Research Repository UCD

Title	A Big Data Approach for 3D Building Extraction from Aerial Laser Scanning
Authors(s)	Aljumaily, Harith, Laefer, Debra F., Cuadra, Dolores
Publication date	2016-05
Publication information	Aljumaily, Harith, Debra F. Laefer, and Dolores Cuadra. “A Big Data Approach for 3D Building Extraction from Aerial Laser Scanning” 30, no. 3 (May, 2016).
Publisher	American Society of Civil Engineers
Item record/more information	http://hdl.handle.net/10197/7450
Publisher's version (DOI)	10.1061/(ASCE)CP.1943-5487.0000524

Downloaded 2024-04-17 21:49:44

The UCD community has made this article openly available. Please share how this access benefits you. Your story matters! (@ucd_oa)

© Some rights reserved. For more information

BIG DATA APPROACH FOR 3D BUILDING EXTRACTION FROM AERIAL LASER SCANNING

Harith Aljumaily[(1)], Debra F. Laefer[(2)], and Dolores Cuadra[(3)]

ABSTRACT: This paper proposes a Big Data approach to automatically identify and extract buildings from a digital surface model created from aerial laser scanning data. The approach consists of two steps. The first step is a MapReduce process where neighboring points in a digital surface model are mapped into cubes. The second step uses a non-MapReduce algorithm first to remove trees and other obstructions and then to extract adjacent cubes. According to this approach, all adjacent cubes belong to the same object and an object is a set of adjacent cubes that belong to one or more adjacent buildings. Finally, an evaluation study is presented for a section of Dublin, Ireland to demonstrate the applicability of the approach resulting in a 92% quality level for the extraction of 106 buildings over 1 km^2 including buildings that had more than 10 adjacent components of different heights and complicated roof geometries. The proposed approach is notable not only for its Big Data context but its usage of vector data.

[1] Department Computer Science and Engineering Carlos III University of Madrid, Av. Universidad 30 – 28911 – Madrid, Spain. haljumai@inf.uc3m.es

[2] School of Civil, Structural and Environmental Engineering; U3D Printing Hub & Earth Institute, University College Dublin, Newstead G25, Belfield, Dublin 4, Ireland. debra.laefer@ucd.ie

[3] Department Computer Science and Engineering Carlos III University of Madrid, Av. Universidad 30 – 28911 – Madrid, Spain. dcuadra@inf.uc3m.es

KEYWORDS: Building Extraction, MapReduce, Big Data, LiDAR, Digital Surface Model, Aerial Laser Scanning

INTRODUCTION

Massive streams of remote sensing data are rapidly becoming less expensive and more readily available at a higher quality. However, processing them efficiently requires appropriate tools, especially for querying and visualization. An example of where this is highly relevant is in large-scale, urban, spatial datasets where data exists from many forms and at various granularities [e.g. high resolution satellite, Light Detection and Ranging (LiDAR), and distributed (wireless) sensor networks, etc.]. In this context, important information can be extracted from LiDAR point clouds such as the location, orientation, and size of objects, as well as possible damage (Walsh et al. 2013, Gao et al. 2014). Unfortunately, traditional approaches are not well suited for the current generation of data and will only become less so in future years. For example, a 500m*500m (or 0.25km^2) tile of LiDAR point cloud data could contain more than 65,500,000 three-dimensional (3D) points. Making these points compatible with a Spatial Data Model (i.e., generating geo Identification Key and Spatial Objects for each point) requires more than 10GB in relational database storage. For this reason, writing queries against this huge data storage arrangement is complicated and time consuming.

Since the increasing density and temporal recurrence of such data already challenges the capabilities of traditional Geographic Information Systems and other spatial databases, developing appropriate querying tools to significantly improve the outcomes of relevant data retrieval exercises is quite critical. Furthermore, the complexity and nature of spatial data makes them ideal datasets for applying parallel processing (Cary et al. 2009). Thus, large-

scale, urban, spatial data can be stored and analyzed in big data platforms, which enables identification of changes in geographic patterns (NESSI 2012).

"Big data" is a high scalability and real time platform created to improve web-based search engines (e.g. Yahoo, Google) and smart data analysis through parallel and distributed computing. The approach can handle large and complex datasets that are beyond the means of traditional databases. Despite their short history, these platforms have been deployed extensively with further rapid adoption predicted (Özdemir et al. 2013; Floridi 2012). Arguably, data mining in Digital Surface Models (DSMs) requires Big Data strategies to overcome scalability issues (Batty et al. 2012). A DSM has many possible applications including as an interim step in the transformation of point clouds into solid models compatible for computational analysis (Truong-Hong et al. 2013). As such, this paper endeavors to extend the viability of Big Data approaches to LiDAR data processing, in particular for city-scale computational modeling, which is an area of significant interest to the civil engineering community (Hinks et al. 2009). As the hardware limitations of working at such a scale continue to decrease, the logistics of populating those systems for micro-climate modelling, disaster prediction, and tunnel risk assessment continue to gain prominence. The auto-identification and extraction of data affiliated with individual buildings is a critical step in this process.

RELATED WORK

To date, few studies have been conducted regarding to the treatment of LiDAR in a Big Data context. In related work, Wu et al. (2007) introduced an approach to automatically align a combination of satellite images and vector data for roads. They divided the data into a set of tiles and automatically computed the best imagery-to-vector translation within a tile by using

a parallel programming model with MapReduce; MapReduce is a software model used to support parallel computing of huge sets of data and consists of two functions Map and Reduce, which operate using key-value data types. Two years later Zhang, et al. (2009) described how spatial queries could be adopted and expressed in a MapReduce model. By using several spatial queries performance evaluations, they proved that MapReduce is appropriate for small-scale clusters and computing intensive applications. The next year Wang, et al. (2010) proposed methods to improve the performance of spatial computation using a key-value based model. Their experiments showed that the MapReduce based spatial system could significantly outperform a traditional Database Management System (DBMS). Subsequently, Liu et al. (2012) used MapReduce to solve two common spatial problems. The first was the bulk-construction of R-Trees (an indexing mechanism for spatial search query processing). The second was the computation and storage of the quality characteristics of aerial digital image using metadata to improve the display of image-based mosaicing. More recently, Aji, et al. (2013) demonstrated the efficiency and scalability of Hadoop-GIS by running queries on a large cluster. In this demonstration, they explained how spatial queries could be translated into MapReduce operators, optimized, and executed in Hadoop.

Such approaches stand in strong contrast to the traditional methods developed for the storage and investigation of LiDAR data, which have not considered the rapid trajectory of escalating data density. Most approaches developed to date have been for datasets of no more than 10 points per square meter, which does not reflect the current system capabilities of not only 50 points per square meter but the existence of full-wave form data that has more than 250 pieces of information for each data point. Manipulation of such dense data sets will soon become a requirement for a wide variety of data mining activities. One common task is

building extraction. A sampling of some of the notable, data-driven approaches to building extraction from remote sensing data are described below.

In 2004, Rottensteiner et al. (2004) used the Dempster-Shafer theory for data fusion to the detect buildings from a combination of LiDAR data and digital images. The approach used three consecutive steps; (1) building detection, (2) roof plane detection, and (3) roof boundary determination. In (Pradhan et al. 2007) considered the affiliated computing requirements for city-scale modelling for disasters. These requirements are standardized data specifications, middleware services and Web-enabled, distributed computing. In Mayunga et al. (2005) proposed a semi-automatic process using a snake model and radial casting for building data extraction from raster data. The approach required the manual identification of the approximate centre of a building, from which the snake contour points were automatically generated. Following this, Theng el al. (2006) employed the same snake contour model but initialized it with a circular casting algorithm, instead of radial casting algorithm. In contrast, San et al. (2010) applied the well- established Hough transform with it. Soon after, Dal Poz and Galvanin (2011) extracted building outlines from a LiDAR-based DSM using a two-step approach: (1) extract patches of aboveground objects and (2) identify edges that correspond to building roof contours using a Markov-random-field-based energy function. Subsequently, Sugihara and Kikata (2012) employed automatic generation of 3D building models through a integrating geographic information systems (GIS) and computer graphics. Awrangjeb et al. (2013) developed an approach to automatically extract 3D roofs by integrating LiDAR data and multispectral orthoimagery. In that method, raw LiDAR points were separated into ground points and non-ground points. The non-ground points were segmented using a technique of image line guided segmentation to extract the roof planes. Li et al. (2013) also proposed a method to fuse LiDAR data and optical imagery to extract different building

features. Their method consisted of four steps: 1) filtering, 2) building detection, 3) wall point removal, and 4) roof patch detection. As part of step 2, an improved detector was used to extract building edges from optical imagery. In the final step, the results from the prior steps were integrated using a mathematical morphology.

Soon after Jochem et al. (2012) proposed a workflow that uses a combination of raster and point cloud based GIS analysis. Although this workflow focused on roof planes, it can be applied to other surface objects such as vegetation. In this workflow, in order to facilitate the processing tasks, the large point cloud dataset is stored and divided into several smaller tiles using a traditional database PostgreSQL/PostGIS solution.

Most recently, Abdullah et al. 2014 proposed a segmentation technique for automatic building detection and roof plane extraction considering only non-ground LiDAR points, in which finding neighbouring points involved segmenting LiDAR starting from the maximum height and progressing downwards. The segmentation was done along individual lines at each height level and depending on the distance between points. These lines were then used to form planes. After all of planes were extracted, non-coplanar points (which represented trees and other non-building structures) were removed by an ad hoc rule-based procedure.

While these various studies have produced important results for building extraction, they generally employ either raster data or a combination of raster data and vector data. Rottensteiner et al. (2004) noted that using raster data in building extraction introduces problems related to shadows, occlusions, and poor contrast. In light of these criticisms, an alternative approach is herein introduced that attempts to facilitate automated building extraction task processing only vector data from aerial LiDAR.

THE PROPOSED APPROACH

The goal of this work is to introduce a new, fully automatic method for building extraction from LIDAR data that needs no pre-processing. The proposed two-step approach involves querying a LiDAR point cloud in the Hadoop platform, which is an open-source software framework written in Java and is used to run distributed applications with huge amounts of data among a Hadoop cluster, which consists of a set of compute nodes (computers). Step 1 is a MapReduce process, in which neighboring points are mapped into cubes. Step 2 employs a non-MapReduce process to extract objects where all of the adjacent cubes are considered to belong to the same object. Although a LiDAR point cloud can include a variety of data, such as 3D coordinates, timestamps, intensity, and RGB measurements, the proposed approach only requires the 3D coordinates as part of the extraction process. Once the point cloud is generated from the aerial laser scanning, it is used directly in the approach without any pre-processing.

The basic idea exploits the distance between every pair of points in the point cloud. The main problem is the computational expense. Applying the MapReduce function considerably reduces the problem size, thereby making it more manageable. This is done by taking the original DSM points generated from the point cloud and mapping them onto a 3D grid in the form of a group of cubes. Notably, this approach is independent of data density or multiple returns, however a higher density will allow a smaller cube size to be selected, which will in turn improve the fineness of detail detection.

The MapReduce software framework is composed of two functions: the function 'Map' processes the original data into key/value pairs, and the function 'Reduce' takes these pairs

and merges them in a way that all values corresponding to a specific key are combined into a single set. Once this is complete, the non-MapReduce function extracts neighbouring cubes, where each set of cubes represents a physical object. These steps (Figure 1) are explained in detail in the following sections.

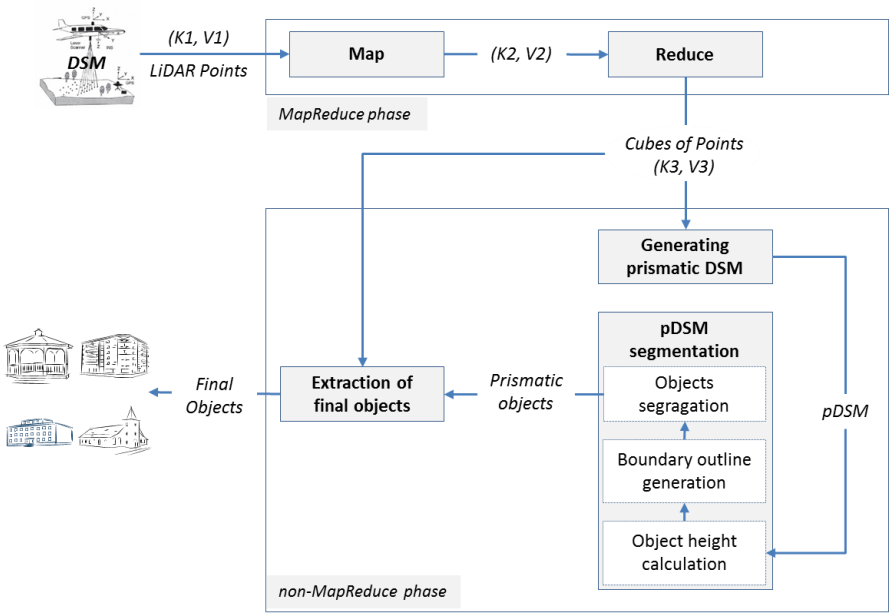


Figure1. Approach design

Mapping neighbour points (MapReduce)

The nearest neighbour algorithm is a well-established and extensively used approach (e.g. Morgan and Tempfli 2000; Lee et al. 2008; Wang and Shan 2009) but when applied indiscriminately to a large dataset is computationally expensive (Athitsos et al. 2008). For this reason, the first part of the proposed approach aims to reduce the exploration neighbourhood. To do that, a distance threshold (D) is defined. In dense urban areas, determining D can be crucial, as many buildings abut one another. In this work, the distance was empirically selected as 0.5 m. Ideally this should be determined as a function of both the data density and typical building spacing.

Once D is defined, all the DSM points are segmented and mapped according to cubes with dimensions D x D x D. Namely, if two neighbour points belong to a single cube then these two points belong to the same object. The other way to describe this concept is to consider the volume of the dataset as an octree of cubes of dimensions D x D x D and to understand all neighbouring points fitting in a single cube as belonging to only one object.

Consider a DSM consisting of a set of 3D points; $Point_1$ (x_1, y_1, z_1), $Point_2$ (x_2, y_2, z_2), …, $Point_n$ (x_n, y_n, z_n), where n is the number of points, and m is a set of identical cubes [$Cube_1$ (X_1, Y_1, Z_1), $Cube_2$ (X_2, Y_2, Z_2), …, $Cube_m$ (X_m, Y_m, Z_m)]. The mapping of a point to the corresponding cube is done according to the following rule:

$Point_i$ (x_i, y_i, z_i) $\in$ $Cube_j$ (X_j, Y_j, Z_j) where:

$$X_j = fix(x_i/D)$$

$$Y_j = fix(y_i/D)$$

$$Z_j = fix(z_i/D)$$

The function fix(v) truncates the value to the greatest integer less than or equal to v.

For example, for a given $point_i$ (x_i=1.4 m, y_i=0.25 m, z_i=0.8 m) and D = 0.5 m, then this point is mapped to the $cube_j$ (X_i=2, Y_i=0, Z_i=1) see figure 2(a). Applying this rule to the set of points shown in figure 2(b) results in figure 2(c).

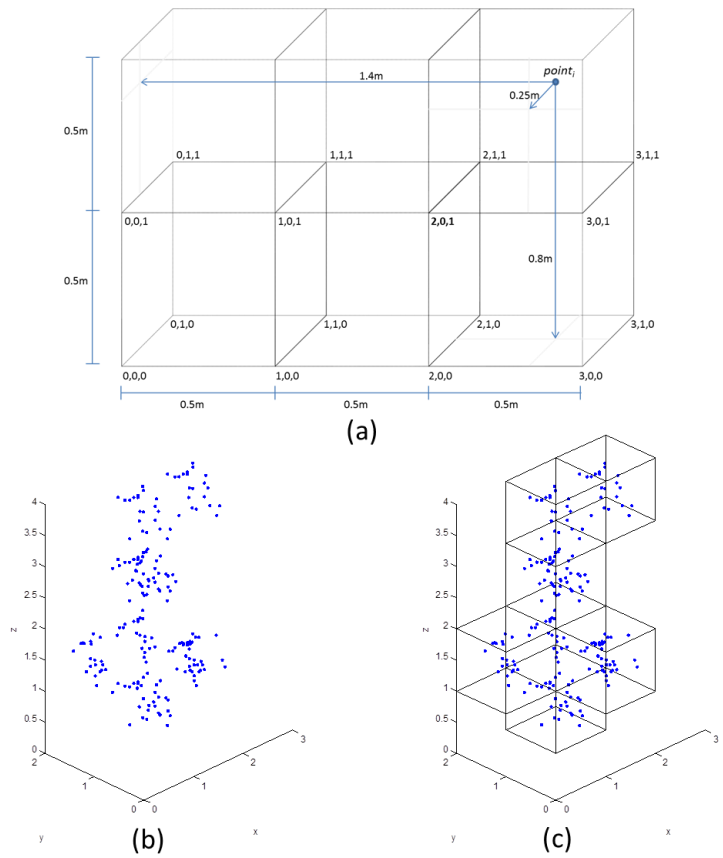


Figure 2. 3D point mapping

Since the Mapper and the Reducer classes in Hadoop have the following general form (White 2012):

Mapper <K1, V1, K2, V2>

Reducer <K2, V2, K3, V3>,

then the Mapper class receives a pair in the form of (K1, V1) and issues a list in the form of (K2, V2). The Mapper key K1 is the file name, while the dataset may be a single file or a set of files in a given directory. The value of Mapper V1 is the content of the line in the corresponding file. Now, only the 3D point coordinates are employed. The Mapper class issues a list of pairs of (K2, V2), where K2 represents the cube, which is calculated according the previous rule, while V2 represents the current point being processed, as follows:

map (K1, V1, context){

```
    K2 = Cube(fix(V1.x/D), fix(V1.y/D), fix(V1.z/D));
    V2 = Point(V1.x, V1.y, V1.z)
    context.write(K2, V2);
}
```

Next, the Reducer receives the list of the pairs (K2, V2) and issues a list of the pairs (K3, V3), where K3 is the cube coordinate, as a unique key in the list, while V3 is a list of all points that belong to K3. The final result of this step will be the list (K3, V3), where all the cubes K3 in the list are sorted in descending order according to the height coordinate of the cubes (i.e., the first cube in the list will be the highest cube in the highest object H_i). The Reduce function is shown as follows:

```
reduce (K2, V2, context) {
    K3 = K2;
    for (v : V2) V3 += v;
    context.write(K3, V3);
}
```

Objects extraction (non-MapReduce)

Although the two steps of the approach are executed in independent processes, this second step is executed after step 1 is complete. The main objective is to remove trees and other obstructions and then to disjoin the results of the previous step (K3, V3) into objects such as buildings, roads, green areas, etc. The basic idea is to find the neighbour points of an object by visiting cubes instead of points. This is done by applying the following steps:

Generating prismatic DSM to remove trees and obstructive objects

To remove trees and other undesirable objects, a prismatic DSM (pDSM) is generated from the original DSM being processed. The pDSM is a totally filled 3D city model. In Rottensteiner, et al. (2002), the prismatic model is derived from the boundary polygons of the building regions using the average building heights. The accuracy of this height averaging was in the range of ±5 m. In the approach presented herein, generating a pDSM is done by filling the DSM being processed by new cubes with the same side length D*D*D. This is the same process as filling a mould with any material. Figure 3 illustrates how this process is performed in the presence of a high density of trees adjacent to buildings. In addition, in the bottom left corner of figure 3a, there is a tower crane. This would continue to obstruct the building extraction process, if a pixel-based approach is used. In contrast, by viewing the underside or bottom view of the same DSM (figure 3b) the presence of each building appears as a deep hole, the depth of which corresponds to the respective building height. If multiple buildings are connected, this bottom view displays these connected holes. A careful examination of figure 3b shows small, insignificant holes distributed across the DSM. These holes indicate the presences of trees, a tower crane, cars, etc. This phenomenon is produced because the laser scanner gives very rich surface details of all existing objects, but does not penetrate the surface. A “hole” represents the perimeter and volume of a building. Trees and other such objects do not appear in the same way, since the area of their connection to the ground is insignificant.

The result of generating the pDSM where a simple rule is used is shown in figure 3c. This rule states that for each NotEmpty(Cube(X, Y, Z)) $\in$ DSM, then the set of cubes ($Cube_0$(X, Y, 0*D), $Cube_1$(X, Y, 1*D), …, $Cube_n$(X, Y, Z-D)) $\in$ pDSM.

This algorithm starts from the first cube (0, 0, 0) in the original DSM, and then checks if the cube of the original DSM being processed is not empty. Once a non-empty cube is located, then the respective portion of the pDSM is filled by a set of cubes starting at 0 and finishing at Z-D. The X and Y coordinates are fixed. This is then repeated. For example, if a cube (0,0,3) is not empty in the original DSM, then the set of cubes ($Cube_0$(0,0,0), $Cube_1$(0,0,1), $Cube_2$(0,0,2)) are added to the pDSM being processed, where 0,1,2,3 are an indexing array.

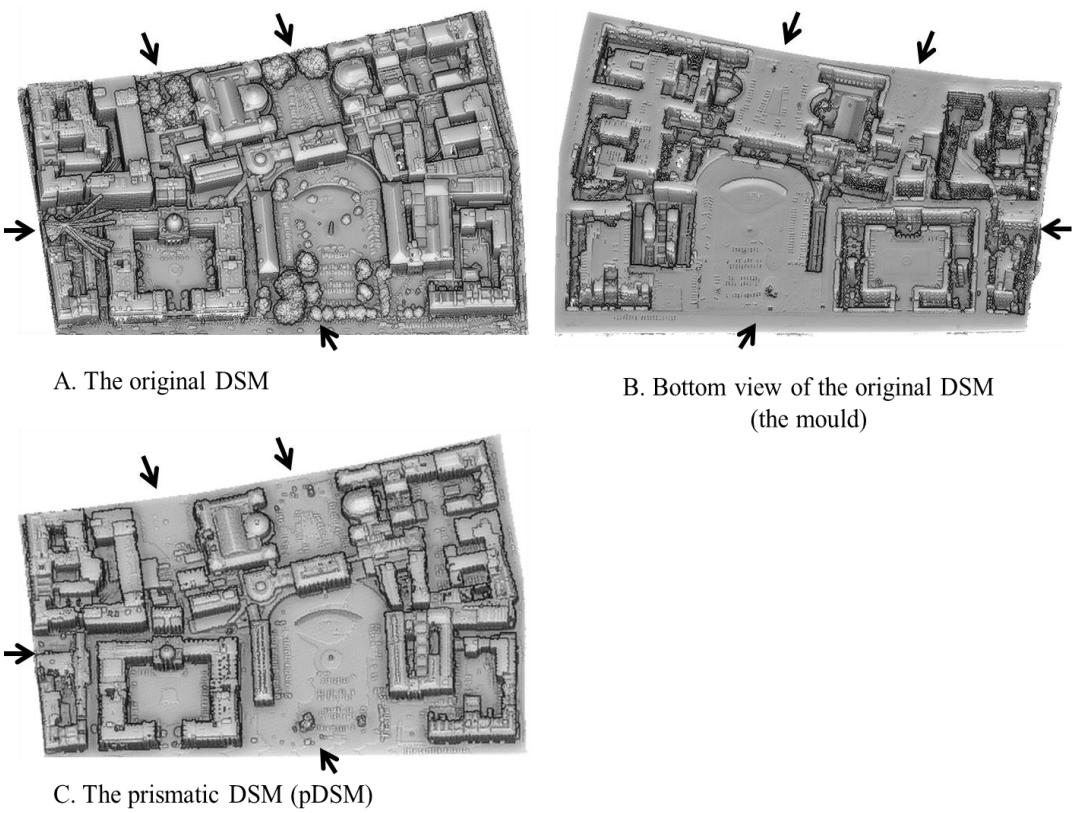


Figure 3. Removing trees and other obstruction objects.

pDSM segmentation

Once the trees and other small objects are removed in the processing of generating the pDSM, then the aim is to disjoin the pDSM into several object models that will be used as references to extract the final objects.

In order to segment a pDSM into separate object models, consider the $Cube_1$ (X_1, Y_1, Z_1) as the first/highest cube of an object model Obj_1. Then all the neighbouring cubes that are visited starting from this cube will be mapped onto Obj_1.

As a rule for a given starting cube $Cube_i$(X_i, Y_i, Z_i), there exists an adjacent cube $Cube_j$(X_j, Y_j, Z_j) where the value of X_j ranges from X_i-1 to X_i+1, the value of Y_j ranges from Y_i-1 to Y_i+1, and the value of Z_j ranges from Z_i-1 to Z_i+1.

This rule is repeated for each neighbouring cube, until no more neighbouring cubes are found. One cube can have as many as 26 adjacent cubes or as few as none. If there is no adjacent cube, the cube being processed is considered to be a boundary cube. To preclude revisitation of a cube, each cube is removed from the candidate dataset, once it has been visited.

While applying the previous rule extracts all prismatic model details (irrespective of complexity), a small problem persists. Near to the ground level, the algorithm will expand to include all the ground cubes because of their adjacency. To overcome this problem, an object's outline is derived during segmenting the pDSM. In this case, the outline should be derived slightly higher than the ground level. So, in order to derive an object's outline correctly, the height of this object should be calculated.

Significant research has been done to address the issue of the heights of objects. For example, in Zhang et al. (2006) building heights were derived by averaging the elevation differences between building measurements and the digital terrain model (DTM). In the same direction, Abdullah at el. (2014) used a height threshold, based on a DTM to divide the LiDAR point cloud into ground and non-ground points. In the work presented herein, the main problem is that object heights need to be calculated "on the fly" without using any exterior tool or relying on any user interaction.

In order to calculate the height of an object, the rule ($h_i = H - T - z_i$) is applied, where h_i is the height of the object being processed, H is the total height of the DSM, T is the ground thickness, and z_i is the Z coordinate of the highest cube of the object.

An example of the application of this rule is shown in figure 4. The pDSM is mapped to identical cubes. The height of the pDSM is H. This value H can be calculated by applying the formula H = Zmax-Zmin, where Zmax is the coordinate of the highest cube in the pDSM, and Zmin is the coordinate of the lowest cube of the same pDSM. In the example, H=14m. However, this height does not represent the height of the objects, because it includes a part of the ground cubes. For this reason, the main objective here is to calculate an approximate value of *T*, which represents the thickness of the ground near the objects being processed and finally to find the approximate height of each object. To calculate *T*, the main pDSM is divided into multiple small models ($pDSM_1$, $pDSM_2$…) depending on the terrain type. This is to say, if the slope of the natural ground level is very high, then the pDSM being processed is divided into smaller areas. In this example, only two models ($pDSM_1$, $pDSM_2$) are used. For each $pDSM_i$ the maximum distance d_i between the ground and the maximum height of the main pDSM is calculated. In the example, the maximum distance of the $pDSM_1$ is d_1 = 9m and in $pDSM_2$ is d_2 = 11m. So, the average of the distances in all the pDSM is d_{av}= (d_1 + d_2)/2 = 10m. Thus, T will be H-d_{av} = (14-10) = 4m.

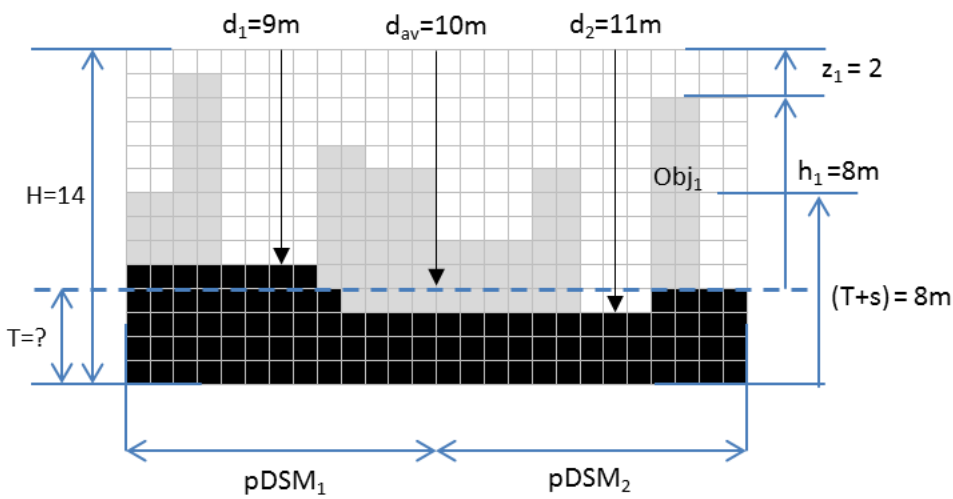
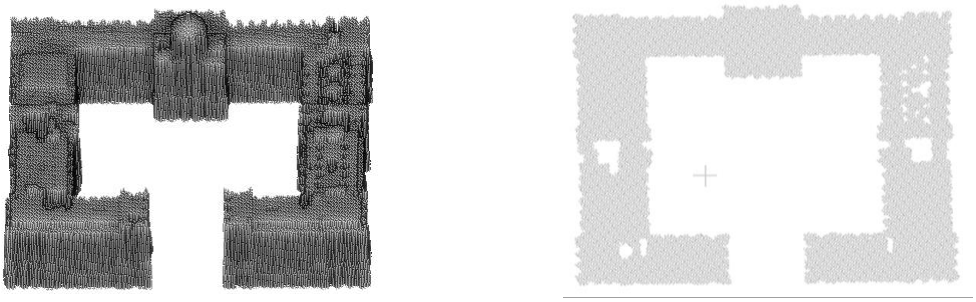


Figure 4. Object model height calculation.

Once the approximate thickness *T* is determined for the ground near the objects being processed, then boundary outline calculation begins. The most important issue in this process is to establish approximately the locations at which the outline should be calculated. The height should be measured slightly above the natural ground level to prevent the algorithm expanding to include adjacent ground cubes, but the height should be as low as possible to retain all architecture details. For the objects of the DSM of figure 4, the outline of obj_i is calculated as the height (T + s), where *s* is a small offset from the ground that is automatically calculated based on the height of the object. Through experimentation of a dense urban dataset (see validation section), a distance of $s = 0.5 * h_i$ has provided good results. In the Figure 4 example, for obj_1 the height of the outline calculation can be (T + s) = (4 + 0.5*(14–4–2)) = 8m.

An outline of an object model is generated by using cube projection onto a two-dimensional (2D) plane, where the projection is the vertical shadow of a cube, and the plane represents the ground surface. All cubes found in the level *h* are projected onto an empty plane. Each cube is projected onto the plane as a 2D point. That is to say, the projection of the Cube(X, Y, Z) of a building creates the outline point Opoint(X, Y) onto the corresponding plane. The resulting projection is the outline of the corresponding building in plan view (see figure 5).



A. an object model from pDSM B. Boundary outline of the object in A.

Figure 5. Two-dimensional boundary outline.

Once the approximate boundary outline is calculated, then the neighbour visiting process in the horizontal direction is controlled. Namely, when the algorithm is applied at the ground level, it extracts only the cubes whose coordinates match with the coordinates of the corresponding outline points. As a rule, if a building outline is created from the set of the points {Opoint (x_1, y_1), Opoint (x_2, y_2), …, Opoint (x_n, y_n)}, then any Cube (X_i, Y_i, Z_i) being processed belongs to this building, if and only if, the pair (X_i, Y_i) matches any pair (x_j, y_j) in the corresponding outline. In this case, the algorithm allows the extraction of Cube (X_1, Y_1, Z_1), Cube (X_2, Y_2, Z_2), …, Cube (X_m, Y_m, Z_m), where the pairs (X_1, Y_1), (X_2, Y_2),…, (X_m, Y_m) match the corresponding pairs in the outline, irrespective of the values Z_1, Z_2, ..., Z_m. Notably, the algorithm starts from the highest cube in the dataset and moves downwards towards the lowest one. This is done, thanks to the first phase of the approach (MapReduce phase), where the cubes of the points are generated in a decreasing order along the Z-axis. So, the first cube in the generated dataset is the highest cube in the highest object of the pDSM being processed. The algorithm repeats, until no more cubes are found in the pDSM. At the beginning, it creates an empty object (*obj*) to save all the cubes related to the highest object of the pDSM. The highest cube cube(X, Y, Z) is first segregated from the pDSM and then added to *obj*. Then all the neighbour cubes related to the first cube are added to *obj*. At the same time, these cubes are segregated from the same pDSM dataset. Once the highest object is segregated, the algorithm next segregates the highest remaining object in the pDSM. This continues until all objects are segregated from the pDSM. At the end of the extraction process, the pDSM dataset being processed will be empty, because all of its cubes will have been segregated and moved automatically to the corresponding files. The example in figure

(6) shows the segregation process, where the pDSM contains three objects *(a)*, *(b)*, and *(c)*. In the first loop (i_1) of the algorithm the object *(b)* is segregated automatically from pDSM (because it is the highest object in the set), and the remaining set becomes pDSM'. In the second and the third loops (i_2 & i_3) object *(c)* and object *(a)* are segregated, respectively. If the pDSM is empty (such as pDSM'''), then the algorithm stops.

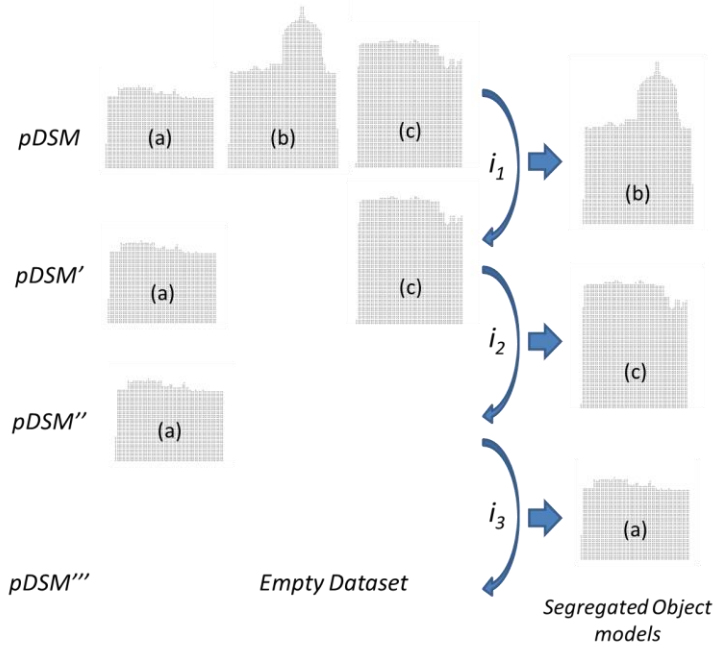
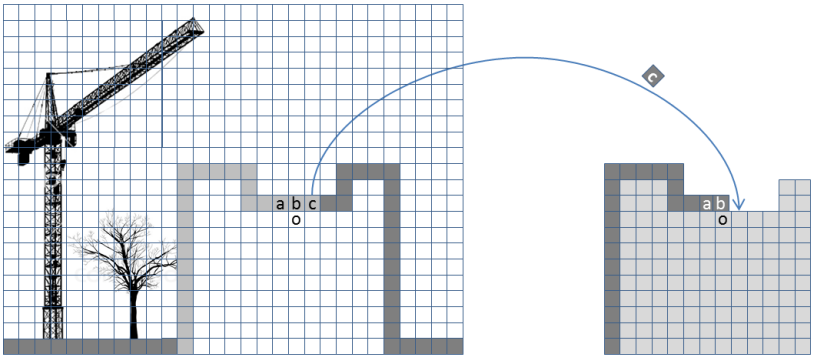


Figure 6. The automatic segregation process of the prismatic object models

Extraction of final objects

The main objective is to extract the final object from the original DSM. This requires using the original DSM and the prismatic object model corresponding to that object. Figure 7.A shows a DSM of a building object (gray cubes) with adjacent trees and a tower crane. When this DSM is subjected to the first phase (MapReduce phase), the DSM will be segmented into a set of cubes of points. When these cubes are subjected to the tree removal phase, a prismatic object model for the same building will be generated (see figure 7.B, the light gray cubes). In this step, all the cubes that have adjacent objects into the prismatic object model are segregated from the original DSM. For example, the cube (o) in the original DSM (figure

7.A) is the same as the cube (o) in the generated object model [i.e., it has the same coordinates (X, Y, Z)]. So in this step, all the objects (a, b, c) of the original cubes will be moved to the cubes (a, b, c) in the new and final object, because of the adjacent neighbourhood rule. If the adjacent cube is null in the original DSM, then this cube will not be moved. Notably, this algorithm moves only the immediate adjacent neighbourhood, and it moves only the cubes that have adjacent neighbourhoods. For this reason, the trees and tower crane are not moved to the final object, although the tree is adjacent to the cubes that have been moved.



A. a DSM of a building B. The prismatic model of the building

Figure 7. The automatic extraction process of the final objects

As a rule, if a $cube_i(X_i, Y_i, Z_i) \in$ a Prismatic Object, then there exists an adjacent cube $Cube_j(X_j, Y_j, Z_j)$ where the value of X_j ranges from X_i-1 to X_i+1, the value of Y_j ranges from Y_i-1 to Y_i+1, and the value of Z_j ranges from Z_i-1 to Z_i+1. This uses the same rule as in section (pDSM segmentation), but with the additional restriction that it is not allowed to grow beyond the cube being processed.

RESULTS, EVALUATION, AND DISCUSSION

This section evaluates the proposed algorithm for building extraction from an architecturally dense and complex portion of Dublin, Ireland. Within a 1 km^2 study area there are 9 tiles

from a DSM (see figure 8). Each tile was saved into a separate file. The preparation of this DSM included several steps that are outside of the scope of this work, including flight path planning, as well as data collection, registration, and filtering (as described in Truong-Hong, 2011). The DSM contained 226,319,306 points. Figure 8 shows the corresponding pDSM, where all the trees and other obstructions present in the initial DSM have been removed.

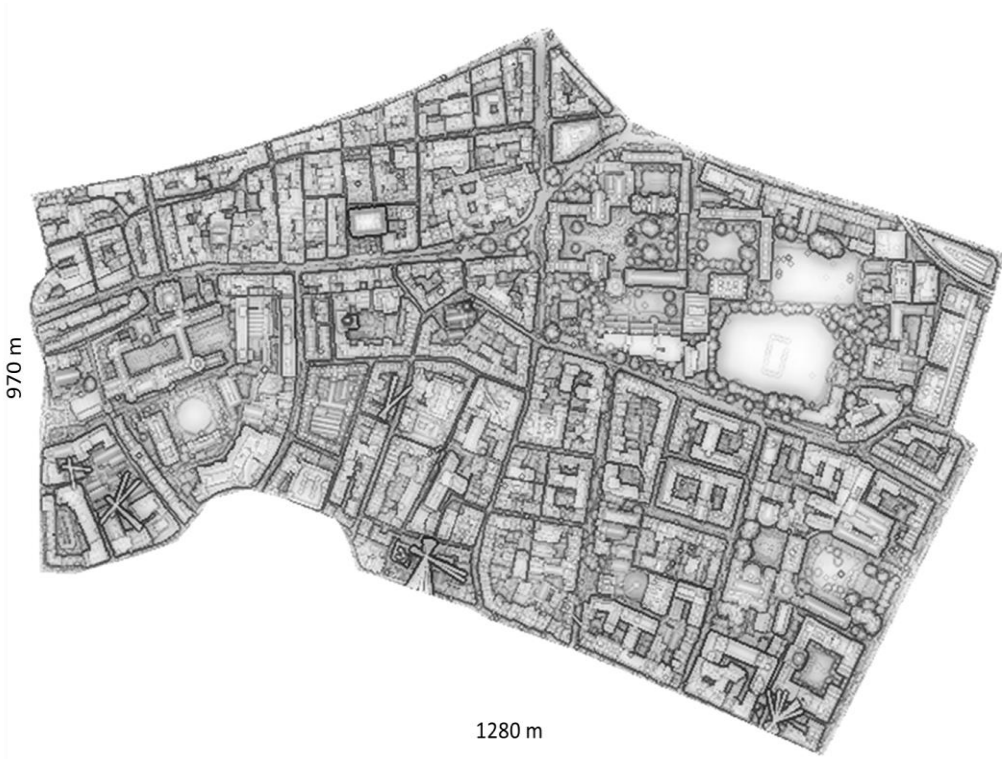


Figure 8. Initial DSM

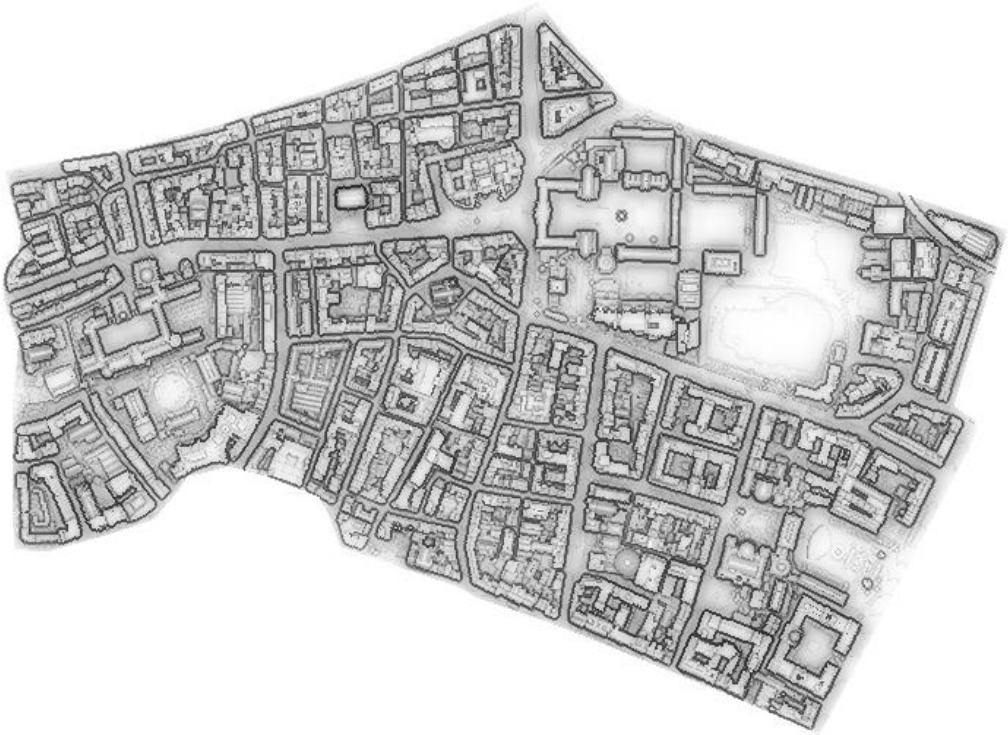


Figure 9. Newly generated pDSM from the initial DSM (note the absence of all trees

Figure 10 shows the set of outputs of the algorithm. Notably, the algorithm converts the DSM being processed into objects stored in separate files. However, in this evaluation study only the 106 objects that form the most significant objects in the study area were processed. These extracted objects are shown in figure 10 in a single image to ease comparison with the DSM (Figure 8) and the corresponding pDSM (Figure 9).

Figure 10. Composite output of the extracted buildings from the algorithm

Figure 10 shows that most of the 106 objects have been isolated successfully. To quantitatively evaluate the algorithm's outputs, measurements of correctness, completeness and fitness measure (F-measure) were used. Normally, these measures are calculated by taking the difference between the extracted buildings and the reference buildings (Maurya et al. 2012). Several different input parameters have been proposed. Ekhtari et al. (2008) used the building pixels, while Song and Haithcoat (2005) determined the volumetric difference. For the proposed approach, a comparison between points in the extracted buildings versus points in the reference buildings was considered.

To obtain the reference buildings, manual extraction was done for each of the 106 buildings based on the fact that most features within the study area (e.g. buildings, roads, trees, cars) were easily distinguishable with the naked eye. This was done using the visualization tool

CloudCompare (Compare, 2014). This software provides editing features such as DSM segmentation. For each reference object, a contour was defined by a successive series of clicks and then saved. The points inside this contour were then saved as a separate reference object.

As such, correctness (which evaluates the exactness of an approach) was defined as the ratio of the relevant points of a specific building to the total number of points of that building (see eqn 1). A point was considered relevant when the algorithm extracted it correctly, with respect to the corresponding reference building. Herein completeness, which measures the ability of the approach to extract the entire set of points relevant for a building (i.e. coverage), was defined as the ratio of the extracted relevant points to the total number of points in the buildings of the study area (see eqn 2). The F-measure as defined by Peukert (2012), which evaluates the overall quality (sometime called fitness), was calculated based on the correctness and completeness metrics, as shown in (eqn 3).

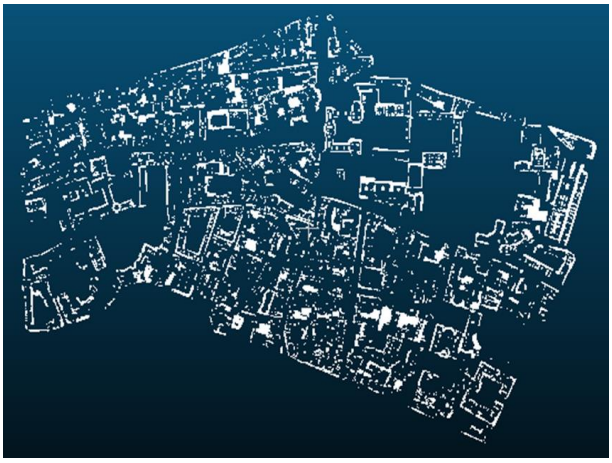
$$correctness = \frac{TP}{TP + FP} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (eqn\ 1)$$

$$completeness = \frac{TP}{TP + FN} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (eqn\ 2)$$

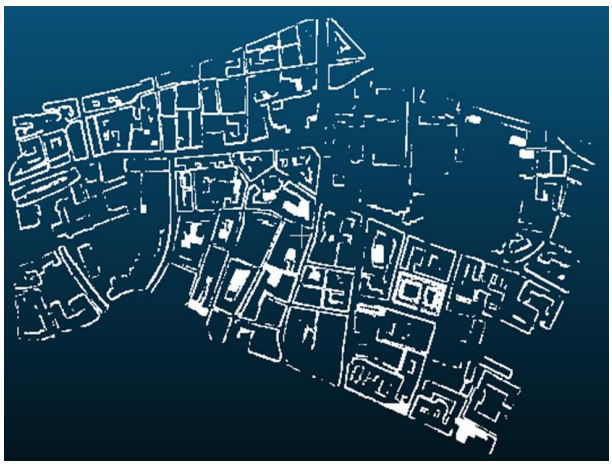
$$Fmeasure = \frac{2*(correctness*completeness)}{(correctness + completeness)} \quad \ldots\ (eqn\ 3)$$

where, True Positives (TP) counts the points correctly included into this object, False Positives (FP) counts the points incorrectly included into this object, False Negatives (FN) counts the points mistakenly excluded for this object (see figure 11).

TP = 114720570 points



FP = 8349057 points



FN = 10009794 points

Figure 11. The extraction result of the study DSM.

Looking at figure 11, one can conclude that most of the FP and FN points are found around the outlines of the objects, which means that the manual extraction of the reference buildings did not coincide exactly with the automatic way that was calculated by the Big Data approach. This is in part an outgrowth of the difficulty of manual extraction where errors may be introduced. Notably, more than 12 hours were needed to extract all the reference objects (approximately 7 minutes per object) versus less than 17 seconds per object using the algorithm.

In a previous study by Hinks (2011), which was performed on the same study DSM using a morphological contour of the scan line data with the same data, the final values were consistently not as good: correctness = 87%, completeness = 82%, and the overall quality = 84%. While, applying the Big Data approach, the evaluation measures across the 106 objects resulted in the following: correctness = 92%, completeness = 90%, and the overall quality = 91%. These measures were calculated based on lump sum evaluation of the points of all of the objects, however in order to show how these measures varied, the original DSM was divided into 20 smaller DSMs each consisting of approximately 4 to 8 adjacent objects. The process was then reapplied. The results are shown in Figure 12, in which DSM #3 obtained the best extraction quality (correctness = 94%, completeness = 94%, and overall quality = 94%), while the DSM #8 has obtained the worst result (correctness = 94%, completeness = 89%, and the overall quality = 92%). The problem of DSM #8 is related to completeness, because it consists of multiples and complex objects (see Figure 13).

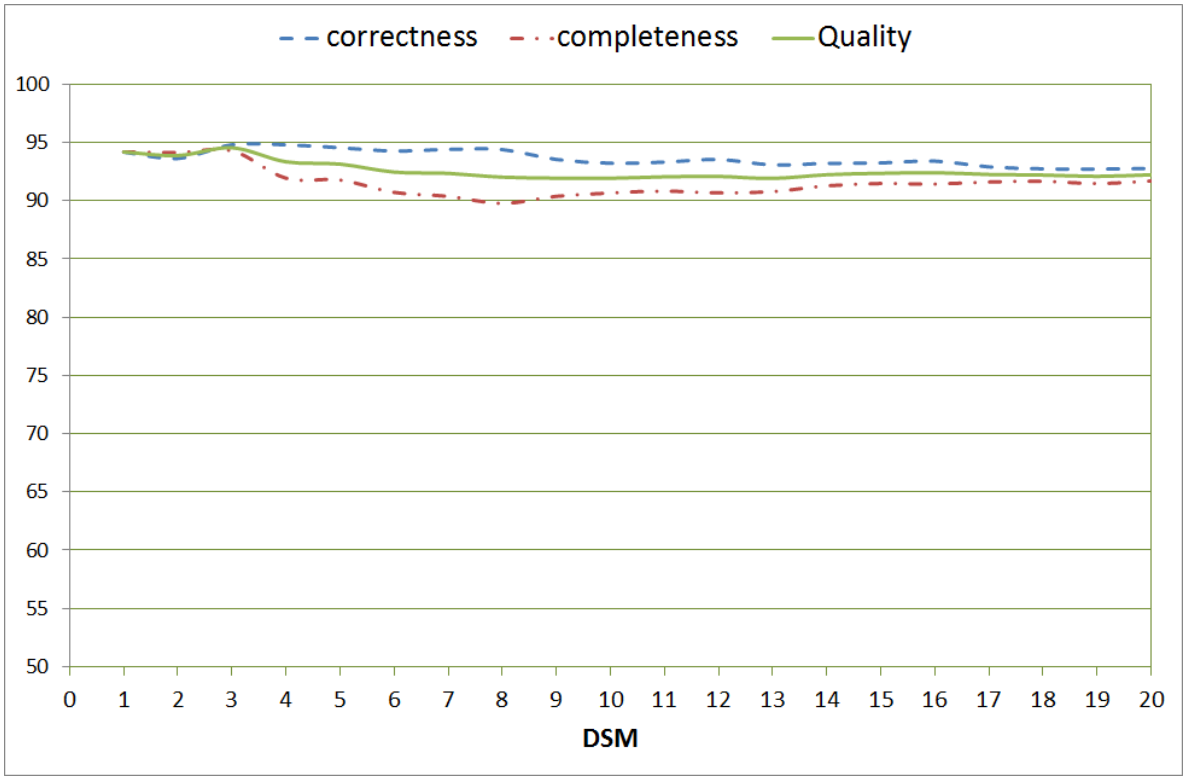
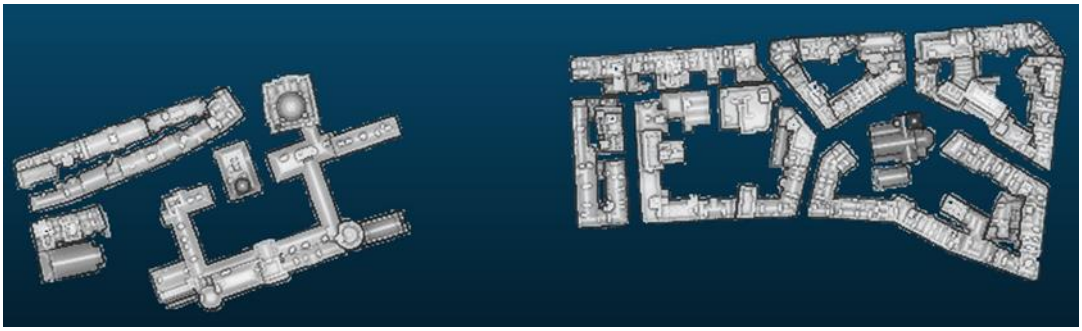


Figure 12. Detail quality study.



DSM #3 DSM #8

Figure 13. Sample DSMs within the study area

Although the experiment was performed in a single Hadoop installation, having a broader vision about the computational efficiency of the approach, where the efficiency evaluation refers to the execution time required to run the algorithm (Eldh 2006) is useful. In order to check the scalability of the proposed approach with a considerably larger number of points, an experiment was done by expanding the study area from the original 1 km^2 to a surrounding area of more than 4 km^2. This expanded dataset contained approximately 1 billion points. The execution time needed for the MapReduce phase is shown in Figure 14. For example, to segment approximately 226,319,306 points (1Km^2) the MapReduce step needed approximately 29.4 minutes. Arguably this time can be improved easily by adding more nodes to the cluster since having more nodes or computers in a cluster with data spread over the cluster can significantly improve the start to end processing time (Xu et al. 2015).

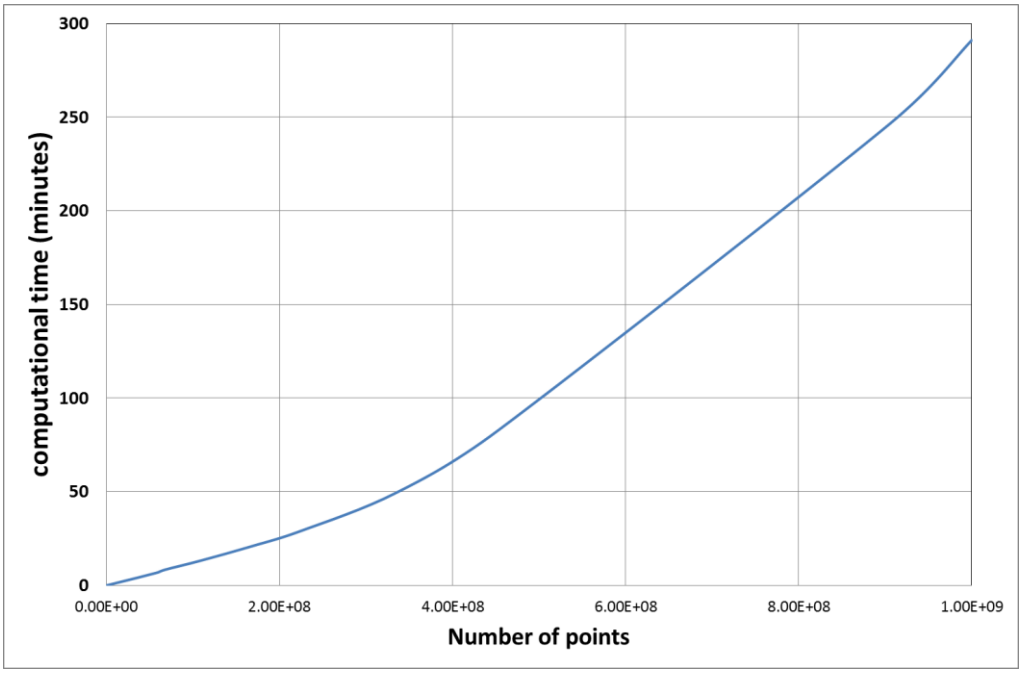


Figure 14. Execution time needed for the MapReduce phase

The time for the second phase of the algorithm is not significant when compared with the time needed for the MapReduce phase. For example, to generate the pDSM for the 1 km^2 area, the algorithm needs about 6.5 minutes. The exact object extraction time depends on the number of affiliated and adjacent points. In the worst case scenario, this time does not exceed 1 minute per extracted object with this highly dense (225 pt/m^2) LiDAR data set.

In summary, more than 92% of the relative points were correctly extracted from the study DSM for the 106 most significant objects. The missing 8% of the total points are likely to have been lost because of the complexity of the manual extraction. Furthermore, the approach was able to remove the heavy vegetation and other obstructions from the DSM and was able to segregate fully automatically the buildings from the roads, parking, and green areas. The approach was also very successful for buildings with complicated roof geometries and with multiple sections of varying heights, where some objects had more than 10 adjacent components of differing heights (see Figure 15).

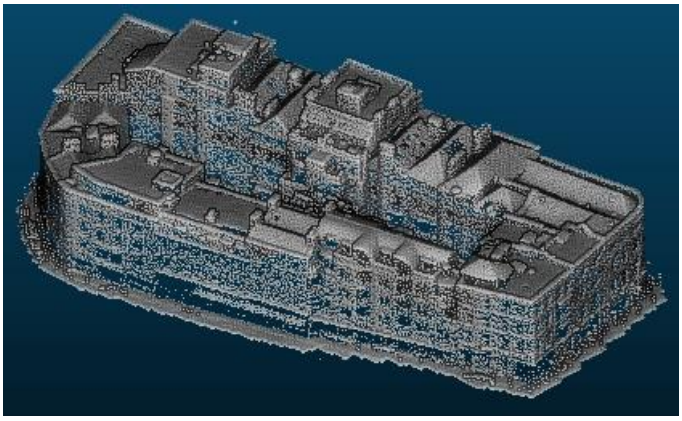


Figure 15. An object with more than 10 adjacent components of differing heights

CONCLUSIONS

Current solutions for building extraction from aerial LiDAR data are hard to scale with respect to the present capabilities and the forward trajectory of data densities levels and the increasing availability of full-wave form data. Consequently, a fully automatic approach has been presented using the significantly more scalable context of Big Data solutions. The approach is also beneficial in that it uses only vector data. The work was tested on a 1 km^2 study area, where more than 106 objects of architectural significance were automatically detected at an average extraction quality level of 92%.

The present study represents a first step towards a larger-scale project that aims to solve one of the greatest challenges in city-scale spatio-temporal analysis, namely the integration of multi-granularity datasets of varying data formats. The work presented herein offers a fundamentally new direction to solve these challenges – namely, integration strategies to preserve semantic data without compromising data accuracy during multi-granular mapping. Future work will determine the critical functionalities that may affect the querying and visualization of multi-granular datasets. Taking into account the previous integration strategies and the critical functionalities, a new data framework is envisioned – one that is

capable of synthesizing existing data sources and inserting them into a global schema within a Big Data strategy.

ACKNOWLEDGMENTS

This work was sponsored with funding from the European Union's grant ERC StG 2012-307836-RETURN. Data was acquired through funding from Science Foundation Ireland grant 05/PICA/I830.

REFERENCES

Abdullah, S., Awrangjeb, M., and Lu, G. Automatic segmentation of LIDAR point cloud data at difference height levels for 3D building extraction. *IEEE International conference on Multimedia and Expo workshops* (ICMEW) (Chengdu, 2014) (pages 1-6).

Aji, A., Wang, F., Vo, H., Lee, R., Liu, Q., Zhang, X., & Saltz, J. (2013). Hadoop GIS: a high performance spatial data warehousing system over mapreduce. *Proceedings of the VLDB Endowment*, *6*(11), 1009-1020.

Athitsos, V., Alon, J., Sclaroff, S., & Kollios, G. (2008). Boostmap: An embedding method for efficient nearest neighbor retrieval. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *30*(1), 89-104.

Awrangjeb, M., Zhang, C., & Fraser, C. S. (2013). Automatic extraction of building roofs using LIDAR data and multispectral imagery. *ISPRS Journal of Photogrammetry and Remote Sensing*, Volume 83, September 2013, Pages 1–18.

Batty, M., Axhausen, K. W., Giannotti, F., Pozdnoukhov, A., Bazzani, A., Wachowicz, M., ... & Portugali, Y. (2012). Smart cities of the future. *The European Physical Journal Special Topics*, 214(1), 481-518.

Cary, A., Sun, Z., Hristidis, V., & Rishe, N. (2009, January). Experiences on processing spatial data with mapreduce. In *Scientific and Statistical Database Management* (pp. 302-319). Springer Berlin Heidelberg.

Compare, C. (2014). 3D point cloud and mesh processing software Open Source Project. License: GNU GPL (General Public Licence), Version: 2.6, http://www.danielgm.net/cc

Dal Poz, A. P., & Galvanin, E. A. (2011, May). Building roof contour extraction from LiDAR data. ASPRS 2011 Annual Conference, Milwaukee, Wisconsin.

Ekhtari, N., Sahebi, M. R., Zoej, M. V., & Mohammadzadeh, A. (2008). Automatic building detection from LIDAR point cloud data. In *21st ISPRS Congress, Commission, WG IV/3, Beijing, China*.

Eldh S., Hansson H., Punnekkat S., Pettersson A., Sundmark D., (2006) A Framework for Comparing Efficiency, Effectiveness and Applicability of Software Testing Techniques. 159-170

Floridi, L. "The search for small patterns in big data." The Philosophers' Magazine 59 (2012): (page 17-18).

Gao, T., Ergan, S., Akinci, B., & Garrett, J. (2014). Evaluation of Different Features for Matching Point Clouds to Building Information Models. *Journal of Computing in Civil Engineering*.

Hinks, T., Carr, H., & Laefer, D. F. (2009). Flight optimization algorithms for aerial LiDAR capture for urban infrastructure model generation. *Journal of Computing in Civil Engineering*, *23*(6), 330-339.

Hinks, T. (2011), Geometric processing techniques for urban aerial laser scan data, PhD thesis, University College Dublin, Dublin, Ireland.

Jochem, A., Höfle, B., Wichmann, V., Rutzinger, M., Zipf, (2012) A. Area-wide roof plane segmentation in airborne LiDAR point clouds. Comput. Environ. Urban Syst. 2012, 36, 54-64.

Lee, D. H., Lee, K. M., & Lee, S. U. (2008). Fusion of lidar and imagery for reliable building extraction. *Photogrammetric Engineering & Remote Sensing*, *74*(2), 215-225.

Li, Y., Wu, H., An, R., Xu, H., He, Q., & Xu, J. (2013). An improved building boundary extraction algorithm based on fusion of optical imagery and LIDAR data. *Optik-International Journal for Light and Electron Optics*, *124*(22), 5357-5362.

Liu, Y., Chen, L., Xiong, W., Liu, L., & Yang, D. (2012, June). A mapreduce approach for processing large-scale remote sensing images. In the 20th International Conference on Geoinformatics (pp. 1-7). IEEE.

Maurya, R., Gupta, P. R., Shukla, A. S., & Sharma, M. K. (2012, March). Building extraction from very high resolution multispectral images using NDVI based segmentation and morphological operators. In *Advances in Engineering, Science and Management (ICAESM), 2012 International Conference on* (pp. 577-581). IEEE.

Mayunga, S. D., Zhang, Y., & Coleman, D. J. (2005, August). Semi-automatic building extraction utilizing Quickbird imagery. In Proc. ISPRS Workshop CMRT (Vol. 13, pp. l-136).

Morgan, M., & Tempfli, K. (2000). Automatic building extraction from airborne laser scanning data. *International Archives of Photogrammetry and Remote Sensing*, *33*(B3/2; PART 3), 616-623.

NESSI White Paper, December 2012, Big Data A New World of Opportunities. http://www.nessi-europe.com/Files/Private/NESSI_WhitePaper_BigData.pdf

Özdemir, V., Badr, K. F., Dove, E. S., Endrenyi, L., Geraci, C. J., Hotez, P. J., ... & Kickbusch, I. (2013). Crowd-funded micro-grants for genomics and “big data”: An actionable idea connecting small (artisan) science, infrastructure science, and citizen philanthropy. *Omics: a Journal of Integrative Biology*, *17*(4), 161-172.

Peukert, E. (2012). "System and method of optimizing performance of schema matching." U.S. Patent No. 8,219,596. 10 Jul. 2012.

Pradhan, A. R., Laefer, D. F., & Rasdorf, W. J. (2007). Infrastructure management information system framework requirements for disasters. *Journal of computing in civil engineering*, *21*(2), 90-101.

Rottensteiner, F., & Briese, C. (2002). A new method for building extraction in urban areas from high-resolution LIDAR data. *International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences*, *34*(3/A), 295-301.

Rottensteiner, F., Trinder, J., Clode, S., & Kubik, K. (2004). Fusing airborne laser scanner data and aerial imagery for the automatic extraction of buildings in densely built-up areas. *International Archives of Photogrammetry and Remote Sensing*, *35*(B3), 512-517.

San, D. K., & Turker, M. (2010). Building extraction from high resolution satellite images using Hough transform. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Science*, *38*(Part 8).

Song, W., & Haithcoat, T. L. (2005). Development of comprehensive accuracy assessment indexes for building footprint extraction. *Geoscience and Remote Sensing, IEEE Transactions on*, *43*(2), 402-404.

Sugihara, K., & Kikata, J. (2012). Automatic generation of 3D building models from complicated building polygons. *Journal of Computing in Civil Engineering*, *27*(5), 476-488.

Theng, L. B. (2006). Automatic building extraction from satellite imagery. Engineering Letters, 13(3), 255-259.

Truong-Hong, L. (2011). Automatic Generation of Solid Models of Building Facades from LiDAR Data for Computational Modelling. , Ph. D. Thesis, University College Dublin

Truong-Hong, L., Laefer, D. F., Hinks, T., & Carr, H. (2013). Combining an angle criterion with voxelization and the flying voxel method in reconstructing building models from LiDAR data. *Computer-Aided Civil and Infrastructure Engineering*, *28*(2), 112-129.

Walsh, S. B., Borello, D. J., Guldur, B., & Hajjar, J. F. (2013). Data processing of point clouds for object detection for structural engineering applications. *Computer-Aided Civil and Infrastructure Engineering*, *28*(7), 495-508.

Wang, J., & Shan, J. (2009, March). Segmentation of LiDAR point clouds for building extraction. In *American Society for Photogramm. Remote Sens. Annual Conference, Baltimore, MD* (pp. 9-13).

Wang, K., Han, J., Tu, B., Dai, J., Zhou, W., & Song, X. (2010, December). Accelerating spatial data processing with mapreduce. In *Parallel and Distributed Systems (ICPADS), 2010 IEEE 16th International Conference on* (pp. 229-236). IEEE.

White T. (2012). Hadoop: The Definitive Guide, 3rd Edition, O'Reilly Media / Yahoo Press, 2012, Pages: 688.

Wu, X., Carceroni, R., Fang, H., Zelinka, S., & Kirmse, A. (2007, November). Automatic alignment of large-scale aerial rasters to road-maps. In Proceedings of the 15th annual ACM international symposium on Advances in geographic information systems (p. 17). ACM.

Xu, G., Yu, W., Chen, Z., Zhang, H., Moulema, P., Fu, X., & Lu, C. (2015). A cloud computing based system for cyber security management. *International Journal of Parallel, Emergent and Distributed Systems*, *30*(1), 29-45.

Zhang, K., Yan, J., & Chen, S. C. (2006). Automatic construction of building footprints from airborne LIDAR data. *IEEE Transactions on Geoscience and Remote Sensing, , 44*(9), pages 2523-2533.

Zhang, S., Han, J., Liu, Z., Wang, K., & Feng, S. (2009, August). Spatial queries evaluation with mapreduce. In *Grid and Cooperative Computing, 2009. GCC'09. Eighth International Conference on* (pp. 287-292). IEEE.