Evaluating the benefits of Octree-based indexing for LiDAR data

This paper presents the implementation and evaluation of an octree-based index atop a commercial spatial database for the hosting, indexing, and querying of three-dimensional pointcloud data from aerial laser scanning.

Abstract

In recent years the geospatial domain has seen a significant increase in the availability of very large three-dimensional (3D) point datasets. These datasets originate from a variety of sources, such as for example Light Detection and Ranging (LiDAR) or meteorological weather recordings. Increasingly, a desire within the geospatial community has been expressed to exploit these types of 3D point data in a meaningful engineering context that goes beyond mere visualization. However, current Spatial Information Systems (SISs) provide only limited support for vast 3D point datasets. Even those systems that advertise their support for in-built 3D data types provide very limited functionality to manipulate such data types. In particular, an effective means of indexing large 3D point datasets is yet missing, however it is crucial for effective analysis. Next to the large size of 3D point datasets they may also be information rich, for example they may contain color information or some other associated semantic. This paper presents an alternative spatial indexing technique, which is based on an octree data structure. We show that it outperforms R-tree index, while being able to group 3D points based on their attribute values at the same time. This paper presents an evaluation employing this octree spatial indexing technique and successfully highlights its advantages for sparse as well as uniformly distributed data on the basis of an extensive LiDAR dataset.

1. Introduction

Recent years have seen an ever increasing availability of three-dimensional (3D) point cloud datasets, such as those generated from Light Detection and Ranging (LiDAR), also known as laser scanning. LiDAR is a remote sensing technology that has gained widespread popularity due to its usage in environmental and disaster management scenarios (e.g. Straatsmas & Baptist, 2008; Olsen, 2009; Laefer & Pradhan, 2006). The introduction of GPS+GLONASS and "fitting" software facilitating data collection with increased accuracy (Burman, 2002), and recent innovations in flight path design demonstrate new possibilities for large-scale 3D data collection in urban environments. This increasing availability of these vast LiDAR-based point cloud datasets (typically containing hundreds of millions of points) has challenged existing means of effective exploitation, as support for efficient management of these datasets is still in its early stages. A major difficulty lies in the efficient storing and indexing of these large datasets in conventional Spatial Information Systems (SISs).

Two main efforts for storing and analysis can be identified thus far. On one side, conventional Geographic Information Systems (GISs) store the data spread across several files. This approach has been followed since the 1960s, when GISs were used to deal with positional data or data with spatial *extent* (Sheckhar & Chawla, 2003). Nowadays, different GIS vendors utilize their own proprietary file formats for the representation of such data. The storage and management of any vast dataset in a file system has the following disadvantages: (1) data inconsistency, (2) data redundancy, (3) lack of multi-user concurrency, and (4) lack of data integrity. Analysis in such a scenario relies on frequent import and export transactions of said files into various Computer Aided Design (CAD) or other proprietary software, such Leica's Cyclone. This process is time intensive and requires the availability of and the training of staff on several software packages.

Database Management Systems (DBMSs) on the other hand, provide means for effective data handling of large data volumes, while facilitating the retrieval of information in vast datasets through Structured Query Language (SQL). An alternative technology called Spatial Database Management System

(SDBMS) relies on a DBMS. In such an arrangement many vendors provide spatial extensions to their Object Relational Database Management System (ORDBMS). PostGIS, for example, is an implementation of the OGC standard (OGC, 2010) is a non-commercial system for the storage of spatial data. However, it does not provide any in-built support for vast 3D point cloud data. Oracle Spatial, a commercial system on the other hand, has recently included support for these data types. Their usefulness and capabilities are further evaluated within this paper.

Looking forward, a scenario where many individuals and organizations are contributing data and trying to access the subsequent combined data is easy to envision. In recent calls for proposals both Ireland's National Road Authority and America's Association for State Highway and Transportation Organizations have sought research proposals for the integration of both terrestrial and aerial remote sensing data (NRA 2010) based on increasing interest in this area (IDOT 2003). Such an environment will further strain the existing strategies to store this data in a meaningful way. Furthermore, there will be a greater desire to exploit the three-dimensional (3D) functionality of the data. A key component of that is to have access to the original data points. This will greatly facilitate the integration of multiple datasets. As such, the traditional approach to store 3D point cloud data across various files or deriving other formats such as Digital Elevation Models (DEMs) for analysis purposes is likely to become less than attractive. As such, new approaches must be considered to fully enable the increasingly rich and 3D nature of the data, such as better support of the raw point cloud data in SDBMSs. This paper show the potential of octree-based indexing for 3D point clouds hosted within an SDBMS.

Applying an SDBMS for LiDAR data hosting allows for improved data integrity, multi-user access, web access, and the use of SQL for spatial queries. However, such a spatial system must support the data types for storing geometries in 3D Euclidean space (such as point, line, surface and volume) that are based on a 3D geometric data model (i.e. vector and/or raster data with underlying geometry and topology). The query language of a 3D spatial system must also support operations and functions to handle 3D data types (Bruenig & Zlatanova, 2004). To date, support for two-dimensional (2D) positional data is widely

74 available in both GIS and SDBMS technology. However, very limited capabilities are provided by
75 commercial products for 3D data (Schön, 2009a). Presently, many of the benefits of these datasets remain
76 relatively unexploited due to the inability of current systems to fully support 3D objects in a spatially
77 accurate and meaningful manner.

78 The speed of data retrieval operations from a database table is a critical issue for handling large datasets.
79 Indexing improves the speed with which operations are performed on a dataset by reducing the amount of
80 data that needs to be analyzed. In the spatial domain, indexes organize the dataset based on either objects
81 or the underlying space for efficient execution of spatial queries. Common indexing techniques for spatial
82 datasets include object-based R-tree indexing and space-based quadtree/octree indexing. Oracle Spatial
83 has provided R-tree indexing for spatial data while the previously supported quadtree has been
84 deprecated (Murray, 2003). However, particular 3D spatial queries (e.g. window queries, nearest
85 neighbor) cannot currently be performed on 3D datasets using Oracle R-tree index as will be further
86 discussed in section 2.1.

87 In this paper, the integration of all required functionality for storing, indexing, manipulating and
88 analyzing 3D point clouds within an SDBMS as a viable solution is considered with respect to an octree
89 index implementation atop Oracle Extensible Indexing Framework (OEIF) (Laefer et al., 2009). This
90 approach greatly benefits spatial queries on a variety of 3D point clouds. The particular contribution of
91 the method described within this paper is its applicability to 3D point clouds of varied distributions, as
92 well as such that contain further semantic information, as is illustrated in section 4.

93 **2. Indexing 3D Point Cloud Data**

94 Indexing provides faster and more intelligent query executions. Typically, the data are structured into a
95 hierarchical tree. Queries then need only follow certain branches and may avoid others. In principle,
96 spatial queries on 3D point clouds could be performed directly on the entire dataset without indexing. In
97 that scenario, for a particular spatial query, the corresponding spatial function analyzes the entire dataset

and then retrieves only the relevant spatial objects. However, since spatial functions are comparatively expensive, it would be rather cost-effective to analyze an entire dataset. Instead, an appropriate spatial index needs to be created. Spatial indexes help retrieving candidate geometries for the specified spatial query, and the corresponding spatial function is then applied to this filtered dataset, which is consequently reduced. In order to find the area of interest and retrieve the most reduced dataset, the suitability of the indexing method is critical.

An effective algorithm for spatial indexing depends on the type and dimension of the spatial objects involved. For efficient querying of 3D point clouds, it is important to index these data taking all three dimensions into account. The data must also be processed in a timely fashion to facilitate efficient execution of spatial queries. Some spatial indexing methods are discussed in the following section, with a particular focus on 3D point cloud data.

2.1. Different Spatial Indexing Approaches

A spatial index organizes the spatial data and the underlying space in order to perform efficient execution of spatial queries either in an object-based or a space-based fashion. Object-based spatial indexes organize the dataset based on the spatial objects distribution, while the space-based spatial indexes subdivide the dataset based on a subdivision of the underlying space.

One of the most popular and enduring object-based indexing techniques is the so called R-tree, which was developed by Guttmann (1984). A popular space-based alternative is the two-dimensional (2D) quadtree (Samet, 1995) and its 3D extension, the octree (Samet, 2006).

An R-tree is a dynamic depth-balanced tree, which indexes the Minimum Bounding Rectangles (MBRs) in 2D or Minimum Bounding Boxes (MBBs) in 3D of spatial objects. The MBRs/MBBs of spatial objects form the leaf nodes of the tree, and multiple MBRs/MBBs are grouped together into larger rectangles/boxes in order to form intermediate nodes of the tree. The process is repeated until only one rectangle/box is left that contains all the data that corresponds to the root node of the tree.

A quadtree is a space-based hierarchical tree structure which applies a recursive subdivision of a 2D space into four quadrants (also known as cells). It can be applied to the indexing of spatial objects embedded in a 2D space. In the basic quadtree structure, the subdivision of the space is in equal sized quadrants. Typically, a quadtree results in an unbalanced tree for irregularly-distributed data. This is beneficial as empty patches of spaces are not stored within the structure and are thus emitted during analysis. Several variations of the quadtree structure have been developed in the literature (Samet, 2006, p.28) for point data and linear data.

The tree-based quadtrees are based on the recursive subdivision of the region into four congruent quadrants until a quadrant is homogeneous. The homogeneity condition for point data could be defined as the maximum number of points that a quadrant contains or other user-defined criteria. Alternatively, the homogeneity condition could be based on the semantic information of the point data (e.g. color). For example, subdivision could occur until a single color percentage threshold is reached.

One adaptation of the quadtree for indexing high volume point clouds is the so called PR quadtree. It divides the underlying space up to a fixed tiling level. Each tile (also known as cell) is assigned with a unique code (also known as cell code), which is used to index the points that are covered by this tile. This is how the quadtree was implemented in Oracle Spatial. The 3D analogue of PR quadtree is the PR octree, which has been implemented by the authors, as described in section 3.This approach has the distinct advantage of being performance efficient, as the current branch level does not need to be stored within the database, which would reduce the efficiency of queries. On the other hand, this approach looses the initial advantage of a space based spatial index to grow naturally from the underlying space and omit empty areas. Indexing point data using a PR quadtree/octree may be very useful since its quadrant/octant can contain the data points along with their location information directly. In addition, it can store the semantic information of data points. The quadtree index was extended by De Floriani et al. (2008) to work with Triangular Irregular Networks (TINs). Theoretically, this approach could be generalized for Tetrahedral Irregular Networks (TENs) based on an octree structure in order to support true 3D functionality.

147 Currently, there is no published work describing this research. Boubekeur et al. (2006) emphasized that
148 hierarchical space division based structures (e.g. octree and k-d tree) are critical for surface representation
149 as they are purely volume based. Therefore, they suggested a combined approach called Volume-Surface
150 tree (VS tree), which combines an octree structure with a set of quadtrees to describe a discrete 3D
151 surface. The VS-tree is constructed by switching back to the quadtree during the recursive split involved
152 in the octree construction, as soon as a certain "height field" is reached. However, this approach was
153 found to break down to mere octree indexing on certain surfaces (Velizhev & Shapovalov, 2008). Several
154 other strategies have been developed for efficient indexing of multi-dimensional data. However, there is
155 limited vendor support for these and true 3D index creation is an ongoing research issue.

156 Efficient indexing of multi-dimensional data and true 3D index creation is still an ongoing research
157 problem as summarized by Schön et al. (2009b). Most of the commercial systems provide only support
158 for 2D index creation with simple 3D extension (Arens, 2005). Alternatively, this paper presents the
159 implementation of a 3D space-based indexing structure based on an octree, which is not currently
160 available commercially (Laefer et al., 2009). In the following section, the advantages of octree indexing
161 over R-tree indexing for 3D point cloud data are discussed.

162 **2.2. Advantages of an Octree Index for 3D Point Cloud Data**

163 Employing an octree structure for indexing 3D point cloud data has distinct advantages over using an R-
164 tree data structure. One major benefit is that the octree can be applied directly on the point geometries, as
165 opposed to merely the bounding boxes that an R-tree relies upon. As such, there is no need to decide how
166 to implement a bounding box for a 3D point. Furthermore, the octree is a hierarchical tree where nodes
167 are disjoint. This means that the regions corresponding to tree nodes are non-overlapping. On the other
168 hand, bounding boxes in R-trees are often overlapping. If bounds overlap, more branches have to be
169 traversed to process a query, which reduces an index's efficiency. In Oracle Spatial, the implementation
170 of the R-tree index stores the tree structure into a table and selects a node using internal SQL statements
171 while each node is visited (Kothuri, 2002). For that reason, query processing using an R-tree index in

Oracle Spatial involves processing several recursive SQL statements, which prolongs the query processing time (Kothuri, 2002).

A standard octree implementation typically results in an unbalanced tree. However, it can be implemented as a balanced tree (i.e. the entire space is subdivided only to a specified tiling level). In this particular case, it only requires storing the tiling level as the tree structure can be rebuilt during query processing by using this tiling level information. This approach has been adopted for the implementation of the octree structure as described later in this paper.

Another advantage of the octree is its capability to maintain the semantics of point data. Because the octree has the ability to store data points directly (instead of merely their bounding boxes), semantic information is accessible from the index and can, in fact, be used to build the index itself. As a practical example, this means that points with specific attributes can be grouped together in one cell. Regarding LiDAR data, an example of semantic criterion might be color based on RGB values, see schematic in fig.1. This criterion could allow cell subdivision until a given percentage of color similarity is reached within a cell. An R-tree index is not capable of grouping the data according to their semantics, therefore loosing valuable information associated to individual points. Figure 1 illustrates how an octree preserves the semantics, choosing color as a semantic criterion; for ease of illustration, all figures are presented in 2D, and the semantic of a region is illustrated as fully homogenously colored region, even though the same principle applies in 3D and to regions of only partial uniformity, e.g. 80% color similarity within cells (for reasons of publishing, instead of color different shapes were substituted to indicate different attributes).

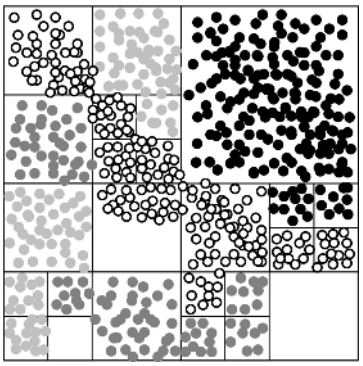

Figure 1: Octree segmentation using a Semantic as Grouping Parameter

A further advantage of using an octree for 3D point cloud indexing lies in the possibility to optimize 3D point cloud visualization (Koo & Shin, 2005). Rendering of 3D point clouds is computationally expensive, but an octree can be used to filter visible points for rendering a specific view frustum, instead of rendering all points in the dataset at once.

Naturally, the decision of which indexing structure to use depends on many factors, such as data distribution and the type of data. The following section presents the approach employed within this paper, an octree implementation atop the Oracle Extensible Indexing Framework (OEIF). This approach is further evaluated in section 4 and discussed in section 5.

3. Implementation of an Octree Structure in Oracle Spatial

This section presents an implementation of a new octree-based index structure for 3D point clouds within Oracle Spatial 11g. The extensibility capability of the Oracle SDBMS was utilized in order to implement this index. The framework for indexing is known as Oracle Extensible Indexing Framework (OEIF) and the new index is a so called domain index. The framework defines a set of interface methods, which is required to be implemented in an object type, called indextype (Belden, 2008). The name of the interface is ODCIIndex where ODCI stands for Oracle Data Cartridge Interface. The methods of the ODCIIndex interface are categorized into four classes: index definition methods, index maintenance methods, index scan methods, and index metadata methods.

An indextype is an object that specifies the routines that manage a domain (application-specific) index. An indextype has two major components: the methods that implement the index's behavior and the operators that the index supports. In this paper, a new indextype has been implemented in OEIF in order to implement the proposed octree structure for LiDAR data indexing in Oracle Spatial. The name of the new indextype is OCTREEINDEX. This index implementation is also comprised of an operator implementation called OT_CLIP_3D, which performs a window query on any given 3D point cloud, which is stored in an SDO_GEOMETRY data type.

Before introducing 3D data types, Oracle Spatial relied heavily on SDO_GEOMETRY. More recently Oracle Spatial has moved to SDO_PC as the main data type employed for the storage of multi-dimensional point cloud data. With this, a set of points are grouped and stored as the Binary Large Object (BLOB) object in a row. While there is no upper bound on the number of points in an SDO_PC object, 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 still provides the greatest flexibility in storage of any geometry type, including 3D data points. In particular, when considering 3D point clouds, it is desirable to store in the same table the locational information and the attribute information (e.g. color, intensity), as semantic information oftentimes directs the feature recognition processes – typically applied at a later stage in the workflow. For these reasons, our implementation relies on SDO_GEOMETRY instead of SDO_PC. However, the approach could easily be adapted to be used with SDO_PC.

A conventional octree is an unbalanced hierarchical tree, which would require storage of its logical tree structure in a SDBMS for reconstruction of the tree structure during query processing. As described in section 2.2., this could introduce inefficient query processing due to the issuance of several internal recursive SQL select statements generated during each node visit. This issue is resolved by constructing a balanced tree structure with a fixed tiling level. In this case, only the tiling level information (as opposed to the whole tree) needs to be stored for tree reconstruction. The pre-selection of an appropriate tiling level for a specific dataset is a crucial factor, which involves considerations regarding 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. This was discussed in section 2.2.

In the current implementation, the user is allowed to specify the tiling level of the octree through the parameter OCTREE_LEVEL during index creation. The 3D region is recursively divided into eight

congruent cells up to the specified tiling level. Each cell is associated with a unique code, which is hereafter referred to as the cell code. The cell code is obtained by using z-ordering (i.e. Morton encoding) of all cells at the specified level (Morton, 1966). Each point is indexed by the associated cell code of the octant that contains the point. The ROWID of the point and the associated cell code are stored in the index storage table. The meta data (e.g. tiling level, index name, index owner, max level, min level) for the entire index are stored as a row in the index meta data table.

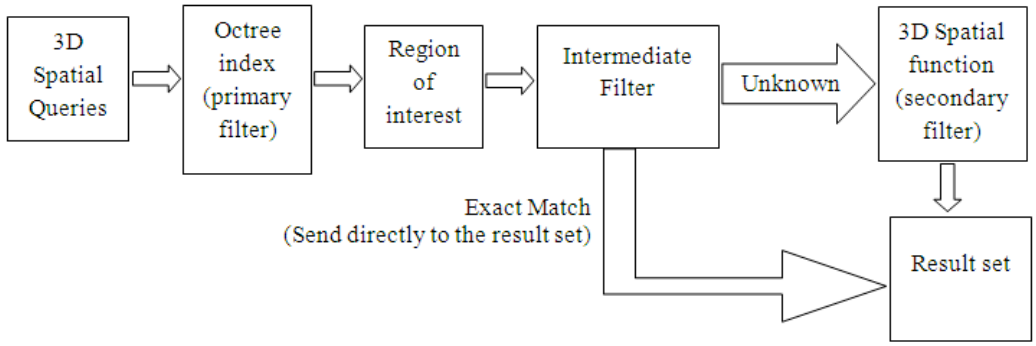

Figure 2: 3D query processing using Octree index.

The 3D query processing using this implementation is illustrated in

Figure **2**. To generate the result set for a spatial query, the octree index acts as the primary filter to find the area of interest or candidate geometries for this query.

Figure **2** illustrates the use of a primary and a secondary filter during the query process. The area of interest is the union 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 exact matches. 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.

The required ODCIIndex interface methods for implementing the proposed octree index for LiDAR data atop OEIF are implemented as Java callouts. A previously available Java API was harnessed for this purpose (Kothuri, 2007, p. 223). It enables applications written in Java to access and process geometry objects managed in Oracle database with Oracle Spatial. The details of this implementation are available in Mosa (2010).

4. Evaluation of the Octree Spatial Index

This section provides an evaluation of the implementation of the octree index presented in the previous section. A window query is used in order to compare response times of the existing Oracle Spatial R-tree and the newly implemented octree index. This evaluation has been conducted on a computer with the Intel Core2 Duo CPU 2.53GHz and 4GB RAM, 7,200 SATA hard drive on Oracle 11g release 11.1.0.6.

The 3D 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 R-tree and with an octree in a separate run. The R-tree index was created using Oracle's existing in-built spatial index. Presently, a 3D window query cannot be performed using Oracle R-tree index, as all but one spatial operator can only be applied to 2D geometries. As such, true 3D window queries cannot be performed using Oracle R-tree index. The overlap of sibling nodes and the uneven size of nodes in an R-tree may develop inefficient query execution (Zhu, 2007). Thus, only the 2D window query could be performed on the 2D R-tree index using the SDO_RELATE operator by providing "inside and touch" masks (Kothuri, 2007, p. 274). This provides functionality similar to a general window query. Here, the 2D R-tree index is created on the 2D projection of the 3D point cloud data.

With the octree index, the 3D window query can be performed on 3D LiDAR point cloud data using the operator OT_CLIP_3D. A tiling level of five was selected for the candidate dataset used in this example.

An increased tiling level would result in a decreased indexed point per cell count, as well as candidate geometries. In contrast, the increase of tiling levels results in an increase of leaf nodes (e.g. total number of leaf node at tiling level 'n' is 8^n), as well as memory consumption during octree manipulation.

The comparison of a 2D (x- and y- ordinates) window query using an R-tree index with a 3D (x, y and z) window query using octree index requires the same number of resulting geometries for a window query as query response time increases with increases in the query window, as well as the number of resulting geometries (Kothuri, 2002). To ensure the same number of resulting geometries for octree and R-tree index, the minimum and maximum value of the z-ordinates of the query window is set to the minimum and maximum value of the underlying space in case of the octree index.

To this end, two randomly selected data subsets from a dense aerial LiDAR flyover of Dublin's city centre (Hinks et al., 2009) were selected for this experiment. One contained just shy of 2.9 million points and the other almost 66 million points. Query response times were compared for the two index types for a variety of window sizes. For the smaller dataset (2.9 million), the octree was twice as fast as R-tree for the small window of $25m^2$ and 8 times as fast for the large window of $2{,}500m^2$. For the larger dataset (66 million), the R-tree outperforms the octree for the small window of $400m^2$ size, but for large windows of $1600m^2$ and above, the octree performed distinctly better, with a six-fold improvement for a $40{,}000m^2$ window.

As datasets combined from different sources are becoming more irregularly distributed with significantly higher densities in some areas. To evaluate potential performance benefits of octree-based indexing on irregularly-distributed LiDAR point clouds, portions of data were selectively removed from a high density aerial LiDAR flyover of Dublin's city centre (Hinks et al, 2009). The dataset contains nearly 66 million points over $.25km^2$, including points on the ground and road surfaces, rooftops, tree canopies and building facades. The initial point distribution was quite uniform, with a density of 225 points per square meter almost everywhere. For evaluation purposes it was of interest to test the octree index structure also

on an irregularly distributed dataset, which was consequently derived from the initial dataset. This was achieved by removing points where the z ordinate was larger than 20 meters or less than 10 meters.

The resulting dataset contains points on building facades. The dataset also contains points on the rooftops and the tree canopies, where the height is less than 20 meters. This generates an irregular distribution of points within the 3D point cloud with a variable density in different areas. Figure 3 presents the 3D rendering of this dataset approximately 17,5 million points. Table 1 presents the density distribution in different area of the dataset.

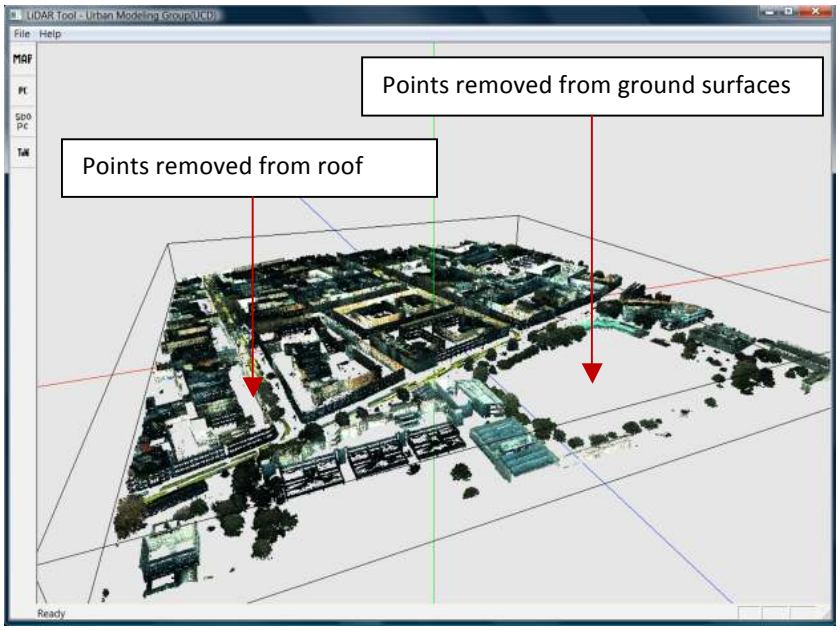

Figure 3: Irregular distribution of 3D point cloud (17,517,406 points in the dataset)

Table 1: Density Distribution of Query Windows (17,517,406 irregularly distributed points)

Window No.	X1	Y1	X2	Y2	Area (m2)	Total Point	Density (points/m2)
1	233,900	316,300	234,000	316,400	10,000	14,579	1.46
2	233,700	316,200	233,800	316,300	10,000	273,688	27.37
3	233,700	316,400	233,800	316,500	10,000	321,059	32.11
4	233,600	316,400	233,700	316,500	10,000	337,032	33.7
5	233,800	316,200	233,900	316,300	10,000	345,263	34.53
6	233,700	316,100	233,800	316,200	10,000	381,920	38.19
7	233,800	316,300	233,900	316,400	10,000	406,465	40.65
8	233,700	316,300	233,800	316,400	10,000	433,147	43.31
9	233,900	316,100	234,000	316,200	10,000	451,694	45.17
10	233,500	316,400	233,600	316,500	10,000	507,923	50.79
11	233,700	316,000	233,800	316,100	10,000	531,261	53.13
12	233,900	316,200	234,000	316,300	10,000	563,964	56.4

13	233,800	316,100	233,900	316,200	10,000	734,972	73.5
14	233,900	316,000	234,000	316,100	10,000	797,447	79.74
15	233,900	316,400	234,000	316,500	10,000	809,561	80.96
16	233,600	316,300	233,700	316,400	10,000	812,631	81.26
17	233,800	316,000	233,900	316,100	10,000	854,245	85.42
18	233,800	316,400	233,900	316,500	10,000	862,660	86.27
19	233,500	316,300	233,600	316,400	10,000	936,929	93.69
20	233,600	316,100	233,700	316,200	10,000	1,018,365	101.84
21	233,500	316,000	233,600	316,100	10,000	1,139,836	113.98
22	233,600	316,200	233,700	316,300	10,000	1,166,117	116.61
23	233,500	316,100	233,600	316,200	10,000	1,168,042	116.8
24	233,500	316,200	233,600	316,300	10,000	1,236,741	123.67
25	233,600	316,000	233,700	316,100	10,000	1,414,439	141.44

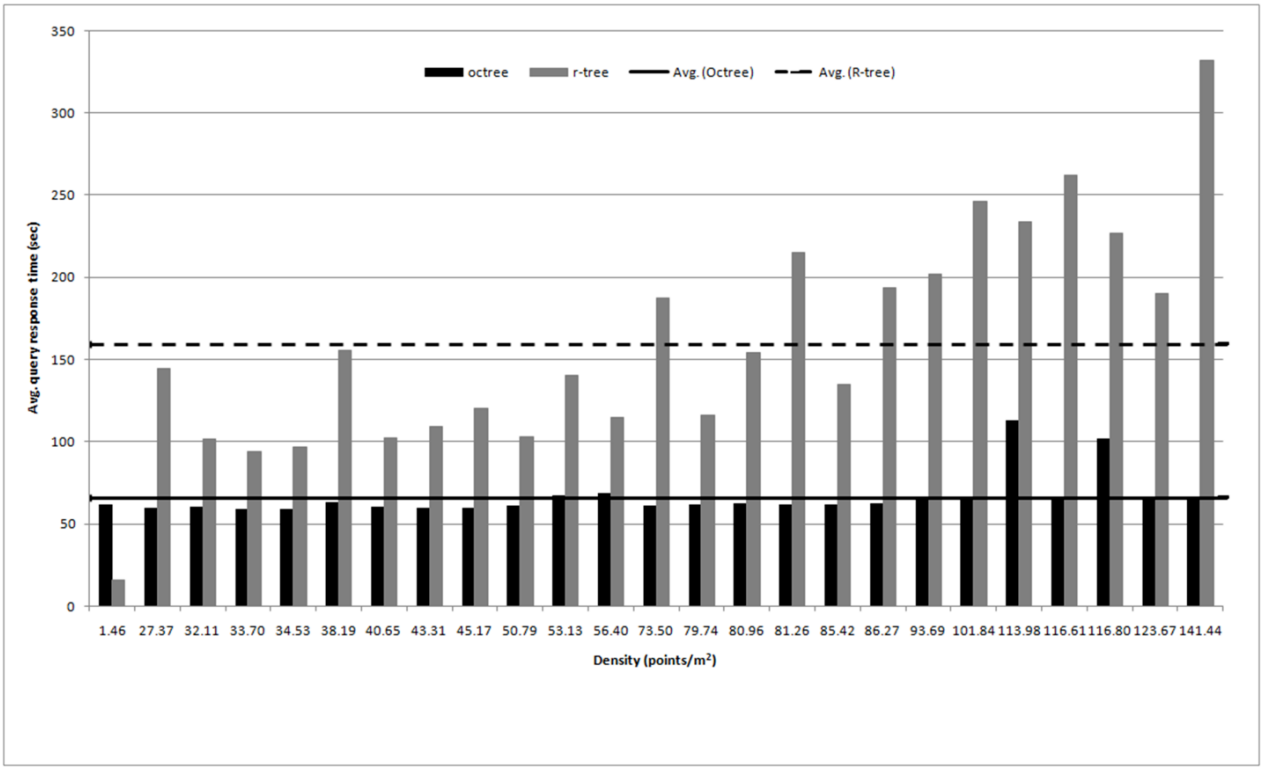

Figure 4: R-tree vs. octree 17,517,406 irregularly distributed points

Table 2 presents the average query response time along with the density distribution in different regions. A square window of size 10,000 m^2 (100m x 100m) was chosen for the window query and moved around the underlying area in a random pattern. A total of 25 query windows were chosen in a 5x5 grid. The query windows are presented in Table 2 in ascending order according to their point density. For every query window, 10 queries were performed and query response time presents the average of these queries.

Tab. 2 illustrates the comparison of the window query response time between the R-tree and the octree index. The minimum density is 1.46 $m^{2,}$ maximum density is 141.44 m^2, and median density 73.5 m^2.

The query response time increases with some variations with the increase of density. Overall, it can be noted from this evaluation series that the octree index performs in a very consistent manner. In most cases, the octree significantly outperforms the in-built R-tree index by an average factor 2,4.

For the window of 27,37 points/m^2 density, the octree is 2,4 times faster than R-tree. Additionally, for window densities below the median value the octree is on average twice as fast as the R-tree. On the other hand, for window densities over the median value the octree is on average three times faster than the R-tree. For the highly dense region (141,44 m^2), the octree is about five times faster than the R-tree. The octree in most cases outperforms the R-tree.

Table 2: R-tree vs. octree (17,517,406 irregularly distributed points)

Window No.	Density (no. of points /m2)	Avg. Query Response Time in sec. (R-tree)	Avg. Query Response Time in sec. (octree)
1	1.46	15.58	61.72
2	27.37	144.22	59.96
3	32.11	101.42	60.42
4	33.7	93.88	59.26
5	34.53	96.51	59.44
6	38.19	155.5	63.32
7	40.65	102.41	60.42
8	43.31	109.32	59.66
9	45.17	120.44	59.68
10	50.79	103.09	60.87
11	53.13	140.68	67.38
12	56.4	114.76	68.58
13	73.5	187.72	61.45
14	79.74	116.39	62.11
15	80.96	154.43	62.68
16	81.26	214.75	61.67
17	85.42	135.05	62.2
18	86.27	193.59	62.89
19	93.69	201.62	64.33
20	101.84	246.1	64.98
21	113.98	234.01	112.83
22	116.61	261.84	65.9
23	116.8	226.73	102.16
24	123.67	190.47	65.49

25	141.44	331.8	66.44

5. Discussion

The research presented in this paper has implemented an octree index for 3D point cloud data, employing Oracle's extensible indexing framework. An operator has been implemented in order to perform a 3D window query. The implementation is described along with some optimizations. The newly implemented octree index and Oracle's inbuilt R-tree index are compared using a high density aerial LiDAR point cloud dataset. The evaluation highlights distinct advantages of using an octree based index for both regularly and irregularly-distributed point cloud data. For regularly-distributed data, the octree index consistently outperforms an R-tree index by two to eight times for almost every window size. For irregularly-distributed data, the octree index consistently outperforms the R-tree index by two to five times for most of the density areas except the lower density area (1,46 points/m^2). However, the current implementation can be optimized further to improve the performance for lower density area.

In this work Oracle Spatial was chosen due to its current ability to store 3D data. However, the implementation could be adapted for other SDBMS. Currently, in Oracle Spatial, the R-tree or quadtree index can be applied on the block extent column of the SDO_PC data type. However, it is possible to implement an octree index on the block extent of the SDO_PC. Currently, using Oracle's SDO_PC, only the block extents are indexed rather the actual point geometries. However, it is possible to access these points directly from the block column and index them. Furthermore, it is also possible to implement a two-step index, where an octree indexes the points inside a block and therefore maintains their semantic information inside that block, and an R-tree index serves as a higher level index that is applied to the block extents, as R-tree indexing is more suitable for polygon objects than an octree index would be. Further work will fully evaluate this approach.

Finally, the method presented in this paper employs only one operator, which implements a window query. A more comprehensive evaluation is needed in order to assess the octree index's full potential for other query operators, such as nearest neighbor or within distance. Since octree indexes are useful for

365 solving the visibility of points during 3D point cloud rendering, an operator can be implemented based on
366 the octree index implemented for this purpose. The operator should have the view frustum or window as
367 input parameter and return only the visible points for this window. It facilitates the extraction of visual
368 points for the specified window through SQL. In this prototype, the tiling level is determined by the user
369 of a dataset. Further work will enhance the prototype by incorporating a feature for automatic tiling level
370 determination.

371 **References**

372 Arens, C., Stoter, J., & van Oosterom, P., 2005. Modelling 3D Spatial Objects in a Geo-DBMS using a
373 3D primitive. *Computers & Geosciences*, 31(2): 165-177.

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

376 Boubekeur, T., Heidrich, W., Xavier, G., and Christophe, S., 2006. Volume-Surface Trees. *Eurographics,*
377 25(3): 399-406.

378 Bruenig, M., and Zlatanova, S. (2004, November 6). 3D Geo-DBMS. *Directions Magazine*. Retrieved
379 March 2010, from http://www.directionsmag.com/article.php?article_id=694

380 Burman, H., 2002. Laser Strip Adjustment for Data Calibration and Verification. Photogrammetric
381 Computer Vision, ISPRS Commission III, Symposium 2002, (pp. A-67-72). Graz, Austria.

382 De Floriani, L., Facinoli, M., Magillo, P., & Debora, D., 2008. A Hierarchical Spatial Index for
383 Triangulated Surfaces. International Conference on Computer Graphics Theory and Applications, (pp. 86-
384 91).

385 Geo-Consortium, 2007. Introduction to Spatial Data Management with PostGIS. Presentation Slides by
386 the Consulting Centre Geographic Information Systems.

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

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

391 Iowa Department of Transportation (IDOT) (2003). Remote Sensing (LIDAR) for Management of
392 Highway Assets for Safety: Application of Advance Remote Sensing Technology to Asset Management.
393 Final Report – October 2003.

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

396 Kothuri, R.K., Ravada, S., and Abugov, D., 2002. Quadtree and R-tree Indexes in Oracle Spatial: a
397 Comparison Using GIS Data. ACM SIGMOD International Conference on Management of Data (pp.
398 546-557). Wisconsin, USA: ACM.

399 Kothuri, R., Godfrind, A., and Beinat, E., 2007. Pro Oracle Spatial for Oracle Database 11g.

400 Laefer, D.F., Bertolotto, M., Schoen, B., and Mosa, A.S.M., 2009. Enablement of three-dimensional
401 hosting, indexing, analysing, and querying structure for spatial databases. Patent Application No.
402 09177926. Europe. (full filing Dec. 2009).

403 Laefer, D. F., Pradhan, A., 2006. Evacuation Route Selection Based on Tree-Based Hazards Using
404 LiDAR and GIS. *J. Transportation Eng.*, 132 (4): 312-20.

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

407 Mosa, A.S.M., 2010. Hosting, Indexing and Visualization of Three-Dimensional (3D) Point Clouds. MSc
408 Thesis, University College Dublin, National University of Ireland.

409 Murray, C., 2003. Quadtree Indexing. Retrieved March 2010, from Oracle:
410 http://www.oracle.com/technology/products/spatial/pdf/qt.pdf (accessed 02 Nov 2010)

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

412 National Roads Authority (NRA), 2010.
413 http://www.nra.ie/Publications/DownloadableDocumentation/GeneralPublications/file,17098,en.pdf

414 OGC 2010. OpenGIS Implementation Standard for Geographic information - Simple feature access - Part
415 2: SQL option, ed. John R. Herring.

416 Olsen, M. J., Johnstone, E., Driscoll, N., Ashford, S.A., Kuester, F., 2009. Terrestrial Laser Scanning of
417 Extended Cliff Sections in Dynamic Environments: Parameter Analysis. *J. Surveying Eng.*, 135 (4): 161-
418 169.

419 Samet, H., 1984. The Quadtree and Related Hierarchical Data Structures. *ACM Computing Surveys*
420 (CSUR), 16 (2): 187-260.

421 Samet, H., 1995. Spatial Data Structures. Modern Database Systems: The Object Model, Interoperability
422 and Beyond, 361–385.

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

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

427 Schön, B., Bertolotto, M. and Laefer, D.F., 2009b. "Storage, manipulation, and visualization of LiDAR
428 data" 3rd Int'l Wkshp, 3D-ARCH'2009: 3D Virtual Reconstruction and Visualization of Complex
429 Architectures, Feb. 25-28, Trento, Italy, ed.s Remondino, F., El-Hakim, S., and Gonzo, L. International
430 Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume XXXVIII-

5/W1, ISSN 1682-1777. Retrieved August 2010, from http://www.isprs.org/proceedings/XXXVIII/5-W1/pdf/schoen_etal.pdf

Sheckhar, S., and Chawla, S., 2003. Spatial databases - a tour. Prentice Hall.

Straatsma M.W., Baptist M.J., 2008. Floodplain roughness parameterization using airborne laser scanning and spectral remote sensing. *Remote Sensing of Environment*, 112(3): 1062-1080.

Velizhev, A. and Shapovalov, R., 2008. GML LidarK Library. http://graphics.cs.msu.ru/ru/science/research/3dpoint/lidark (accessed Nov 02, 2010).

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