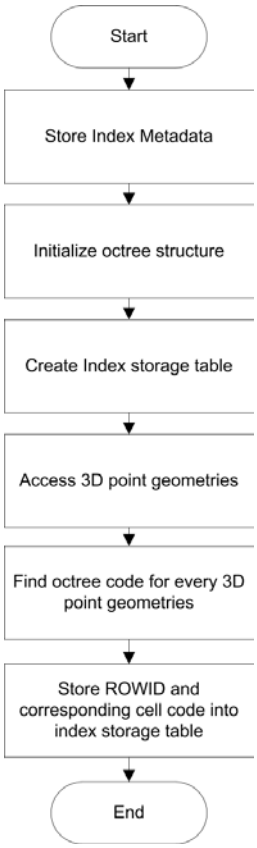


Figure 4. Steps needed for Index Creation

The ODCICreate method is invoked when a user issues the “CREATE INDEX” SQL statement of indextype OCTREEINDEX. This starts the index creation process. At first, metadata information regarding the index is stored into an index metadata table named OCTREE_INDEX_METADATA. Next, the octree structure is initialized, and the 3D bound of the 3D point cloud sample as a whole is created. The 3D points that are stored as point geometry data types are accessed from the base table through the Java Database Connectivity (JDBC) con-

nection with the database. These are inserted into the octree structure, which returns the cell code for each point.

3.2. Implementation of an Operator

Window queries are among the most commonly used first-step-analysis operations for LiDAR data. This query has been implemented and is referred to as OT_CLIP_3D. The operator returns all point geometries that are inside and on the boundary of the specified 3D cube and takes two SDO_GEOMETRY objects as input. The first input is a 3D point geometry or a column of the type SDO_GEOMETRY that contains a 3D point geometry on which the operator is applied. The second input is a simple solid of type SDO_GEOMETRY, which specifies the query window. Every operator must be tied to an index for index-based evaluation. Oracle's extensible indexing framework requires the implementation of index scan methods to evaluate the operators. These are ODCIIndexStart, ODCIIndexFetch and ODCIIndexClose. Fig. 5 illustrates the invocation sequence of index scan methods.

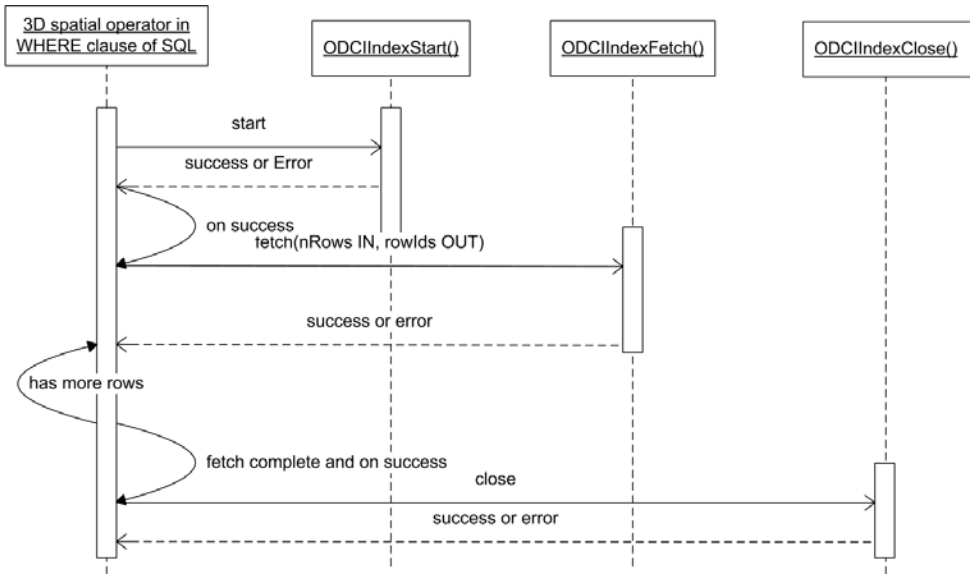


Figure 5. Index Invocation

At first, the interface method ODCIIndexStart is invoked by Oracle with the operator name, arguments, and the lower and upper bounds describing the predicate. This method is invoked to begin the operator evaluation. A series of fetches are performed by invoking the ODCIIndexFetch method to obtain row identifiers or rows that satisfy the operator predicate. The number of expected rows (nRows) in every fetch is specified by Oracle during each invocation of this method. The ROWIDs are placed into the placeholder array (rowIds). Finally, before the destruction of the SQL cursor, ODCIIndexClose is invoked by Oracle to end the processing of the operator. Figure 6 illustrates the window query performed on a 3D point cloud. The result set returns all point geometries that are inside or on the boundary of the query window. In this example, all the square points are the ones inside or on the boundary of the query window (illustrated by the box of dotted lines).

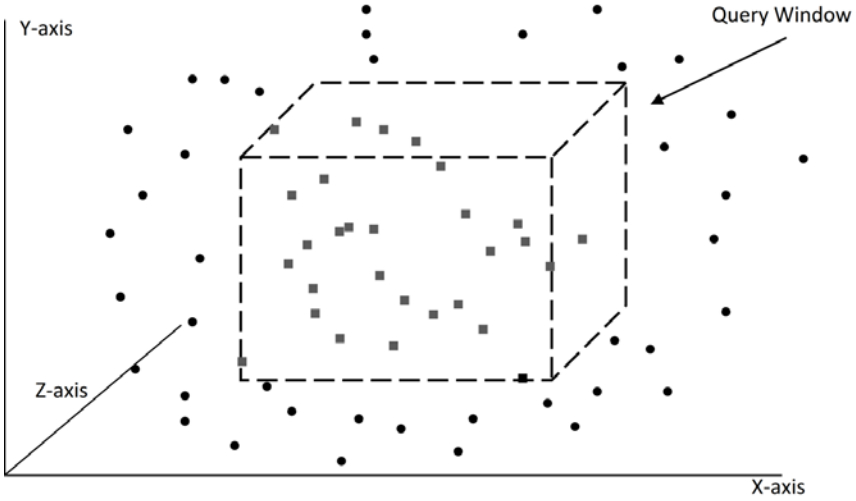


Figure 6. Query Window.

The OT_CLIP_3D operator is evaluated through the octree index. Figure 7 demonstrates the evaluation process. For ease of illustration, the example is shown as a 2D case. The query window is drawn in dotted lines and the resulting geometries as solid squares. The octree is traversed in order to identify cells that interact or are topologically related with the query window. Possible topological relations are “inside”, “intersect”, “touch”, and “covered by” (Egenhofer & Franzosa, 1991).

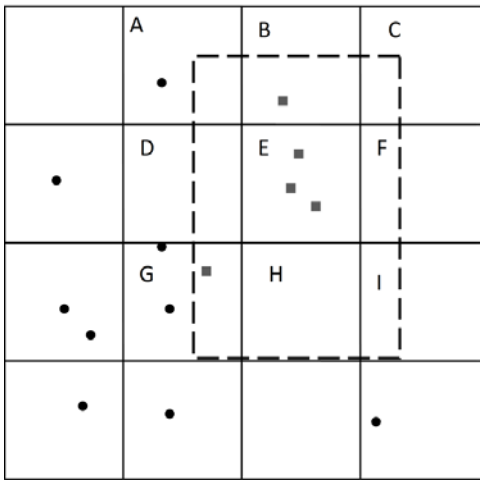


Figure 7. Octree Query Window

Blocks that are (1) inside the query window or (2) intersect with the query window or (3) touch the query window, or (4) are covered by the query window are identified, and the area of interest is the union of these blocks. This area is searched in order to generate the result set for this query. In fig. 7 blocks labeled A, B, C, D, F, G and I intersect with the query window. Block E is inside the query window, and block H is covered by the query window.

The intermediate filter is used to identify any exact match, such as block E and H. These blocks are inside the query window, which implies that all the points belonging to these blocks are also inside the query window. These points are sent directly to the result set and the blocks are labeled as known regions. Points belonging to this region are also labeled as known. The points that are covered by the other blocks are passed through the secondary filter. Their points are labeled as unknown and require further processing.

This section presented a comprehensive description on the octree implementation on top of Oracle Spatial 11g, which is expected to provide a more effective mechanism for the storage of 3D LiDAR point clouds. The following section presents a study that benchmarks this approach.

4. Evaluation

This section provides an evaluation of the implementation of octree indexing that has been presented in the previous sections. In particular, window query response times on a 3D LiDAR point cloud dataset are compared between R-tree and octree indexing. The evaluation has been conducted on a computer with the Intel Core2 Duo CPU 2.53GHz and 4GB RAM, 7,200 SATA hard drive using Oracle 11g release 11.1.0.6.

The 3D LiDAR point cloud dataset is stored in Oracle's SDO_GEOMETRY data type. In order to perform spatial queries on 3D point cloud data, the dataset is indexed using the R-tree and the octree index. Two randomly selected datasets from a dense aerial LiDAR flyover of Dublin's city centre (Hinks et al., 2009) were selected. One contained nearly 2,9 million points and the other almost 66 million points. Query response times are compared for the two index types for a variety of window sizes. The R-tree index was created using Oracle's existing, in-built spatial index. An octree index structure is implemented in Oracle's extensible indexing framework, as described in section 3.1.

In Oracle Spatial, the R-tree index supports only one operator with which a 3D spatial query can be performed. However, this operator does not provide window query functionality. Consequently, in order to perform a window query on a LiDAR dataset, which is one of the most common first-step analyses for point cloud datasets, a 2D index was created in order to allow for a 2D window query. It is assumed that since the 2D R-tree index is created on the 3D point clouds

that the index is created on the 2D projection of the 3D point cloud data. The SDO_RELATE operator provides functionality similar to a general window query. The "inside and touch" masks (Kothuri, 2007, p. 274) and the 2D query window are specified to perform the window query.

The implemented octree index supports the operator OT_CLIP_3D, which performs a 3D window query on 3D point cloud data. This operator is used to execute 3D window query on 3D LiDAR point cloud data. To create the octree index, it is very important to determine the tiling level for efficient query processing. For this purpose, the total number of points in the table and the area of the dataset are considered. The total number of indexed points per cell and cell volume are decreased as the tiling level increases, which in turn decreases the total number of candidate geometries. On the other hand, the total number of leaf nodes increases. For example, the total number of leaf nodes at tiling level 'N' is 8^N. As such, memory consumption increases at higher tiling level during manipulation of the octree structure. Tiling level five (5) was experimentally selected for this dataset. Further work will be conducted for automatic tiling level determination. This will in part be dependent upon the uniformity of the dataset. If terrestrial data is incorporated selectively in a larger aerial set, the result may be highly non-uniform.

In this example, a fairly uniform aerial LiDAR dataset was used as it represents a portion of Dublin's city centre in Ireland, where relatively few large occlusions exist as the average building height is low. With this dataset, the response time of a 2D window query (x- and y-coordinates) using an R-tree index is compared with the response time of a 3D window query (x-, y- and z-coordinates) using an octree index. Basically, the query response time increases with an is expected to increase query window size, as well as the number of resulting geometries (Kothuri, 2002). For that reason, the same number of resulting geometries for a window query using

both indexes is considered. To ensure this, in case of using an octree index, the maximum and minimum value of the z-coordinates of the query window is set to the minimum and maximum value of the underlying space. The same value is used for x- and y- coordinates for the octree and R-tree index. As a result, the total number of resulting geometries is equal for both indexes.

Table 2 presents the average query response time with the increase of the window size using the R-tree and the octree index. A square rectangle was used as a query window. For every window size up to 625 queries were performed around the underlying space, and the query response time is the average of these queries. Fig. 8 and 9 illustrate the response times. The octree index nearly consistently outperforms the R-tree index for all window sizes.

The dataset used in fig. 9 is about 23 times the size of the dataset in fig. 8. In fig. 8, the octree is twice as fast as the R-tree for the small window of 25m^2 and 8 times faster for the large window of 2,500m^2 in size. In fig. 9, for the small window of 400m^2 size, the R-tree outperforms the octree, but once the window reaches 1,600m^2, the octree is better, with a six-fold improvement for a 40,000m^2 window.

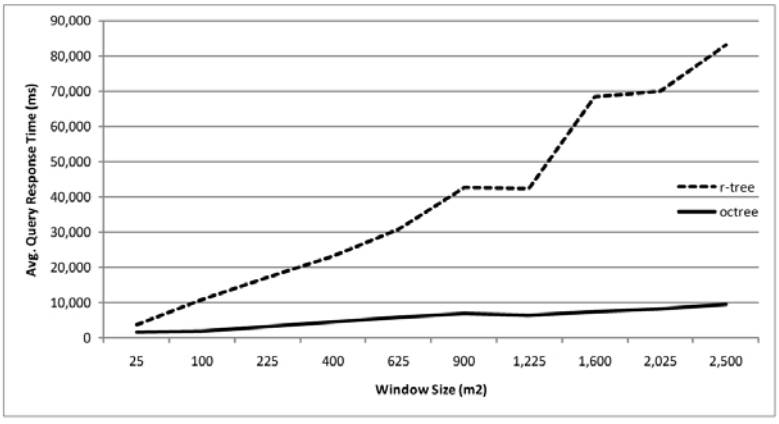


Figure 8. R-tree vs. Octree 2,881,899 points in the dataset

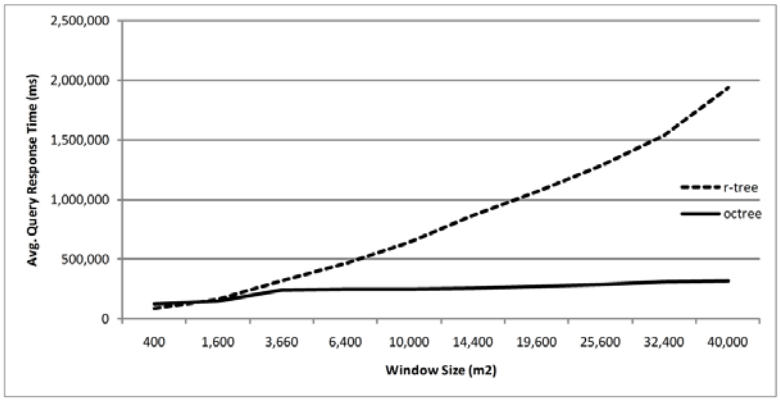


Figure 9. R-tree vs Octree 65,562,235 million points in the dataset

Table 2. Evaluation Results

Small Dataset of 2,881,899 Million Points			Large Dataset of 65,562,235 Million Points		
Window Size (m2)	**Avg. Query Response Time in ms (R-tree)**	**Avg. Query Response Time in ms (Octree)**	**Window Size (m2)**	**Avg. Query Response Time in ms (R-tree)**	**Avg. Query Response Time in ms (Octree)**
25	3,720.40	1,686.28	400	83,026.65	128,814.14
100	10,868.86	1,975.92	1,600	166,708.00	149,541.69
225	17,170.21	3,121.44	3,660	321,180.30	243,061.55
400	23,269.12	4,628.71	6,400	467,678.50	245,920.08
625	30,816.40	5,804.15	10,000	641,871.10	250,993.00
900	42,541.17	6,934.25	14,400	864,345.50	257,525.44
1,225	42,277.08	6,329.17	19,600	1,065,853.90	269,746.34
1,600	68,354.84	7,521.83	25,600	1,281,446.50	286,461.88
2,025	70,115.16	8,328.33	32,400	1,535,893.00	310,641.75
2,500	83,238.25	9,462.75	40,000	1,933,097.50	321,632.50

5. Conclusions and Summary

This paper presents the implementation and evaluation of an octree index, intended for 3D point cloud data from laser scanning, employing Oracle's extensible indexing framework. However, its functionality may be cross-applicable to other point cloud datasets and implementable in other SDMS as they expand their 3D capabilities. The goal is to improve the support of the storage and

410 analysis of point cloud data in order to enable the integration of multiple datasets. The intention
411 is to significantly improving query capabilities for users from different discipline-based needs,
412 hence the ability to store the actual raw LiDAR data has enjoyed increasing focus in recent years.
413 To this end, an operator using our octree index has been implemented to perform 3D window
414 queries. This implementation is described along with some optimizations. The newly implement-
415 ed octree index and Oracle's inbuilt R-tree index are compared using data from a dense, aerially-
416 based 3D point cloud. The octree consistently outperformed the R-tree for almost every window
417 size and more so with increases in query window size to as much as an eight-fold difference. The
418 considerably improved performance, while notable in itself, needs to be considered further in
419 light of the additional functionality offered by the octree in terms of a more appropriate storage
420 and indexing of point cloud data in particular. As such, groupings into appropriate cells may oc-
421 cur according to a predefined semantic, such as for example color, intensity or elevation infor-
422 mation. Furthermore, the approach is not plagued with the R-tree's related uncertainty when try-
423 ing to select a bounding box for point geometries, as the data in the octree is processed directly
424 instead of as a derived object.

425 Since only one operator is implemented, further work envisions a more comprehensive evalua-
426 tion in order to assess the octree's full potential for other query operators, such as nearest neigh-
427 bor or within distance. In this prototype, tiling level is determined by the user for a dataset. Fur-
428 ther work will enhance the prototype by incorporating a feature for automatic tiling level deter-
429 mination, along with exploitation of the visualization-based efficiencies that this approach will
430 engender. In Oracle Spatial, the new SDO_PC data type applies an R-tree index only to the
431 groups of clusters that contain the point geometries. An alternative approach to the one presented
432 in this paper may rely on a two-step index, where an octree index is applied to the points inside a

block, and an R-tree is applied as a higher level index to the block extents as polygons are better indexed by R-tree. There may also be specific cases where the converse of ordering proves advantageous with an octree over an R-tree. Future work will further evaluate these options.

Much need has been expressed for the web-dissemination of LiDAR data. The earlier mentioned UCSD example provides non-raw data through a web interface. The hope is that through better storage and indexing of 3D point cloud data, web dissemination of substantial raw LiDAR datasets will not remain a feature of the far future.

Acknowledgement

This work was generously support by Ireland's National Digital Research Centre's grant EoI/0701/008. Data for this work was provided by Science Foundation Ireland's sponsored grant 05/PICA/I830.

References

Arens, C., Stoter, J., & Oosterom, P. van (2005). Modelling 3D Spatial Objects in a Geo-DBMS using a 3D primitive. Computers & Geosciences, 31(2), 165-177.

Bayer, R. (1971). Binary B-trees for Virtual Memory. Proceedings of the 1971 ACM SIGFIDET Workshop on Data Description, Access and Control (pp. 219-235). San Diego, California: ACM.

Belden, E., Chorma, T., Das, D., Hu, Y., Kotsovolos, S., Lee, G., et al. (2008). Oracle Database Data Cartridge Developer's Guide, 11g Release 1 (11.1).

Bentley, J.L. (1975). Multidimensional Binary Search Trees Used for Associative Searching. Communications of the ACM, 18(9), 509-517.

453 Boubekeur, T., Heidrich, W., Xavier, G., & Christophe, S. (2006). Volume-Surface Trees. Eu-
454 rographics, 25(3), 399-406.

455 Bruenig, M., & Zlatanova, S. (2004, November 6). 3D Geo-DBMS. Directions Magazine. Re-
456 trieved March 2010, from http://www.directionsmag.com/article.php?article_id=694 (accessed
457 Sep 1, 2010)

458 Egenhofer, M.J., & Franzosa, R.D. (1991). Point-Set Topological Spatial Relations. International
459 Journal of Geographical Information Systems, 5(2), 161-174.

460 De Floriani, L., Facinoli, M., Magillo, P., & Debora, D. (2008). A Hierarchical Spatial Index for
461 Triangulated Surfaces. International Conference on Computer Graphics Theory and Applica-
462 tions, (pp. 86-91).

463 Francica, J. (2007, August). Informix Spatial Data Technology: Update and Positioning. Re-
464 trieved March 2010, from Directions Magazine:
465 http://www.directionsmag.com/article.php?article_id=2520 (accessed Sep 1, 2010)

466 Geo-Consortium. (2007). Introduction to Spatial Data Management with PostGIS. Presentation
467 Slides by the Consulting Centre Geographic Information Systems.

468 Guttman, A. (1984). R-trees: A Dynamic Index Structure for Spatial Searching. Proceedings of
469 the 1984 ACM SIGMOD International Conference on Management of Data (pp. 47–57). ACM
470 NY, USA.

471 Hinks, T., Carr, H., & Laefer, D.F. (2009). Flight Optimization Algorithms for Aerial LiDAR
472 Capture for Urban Infrastructure Model Generation. Journal of Computing in Civil Engineering,
473 23(6), 330-339.

Hua, L., Zhengdong, H., Qingming, Z., & Peng, L. (2008). A database approach to very large LiDAR data management. The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, XXXVII Part B1 Commission I, pp. 463-468. Beijing.

Koo, Y.-M., & Shin, B.-S. (2005). An Efficient Point Rendering Using Octree and Texture Lookup. Computational Science and Its Applications-ICCSA 2005, 3482, 1187–1196.

Kothuri, R.K., Ravada, S., & Abugov, D. (2002). Quadtree and R-tree Indexes in Oracle Spatial: a Comparison Using GIS Data. ACM SIGMOD International Conference on Management of Data (pp. 546-557). Wisconsin, USA: ACM.

Kothuri, R., Godfrind, A., & Beinat, E. (2007). Pro Oracle Spatial for Oracle Database 11g.

Laefer, D.F., Bertolotto, M., Schön, B., & Mosa, A.S. (full filing Dec. 2009). Enablement of three-dimensional hosting, indexing, analysing, and querying structure for spatial databases. Patent Application No. 09177926. Europe.

Li, J., Jing, N., & Sun, M. (2001). Spatial Database Techniques Oriented to Visualization in 3D GIS. In Proceedings of 2nd International Symposium on Digital Earth. 24-28.06.2001, Canada.

Morton, G.M. (1966). A Computer Oriented Geodetic DataBase and a New Technique in File Sequencing. IBM, Ottawa, Canada.

Murray, C. (2009, March). Oracle Spatial Developer's Guide 11*g* Release 1 (11.1) B28400-04.

Murray, C. (2003, December). Quadtree Indexing. Retrieved March 2010.

Oosterom, P. van (1990). Reactive Data Structures for Geographic Information Systems. PhD Thesis, Leiden University, The Netherlands.

Comment [bs2]: I don't think it is right to use this reference. It doesn't match to what it references either in the text.

494 Sagan, H. (1994). Space-Filling Curves. New York: Springer-Verlag.

495 Samet, H. (2006). Object-Based and Image-Based Image Representations. In H. Samet, & A.
496 Palmeiro (Ed.), Foundations of Multidimensional and Metric Data Structures (pp. 211-220).

497 Schön, B., Laefer, D.F., Morrish, S.W., & Bertolotto, M. (2009a). Three-Dimensional Spatial
498 Information Systems: State of the Art Review. Recent Patents on Computer Science, 2(1), 21-31.

499 Schön, B., Bertolotto, M. & Laefer, D.F. (2009b). Storage, manipulation, and visualization of
500 LiDAR data. Ed. Remondino, F., El-Hakim, S., and Gonzo, L. International Archives of Photo-
501 grammetry, Remote Sensing and Spatial Information Sciences, Volume XXXVIII-5/W1, ISSN
502 1682-1777.

503 Stanzione, T., & Johnson, K. (2007). GIS Enabled Modeling and Simulation (GEMS). 2007
504 ESRI International User Conference. 18–22.06.2007, San Diego, California, USA. Retrieved
505 March 2010

506 Velizhev, A. & Shapovalov, R. (2008). GML LidarK Library.
507 http://graphics.cs.msu.ru/ru/science/research/3dpoint/lidark (accessed July 13, 2010).

508 Wang, J., & Shan, J. (2005). Lidar Data Management with 3-D Hilbert Space-Filling Curve.
509 ASPRS 2005 Annual Conference. 7-11.03.2005, Baltimore, USA. Retrieved March 2010

510 Zhu, Q., Gong, J., & Yeting, Z. (2007). An Efficient 3D R-tree Spatial Index Method for Virtual
511 Geographic Environments. ISPRS Journal of Photogrammetry and Remote Sensing, 62(3), 217-
512 224.