Web-Based Real-Time LADAR Data Visualization with Multi-user Collaboration Support

Ciril Bohak[1(✉)], Byeong Hak Kim[2], and Min Young Kim[2]

[1] Faculty of Computer and Information Science, University of Ljubljana, Večna pot 113, 1000 Ljubljana, Slovenia
ciril.bohak@fri.uni-lj.si
[2] School of Electronics Engineering, Kyungpook National University, 1370 Sankyuk-dong, Buk-gu, Daegu, Korea
durumy98@hanmail.net, minykim@knu.ac.kr

Abstract. In this paper we present a web-based visualization system developed for visualizing real-time point cloud data obtained from LADAR (or other) sensors. The system allows direct visualization of captured data, visualization of data from database or visualization of preprocessed data (e.g. labeled or classified data). The system allows the concurrent visualization from same or different data-sources on multiple clients in the web browser. Due to the use of modern web technologies the client can also be used on mobile devices. The system is developed using modern client- and server-side web technologies. The system allows connection with an existing LADAR sensor grabber applications through use of UDP sockets. Both server- and client-side parts of the system are modular and allow the integration of newly developed modules and designing a specific work-flow scenarios for target end-user groups. The system allows the interactive visualization of datasets with millions of points as well as streaming visualization with high throughput speeds.

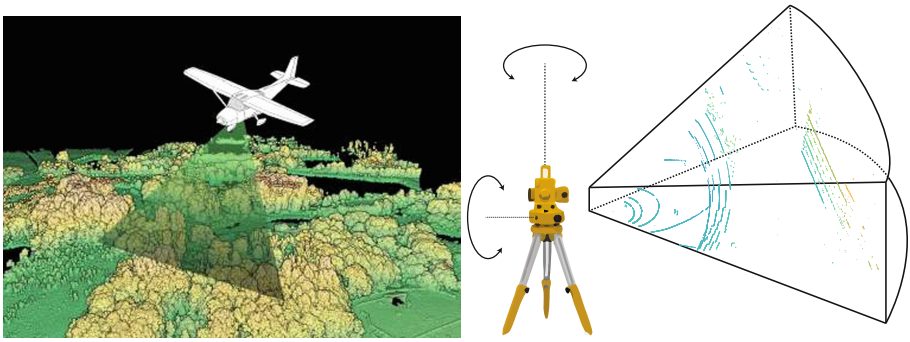
Keywords: LiDAR · LADAR · Point cloud data · WebGL
Data visualization

1 Introduction

There are many different ways of storing and presenting the data describing the world around us. One of more popular ways in the recent years is use of point cloud representation — a set of points in space which store different parameters and information regarding the specific spatial location. Such representation is mostly used for representing the shapes of the objects and surrounding world, but it can also be used for storing other information (e.g. meteorological data, physical measurements, simulation data etc.). In our case the point cloud data is representing the surrounding world and objects in it.

L. T. De Paolis and P. Bourdot (Eds.): AVR 2018, LNCS 10850, pp. 214–224, 2018.
https://doi.org/10.1007/978-3-319-95270-3_17

Point cloud data for shape representation can be acquired using several different sensors (e.g. depth camera [1], LiDAR [2] or LADAR [3]) or can be even extracted from image data [4]. In our case the LiDAR and/or LADAR sensors were used to acquire the point cloud data representing the shape of the world (terrain, vegetation, buildings, etc.). While the LiDAR data is acquired from the planes (see Fig. 1a) the LADAR data is acquired from stationary ground position using gimbal system (see Fig. 1b).



(a) (b)

Fig. 1. Figure shows the acquisition process for LiDAR terrain data by plane (a) and an example of LADAR sensor structure (b)

Point cloud data obtained with LADAR or LiDAR system are mostly used for detection and tracking purposes [5,6], but to get better perception of the data it is crucial to visualize them. Real-time visualization of large point cloud data sets with up to tens or hundreds of millions of points poses a challenge even with today's GPU hardware. This is even more true if we want to display real-time streaming data with lots of newly added points per time frame (e.g. hundreds of thousands of points per second). To achieve such performance even on consumer accessible hardware (or even on mobile devices) several limitations have to be defined. A well accepted approach for processing as well as visualization of point cloud data is presented in [7] where authors present a programming library for platform specific applications. An immersive visualization using dedicated platform specific software for CAVE setup is presented in [8]. Some approaches convert the point cloud data into mesh geometry before visualization. Such approach which uses rapid Delaunay triangulation for generating the mesh model is presented in [9].

A very important aspect for fast real-time visualization of large point cloud data sets is also the ability to process and provide such large amount of data fast enough to the visualization system. LASTools [10] were developed for such purpose, which are fast command line tools for simple processing of point cloud data. For streamed data this speed is defined by speed of input sensors, but providing the already stored large amount of data at high speed is also important [11].

While the data visualization for as single user is the main goal, it is often appreciated if the same data visualization can bee seen by multiple users at the same time or even if those users have an option of mutual interaction with the data. Such collaborative approach is well known in online productivity tools (e.g. Google Docs) and even in some specialized scientific tools. Such example for collaborative annotation of medical data with use of modern web technologies [12] does not only allow the collaboration between user at a distance but also allows replacement of specialized software and hardware with a modern web browser. A similar concept is exploited in the presented system.

In the following section we present the developed system for real-time visualization of 3D point cloud data with support for real-time data streaming, collaborative viewing and labeling of the data. In Sect. 3 we present evaluation of the developed system and it's results. In the final Sect. 4 we present the conclusions and give the pointers for possible future work.

2 Developed System

The main goal was to determine whether it is possible to develop a real-time massive point cloud visualization system with support for visualization of real-time streamed data with support for multi-user viewing and labeling of the displayed data. Figure 2 shows the basic structure of the system: (1) Data Sender[1] is a standalone application for reading raw sensor data and sending it to desired application, (2) Data Preparation Module is a server based application for acquiring the data, packaging the data into desired form and sending the data to (3) Data

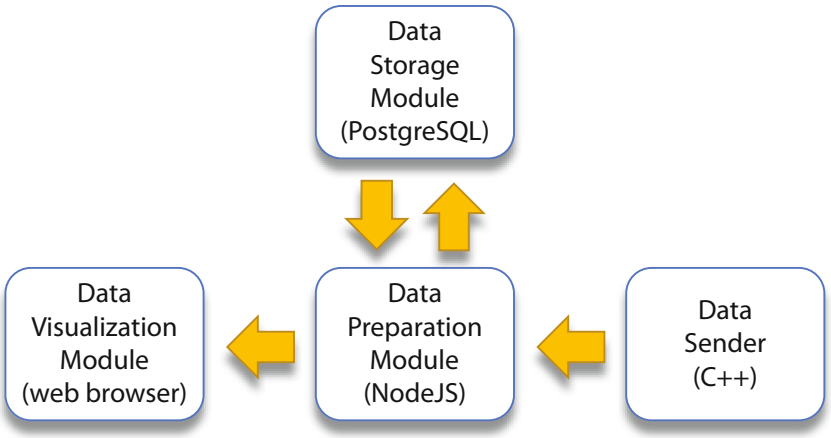


Fig. 2. The basic system structure.

[1] The application is developed in C++ and does was not developed as part of the presented work.

Storage module or (4) Data Visualization Module – a web application with support for WebGL for real-time visualization of point cloud data. The developed modules (2–4) are presented in more detail in the following subsections.

2.1 Data Preparation Module

The Data Preparation Module is a NodeJS[2] based server-side application developed for linking the Data Visualization Module with data resources. Data resources can be either real-time data provider such as Data Sender or off-line providers such as Data Storage Module. While data from Data Sender is acquired from sensors and directly sent to Data Visualization Module, it is up to the Data Preparation Module to request the data from Data Storage Module.

The application supports multiple connections and can simultaneously handle multiple data streams (live streams and data storage requests) and pipe them to multiple users individually. This allows multiple users to request data from diverse data sources. The module will prepare the data and send it to the users who requested it. The system does not support user management, but distinguishes between clients using uniquely generated tokens. The layer of security was not implemented at this stage of development due to specific use-cases where there is not such need (completely independent computer network or single computer deployment).

The Data Preparation Module also allows to simultaneously send the data obtained from Data Sender to data storage module, where the data is stored, and to multiple data visualization modules where the data is displayed in real-time. The communication between Data Preparation Module and Data Visualization Module is implemented using Socket.IO[3] – a library for fast and reliable continues communication. The communication with Data Sender is implemented using standard UTP sockets.

The data from Data Sender is received in form of binary blocks, where each block contains information from 200 points. Each point is represented with x, y and z position values with 32-bit float precision. Each package is directly submitted to all the clients (Data Visualization Modules) who are subscribed to live data streaming.

For preparation of stored data, module allows customization of query commands sent to the Data Storage Module. Some parameters of for query commands can also be defined by users through settings in Data Visualization Module. The data obtained from the Data Storage Module is packed and sent in JSON form to the client which requested it, but can also be packed into binary form for faster transmission.

While the basic functionality of module is to obtain, prepare and send the point cloud data, it can be easily extended for use with data from other domains as well. Such example might be the use of ortho-photo data or other GIS data for purposes of better/different final visualization.

[2] https://nodejs.org/.
[3] http://socket.io/.

2.2 Data Storage Module

Data Storage Module consists of a NodeJS based server-side application for interfacing with PostgreSQL[4] relational database with Pointcloud extension[5]. The extension deals with the variability of point dimensionality by using custom database schema which describes the content of individual point in the database. This takes care of number of point dimensions and their data types and possible scaling and/or offsets between the actual values and values stored in database.

The above presented storage is not the fastest possible option for storing large amounts of point cloud data, but offers a good trade-off between complexity of use, speed and price [11].

2.3 Data Visualization Module

The Data Visualization Module was developed as web application using Angular[6] and Bootstrap[7] for implementing interactive responsive GUI, and ThreeJS[8] and Potree [13] for implementing fast and reliable 3D visualization using WebGL technology.

The module's GUI consists of two parts as can be seen in Fig. 3: (1) preferences panel on the left, where users can select the data source and set the

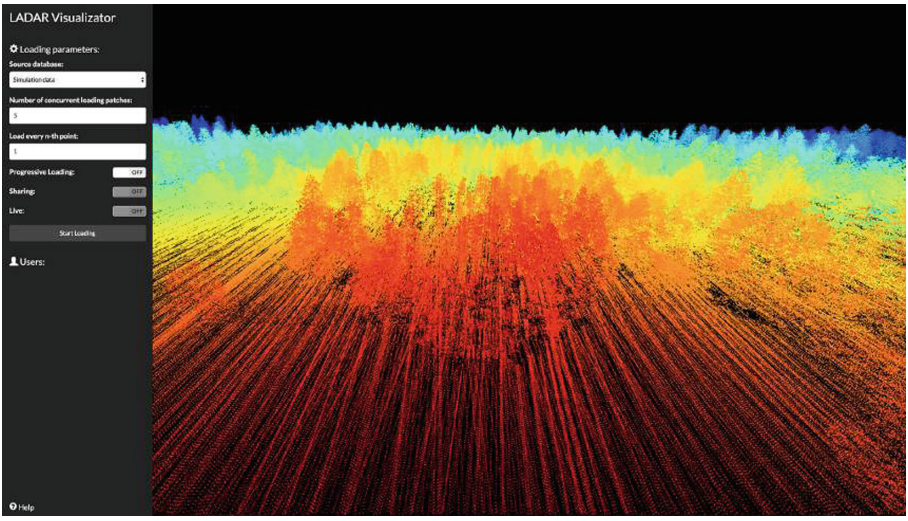


Fig. 3. The data visualization module GUI.

[4] https://www.postgresql.org/.
[5] https://github.com/pgpointcloud/pointcloud.
[6] https://angular.io/.
[7] http://getbootstrap.com/.
[8] https://threejs.org.

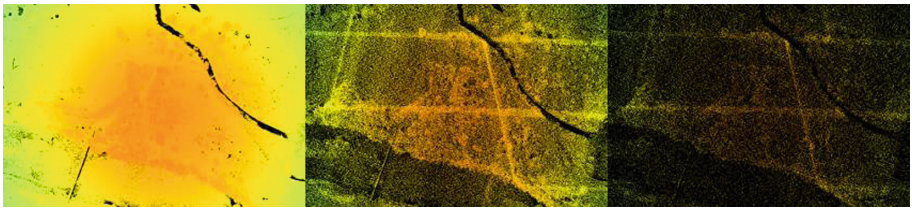
parameters of the visualization and (2) visualization panel on the right, where users can see and interact with displayed data.

In preference panel users can select data source from one of the existing databases or they can turn on the live data receiving option. User can set next database point loading parameters:

Number of Concurrent Loading Patches: points in database are stored in patches containing several number of neighboring points according to their spatial position. This allows faster retrieval of neighboring points. We can choose how many patches of points we want to load in one call and thus speeding up the database response time.

Load Every n-th Point: in dense point clouds it is not necessary to load all the points to get the idea of the shape they are representing. For this reason users can choose the density of points they want to display selecting only every n-th point in the database according to the geolocation sorting. This also means that client will draw less points which will result in higher frame rate. An example of visualization of same data with different setting can be seen in Fig. 4. More complex methods of level-of-detail approach are planned for future work.

Progressive Loading: when the option of loading every n-th point has value higher than 1 this switch will define whether after every n-th point is loaded the loading will continue with incremented offset or not.



(a) Every point is loaded. (b) Every 10-th point is loaded. (c) Every 50-th point is loaded.

Fig. 4. The comparison of the same data visualization with different setting for loading every n-th point.

Users can navigate in the visualization scene with use of mouse and keyboard. Mouse is used for simple view rotation and zooming. The keyboard is used for moving the camera view through the scene, speed of navigation can be adjusted by changing the velocity multiplier.

When switch for Live data visualization is active the visualization module is waiting for the data to be piped from Data Sender through Data Preparation Module in real-time. If no Data Sender is connected to Data Preparation Module or no data is sent, the Data Visualization Module will wait and no data will be rendered.

Colors of visualization display the distance from camera to individual point. A jet color-map is used to represent distances from the viewpoint (red being

closest and dark blue being the farthest away). Color-map is scaled for individual data set and can be adjusted easily. To make distance even more apparent the size of rendered points decreases with distance.

The visualization module also supports selection of desired points in the point cloud. Because the interaction with individual point would require too much overhead only selection of point groups is supported in the current version. The size of selection group can be defined as parameter. In our case the size of few thousands points per group in data set with millions of points proved to be best option for good performance. To allow faster selection of groups of points we allow two-level point selection: (1) single group under mouse cursor is selected and (2) all the groups with points under mouse cursor are selected. The focused points are rendered with pink color while selected points are rendered in white (both colors are not in jet color-map and are thus easily distinguishable from the rest). An example of focused and selected point groups can be seen in Fig. 5.

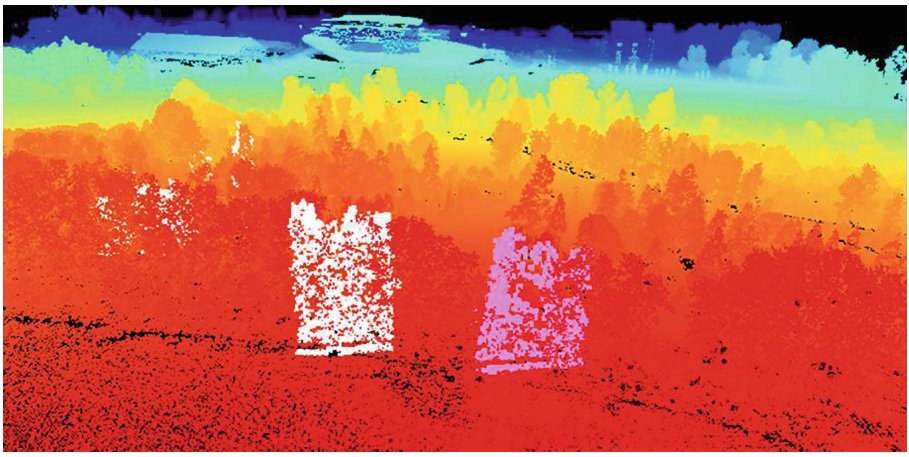


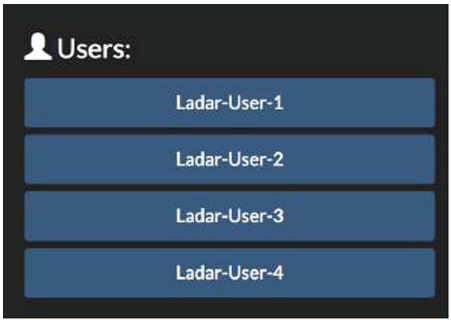
Fig. 5. The example of focused points (pink) and selected points (white). (Color figure online)

To get even better visualization experience, complete Potree visualization pipeline could be integrated, adding point shadowing and other more advanced visualization features. In our case we only followed few implementation solutions used in Potree library (point render size, server-client communication and few other features).

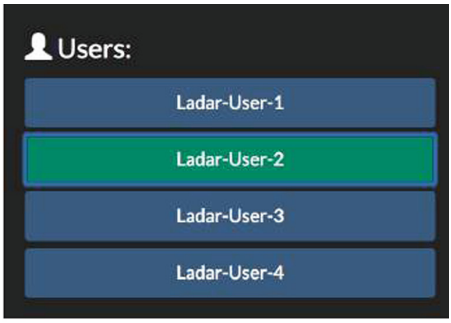
2.4 User Collaboration

Collaboration allows users to share their view on data with other users in real time similar to [14]. The user initiates sharing by activating share switch in preferences panel. Once activated other users can see such users in list of users

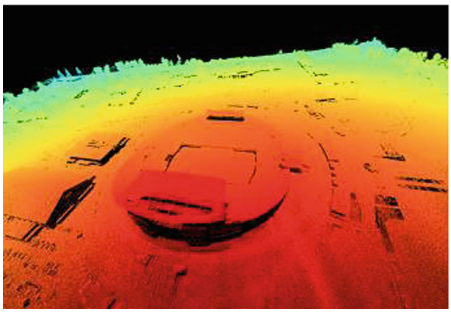
(as shown in Fig. 6a) and connect to a selected user by clicking on button with his/her designation (see Fig. 6b). Users that want to join the shared session must do so before the host user initiates data loading. After the loading is initiated by host other users can not connect anymore. The initiation of the loading starts loading the data from same database for all the connected users. The data loading is independent for each user due to the different speeds of the network connections and request handling speed of individual client. Users can navigate the scene separately and change their view on the data as is presented in Fig. 6c (view of the main user) and 6d (view of the connected user).



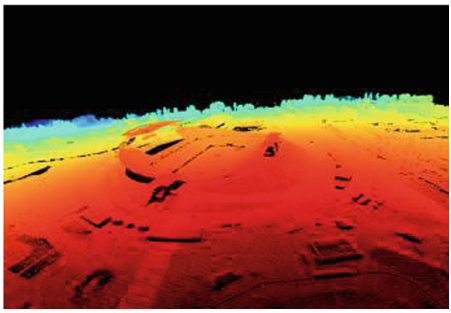
(a) List of users who are sharing their view.



(b) Connected to a selected user.



(c) The view on the data for main user.



(d) The view for user connected to user (c)

Fig. 6. The sharing options.

The sharing is implemented as presented in the Fig. 7. Currently only the following types of sharing is supported by the system: (1) camera view parameters, (2) loading initiation cancellation and (3) data selection. While view and loading initialization are only shared in one direction (from host to connected users) the data selection is supported in both directions and the selection state is kept on the server. The framework supports simple implementation for sharing other data or parameters as well.

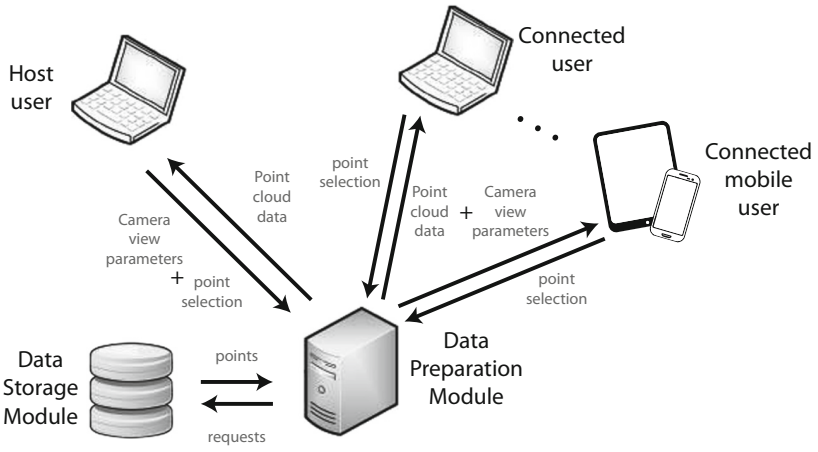


Fig. 7. The diagram is showing how the shared data is distributed.

3 Evaluation and Results

We only conducted preliminary evaluation of the system. The developed system allows interactive visualization of datasets with up to tens of millions of points on a consumer based hardware: i7 (2.3 GHz), 16 GB RAM, NVIDIA GeForce GT 650M 1024 MB. The visualization had on average more than 25 FPS. Streaming live data from the Data-Sender module on local network speed was approx. 200.000 points per second which presented the upper limit for the sending system. Streaming data from the server-side database system speed was approx. 30.000 points per second. The bottleneck in this case was the database system which ran on a low-end infrastructure, also the time for querying the database was a bit slow. The visualization test was performed with Autzen sample data supplied with liblas[9] library, which was imported into PostgreSQL database on virtual machine with single core (2.3 GHz) and 4 GB RAM. The loading of 10.6 million points in the dataset took on average 341 s while sending the data in JSON format over the wireless network connection.

The collaborative features worked fine on local network as well as over the Internet. This was expected since the data is sent to each client individually except the annotation information which is only sent between connected clients. As part of future work we are also planning more in-depth evaluation of the system.

4 Conclusion

The presented system is used for real-time visualization of real-time as well as stored point cloud data. The preliminary results show that the presented system

[9] https://www.liblas.org/samples/.

can be used in real-life scenarios. While there are some downsides to slow data streaming from the database it is still sufficient for the tested scenarios. However in the future we are planning to test different database solutions for faster data throughput. We are also planning on improving the client-side visualization implementation for easier user interaction and navigation as well as support for level-of-detail display of data with support for user defined detail parameters. Other possible future work contains reimplementation of user annotation and selection of the displayed data.

References

1. Du, H., Henry, P., Ren, X., Cheng, M., Goldman, D.B., Seitz, S.M., Fox, D.: Interactive 3D modeling of indoor environments with a consumer depth camera. In: Proceedings of the 13th International Conference on Ubiquitous Computing, pp. 75–84. ACM (2011)
2. Haala, N., Peter, M., Kremer, J., Hunter, G.: Mobile LiDAR mapping for 3D point cloud collection in urban areas - a performance test. Int. Arch. Photogrammetry Remote Sens. Spat. Inf. Sci. **37**, 1119–1127 (2008)
3. Molebny, V., McManamon, P.F., Steinvall, O., Kobayashi, T., Chen, W.: Laser radar: historical prospective-from the east to the west. Opt. Eng. **56**, 56 (2016). https://spie.org/publications/journal/10.1117/1.OE.56.3.031220?SSO=1
4. Rosnell, T., Honkavaara, E.: Point cloud generation from aerial image data acquired by a quadrocopter type micro unmanned aerial vehicle and a digital still camera. Sensors **12**(1), 453–480 (2012)
5. Wang, C.C., Thorpe, C., Suppe, A.: LADAR-based detection and tracking of moving objectsfrom a ground vehicle at high speeds. In: IEEE IV2003 Intelligent Vehicles Symposium. Proceedings (Cat. No.03TH8683), pp. 416–421, June 2003
6. Navarro-Serment, L.E., Mertz, C., Hebert, M.: Pedestrian detection and tracking using three-dimensional LADAR data. Int. J. Robot. Res. **29**(12), 1516–1528 (2010)
7. Rusu, R.B., Cousins, S.: 3D is here: Point cloud library (pcl). In: IEEE International Conference on Robotics and automation (ICRA) 2011, pp. 1–4. IEEE (2011)
8. Kreylos, O., Bawden, G.W., Kellogg, L.H.: Immersive visualization and analysis of LiDAR data. In: Bebis, G., Boyle, R., Parvin, B., Koracin, D., Remagnino, P., Porikli, F., Peters, J., Klosowski, J., Arns, L., Chun, Y.K., Rhyne, T.-M., Monroe, L. (eds.) ISVC 2008. LNCS, vol. 5358, pp. 846–855. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89639-5_81
9. Su, T., Wang, W., Lv, Z., Wu, W., Li, X.: Rapid delaunay triangulation for randomly distributed point cloud data using adaptive hilbert curve. Comput. Graph. **54**, 65–74 (2016). Special Issue on CAD/Graphics 2015
10. Hug, C., Krzystek, P., Fuchs, W.: Advanced lidar data processing with lastools. In: ISPRS Congress, pp. 12–23 (2004)
11. van Oosterom, P., Martinez-Rubi, O., Ivanova, M., Horhammer, M., Geringer, D., Ravada, S., Tijssen, T., Kodde, M., Gonçalves, R.: Massive point cloud data management. Comput. Graph. **49**(C), 92–125 (2015)
12. Lavrič, P., Bohak, C., Marolt, M.: Collaborative view-aligned annotations in web-based 3D medical data visualization. In: MIPRO 2017, 40th Jubilee International Convention, 22–26 May 2017, Opatija, Croatia, proceedings, pp. 276–280 (2017)

13. Schütz, M.: Potree: Rendering Large Point Clouds in Web Browsers. Master's thesis, Institute of Computer Graphics and Algorithms, Vienna, University of Technology, Favoritenstrasse 9–11/186, A-1040 Vienna, Austria, September 2016
14. Marion, C., Jomier, J.: Real-time collaborative scientific WebGL visualization with WebSocket. In: Proceedings of the 17th International Conference on 3D Web Technology, pp. 47–50. ACM (2012)