ISSN 2347 - 3983
Volume 8. No. 9, September 2020
International Journal of Emerging Trends in Engineering Research
Available Online at http://www.warse.org/IJETER/static/pdf/file/ijeter123892020.pdf
https://doi.org/10.30534/ijeter/2020/123892020

Efficiently Displaying Geometric Primitives Extracted from LiDAR Data Points

Nakhoon Baek
School of Computer Science and Engineering, Kyungpook National University, Daegu 41566,
Republic of Korea, oceancru@gmail.com

ABSTRACT

The LiDAR (Laser Imaging, Detection and Ranging) system is one of the most useful data capturing tools, with high-accuracy sensors. Laser lights illuminate targets, and the system measures the distance to the target objects. The LiDAR systems can be used to obtain various kinds of geological and geometric data, including terrain surfaces, outside buildings, and others. From these LiDAR-scanned data points, we need a variety of operations, including LiDAR file format supports, point cloud manipulation features, geometric primitive extractions, reconstruction of 3D objects from the point clouds, and the efficient rendering of the point clouds and the extracted geometric primitives. We have analyzed each of those technical issues and presented practical engineering solutions. We finally integrated all the features into the system to efficiently display the geometric primitives, which are numerically extracted from the original LiDAR data points.

Key words : laser scanning, point cloud, efficient rendering, geometric primitives, integrated system

1. INTRODUCTION [1]

Recently, we have a wide variety of geometric and non-geometric data from various sampling devices [1,2]. Since the size of the sampled data are matter, we need a simple and intuitive way of handling those sampled data. In this paper, we present an efficient system for the large-scale laser-scanned data display, with geometric primitive extraction and some additional features.

The laser-scanning system is called as LiDAR, which comes from *Laser Imaging, Detection and Ranging* [1]. It is also known as *laser scanning* or *3D scanning* [2]. Theoretically, the laser lights will illuminate the target points, and the reflection enables us to measure the distances to the target points. At this time, the LiDAR systems are widely used to obtain various types of data, including geological data, terrain surfaces, interior obstacles, and much more. Variety of geometric and geological applications use the LDAR systems and the laser scanning technology to get the raw data points, their derived geological terrains, outside of buildings, interior scanning, their related geometric models, and others [1,2,3,4,5].

The LiDAR systems become one of the most widely used 3D point cloud acquisition technologies [6]. Typical LiDAR systems consist of some technical components: laser generator, optics, detector, sensors, etc. Those systems can have various settings, including airborne systems, terrestrial systems, and indoor systems [7]. Mostly, the LiDAR system produces very large data sets, with many 3D sampling points. A conceptual explanation of the large-scale landscape LiDAR system is shown in Figure 1. As another kind of LiDAR systems, the terrestrial LiDAR system [8] is also shown in Figure 2.

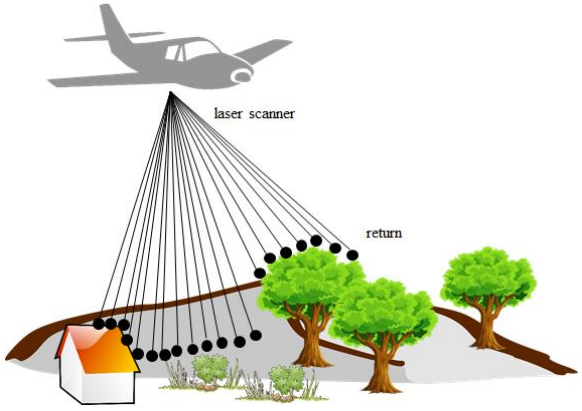


Figure 1: An airborne LiDAR system.

For handling LiDAR data, we meet difficulties of lacking well-known standard formats and/or systems. Currently, the LiDAR system works with various data file formats and data acquisition methods. The large number of data points from the LiDAR system makes another kind of problems. Typically, the large-scale laser-scanned data points count to millions or billions of 3D data points, and they call it as the *point clouds* [9].

[1] This work has supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (Grand No.NRF-2019R1I1A3A01061310).

Figure 2: A terrestrial LiDAR system.

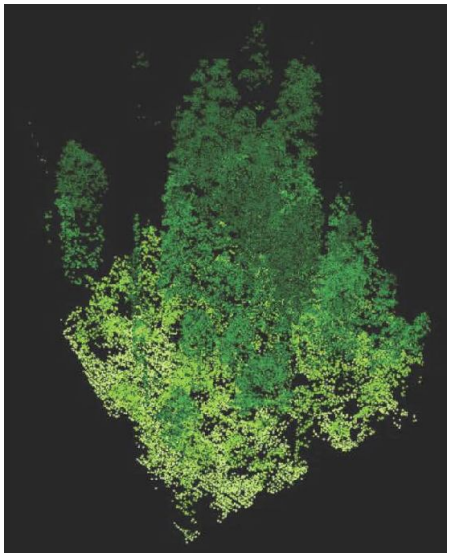


Figure 3: A point cloud from LiDAR scanner [14].

A typical point cloud has millions or even billions of 3D points, which should be efficiently handled even with the ordinary PC's. Most of the high-level application programs, including vision processing tools and *Geographic Information System* (GIS) applications, usually extract some geometric information from the point cloud, in their preprocessing steps [10].

In this paper, we present our systematic ways of handling these point clouds from LiDAR sensors. We integrated the LiDAR data file formats, the point cloud handling and editing operations, geometric primitive extractions, and efficient visualization methods into a unified framework. The following sections describe the new features and their implementations.

2. UNDERLYING FILE FORMATS

To handle the point clouds from the LiDAR scanner, we start from the error estimation of the point cloud registration. As one of the simplest cases, we examine the point-to-point correspondence cases. A 3D *point cloud* is a set of points

$$\{ \mathbf{p}_i \mid i = 1, \ldots, n \},$$

which is embedded in the 3D space [web3d paper]. We refer it as the reference surface, $\mathbf{p}_i = (x_i, y_i, z_i)$ with indices $i = 1, \ldots, n$. Another test surface is denoted as another set of points of $\mathbf{q}_i = (x_i, y_i, z_i)$, with the same indices $i = 1, \ldots, n$ [11].

Denoting the transformation matrix and the translation vector as R and t, the sum of squared errors can be calculated as:

$$E(\mathbf{R},\mathbf{t}) = \sum \left\| \mathbf{p}_i - \mathbf{R}\mathbf{q}_i - \mathbf{t} \right\|^2,$$

where $\mathbf{p}_i$ and $\mathbf{q}_i$ are the points on the reference surface and the test surface, respectively. We can achieve the suitable registration of the test surface through minimizing the estimated error $E(\mathbf{R},\mathbf{t})$ with the best values of **R** and **t** [11].

In the more general cases, there are no direct correspondences between points of the surfaces are given. Additionally, the number of points to be registered may be different. In this case, we cannot calculate the transformation matrix **R** and the translation vector **t** directly. Instead, they find suitable solutions numerically, with iterative methods. We call this approach as *iterative closest point* (ICP) methods [6,12].

For laser-scanned data points, we can use iterative closest point methods to extract geometric features. A scanned point cloud can be described as a set of points $\{(x_c, y_c, \theta_c)\}$, with a set of range measurements r_{ci} at the measuring bearings φ_{ci}. We can also specify a geometric feature as another set of range measurements r_{ri} at the bearings φ_{ri} [13]. We can calculate the projected scan readings, (r'_{ci}, φ'_{ci}) as follows:

$$r'_{ci} = \sqrt{(r_{ci}\cos(\theta_c + \varphi_{ci}) + x_c)^2 + (r_{ci}\sin(\theta_c + \varphi_{ci}) + y_c)^2}$$

$$\varphi'_{ci} = \arctan2(r_{ci}\sin(\theta_c + \varphi_{ci}) + y_c, r_{ci}\cos(\theta_c + \varphi_{ci}) + x_c),$$

where *arctan2* function is the four-quadrant version of the arc tangent function [13].

Some of the technical problems with laser-scanned point clouds are directly related to the extremely large size of the point cloud. Figure 3 is a typical example of the LiDAR-scanned data, with more than millions of sampled points. The first problem with the LiDAR-scanned data may be file formats. Unfortunately, there is still no dominant standard specification for the LiDAR-scanned data points. There are several candidate file formats including LAS [15,16,17], E57 [18], PCD [19], XYZ [20], and others. Reversely, a LiDAR data handling system has better to support as many file formats as possible.

The LAS file format (short for LASer) is designed to store three-dimensional point data, developed by the *American*

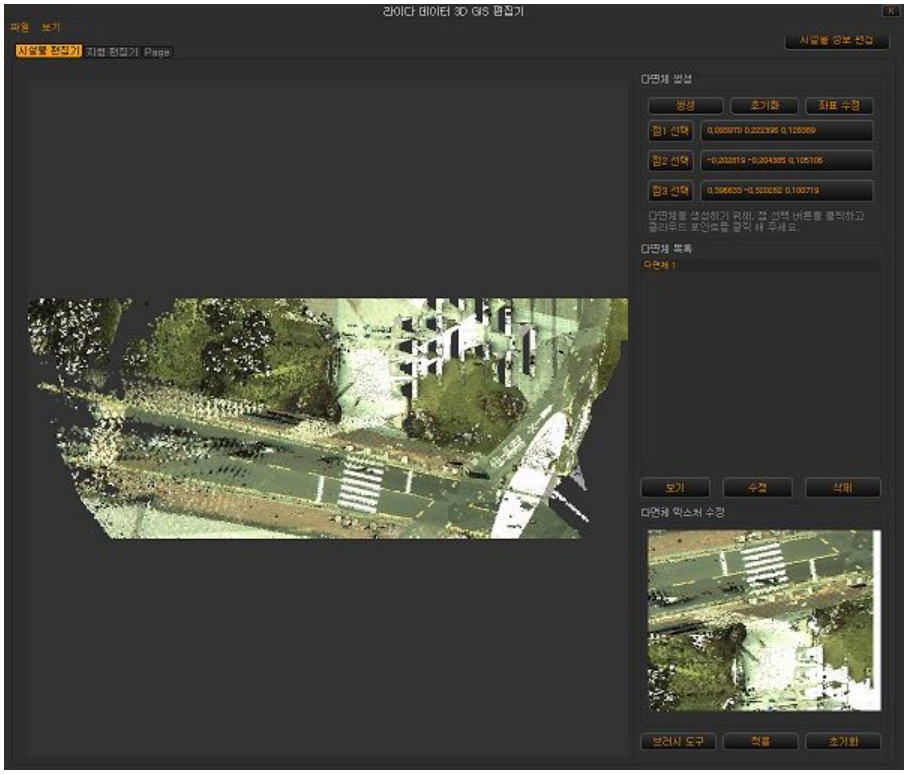


Figure 4: Our graphical user interface for point-cloud handling operations.

Society for Photogrammetry and Remote Sensing (ASPRS) [15,16]. A LAS file consists of sequences of binary formats, each of which stores some data points from the laser scanner. It can additionally store the intermediate results and partial results by various applications. This file format has recently begun to receive attention in the terrestrial laser scanning community [17].

The E57 file format is specified by the *American Society for Testing and Materials* (ASTM). It is also known as ASTM E57 3D file format, or the ASTM E2807 standard [21]. This file format is compact enough to store point clouds, derived images, and other metadata from various sensors and 3D scanning systems. From a global point of view, an E57 file has a hierarchical tree structure, which is basically based on the XML data format. In its low-level view, the actual data points in an E57 file are represented in compressed binary forms. The format can support XML-based flexibility and also efficiency with compressed streams of binary data [18, 21].

In our system, we support several file formats for LiDAR-scanned data points. The LAS and the E57 file formats are the core of our file format support. In the following sections, we will show how to handle point clouds stored in these file formats.

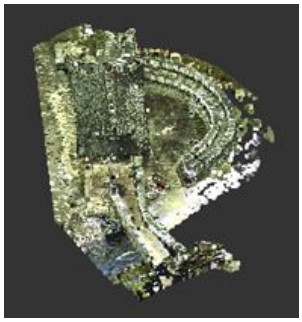
3. IMPLEMENTATION RESULTS

3.1 Fine-handling of Point Clouds

In most cases, it is hard to edit millions of points on the single screen. It is even hard to manage those point clouds. The first feature of our system is the basic handling operations of point clouds. Our system provides graphical interfaces to the point cloud handling, as shown in Figure 4.

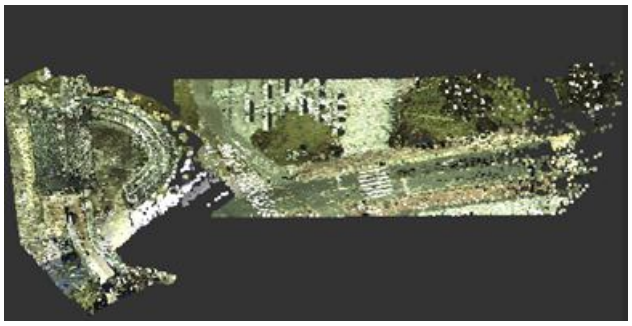
As the basic point cloud handling operations, we provide many editing operations, including the following:

- **preview images** – our system generates fast preview images on-the-fly, for fast recognition and easy identification of the point cloud.
- **union of point cloud sets** – adjacent and/or logically-related point clouds can be merged into a single union point cloud.
- **point cloud compression** – to reduce the number of points from the original LiDAR-scanned point clouds, we apply point cloud compression techniques.
- **height map generation** – a point cloud can be projected onto a plane, to generate a height map
- **digital terrain model generation** – we can convert the LiDAR point cloud into a digital terrain model, for more advanced uses.



(a) a point cloud

(b) another point cloud



(c) merged result

Figure 5: An example of point-cloud merge operation

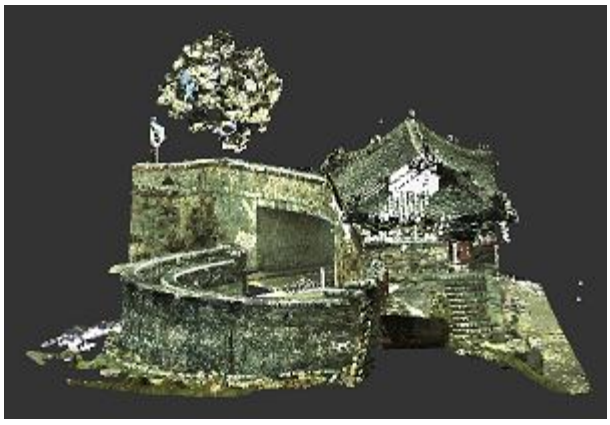
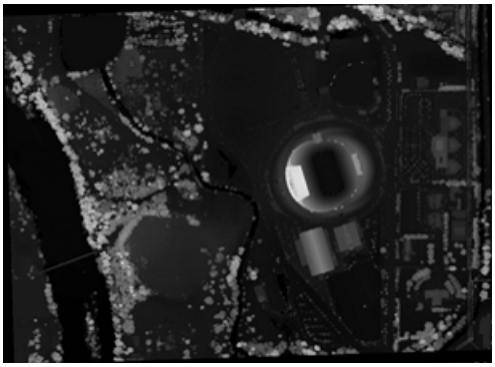


Figure 6: The merged result of 15 point clouds.

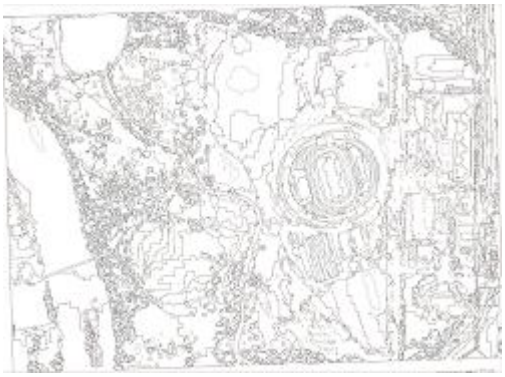
One of the most important features of the point cloud editing is the merging operation with several point clouds. Typically, we scan a single object several times from even different scanning locations, and then, we have many sets of point clouds for a single object. In this case, we can merge the logically-related point clouds into a single one. Figure 5 shows a single step of merging two point-clouds into a large one. Figure 6 shows the result of combing 15 point-clouds into a single point cloud.

In some cases, the 3D point cloud can be regarded as a *height map*. The 3D data points are projected onto a 2D plane, and we can assign the height information to each of 2D points. We construct the height map in a straight-forward manner. The height map is represented as a 2D array internally. All the 3D source points are projected onto the 2D target plane, and we measure the distance from the 3D point to the target plane as the height of that point, as shown in Figure 7.(a).

We can also combine various image-filtering operations to the height map. As an example, we can use the Gaussian filters to remove small objects from the height map, if needed. Median filters can be applied to removes device measuring noises. Figure 7.(b) represents the generated contours from the filtered height maps.



(a) a height map



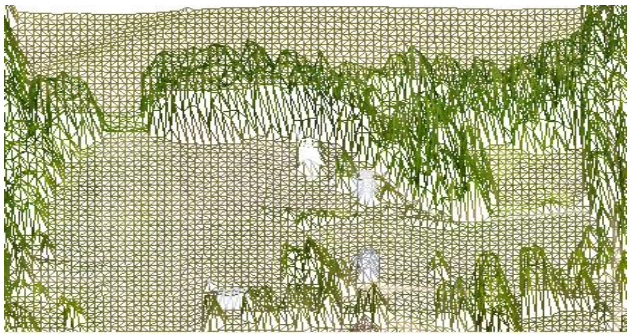
(b) its contour extraction

Figure 7: An example height map and its contour extraction.

In addition to the image handling features, our system supports Digital Terrain Model (DTM) editing features, especially for the airborne LiDAR-scanned point clouds. Typically, a DTM data means the pure ground surface, removing as many artificial objects including buildings and

even plants, from the LiDAR data. In this case, the object removal operations are essential in addition to the normal handlings of the LiDAR point cloud [22]. We use the adaptive TIN (triangular irregular networks) model, as described in [23].

Internally, our system uses the 1 byte signed integer data type to store the depth values, if possible. Thus, the height maps, and some of its derived DTMs may be represented with their approximated integer values, even though they are originally stored and calculated as floating-point numbers. Figures 8 shows an example DTM and its modified version, with our system's DTM editing features.



(a) a digital terrain model

(b) its modified model

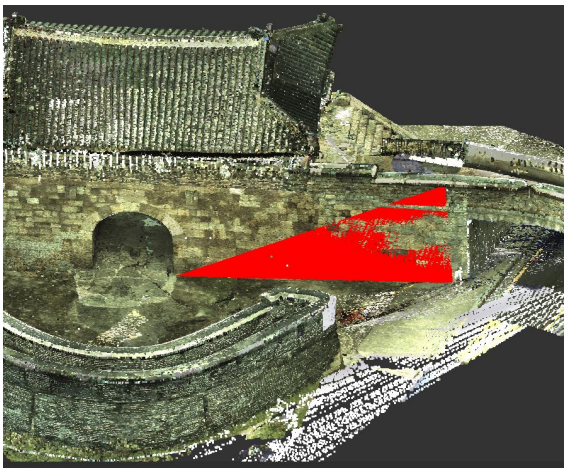
Figure 8: An example Digital Terrain Model (DTM) and its modification.

3.2. Geometric Features Extraction

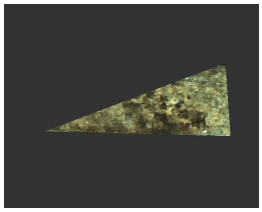
For more efficient handling of LiDAR-scanned data points, we also provide geometric feature extractions. The typical geometric feature extractions from the LiDAR-scanned data points can be mathematically interpreted as the well-known 3D data registration problem [6]. In the early stages of our implementation, we tried to implement our in-house versions of the 3D registration algorithms. However, we rapidly found that we already have a set of well-implemented public versions of this 3D data registration algorithms.

As an example, the *Computational Geometry Algorithms Library* (CGAL) [24] provides those registration algorithms through its own API functions. Currently, many applications in the various areas already use CGAL to solve their geometric operations. Using our in-house implementations and also CGAL API functions, we can extract a set of geometric primitives, from the point clouds. Those geometric primitives are again combined to construct 3D geometric models, which, at least theoretically, can be used other 3D modeling programs.

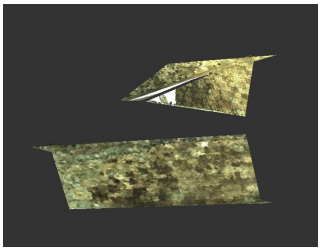
Actually, we have some research results on 3D geometric model reconstruction from point clouds [25]. However, in our case, it is not easy to use those methods directly, since our LiDAR scanned data has its own properties. Those are representing real-world buildings, with relatively large-scale data points. Another problem is that the previous research results assume a single closed surface, rather than a set of overlapped point clouds, such as our LiDAR point clouds. In practice, typical LiDAR scanners are hard to get the bottom side during its building interior scanning.



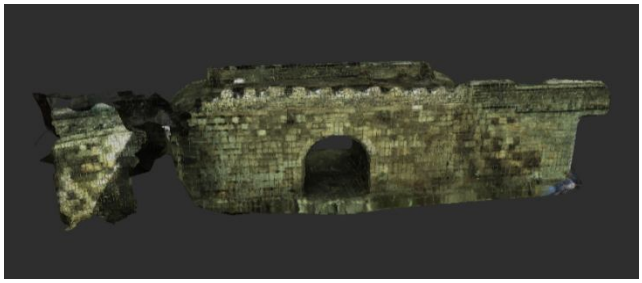
(a) a geometric primitive from the point cloud



(b) extracted results



(c) progressive results



(d) the final result

Figure 9: Conversion of point-clouds to 3D geometric models.

Our proposed scheme starts with manually selecting a suitable geometric primitive, for the target point clouds. The selectable primitives can be ranged from simple triangles and squares to 3D shapes of cylinders, cubes, spheres, and others. After selecting a geometric primitive, our system executes the geometric primitive extraction and fitting with the registration algorithms. Then, our system also extract the optimal texture images for the geometric primitive, from the grabbed framebuffer. Figure 9 shows an example sequence of this semi-automatic geometric primitive extraction, from the large-scale point clouds [9].

3.3. Customized Display Methods

One of the remaining problems would be the efficient display of the original LiDAR-sampled point clouds and also the extracted geometry primitives. For general cases, we have many 3D graphics rendering systems, including OpenGL [26], DirectX [27], X window system [28], Display PostScript [29], Cairo [30], OpenInventor [31], Qt [32], and others. However, those graphics libraries are tuned for general 3D rendering, rather than the LiDAR-scanned point clouds [33].

Since the graphics libraries developed in a stepwise manner, the modern graphics architecture is actually a kind of library stacks. In the case of modern OpenGL library implementation, it works over the operating system kernels, X window system, and GLX extensions. Even the two-dimensional graphics output support with the X window system simultaneously work on the same system. Our system presents a new way of accelerated 3D rendering, directly based on the Linux kernel support. In this case, we have no need to integrate it with any graphics window systems and any acceleration extensions. We represent the concepts and designs for our graphics pipeline with fixed function graphics features.

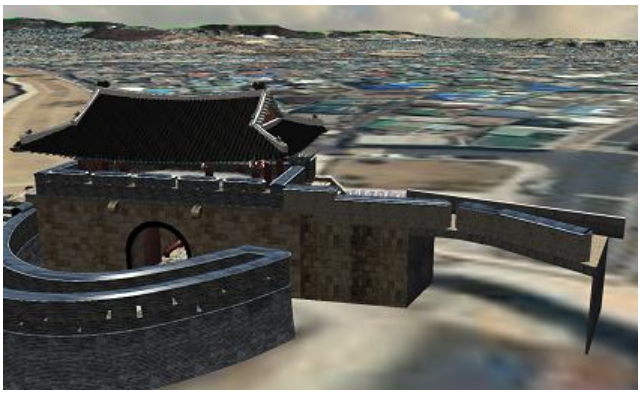
In our design and implementation of the customized display methods, we adopted to use the *Graphics Processing Unit* (GPU) more actively. In the modern computer architecture, the GPUs are most essential and core unit for the 3D graphics support. In the case of Linux operating systems, its kernel architecture has the *Direct Rendering Manage* (DRM) module [34,35], which can access the GPUs and the framebuffer directly. Thus, with the DRM modules, we can directly control and tune the GPU actions for more accelerated and customized display of our target point clouds and extracted geometric primitives. More precisely, the DRM module provides kernel function calls to access GPUs directly, especially in the case of Linux operating systems [36]. For graphics output, the upper-level libraries, such as OpenGL and DirectX, use the DRM module to control GPUs and also to transfer the graphics data to the GPUs.

Our system has the kernel of 3D transformations and rendering features. To derive GPU core instructions, we use the fixed GPU instruction sequences extracted from the existing implementations of OpenGL. Figure 10 shows the results of our DRM-based direct 3D graphics rendering system. As shown here, point-clouds from the LiDAR sampled data and its reconstructed 3D models successfully works with our system.

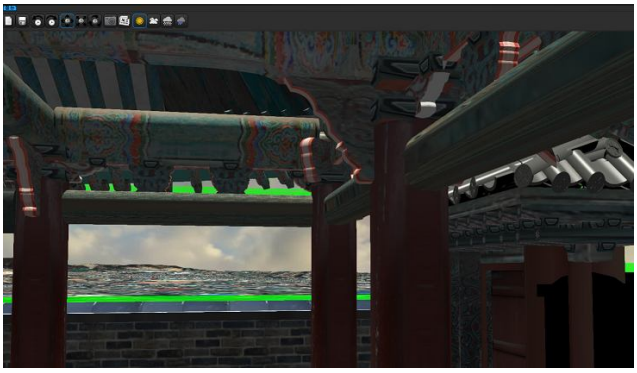
Table 1: Experimental results from our system.

sample case	no of raw data points from LiDAR device	no of data points after compression	compressed ratio	speed-ups (times)
case (a)	318,046,976	2,754,480	0.866 %	115.47
case (b)	248,031,609	4,219,145	1.701 %	58.79
case (c)	385,000,039	5,407,008	1.404 %	71.20
case (d)	430,944,637	3,117,913	0.724 %	138.22

Finally, our system combines all the acceleration methods described in the previous sections. Table 1 shows the final speed-up factors of our system. We started from raw-level point cloud with LiDAR scanned systems. Various geometric algorithms process them, and extract the primitives. Our system shows remarkable speed-ups of around 100 times faster than the generic rendering of the original point clouds.



(a) our DRM-based graphics output



(b)another output

Figure 10: Examples of our DRM-based 3D graphics outputs.

4. DISCUSSION AND CONCLUSIONS

Recently, the LiDAR-scanning technology is widely applied in various applications. In its technical uses, we should use more efficient and much customized way of handling the point clouds and the derived geometric data [37,38,39]. In this work, we focused on the following technical issues:

- LiDAR data file formats
- Point cloud manipulation features
- Geometric primitive extraction
- Reconstruction of 3D objects from the point clouds
- Efficient rendering of the point clouds and geometric primitives

We have analyzed each of the technical issues and presented practical engineering solutions. We finally integrated all the features into the system for LiDAR-scanned data handling. Our implementation shows remarkable speed-ups, for the handling of LiDAR-scanned point clouds, geometric primitive extraction, and customized display of the reconstructed 3D graphics models.

Our current implementation can be improved with some parallel processing features. For example, our current extraction algorithms will be more accelerated with their re-implementations in parallel programming systems including OpenMP, OpenCL, and/or CUDA. Our system will be available to the commercial markets, in the near future.

REFERENCES

1. G. E. Marshall and G. E. Stutz. ***Handbook of Optical and Laser Scanning***, 2nd Ed., CRC Press, 2011.
2. G. Vosselman and H.-G. Mass, ***Airborne and Terrestrial Laser Scanning***, CRC Press, 2010.
3. W. R. Benner Jr., ***Laser Scanners: Technologies and Applications***, Pangolin, 2016.
4. M. Maltamo and E. Næsset, ***Forestry Applications of Airborne Laser Scanning: Concepts and Case Studies***, Springer, 2016.
5. J. Shan and C. K. Toth, ***Topographic Laser Ranging and Scanning: Principles and Processing***, 2nd Ed., CRC press, 2018.
6. C. Cao, M. Preda, and T. Zaharia, **3D Point Cloud Compression: A Survey**. *The 24th International Conference on 3DWeb Technology*. Association for Computing Machinery, 2019, Web3D '19, pp. 1–9.
7. I. Dowman, **Integration of LIDAR and IFSAR for Mapping**, *International Archives of Photogrammetry and Remote Sensing*, 35, 90-100, 2004.
8. P. Hobbs, C. Pennington, S. Pearson, L. Jones, C. Foster, J. Lee and A. Gibson, ***Slope dynamics project report, Norfolk Coast (2000 - 2006)***, 2008.
9. N. Baek, W. Shin and K. J. Kim, **Geometric Primitive Extraction from LiDAR-scanned Point Clouds**, *Cluster Computing*, 20(1):741-748, 2017.
10. W. Shin and N. Baek, **Editing LiDAR-Based Terrains with Height and Texture Maps**, *ICISS 2016*, 2016.
11. G. Krell, et al., **Assessment of iterative closest point registration accuracy for different phantom surfaces captured by an optical 3D sensor in radiotherapy**, *Computational and Mathematical Methods in Medicine*, 2017(2938504), 13 pages, 2017.
12. Z. Zhang, **Iterative Closest Point (ICP)**, In: K. Ikeuchi, *Computer Vision*, Springer, 2014.
13. G. A. Kumar, et al., **A LiDAR and IMU Integrated Indoor Navigation System for UAVs and its application in real-time pipeline classification**, *Sensors*, 17:1268, 2017.
14. A. Samberg, **An implementation of the ASPRS LAS standard**, *IAPRS*, vol. XXXVI, 2007.
15. The American Society for Photogrammetry & Remote Sensing, ***LAS Specification***, Version 1.4, ASPRS, 2011.
16. The LAS ASPRS LiDAR data translation toolset, http://www.liblas.org/. 2016.
17. D. Huber, **The ASTM E57 file format for 3D imaging data exchange**, *Proceedings of the SPIE*, Vol. 7864A, 2011.
18. The Point Cloud Data file format, http://pointclouds.org/documentation/tutorials/pcd_file_format.php (retrieved in Aug 2020).
19. XYZ file format, https://en.wikipedia.org/wiki/XYZ_file_format (retrieved in Aug 2020).
20. https://www.unavco.org/software/visualization/idv/IDV_datasource_point_cloud.html (retrieved in Aug 2020).
21. ASTM E2807, ***Standard Specification for 3D Imaging Data Exchange***, Version 1.0, 2011.
22. Y. Feng, et al., **Urban DEM generation from airborne Lidar data**, *Urban Remote Sensing Event*, ISSN 2334-0932, 2009.
23. T. Strutz, ***Data Fitting and Uncertainty***, 2nd Ed., Springer, 2016.

24. CGAL homepage, http://www.cgal.org/ (retrieved in Aug 2020).
25. B. Matthew, et al., **A Benchmark for Surface Reconstruction**, *ACM Transactions on Graphics*, Vol.32, 2013.
26. M. Segal and K. Akeley, ***The OpenGL Graphics System: A Specification***, Version 4.5 (Core Profile), Khronos Group, 2016.
27. F. Luna, ***Introduction to 3D Game Programming with DirectX 12***, Mercury Learning & Information, 2016.
28. D. Young, ***The X Window System: Programming and Applications with Xt, OSF/Motif***, 2nd Edition, Prentice Hall, 1994.
29. Adobe Systems, ***Programming the Display Postscript System With X (APL)***, Addison-Wesley, 1993.
30. http://www.cairographics.org/ (retrieved in Aug 2020).
31. J. Wernecke, ***The Inventor Mentor: Programming Object-Oriented 3D Graphics with Open Inventor***, Addison-Wesley, 1994.
32. G. Lazar, ***Mastering Qt 5***, Packt Publishing, 2017.
33. B. Im, N. Baek and J. Lee, **An Efficient Implementation of LiDAR Data and its Geometric Representation Extraction**, *ICITCS 2016*, 2016.
34. K. Packard and E. Anholt, ***The Graphics Execution Manager: Part of the Direct Rendering Manager***, on-line article, 2008.
35. R. E. Faith, **The Direct Rendering Manager: Kernel Support for the Direct Rendering Infrastructure**, http://dri.sourceforge.net/doc/drm_low_level.html, 2016.
36. J. Fonseca, ***Direct Rendering Infrastructure: Architecture***, 2005.
37. A. S. Alon, E. D. Festijo and C. D. Casuat, **Tree Extraction of Airborne LiDAR Data Based on Coordinates of Deep Learning Object Detection from Orthophoto over Complex Mangrove Forest**, *IJETER*, 8(5):2107-2111, 2020.
38. M. Kumar and R. H. Sree, **Home Computerization Monitoring System with Google Supporter**, *IJETER*, 8(6):2240-2244, 2020.
39. N. Baek, **A Simplified Implementation of the Fixed-Function Graphics Pipeline: DRM Approach**, *IJATCSE*, 9(2):1551-1555, 2020.