LiDAR Data Management Pipeline; from Spatial Database Population to Web-Application Visualization

Paul Lewis
National Centre for Geocomputation,
National University of Ireland,
Maynooth, Co.Kildare, Ireland
+353 (0)1 7086204
paul.lewis@nuim.ie

Conor P. Mc Elhinney
National Centre for Geocomputation,
National University of Ireland,
Maynooth, Co.Kildare, Ireland
+353 (0)1 7086204
conormce@cs.nuim.ie

Timothy McCarthy
National Centre for Geocomputation,
National University of Ireland,
Maynooth, Co.Kildare, Ireland
+353 (0)1 7086180
tim.mccarthy@nuim.ie

ABSTRACT
While the existence of very large and scalable Database Management Systems (DBMSs) is well recognized, it is the usage and extension of these technologies to managing spatial data that has seen increasing amounts of research work in recent years. A focused area of this research work involves the handling of very high resolution Light Detection and Ranging (LiDAR) data. While LiDAR has many real world applications, it is usually the purview of organizations interested in capturing and monitoring our environment where it has become pervasive. In many of these cases, it has now become the de facto minimum standard expected when a need to acquire very detailed 3D spatial data is required. However, significant challenges exist when working with these data sources, from data storage to feature extraction through to data segmentation all presenting challenges relating to the very large volumes of data that exist. In this paper, we present the complete LiDAR data pipeline as managed in our spatial database framework. This involves three distinct sections, populating the database, building a spatial hierarchy that describes the available data sources, and spatially segmenting data based on user requirements which generates a visualization of these data in a WebGL enabled web-application viewer. All work presented is in an experimental results context where we show how this approach is runtime efficient given the very large volumes of LiDAR data that are being managed.

Categories and Subject Descriptors
H.2.8 [**Database Management**]: Database Applications – *spatial database and GIS.*

General Terms
Performance, Design.

Keywords
LiDAR, Spatial Database, WedGL, PostGIS.

1. INTRODUCTION
Many academic, governmental, and commercial organizations are charged with environmental mapping and monitoring tasks, many of which require accurate 3D geographical information. These requirements can be achieved using LiDAR that has typically been captured from any one or all of the following sources:

1. Static Terrestrial LiDAR
2. Mobile Terrestrial LiDAR
3. Aerial LiDAR

These geospatial data sources are particularly suited to the case of urban modeling and infrastructural mapping where high resolution 3D geographical information can be collected very quickly. From this detailed information bespoke and comparative analysis can be performed in order to monitor, understand, and plan the status and/or requirements for a survey area. It is in the context of managing such urban and road-network survey data that we present our LiDAR data management pipeline.

Generally, the typical workflow when handling LiDAR involves processing the raw data into a specific format that can be easily viewed. This is due to the sheer volume of LiDAR data which precludes easy access and viewing of the data or specific spatial regions with the data. Typically, while the raw data is provided at the conclusion of the survey only processed simplified formats, such as a Digital Elevation Model (DEM), of the data are ever viewed. These reasons have led to interest and research into use of Database Management Systems (DBMSs) for the storage and retrieval of LiDAR data.

A select amount of research work has been completed in the context of managing vast amounts of 3D spatial data in a DBMS, [1,7,22,24]. Generally they point to a strong desire to store these data in a DBMS as they offer the full optimizations inherent to modern databases such as transaction guarantees and multiuser, random access of very large datasets. They also offer advanced features such as back-up and restore capabilities. To access LiDAR data through a DBMS first the raw LiDAR data is gridded into spatially equal tiles. The spatial boundaries of these files are then stored in a DBMS which provides access to these raw files through spatial operations. We are interested in Spatial DBMS (SDBMS) where the raw LiDAR data are stored as individual spatial points allowing for unrestricted access to all the LiDAR and its attributes. This would allow organizations who purchase or collect LiDAR data to easily access and view their data.

Storing LiDAR data in this manner as a spatial element in a SDBMS will allow for the optimization of LiDAR processing

workflows. The industrial standard is to cluster the LIDAR data into spatial girds and then store each of these grids as a binary file in a unique folder for that survey, as shown in Figure 1. It is this methodology that prevails in most software suites and has proven to be a significant constraint in a number of LiDAR analysis requirements. These constraints include an inability to easily define a spatial context that improves our ability to understand/visualize what LiDAR is available from one or multiple surveys. Extending from this is having the ability to easily segment a chosen sub-section of LiDAR for bespoke uses. Using an example of a road-edge detection processing requirement, we find that these systems do not provide a context for spatial optimization across an individual or numerous data sets/surveys where logical segmentation can be easily implemented based on a targeted approach to data processing rather that the more common brute force method.

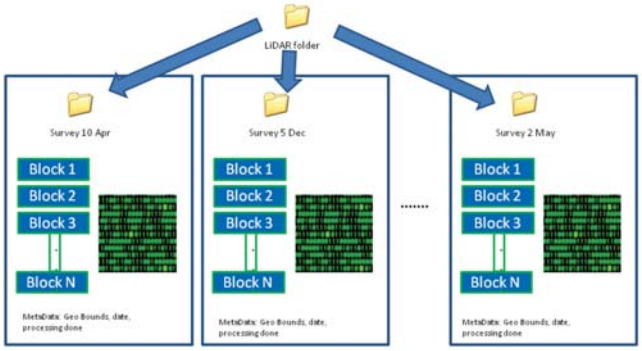


Figure 1. This example of a Survey-based approach for LiDAR data management in commercial software is typical.

In presenting our LiDAR data-framework pipeline we demonstrate how, in the first instance, a spatial approach to the storage of the raw LiDAR data has consequent benefits to managing these data from a bespoke segmentation requirements perspective. We show through our data handling pipeline how a very large scale spatial database is optimally constructed. How we can hierarchically represent these data sources in a geographically meaningful way and how these hierarchies can be used such that a bespoke LiDAR segmentation requirement can be easily performed. Finally, this process pipeline is interactively realized in a web-application that quickly allows a user to interact with the results of any segmented sections of LiDAR for requirements such as validation of processing results through to the perusal of data sources for optimization and quality.

2. BACKGROUND AND RELATED WORK

2.1 LiDAR Frameworks

As Figure 1 shows, and Section 1 mentioned, the standard methodology for commercially available LiDAR software solutions is to use spatial grids that define the LiDAR data contents of a binary file format, for each independent survey. Having examined a number of these commercial solutions highlighted at [8,20] we see that the survey based approach is typically unavoidable as these are independent standalone installations of software that use locally stored LiDAR. However, with the expansive growth of Cloud computing services we also see emerging deployments of distributed web-applications that handle LiDAR. Currently there are two significant browser based online implementations, [3,21], that are comparable in scope with the framework presented in this paper. However, it is currently not possible to examine these services in the context of the results presented here as no publications, with a sufficient level of detail, exist about them.

In [13], OpenTopography's Service Oriented Architecture is described which details their underlying use of the LAS format, [5,26], to store the raw LiDAR point data. This could be compared, at a methodological level, to our system as we store the LiDAR directly in the spatial database. However, their paper does not give enough details on the runtime performances for LiDAR data importation or retrieval, as is presented here. We could make a broad comparison based on the size and amount of raw LiDAR stored in our respective frameworks; however, this is meaningless without a formally structured study and approach, as both frameworks are handling raw LiDAR from numerous different sources to different levels of detail. Also, OpenTopography maintain Digital Elevation Models (DEMs) of their core data which we do not.

In [22], a grid approach is used to build the database index. This structured approach optimized the grid size for the indexed storage of LiDAR point clouds based on a comprehensive set of query performance tests. These results are not detailed other than one basic metric being mentioned which references the average number of rows (points) their system handled per query when the index was being built. However, this paper did define aspects of the hardware requirements that are important to such a frameworks implementation. Our paper does not focus on the hardware aspect of the framework, from which our results were dependent, but future publications could look at such a comparison. It is worth noting in [22] that future work is suggested that could investigate a spatial database solution; while our paper does focus on this issue there is no published paper, to our knowledge, that would allow us make such a detailed comparison.

A number of other publications exist that discuss frameworks which incorporate LiDAR point clouds in a databases solution. However, a comprehensive analysis and solution to the issue of importing voluminous LiDAR data into DBMSs has been to date absent. In [25], a LiDAR management system is detailed that defines their system as it interacts with users through an authentication interface, how it can manage these projects, and how data handling and processing fits into this architecture. In [15], a comprehensive Urban data management system is detailed that leverages many aspects of Geographical Information Systems (GISs), including access to LiDAR stored in a database component. In [10], a grid-computing solution is detailed for processing large volumes of LiDAR which is stored in a database. In [19], this work highlights approaches from the acquisition and processing perspective where optimized triangular meshes are generated from LiDAR point clouds stored in a database. In [18], an Oracle 11g database schema and solution is presented in a generalized way for the storage of detailed LiDAR point clouds of buildings, although no data import or query processing performance metrics are given. In [2], a Microsoft SQL Server 2008 database solution is presented, they also implement a grid index methodology for the spatial data handling aspects of their framework, however, while this paper presents some very basic metrics on query access runtimes it lacks any significant discussion for a comprehensive comparison. In [7], the authors implement a database KD-tree approach for their LiDAR with a table of results from a point-cloud rendering performance test from their visualization component. It is also worth noting in this paper that the authors briefly mention Level-Of-Detail data structures as being inappropriate to optimized visualizations of LiDAR point clouds, as is discussed in Section 2.2. As is detailed

in Section 6.2 we leverage the GPU based processing power of WebGL to handle the LiDAR visualization and interaction. As this implementation is a browser based web-application our point cloud rendering solutions performance would be dependent on the specifications of the system hosting the session. Finally, in [4], a working paper is presented on the state-of-art for LiDAR data management in a GIS context. It presents a detailed overview of various aspects of a GIS based LiDAR data-management pipeline such that the authors intend implementing and publishing on this topic in the future.

2.2 LiDAR Data Structures

Numerous methodologies are available when optimized segmentation and storage of LiDAR point clouds is required. While these approaches have validity and applications in various LiDAR data management contexts, our empirical testing and implementations have shown that gridded, tiled and/or Level-of-Detail (LOD) structures are not optimal in many instances where a database implementation, such as that presented here, is used. One example of implementing a non-database approach is in [6] where a distributed implementation is described that supports both grid and quadtree data structures.

When dealing with raw LiDAR files, storing them in LODs, grids, or tiled structures enables faster access to the data for a user. While a tiled data structure approach, such as a voxel implementation, can work for what is typically well distributed aerial LiDAR, it is not optimal for terrestrial LiDAR, especially mobile terrestrial LiDAR, where data distributions are non-uniform across space. Extending on from this is the possibility of implementing data-source specific solutions where terrestrial data is optimized in a different data structure than aerial data. However, implementing different data structure approaches makes it more difficult and less efficient to generalize our multi-source LiDAR data such that storage and access methodologies are consistent. Our framework is designed as an alternative to storing multiple copies of the same data in different spatial dimensions, i.e. tiles or grids. By storing the LiDAR point cloud data in the database we can output tiles or grids of any spatial dimension and shape.

Also, a pre-determined LOD implementation approach can suffer from the same problems that were described above, especially when working with the non-uniformly distributed LiDAR point clouds captured from a mobile terrestrial platform. Once again, our database approach can be spatially adapted, in terms of its geographical point density, to determine dynamic LOD's. Our indexing methodology facilitates simple interfaces to request point clouds based on many possible point cloud attributes, an LOD filter just extends this functionality. For example, one point per square meter or every 10^{th} point in the query area can be returned in a dynamic fashion without any significant changes in runtime efficiencies. This allows a user to quickly search an area at a dynamically definable LOD, and if the returns are not of interest, further sub-area selects at a new or different LOD requirement, if necessary, are easily accomplished.

3. LIDAR SPATIAL DATA-FRAMEWORK

Empirical experience working with geo-referenced LiDAR data suggests that the primary obstacles in the processing of these data are their considerable size and the inability to easily constrain the data based on spatial attributes. Leading on from this is the extraction and preparation difficulties when using these data for bespoke requirements. For example, an algorithm has recently been developed for the detection of road edges from terrestrial LiDAR, [16], yet this operation is being constrained by the survey-processing methodology that prevails in industry standard software suites. The efficiency of automated processing algorithms relies on the raw LiDAR data input being spatially optimised for the algorithm. Most software suites do not easily allow access to the raw data at different or irregular spatial dimensions. It is also extremely difficult to search for all the data in a region irrespective of the survey it belongs to. Thus, towards a solution to this type of problem we have implemented a Spatial DBMS (SDBMS) approach that is broadly defined in Figure 2.

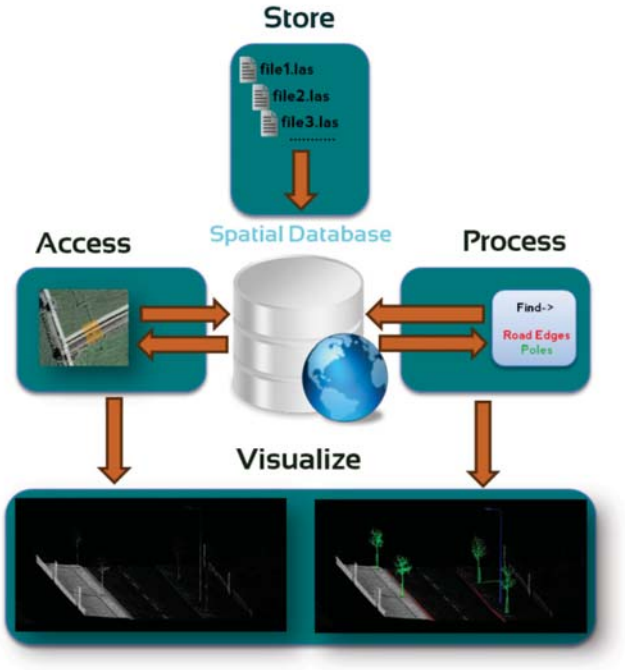


Figure 2. LiDAR Data Management Framework Overview.

The first stage in our solution is the development of a platform and methodology where large volumes of LiDAR data can be stored in an accessible form. However, LiDAR data is extremely voluminous where quantities well in excess of 36GB's per hour are commonly produced from Mobile Mapping Systems. These data are typically made up of, but not restricted to, 3D geo-referenced point-cloud information along with other properties such as Amplitude, Reflectance and RGB. This stage in our processing pipeline is detailed in Section 4.

The second stage involves building a 2D spatial hierarchy of the 3D LiDAR data uploaded into the system. The importance of this step is that it gives a spatial context where optimally located LiDAR data can be output and viewed based on either a user or system spatial request, such as data verification or a road-edge extraction algorithm. This enables a user to more easily validate results as both the raw data and processed results can be cross referenced in both a subjective visual inspection through the WebGL viewer or through an objective analytical validation process. Building these spatial hierarchies is defined in Section 5.

Finally, given that a process, either user or system defined, has determined a target LiDAR data set, based on some spatial constraints, retrieving these data in an optimal fashion is detailed in Section 6. This involves the efficient retrieval of LiDAR sub-sections from the 29 billion point 4.2 terabyte sized spatial database that currently hosts this information.

4. POPULATING A LIDAR SDBMS

4.1 SDBMS Architecture

The LiDAR data is stored in a PostgreSQL relational database management system. Integrated into this database solution are the PostGIS spatial extensions and the pg_bulkload data loading utility. It is through the use of the pg_bulkload utility that LiDAR data input and spatial index generation have been optimized. In Figure 3, we provide a detailed sample from our DB schema which incorporates a number of levels of spatial detail, as described in Section 4, from which data extraction is optimized.

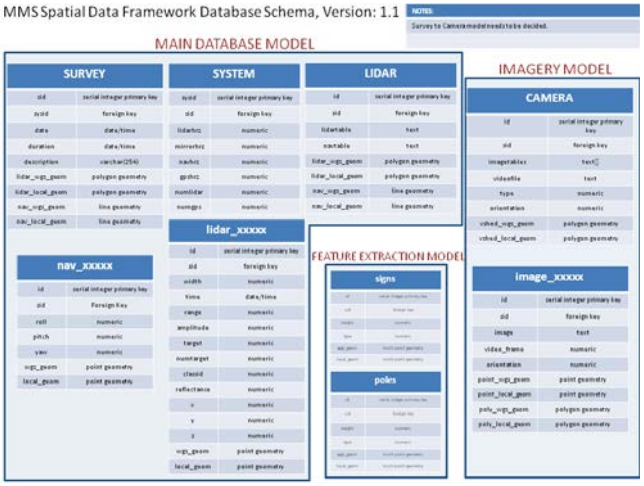


Figure 3. Sample from our LiDAR spatial database schema.

Using this platform we have shown in [14] how this technology decision has enabled us to optimize the large volumes of LiDAR data for processing by feature extraction algorithms. However, the core component to being able to implement this approach is the ability to use the extensive functionalities provided by PostgreSQL's PostGIS spatial extensions. Through PostGIS we have been able to generate and store Open Geospatial Consortium (OGC) compliant spatial objects in a standard object-relational database. Extending from this functionality is the spatial indexing options provided by PostGIS. However, while it is trivial to generate a PostGIS spatial object (geometry) and its associated spatial-index, doing so for huge volumes of LiDAR data is not trivial. It is at this point that we need to input these large volumes of spatial data in an efficient manner. To this end, we highlight our tested upload procedure and the runtime results in the rest of this section.

4.2 LiDAR Upload Procedure

Through a detailed set of experiments, [17], we have implemented and tested customized variations of the built-in PostgreSQL bulk data input functionality of the COPY command and the pg_bulkload bulk data loading utility. The PostgreSQL COPY command moves data from a file into a DB table through a SQL statement. Its performance is better on initial data loading if the destination table is empty. Performance is also improved when the data are loaded without an index constraint; thus dropping and re-creating the indexes after loading is optimal. Pg_bulkload implements an alternative methodology and has been developed for PostgreSQL by the Nippon Telegraph and Telephone Corporation. It is an optimized high volume data loading utility that skips some of the processing overheads used by COPY. It is designed to load huge amounts of data to a database where you can choose whether database constraints are checked, whether errors are ignored during the loading and to have the index updated as a synchronous operation. Pg_bulkload has two operational modes: Direct and Parallel. Direct uses one core of the system to upload the data while Parallel attempts to distribute as much processing as possible across the system cores.

Our testing of upload procedures determined an optimal solution based on an extensive set of comparative results between the use of COPY and pg_bulkload in a number of scenarios. For this testing four large LiDAR data files were prepared, as shown in Table 1. We first selected a large dataset over 6GB's in size and a small data-set over 800MB's in size. We created two files from each containing 10 columns and 14 columns respectively. By keeping the data constant and only changing the row size we intended to examine the effect row size had on the uploading approaches. We used these sample data-sets in a series of three experimental approaches where our attempt to develop a method to predict the length of time uploading a file would take was based on the file attributes.

Table 1. LiDAR Data test file properties

File	Rows	Cols	Size (MBs)	Avg. Row Size
La	66 million	10	4359.52	0.0675
Lb	66 million	14	6757.94	0.1046
Sa	8 million	10	526.90	0.0663
Sb	8 million	14	821.85	0.1034

The procedure that showed the best LiDAR data upload was achieved from a comparison between the most efficient approaches we found for both the COPY and pg_bulkload options [17]. For our data we found that by pre-processing the files, using python, to predefine the geometry data format, significantly decreased the upload time. This process involved using the Longitude, Latitude, and Altitude fields in the LiDAR data file to concatenate a canonically suitable representation of the PostGIS base-geometry data type. This base data type is an Extended Well Known Text (EWKT) representation, in three dimensional space, of the OGC Simple Features for SQL specifications. Our optimal process for loading large volumes of LiDAR into a PostGIS database table is:

1. Python – Process the original file to add the PostGIS geometry data type representation into it.
2. Create the Table – An empty table is created from the input file header fields.
3. Create geometry column – Add a PostGIS POINT-geometry data type field to the table.
4. Create Spatial Index – Create the spatial index on the PostGIS geometry field.
5. Load Data – Populate the whole table with all the raw LiDAR file data.

4.3 Results

In Table 2 we present the fastest runtime results in our comparative tests between the standard PostgreSQL COPY command and the pg_bulkload utility. Significantly, it can be seen that the speed-up gain using pg_bulkload in parallel mode is, on average, 13% that of the standard COPY procedure.

Table 2. Comparative upload times (minutes) for the optimized COPY and pg_bulkload procedures.

File	Python	SQL COPY	PG Direct	PG Parallel
La	5.14	55.40	52.42	48.08
Lb	6.26	59.47	56.03	51.75
Sa	0.63	5.87	5.28	5.09
Sb	0.79	6.42	5.82	5.59

As expected, the time to upload a row of data for the two files with 10 columns, Figures 4 and 5 (a), was shorter than when uploading the data for the 14 column case, Figures 4 and 5 (b). However, uploading the smaller file had a consistently shorter per row upload time. Also, the percentage increase in row processing time for these extra 4 columns was always significantly smaller than the percentage increase in row size (kilobytes). Adding these columns to the files resulted in a file size change ranging between 33-56%, while the resulting row uploading time change increased by only 8-15%. In Figures 6 and 7, the timing information per kilobyte is plotted. For all experiments an increase in row size resulted in a reduction in the row processing time per kilobyte. These results show that, with PostgreSQL, as the number of rows of data to upload increases the time per row increases and that an increase in row size will lead to a decrease in time per kilobyte. This implies that there is a non-linear relationship between upload time, the number of rows and row size.

Having completed these sets of experiments, the next section in our framework pipeline is to generate meaningful 2D spatial representations of these very large data tables such that an informed process or user decision can be made when applying a spatial constraint to these data.

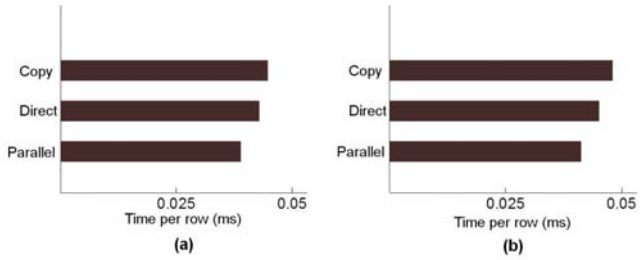


Figure 4. Timing plots for data load time per row for files (a) La and (b) Lb.

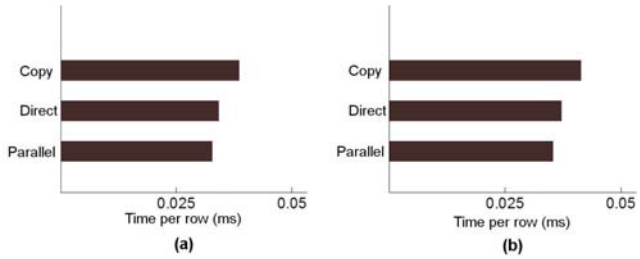


Figure 5. Timing plots for data load time per row for files (a) Sa and (b) Sb.

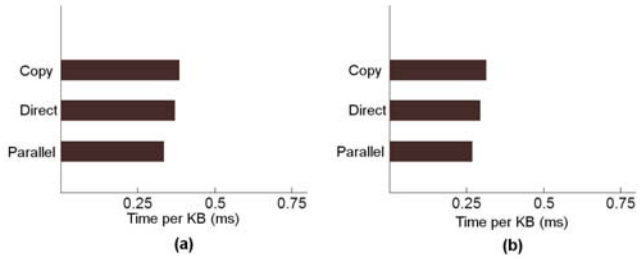


Figure 6. Timing plots for data load time per kilobyte for files (a) La and (b) Lb.

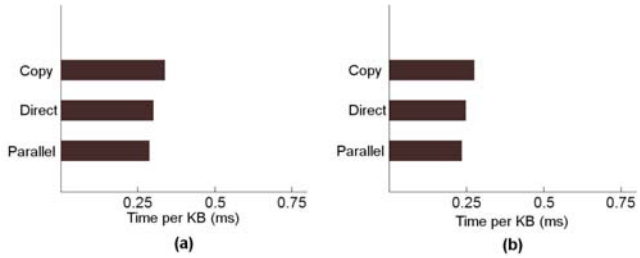
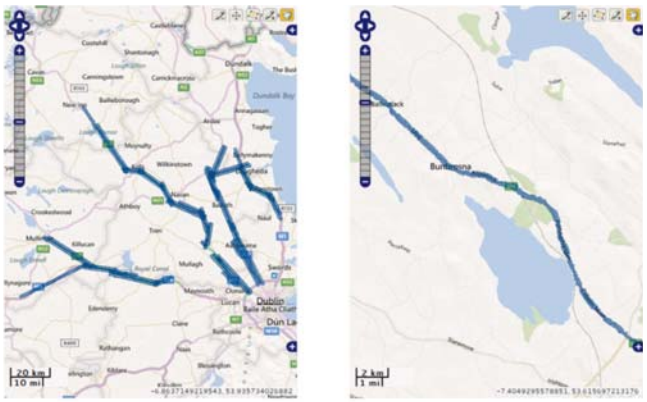


Figure 7. Timing plots for data load time per kilobyte for files (a) Sa and (b) Sb.

5. LIDAR SPATIAL HIERARCHY

5.1 LiDAR Concave Hull

To efficiently understand what data the SDBMS holds a 2D spatial hierarchy was implemented where all the spatial data sources; mobile terrestrial, aerial, and terrestrial-static are stored at different levels of spatial detail. This does not affect the raw spatial detail of the raw LiDAR but allows a user to process or evaluate the available LiDAR in approximate coverage zones. Two examples are highlighted in Figure 8, in both the user moved to a localized area of a Bing Maps UI where the LiDAR coverage is shown as a blue transparent polygon. In Figure 8 (a), the full coverage of an aerial survey containing 1,700 million points is shown and in Figure 8 (b) the LiDAR coverage for a sub-section of a 400 million point mobile terrestrial survey is shown. Within this UI the user can use standard GIS point and polygon creation tools to intersect a planar view of the available data in this area, which is presented in Section 6.



(a) (b)

Figure 8. Approximated LiDAR coverage zone for (a) aerial LiDAR and (b) mobile terrestrial LiDAR.

The procedure to perform this operation has been implemented as a bespoke version of the PostGIS 2.0 ST_ConcaveHull. The PostGIS version, unfortunately, is still a beta implementation and as such is currently only available as a Procedural Language/PostgreSQL Structured Query Language (PL/pgSQL) option. This presents a significant performance issue especially as the quantity of LiDAR being processed is so large. Thus, we have implemented our own version where a data source specific requirement is required. This is because the three different LiDAR data source types have very different point density properties, at a minimum. For example, mobile terrestrial LiDAR can have a point density in excess of 4,000 points per m^2. Point densities reduce as a function of distance orthogonal to the survey vehicles direction of operation. Effectively this LiDAR data source produces a very dense and detailed route corridor point cloud. On the other hand aerial platform LiDAR, which can also be presented in a corridor like fashion representing the aerial vehicles flight path, the point densities are very much uniform across the scan area and typically are less than 10 points per m^2.

The first stage in our process is to snap all the LiDAR data to a spatial grid. A sub-step in the first stage involves the application of a sub-sampling threshold to the LiDAR. This sub-sampling is applied differently dependent on the LiDAR capture source; in the case of mobile terrestrial LiDAR areas with high point densities, close to the survey vehicle, are sub-sampled with a higher threshold than areas with lower point densities. The second stage is to generate concave hulls for all the sub-sampled and gridded data. Finally, a spatial union is performed on all the concave hulls for the LiDAR data being processed.

This final stage can take the form of a number of different spatial unions where the highest accuracy concave hull is a direct 2D spatial representation of a single table of raw LiDAR. This idea follows through to unions that define all the data in different spatial configurations; local, regional, national, etc. The spatial union can also be applied in a number of other ways such as modeling the data using the survey based approach that exists in typical commercial software suites.

5.2 Results

In this section two sets of results are presented. They examine how the procedure described in the previous section performed in a runtime scenario from a dual perspective, one looks at the procedure with our threshold controlled sub-sampling system implemented while at the same time the grid size is changed to see its effect. The results presented in Table 3 show a clear improvement when the threshold procedure is applied. This is because the LiDAR that is being applied to the grid operation at each pass of the algorithm is much reduced based on its collection source and point density due to the sub-sampling.

Table 3. Comparative runtime results for concave hull generation over a 1 million point LiDAR database.

Grid Size (meters)	Without Threshold (seconds)	Threshold Applied (seconds)
50*50	1763.40	1.058
40*40	1541.40	1.058
30*30	912.60	1.093
20*20	525.00	1.112
10*10	277.20	1.305
5*5	154.80	1.559

We applied this procedure to a survey containing 1.9 billion LiDAR points in our framework, which required 89.5 minutes to compute. Finally, we have tested our upload and concave hull computation process on two surveys with the results displayed in Table 4. The first survey was a small mobile terrestrial survey with 91 million points with each point storing its XYZ position and a color attribute in RGB requiring 42 minutes to upload and compute the coverage. The second survey was a large aerial survey with over 1,123 million points alongside an intensity value for each point, a little over 12 hours were required to input the data and compute the LiDAR coverage. In the next section, we will demonstrate the advantages to storing these data in a SDBMS.

Table 4. Timing results for uploading and computing concave hulls for a mobile and aerial LiDAR survey.

Survey Type	No Points	Attributes	Time (H:M:S)
Mobile	91,329,780	XYZ RGB	00:41:56
Aerial	1,123,594,793	XYZI	12:14:26

6. LIDAR SPATIAL SEGMENTATION

6.1 2D LiDAR Segmentation Interface

Having approached each stage of this framework pipeline with a spatial constraint perspective to the fore, it is possible at this stage to optimize the LiDAR data being output. This can be achieved, once again, through procedures that leverage the power of the PostGIS platform through its numerous, integrated, spatial functionalities. Due to the spatial-indexing and spatial-hierarchies that have been generated for the LiDAR, it is relatively easy to highlight use cases which generate bespoke LiDAR subsets from our 29 billion points stored in a 4.2 TB database.

These use-cases could be in any relevant form where the process being initiated needs access to LiDAR. Consequently, this could

be an automated processing algorithm, such as the road-edge extraction algorithm mentioned earlier. Using this example [16], we see that subsets of LiDAR can be spatially optimized such that this algorithm's processes can be easily constrained based on the known LiDAR in the system, but can also be informed from the availability/knowledge of other spatial data that can be used to build the spatial constraint. In Figure 9, we can see a sample of the User Interface (UI) which highlights how this can happen. Alternatively, this example use-case could just as easily be a LiDAR awareness approach where LiDAR can be quickly and easily segmented for a user to view the outputs such that they are suitable for a given process or requirement.

Figure 9. User initiated sub-sectioning of LiDAR from the framework for a road-edge detection processing algorithm.

In this example both point or polygon spatial-constraint geometries can be created by a user, polygons have been created in this case as can be seen in Figure 9. This operation provides the geographical context for the LiDAR query where the user or process can choose between a 2D or 3D implementation. Either choice will result in a 3D LiDAR point-cloud being returned. However, in the 3D case the extra altitude (Z) parameter is added to the spatial query and can be used to bind the point-cloud in the Z domain, i.e. a road edge extraction algorithm will not require data above the road surface. These optimizations can significantly reduce the number of points returned which subsequently decreases the query time, required processing time and the rendering time.

Extending on from this example is the possibility of numerous different user or process driven use-cases that may require the spatially segmented LiDAR data to be returned based on attribute constraints as well. For every independent LiDAR attribute, including the table's primary key, an index exists such that searching any LiDAR table becomes a function of the index being searched and not a dependent process on the table's primary key. Thus, in the previous example we could further constrain the returned point cloud data by requesting only points whose intensity values fall within a certain range, which allows us to continue to leverage the power of the database. Using the optimizations that are inherent to the PostgreSQL database, a standard spatial query will optimize its search based on the spatial index and the primary key of the table. For queries that extend such a search, based on other attributes, the primary key element is easily replaced with the index of the chosen constraint. In the main, runtimes on these attribute constrained queries will be comparable in the single constraint case, although as the number of attribute constraints increases runtimes would logically extend.

6.2 WebGL Visualization

Once this spatially constrained query has been completed the resulting LiDAR can be visualized in a 3D viewer, possibly only for relevance or validation purposes, but could also be downloaded for use in a particular bespoke process or requirement. In the case of browser based visualization, WebGL, in a Chrome browser, has been leveraged where the functionally complete point clouds, representing the polygons selected in Figure 9, can be visualized and interacted with, as shown in Figure 10. WebGL is an emerging technology with developing standards, [11,12], where its functionality and power to render 3D data is driven by its ability to execute programs on the client system Graphics Processing Unit (GPU).

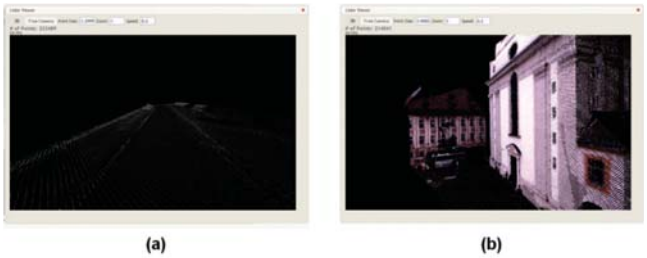


Figure 10. 3D point cloud WebGL viewer showing polygons selected from Figure 9. (a) mobile terrestrial data and (b) full color mobile terrestrial data.

Through this viewer, we can show both intensity and color point clouds depending on the LiDAR data, which, for this viewer, is a specific example of the LiDAR attribute constraint approach that can be leveraged through the database indexing functionality. The viewer can also be used, as shown in Figure 11, for visual inspection of both LiDAR and processed results. In this example, processed road edges and pole objects are plotted in the Map UI. A polygon is then selected and the resulting LiDAR and objects are displayed color coded in the viewer. This could be a powerful aid to organizations wishing to perform validation of processing results from large LiDAR surveys.

Figure 11. User defined LiDAR sub-section as displayed in the 3D point cloud WebGL viewer.

Extensions being built into this viewing platform include the functionality to download the segmented LiDAR dataset in a standardized file format. This download functionality is currently being implemented to support the existing LiDAR Exchange Format (LAS) standard, as defined in [5,26]. However, as is highlighted in [23] many formats exist for which bespoke download options could be developed but this would only be feasible on a requirements or use-case basis. Interestingly, the latest improvement on implementations of the LAS storage format

is LASzip, [9], where savings to the order of 10 to 20 percent of the original file size can be achieved and is completely lossless.

6.3 Results

Acquiring these spatially sub-sectioned areas of LiDAR data obviously comes with a processing and data transfer expense. There are a number of factors which influence the runtime of a query and subsequent display of the point cloud in the viewer. These include the following:

1. The size of the tables in the database.
2. The dimensions of the polygon.
3. The point density of the data.
4. The complexity of the SQL query.
5. And the transfer and subsequent loading of the points into the viewer.

It has also been shown in [22] that the configuration of hardware in such a system can lead to performance increases of greater than 4 times. For these reasons we will test the runtimes for both aerial data, where there is generally a uniform point density and table size, and mobile terrestrial data, where the point density can vary significantly as does the table size. Each test query selected a random sub-sectioning point and reported access times to LiDAR in these spatially constrained areas on a 20 query average. This timing procedure included the preparation of the spatial query object, the operation of the spatial SQL statement, runtime of the query on the database and transferring the resulting LiDAR 3D point cloud data to the WebGL viewer.

In Table 5, we display results for mobile terrestrial LiDAR data and demonstrate the effect of using 2D and 3D box spatial objects. The length parameter is maintained at 20 meters and represents a length along the known road center-line geometry and can be defined at any point forward, back or around the query space focal point. The width is the dimension orthogonal to the direction the vehicle was travelling. It can be seen in Table 5 that there is a strong correlation between the number of points returned by the spatial query and the runtime.

Table 5. Timing results for mobile terrestrial LiDAR comparing 2D and 3D spatial queries

Dimensions (m) – (length x width x height)		Time – average (s)	Points – average
3D	20x5x1	9.42	65596
2D	20x5	10.37	91030
3D	20x20x1	23.76	236645
2D	20x20	29.12	319307
3D	20x40x1	33.73	355854
2D	20x40	49.64	551149

Also in Table 5 we highlight the results for a 3D box spatial object query where height of the polygon is fixed at 0.54 meters above and below the surface of the road. Effectively, this example filters the LiDAR in an optimal way for the roadside feature extraction algorithms, where all points returned are within half a meter above and below the road surface. It is important to note that the 3D query, although more complex, has a quicker execution time in all cases. This highlights how important the PostGIS spatial indexing, as initiated and updated during the pg_bulkload data ingestion stage described in Section 4.2, is to the efficient retrieval of spatially optimized LiDAR point clouds.

Table 6. Timing results for aerial LiDAR

Dimensions (m) – (length x width)	Time – average (s)	Points – average
10x10	2.05	1386
50x50	5.31	30557
100x100	24.61	140521
250x500	60.98	1525803

Table 6 displays the results for our aerial data and further demonstrates the impact of point density on runtime. This allows us to view much larger areas of LiDAR data in the same time due to the point density. Finally, a plot of query time versus number of returned points for both mobile and aerial data is displayed in Figure 11. There is not a clear linear relationship between the number of points and runtime. This is not unexpected as there are a number of other factors that we have listed that impact on runtime. There also seems to be an increased efficiency in the aerial data case when querying larger areas. We believe this is due to the physical table size of the aerial data. In this experiment, its size is typically 10% that of the table size of the mobile data, this allows for much faster querying of the data.

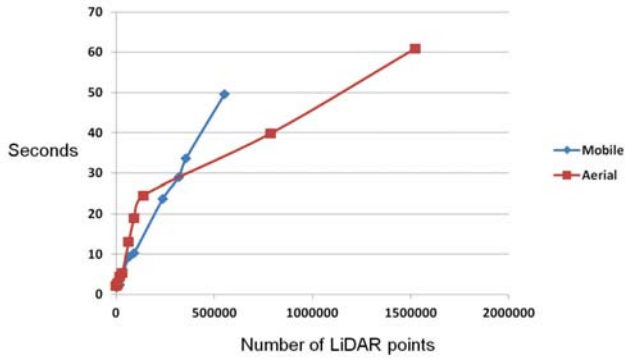


Figure 12. Graph plot for analysis of the timing results for mobile and aerial LiDAR data.

7. CONCLUSIONS

We have demonstrated a system which can store billions of LiDAR points and their attributes. It enables easy access to and interaction with the data in a web application through a 2D map and 3D point cloud viewer. The storage of LiDAR as a spatial object which can be queried allows for the development of spatially optimized workflows for processing algorithms. As highlighted in the previous sections, it is the LiDAR data

management aspects of the framework being integrated into a PostgreSQL object-relational database that fundamentally underpins our optimizations of the various spatial operations. Large volumes of 3D LiDAR point cloud data can now be easily segmented either geographically through our comprehensive LiDAR browsing web-application or based on non spatial constraints that model the traditional survey model. Based on our requirements for a LiDAR spatial segmentation solution we have shown how a PostgreSQL database solution with PostGIS spatial extensions can be an efficient and effective platform for such work.

8. FUTURE WORK

From this paper it is clear to see that we are developing four significant bodies of research work: LiDAR data import using pg_bulkload, spatial hierarchy generation using a concave-hull approach, data segmentation based on spatial and/or attribute constraints and point cloud visualization in a WebGL enabled browser. Going forward, it is hoped that we will be able to fully extend and publish on each of these sections of work in a more detailed way. Each of these four areas requires research to discover the optimal implementation. In particular, investigations into the use of GPU technology in data management and processing of DBMSs are only recently receiving attention. Also worth considering are two other possible directions, a hardware profiling comparison with other work that has been published by OpenTopography, and data scaling as a function of availability and accessibility to the data as opposed to the storage and size on disk overheads required to maintain such large volumes of LiDAR.

9. ACKNOWLEDGMENTS

Research presented in this paper was funded by the NRA research fellowship program, ERA-NET SR01 projects and by a Strategic Research Cluster grant (07/SRC/I1168) by Science Foundation Ireland under the National Development Plan. The authors gratefully acknowledge this support.

10. REFERENCES

1. Breunig, M. and Zlatanova, S. 3D geo-database research: Retrospective and future directions. *Computers & Geosciences*, (2011), 1-13.

2. Chen, Y., Zhang, H., and Fu, X. Organization and Query of Point Clouds Data Based on SQL Server Spatial. *ICCSIT*, (2010), 178-181.

3. Dielmo Technology. Lidar Online - The Lidar Social Network. 2012. https://www.lidar-online.com/product-list.php.

4. Ferede, H., Agency, N.G.-intelligence, and Mazzuchi, T.A. Multi-dimensional data discovery. *ASPRS/MAPPS*, ASPRS (2009).

5. Graham, L. The LAS 1.1 Standard. *Photogrammetric Engineering & Remote Sensing 71*, July (2005), 777-780.

6. Hongchao, M. and Wang, Z. Distributed data organization and parallel data retrieval methods for huge laser scanner point clouds. *Computers & Geosciences 37*, 2 (2011), 193-201.

7. Hua, L.I.U., Zhengdong, H., Qingming, Z., and Peng, L.I.N. A database approach to very large LiDAR data management. *ISPRS Congress*, ISPRS (2008), 463-468.

8. Idaho LiDAR Consortium. Commercial LiDAR Tools. 2011. http://www.idaholidar.org/tools/commercial.

9. Isenburg, M. LASzip : lossless compression of LiDAR data. *European LiDAR Mapping Forum*, (2012).

10. Jaeger-Frank, E., Crosby, C., Memon, A., et al. A Three Tier Architecture for LiDAR Interpolation and Analysis. In V. Alexandrov, G. van Albada, P. Sloot and J. Dongarra, eds., *Computational Science – ICCS 2006*. Springer Berlin / Heidelberg, 2006, 920-927.

11. Khronos Group. WebGL Specification 1.0. 2011. https://www.khronos.org/registry/webgl/specs/1.0/.

12. Khronos Group. WebGL Specification Draft. 2012. http://www.khronos.org/registry/webgl/specs/latest/.

13. Krishnan, S., Viswanath Nandigam, C., Crosby, M.P., Cowart, C., Baru, C., and Arrowsmith, R. OpenTopography: a services oriented architecture for community access to LIDAR topography. *Earth Science*, (2011).

14. Lewis, P., Mc Elhinney, C.P., Schön, B., and Mc Carthy, T. Mobile Mapping System LiDAR Data Framework. *3D Geo-Information 2010*, (2010).

15. Matejicek, L., Engst, P., and Janour, Z. A GIS-based approach to spatio-temporal analysis of environmental pollution in urban areas: A case study of Prague's environment extended by LIDAR data. *Ecological Modelling 199*, 3 (2006), 261-277.

16. Mc Elhinney, C.P., Kumar, P., Cahalane, C., and McCarthy, T. Initial results from European Road Safety Inspection (EURSI) mobile mapping project. *ISPRS Commission V Technical Symposium*, ISPRS (2010), 440-445.

17. Mc Elhinney, C.P., Lewis, P., and Mc Carthy, T. Mobile Terrestrial LiDAR Data-Sets in a Spatial Database Framework. *MMT 2011, 7th International Symposium on Mobile Mapping Technology*, (2011).

18. Ming, G., Yanmin, W., Youshan, Z., and Junzhao, Z. Research on Database Storage of Large-Scale Terrestrial LIDAR Data. *2009 International Forum on Computer Science-Technology and Applications*, IEEE (2009), 19-23.

19. Mumtaz, S.A. Integrating Terrestrial Laser Scanning Models into 3d Geodatabase. *ICAST*, (2008), 124-130.

20. NSF OpenTopography. LiDAR Tool Registry. 2011. http://opentopo.sdsc.edu/gridsphere/gridsphere?cid=contributeframeportlet&gs_action=listTools.

21. NSF OpenTopography. Lidar Access Facility. 2012. http://opentopo.sdsc.edu/gridsphere/gridsphere?cid=datasets.

22. Nandigam, V., Baru, C., and Crosby, C. Database Design for High-Resolution LIDAR Topography Data. *Scientific and Statistical Database 6187/2010*, (2010), 151-159.

23. Samberg, A. An Implementation of the ASPRS LAS Standard. *ISPRS Workshop on Laser Scanning and SilviLaser*, (2007), 363-372.

24. Schön, B., Bertolotto, M., Laefer, D.F., and Morrish, S.W. Storage, Manipulation and Visualization Of Lidar Data. *International Society Of Photogrammetry and Remote Sensing*, 3D-ARCH 2009 (2007).

25. Sharma, N., Parikh, J., and Clark, M. A Lidar Collaboratory Data Management System. *2006 IEEE International Symposium on Geoscience and Remote Sensing*, (2006), 817-820.

26. The American Society for Photogrammetry & Remote Sensing. *LAS 1.4 Draft Specification*. 2011.