ISSN 1213-6972 Journal of WSCG http://www.wscg.eu Vol.28, No.1-2, 2020

Visualizing Massive Pristine LIDAR Amplitude Responses

Werner Benger
AirborneHydroMapping GmbH
A-6020 Innsbruck, Austria
w.benger@ahm.co.at

Wolfgang Leimer
AirborneHydroMapping GmbH
A-6020 Innsbruck, Austria
w.leimer@ahm.co.at

Ramona Baran
AirborneHydroMapping GmbH
A-6020 Innsbruck, Austria
r.baran@ahm.co.at

Center for Computation & Technology
Louisiana State University
Baton Rouge, USA

ABSTRACT

Airborne light detection and ranging (LIDAR)- based bathymetry is a highly specialized field within the widely known and used geoscientific surveying technology based on green spectrum lasers. Green light can penetrate shallow water bodies such that river and lake beds can be surveyed. The result of such observations are point clouds, from which geometries are extracted, such as digital elevation maps for lake, river or sea floors. The quality of those maps is crucially dependent on the amount of reliable information that can be extracted from the noisy LIDAR signals. The primary LIDAR data consists of amplitude response curves for each emitted laser signal. A direct visualization of these "raw", pristine data is crucial to verify, assess and optimize subsequent data processing and reduction methods. In this article we present a method for scientific visualization of these amplitude response curves *en mass*, i.e. for millions and tens of millions thereof simultaneously. As part of this direct visualization also preliminary analysis and data reduction operations can be performed interactively. This primary and direct inspection allows studying and evaluating the full potential of acquired data sets such that data processing methods can be fine-tuned to squeeze out all needed information of interest. Ideally such improved data processing avoids subsequent surveying when output from already measured data sets is improved, resulting in reduced economical and environmental costs.

Keywords: airborne LIDAR, visual analysis, data processing, geoscience, bathymetry, scientific visualization, big data, GPU shaders

1 INTRODUCTION

The rendering of large point clouds has been a topic of research for a long time [14] and has even been used as an alternative faster than traditional rendering methods based on triangular meshes. Once the data sets become larger than RAM, out-of-core methods [8] become mandatory and eventually a hierarchical representation is needed. High performance in rendering is tightly related to efficient usage of GPUs. With the advancement and availability of WebGL even browser-based rendering of arbitrarily large point clouds have become possible [10]. However, rendering performance drops significantly once the amount of data to be visualized per frame no longer fits onto GPU RAM. For such situations a purely CPU-based solution may scale better, as presented by [13] using balanced P-k-d Trees. Such a sorting method for points without memory overhead is similar to the space-filling curves [9] used in smoothed particle hydrodynamics for extremely large data [12].

While point clouds are often considered as the primary data source to derive geometries such as triangulate meshes off them, rather little attention is given to the origin of the point clouds itself. In many cases those point clouds are based on LIDAR data acquisitions and the provided point clouds are taken as-is. However, LIDAR technology does not produce point clouds per se, but the instruments measure a time series of amplitude signals from the reflected laser source. The point cloud used for further processing is derived from this primary data, see Fig. 1 and Fig. 2.

These sequences of amplitude modulations constitute the actual "raw" data from the LIDAR sensor, out of which the final point cloud has to be derived. This is usually done by identifying maxima in the amplitude modulation and outputting their geometrical location along the laser shot direction. As these received signal curves may very well contain multiple maxima, this identification will produce many geometrical points per

Plzen, Czech Republic.

laser shot. Finding only valid maxima within the noisy signal is a matter of signal processing artwork such that the quality of the results depends on the respective algorithm. While extensive work has been undertaken in this area [11], special cases still occur where an automatized algorithm may miss some relevant information. This is particularly of interest in cases of complex geometries such as vegetation, or for underwater LIDAR penetration, Fig. 1. By inspection of the final point cloud after data processing, it is impossible to find such missed features in the raw data, they only give hints of regions where more information would be desirable. Thus, a comprehensive visualization method that allows to investigate, study and explore the entire raw LIDAR data is a must. Only then it is possible to verify and optimize point extraction algorithms, such as to fine tune them and to squeeze out the last little bit of valid information from the already available and recorded data, e.g. in order to complement the representation of relevant object geometries in the resulting point cloud.

This primary data, the amplitude time series per laser shot, is larger by two orders of magnitude than the point cloud derived from them. Due to this huge amount of data amplitude signals have so far only been visualized individually. Here we present an approach for the massive visualization of *all* amplitude time series signals that are acquired from a LIDAR observation, a massive amount of information up to 100× larger than just point clouds. Furthermore, we emphasize the practical implications of improved point extraction based on LIDAR amplitude signal analysis especially for bathymetric (underwater) LIDAR data acquisition.

2 DATA STRUCTURES

Our data model of choice is the Fiber Bundle HDF5 "F5" model , which casts all data into a hierarchy of five (plus two optional) levels [3]. The premise is that nature is best described by a differentiable manifold [4]. This data model is very suitable for I/O and is directly compatible with GPU data structures such as vertex and index buffer objects. In particular, it maps easily onto the underlying Hierarchical Data Format V5 (HDF5) as it allows for random data access and partial file I/O. These features are usually not available via proprietary file formats for the raw data delivered by the LIDAR hardware manufacturers. Reading those native files may last several days, while after conversion into the F5 layout and storing as HDF5, reading them is reduced to mere seconds or milliseconds. The initial step is thus to convert the data from the respective proprietary, hardware-specific file formats into the open, application-independent HDF5 file format [5].

2.1 Review: The HDF5 File Format

Common data formats used in geoscience (e.g. LAS[1], Shapefile[2]) are often restricted to certain data fields and data types. The HDF5 file format provides more flexibility with the data layout. It is hierarchically structured similar to a file system, where folders are equivalent to HDF5 Groups and files equivalent to HDF5 Datasets. HDF5 files are not limited in size and number of objects. Datasets provided additional structures such as dimension and type information. Each of them may be compressed independently with various distinctly tune-able compression methods. The file format allows to construct compound data types like higher dimensional geometric vectors or points from simple, hardware-independent basic types and is therefore self-describing. Furthermore it offers important features such as partial data loading and random access, which is crucial for indexing operations relating data structures to each other even beyond files. Finally it is a free and open standard providing a high-level API with interfaces for C, C++, Java, Python and Fortran. In its most recent version it also offers remote data access via web services, thus enabling inspection of big data via the internet without modifying the main application.

2.2 Review: The F5 Data Model

The F5 Data Model bundles geometric entities, their topological properties and attributes into a combined, complex object that decomposes into the following layers:

1. **Slice** Combines all data related to a certain moment in time, or - in case of a time series - until the next sampling moment in time.

2. **Grid** Combines all data related to a certain geometric entity.

3. **Skeleton** Combines all data of a Grid that are related to a specific topological property of this Grid, for instance its vertices, edges, triangles, agglomerations thereof or hierarchical instances. A Skeleton defines an "index space" where each index refers to one such element of a topological space.

4. **Representation** Combines all data of a Skeleton that are relative to either other Skeletons or to charts that define coordinate systems. Basically, a Representation only makes sense in relation to the object that it refers to, for instance the Representation of triangles relative to vertices, or the Representation of vertices relative to Cartesian coordinates.

[1] https://www.loc.gov/preservation/digital/formats/fdd/fdd000418.shtml
[2] https://www.esri.com/library/whitepapers/pdfs/shapefile.pdf

5. **Field** Combines all numerical values that have a specific semantic meaning, for instance physical fields such as "temperature" or "velocity".

6. **(Fragment)** An optional 6^{th} level to optimize performance for big data by allowing to split a contiguous data set into many smaller fragments.

7. **(Compound)** An optional 7^{th} level to allow storing compound data types as distinct arrays instead of interleaved elements, i.e. offering the choice of using a layout of the kind XXXYYYZZZ instead of XYZXYZXYZ.

This data model is capable to handle geometrical objects such as triangular meshes, point clouds, curvilinear grids, uniformly rasterized images or time series within one universal framework. It also allows to specify relationships between such objects and hierarchical multi-resolution instances. The complexity of the model grows with the complexity of the data, simple cases are modeled with just the minimally required layers filled out.

2.3 Topology of LIDAR Signals

With a LIDAR equipment on board an airplane or drone (UAV), a laser source emits pulsed signals at a certain frequency and a receiver measures the intensity of the reflected light, Fig. 1. Both events are recorded with a time stamp at sub-nanosecond time resolution which allows geometrical reconstruction of the observation events (in one nanosecond light travels ca. 30cm). If

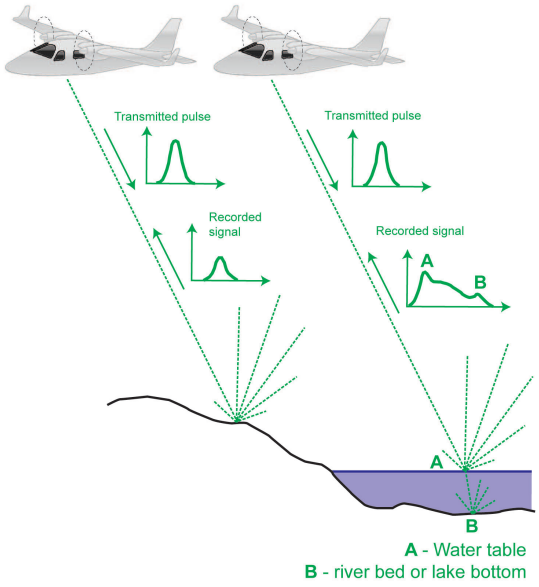


Figure 1: Data acquisition from airborne LIDAR mapping. The laser equipment emits signals at a certain frequency and measures the time when the reflected signal arrives at the airplane again.

the laser beam hits a mirror orthogonal to its view direction, then the received signal is an exact copy of the emitted pulse, yet subject to distortions happening in the detector itself. The location of such a mirror target can then be reconstructed from the time stamps within the precision of the clock. Eventually this location is assigned three-dimensional coordinates, forming one element of a point cloud that describes the entire observed geometry from the particular flight, as in Fig. 1.

Since actual geometries deviate significantly from an idealized perfect mirror, the actually received signal will be a superposition of many reflections that are detected at different times according to their reflections at different locations in space, Fig. 1 and Fig. 2.

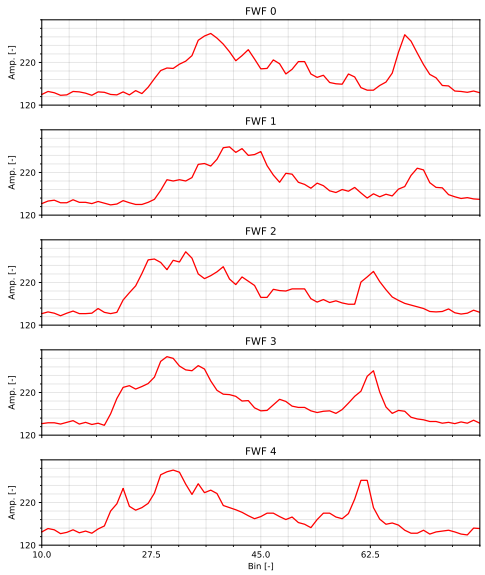


Figure 2: Received amplitude modulation over time for subsequent laser shots.

Within the context of the fiber bundle data model, data are identified via their base and fiber space. For LIDAR point clouds the base space is the geometry, the fiber space consists of all the attributes on them (such as color, intensity, time stamp, ...). Since each point in the final point cloud has been extracted from a specific amplitude modulation those time series can be assigned to each point as additional data, thereby expanding the fiber space without changing the base space. This has the advantage that the raw data can be investigated immediately for each identified point. These points have to reside at a maximum of the signal curve. However, if this point has been extracted from a local maximum, then other points will exist for the same signal curve, requiring duplication of the same curve onto all those points extracted from it.

Alternatively the base space can be considered as constructed from the sequence of “laser shots”. Each laser shot has been emitted at a certain time, forming a one-dimensional sequence with monotonously increasing timestamp. Using the flight path’s trajectory information geometrical 3D coordinates can also be as-

signed to each individual time stamp. Here, the amplitude modulations are fibers over the base space that is constituted by the flight trajectory. In this approach there is no need for data duplication, but the relationship between extracted geometries and the original raw data is missed. While superior with respect to data efficiency, this approach is thus less suitable to study the computational performance of the algorithm which is the ultimate objective of the visualization approach.

2.4 Massive Variable-Length Multiplicity

Multiplicity is a property of a field that tells its number of components. For instance, a scalar field has multiplicity 1, a coordinate field and a velocity field both have multiplicity 3, even though they have algebraically distinct properties. Within the classical fiber bundle data model as illustrated in Fig. 3, the multiplicity of a field must be constant for each point. This invariance of the multiplicity is essential for performance on I/O, parallelization and GPU processing as it directly fits with the requirements of vertex buffer objects. A field on a Grid's vertex Skeleton is just a vertex array in OpenGL. While this concept applies well to vertices and vertex-

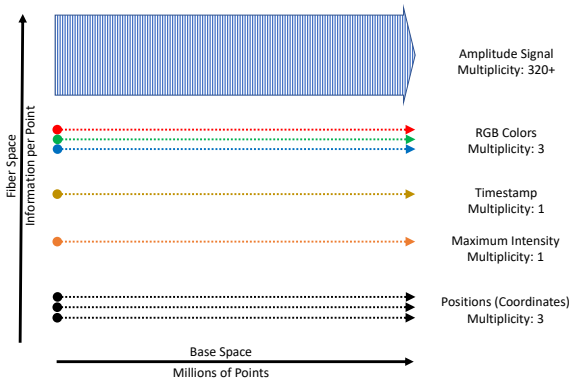


Figure 3: Fiber Bundle data layout for amplitude time series signals on point clouds: The amplitude signal is modeled as a 2D field given over a 1D base space.

related data as well as to homogeneous topologies such as triangular meshes, it fails to cover inhomogeneous cases such as mixed meshes constructed intermittently from quads, triangles, or arbitrary polygons (only after explicit triangulation they become homogeneous). They can be covered by extending the element type by a variable-length data type per point. Such data structures are possible both in C++ and in HDF5, but inefficient, both in terms of I/O and handling on the GPU, so they should be avoided. Staying at the classical fiber bundle model as defined above as close as possible guarantees better performance.

The LIDAR raw signal as received by the instrumental detector is a time series of the light amplitude over time which is different for each laser shot (Fig. 2). The number of sampling points ("bin") for this physically continuous signal varies from a few dozen to a few hundreds. The sampling interval in time was 0.57 nanoseconds for our hardware equipment, which corresponds to about 8.5cm in space considering the light travel time during that period, including double traversal time due to the reflection on the observed point (Fig. 1). Thus, when taking each single laser shot as the elements of a topological space within the framework of the fiber bundle model, then the field of the corresponding amplitude signal will have varying multiplicity (Fig. 2). In order to maintain high performance while not losing any potentially important information each signal is expanded to the maximally found length. While this approach blows up the raw data by adding zero values, it is still preferable over dealing with variable-length data. When storing data to disk, high-speed compression methods such as LZ4 are able to reduce disk storage requirements without performance penalties, and even performance gain [1].

2.5 Data Processing Pipeline

Prior to visualization, the raw data delivered from the scanner has to be processed. In this case the source data consists of the flight trajectory, the point cloud and the amplitude response. To speed up the subsequent processes all source data is imported to the F5 fiber-bundle format first. The second step is to transform the point cloud position from the scanner's internal coordinate system to a geographic one, in this case UTM coordinates. This also includes the direction vector tracing a point to the scanner's position at the time of echo signal detection. To visualize the amplitude response as described in the next section 3 a spacial point is needed to mount the amplitude response onto. One possibility would be to calculate the coordinate location of the start of the amplitude response: $\vec{o} + \vec{d}(c_a * (t_r - t_e)/2)$, where $\vec{o}$ is the origin of the laser pulse, $\vec{d}$ is the direction vector, c_a is the speed of light in air, t_e the time of laser pulse emission, t_r the starting time of received signal record, and division by two because the laser pulse travels back and forth). The advantage is that amplitude responses can be visualized and processed independently from the point cloud. Some scanners are storing the amplitude response apart from the point cloud. In this case the relation "points belonging to an amplitude response" is not available but must be recomputed if it is relevant for data analysis. This is done by calculating the instant of time where the laser pulse is reflected by a detected point ($t_e + d * 2/c_a$, where d is the distance between scanner and point) and search for the amplitude response with an overlapping time interval. This is a significantly time consuming process for millions of points. For better performance these correlations and intermediate values must be precomputed.

3 VISUALIZATION METHODOLOGY

3.1 Basic Functionality

It is natural to display a time series as a curve such as in the technical display of Fig. 2. For LIDAR signals, each amplitude of the time series has a spatial correspondence - namely a point with 3D coordinates - that allows for assigning the amplitude value to locations in space along the line of the respective laser shot. A

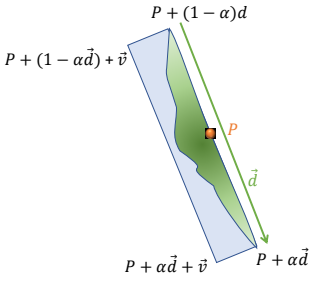


Figure 4: "Mounting" an amplitude signal to a point P, given the laser shot direction $\vec{d}$, at positional location $\alpha \in [0,1]$ and a billboard view direction vector $\vec{v} = \vec{d} \times \vec{c}$ with $\vec{c}$ the view direction from the camera.

billboard-like display does the job, where one orientation if the billboard is determined by the direction of the laser shot in 3D and the other one by the plane orthogonal to the view direction, Fig. 4. The coordinates of this laser-shot oriented billboard are easily computed by an OpenGL geometry shader, based on a vertex coordinate P with a vector vertex attribute specifying the direction $\vec{d}$, a scalar vertex attribute α (can be omitted if P refers to always the same position within the amplitude signal) and a uniform variable specifying the view direction (which is, for instance, available via the modelview matrix), as illustrated in Fig. 4.

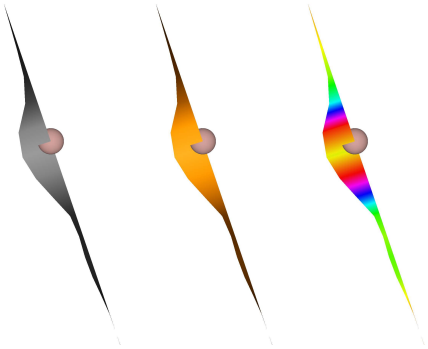


Figure 5: Visualization of a time series along the downwards direction of a laser shot with different color-coding of the amplitude value. The single point is located at the amplitude maximum, which is usually the only source of information constituting final point clouds.

The geometry shader thus "blows up" a single vertex into a screen-orthogonal quad and provides two-dimensional texture coordinates for its four corners. The first component of the texture coordinates correlates to the time index of the amplitude signal, normalized to $[0,1]$ (along $\vec{d}$ in Fig. 4). The second component corresponds to the geometrical distance from the precise line of the laser shot, also normalized to $[0,1]$ (along $\vec{v}$ in Fig. 4). It is then the duty of the OpenGL fragment shader to display the values of the time series based on those texture coordinates, interpolated by the hardware for each pixel, in a visually pleasing and insightful way similar to Fig. 2; but an opaque area - Fig. 5 - is preferable to a mere contour line.

3.2 Base Space Chopping

In order to access the - huge amount - of amplitude signal data per vertex, all these amplitude data need to be provided as a texture buffer to the GPU which extends the capabilities of the commonly used vertex attributes by orders of magnitude: An amplitude signal may encompass hundreds of entries, whereas vertex attributes are limited to the types supported by OpenGL, the largest one being a 4×4 matrix that could store at most 16 values. This one contiguous buffer[3] (OpenGL's `glTextureBuffer()`, accompanied by GLSL's `usamplerBuffer`) needs to be accessed as a two-dimensional array then, with one index given by the number of the vertex for which the fragment shader is called, the second index given by the texture coordinate along the direction of the laser shot.

However, while the size of a buffer is merely limited by the amount of total RAM available, the size of a texture buffer constrained to the extent given by `GL_MAX_TEXTURE_BUFFER_SIZE`. This maximally allowed texture buffer size is easily exceeding by the vast amount of amplitude signal information. To handle this constraint, the base space (Fig. 3) needs to be chopped into smaller pieces with the texture referencing a subset of the entire buffer for each such piece via `glTextureBufferRange()` with increasing offset, as illustrated in Fig. 6. This offset is required to be an integer multiple of the hardware-dependent value of `GL_TEXTURE_BUFFER_OFFSET_ALIGNMENT`[4].

Consequently, the base space cannot be chopped at arbitrary vertex indices, but only at index locations that are an integral multiple of both the hardware-dependent texture buffer offset alignment and the multiplicities of the fiber space. The "minimal chop size" is then determined by the *least common multiplier* of the alignment and the multiplicities. For instance, for a common alignment of 16, an amplitude signal length of 300 results in a minimal chop length of 1200, whereas a signal length of 320 results in a minimal chop length of 320 (which already is an integer multiple of 16). Of course, minimizing the number of rendering instances is mandatory, thus the largest integral multiple of this minimal chop length that still fits into the

[3] https://www.khronos.org/registry/OpenGL-Refpages/gl4/html/glTexBuffer.xhtml

[4] https://www.khronos.org/registry/OpenGL-Refpages/gl4/html/glTexBufferRange.xhtml

GL_MAX_TEXTURE_BUFFER_SIZE will be the one to be used such that all but the last chop will be of the same - maximally possible - size for the rendering instances. This is the effectively used chop length as illustrated in Fig. 6.

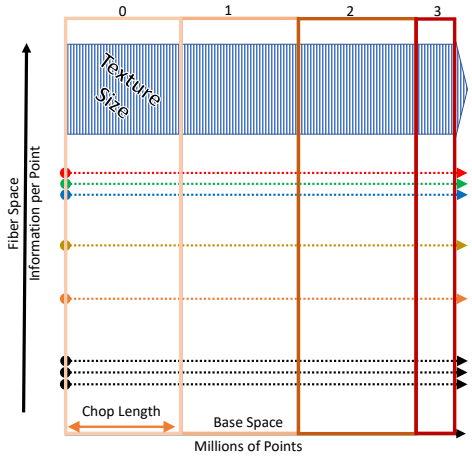


Figure 6: Chopping of the base space (per fragment) such to allow multiple rendering passes to operate on texture sizes that fit into the GPU hardware limits. The last render pass receives a partial texture.

The maximal number of chops per rendering instance is given by the GL_MAX_TEXTURE_BUFFER_SIZE divided by the minimal chop length, meaning that many chops can be rendered at once in a single rendering call. Thus, the number of rendering instances actually needed for the entire data set is given by this maximal number of chops divided by the total number of vertices. This will be a fractional number: Its integer part gives the number of "full-length" passes, the remainder part gives the last chop's length. Table 1 gives some exemplary numerical values for a typical data set totaling more than 2GB of raw, uncompressed data corresponding to a single fragment in the F5 data model.

Vertices	3 420 772
Amplitude signal length ..	320
Amplitude signal type ...	16-bit integer
Storage size	2GB ≈ 2 189 294 080
compressed disk space ...	200MB ≈ 201 920 475
OpenGL max. texture size	128M = 134217728
OpenGL offset alignment	16
Min. chop length (lcm) ..	320
Max. number of chops ...	419430 = 134217728 / 320
Total number of instances	8.1 = 3420772/419430

Table 1: Exemplary data for base space chopping using an NVidia GTX 1650 graphics card for an amplitude signal stored as 16-bit integers on about 3.5 million points.

3.3 Performance

It is evident that with data sizes of 2GB *per fragment* the available memory of smaller graphics cards are quickly reached even with medium-sized point clouds encompassing a few tens of million vertices. As the GPU drivers are able to trade CPU RAM for performance, the rendering rate quickly drops by a factor of 20 or more from 14 frames per second to less than 1 frames per second (data measured for an NVidia GTX 1650 GPU, utilized for laptops). Still, this makes rendering huge data at least possible even on less powerful hardware, while navigation within singular fragments - ideally the "most visible" one [2] - remains interactive. A more powerful graphics card such as a GTX 1080 GPU with 8GB of on-board graphics RAM utilizes 6.3GB for a larger dataset of 14 Mio points and can maintain rendering rates faster than 15fps throughout all navigation events.

IO and GPU transfer The main bottlenecks occur during data transfer at two stages, firstly from the disk to CPU RAM, secondly from CPU RAM to GPU RAM. HDF5 offers a wide choice of compression filters. The high-speed lossless LZ4[5] filter easily achieves data reading rates of more than 500MB/s. For massive data such as the 2GB required per dataset fragment for amplitude signals, this still results in noticeable 4 seconds. Even though this is a one-time effort only, such initial I/O operations significantly stall an interactive application and strongly justify asynchronous data loading in some subthread.

But also the second part of CPU-GPU data transfer done via OpenGL's glBufferData() call lasts 1-2 seconds for realistic rates of 1.5GB/s. Such behavior also results in noticeable "stutter" of a rendering application each time a new fragment is loaded. As of now we have not yet implemented asynchronous GPU data transfer [6] but such would be a next step for enhanced user experience.

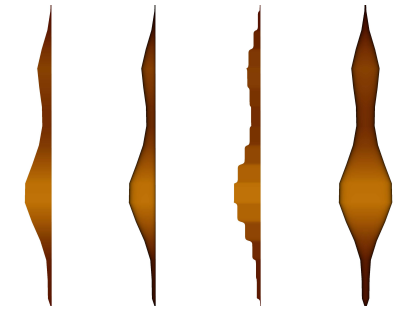
4 HANDS-ON DATA ANALYSIS

4.1 Visual Enhancement Techniques

The fragment shader provides various opportunities of enhancements for the most insightful visualization of LIDAR amplitude series among millions of combined renderings each of them containing complex information on their own. Some of these opportunities are envisioned first in (Fig. 7(a)) for a single LIDAR amplitude series before pointing out their meaning for the visualization of multiple LIDAR amplitude series as presented in Fig. 8.

Beyond the mere factual plain data display as in Fig. 7(a), contours (Fig. 7(b)) are easily added by overlaying the amplitude signal value of the pixel from the

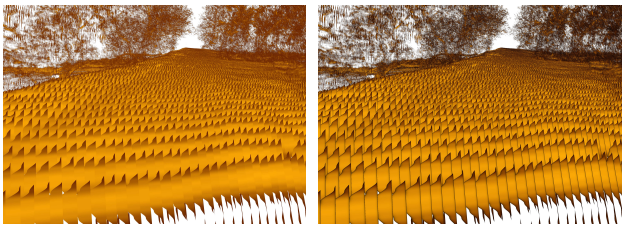
[5] https://lz4.github.io/lz4/



(a) plain (b) contours (c) bins (d) symmetry

Figure 7: Visualization options for LIDAR amplitude series.

laser shot axis with the color value. Once this technique is applied to massive data Fig. 8(a), the singular amplitude signals are exposed with much better contrast Fig. 8(b). The interpolation of the signal bins, i.e. the texel access, is a potential tuning parameter as well such to change linear interpolation to nearest neighbor Fig. 7(c) as means to more accurately display the exact data values instead of a smooth visualization. In some situations displaying the amplitude signal symmetrically as in Fig. 7(d) is more appropriate since the choice to limit the visualization to the "left side" is entirely arbitrary.



(a) Massive Plain Rendering (b) Massive Contour Rendering

Figure 8: Visualization of a series of amplitude signals representing terrain surface and vegetation (trees) in the background using differing shading enhancements as described in the text. Colorization by amplitude replicates the geometric shape. 8(a) plain rendering, 8(b) contour rendering.

4.2 Feature Detection

The ultimately outcome of analysing a cloud of amplitude signals is the production of a cloud of points that correspond to physical objects. Such data reduction functionalities can be built into the fragment shader to provide immediate, real-time interactive assessment capabilities to the raw data as shown fully in Fig. 8. Since the fragment shader has full access to the texture describing the amplitude at each point, it can also compute the gradient of the signal. Thus, a color value can be limited to pixels where the sign changes for the texel left and right, i.e. indicating a local maximum at each of the pixels in Fig. 9 (same data as Fig. 8). In practice, the recorded amplitude series come with noise

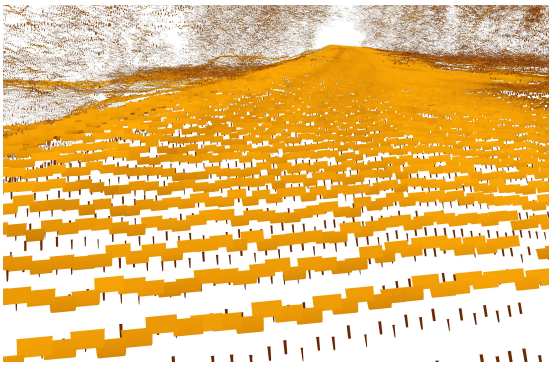
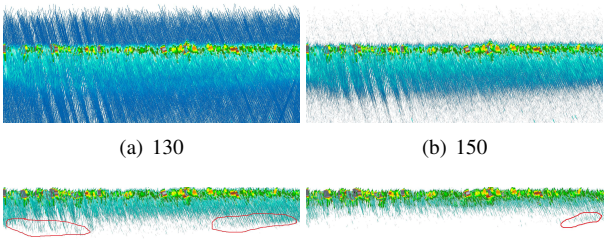


Figure 9: Visualization of same series of amplitude signals as in Fig. 8, but display of pixels restrained to bins containing a local maxima.

such that displaying any local maxima as shown in Fig. 9 results in undesirable visual clutter (Fig. 10(a)). A simple thresholding by user-specified noise level already improves the situation significantly, as shown in Fig. 10(c) and Fig. 10(d), where the river bed becomes distinguishable from the waterbody in the left and right part of the cross-section. Of course, more complex data



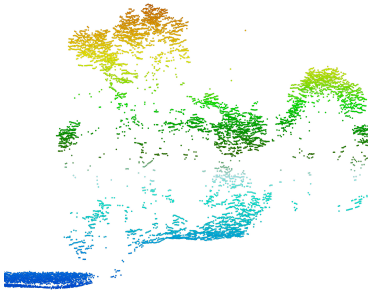
(a) 130 (b) 150

(c) 180 (d) 200

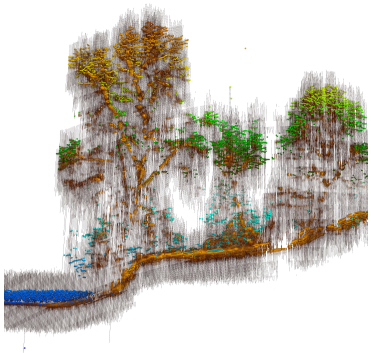
Figure 10: Cross-sectional view of a series of amplitude signals through a waterbody demonstrating the effect of realtime noise reduction on bins containing local maxima only, using increasing threshold values for data cutoff.

analysis and feature extraction methods can be incorporated into the fragment shading step, but come at the cost of reduced rendering speed. Simple but fast analysis parameters may already prove useful for subsequent data analysis routines that are performed on the GPU. For instance, Fig. 9 depicts the global maxima (band of broad stripes) followed by local maxima (band of thinner stripes). Such local maxima are not necessarily physical targets, but may be the result of the signal detector's response curve exhibiting overshooting and ringing behaviors. The mathematically correct approach to remove these artifacts is via deconvolution [7] with the perfect signal response of a delta function. However, they drastically increase the noise level numerically . Performing a full deconvolution at each frame rendering is unreasonable, but can be done as a one-time preprocessing step with the presented visualization method as verification and data assessment tool.

The immediate visualization of raw amplitude signal series from LIDAR datasets alone already allows for a direct and detailed exploitation of their potential. This is specifically emphasized in Fig. 11, where different amplitude visualization and point extraction methods are combined and compared in order to improve feature detection results in vegetation and terrain mapping (Fig. 11(a) compared to Fig. 11(c)).



(a) Maxima Points, colored by altitude



(b) Points + Full Amplitude

(c) Points + Noise cutoff at 150

Figure 11: Improvements of feature detection for vegetation and terrain mapping shown for a cross-section through a densely vegetated river foreland area. The combined visualization of originally digitized 3D points at data acquisition Fig. 11(a) and signal amplitude maxima Fig. 11(c) indicates that feature recognition can be substantially improved by further amplitude signal processing.

4.3 Water Clarity

Beyond point cloud extraction and identification of physical objects, the full amplitude signal provides more information that is not even of geometrical nature. Particularly, when observing lakes or rivers with a green laser source capable of penetrating the water body, the attenuation coefficient becomes measurable. The attenuation coefficient can be viewed as a proxy for determining the water clarity respectively water turbidity as an areal measurement. Such measures are important specifically for quantifying active fluvial sediment transport processes, or for habitat modeling in rivers and lakes and assessing their ecologic state.

The Beer-Lambert law is valid in the linear regime where the material is not highly scattering and states that the optical attenuation $I(s)/I_0$ is an exponential function of the path length s through the medium: $I(s) = I_0 e^{-\kappa s}$ with κ as the attenuation coefficient. This attenuation coefficient is constant for a fluid with homogeneous scattering properties. This exponential decay of the reflected light intensity along the path of the ray is immediately suitable for analysis from the amplitude signal.

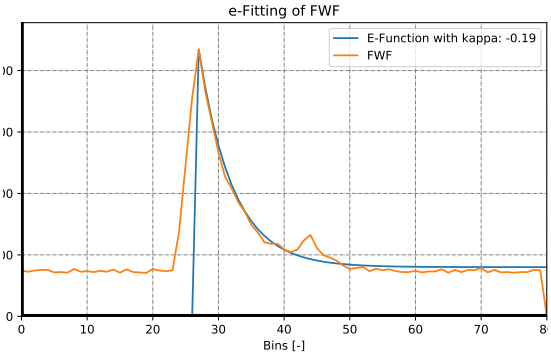
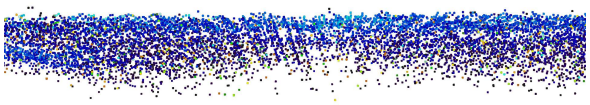
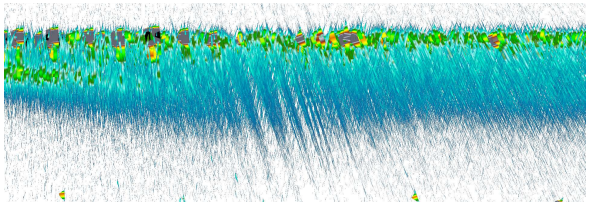


Figure 12: e-Fitting

A point cloud with reflectivity information contains this information as well, but provides only non-equidistant, lower spatial resolution along the ray of penetration (Fig. 13(a)). The information of subsequent connected data values is instead available from an extracted point cloud, whereas it is primarily available in the full signal (Fig. 13(b)). Actual amplitude signals are noisy, so fitting an exponential curve to each individual curve is needed to determine the attenuation coefficient for each laser shot (Fig. 12). Such analysis is too time-consuming to be performed at the rendering stage - which is aiming toward running at multiple frames per second, but it makes sense as a pre computation step. In future these pre-computed attenuation coefficients may be used at the visualization stage to subtract an exponential bias curve in order to enhance peaks above the exponential decay, instead of simple noise thresholding as in Fig. 10.



(a) Point Cloud of a water body



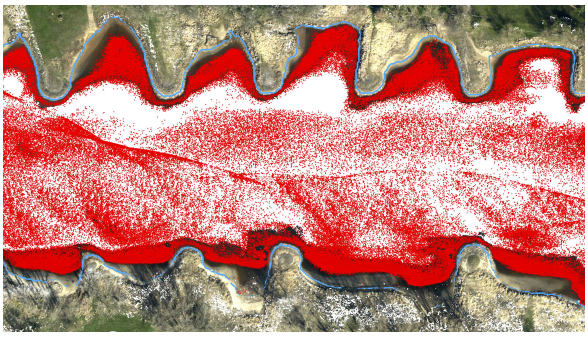
(b) Myriads of Amplitude Signals

Figure 13: Cross-section through a waterbody. (a) originally digitized 3D points at data acquisition colored by reflectivity decreasing from water surface (light blue indicating higher reflectivity) to depth (dark blue equaly lower reflectivity). (b) Associated amplitude signal series showing signal fade out with water depth with largest maxima at water surface and water ground (green to red color).

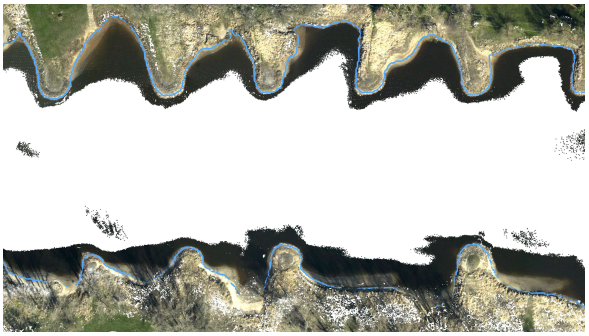
5 RESULTS

The presented visualization enhancement approaches of raw LIDAR amplitude signals are of great advantage when analyzing complex topobathymetric datasets, because they allow for direct and fast data access without time-consuming data post-processing efforts. Especially, the full amplitude visualization exposes river bed areas where the point cloud shows no coverage Fig. 15(a). Displaying local maxima also shows those signals on the water surface that triggered the original point cloud (compare Fig. 15(c) with Fig. 15(a)). Moreover, full amplitude visualization can also reveal terrain coverage below dense vegetation where the original point cloud indicates no coverage (Fig. 11). In practice, these two findings are of particular relevance to improve terrain extraction from LiDAR point clouds above and below water, and thus to significantly improve the quality level of survey data, which are already of high quality when considering only the original point clouds Fig. 14.

Further insights on water conditions in terms of clarity respectively turbidity derived from signal analysis can be of relevance for determining sediment transport processes in rivers related to floods or during snow melt. We are currently evaluating the potential of signal analysis for automated substrate mapping in water areas in terms of grain-size distribution along the river bed and surface roughness, which are verified by ground truth data.



(a) Enhanced Coverage



(b) Original Coverage

Figure 14: Comparison of a river bed based on amplitude signal analysis versus immediate LIDAR point cloud data.

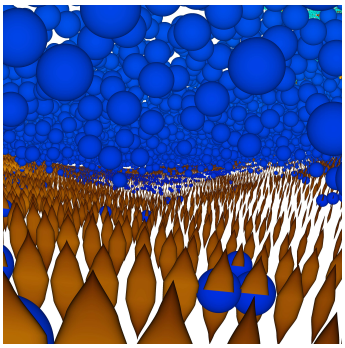
6 CONCLUSION

In this article we presented a rendering method for the direct, immediate visualization of massive raw amplitude response curves from LIDAR data acquisitions. The method provides intuitive insight into the potential of the data sets to allow for optimizing further data processing steps and potentially reducing the actual number of observation flights needed for airborne LIDAR bathymetry. Simple data processing steps can be performed interactively in realtime. As per our knowledge this is the first time that a direct, simultaneous visualization of dozens of millions of LIDAR amplitude response curves has been done. Its formulation within the fiber bundle model ensures a rigid mathematical basis yielding high performance independent of the application domain.

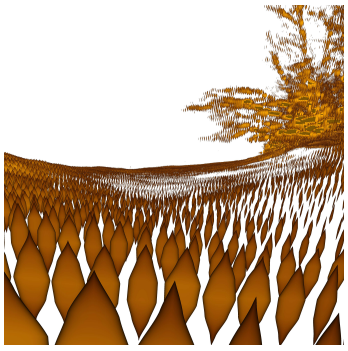
Our visualization technique has potential beyond LIDAR and can be applied to e.g. multispectral imaging with hundreds of spectral channels as well. Furthermore, the technique straightforwardly extends to other topological properties than vertices, e.g. edges or triangles, and can thus be applied also to high dimensional data given on more complex geometries.

REFERENCES

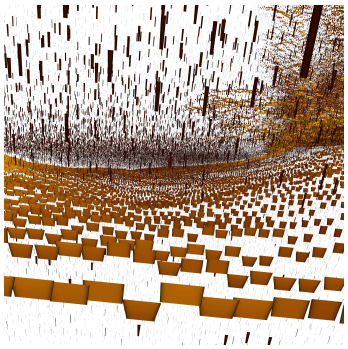
[1] F. Alted. Why modern cpus are starving and what can be done about it. *Computing in Science and Engg.*, 12(2):68–71, Mar. 2010.



(a) Point Cloud + Amplitude Signals



(b) Amplitude Signals



(c) Signal Maxima

Figure 15: Detailed view of a topobathymetric LIDAR dataset looking from a position at the river bed towards the water surface. (a) combined visualization of originally digitized 3D points at data acquisition (blue) and associated full amplitude signal in symmetric display (compare to Fig. 7(d)). (b) Visualization of amplitude signal series without points. (c) Visualization of amplitude signal maxima (compare to Fig. 9).

[2] W. Benger, D. Hildenbrand, and W. Dobler. Optimizing refined geometric primitive‘s leaflet visibility for interactive 3d visualization via geometric algebra. In *Proceedings of Computer Graphics International 2018*, CGI 2018, pages 267–272, New York, NY, USA, 2018. Association for Computing Machinery.

[3] W. Benger, M. Ritter, S. Acharya, S. Roy, and F. Jijao. Fiberbundle-based visualization of a stir tank fluid. In 17^{th} *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, pages 117–124, 2009.

[4] D. M. Butler and M. H. Pendley. A visualization model based on the mathematics of fiber bundles. *Computers in Physics*, 3(5):45–51, sep/oct 1989.

[5] T. Habermann, A. Collette, S. Vincena, J. J. Billings, M. Gerring, K. Hinsen, W. Benger, F. R. Maia, S. Byna, and P. de Buyl. The hierarchical data format (hdf): A foundation for sustainable data and software. *2014 AGU Fall Meeting*, 2014.

[6] L. Hrabcak and A. Masserann. Asynchronous buffer transfers. In P. Cozzi and C. Riccio, editors, *OpenGL Insights*, chapter 28, pages 391–414. CRC Press, July 2012.

[7] J. Pfleiderer. Methods of deconvolution. In W. C. Seitter, H. W. Duerbeck, and M. Tacke, editors, *Large-Scale Structures in the Universe Observational and Analytical Methods*, pages 298–305, Berlin, Heidelberg, 1988. Springer Berlin Heidelberg.

[8] R. Richter, S. Discher, and J. Döllner. *Out-of-Core Visualization of Classified 3D Point Clouds*, pages 227–242. Springer International Publishing, Cham, 2015.

[9] H. Sagan. *Space-Filling Curves*. Universitext. Springer-Verlag New York, 1994.

[10] M. Schütz. Potree: Rendering large point clouds in web browsers. Master's thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/E193-02, A-1040 Vienna, Austria, Sept. 2016.

[11] R. Schwarz, G. Mandlburger, M. Pfennigbauer, and N. Pfeifer. Design and evaluation of a full-wave surface and bottom-detection algorithm for lidar bathymetry of very shallow waters. *ISPRS Journal of Photogrammetry and Remote Sensing*, 150:1 – 10, 2019.

[12] V. Springel. The cosmological simulation code gadget-2. *Monthly Notices of the Royal Astronomical Society*, 364(4):1105, 2005.

[13] I. Wald, A. Knoll, G. P. Johnson, W. Usher, V. Pascucci, and M. E. Papka. Cpu ray tracing large particle data with balanced p-k-d trees. In *Proceedings of the 2015 IEEE Scientific Visualization Conference (SciVis)*, SCIVIS 15, pages 57–64, USA, 2015. IEEE Computer Society.

[14] M. Wimmer and C. Scheiblauer. Instant points: Fast rendering of unprocessed point clouds. In *Proceedings of the 3rd Eurographics / IEEE VGTC Conference on Point-Based Graphics*, SPBG 06, pages 129–137, Goslar, DEU, 2006. Eurographics Association.