EFFICIENT LOADING AND VISUALIZATION OF MASSIVE FEATURE-RICH POINT CLOUDS WITHOUT HIERARCHICAL ACCELERATION STRUCTURES

J. Otepka[1,*], G. Mandlburger[1], Markus Schütz[2], N. Pfeifer[1], M. Wimmer[2]

[1] TU Wien, Department of Geodesy and Geoinformation, Vienna, Austria - (johannes.otepka, gottfried.mandlburger, norbert.pfeifer)@geo.tuwien.ac.at
[2] TU Wien, Institute of Visual Computing & Human-Centered Technology, Vienna, Austria - (mschuetz, wimmer)@cg.tuwien.ac.at

Commission II, WG II/3

KEY WORDS: Visualizing Huge Point Clouds, Parallel File Loading, Attribute Inspection, Real-Time Progress Rendering

ABSTRACT:

Nowadays, point clouds are the standard product when capturing reality independent of scale and measurement technique. Especially, Dense Image Matching (DIM) and Laser Scanning (LS) are state of the art capturing methods for a great variety of applications producing detailed point clouds up to billions of points. In-depth analysis of such huge point clouds typically requires sophisticated spatial indexing structures to support potentially long-lasting automated non-interactive processing tasks like feature extraction, semantic labelling, surface generation, and the like. Nevertheless, a visual inspection of the point data is often necessary to obtain an impression of the scene, roughly check for completeness, quality, and outlier rates of the captured data in advance. Also intermediate processing results, containing additional per-point computed attributes, may require visual analyses to draw conclusions or to parameterize further processing. Over the last decades a variety of commercial, free, and open source viewers have been developed that can visualise huge point clouds and colorize them based on available attributes. However, they have either a poor loading and navigation performance, visualize only a subset of the points, or require the creation of spatial indexing structures in advance. In this paper, we evaluate a progressive method that is capable of rendering any point cloud that fits in GPU memory in real time without the need of time consuming hierarchical acceleration structure generation. In combination with our multi-threaded LAS and LAZ loaders, we achieve load performance of up to 20 million points per second, display points already while loading, support flexible switching between different attributes, and rendering up to one billion points with visually appealing navigation behaviour. Furthermore, loading times of different data sets for different open source and commercial software packages are analysed.

1. INTRODUCTION

Modern surveying sensors capture the real world with high details producing an enormous amount of data in case of large projects. The newest generation of airborne laser scanning systems (e.g. *RIEGL* VQ-1560 II, Leica TerrainMapper) feature an effective pulse repetition rates of 2 MHz or more. Hence, such systems can measure more than 2 million points per second not even considering multi-target returns. In addition to conventional linear-mode LiDAR (Light Detection And Ranging), single photon sensitive laser scanners increase the measurement rate by utilizing highly sensitive sensor arrays, potentially achieving high areal capturing performance by flying from high altitude (Degnan, 2016), (Stoker et al., 2016) at the price of a higher measurement noise (Ullrich, Pfennigbauer, 2016). In addition to LiDAR, 3D point clouds obtained from multi-view stereo via dense image matching (Hirschmüller, 2008), (Haala, Rothermel, 2012) are widely used today, with the clear benefit of inherently providing color information for each matched point. The quality of photogrammetrically derived point clouds is constantly improving considering the ongoing progress in camera technology w.r.t. geometric and radiometric resolution. Multi-head cameras with nadir and oblique viewing directions mounted into a single camera frame are becoming state-of-the-art. With such sensors, city regions are typically captured with a Ground Sampling Distance (GSD) in the order of 10 cm and below (Toschi et al., 2017). The latest trend in airborne sensing is combining active laser scanners and passive cameras in a comprehensive hybrid sensor (Toschi et al., 2018), (Mandlburger et al., 2017). Therefore, resulting point clouds do have 100+ points/m^2. For large cities like Vienna, which has an area of approx. 415 km^2, this would mean a final point cloud of 50+ billion points.For many applications like city modelling, administrative planning, flood simulations, natural hazard and landslide monitoring, vegetation and forestry studies, the captured point clouds constitute the data basis rather than the final product. Usually in-depth analysis based on specific processing pipelines containing feature extraction, semantic labelling, and surface modelling steps lead to the final products, which are often geometric models like 3D-meshes or raster models.

Since there is no single optimal spatial index for all situations, processing software often use different spatial acceleration structures than point cloud viewers do. Rendering requires data structures that provide quick access to varying levels of detail of the model based on the position and direction of the viewer. Processing tasks, on the other hand, usually need fast access to all points and their attributes within a certain neighbourhood without levels of detail information (Weinmann et al., 2015). E.g., the point cloud processing framework OPALS (Pfeifer et al., 2013) uses a coarse persistent tiling structure and a 2D or 3D in-core kd-tree structure of each tile depending on the task to perform. While OPALS can quickly modify, filter, and augment all the data in a region, it cannot quickly display the res-

[*] Corresponding author

ults. With state-of-the-art methods, rendering of massive points clouds would require a time consuming hierarchical structure generation whenever new features are computed or point coordinates have changed (e.g. during strip adjustment or coordinate transformation).

Many processing tasks rely on per-point attributes. As described by (Otepka et al., 2013) attributes can be categorized into, i) directly measured features such as RGB, near Infrared, signal amplitude, echo width, scan angle, number of echos, and echo number, ii) features extracted from the point cloud itself like range corrected amplitude values, local normal vector, surface roughness, curvature, and point distribution features, and iii) features computed by combination with other data sources, like normalized height based on existing DTM, Normalized Difference Vegetation Index (NDVI) values for Laser Scanning (LS) point clouds extracted from multi-spectral images.
These features are used in a variety of basic processing steps like classification, but they are also used in very specific ways for certain applications. For comprehensive surveying projects it is usually necessary to establish, check and/or improve the georeferencing of the data. Depending on the measuring platform (static or kinematic, terrestrial or airborne) different algorithms, like ICP, strip adjustment or hybrid adjustments, are utilized to co-register different data subsets (Glira et al., 2019). In general, planar non-penetrable surface are selected to perform the co-registration. To find such regions appropriate features are extracted and attached to the point clouds.

Segmentation and classification tasks often use point features to analyse points independently. E.g. (Mallet et al., 2011) used Support Vector Machines (SVM) and attributes of different categories to point-wise classifying full-waveform LiDAR data of urban areas. (Vosselman et al., 2017), on the other hand, used segments and Conditional Random Fields (CRF) to classify points in large chunks rather than separately. Nevertheless, attributes of different categories were also used in the overall process again. (Kumar et al., 2019), who apply neural networks for semantic segmentation, also rely on features computed in neighborhoods of different size. No hand-crafted features are required in deep learning (Liu et al., 2019), but it may be of interest to study the learned features, e.g. by visualizing them.

Also modelling strategies often utilize point-wise attributes. When deriving Digital Terrain Models (DTM), the full-waveform echo width attribute can help to discriminate between ground and low vegetation points (Wagner et al., 2008). For separating leafs from woody elements in point clouds of single or multiple trees, (Wang et al., 2017) used features from normal vector computation. (Nebiker et al., 2010) stressed the importance of ‘rich’ point clouds for city modeling, which are dense point clouds carrying semantic information, i.e. additional attributes.

Nowadays, an abundant list of commercial, free, and open-source point cloud viewer exists. However, when demanding the capability of displaying arbitrary attributes, the list narrows down. Such mature viewers usually build hierarchical structures to provide responsive navigation facility. Whereas some viewer do this in memory only (e.g. CloudCompare and FugroViewer) others rely on persistent structures (e.g. Potree, RiProcess, Euclideon) which removes memory limitation and therefore allows rendering even larger point clouds (>billion points). Nevertheless, creating hierarchical structures costs performance and consumes resources. In contrast, progressive rendering does not require such structures and therefore can be expected to provide unmatched loading performance while still providing high quality real-time rendering capabilities. As the analysis will show that up to 20 million points per second can be loaded and rendered when using the LAS file format.

Therefore, this method is ideal for a set of situation, at different stages within the processing pipeline: The *initial point cloud* will be viewed to obtain an overview, and roughly check quality and completeness, but also to check for outliers. An *intermediate point cloud* will be viewed for parameterize further processing, e.g. by selecting thresholds or to judge the success of previous operations. At any stage the points and features can be viewed and distributed to colleagues or customers.

This work extends the approach of (Schütz et al., 2020) in several ways at which the main contributions are

- implementing an efficient multi-threaded LAZ file reader
- emphasizing the importance of fast visualization of arbitrary attributes in the context of topographic point cloud processing pipelines
- performance comparison of loading times with a variety of open source and commercial point cloud viewer and processing software

2. STATE OF THE ART AND RELATED SOFTWARE

The LAS file format is established as quasi-standard exchange format for point cloud data, especially in the context of airborne sensing (ASPRS, 2019). Although it was originally developed by the American Society for Photogrammetry and Remote Sensing (ASPRS) for handling LiDAR points clouds, it is nowadays regularly used for image matching point clouds as well. Virtually all current viewer software can read LAS files and many of them support colorization based on the standard point attributes. Depending on the format version and the used point type, different standard attributes[1] are supported, such as intensity (2 bytes), GPS time (8 bytes), return number, number of returns, classification (varying sizes), point source id (1 byte), etc. Some Point Data Record Formats (PDRF) also support 16 bit RGB values, which are usually used for point clouds from image matching or hybrid sensor systems (2, 3, 5, 7). The standard PDRFs 8 and 10 additionally provide an 16 bit nIR attribute. Furthermore, from the initial version of the standard on, LAS has allowed for appending arbitrary amounts of bytes to each point. The so called *extra bytes* allow storing arbitrary user-defined attributes for each point within LAS. Practical relevance of this smart concept came with the release of LAS version 1.4, which introduced a new Variable Length Record (VLR) for describing the stored features in detail (name, description, data type, scale, offset, and valid sample range). Although the standard was release in November 2011, there is still only a handful of software packages that can properly handle attributes using *extra bytes* records. Especially for large point clouds, each additional attribute noticeably increases storage consumption, which is why such features are often immediately dropped after processing or only stored as intermediate results in internal structures. On the other hand, extracted features often describe the local neighborhood of a point which are useful for subsequent processing steps. Therefore, it is justifiable to store attributes in an exchangeable way. Since hardly

[1] `https://opals.geo.tuwien.ac.at/html/stable/ref_fmt_las.html`

any point cloud processing software exports computed attributes into LAS files, it comes as no surprise that only a few point cloud viewer directly support *extra byte* attributes.
As an addition to LAS, Martin Isenburg developed a compressed counterpart called LAZ (Isenburg, 2013). It is available under the liberal LGPL (GNU Lesser General Public License), which allows integrating the code in open and closed source applications free of charge. Although LAZ is (currently) not included in the LAS standard, it has actually increased the significance and the spread of the format.

Table 1 lists a range of open source and (free) commercial point cloud software packages based on software products described by (Rooms, 2020). Whereas some packages are viewers only, others provide mature editing functionality. Most commercial products provide a free standalone viewer. In those cases, the free viewer rather than the full featured point cloud software is listed. Although the presented table is not complete, it covers the most commonly used software packages to the best of our knowledge. As can be seen, nearly all products support colorzing points based on RGB, intensity, or classification. However, only CloudCompare, opalsView and Potree (currently under development) support colorization based on arbitrary point attributes. Hence, users often have to misuse standard LAS fields for transferring non standard attributes between software packages. This constitutes an unsatisfactory and error-prone solution that also limits the number of transferable attributes. This is why the evaluated rendering strategy also focuses on arbitrary attribute handling and fully supports *extra byte* attributes stored within LAS files.

Most software packages use some sort of spatial index structures allow rendering large points clouds. The only exception is lasview from LAStools, which thins out the data to a degree that all remaining points can be directly rendered without any acceleration structure in order to get a quick but rough overview of the data requiring very little memory. However, details get lost in case of larger point clouds. A few software packages rely on explicitly created acceleration structures, like e.g. Potree, Bentley Pointools View, Leica Cyclone TruView or Euclideon. Such concepts are necessary for rendering multi billion points or streaming them through the internet. Therefore, they are typically needed when publishing final results. Some viewers (CloudCompare, Scene LT, 3DReshaper Free Viewer, etc.) use a reduced point set while navigating through the data to provide fluid and smooth motions. After the motion has stopped, missing points are added. This rendering concept is related to our method.

In the fields of computer graphics, most work has focused on rendering large point clouds by creating and rendering hierarchical level-of-detail structures. Grouping points into a multiresolution tree structures, where each node stores a subset or a representative model of the original model, is the key breakthrough for efficient rendering arbitrarily large point clouds on the GPU. (Rusinkiewicz, Levoy, 2001) was the first using point-based hierarchical structures to render large meshes. (Dachsbacher et al., 2003) and (Gobbetti, Marton, 2004) improved this concept and offered GPU-friendly structures. (Tredinnick et al., 2016) and (Ponto et al., 2017) proposed a progressive rending technique that re-projects the previous frame into the current one and fills holes by rending additional points. The full amount of data is visible after a few frames meaning that the final images is visible. In computer graphics this is often referred to as convergence is achieved. Their work differs in that it focuses on hierarchical structures, which is not required by the evaluated progressive method.

Due to advances of technical devices, computational power and algorithmic concepts (Stotko et al., 2018) Virtual and Augmented Reality applications have experienced an signified boost within the last years. For completeness it is mentioned that the evaluated progressive rendering method is also suitable for such applications (Schütz et al., 2020) but a thorough analysis is beyond the scope of this paper.

3. PROGRESSIVE RENDERING

The concept of the evaluated progressive rendering method was introduced by (Schütz et al., 2020). For convenience to the reader, a summary with reduced technical details is presented in the following. The central idea of the evaluated method is reprojecting visible points of previous frame into the current frame and filling holes by progressively rendering all points over multiple frames until convergence. Maintaining real-time frame rates and keeping the application responsive at all times, is the sought-after target behaviour. Due to the spatial coherence of consecutive frames, most of the points visible in the previous frame will also be visible in the final image of the subsequent frame. Furthermore, filling randomly selected rather than sequentially loaded points, helps to achieve a pleasant and uniform image convergence.

Next, the necessary data structures, the shuffling algorithm, and the actual render pipeline is described. The concept especially considers the possibility of rending arbitrary point attributes and how they can be switched as fast as possible during rendering.

3.1 Data Structure

Our rendering concept makes use of two main data structures, on the CPU side for loading and reorganizing the attributes and one GPU side for shuffling and rendering the points. Since we require all points loaded into the GPU memory, we aim at keeping the GPU point size as small as possible, while the entire point set including all attributes is kept in CPU memory. The latter usually provides more memory and swapping attribute buffers to disk won't effect the rendering speed.

To stream attributes with minimal memory bandwidth consumption and, therefore, maximum speed from CPU to GPU, it is necessary to organize the attributes in a Structure of Arrays (SoA) fashion. Since point cloud file formats are typically written as Array of Structures (AoS), it is necessary to reorganize the data during loading. On the GPU side, we use a shuffled vertex buffer with 16 bytes per point: 12 bytes for three reduced float coordinates and 4 bytes for the displayed attribute. For single attributes, a 4 byte float value is available, whereas for RGB each value of the 3 channels is limited to a single unsigned byte. It is up to the vertex shader to interpret the attribute data correctly. The actual point data are not directly uploaded into the vertex buffer. Instead, we use a *Distribute* buffer with 16 bytes per point during the initial upload and 4 bytes per point when switching attributes. Finally, a *Reproject* buffer is used to store all visible points (coordinates, attribute, and point index within the VBO) at the end of a frame.

3.2 Incremental Parallel Shuffling

Rendering randomly selected points clearly improves the perceived visual quality during convergence to the final image,

Software Package (Company name)	License	Viewer only	LAS	LAZ	*extra bytes* Attributes	Attribute based Colorization	App. type
CloudCompare	open source	–	x	x	x	full	standalone
MeshLab	open source	–	–	–	–	RGB, I	standalone
ParaView	open source	–	–	–	–	full	standalone
Potree/Entwine	open source	x	x	x	in dev.	full in dev.	client/server
3DReshaper Free Viewer	free commercial	x	x	x	–	RGB, I, class	standalone
Arena4D (Veesus)	commercial	–	x	x	–	RGB, class	standalone
Bentley Pointools View	free commercial	x	–	–	–	RGB, I	standalone
Capturing Reality	commercial	–	–	–	–	RGB, I, class	standalone
EdgeWise (ClearEdge3D)	commercial	x	x	–	–	RGB, I, class	standalone
Euclideon	commercial	x	x	x	–	RGB, I, class	client/server
FugroViewer (Fugro)	free commercial	x	x	x	–	LAS attributes	standalone
VisionLidar LTD (GeoPlus)	free commercial	x	x	x	–	RGB, I, class	standalone
lasview (LAStools)	mixed	–	x	x	–	LAS attributes	standalone
Leica Cyclone TruView	free commercial	x	–	–	–	RGB, I, class	client/server
MARS (Merrick)	free commercial	x	x	–	–	RGB, I, class	standalone
opalsView (TU Wien)	mixed	–	x	x	x	full	standalone
PointCab	commercial	–	x	x	–	RGB, I, class	standalone
Quick Terrain Reader (AppliedImagery)	free commercial	x	x	x	–	RGB, I	standalone
RiProcess (RIEGL)	commercial	–	x	x	write only	LAS attributes	standalone
Scene LT (FARO)	free commercial	–	x	x	–	RGB, I, class	standalone
TerraScan (Terrasolid)	commercial	x	x	x	–	LAS attributes	CAD plugin

Table 1. List of open source and commercial point cloud software products.

compared to rendering points in their original and potentially sorted order. When points are shuffled while loading, rendering N random points is identical to render any N consecutive points of the (shuffled) vertex buffer. Since we aim to display points while loading, a shuffling algorithm is required that can incrementally shuffle points as they become available. We use an algorithm described by (Preshing, 2012) to compute a permutation of a sequence of numbers $[0, ..., P-1]$, where P is a prime that is congruent to 3 (mod 4). This approach maps each number in the sequence to another number of the same set without collisions or duplicates. In our case, we use the point index within the loaded array of points and compute the final index position within shuffled vertex buffer:

$$\text{permute}(i) = \begin{cases} i^2 \bmod P, & \text{if } i \leq \frac{P}{2} \\ P - i^2 \bmod P, & \text{if } i < P \\ i, & \text{otherwise} \end{cases} \quad (1)$$

Since the formula only depends on the current index and the prime number, efficient compute shaders can be utilised to copy points to their final position inside the shuffled vertex buffer without synchronization between threads. The only precondition is that the total number of points needs to be known in advance. This is, however, of minor concern since most file formats store this information in the header or, else, a conservative estimate is still sufficient. The points above the selected prime P stay simply unshuffled. There number is so low, that they won't affect the visual render appearance.

The disadvantage of the presented prime based permutation algorithm is its relatively low randomness resulting in clearly visible patterns. Since Equation 1 is bijective, it can be applied recursively still resulting an unique target index. It turnes out that a single recursion (i.e. applying the formula twice) result in a sufficient randomness for our rendering method so that no patterns can be visually observed any more.

3.3 Loading Strategies

Our rendering method is capable of displaying points while the remaining data are still being loaded. Therefore, files are loaded and transformed into GPU friendly buffers, which can then be efficiently sent to the GPU's *Distribute* buffer within the render loop. We have tested different loading strategies using our own LAS reader class and LASlib (from LAStools by Martin isenburg) for reading LAS and compressed LAZ (Isenburg, 2013) files. It turns out that the parsing of the LAS byte stream into GPU friendly buffers is the limiting factor when reading such files from SSD. Performing the parsing with multiple threads can clearly speed up the overall loading process. This fact is even more evident, when reading LAZ file since the decompression requires much more CPU resources. Since LAZ files are organised in chunks (50,000 points per default), it is possible to directly seek to the beginning of such point patches without any decompression operation. By creating multiple LASlib reader objects and seeking to different chunks, we were able to read LAZ files in parallel which increased the reading performance by a factor of 6 on an Intel hexa-core CPU.

3.4 Rendering Pipeline

The evaluated rendering pipeline reprojects points from the previous frame into the current and adds missing data by rendering a certain number of random points. Over the course of multiple frames, the result will converge to the same image as rendering all points at once, neglecting render order and z-fighting issues. The method consists of three steps:

1. **Reproject**: Render all the points that were visible in the previous frame, reprojected to the current frame.
2. **Fill**: Render a batch of random points to fill holes from the shuffled vertex buffer.
3. **Prepare**: Create a new vertex buffer from all points that are visible in the rendered image. This vertex buffer will be used in the reprojecting step of the next frame.

Reprojecting the visible points from the previous frame is an efficient method for rendering only a minimal subset of points and achieving an image that is already close to convergence. Nevertheless, previously occluded parts or parts outside the previous frustum may become visible. The *Fill* step tackles those uncompleted regions by adding random points. We chose to fill

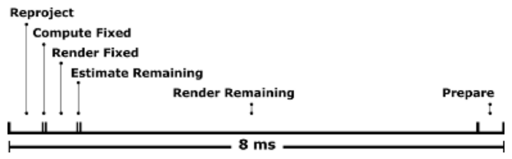


Figure 1. The adaptive fill budget renders a fixed amount of points first, and then an estimated additional amount that can be rendered in the remaining time of the frame.

points in a randomly shuffled order which results in a relatively uniform and visually appealing convergence. Using unshuffled points which are spatially sorted or structured in some way, will result in unpleasant flickering artifacts during motion because in each frame, parts of the image will fully converge, while other parts will see no progress at all until later frames.
The central challenge of the *Fill* step is to select the appropriate number points to be render within each frame. If too many points are added, the frame rate drops and real-time navigation capability gets lost. On other hand, too few points will increase the number of frames that are necessary to achieve convergence. It is impossible to pre-select an appropriate value since render performance strongly depends on viewpoint, zoom level, data complexity, and hardware performance of the GPU. Since the first two parameters change every frame, an adaptive fill budget strategy was developed that estimates the number of points to be filled for each frame.

3.5 Handling Point Attributes

As described in Section 1, point-wise attributes are important features for many different processing tasks. Hence, we see it as a necessity that the viewer can visualize arbitrary attributes based on user-defined color maps. Our data sets contain up to 40 different attributes using up to 100 bytes per point (bpp). Assuming a graphics card with 8 GB memory, a maximum of $\frac{8*1024^3}{100} \approx 85$ points can be stored on the GPU. Since we generally only need up to 4 attributes at once on the GPU, the remaining attributes unnecessarily consume GPU memory. Furthermore, our rendering pipeline is also strongly affected by memory bandwidth, because the *Reproject* vertex buffer is recomputed each frame. More bytes per vertex results in slower buffer generation and reduces the adaptive fill budget. Consequently, we limit the GPU points size to 16 bytes, comprising of $3 \cdot 4 = 12$ bytes for the XYZ coordinates and another 4 bytes encoding 1-4 attributes. Hence, we can store many more points on the GPU, namely $\frac{8*1024^3}{16}$ = 536M points (assuming again 8 GB GPU memory). In reality, the entire GPU memory will not be available, since other parts of the viewer and other applications including the operating system also require some GPU memory.

In order to be able to instantly visualize any attributes, we keep them in main memory and stream them to the GPU based on user requests. At the start of each frame, the main thread sends multiple patches of attributes to the GPU. A compute shader distributes the vertex attribute data to the respective vertices with the same shuffle algorithm as during the initial loading step, thereby overriding the previous vertex attribute data. Streaming a new attribute from CPU to GPU happens at rates of 125-800 million points per second depending on GPU memory speed and attribute size. For the *Vienna* data set with 124M points, switching attributes takes 0.155-0.356 ms on a RTX 2080 TI, whereas on the GTX 1650 and the GTX 1050 TI switching times are approx. twice as long.

4. EVALUATION

In the following section our viewer concept is tested and evaluated with the data sets shown in Figure 2. We provide a comparison by listing the loading times of selected software packages (cf. Table 1). The list of investigated software mainly focuses on freely available packages. Although a comparison of frame rates, render performance and quality is also of interest[2], the rendering and navigation strategies vary strongly among different products, which complicates the definition of appropriate measures. Therefore, we limit the comparisons to loading times .

When it comes to visualising arbitrary attributes directly from LAS files, the software list reduces to the following packages: CloudCompare, opalsView, and Potree/Entwine. Although Potree and Entwine are fully capable of storing arbitrary attributes in their hierarchical structures, the actual viewing part currently cannot utilize the additional attributes for visualization.[3] Nevertheless, we still put them into the category of 'arbitrary attribute viewers' . All tests were performed on a standard desktop computer equipped with the following hardware: Intel Core i7-4771 3.5 GHz CPU (4 cores/8 threads), NVIDIA GeForce GTX 1650 GPU with 4 GB GPU memory, 120 GB Intel + 1 TB Samsung 860 EVO SSD, and 16 GB RAM.

4.1 Data Sets

The performance analyses were carried out using four different data sets: Three Airborne Laser Scanning (ALS) point clouds with 19M, 124M, and 407M points, respectively, and one Terrestrial Laser Scanning (TLS) point cloud with 208M points as described in Table 2. The two smaller data sets (*Forest* and *Vienna*) existed in two different variants: With standard LAS attributes (point record format 1 and 2, respectively) and with an extended attribute set (31 and 40 attributes in total). The latter is used for testing the attribute capabilities of different software packages.

4.2 Performance Analyses

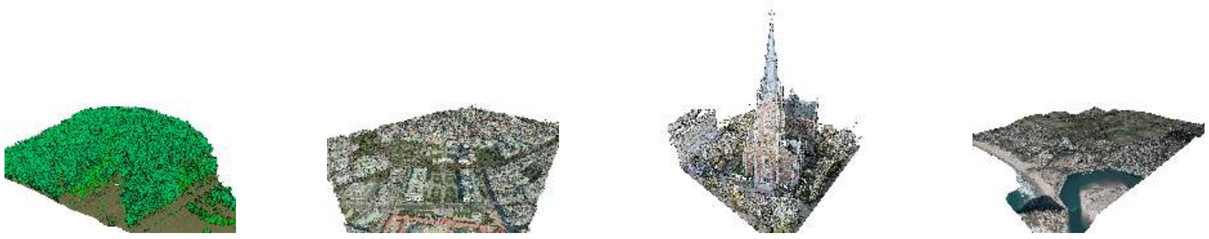
The performance analyses were carried out in two groups, (i) software packages that only support standard attributes and (ii) point cloud software supporting the full set of attributes stored in LAS *extra bytes*. In the first group, all four data sets were tested and for the *Forest* and *Vienna* data set only the standard LAS attributes were used. Table 3 shows considerable differences between LAS and LAZ reading performance. Whereas some packages read the same content from compressed LAZ nearly as fast as from uncompressed LAS, others are up to 3 times slower. In cases were no obvious log file or log window existed, loading times were measured by hand. The timer was started when the actual import began and stopped when a point cloud was displayed or a finished import dialog was presented (i.e. some user interaction was required).

The last two columns of Table 3 show the points per second loading performance averaged over all four data sets. It clearly shows the outstanding performance of our concept. Whereas LAS files are read more than 5 times faster in average compared to other software packages, LAZ files are read even faster: our multi-threaded LAZ reader is in average more than 10 times

[2] Details on our method are given in (Schütz et al., 2020).
[3] This is feature is currently being developed.

Data Set	File Size [GB] LAS	LAZ	#points [Mio]	#attributes	bpp	Density pt / m^2	Area km^2	Acquisition RIEGL	Sensor
Forest (standard)	0.52	0.18	19.2	11	28	136	0.14	ALS	RIEGL LMS-Q1560
Forest (extended)	1.8	1.0	19.2	31	100				
Vienna (standard)	3.1	1.2	124.5	15	36	56.43	2.58	ALS	RIEGL LMS-Q1560
Vienna (extended)	10.6	5.3	124.5	40	88				
St. Elisabeth	5.7	1.7	208.6	12	28	34 T	0.01	TLS	RIEGL VZ-2000i
Morro Bay	13.5	1.8	407.0	13	34	22.06	18.45	ALS	Leica ALS70 + Optech Orion

Table 2. Key parameters of used data sets. For data set *Forest* and *Vienna* standard and extended attribute sets are available. ALS: Airborne Laser Scanning. TLS: Terrestrial Laser Scanning.



(a) Forest (b) Vienna (c) St. Elisabeth Church (d) Morro Bay

Figure 2. Data sets used in this paper

faster. The only tested software that comes close to the progressive rendering method is the Merrick Advanced Remote Sensing Software (MARS) package. For the Morro Bay data set, MARS displays a strongly reduced point cloud after 54 s already. However, it takes at another 39 s with heavy SSD access before one can navigate within the point cloud. So the real import duration is rather above 90 s. We acknowledge that some of the tested software use accelerations structures for supporting sophisticated feature analysis, which slows down loading times. Since we do not generate any spatial index, such features may not be provided if not implemented directly on the GPU.

From a practical point of view, displaying points while loading and still providing a responsive 3d windows is a valuable feature that can clearly speed up certain operations. An incorrect specified file can be usually identified after a few seconds of loading. In other viewers one has to wait til the file is fully loaded before realizing ones error. In some situations the object or area of interest is maybe fully loaded at the begin. Due to the responsive navigation, the user can instantly inspect the object and could be finished even before the file is fully loaded.

From the authors point of view, however, one of the most valuable feature of the presented viewer is the capability of efficiently visualizing arbitrary point attributes of large point clouds, as shown in Figure 3. To the best of our knowledge, there are only three other viewers that can directly visualize arbitrary *extra bytes* attributes from LAS and LAZ files. Hence, the full set of attribute evaluation test was limited to four software packages. Besides the loading times of the extended LAS/LAZ files, also the duration of switching attributes was investigated. Comparing the last two columns of Table 3 and Table 4, it is clearly visible that the overall performance measured as points-per-second has dropped. This is not surprising since the extended files are 3-5 times larger. The actual throughput performance in bytes stays roughly the same. Our method also outperforms the other packages in switching attributes. However, from a practical point of view this is of minor concern, as even the slowest measured switching duration of 4.4 s is still acceptable.

For completeness it is mentioned that a standard desktop computer with medium performance was deliberately selected for testing. 4 GB GPU memory is actually not enough to fully upload the *Morro Bay* data set, since 407M points with 16 bytes per point results in a memory consumption of 6.1 GB. Graphics card drivers usually implement a shared memory concept by occupying CPU Ram when needed. Due to this inherent mechanism, it was possible to load the *Morro Bay* point cloud where 3.4 GB GPU and 4.5 GB shared memory were used. When the computer started to use shared memory, the loading process perceivable slowed down and responsive navigation got lost for about 20 s. Once the file was fully loaded, the navigation became smooth again. Using a better graphics card (e.g. NVidia RTX 2080 Ti with 11 GB), the file was loaded in about 14 s and the viewer was responsive at any time (see `https://www.youtube.com/watch?v=YFAThGdXL8s` and (Schütz et al., 2020)). The limited GPU and CPU memory has presumably affected the performance of other tested software as well, specially for the larger data sets. Hence, a more powerful system would have decreased the import duration, but the overall trend and ratios between the different products should roughly stay the same. This will be verified in future tests. It should be noted that all tests were made on SSDs only. Import results and ratios between different software and formats may be significant differently on hard disks, but definitely slower.

5. DISCUSSION AND CONCLUSION

In this investigation, we make use of a progressive rendering method that can render any point cloud that fits in GPU memory in real time without the need to generate acceleration structures in advance (Schütz et al., 2020). The central idea is to distribute the actual rendering over multiple frames til convergence is achieved. The method reprojects visible points from the previous into the current frame and randomly adds points to generate visually appealing results. Points are displayed already while loading but in a way, that the applications stays responsive at any time.

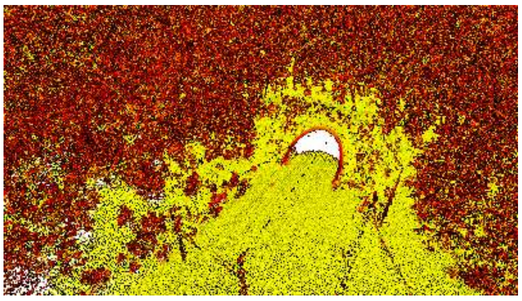
Point-wise attributes are important features throughout the entire processing pipeline. Tasks like outlier detection, georefer-

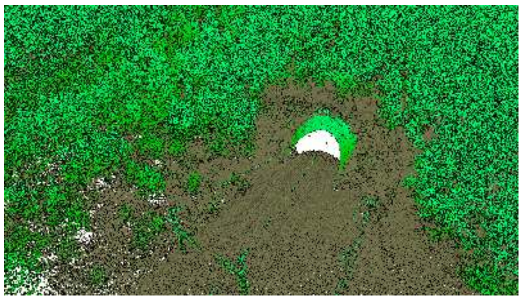
	Forest / 28 bpp		Vienna / 36 bpp		St. Elisabeth / 28 bpp		Morro Bay / 34 bpp		avg. Mio pts / s	
	LAS	LAZ	LAS	LAZ	LAS	LAZ	LAS	LAZ	LAS	LAZ
Evaluated Method	00:01	00:02	00:07	00:20	00:18	00:32	01:07	01:01	13.6	7.2
CloudCompare	00:19	00:32	01:54	03:43	03:26	06:08	08:12	10:45	1.0	0.6
Entwine[4]	00:36	00:48	04:48	06:23	08:03	10:30	15:50	18:54	0.5	0.4
FugroViewer	00:08	00:22	–[1]	–[1]	–[1]	–[1]	–[1]	–[1]	2.4	0.9
QT Reader	00:05	00:19	00:39	02:12	01:07	03:43	03:58	–[2]	2.9	1.0
SCENE LT	00:16	00:29	01:40	03:17	02:59	04:58	06:29	06:59	1.2	0.7
VisionLidar	00:17	00:53	01:40	06:09	03:03	09:20	08:31	17:07	1.1	0.4
3DReshaper	00:11	00:23	00:55	02:14	01:40	03:41	5:36	07:25	1.8	0.9
MARS[3]	00:02	–	00:24	–	00:25	–	00:54	–	7.6	–
ReCap	01:43	01:53	10:57	12:21	11:32	13:36	36:15	38:00	0.2	0.2
Arena4D VPC Creator[4]	00:40	00:51	04:23	05:46	03:57	06:08	15:38	19:16	0.6	0.4

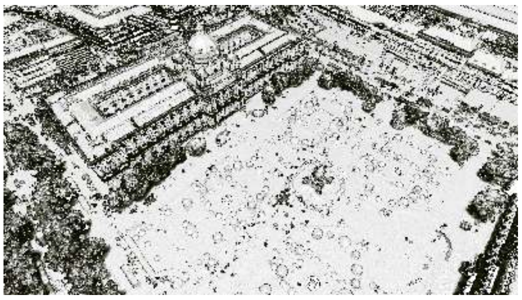
Table 3. Import/Loading duration [mm:ss] for LAS/LAZ files with standard attributes. [1]Crash while loading. [2]Loading aborted without message. [3]Loading without indexing/No LAZ support. [4]Hierarchical Structure Generator only.

	Forest / 100 bpp				Vienna / 88 bpp				avg. Mio pts / s	
	Loading		Switching Attr.		Loading		Switching Attr.		Loading	
	LAS	LAZ	min [s]	max [s]	LAS	LAZ	min [s]	max [s]	LAS	LAZ
Evaluated Method	00:04	00:15	0.08	0.09	00:19	01:07	0.6	0.7	5.64	1.56
CloudCompare	00:47	02:06	0.4[1]	0.5[1]	07:04	11:23	2.2[1]	4.4[1]	0.35	0.17
opalsView	01:20	02:23	2.1[1]	2.2[1]	–[2]	–[2]	–[2]	–[2]	0.24	0.13
Entwine[3]	01:07	02:34			15:22	19:47			0,19	0,08

Table 4. Loading [mm:ss] and Switching Attribute [s] duration for LAS/LAZ files with extended attributes. [1]Hand stopped. [2]Out of memory. [3]Hierarchical Structure Generator only.


(a) Planarity: For identifying planar regions


(b) NormalizedZ: Height above terrain


(c) EchoRatio: Vertical penetration measure

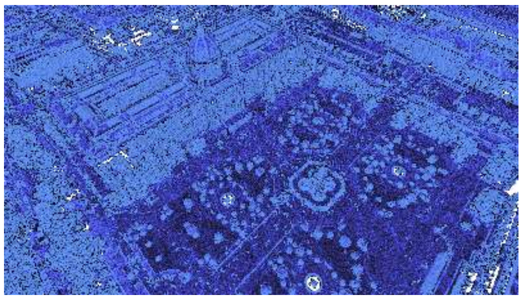

(d) NormalSigma0: For detecting smooth or rough areas

Figure 3. Visualisation of Different Attributes

encing, segmentation, semantic classification, and change detection all make use of various attributes during computation. Although such computational intensive steps are often carried out offline, a visual inspection of the point data including their attributes is usually needed at various stages of the processing pipeline. Since such pipelines often utilize application specific spatial indices different from the hierarchical acceleration structures generally used for rendering, productivity is improved if the generation of such structures can be omitted for viewing the data. On moderate GPU hardware with 4 GB memory, our method can rapidly render up to 250M points. In general, visualizing larger point clouds is still be possible, but partly reduced loading performance and navigation speed have to be accepted due to the limited bandwidth of shared CPU memory. On high-end graphics cards with a lot of GPU memory, the number of renderable points linear increases with the size of the GPU memory. Our method has been successfully tested on a RTX Titan with 24 GB memory rendering 1 billion points.

Currently our method has a large CPU memory footprint since the full point clouds including all attributes are kept in memory. Due to its organisation as structure of arrays (see 3.1), it is straight forward to write back attributes into separate binary files after they have been loaded. Once attribute files have been created and relevant data uploaded to GPU, its CPU memory can be released. When switching attributes, the corresponding attribute file can be loaded and directly uploaded to the GPU. If those temporary attribute files are placed on an SSD, the final attribute switching duration is hardly affected due to the fast reading speed of today's SSDs on large files (typically $>$ 500 MB/s).

Our tests have shown that the reading speed of LAZ files can be dramatically improved if the file is read/parsed with multiple threads. LAZ files are organized in chunks (50K points per default), which can be directly accessed and parsed in parallel. Based on the LASlib, our implementation was able to read LAZ files up to 6 times faster than in standard single thread mode. Finally, we want to encourage other developer to fully support *extra byte* attributes in their software. The LAS file format allows exchanging arbitrary point-wise attributes in a flexible and efficient manner.

ACKNOWLEDGEMENTS

The authors with to thank *Riegl Laser Measurement Systems* for providing the Forrest, Vienna and St. Elisabeth church data set and *Open Topography* for funding and hosting the data set of Morro Bay.

REFERENCES

ASPRS, 2019. LAS Specification 1.4 - R15. https://www.asprs.org/committees/standards-committee. Accessed 2020/05/04.

Dachsbacher, C., Vogelgsang, C., Stamminger, M., 2003. Sequential point trees. *ACM Transactions on Graphics*, 22, 657.

Degnan, J. J., 2016. Scanning, Multibeam, Single Photon Lidars for Rapid, Large Scale, High Resolution, Topographic and Bathymetric Mapping. *Remote Sensing*, 8(11), 923–958.

Glira, P., Pfeifer, N., Mandlburger, G., 2019. Hybrid Orientation of Airborne LIDAR Point Clouds and Aerial Images. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, IV-2/W5.

Gobbetti, E., Marton, F., 2004. Layered point clouds: A simple and efficient multiresolution structure for distributing and rendering gigantic point-sampled models. *Computers & Graphics*, 28, 815-826.

Haala, N., Rothermel, M., 2012. Dense Multi-Stereo Matching for High Quality Digital Elevation Models. *PFG Photogrammetrie, Fernerkundung, Geoinformation*, 2012(4), 331–343.

Hirschmüller, H., 2008. Stereo Processing by Semiglobal Matching and Mutual Information. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(2), 328–341. http://dx.doi.org/10.1109/TPAMI.2007.1166.

Isenburg, M., 2013. LASzip: lossless compression of LiDAR data. *Photogrammetric Engineering & Remote Sensing*, 79, 209-217.

Kumar, A., Anders, K., Winiwarter, L., Höfle, B., 2019. Feature relevance analysis for 3d point cloud classification using deep learning. *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences*, IV-2W5, 373–380.

Liu, W., Sun, J., Li, W., Hu, T., Wang, P., 2019. Deep Learning on Point Clouds and Its Application: A Survey. *Sensors*, 19(19), 4188.

Mallet, C., Bretar, F., Roux, M., Soergel, U., Heipke, C., 2011. Relevance assessment of full-waveform lidar data for urban area classification. *Isprs Journal of Photogrammetry and Remote Sensing*, 66, 85-91.

Mandlburger, G., Wenzel, K., Spitzer, A., Haala, N., Glira, P., Pfeifer, N., 2017. Improved topographic models via concurrent airborne LiDAR and dense image matching. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, IV-2/W4, 259–266.

Nebiker, S., Bleisch, S., Christen, M., 2010. Rich point clouds in virtual globes – A new paradigm in city modeling? *Computers, Environment and Urban Systems*, 34(6), 508 - 517. GeoVisualization and the Digital City.

Otepka, J., Ghuffar, S., Waldhauser, C., Hochreiter, R., Pfeifer, N., 2013. Georeferenced Point Clouds: A Survey of Features and Point Cloud Management. *ISPRS Int. Journal of Geo-Information*, 2(4), 1038–1065.

Pfeifer, N., Mandlburger, G., Otepka, J., Karel, W., 2013. OPALS – A framework for Airborne Laser Scanning data analysis. *Computers, Environment and Urban Systems*, 45, 125-136.

Ponto, K., Tredinnick, R., Casper, G. R., 2017. Simulating the experience of home environments. *2017 International Conference on Virtual Rehabilitation (ICVR)*, 1-9.

Preshing, J., 2012. How to generate a sequence of unique random integers. https://preshing.com/20121224/how-to-generate-a-sequence-of-unique-random-integers. Accessed 2020/05/04.

Rooms, F., 2020. Point clouds – 4: A cloud of software. https://blog.bricsys.com/free-point-cloud-software. Accessed 2020/05/04.

Rusinkiewicz, S., Levoy, M., 2001. QSplat: A Multiresolution Point Rendering System for Large Meshes. *Proceedings of SIGGRAPH*, 2000, 343–352.

Schütz, M., Mandlburger, G., Otepka, J., Wimmer, M., 2020. Progressive Real-Time Rendering of One Billion Points Without Hierarchical Acceleration Structures. *Computer Graphics Forum (Proceedings of Eurographics)*, 39(2).

Stoker, J. M., Abdullah, Q. A., Nayegandhi, A., Winehouse, J., 2016. Evaluation of Single Photon and Geiger Mode Lidar for the 3D Elevation Program. *Remote Sensing*, 8(9), 716–767.

Stotko, P., Krumpen, S., Hullin, M. B., Weinmann, M., Klein, R., 2018. SLAMCast: Large-Scale, Real-Time 3D Reconstruction and Streaming for Immersive Multi-Client Live Telepresence. *CoRR*, abs/1805.03709, 2102-2112. http://arxiv.org/abs/1805.03709.

Toschi, I., Ramos, 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. Photogramm. Remote Sens. Spatial Inf. Sci.*, XLII-1/W1, 519-526.

Toschi, I., Remondino, F., Rothe, R., Klimek, K., 2018. Combing Airborne Oblique Camera and LIDAR Sensors: Investitation and new Pesorectives. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-1, 437–444.

Tredinnick, R., Broecker, M., Ponto, K., 2016. Progressive feedback point cloud rendering for virtual reality display. *2016 IEEE Virtual Reality (VR)*, 301–302.

Ullrich, A., Pfennigbauer, M., 2016. Linear LIDAR versus Geiger-mode LIDAR: impact on data properties and data quality. *Proc. SPIE*, 9832, 983204–983217.

Vosselman, G., Coenen, M., Rottensteiner, F., 2017. Contextual segment-based classification of airborne laser scanner data. *ISPRS Journal of Photogrammetry and Remote Sensing*, 128, 354 - 371.

Wagner, W., Hollaus, M., Briese, C., Ducic, V., 2008. 3D vegetation mapping using small-footprint full-waveform airborne laser scanners. *International Journal of Remote Sensing*, 29, 1433-1452.

Wang, D., Brunner, J., Ma, Z., Lu, H., Hollaus, M., Pang, Y., Pfeifer, N., 2017. Separating tree photosynthetic and non-photosynthetic components from point cloud data using dynamic segment merging. *Forests*, 9(5), 252.

Weinmann, M., Jutzi, B., Hinz, S., Mallet, C., 2015. Semantic point cloud interpretation based on optimal neighborhoods, relevant features and efficient classifiers. *ISPRS Journal of Photogrammetry and Remote Sensing*, 105, 286 - 304.