Virtanen, Juho Pekka; Daniel, Sylvie; Turppa, Tuomas; Zhu, Lingli; Julin, Arttu; Hyyppä, Hannu; Hyyppä, Juha

Interactive dense point clouds in a game engine

Published in:
ISPRS Journal of Photogrammetry and Remote Sensing

DOI:
10.1016/j.isprsjprs.2020.03.007

Published: 01/05/2020

Document Version
Publisher's PDF, also known as Version of record

ISPRS Journal of Photogrammetry and Remote Sensing 163 (2020) 375–389

Contents lists available at ScienceDirect

ISPRS Journal of Photogrammetry and Remote Sensing

journal homepage: www.elsevier.com/locate/isprsjprs

Interactive dense point clouds in a game engine

Juho-Pekka Virtanen[a,c,*], Sylvie Daniel[b], Tuomas Turppa[c], Lingli Zhu[c], Arttu Julin[a], Hannu Hyyppä[a], Juha Hyyppä[c]

[a] Aalto University School of Engineering, Department of Built Environment. P.O. Box 14100, FI-00076 Aalto, Finland
[b] Laval University, Department of Geomatics Sciences. 1055, Avenue du Seminaire, Quebec, QC G1V0A6, Canada
[c] National Land Survey of Finland, Finnish Geospatial Research Institute FGI, Geodeetinrinne 2, FI-02430 Masala, Finland

ARTICLE INFO

Keywords:
Point cloud
Game engine
VR

ABSTRACT

With the development of 3D measurement systems, dense colored point clouds are increasingly available. However, up to now, their use in interactive applications has been restricted by the lack of support for point clouds in game engines. In addition, many of the existing applications for point clouds lack the capacity for fluent user interaction and application development. In this paper, we present the development and architecture of a game engine extension facilitating the interactive visualization of dense point clouds. The extension allows the development of game engine applications where users edit and interact with point clouds. To demonstrate the capabilities of the developed extension, a virtual reality head-mounted display is used and the rendering performance is evaluated. The result shows that the developed tools are sufficient for supporting real-time 3D visualization and interaction. Several promising use cases can be envisioned, including both the use of point clouds as 3D assets in interactive applications and leveraging the game engine point clouds in geomatics.

1. Introduction

Methods for producing dense, colored point clouds are increasingly available. Three-dimensional measuring and reconstruction can be achieved with image-based techniques (e.g., Toschi et al., 2017; Micheletti et al., 2015), laser scanning, or a number of other techniques (e.g., structured light systems). 3D measuring methods have improved both in efficiency (e.g., Nocerino et al., 2017; Kukko et al., 2017; Li et al., 2019) and consumer availability (e.g. Diakité and Zlatanova, 2016; Hyyppä et al., 2017; Zollhöfer et al., 2018). In addition, point cloud data sets are already being offered as open data by national mapping agencies. There are also projects focused on digitally archieving individual sites in more detail (Reardon, 2012). The availability of measuring systems and complete data sets has stimulated the research on point cloud data processing and application. Analysis methods using point clouds have been introduced to ecology (Saarinen et al., 2018), forestry (Hyyppä et al., 2017), city and building modeling (Dorninger and Pfeiffer, 2008; Haala and Kada, 2010; Maas and Vosselman, 1999) and map updating (Matikainen et al., 2017). Consequently, dense point clouds have been identified as a significant data type by several researchers (Virtanen et al., 2017; Cura et al., 2017; Poux et al., 2016; Otepka et al., 2013). This has led to the development of management systems for massive point cloud data sets (van Oosterom et al., 2015; Cura et al., 2017; El-Mahgary et al., 2020), and point cloud visualization (Deibe et al., 2019).

Several authors have studied point cloud visualization, including Nebiker et al. (2015), Richter et al. (2015), and Tredinnick et al. (2015), with direct point-based rendering gradually emerging as a viable alternative to the more traditional polygonal mesh methods (Goswani et al., 2013; Tschirschwitz et al., 2019). In some cases, point based rendering has even been suggested as a solution for rendering large meshes (Rusinkiewicz and Levoy, 2000). Aims in research on point cloud visualization have varied from data integration (Nebiker et al., 2015) to the use of immersive display devices (Tredinnick et al., 2015; Zhao et al., 2019; Kharroubi et al., 2019) or browser based 3D-visualization (De La Calle et al., 2011; Zeng et al., 2012; Discher et al., 2019; Deibe et al., 2019). Many of the applied systems have been customized for the manufacturers themselves (Ye et al., 2016). Powered by WebGL, some renderers have recently become available for the visualization of point cloud data over the Web (Ye et al., 2016; Martinez-Rubi et al., 2015; Discher et al., 2019; Schütz, 2016). Point cloud visualization has also been realized on top of the online virtual globe Cesium (CesiumJS, 2019; Discher et al., 2019). The platform supports the creation of the interactive web apps for sharing dynamic geospatial

* Corresponding author at: Aalto University School of Engineering, Department of Built Environment. P.O. Box 14100, FI-00076 Aalto, Finland.
E-mail addresses: juho-pekka.virtanen@aalto.fi (J.-P. Virtanen), sylvie.daniel@scg.ulaval.ca (S. Daniel), tuomas.turppa@nls.fi (T. Turppa), lingli.zhu@nls.fi (L. Zhu), arttu.julin@aalto.fi (A. Julin), hannu.hyyppa@aalto.fi (H. Hyyppä).

https://doi.org/10.1016/j.isprsjprs.2020.03.007
Received 4 April 2019; Received in revised form 5 March 2020; Accepted 5 March 2020
Available online 08 April 2020

data, and utilizes the 3D Tiles format (OGC, 2019) for streaming massive geospatial datasets, including point clouds, for visualization in a web browser. In Cesium 3D Tiles (OGC, 2019), a tileset is a set of tiles organized in a tree structure such as k-d trees, quadtrees, octrees, or grids. Each tileset contains tileset metadata and a tree of tile objects. The tile's content is a binary blob containing a feature table and a batch table. A feature table stores the feature's position and its appearance properties. A batch table stores additional application-specific properties.

Game engines have become highly efficient tools for developing interactive 3D applications, for both professional (Indraprastha, & Shinozaki, 2009; Virtanen et al., 2015; Wirth et al., 2019) and entertainment purposes (e.g., Rua & Alvito, 2011; Alatalo et al., 2016). They are also commonly used for building VR environments viewed with HMDs (e.g., Donghui et al., 2017; Jamei et al., 2017; Rua & Alvito, 2011; Alatalo et al., 2016). One of the advantages of game engines as a development platform is that they allow efficient development of applications where the users interact with the 3D environment (Wirth et al., 2019). This interaction potential is essential for many professional applications, as it allows the user not only to benefit from the visualization but also to provide output (Nguyen et al., 2016). However, in using the game engines for visualization of geospatial data, one of the commonly encountered challenges has been the amount of work required in producing game engine compatible models from dense point cloud data (e.g. Tschirschwitz et al., 2019). While tools exist to allow automatic generation of suitable textured mesh models in some cases (Julin et al., 2019), their generation still represents an additional computational step compared to direct point cloud visualization. Compared to mesh models, the point clouds would offer a direct, unfiltered access to original 3D reconstruction data and allow easy data integration from any point cloud production method (Discher et al., 2019).

While systems for visualizing point clouds in VR and rendering them in game engines have been developed (e.g., Point Cloud Viewer and Tools, 2017; Point Cloud Plugin, 2019), the systems that allow the use of point clouds in game engines as moveable, interactable assets are still missing. And current point cloud visualization software is still not flexible enough to allow application development. Our aim is to develop and implement a game engine extension facilitating the use of dense colored point clouds as interactable objects in a 3D game engine. We review the relevant point cloud organization and rendering solutions, and present our rendering approach combining different shader types with an octree-based implementation allowing interaction with 3D point cloud including dynamically movable point cloud based objects as a part of the octree.

As a demonstrator, we implement an application using the developed tools, in which the user is able to visualize and interact with a set of point clouds in a VR environment. In this article, we first review the relevant optimization and rendering strategies, then we present the development and architecture of the extension, and finally we present its use in a demonstrator application and evaluate the results.

2. Dense point cloud storage and visualization

2.1. Influence of point cloud acquisition methods to visualization

3D point clouds depicting the environment can be obtained with a variety of methods, such as SAR (Karjalainen et al., 2012), laser scanning (Vosselman & Maas, 2010), photogrammetry (Szeliski, 2011), and depth cameras (Zollhöfer et al., 2018), to name a few. To some extent, the measuring techniques also influence the visualization of the point cloud data sets.

In laser scanning (LS) methods, a common limitation is the inability of a laser scanner operating on a single wavelength to capture surface color information. In a majority of systems, this has been overcome by adding an imaging sensor and then resolving the calibration and synchronization between the laser scanner and camera (Lerma et al., 2010). However, this implies that the quality of point cloud colorization is subject to the quality of the camera system used to obtain imagery for colorization (as noted for texturing by Julin et al., 2019). In photogrammetric reconstruction, the 3D geometry and textures can be derived from the same image set. However, purely image based reconstruction can be problematic in e.g. indoor environments, where flat, textureless surfaces lack the detectable features (Lehtola et al., 2014). Laser scanning systems, on the other hand, may suffer from artifacts originating from sensor errors, causing stray points in the data, co-registration errors, temporal changes in the scene, sensor noise etc.

In LS, the point density varies according to distance and multiple other factors e.g. the scanning mechanism, possible platform movement, laser beam divergence, target geometry and reflectance etc. This generates varying artifacts in terrestrial (TLS) and mobile laser scanning (MLS) (Vaaja et al., 2011). The emerging systems capable of simultaneous localization and mapping (SLAM) significantly increase the efficiency of 3D mapping, especially in indoor environments (Nocerino et al., 2017). In addition to indoor mapping, SLAM systems have seen a lot of development and applications in robotics (Nüchter et al., 2007). In these cases, the point cloud density varies by distance and movement of the system, often leaving the trajectory of the system visible in the data.

The footprint size of the laser beam on the target varies according to the used sensor and scanning technique, distance, and target geometry etc. In addition, different laser scanning systems produce different point cloud densities, especially if comparing ALS point clouds to those obtained with contemporary TLS instruments. More dense airborne data can be obtained from UAV-LS, from a lower flying altitude (Jaakkola et al., 2010). The ability of laser scanning systems to penetrate vegetation also varies, with full-waveform systems achieving better canopy penetration (Yu et al., 2014).

For individual objects, structured light scanners (Lachat et al., 2017), photogrammetry or laser scanning can be applied. The Microsoft Kinect represents one of the most popular devices based on structured light projection. It exploits an RGB camera, an IR camera, and an IR projector (Weinmann, 2016). Such a solution is used not only for objects but also for scanning the indoor and outdoor environment (Labrie-Larrivée et al., 2016). Lechat et al. (2017) report attaining dense point clouds with geometric errors remaining below 5 mm for large (~1 m) cultural heritage artifacts using the Faro Freestyle instrument. Typically, depth camera instruments suffer from a higher amount of noise and more limited range than laser scanners, also affecting the visualization of depth camera point clouds.

In addition, ready point cloud data sets are increasingly offered. Utilizing airborne laser scanning (ALS) systems, national mapping agencies have initiated large surveying campaigns. Some of these point cloud data sets have been released as open data (e.g., Actueel Hoogtebestand Nederland, 2019; Geoportal Thüringen, 2019; National Land Survey of Finland, 2019; Scottish Remote Sensing Portal, 2019). In many openly released ALS datasets, the point density has been quite low, reducing their utility in point cloud visualization. At the same time, these point cloud data sets are often classified and represent meaningful features (e.g. ground, vegetation, building roofs etc.), offering also additional potential for visualization.

2.2. Point cloud storage and retrieval strategies

As dense point clouds may contain hundreds of billions of points, sophisticated organization is a requirement for efficient visualization and processing. Appropriate spatial data structures are needed to speed up search queries. In the following, we review the key issues and their existing solutions concerning visualization of dense point clouds.

The simplest data structures (arrays and lists) are not suitable for storing large amounts of data alone, as unorganized point sets easily result in excessive search times. Over the last three decades, a variety of

different spatial indices have been developed to overcome this. These spatial indexing systems subdivide the domain either in a regular or irregular manner. As there is no optimal index for all situations, a few common approaches are used, depending on the case.

2.2.1. Voxel and grid structures

Voxelization, referring to the conversion of point data into volumetric data stored in a 3D array of voxels (Karabassi et al., 1999) can be used to store point data. For example, it has been applied to ALS-derived point cloud (Wand et al., 2017). Voxelization has also been integrated into 3D surface reconstruction (Xu et al., 2015), rendering (Crassin, 2011), robotics (Hornung et al., 2013) and point cloud segmentation and clustering (Chen et al., 2019). While voxel systems are simple to implement, they also become limited in performance and memory requirements with large data sets. Furthermore, a lack of multiple resolutions prevents the efficient execution of spatial algorithms. In some cases, this has been overcome by combining multi-level octree structure with voxel-style algorithms (Poux and Billen, 2019a).

Additionally, voxel (or 2D grid) structures are often applied as a starting point for producing more sophisticated, multi-level tree structures accomplished by merging and subsampling voxels or 2D grid cells, as presented for example in Yu and Mei (2019). In Deibe et al. (2019), a regular 2D grid-structure is used to attain multiple levels of detail in rendering by producing multiple non-duplicate subsets of each grid tile beforehand and obtaining the different levels of detail by selectively combining these. In database storage, it is also common to utilize a regular grid for storing the point cloud (van Oosterom et al., 2015).

2.2.2. Tree data structures

The size of the massive point clouds is usually larger than the main memory of the systems that have to handle them. In order to avoid subsampling the data (decreasing point density) or limiting their visualization to only small portions at once, multi-resolution data structures are used. The k-d tree, or k-dimensional tree, is a data structure used for organizing a number of points in a space with k dimensions. It is a very fast indexing method for nearest neighbor queries relying on a binary tree search. The k-d tree may rapidly get unbalanced in the case of update operations (insertion or deletion of points). Dedicated methods have been proposed to build a balanced k-d tree (Brown, 2015).

A subtype of the k-d tree—the quadtree (2D index)—is a tree data structure in which each internal node has exactly four children. Octrees are a three-dimensional analog of quadtrees, with each internal node having exactly eight children. While quadtrees are most often used to partition a two-dimensional space by recursively subdividing it into four quadrants or regions, octrees are most often used to partition a three-dimensional space by recursively subdividing it into eight octants. Due to the compression and multi-resolution properties of quad- and octree structures, global modeling and efficient algorithm execution are feasible (Elseberg et al., 2012), and octrees have indeed been applied for point cloud storage in analysis tasks as well (e.g. Bassier et al., 2019). Also, they are well-suited for update operations. By applying solutions commonly used with voxel data, the octree structure has been applied for multi-scale point cloud analysis (Poux & Billen, 2019b).

Even if efficient, quad- and octree structures cannot compete with k-d trees in terms of speed when performing nearest neighbor queries. However, quad- and octrees are often used for visualization, since they can be easily adapted to support level-of-detail (LOD) structures. Such LOD information is required to realize efficient out-of-core (i.e., external memory) rendering techniques (Richter et al., 2015). In order to achieve an LOD representation, quantized points are used. They are specified using the quadtree or octree structure. If there are one or more points in each grid cell, only one representative point is selected by a random manner and is stored in the node of the tree. Several authors have presented efficient techniques for out-of-core multi-resolution construction and high-quality interactive visualization of massive point clouds (Goswani et al., 2013; Kuhn and Mayer, 2015; Discher et al., 2019).

From a rendering point of view, the challenge in octree structure is choosing a suitable octree depth: if more points are stored in each node, they become heavier to render, but constitute less draw calls, if each of the nodes invokes a single draw call (Schütz, 2016). Some approaches introduce a hierarchical level of detail (LOD) organization based on multi-way kd-trees, which simplifies memory management and allows control over the LOD-tree height (Goswami et al., 2013). Implementations combining different tree structures have been proposed to better adapt the structure to the properties of laser scanning point clouds. Yu and Mei (2019) combined a variable cell size 2D index with a single level octree, noting that the structure is efficient for point clouds depicting larger areas of terrain where the height-component is less significant.

While in many implementations, the tree structures are produced in a pre-processing stage (Schütz, 2016) they can also be produced adaptively in real-time to facilitate loading data from a predefined grid structure, as in (Deibe et al., 2019). Schütz et al. (2019)on the other hand, use the modifiable nested octree structure of the Potree system (Schütz, 2016) to produce level-of-detail information for points, but do not apply this for point storage.

2.2.3. Database and file implementations

In the case of huge point clouds as obtained by current measurement technologies, the data clearly exceeds the amount of available computer memory. In addition, even if a sufficient amount of RAM is available, memory fragmentation makes it hard to allocate a large enough continuous memory block. Hence, the data needs to be stored in secondary storage (such as hard disk drives or solid state disks) and can be loaded into memory chunk-wise only. Several approaches have been tested and utilized for storing large point clouds.

Many of the systems that utilize tree structures rely on storing individual leaf nodes as binary files, which are then loaded as required when traversing the tree (e.g. Schütz, 2016). In similar manner, Deibe et al (2019) store each of the pre-produced sampling levels from each grid cell as individual file on the disk. However, Schütz et al. (2019) apply a different system, storing the entire cloud in a single file, with the intent of minimizing the reading operations. This is enabled by their iterative drawing buffer generation.

Database systems can also be applied for point cloud storage, with the most common approach being the division of point cloud into a regular grid and storing grid cells as compressed binary patches in rows of the database table (van Oosterom et al., 2015). Other division schemes, relying e.g. on the semantic information associated with the points can be applied as well (El-Mahgary et al., 2020), this also being applicable for file-based storage.

2.3. Point cloud rendering

From the visualization perspective, the dense point cloud datasets are not problem-free: While perspective projection of point cloud points can be carried out with the same algorithms used for triangle mesh vertices, rendering point clouds is inherently problematic due to lack of connectivity information. This introduces several issues for rendering, such as the inability to represent linear edges and a minimum display distance caused by missing data between points, which eventually causes the user to “see through” the surface.

2.3.1. Frustum culling

In computer graphics, the frustum is the visible region of space. Non-visible elements do not need to be drawn, hence increasing performance. The main bottleneck in a software culling implementation is transmitting the point information to the graphics card when a large number of them are within the viewing frustum. Elseberg et al. (2012) as well as Tredinnick et al. (2015) mitigate this problem using the octree structure. This approach allows the rendering quality to be dynamically adjusted. This is achieved by sending to the graphics card only

single vertices of volumes in the octree that fall below a level-of-detail threshold.

Frustum culling using nodes of a tree structure as units to be displayed or not is a common solution employed in several systems (e.g. Deibe et al., 2019; Schütz, 2016). The main benefit of this approach is that the decision to draw does not have to be made point-per-point, but rather by the node extents. In addition to excluding points from rendering, the viewing frustum is also commonly applied for choosing nearby nodes for drawing with a higher level of detail (e.g. Deibe et al., 2019). Schütz et al. (2019) point out that this commonly results in some artifacts in visualization, arising from observable borders of nodes of different levels of detail, that do not correlate with the geometric features of the point cloud.

2.3.2. Point drawing

Point cloud rendering algorithms can be divided into using primitives, such as quads (in screen space) and scaling them according to perspective projection, or alternatively, projecting primitives, such as disks, following surface orientation (in object space) (Sainz and Pajarola, 2004). In point cloud visualization, at least four different approaches have been proposed to overcome the issues caused by the lack of connectivity information between points (Kaushik et al., 2012):

1. Hole detection and filling in screen-space: Individual samples are projected on the screen, and the pixels not receiving samples are detected. The surface is then interpolated from the neighboring samples;
2. Generating more samples: A surface is adaptively interpolated in object-space to guarantee that every pixel receives at least one sample.
3. Splatting: A surface sample is projected on the screen (the result of the projection is called a splat), and its contribution is spread into the neighboring pixels to guarantee coverage. Higher quality methods average the contributions of all splats contributing to the pixel.
4. Meshing: A polygon mesh is used for interpolating the surface samples.

A comprehensive survey of point-based rendering is provided by Kobbelt and Botsch (2004). Splatting is advantageous since it can create a visually-continuous surface without the need for costly triangulation. In the context of LiDAR point cloud, Kovac and Zalik (2010) applied first a two-pass rendering to efficiently blend splats, then Kuder and Kovac (2012) combined splatting with deferred shading in a three-pass algorithm. Gao et al. (2012) improved such approaches by addressing the drawback of splatting related to areas with sparse or no points due to occlusions. Splatting still involves heavy precomputing as it needs normals and radii. This is a problem when the dataset is massive or when visualization should be performed on the fly.

Another way to render point clouds efficiently without preprocessing is to increase the size of the points. Schautz and Wimmer (2015) shows that this simple method can achieve good quality results by drawing, for each point, camera aligned paraboloids instead of flat squares. Allowing a varying point size also helps mitigate issues caused by varying point cloud density, and simultaneous display of point cloud segments of different levels of detail (Scheiblauer and Wimmer, 2011).

Another efficient approach for point cloud rendering is to use screen space operators. Marroquin and al. (2008) use the pull-push algorithm to reconstruct a filled color and depth buffer. However, they rely on precomputed normals to perform both reconstruction and shading. Preiner et al. (2012) use screen space nearest neighbor queries to compute normals and radii to feed splatting algorithm in real-time. Their algorithm is however very computational intensive. Bouchida et al. (2018) recent method uses as input a framebuffer with depth and an optional color after a single geometric pass that renders one pixel per point. The method is then based on three screen space pyramidal operators: a hidden point removal operator; the push-pull algorithm; the depth texture pyramid. Their method achieved good visual results on a wide range of point clouds types from photogrammetry to noisy mobile mapping. Hofer et al. (2018) also base their solution on manipulating depth information via 2D image processing, and attain a surface reconstruction for conventional mesh rendering via them.

2.3.3. Game engine implementations

In addition to commercially available game engine point cloud extensions (Point Cloud Viewer and Tools, 2017; Point Cloud Plugin, 2019), the utilization of game engines in point cloud visualization has been touched upon in research literature. Proposed applications include the utilization of VR for manual point labeling (Wirth et al., 2019) and use of real-time point cloud processing with depth camera data (Hofer et al., 2018). In many of the presented implementations, the point organization, retrieval and rendering strategies presented earlier have been implemented for commercial game engines, negotiating their specific limitations, such as maximum vertex count for an individual object (e.g. Vincke et al., 2019).

Vincke et al (2019) utilize a conversion from point clouds to mesh faces prior to game engine use, splitting the point cloud to segments to improve performance. Kharroubi et al (2019) utilize an octree structure and file based storage in combination with the Unity game engine, relying on work by Fraiss (2017). For rendering relief-like depth camera data, Hofer et al. (2018) utilize a set of buffers in Unity.

As game engines offer the possibility for drawing individual pixels, sets of pixels or mesh objects, different point rendering schemes can be implemented (Fraiss, 2017). As different drawing strategies carry varying computational costs, it may prove useful to combine them by utilizing a different drawing scheme for far-away points and those close to the camera (Santana et al., 2019).

3. Materials and methods

To facilitate interactive point cloud visualization in game engine environment, a set of methods were developed and implemented on the Unity game engine (Version 2018.2.15). The following chapters describe the key solutions in data storage, rendering and object interaction.

3.1. Point cloud loading & storage in game engine

Point cloud import is realized from LAS, PLY, or ASCII formats, reading the point coordinates, intensity, classification (if available), and RGB values. To maintain functionality with precision limits imposed by the 32-bit floats used in coordinates, a coordinate shift is applied to the point cloud to center it at the origin. In addition, the cloud can be scaled if wanted. This also allows easy utilization of a point cloud originating in different units.

The most significant processing step performed for the point cloud prior to rendering is its division into an octree. The scaled and shifted point cloud is divided into an octree structure, with a depth specified by the user. An octree with n-depth is generated to fill the bounding box of the point cloud. Point data is then fed to the root, which will divide the data into its immediate children and repeat the division for each child until the final depth has been reached. Unlike in Schütz (2016), the point cloud is only stored in the final octree nodes. This allows controlling the rendered area is as small regions as possible via occlusion culling, and simplifies the implementation of interaction functions (in Section 3.4) as several overlapping levels in octree do not have to be considered.

After processing, the point geometry, normal vector, and RGB values are stored in a binary format along with octree structure information. A single point consists of coordinates (three 32-bit floats), a normal index (16-bit unsigned integer) used to access normal values from a separate lookup table, and a point ID (16-bit unsigned integer).

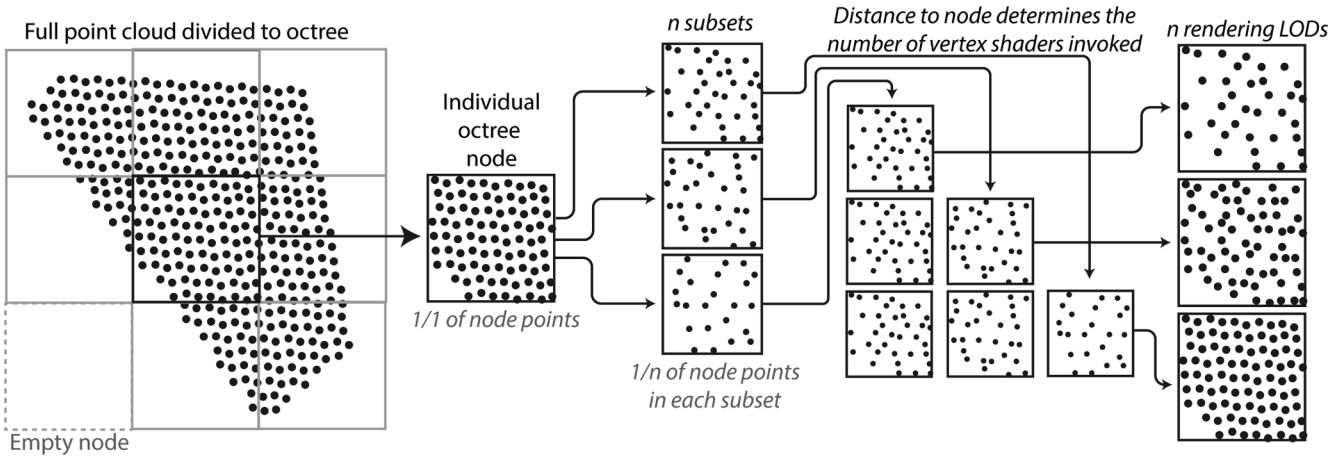

Fig. 1. Principle of utilizing sampled point sets to produce LODs by controlling the number of vertex shaders invoked. The points of an individual node are divided into n subsets, after which n LOD levels can be produced by invoking n shader instances drawing individual points. The number of LOD levels is arbitrary, but it has to be defined prior to octree generation.

The point ID can be used later to bind additional information layers to points. Prior to rendering, the data is stored in two different structured buffers, one of them containing the performance-wise aligned point data for fast reading and handling, the other buffer providing a mask for quick data manipulation.

3.2. LOD via vertex shader invocation

To generate the amount of LOD levels (as specified by the user), points are captured to each node by iterating the parent node *n* times (Fig. 1). Each iteration captures every *n*th point of the total points in an individual octree node. The resulting subsets of the particular node each contain a set of unique points, forming the total set of node points when combined. A similar concept has been utilized in 2D grid structure by Deibe et al. (2019).

The point subsets are utilized in rendering to produce different LODs by invoking a desired amount of vertex shader instances for each node. For each draw call, *n* vertex shader instances are invoked, determined by the distance between the user and the rendering node, each facilitating the drawing of a prebuilt subset of points in the specific node (Fig. 1). For higher levels of detail, more shader instances are invoked, each drawing a subset of the node's points. The applied shaders support real-time lighting (directional and point lights), which are calculated in the vertex shader. The impact of the LOD system is illustrated in Fig. 2, showing from top view how the point cloud density is reduced as the range to camera increases. The nodes that remain outside the camera view frustum are remain hidden. The rendering procedure is summarized in Fig. 3.

This approach detaches the rendering LOD from the octree depth, unlike in systems where increasing the LOD is performed by traversing the octree to the next level (e.g. Schütz, 2016). This is especially beneficial for maintaining the amount of draw calls: for a conventional octree structure, with all nodes populated and three LOD levels, the highest LOD would invoke 21 draw calls (1 for the first level, 4 for the consecutive level, and 16 for the children of the middle-level). In our solution, three LOD levels can be realized with a minimum of three draw calls for the highest LOD. Our solution maintains a more consistent amount of draw calls over a range of LODs.

3.3. Multi-shader point cloud rendering

For rendering the point cloud, a multi-shader strategy is applied based on the distance between the point cloud and the virtual camera, in similar fashion as in Santana et al. (2019). Prior to rendering, a frustum culling utilizing the octree structure is implemented to reduce rendering load. The octree nodes outside the viewing frustum are locked out of rendering. For the remaining nodes, we calculate the distance to the virtual camera, and then choose a shader program accordingly. For distant nodes, a simplistic vertex- and fragment shader is applied. For nodes closer to the camera, two alternative approaches are implemented: a geometry shader method and a vertex dereferencing method (the latter one is offered for compatibility with older hardware, these methods are described later). The multi-shader strategy allows us to combine highly efficient rendering of points as individual pixels (in "Far pass") for more distant points, and rendering larger quad primitives oriented towards the camera for closer points (Fig. 3). As described in Section 3.2, the LOD of rendering is controlled via invoking a varying amount of shaders. This applies to both rendering the more distant points with the vertex shader, and the closer points with either of the two shader methods.

The **geometry shader method** uses a two-pass shader, with the first pass involving a geometry shader for producing renderable geometry from points. In this approach, there is no need for additional data, and the point geometry can be generated in render time, only when needed, and it also grants great flexibility to manipulate the geometry. The disadvantages of this method are the backward compatibility with older hardware (especially pre-DirectX 11.0) and potential performance differences between graphic cards.

The **Vertex dereferencing method** utilizes a simple vertex-fragment shader pipeline where we feed each vertex *n* times for each n-polygon. These duplicate instances are then dereferenced in a vertex shader by division into individual vertices from a predefined list defining the geometry, using a vertexID passed to the shader. This approach is much more backward-compatible and also more available to different platforms, as it does not use any advanced shader stages and takes the same amount of CPU memory. However, due to the LOD system, this will introduce one additional shader state change, as it needs to change between drawing points and drawing triangles or quads.

3.4. Real-time interaction with point cloud objects

To facilitate real-time interaction with the point cloud objects in the game engine, a set of functions were developed. These allow the use of point cloud segments as dynamically moved objects in the game engine, selecting such objects with ray-casting, and selective re-coloring or hiding of points dynamically. The dynamically manipulatable point clouds and ray-casting them are analogous to using different mesh models to represent different objects in a game engine, and facilitating e.g. object selection via ray-casting.

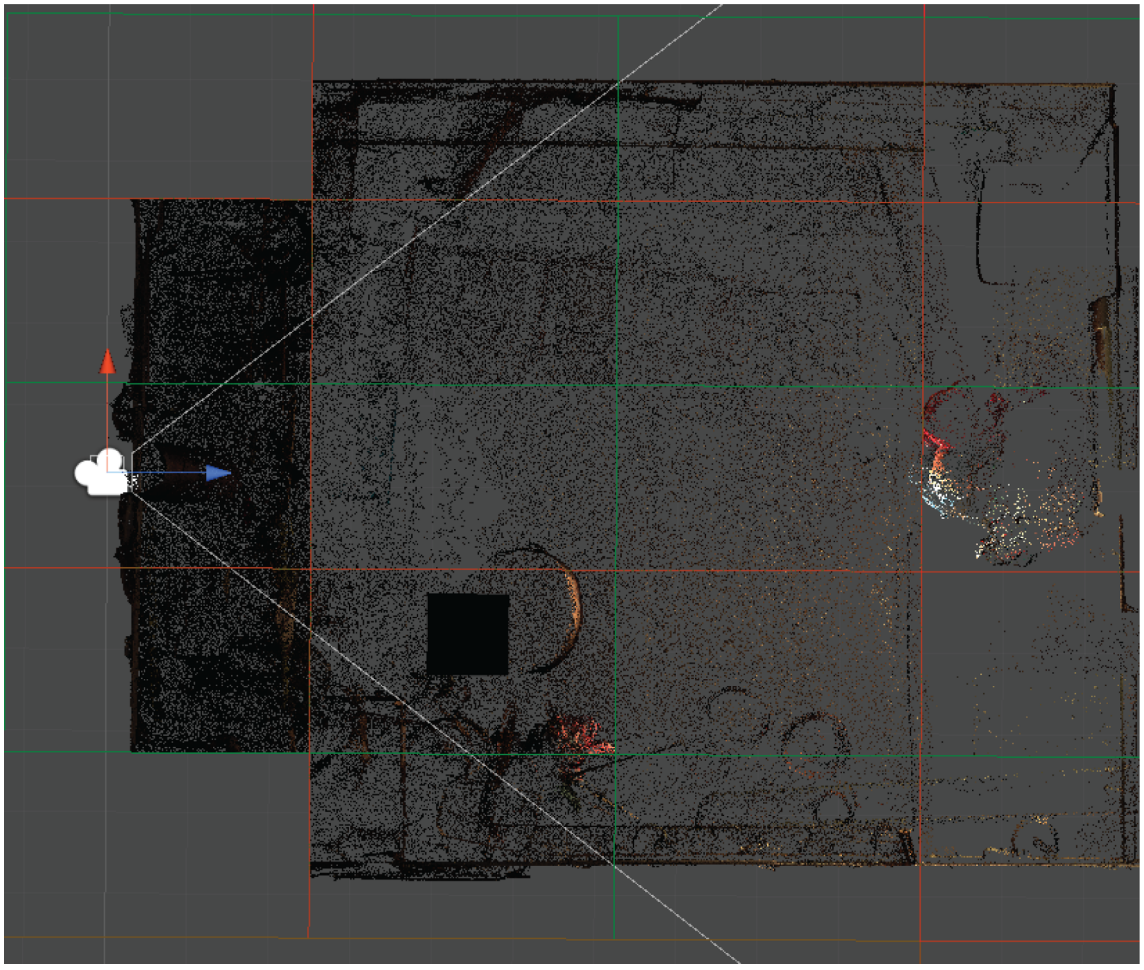

Fig. 2. The impact of the utilized LOD system on point cloud visualization illustrated from the top view. The octree nodes further away from the camera are rendered with a lower point density. The effect has been exaggerated in the figure for illustrative purposes.

3.4.1. Capture node - a dynamic nested sub-octree

To facilitate dynamic manipulation of pre-defined point cloud segments in the octree structure, a dynamic, nested sub-octree solution is utilized, referred to as "capture nodes".

The capture nodes differ from normal octree nodes in a scene by not being static parts of the octree structure; they are rather being linked to the node of the octree in which they currently reside, making them a dynamic part of the octree. This allows the capture nodes to be used to

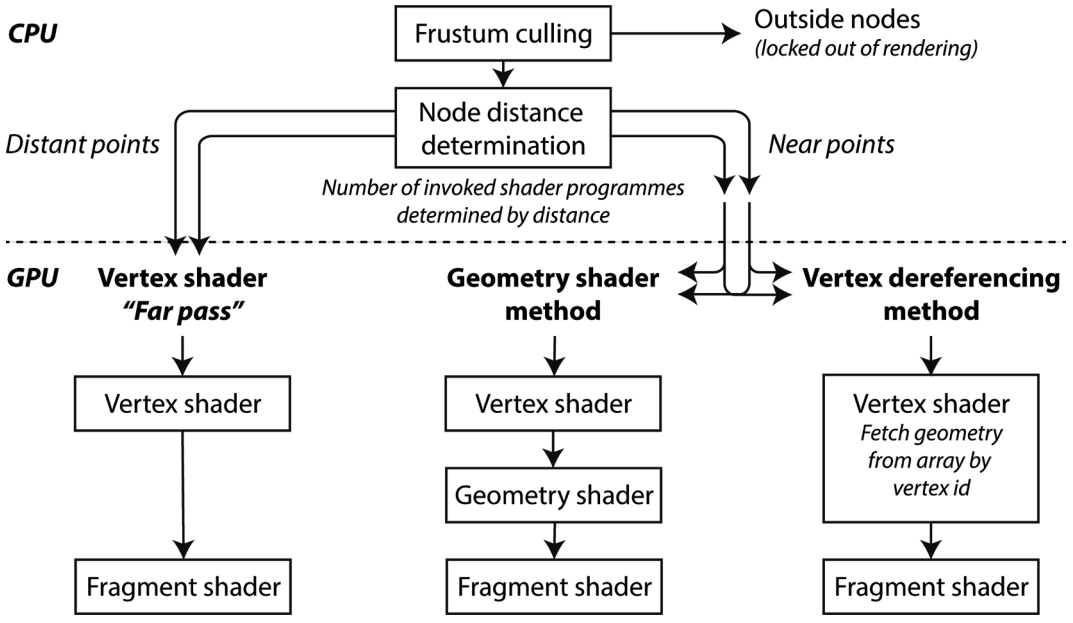

Fig. 3. Summary of rendering pipeline. The two illustrated methods for rendering near points are alternatives.

form separate objects in a point cloud. Additionally, separate point clouds can be loaded into capture nodes, which can contain their own octree structure, if necessary. Capture nodes can be used to clip off or duplicate parts of the octree point cloud in a copy, paste, and cut manner to separate objects from the point cloud.

For applying transformations to point cloud objects in capture nodes, node-specific transformation matrices can be edited and sent to the shader. This can be utilized to move the point cloud object or to bind them with physics simulation, executed in the game engine using mesh or primitive collider objects. This effectively allows using point cloud segments much like mesh models are utilized in game engine application development, whilst maintaining the octree-based occlusion culling in point cloud rendering.

3.4.2. Ray-casting via simplified mesh equivalents

To facilitate ray-casting in point cloud scenes, simplified mesh equivalents of point cloud segments are utilized. These are produced in game engine with a self-implemented Marching Cubes algorithm (original form in Lorensen and Cline, 1987).

For raycasting, a simplified collision mesh is generated from the voxelized space around the point cloud, which is divided into separate meshes according to the octree structure and linked to corresponding nodes. The point of interest is then found when the ray intersects the surface of the collision mesh, after which the queried points can then be searched from the linked node. Therefore, raycasting does not use the octree structure to the full extent but benefits from the frustum culling, as the out-of-frustum colliders will be excluded from the search.

The simple mesh equivalents can also be applied to form colliders that can be applied in interactable objects and physics. (As such, it is not required for point cloud visualization.)

3.4.3. Point editing masks

To facilitate point recoloring and removal, a 32-bit floating point mask is associated with each point and sent to the buffer structure in the GPU. Values above 0 are reserved for RGB values, and the values below define other functions such as removing points. As the removal of the point is done within the graphical pipeline in the GPU, it is separated from the actual point data structures for faster transfers between the CPU and the GPU.

This allows re-coloring and hiding points dynamically in the game engine without this necessitating the reconstruction of the octree structure. Therefore, the point editing functions can be used in real-time.

3.5. Data for the demonstrator application

To test the developed Unity extension in practice, a demonstrator application combining indoor and object point clouds was developed. To facilitate performance comparison, textured indoor mesh models were also produced from the same datasets.

3.5.1. Indoor 3D point cloud acquisition

Two rooms, a storage room and a boiler room, from a basement of a 1960s detached house in Helsinki, Finland, were scanned using Faro Focus S350 TLS (Fig. 4). To produce data sets as occlusion-free as possible, scanning was performed with a low point density, utilizing instead a large number of scan locations (6 for the first room and 9 for the second). For aesthetic reasons, no reference targets were installed in the interiors prior to scanning.

The interiors were photographed from a tripod with a Nikon D800E digital single-lens reflex (DSLR) camera using a Nikkor AF-S 14–24 mm lens with focus and zoom locked at 14 mm (Fig. 2). A total of 65 images were obtained from the first room and 81 from the second room. The NEF (Nikon Electronic File) images were preprocessed using Adobe Lightroom software (version 6.12) and exported as JPEG files. The main purpose of the pre-processing was to adjust the tonal scales of the images to fix possible overexposure and underexposure, as well as to exclude all blurred or otherwise failed images from the data set.

Faro SCENE software (Version 6.0.0.31) was used to process the TLS scans. Point cloud based co-registration methods of the SCENE software were applied: Preliminary registration was performed with the interactive “top view based” method. For refining the registration, an automatic “cloud to cloud” method was used (effectively being an implementation of Iterative Closest Point (ICP) based on Besl and McKay, 1992). Mean scan point tensions (mean of distances between reference pairs) of 1.531 mm and 1.267 mm were obtained, with the percentage of points with mismatch below 4 mm being 87.3% and 83%, respectively, indicating a successful registration and high overlap of scan point clouds.

Reality Capture software (1.0.2.3012) by Capturing Reality was used to produce the final 3D point cloud data (Fig. 5). Both the TLS scan data and the imagery were used to achieve the final colorized 3D point cloud. The prior registration of the scans was maintained, allowing the pre-registered TLS “block” to act as the reference frame of the project, helping to orient the images and provide metric scale for the photogrammetric reconstruction.

3.5.2. Object scanning

For producing interactable objects, a set of 17 individual items were scanned with the Faro Freestyle handheld 3D scanner (Fig. 6). Scanning was performed with the “interpolation” function of the scanner disabled, as it has been reported by the manufacturer to reduce the metric accuracy of the results. Instead, a very slow scanner movement was preferred to produce sufficiently dense point clouds. Faro SCENE (version 6.0.0.31) was used for processing the scans.

For combining several scans from a different orientation, CloudCompare (version 2.8) was used. The point clouds were manually segmented to remove table surface and markers and manually oriented to close proximity. The iterative closest point (ICP) algorithm (based on Besl and McKay, 1992) was used to refine the orientation. Manual orientation was performed in cases where the ICP failed to produce adequate results. The scanned objects varied in size from approx. 80 cm long to less than 5 cm long, resulting in point counts ranging from 1,076,684 points (for the largest object) to 4,659 points (for the smallest object).

3.5.3. Indoor mesh model for performance evaluation

The Reality Capture software was also applied to produce a textured 3D mesh model from one of the interiors. This was done to facilitate the comparison of rendering performance between the point clouds (with implemented tools) and mesh models (utilizing conventional Unity shaders and tools). Same TLS & image datasets were utilized in producing the textured mesh.

The mesh model was produced with game engine application in mind, aiming for sufficiently low polygon counts etc. The resulting indoor mesh model of the boiler room contained 599,920 vertices and a total of four 4096 × 4096 (4k) sized texture files.

4. Results

4.1. Unity point cloud extension

The presented methods were implemented in an extension for the Unity game engine (Version 2018.2.15) consisting of point cloud processing tools (running in the Unity editor) and the rendering and interaction tools, facilitating real-time rendering and interaction. The point cloud preparation and rendering and interaction tools were implemented in C#, whereas the shader programmes were written in the Open GL Shading Language, GLSL. The rendering and interaction tools can be run either in the Unity editor to facilitate interactive point cloud visualization or in stand-alone applications built with Unity. The system thus facilitates first the preparing of the point cloud for game engine use in the Unity editor, secondly for visualizing it in the editor, and finally

Fig. 4. Scanning and photographing the basement interiors.

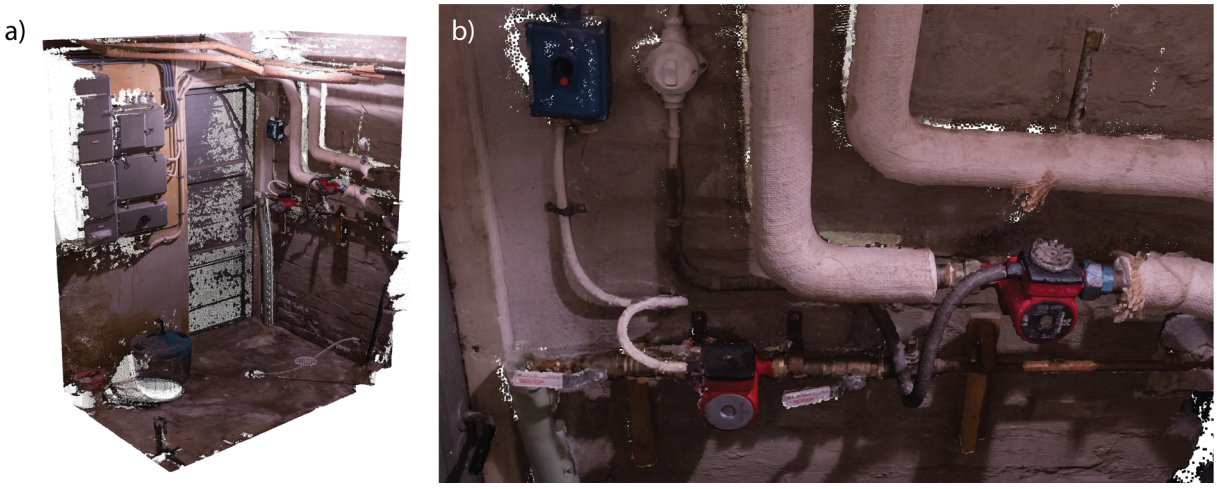

Fig. 5. A single basement interior point cloud, shown with two walls and a roof segmented for illustration (a), and a close-up showing the obtained details (b).

for using Unity to build stand-alone applications that utilize the extension for efficient visualization of the prepared point clouds (Fig. 7).

The point cloud processing tools facilitate the import of point cloud data from an external file, processing it to an octree structure with pre-generated LOD levels and writing the prepared point cloud as a binary file. Visualization of the point cloud is initiated by loading the prepared point cloud from the binary file to the octree. This may be performed either in the editor (to allow point cloud visualization in the Unity editor) or in a stand-alone application produced with Unity (visualizing the point cloud in the application made with Unity). In the following performance experiments, only the performance of stand-alone applications was evaluated.

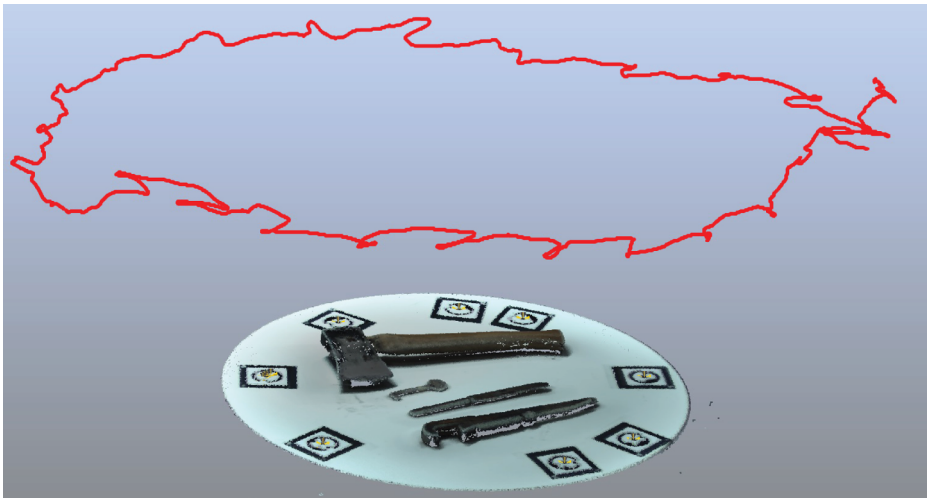

Fig. 6. An individual point cloud from Faro Freestyle, showing a collection of smaller objects being scanned, the markers mounted on the table, and the trajectory of the scanner (red). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

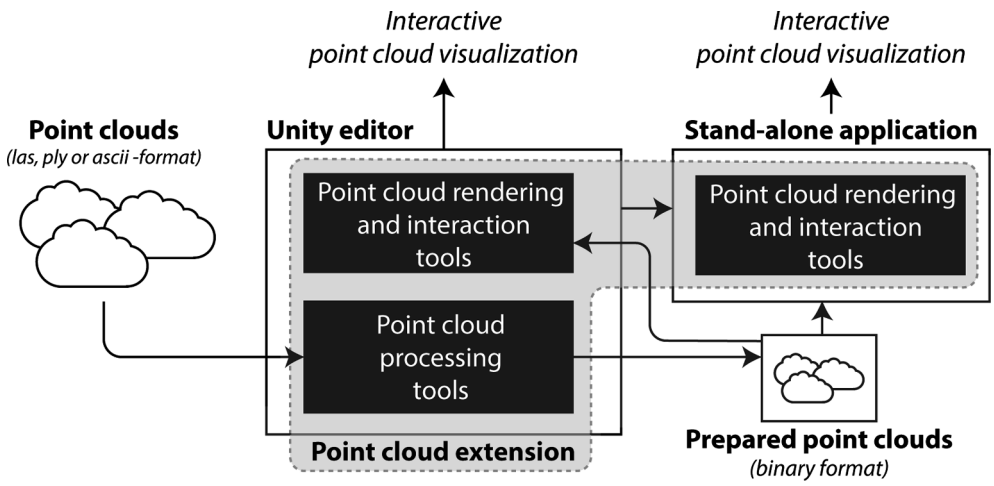

Fig. 7. Extension and tools in the Unity editor and stand-alone applications.

4.1.1. Adjustments and additional optimization methods implemented

In addition to the two described shader methods, multiple different shader programs have been developed for the point cloud rendering and interaction extension for use in different visualization scenarios. These include, for example, a shader that renders distance fog with the point clouds, intended for larger outdoor scenes. As an alternative to the distance-based LOD level selection described, the LOD selection can also be made dependent of attained rendering frame times. This allows the application to determine the LOD with the aim of maintaining a specified rendering performance.

When point data can be assumed to contain a lot of invisible data, an extra culling step can be performed in a vertex shader by comparing the rendering camera's direction to the points normal and then discarding it by placing it outside the view space if the dot product is equal or more than 0.0. This allows in addition to conventional occlusion culling, the culling of the points according to their normal, much like in conventional rendering of polygon meshes. If normal information is not available, normal estimation can also be performed for the point cloud, processing being executed by the GPU. The closest two neighbors for each point are utilized in normal estimation.

In the octree-based rendering used, the last children are responsible for rendering their part of the space: render handling is placed as deep into the tree as possible. This is intended to allow CPU-side frustum culling to stop unnecessary data from being passed further down the rendering pipeline to the GPU. This maintains efficiency and keeps the size of structured buffers to a minimum, to ensure that enough continuous space can be allocated for each buffer and masks can be transferred CPU-GPU with minimal impact to performance. However, this increases the amount of draw calls as the octree depth; and consequently, the number of nodes increases. The management of the octree depth, and therefore the number of nodes, becomes a balancing question. Depending on the depth of the octree, it can be beneficial for rendering performance in some cases to directly address the final depth, i.e., draw all points from the lower level nodes in a single draw call rather than try to limit the number of points drawn by drawing lower levels node by node.

4.2. The demonstration application

The presented extension was used in producing a demonstration application visualizing the test data sets described earlier. The aim of the demonstrator was to showcase the use of interactive point clouds with real-time lighting in a virtual environment. The application was developed in Unity, using the point cloud processing tools of the extension to prepare the test point clouds. The final application was built as a stand-alone for the Windows operating system. The HTC Vive Pro VR headset with the associated handheld controls and external tracking system was used in the demo. The headset supports stereoscopic visualization with a resolution of 1440 × 1600 px per eye, with a maximum refresh rate of 90 hz. The headset uses 1.4x supersampling to compensate for the barrel distortion that results in the required rendering resolution of 2016 × 2240 px per eye.

The demo consists of two basement areas: a boiler room (14,680,146 points + objects) and a storage room (26,005,015 points + objects). Both rooms (Fig. 8) are illuminated with real-time dynamic lighting with an animated movement pattern. The user is able to move from one room to another by placing their controller through the doorway and holding the trigger down. Both rooms have a set of

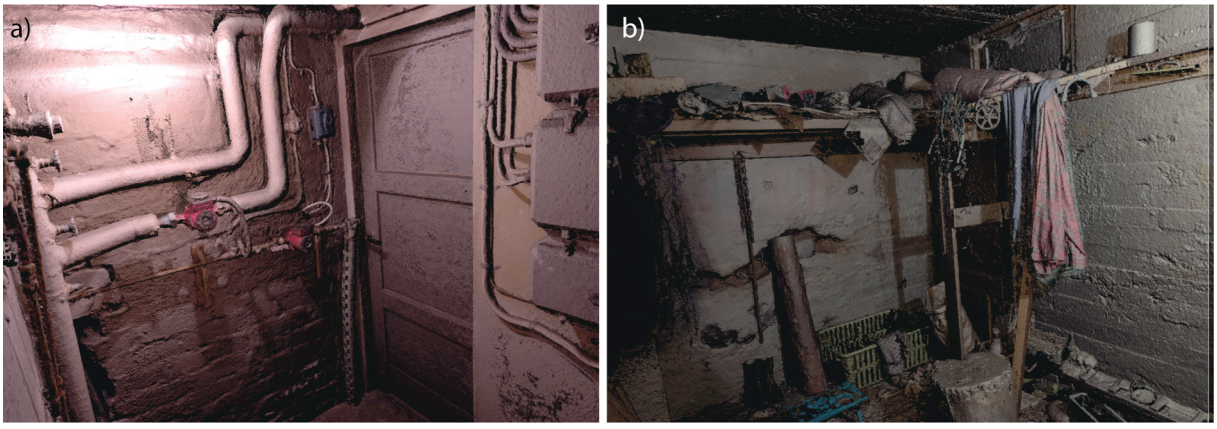

Fig. 8. User exploring the interiors in VR. The scenes are illuminated by dynamic lighting.

separate, interactable point cloud objects with physics simulation. Objects can be grabbed, carried, and thrown around (Video 1). Real-time point cloud editing is done with a brush tool that can be used to recolor or remove points in a point cloud within the user-defined radius, mimicking traditional graphic editors (Fig. 9). Editing functions allow the user to either re-color (Video 2) or erase the points (Video 3). The demo also featured the possibility to resize the points in real time with the Vive controller as well as a measuring tool to calculate the distance between each point.

Video 1.

Video 2.

Video 3.

Fig. 9. (a) User painting the points with selected color, (b) erasing points, and (c) moving objects. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

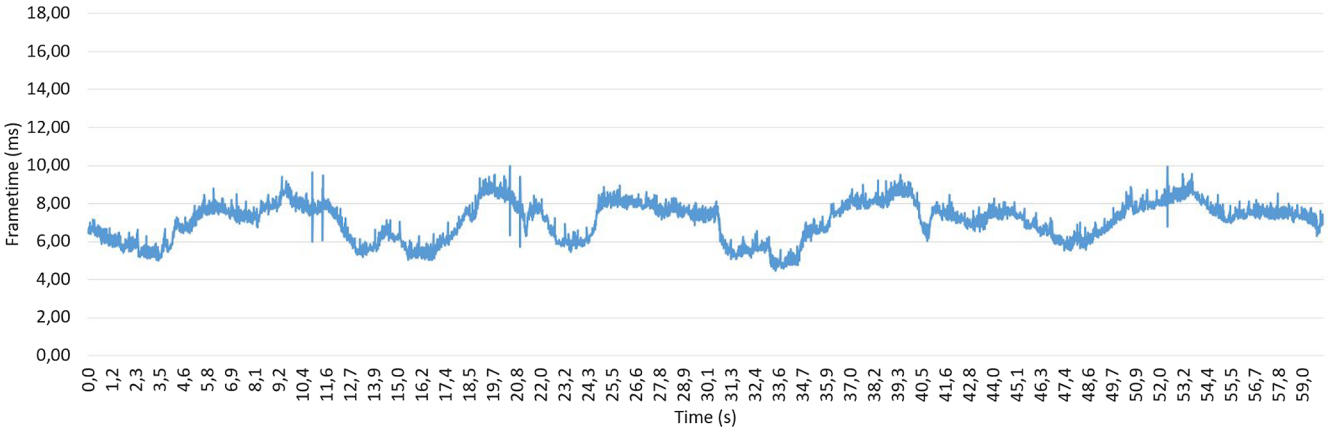

Fig. 10. The frametime (ms) while studying the point cloud with the geometry shader method.

4.2.1. Performance of point cloud rendering

The demonstrator application was evaluated by testing the performance of the produced Unity standalone executable on a desktop PC (Intel Core i7 3.7 GHz CPU; 64 GB RAM; GeForce GTX 1080 Ti GPU). For the performance evaluation, a total of three different versions of the demonstrator were built: a version for each of the shading strategies (vertex dereferencing & geometry shader) and a version utilizing a textured mesh model instead of the point cloud.

The performance evaluation was carried out by launching the application, waiting for the CPU load to stabilize and then studying the environment in the application for a duration of 1 min, logging the rendering frametimes from the Steam VR application (version 1.8.21). To estimate the potential impact of point cloud editing tools on the performance, the geometry shader version of the application was evaluated twice, once with the user only studying the point cloud, and a second time with the user performing point cloud coloring the application. Figs. 10–13 provide the frametimes from all four experiments.

Out of the two point cloud shading methods utilized, the geometry shader method appeared to offer a better performance than the vertex dereferencing method (during the experiment, average frametime of 7.1 vs. 13.1 ms). Most of the frametime fluctuations (Figs. 10–12) are likely explained by the varying position of the user's head and gaze

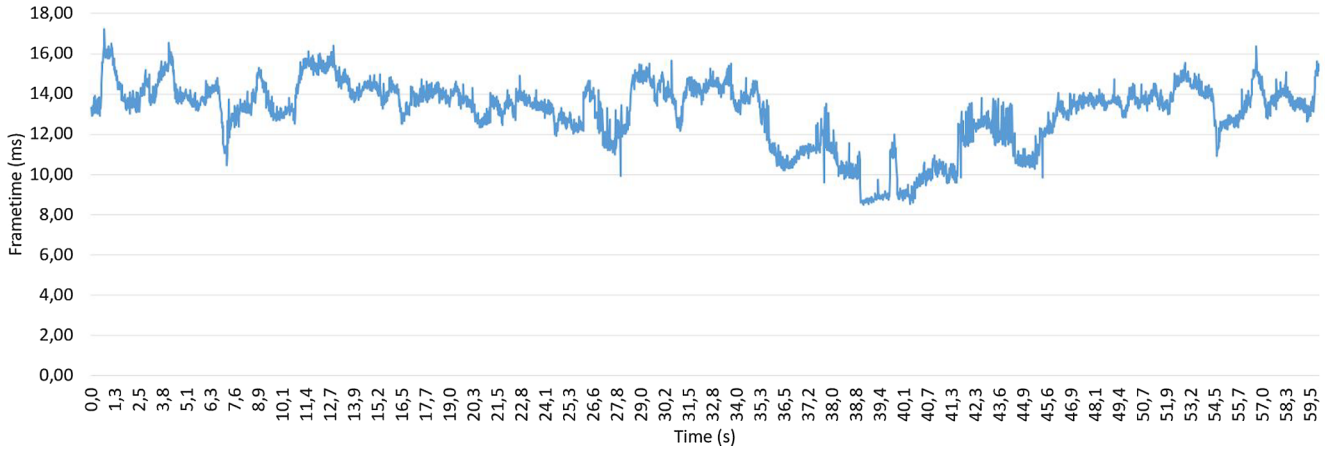

Fig. 11. The frametime (ms) while studying the point cloud with the vertex dereferencing method.

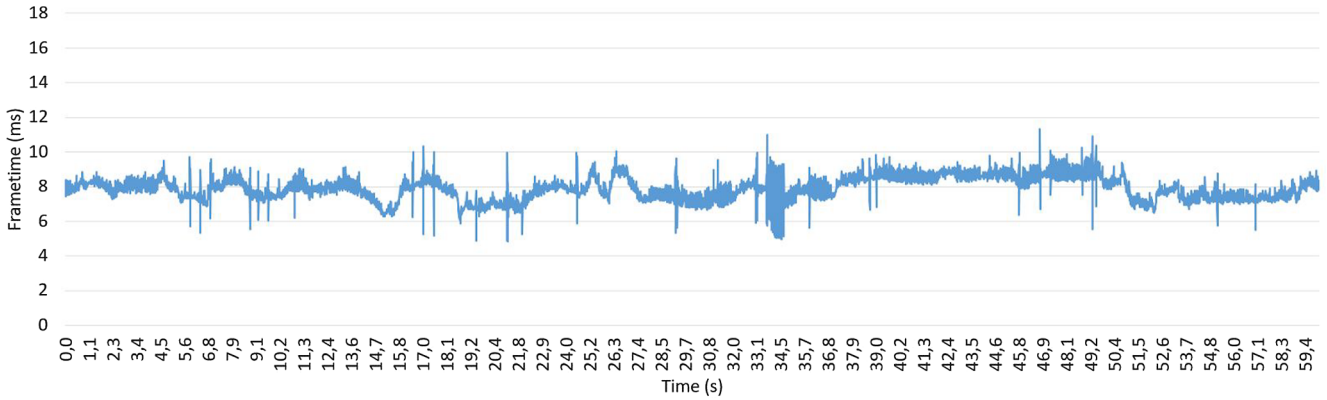

Fig. 12. The frametime (ms) while manually coloring segments of the point cloud with the geometry shader method.

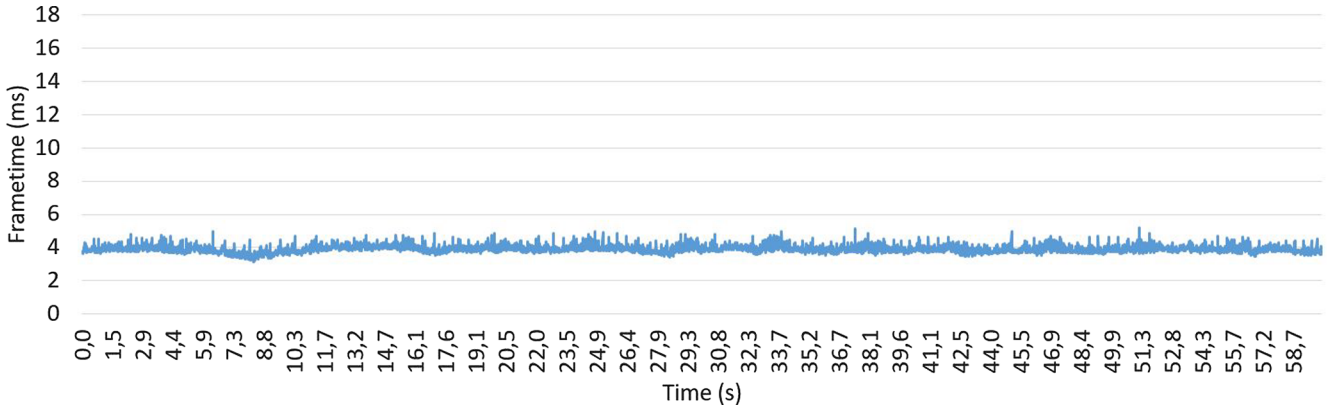

Fig. 13. The frametime (ms) while studying the mesh model.

direction, affecting the number of points rendered. In most cases, the frametimes remained below 15 ms, indicating a frame rate of over 66 FPS. With the geometry shader method, the system was able to attain a frametime below 11 ms, allowing the maximum frame rate of the VR HMD (90 FPS) to be used. The mesh model offered an even higher rendering performance than the point cloud (average frametime of 3.9 ms).

Performing manual coloring operations on the point cloud seemed to inflict a small performance loss with the geometry shader method (during the experiment, average frametime of 7.9 ms). Utilizing the point editing tools also increased the variations of performance. However, the frametimes still remained mostly below 11 ms, permitting fluent VR interaction.

5. Discussion

Concerning the presented methods for point cloud rendering in game engines, the work demonstrates that in real-time rendering, different shader programmes can be combined to limit the use of more computationally intensive shaders to only render a subset of all rendered points. Secondly, via a dynamic, nested octree structure, it is possible to utilize an octree for occlusion culling and data management, but still allow moving sub-objects in the scene. Thirdly, by controlling shader invocation for octree node subsets, the rendering LOD can be detached from the octree level, allowing better control of draw calls in real time rendering. Finally, point editing masks allow re-coloring and hiding of arbitrary segments in large point clouds, in real-time. While many of these approaches have been mentioned in prior literature (Deibe et al., 2019; Santana et al., 2019; Wirth et al., 2019), we have demonstrated that they can be utilized concurrently in a commercial game engine.

Looking at the results obtained with the Unity implementation, it can be concluded that a game engine platform can be applied in building interactive applications using dense colored point clouds. With the presented point cloud rendering and interaction extension, the use of point clouds in a game engine is not restricted to visualization only but may as well include the use of point clouds as interactable objects, attaching physics simulation to them, and editing the point clouds within the game engine application, in runtime.

Compared with existing point cloud extensions of game engines (Point Cloud Viewer and Tools, 2017; Point Cloud Plugin, 2019), and browser based point cloud visualization systems (Discher et al., 2019; Schütz, 2016) similar visualization and editing features are offered by our system. As both of the existing game engine extensions for point clouds operate on game engines compatible with commercial VR systems, we assume they are also applicable for VR visualization. VR is also mentioned in Discher et al. (2019), but they do not indicate whether VR visualization or any VR functionalities for e.g. editing have been implemented. Wirth et al (2019) utilize VR for point labeling, but do not specify the associated point cloud rendering capabilities. As the Potree system (Schütz, 2016) is based on WebGL, we can assume that implementation of it with a WebVR device would be feasible. Utilizing multiple point clouds as objects in a virtual scene is mentioned in Point Cloud Plugin (2019) and Discher et al. (2019), as has also been demonstrated in Potree. However, these do not mention coupling the point cloud objects with game engine physics, demonstrated in our work. Neither have the VR tools for editing point clouds been demonstrated in these existing implementations. These developments bring the point clouds closer to conventional textured mesh models that are typically used as 3D content in game engines, and increase their applicability in immersive visualization. The editing functionalities are a prerequisite for the use of VR in point cloud processing tasks.

In the demonstrator, rendering times below 15 ms were commonly attained for stereoscopic visualization with the HTC Vive Pro VR HMD,

with rendering times below 11 ms reached with the geometry shader method. This indicates that the presented rendering methods are capable of producing a dataflow of over nine megapixels per second, required by the contemporary VR HMD. This was attained on a high-end PC with a powerful GPU, leaving the applicability of the proposed system on low-end computers or mobile systems remaining to be proven. Discher et al (2019) report similar framerates on a PC (56-122 FPS), attained in browser, but with a smaller point count (2–8 M points). Their rendering resolution is not given, but we can assume that the rendering is monoscopic. Bouchiba et al. (2018) likewise report a high rendering performance, but also concerning monoscopic rendering with a 1248 × 768 px resolution.

In experiments, both mesh and point cloud visualization were able to attain a sufficient rendering performance in VR. The frametimes when rendering mesh models were significantly smaller, indicating higher performance. Here, the larger data volumes (599,920 vertices vs. 14,680,146 points) and lack of hardware optimization impose performance limitations on point cloud rendering. In addition, the established tools (e.g., processing software), format support, and resources like asset libraries are largely absent for point clouds. Nevertheless, the direct application of point clouds for visualization holds potential for reducing the costs associated with applications requiring 3D reconstructions of existing environments. The steps required for mesh model production can be avoided in point cloud visualization. This also simplifies data integration: in the presented case, point clouds from several different methods were effortlessly combined. The feasibility of interactive point cloud visualization methods can be expected to further improve along with the development of graphics hardware and software (e.g., increases in rendering performance or new technologies like real-time ray tracing).

Looking at the data acquisition process, several observations concerning the use and production of point clouds can be made. First, when producing point clouds intended for aesthetic rather than metric applications, the perceived visual quality of the point clouds should also be considered. In tests carried out with synthetic data, Zhang et al. (2014) found an almost direct correlation between point density and human perception. In the same study, color noise had a smaller influence than spatial noise on perceived quality (Zhang et al., 2014). This would indicate that visualization point clouds should preferably be as dense and noise-free as possible, and that measuring for visualization purposes places new demands on the production of point cloud data.

Secondly, point clouds offer a data format usable for combining 3D measuring data, regardless of the sensor technology used. This has also been acknowledged by other authors (e.g., Persad & Armenakis, 2017). In addition to scale, rotation, and translation, differences in overlap, point distribution, density, and occlusions may complicate fusing 3D point clouds, as noted by Persad and Armenakis (2017). These notions also apply when using several measuring systems for producing visualization data for the same project, as encountered in our work. Similar point cloud densities should be attained if the point clouds are to be used together.

Thirdly, the use of point clouds in interactive applications raises the importance of object detection and separation from a point cloud to produce a semantic structure. Point cloud segmentation (e.g., Dong et al., 2018) is a prerequisite for this; but for producing interactive applications, some level of semantic interpretation (e.g., Shi et al., 2019) is required. Further, if the objects in a point cloud are to be moved interactively, hole-filling algorithms for point cloud data sets are also required (e.g., Barazzetti, 2018).

From an application perspective, the availability of low-cost sensors (e.g., Henry et al., 2012) and computational methods (e.g., Gonzalez-Aguilera et al., 2018) capable of efficiently producing 3D point cloud data of real-life objects and scenes could enable new applications for non-expert users, attained with game engine point cloud rendering and interaction. These potential applications include virtual configuration of furniture (Oh et al., 2008), virtual training (Gavish et al., 2015), and therapy (Reger and Gahm, 2008), to offer some examples. Many of the current examples of such applications are produced using manually produced game engine models. Compared with manual modeling, the use of point clouds obtained with affordable methods would significantly reduce the costs of producing virtual environments for many use cases.

In addition to the applications mentioned above, the use of game-engine-based point cloud visualization in VR may find a use in geomatics. As such, dense point clouds may operate as a basis for visualization in applications like city planning and architecture. In this case, producing an immersive, near-photorealistic visualization directly from the point clouds allows a significant increase in efficiency when compared with traditional 3D modeling. This is especially relevant when discussing highly efficient 3D mapping methods such as MLS or airborne laser scanning. In addition, improving the applicability of point clouds is an enabling component in several future visions involving the increased application of point cloud data sets (Nebiker et al., 2010; Virtanen et al., 2017). Further, VR environments could also be applied to point cloud editing to offer more efficient ways to interact with dense terrestrial data sets. These applications could include manual classification of dense point clouds for teaching classification algorithms, cleaning of point cloud data sets from clutter originating from pedestrians and vehicles, and object separation from indoor point clouds, to offer a few examples.

6. Conclusion

In the presented work, an extension facilitating the rendering and interaction with point clouds, and the required preprocessing, was implemented for the Unity game engine. The developed extension contains the tools to prepare the point cloud prior to game-engine use and the rendering system to facilitate real-time rendering of dense point clouds in Unity. The implemented interaction functions allow the use of conventional game engine tools, such as rigid body physics, with point cloud objects. In addition, custom tools for interactively painting colors and removing points were implemented. The implementation of physics and raycasting rely on mesh counterparts coupled with the point cloud components, which facilitate interaction while the point cloud is used for visualization. To test the developed extension, a demonstrator application was produced and its performance evaluated in rendering a set of point clouds obtained with TLS, photogrammetry, and 3D object scanning. The performance tests found the system to perform with a sufficient frame rate for using VR HMDs for visualization, even with point counts in excess of 10 million points.

Several potential benefits have been perceived for direct visualization of dense point clouds. First, the points are often more efficient for initial data processing: direct application of point cloud data for visualization omits the need for a separate modeling process for visualization. In addition, the visualized point cloud is theoretically closer to the original scan data than a mesh model constructed via a separate process. Furthermore, several of the modeling workflows used lead to the omittance of small details (e.g., from building facades), which could be avoided by direct application of point clouds. Finally, managing the levels of detail (LOD) when visualizing a 3D scene is easier with points, as it can be accomplished by displaying more or less points with varying volume according to the requested LOD.

Our development efforts allow the use of point clouds beyond capabilities offered by current existing software functions. By realizing the real-time lighting and interaction functions, the work has the potential of improving the user experience associated with point cloud visualization, expanding the applicability of dense point clouds to new realms, and offering new tools for geomatics practitioners working with dense point cloud data sets.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

This research project was supported by the Academy of Finland, the Centre of Excellence in Laser Scanning Research (CoE-LaSR) (No. 272195, 307362). The Strategic Research Council at the Academy of Finland is acknowledged for financial support for the project, “Competence-Based Growth Through Integrated Disruptive Technologies of 3D Digitalization, Robotics, Geospatial Information and Image Processing/Computing – Point Cloud Ecosystem (No. 293389, 314312)” and the Business Finland for the innovation project “VARPU” (7031/31/2016). The support from the European Social Fund for the project S21272 is acknowledged.

References

Actueel Hoogtebestand Nederland, 2019, < http://www.ahn.nl/index.html > (accessed on14.3.2019).
Alatalo, T., Koskela, T., Pouke, M., Alavesa, P., & Ojala, T. 2016. VirtualOulu: collaborative, immersive and extensible 3D city model on the web. In: Proc. 21st International Conference on Web3D Technology. ACM, Anaheim, CA, 22-24 July, pp. 95–103. https://doi.org/10.1145/2945292.2945305.
Barazzetti, L., 2018. Point cloud occlusion recovery with shallow feedforward neural networks. Adv. Eng. Inf. 38, 605–619. https://doi.org/10.1016/j.aei.2018.09.007.
Bassier, M., Van Genechten, B., Vergauwen, M., 2019. Classification of sensor independent point cloud data of building objects using random forests. J. Build. Eng. 21, 468–477. https://doi.org/10.1016/j.jobe.2018.04.027.
Besl, P.J., McKay, N.D., 1992. Method for registration of 3-D shapes. Proc. Sensor Fusion IV: Control Paradigms and Data Structures. International Society for Optics and Photonics, Boston, MA. pp. 586–607. https://doi.org/10.1117/12.57955.
Bouchiba, H., Deschaud, J.E., Goulette, F., 2018. Raw point cloud deferred shading through screen space pyramidal operators.
Brown, R.A., 2015. Building k-d Tree in O (knlog n) Time. J. Comput. Graph. Tech. 4 (1).
CesiumJS, retrieved from: < https://cesium.com/cesiumjs/ > (Accessed on 10.12.2019).
Chen, Y., Wang, S., Li, J., Ma, L., Wu, R., Luo, Z., Wang, C., 2019. Rapid urban roadside tree inventory using a mobile laser scanning system. IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens. 12 (9), 3690–3700.
Crassin, C., 2011. GigaVoxels: A Voxel-Based Rendering Pipeline For Efficient Exploration Of Large And Detailed Scenes. PhD thesis. Grenoble University.
Cura, R., Perret, J., Paparoditis, N., 2017. A scalable and multi-purpose point cloud server (PCS) for easier and faster point cloud data management and processing. ISPRS J. Photogramm. Remote Sens. 127, 39–56.
Deibe, D., Amor, M., Doallo, R., 2019. Supporting multi-resolution out-of-core rendering of massive LiDAR point clouds through non-redundant data structures. Int. J. Geograph. Inform. Sci. 33 (3), 593–617. https://doi.org/10.1080/13658816.2018.1549734.
De La Calle, M., Gómez-Deck, D., Koehler, O., Pulido, F., 2011. Point cloud visualization in an open source 3d glob3. International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume XXXVIII-5/W16, 2011 ISPRS Trento 2011 Workshop, 2-4 March 2011, Trento, Italy.
Diakité, A.A., Zlatanova, S., 2016. First experiments with the tango tablet for indoor scanning. ISPRS Annals Photogram., Remote Sens. Spatial Inform. Sci. 3.
Discher, S., Richter, R., Döllner, J., 2019. Concepts and techniques for web-based visualization and processing of massive 3D point clouds with semantics. Graph. Models, 101036.
Dong, Z., Yang, B., Hu, P., Scherer, S., 2018. An efficient global energy optimization approach for robust 3D plane segmentation of point clouds. ISPRS J. Photogramm. Remote Sens. 137, 112–133. https://doi.org/10.1016/j.isprsjprs.2018.01.013.
Donghui, C., Guanfa, L., Wensheng, Z., Qiyuan, L., Shuping, B., Xiaokang, L., 2017. Virtual reality technology applied in digitalization of cultural heritage. Clust. Comput. 1–12. https://doi.org/10.1007/s10586-017-1071-5.
Dorninger, P., Pfeifer, N., 2008. A comprehensive automated 3D approach for building extraction, reconstruction, and regularization from airborne laser scanning point clouds. Sensors 8 (11), 7323–7343. https://doi.org/10.3390/s8117323.
Elseberg, J., Borrmann, D., Nüchter, A., 2012. One billion points in the cloud – an octree for efficient processing of 3d laser scans. ISPRS J. Photogramm. Remote Sens. https://doi.org/10.1016/j.isprsjprs.2012.10.004.
El-Mahgary, S., Virtanen, J.P., Hyyppä, H., 2020. A simple semantic-based data storage layout for querying point clouds. ISPRS Int. J. Geo-Inf. 9 (2), 72.
Fraiss, M. Rendering Large Point Clouds in Unity. Bachelor's thesis, TU Wien.
Gao, Z., Nocera, L., Neumann, U., 2012. Visually-complete aerial LiDAR point cloud rendering. In: Proceedings of the 20th international conference on advances in geographic information systems. ACM, pp. 289–298.
Gavish, N., Gutiérrez, T., Webel, S., Rodríguez, J., Peveri, M., Bockholt, U., Tecchia, F., 2015. Evaluating virtual reality and augmented reality training for industrial maintenance and assembly tasks. Interact. Learn. Environ. 23 (6), 778–798. https://doi.org/10.1080/10494820.2013.815221.
Geoportal Thüringen, 2019, < http://www.geoportal-th.de/de-de/downloadbereiche/downloadoffenegeodatenth%C3%BCringen/downloadh%C3%B6hendaten.aspx > (accessed 14.3.2019).
Gonzalez-Aguilera, D., López-Fernández, L., Rodriguez-Gonzalvez, P., Hernandez-Lopez, D., Guerrero, D., Remondino, F., Menna, F., Nocerino, E., Toschi, I., Ballabeni, A., Gaiani, M., 2018. GRAPHOS–open-source software for photogrammetric applications. Photogram. Rec. 33 (161), 11–29. https://doi.org/10.1111/phor.12231.
Goswami, P., Erol, F., Mukhi, R., Pajarola, R., Gobbetti, E., 2013. An efficient multi-resolution framework for high quality interactive rendering of massive point clouds using multi-way kd-trees. Visual Comput. 29 (1), 69–83. https://doi.org/10.1007/s00371-012-0675-2.
Haala, N., Kada, M., 2010. An update on automatic 3D building reconstruction. ISPRS J. Photogramm. Remote Sens. 65 (6), 570–580. https://doi.org/10.1016/j.isprsjprs.2010.09.006.
Henry, P., Krainin, M., Herbst, E., Ren, X., Fox, D., 2012. RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments. Int. J. Robot. Res. 31 (5), 647–663. https://doi.org/10.1177/0278364911434148.
Hofer, H., Seitner, F., Gelautz, M., 2018, December. An End-to-End System for Real-Time Dynamic Point Cloud Visualization. In: 2018 International Conference on 3D Immersion (IC3D), pp. 1–8. IEEE. https://doi.org/10.1109/IC3D.2018.8657915.
Hornung, A., Wurm, K.M., Bennewitz, M., Stachniss, C., Burgard, W., 2013. OctoMap: an efficient probabilistic 3D mapping framework based on octrees. Auton. Robots 34 (3), 189–206. https://doi.org/10.1007/s10514-012-9321-0.
Hyyppä, J., Virtanen, J.P., Jaakkola, A., Yu, X., Hyyppä, H., Liang, X., 2017. Feasibility of Google Tango and Kinect for Crowdsourcing Forestry Information. Forests 9 (1), 6. https://doi.org/10.3390/f9010006.
Indraprastha, A., Shinozaki, M., 2009. The investigation on using Unity3D game engine in urban design study. J. ICT Res. Appl., vol. 3, 1, pp. 1–18. http://dx.doi.org/10.5614%2Fitbj.ict.2009.3.1.1.
Jaakkola, A., Hyyppä, J., Kukko, A., Yu, X., Kaartinen, H., Lehtomäki, M., Lin, Y., 2010. A low-cost multi-sensoral mobile mapping system and its feasibility for tree measurements. ISPRS J. Photogramm. Remote Sens. 65 (6), 514–522. https://doi.org/10.1016/j.isprsjprs.2010.08.002.
Jamei, E., Mortimer, M., Seyedmahmoudian, M., Horan, B., Stojcevski, A., 2017. Investigating the role of virtual reality in planning for sustainable smart cities. Sustainability 9 (11), 2006. https://doi.org/10.3390/su9112006.
Julin, A., Jaalama, K., Virtanen, J.P., Maksimainen, M., Kurkela, M., Hyyppä, J., Hyyppä, H., 2019. Automated multi-sensor 3D reconstruction for the Web. ISPRS Int. J. Geo-Inf. 8 (5), 221. https://doi.org/10.3390/ijgi8050221.
Karabassi, E.A., Papaioannou, G., Theoharis, T., 1999. A fast depth-buffer-based voxelization algorithm. J. Graph. Tools 4 (4), 5–10. https://doi.org/10.1080/10867651.1999.10487510.
Karjalainen, M., Kankare, V., Vastaranta, M., Holopainen, M., Hyyppä, J., 2012. Prediction of plot-level forest variables using TerraSAR-X stereo SAR data. Remote Sens. Environ. 117, 338–347. https://doi.org/10.1016/j.rse.2011.10.008.
Kaushik, M., Nagwanshi, K.N., Sharma, L.K., 2012. A overview of point-based rendering techniques. Int. J. Comput. Trends Technol. V3 (1), 19–24.
Kharroubi, A., Hajji, R., Billen, R., Poux, F., 2019. Classification and integration of massive 3d points clouds in a virtual reality (VR) environment. Int. Arch. Photogram., Remote Sens. Spatial Inform. Sci. 42, 165–171.
Kobbelt, L., Botsch, M., 2004. A survey of point-based techniques in computer graphics. Comput. Graph. 28 (6), 801–814.
Kovač, B., Žalik, B., 2010. Visualization of LIDAR datasets using point-based rendering technique. Comput. Geosci. 36 (11), 1443–1450.
Kuder, M., Žalik, B., 2012. Web-based LiDAR visualization with point-based rendering. In: 2011 Seventh international conference on signal image technology & internet-based systems. IEEE, pp. 38–45.
Kukko, A., Kaijaluoto, R., Kaartinen, H., Lehtola, V.V., Jaakkola, A., Hyyppä, J., 2017. Graph SLAM correction for single scanner MLS forest data under boreal forest canopy. ISPRS J. Photogramm. Remote Sens. 132, 199–209. https://doi.org/10.1016/j.isprsjprs.2017.09.006.
Kuhn, A., Mayer, H., 2015. Incremental division of very large point clouds for scalable 3d surface reconstruction. In: Proceedings of the IEEE international conference on computer vision workshops, pp. 10–18.
Lachat, E., Landes, T., Grussenmeyer, P., 2017. Performance investigation of a handheld 3D scanner to define good practices for small artefact 3D modeling. Int. Arch. Photogram., Remote Sens. Spatial Inform. Sci. 42.
Labrie-Larrivée, F., Laurendeau, D., Lalonde, J.F., 2016. Depth texture synthesis for realistic architectural modeling. In: 13th Conference on Computer and Robot Vision. IEEE, Victoria, Canada, 1-3 June, pp. 61–68.
Lehtola, V.V., Kurkela, M., Hyyppä, H., 2014. Automated image-based reconstruction of building interiors–a case study. Photogramm. J. Finland 24 (1), 1–13.
Lerma, J.L., Navarro, S., Cabrelles, M., Villaverde, V., 2010. Terrestrial laser scanning and close range photogrammetry for 3D archaeological documentation: the Upper Palaeolithic Cave of Parpalló as a case study. J. Archaeol. Sci. 37 (3), 499–507.
Li, J., Yang, B., Cong, Y., Cao, L., Fu, X., Dong, Z., 2019. 3D forest mapping using a low-cost UAV laser scanning system: investigation and comparison. Remote Sens. 11 (6), 717.
Lorensen, W.E., Cline, H.C., 1987. Marching cubes: A high resolution 3D surface construction algorithm. In: Proc. 14th Ann. Conf. Computer Graphics and Interactive Techniques (SIGGRAPH 87), ACM, Anaheim, CA, 27-31 July, pp. 163–169.
Maas, H.G., Vosselman, G., 1999. Two algorithms for extracting building models from raw laser altimetry data. ISPRS J. Photogramm. Remote Sens. 54 (2–3), 153–163.

https://doi.org/10.1016/S0924-2716(99)00004-0.
Marroquim, R., Kraus, M., Cavalcanti, P.R., 2008. Efficient image reconstruction for point-based and line-based rendering. Comput. Graph. 32 (2), 189–203.
Martinez-Rubi, O., Verhoeven, S., Van Meersbergen, M., Van Oosterom, P., GonÁalves, R., Tijssen, T., 2015. Taming the beast: Free and open-source massive point cloud web visualization. In proc. Capturing Reality Forum, The Survey Association, Salzburg, Austria, 23-25 November.
Matikainen, L., Karila, K., Hyyppä, J., Litkey, P., Puttonen, E., Ahokas, E., 2017. Object-based analysis of multispectral airborne laser scanner data for land cover classification and map updating. ISPRS J. Photogramm. Remote Sens. 128, 298–313. https://doi.org/10.1016/j.isprsjprs.2017.04.005.
Micheletti, N., Chandler, J.H., Lane, S.N., 2015. Investigating the geomorphological potential of freely available and accessible structure-from-motion photogrammetry using a smartphone. Earth Surf. Proc. Land. 40 (4), 473–486. https://doi.org/10.1002/esp.3648.
National Land Survey of Finland, 2019, < https://tiedostopalvelu.maanmittauslaitos.fi/tp/kartta?lang=en > (accessed on 14.3.2019).
Nebiker, S., Cavegn, S., Loesch, B., 2015. Cloud-Based geospatial 3D image spaces—a powerful urban model for the smart city. ISPRS Int. J. Geo-Inf. 4 (4), 2267–2291. https://doi.org/10.3390/ijgi4042267.
Nebiker, S., Bleisch, S., Christen, M., 2010. Rich point clouds in virtual globes–a new paradigm in city modeling? Comput. Environ. Urban Syst. 34 (6), 508–517. https://doi.org/10.1016/j.compenvurbsys.2010.05.002.
Nguyen, M.T., Nguyen, H.K., Vo-Lam, K.D., Nguyen, X.G., Tran, M.T., 2016. Applying virtual reality in city planning. In: International Conference on Virtual, Augmented and Mixed Reality. Springer International Publishing, pp. 724–735.
Nocerino, E., Menna, F., Remondino, F., Toschi, I., Rodríguez-Gonzálvez, P., 2017. June. Investigation of indoor and outdoor performance of two portable mobile mapping systems. In: Proc. Videometrics, Range Imaging, and Applications XIV. International Society for Optics and Photonics. (Vol. 10332, p. 103320I).
Nüchter, Andreas, Lingemann, Kai, Hertzberg, Joachim, Surmann, Hartmut, 2007. 6D SLAM—3D mapping outdoor environments. J. Field Robot. 24 (8–9), 699–722. https://doi.org/10.1002/rob.20209.
OGC, 2019. 3D Tiles Specification 1.0. Retrieved from: < http://docs.opengeospatial.org/is/18-053r2/18-053r2.html > (Accessed on 10.12.2019).
Oh, H., Yoon, S.Y., Shyu, C.R., 2008. How can virtual reality reshape furniture retailing?. Cloth. Text. Res. J., vol. 26, 2, pp. 143–163. https://doi.org/10.1177%2F0887302X08314789.
Otepka, J., Ghuffar, S., Waldhauser, C., Hochreiter, R., Pfeifer, N., 2013. Georeferenced point clouds: a survey of features and point cloud management. ISPRS Int. J. Geoinf. 2, 1038–1065. https://doi.org/10.3390/ijgi2041038.
Persad, R.A., Armenakis, C., 2017. Automatic co-registration of 3D multi-sensor point clouds. ISPRS J. Photogramm. Remote Sens. 130, 162–186. https://doi.org/10.1016/j.isprsjprs.2017.05.014.
Point Cloud Viewer and Tools, < https://www.assetstore.unity3d.com/en/#!/content/16019 > (accessed on 1.4.2019).
Point cloud plugin, < https://pointcloudplugin.com/ > (accessed on 17.12.2019).
Poux, F. and Billen, R., 2019a. A Smart Point Cloud Infrastructure for intelligent environments. Laser scanning: an emerging technology in structural engineering.
Poux, F., Billen, R., 2019. Voxel-based 3D point cloud semantic segmentation: unsupervised geometric and relationship featuring vs deep learning methods. ISPRS Int. J. Geo-Inf. 8 (5), 213.
Poux, F., Neuville, R., Hallot, P., Billen, R., 2016. Smart point cloud: Definition and remaining challenges. ISPRS Annals Photogram., Remote Sens. Spatial Inform. Sci. 4 (W1), 119–127.
Preiner, R., Jeschke, S., Wimmer, M., 2012. Auto splats: dynamic point cloud visualization on the GPU. EGPGV 139–148.
Reardon, S., 2012. A digital ark, come fire or flood. New Sci. 216 (2890), 22–23. https://doi.org/10.1016/S0262-4079(12)62875-9.
Reger, G.M., Gahm, G.A., 2008. Virtual reality exposure therapy for active duty soldiers. J. Clin. Psychol. 64 (8), 940–946. https://doi.org/10.1002/jclp.20512.
Richter, R., Discher, S., Döllner, J., 2015. Out-of-core visualization of classified 3d point clouds. In: 3D Geoinformation Science. Springer International Publishing, pp. 227–242.
Rua, H., Alvito, P., 2011. Living the past: 3D models, virtual reality and game engines as tools for supporting archaeology and the reconstruction of cultural heritage–the case-study of the Roman villa of Casal de Freiria. J. Archaeol. Sci. 38 (12), 3296–3308. https://doi.org/10.1016/j.jas.2011.07.015.
Rusinkiewicz, S., Levoy, M., 2000. QSplat: a multiresolution point rendering system for large meshes. In: Proc. 27th annual conference on Computer graphics and interactive techniques, pp. 343–352.
Saarinen, N., Vastaranta, M., Näsi, R., Rosnell, T., Hakala, T., Honkavaara, E., Wulder, M.A., Luoma, V., Tommaselli, A.M., Imai, N.N., Ribeiro, E.A., 2018. Assessing biodiversity in boreal forests with UAV-based photogrammetric point clouds and hyperspectral imaging. Remote Sens. 10 (2), 338. https://doi.org/10.3390/rs10020338.
Sainz, M., Pajarola, R., 2004. Point-based rendering techniques. Comput. Graph. 28 (6), 869–879. https://doi.org/10.1016/j.cag.2004.08.014.
Santana, J.M., Trujillo, A., Ortega, S., 2019. Visualization of large point cloud in unity. Eurographics (Posters) 23–24.
Scheiblauer, C., Wimmer, M., 2011. Out-of-core selection and editing of huge point clouds. Comput. Graph. 35 (2), 342–351.
Schütz, M., 2016. Potree: Rendering large point clouds in web browsers. Technische Universität Wien, Wiedeń.
Schütz, M., Wimmer, M., 2015. High-quality point-based rendering using fast single-pass interpolation. In: 2015 Digital Heritage, Vol. 1, pp. 369–372. IEEE.
Schütz, Markus, Krösl, Katharina, Wimmer, Michael, 2019. Real-time continuous level of detail rendering of point clouds. 2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR). IEEE, pp. 103–110.
Scottish Remote Sensing Portal, 2019, < https://remotesensingdata.gov.scot/ > (accessed on 14.3.2019).
Shi, W., Ahmed, W., Li, N., Fan, W., Xiang, H., Wang, M., 2019. Semantic geometric modelling of unstructured indoor point cloud. ISPRS Int. J. Geo-Inf. 8 (1), 9. https://doi.org/10.3390/ijgi8010009.
Szeliski, R., 2011. Computer Vision: Algorithms and Applications. Springer, London.
Toschi, I., Ramos, M.M., Nocerino, E., Menna, F., Remondino, F., Moe, K., Poli, D., Legat, K., Fassi, F., 2017. Oblique photogrammetry supporting 3D urban reconstruction of complex scenarios. Int. Arch. Photogram., Remote Sens. Spatial Inform. Sci. 42.
Tredinnick, R., Broecker, M., Ponto, K., 2015. Experiencing interior environments: New approaches for the immersive display of large-scale point cloud data. Proc. IEEE Virtual Reality (VR) 297–298.
Tschirschwitz, F., Büyüksalih, G., Kersten, T.P., Kan, T., Enc, G., Baskaraca, P., 2019. Virtualising an ottoman fortress–laser scanning and 3d modelling for the development of an interactive, immersive virtual reality application. Int. Arch. Photogram., Remote Sens. Spatial Inform. Sci., vol. 42, 2/W9.
Vaaja, M., Hyyppä, J., Kukko, A., Kaartinen, H., Hyyppä, H., Alho, P., 2011. Mapping topography changes and elevation accuracies using a mobile laser scanner. Remote Sens. 3 (3), 587–600. https://doi.org/10.3390/rs3030587.
van Oosterom, P., Martinez-Rubi, O., Ivanova, M., Horhammer, M., Geringer, D., Ravada, S., Tijssen, T., Kodde, M., Gonalves, R., 2015. Massive point cloud data management: design, implementation and execution of a point cloud benchmark. Comput. Graph. 49, 92–125. https://doi.org/10.1016/j.cag.2015.01.007.
Vincke, S., Hernandez, R.D.L., Bassier, M., Vergauwen, M., 2019. Immersive visualisation of construction site point cloud data, meshes and BIM Models in a VR environment using a gaming engine. Int. Arch. Photogram., Remote Sens. Spatial Inform. Sci.-ISPRS Arch. 42, 77–83.
Virtanen, J.P., Hyyppä, H., Kämäräinen, A., Hollström, T., Vastaranta, M., Hyyppä, J., 2015. Intelligent open data 3D maps in a collaborative virtual world. ISPRS Int. J. Geo-Inf. 4 (2), 837–857. https://doi.org/10.3390/ijgi4020837.
Virtanen, J.P., Kukko, A., Kaartinen, H., Jaakkola, A., Turppa, T., Hyyppä, H., Hyyppä, J., 2017. Nationwide point cloud—the future topographic core data. ISPRS Int. J. Geo-Inf. 6 (8), 243. https://doi.org/10.3390/ijgi6080243.
Vosselman, G., Maas, H.G., 2010. Airborne and Terrestrial Laser Scanning. CRC Press.
Wang, L., Xu, Y., Li, Y., 2017. Aerial LIDAR point cloud voxelization with its 3D ground filtering application. Photogramm. Eng. Remote Sens. 83 (2), 95–107.
Weinmann, M., 2016. Reconstruction and analysis of 3D scenes: from irregularly distributed 3d points to object classes. Springer, 1st ed. 2016 edition, 233 pages.
Wirth, F., Quchl, J., Ota, J. and Stiller, C., 2019, June. PointAtMe: Efficient 3D Point Cloud Labeling in Virtual Reality. In: 2019 IEEE Intelligent Vehicles Symposium (IV). IEEE. https://doi.org/10.1109/IVS.2019.8814115, pp. 1693–1698.
Xu, C., Fréchet, S., Laurendeau, D., Mirallès, F., 2015. Out-of-core surface reconstruction from large point sets for infrastructure inspection. In: Proc. IEEE 12th Conference on Computer and Robot Vision (CRV), pp. 313–319.
Ye, M., Wei, S., Zhang, D., 2016. An Approach of Web-based Point Cloud Visualization without Plug-in. In: IOP Conference Series: Earth and Environmental Science, 46(1).
Yu, A., Mei, W., 2019. Index model based on top-down greedy splitting R-tree and three-dimensional quadtree for massive point cloud management. J. Appl. Remote Sens., vol. 13, 2, 028501. https://doi.org/10.1117/1.JRS.13.028501.
Yu, X., Litkey, P., Hyyppä, J., Holopainen, M., Vastaranta, M., 2014. Assessment of low density full-waveform airborne laser scanning for individual tree detection and tree species classification. Forests 5 (5), 1011–1031. https://doi.org/10.3390/f5051011.
Zeng, X., 2012. WebGL based LiDAR Point clouds visualization. J. Hunan Univ. Sci. Technol., China 27 (4), 60–64.
Zhang, J., Huang, W., Zhu, X., Hwang, J.N., 2014. A subjective quality evaluation for 3D point cloud models. In: Proc. 2014 International Conference on Audio, Language and Image Processing, pp. 827–831.
Zhao, J., Wallgrün, J.O., LaFemina, P.C., Normandeau, J., Klippel, A., 2019. Harnessing the power of immersive virtual reality-visualization and analysis of 3D earth science data sets. Geo-spatial Inform. Sci. 1–14.
Zollhöfer, M., Stotko, P., Görlitz, A., Theobalt, C., Nießner, M., Klein, R., Kolb, A., 2018. State of the Art on 3D reconstruction with RGB-D cameras. Comput. Graphics Forum 37 (2), 625–652.