ISSN 2464-4617 (print)
ISSN 2464-4625 (online)
Computer Science Research Notes - CSRN 3301
http://www.wscg.eu
WSCG 2023 Proceedings

The Method of Mixed States for Interactive Editing of Big Point Clouds

Werner Benger
Airborne HydroMapping GmbH
A-6020 Innsbruck, Austria
w.benger@ahm.co.at
Center for Computation & Technology
Louisiana State University
Baton Rouge, LA-70803

Anca Voicu
Prowasser
C-tin Brancoveanu-64, Timisoara, Romania
anca.voicu@prowasser.ro

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

Loredana Gonciulea
Prowasser
C-tin Brancoveanu-64, Timisoara, Romania
loredana.gonciulea@prowasser.ro

Cosmin Barna
Prowasser
C-tin Brancoveanu-64, Timisoara, Romania
cosmin.barna@prowasser.ro

Frank Steinbacher
Airborne HydroMapping GmbH
A-6020 Innsbruck, Austria
f.steinbacher@ahm.co.at

ABSTRACT

We present a novel methodological approach for the interactive editing of big point clouds. Based on the mathematics of fiber bundles, the proposed approach to model a data structure that is efficient for visualization, modification and I/O including an unlimited multi-level set of editing states useful for expressing and maintaining multiple undo histories. Backed by HDF5 as high performance file format, this data structure naturally allows persistent storage for the history of modification actions, an unique new feature of our approach. The challenges of visually based manual editing of big point clouds are discussed and a proper rendering solution is presented. The implemented solution and its features as consequences of the underlying methodology are compared with two major mainstream applications providing point-cloud editing tools as well.

Keywords: point clouds, interaction, classification, data editing, fiber bundle data model, undo history

1 INTRODUCTION

Correct and accurate classification is an essential step in the LiDAR (Light Detection And Ranging) point cloud processing. More specifically, the classification of ALB (Airborne Laser Bathymetry) data focuses primarily on the definition of terrain and water surface points. The latter is required for the final refraction and runtime correction of points lying beneath the water surface. According to the summary of [LG17], morphological (e.g., [Sit01, MWSCC09]) and surface (e.g., [KP97, LKS00]) based filters and their extensions/variants are among others utilized for the terrain detection. All described approaches solely represent automatically calculated classification results. Considering 3D data and their interpretation, recent algorithms cannot replace the human capability of cognitive abstraction and anticipation. Thus, a manual inspection and editing procedure of automatically classified point clouds is crucial in order to correct erroneous classification results if required, and to ensure data quality. The quality of automated ALB point classification results strongly depends on the overall raw data quality, which is mainly influenced by weather and water conditions during the survey, and on the general morphological appearance of a project area. A high humidity during an ALB survey results in increased flaw echo detection, and such noisy points need to be separated from the actual points of interest (Fig. 1). Further factors reducing the quality of the automated classification are:

- the terrain complexity, e.g. high mountainous relief vs. uniform flatlands [CZWea13];
- a dense vegetation canopy with shadowing effects reducing the actual terrain coverage [WSB$^+$12];
- high water turbidity limiting water penetration [Man20];
- white water areas impeding water-ground detection;
- water-bottom material consisting of substrates with increased light absorbing characteristics (e.g., underwater plants, organic matter mixed with bed material) hampering water-ground detection[Man22].

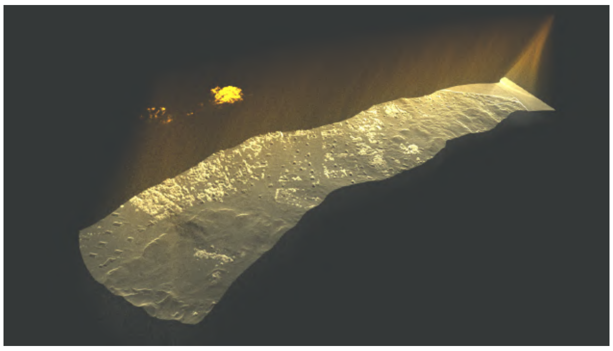


Figure 1: ALB scan strip with flaw echoes (yellow to orange) and relevant point data (white) that can be clearly separated by visual inspection.

Thus, the manual correction of the classification can be highly time consuming depending on these conditions as well as the spatial extent of the project area. To minimize the manual editing effort on one hand, recent software development is focused on the improvement of the automatic classification of ALB data by incorporating and combining further raw data attributes and derived geometric parameters into the classification process (e.g. [SDB$^+$21]) , or utilizing machine/deep learning approaches for specific classification purposes (e.g. [HEA$^+$21]). On the other hand, an efficient manual point-cloud editing toolkit together with a fast and flexible point-cloud visualization and navigation regardless of file size are important prerequisites for minimizing manual correction efforts on classification results. Manual editing tools are often standard components of various available LiDAR software packages, such as Terrasolid by Terrasolid Ltd., RiProcess by RIEGL LMS, or LAStools by rapidlasso GmbH. In this paper, however, we present the manual point cloud editing tools integrated in HydroVish, and how we optimize therewith the general ALB point-cloud processing.

This article's structure is: Sec. 2 reviews the foundations of the visualization environment, the data organization model and the underlying file format; Sec. 3 discusses the aspects of user-friendly and versatile selection of regions within a point cloud; Sec. 4 focuses on persistent storage of an unlimited undo history for big datasets while sustaining interactive performance; finally, section Sec. 5 presents a comparison of our implementation with two existing external applications.

2 PREREQUISITES

The work as presented here is implemented in the Vish Visualization Shell [BRH07], a general-purpose framework for visualization algorithms. Its specialization to bathymetric datasets, HydroVISH, is used in production by AHM GmbH[1] for large point clouds acquired by high-resolution airborne observations.

[1] www.ahm.co.at

2.1 Visualization Pipeline & Networks

Haber & McNabb [HM90] described a conceptual visualization process with three basic stages, transforming raw data into displayable images. These steps occur in most visualization processes and aim to convert data into information while maintaining the integrity of the content with best accuracy.

Modular visualization environments turn these stages into component parts that can be connected at runtime such to allow customization for a particular task. This type of visualization system is very widely used by the scientific community for its flexibility and programmability. The connections between visualization components may be set up via scripts or graphically via a user interface that allows to build a network representing the data flow. This approach of "visual programming" does not require any coding capabilities and is quickly intuitive to end-users. Beyond the underlying data flow also the control flow — the interaction of a user interface with steering parameters influencing the data processing — is important for flexibility and configurable user experience. For instance, the ability to couple two buttons with the same hotkey or mouse click per user-driven configuration allows for personalized preferences on how editing appears most convenient for a particular experience.

2.2 The F5 Fiber Bundle Data Model

The mathematics of fiber bundles provides a framework to model data for scientific visualization [BP89, Ben04]. This concept considers data sets based on the properties of their "base space" versus its "fiber space". A fiber bundle is basically a set of points with neighborhood information and equally-sized data sets attached to each such point. For instance, a multidimensional homogeneous array constitutes a fiber bundle in the mathematical sense. The F5 data model [Ben09] builds upon this concept by grouping all dataset properties with the same number of elements, thereby defining "index spaces" . Any suitable dataset is dissected into a hierarchy of five levels according to its properties:

1. *Time* (temporal slicing),
2. *Grid* (geometrical entity),
3. *Skeleton* (topological property),
4. *Representation* (coordinates, relationships) and
5. *Field* (binary representations of numerical values).

An additional optional sixth level allows to split up a *Field* into a set of named *fragments* (chunks of data contiguous in memory) for easier handling of large datasets. Each of these fragments may come with its own size, but all fragments of the same name must be of identical size within the same *Skeleton*.

The bottom line is the ability to handle datasets based on explicit properties instead of a set of implicit built-in assumptions: the model answers the question "how is it?" about a dataset instead of the question "what is it?". Algorithms need to be implemented in a way such to only request relevant properties of a dataset rather than the "type" of a dataset. This approach allows to cover a wide range of data categories using the same software infrastructure. Point clouds are a rather simple subset of this general framework, but by virtue of the fiber bundle data model the presented editing framework immediately applies also to other data types such as triangular meshes or line sets.

2.3 The HDF5 File Format

The hierarchical file format v.5 (HDF5 [HDF23]) is a general-purpose, self-descriptive open file format designed for high performance computing. It resembles a "file system within a file" with many features beyond an actual file system. For instance, structured data such as multidimensional arrays and user-defined compound types are supported natively. A wide range of data compression algorithms is available via a plugin system which allows optimization for different application domains and scenarios. The hierarchical data organization as used for the F5 data model from section 2.2 is very suitable to be directly mapped onto an HDF5 file.

3 ON-SCREEN POINT SELECTION

The point selection tool allows to draw an outline on the screen which is then projected into 3D space for selecting actual points. It provides multiple functionalities:

1. Polygon shape: each mouse click adds another point, points are connected via straight lines.
2. Free-Hand shape: While the mouse button is pressed, points are added to the shape while the mouse moves.
3. The shape can be stored and loaded, either as part of the visualization network which contains the state of all parameters of the current setup, or explicitly as a polygonal set of points in various file formats.
4. The shape can be dragged along the screen, similar to a "brush" in Photoshop™.

The drawing tool provides output actions, which by utilizing the capabilities of the visualization network, can be configured to perform different data editing actions according to the choice of the user:

1. Without configuration, drawing an on-screen selection requires an explicit button to be clicked to select points. This mode is useful if the user wants to carefully draw some shape first before applying a selection.
2. The selection can be performed immediately at each change of the shape, working for both adding more points to the outline shape (polygon mode, free-hand mode), as well as for dragging a fixed shape on the screen ("brush mode").
3. The selection can be performed when the outline drawing is "closed", i.e. the "last" point of an outline has been drawn and the outline is cleared such to start a new draw operation. This is usually done with a modified mouse-click, such as alt-mouse or right-mouse such to differentiate this operation from the shape drawing. This mode resembles the painting of polygons or free-hand forms on a white board, but in this case selects points within the point cloud.

3.1 Masked Editing via "Dots"

An additional level of security is given by displaying the what-if of a data editing operation, i.e. before actually performing the data modification immediately. This mode of editing is similar to utilizing a "selection" in Photoshop™ in order to limit some filter operation on a photo to this selected region. Similarly here we first mark - and visually enhance - the set of points that are intended for subsequent modification. This mask of points can be modified, like adding or removing parts of a selection, before an "apply" action (triggered by a button in the GUI, a hotkey, or a certain mouse even) modifies the actual data.

Per-point color attributes are sensitive pieces of information that already convey important properties such as RGB photographic data, height information, labels information (as elaborated in Sec. 3.2), etc. or an combination of those. In particular we display dots not as singular pixels on the screen, but via extended geometries resembling little spheres. We call these "dots" to distinguish them from single-pixel display methods. This "dot" display allows for highly precise study of fine details in the point cloud with intuitive depth perception; this is not possible by displaying all data points as single pixels. We explored several methods on to display markers on these "dots" without impacting their ability display of basis attributes, as demonstrated in Fig. 2. Some of the possible choices may be due to personal, aesthetic reasons, but there are also constraints as the choice influences the appearance when zooming out: as points shrink in screen-space, the marker information may get lost once point size approaches a single pixel - which is unacceptable when the conveying the selection information is important. Thus, special care must be taken for the modified rendering information, for instance using a view-distance-dependent fraction of the marker information versus basis information such that the marker information becomes dominant on overviews, as demonstrated in Fig. 3. In order to compensate for small dots (in screen-space) such that these

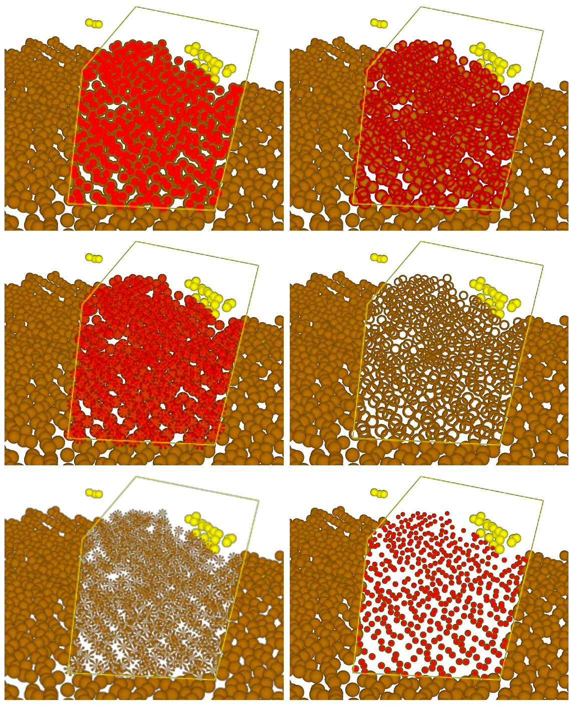


Figure 2: Displaying selections as per-point attribute independent of underlying colorization: colorized inner core, colorized outer rim, colorized sections, transparent core, transparent sections, size adjustment.

will be rendered more like non-circular dots in an effort to counter anti-aliasing (which would hide those) we employ OpenGL's `fwidth()` function to consider the screen-space derivative of a dot's texture. In the GLSL shader this compensation works as follows:

```
in vec2 SplatTexture;
in float Mask;
in vec4 MaskColor, PointBaseColor;
uniform float Threshold, MaskRadius;

float   R2, T, dR_dPixel;

R2 = dot(SplatTexture,SplatTexture);
dR_dPixel = fwidth(R2);
T = 1.0-R2;
T +=.5 *dR_dPixel*dR_dPixel;
T = clamp(T,-1.0,1.0);

if (T <0.0) discard; // Make dots round.

if (Mask > Threshold &&
    R2+dR_dPixel > MaskRadius )
        color = MaskColor;
else
        color = PointBaseColor;
```

`MaskColor` is the color for the marked regions in modification points colored by the `PointBaseColor` (which may be true RGB colors from observations, height maps, intensity maps or any other color attribute). Global parameter `MaskRadius` allows to fine-tune the visibility of the mask, a value of 0.5 values masking and colorization equally. Per-point attribute `Mask` defines a value between 0.0 and 1.0 specifying the strength of the mask; for a boolean mask, those values will be either 0.0 or 1.0; global parameter `Threshold` determines at which strength the mask should be displayed at all (0.5 per default). Selections can therefore be "hard" or "fuzzy", which can be useful when e.g. assigning RGB color values in an airbrush-like manner.

OpenGL point sprites receive texture coordinates in variable `SplatTexture` for each pixel in the range $[-1,+1]\times[-1,+1]$. The code computes a radial distance from these and discards all fragments beyond a constant distance, effectively creating round dots on the screen from the rectangular point sprite. This radius is adjusted dependent on the size of the point sprite on screen such that smaller sprites have less pixels cut off, thus appear larger in relation. The same mechanism is applied to the section of the dot that is colorized with the mask indicator - thus smaller sprites are weighted stronger and appear more prominently. The size of the sprites per point is determined by the previous geometry shader (code not shown here) based on view distance; thus more distant dots that got marked remain visible when zooming out.

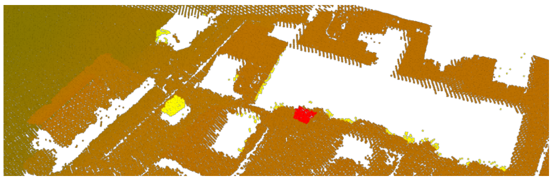


Figure 3: A good point-wise selection information must display information also when "zooming out" in overview mode.

Just making points transparent does not work easily within a three-dimensional scene as this would require depth-sorting of points or an equivalent technique suitable for millions of objects - very likely impacting performance, so we favored to use techniques without any such overhead. For editing a photo, a mask can be displayed by some two-dimensional overlay, but for editing three-dimensional point clouds a simple per-point overlay is insufficient because points in the foreground may hide points in the background - a situation that cannot occur when editing photos. A possible way to address the visual clutter is achieved by shifting marked points in screen space towards the observer, thereby "boosting" their visibility over other points (similar to OpenGL's `glPolygonOffset()` function), as demonstrated in Fig. 4.

3.2 Label-Constrained Editing

Labelling points by assigning integer numbers (representing specific meanings) to each point is the result

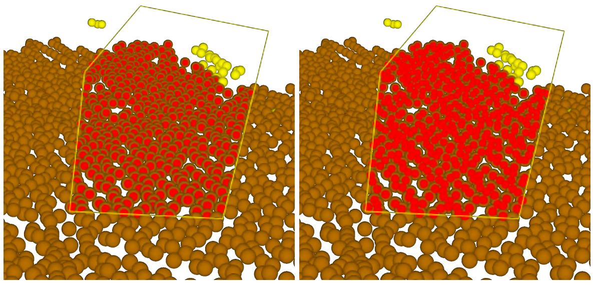
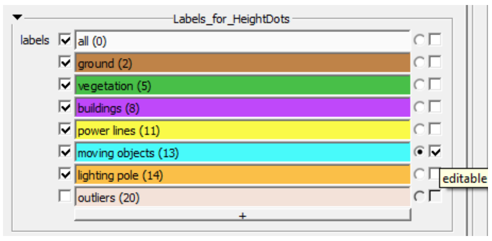


Figure 4: "Visibility Boost": enhance the visibility of marked dots to avoid visual clutter.

of a classification process to identify objects in raw data. Manual correction of automatic pre-classification is enhancing accuracy and providing the essential input data for refined training of machine learning algorithms. With such pre-classified data sets only some points usually need to be re-labeled; it is thus desirable to define sets of labels that should be subject to editing whereas other sets of labeled points shall remain unmodifiable, as demonstrated in Fig. 5.



Labels_for_HeightDots

labels				
✓	all (0)		○	☐
✓	ground (2)	🟫	○	☐
✓	vegetation (5)	🟩	○	☐
✓	buildings (8)	🟪	○	☐
✓	power lines (11)	🟨	○	☐
✓	moving objects (13)	🟦	●	✓
✓	lighting pole (14)	🟧	○	☐ editable
☐	outliers (20)		○	☐
	+			

Figure 5: GUI element to control label-constrained editing: visibility checkbox, description, colorization, integer value, selected assignator, write-property.

3.3 Field-Based Editing

It is rather straightforward to also consider any data field – i.e., point attributes – for constrained editing, including per-point scalar values with global or local threshold or range constraints such as height information ("only allow modification of points beyond 834m elevation above sea level") or point neighborhood information such as planarity [RBC$^+$12] ("only allow modification of points that reside within a plane given a minimal deviation tolerance").

Geometry-Constrained Editing Geometry is per-point coordinate information and can be constrained by an axis-aligned bounding box, or any other formula implementing an inside/outside check of a volumetric region. In practice, having the ability to define an axis-aligned bounding box, as demonstrated in Fig. 6, handles the systematic manual traversal of an extended data volume. Such a selected geometric region can, but does not necessarily have to, correspond to the geometric properties of data fragments. An additional geometric constraint can be defined by specifying a depth range as seen from the observer, thus effectlively defining a cross-section for editing.

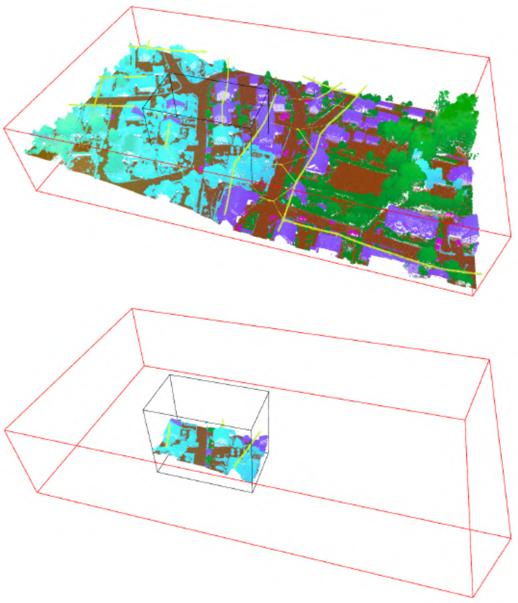


Figure 6: Example of a bigger point cloud (red bounding box, top) and a smaller tile of this point cloud (blue bounding box, bottom) in HydroVish. Editing can be restricted to arbitrary tile shapes with a shiftable box selection allowing to systematically traverse the entire volume in regular tiles.

Editing Coordinates Coordinates are another data field that can be subject to an editing operation, thereby enabling modification of a point cloud's geometry, just as in CAD applications. Furthermore, as long as the number of point remains constant, also additional properties of an underlying geometric object are unaffected: if the connectivity information within a triangular mesh is not touched, then the point cloud editor is directly applicable to editing surface geometries as well.

Editing Colors Color information are RGB values per point; an editor may modify such attributes instead of integer labels, resulting in the same functionality of a Photoshop™ painting with a brush on a photo - but now painting on a three-dimensional point cloud, or even triangular mesh. However, in contrast to modifying label information a smooth transition from current value to new value may be desirable in this case, i.e. using a "soft" instead of a "hard" brush. This feature can be realized by computing the distance from each point in a selection to the boundary of the defining brush shape and weighting the resulting color accordingly.

4 UNDO OPERATIONS

The common approach to undo actions is a global history of states that allows to go "backward", and sometimes also to go "forward" (redo operation). Undoing an undo action is equivalent to a redo action.

4.1 Three Level Handling

In our context undo/redo operations act on

1. drawing a free-style shape on-screen,
2. applying an on-screen selection to a mask,
3. modifying point attributes by a given mask.

All these operations are inherently independent and thus have their own undo history. It may be an option to merge all actions with a time stamp into a global history such to conform to the "standard" approach. However, as the application is capable to also perform more than single editing approach, such as editing entirely independent data sets within the same view, mixing such histories would disable the ability to treat each editing independently. Even with a single data set, the ability to undo data modifications immediately, without bothering to go through the masking and lasso history, is beneficial in speeding up practical work. Both the masking and lasso history are hardly ever needed. The three types of undo are useful with different relevance depending on the respective editing scenario.

4.2 Fragmented Data & Mixed States

To allow undo and redo operations a history of data states must be remembered. The "brute-force" aproach would be to always keep a copy of the entire data set after each operation. While this is the simplest way to implement a history, it is undesirable for large data due to performance and memory requirements. A more elaborated approach is to remember only the differences between two data modification states under the assumption that the majority of data remains unchanged. However, this comes with an additional computational effort, and computing differences of the entity of a big data set is inefficient as well.

In our context of the F5 data model as presented in Sec. 2.2 all data are split up into fragments such that each fragment contains only a few million points (out of hundreds or thousands of millions in its entity). Only those fragments that are visible as part of an editing operation are loaded into RAM (and eventually the GPU), all others remain on disk. Consequently, a history of data modification operations only needs to take those data fragments into account that are affected by each operation. An option is then to keep a list of the differences of fragments that are modified at each operation. However, as computing and applying differences of millions of points does take noticeable computation time, we decided against using differences, but rather to keep copies of the involved data fragments before modification. An undo operation can then be implemented by merely switching pointers to previous data fragments, avoiding any copy or computational operation for millions of points and/or their attributes.

For instance, let us denote an editing action as the transformation of a set of fragments {A,B,C,D,E} to another state {A',B',C',D',E'} (out of possibly many more). A first editing operation modifies fragments {A,B}, a second editing operation modifies fragments {C,D,E}. To cover this situation of two operations, the undo history must remember three states, denoted hereby as ①, ②, ③:

①	A	B			
②	A'	B'	C	D	E
③			C'	D'	E'

Each such state is stored as an "undo field" in the fiber bundle data model as introduced in Sec. 2.2 (Fig. 7 shows the structure as stored in an HDF5 file). When editing a fragment, also those fragments from the previous undo field in the current undo field have to be copied, even when they have not been edited. This means that an undo field contains both un-edited as well as edited data fragments, thus is a mix of edited and un-edited states. In the example given here, the undo field ② is largest as it contains five data fragments, the other two undo fields ① and ③ require less memory. An undo operation needs to replace fragments as stored in ③ by the respective fragments as stored in ② (but not all fragments in ②!). A subsequent undo operation then needs to replace fragments in undo field ② by the fragments stored in ①, if those exist. Thus, at each such operation, only the actually modified fragments are changed, same as when using differences: even though undo field ② contains five fragments, only two or three are replaced in this scenario at each undo operation.

4.3 I/O - Persistent Editing History

The usual approach of handling data is to

- load data from disk to RAM;
- if out of memory, let the operating system swap RAM data to disk, or utilize internal temporary files to store RAM information on disk;
- once data modification is finished, save the resulting data from RAM to exportable file formats;
- during data export, possibly load data from swap space (OS-provided or temporary files).

Whereas, via using HDF5 and the fiber bundle data model this functionality is simplified significantly:

- the file is parsed for metadata, only those are loaded;
- data fragments are only loaded when needed;

- modified data fragments are stored to the HDF5 if the application runs out of memory or terminates.

Hereby the HDF5 file serves as disk-image of the in-RAM data structure with no need of computationally intensive data transformations. This functionality mimics a memory-mapped file but is much more flexible since the data items can be extended dynamically at many "branches" of the hierarchically stored data. Via HDF5 highly performing compression filters such as LZ4 or ZSTD are available, such that disk space usage is minimal while I/O performance is higher than reading uncompressed data. Some of these high performance compression filters even claim [Alt10, Alt23] to be designed to unpack data faster than a traditional `memcpy()` operation could load them into CPU cache. In consequence, there is no need for "swap files" or "temporary files", and even an explicit "save" operation becomes superfluous: any data modification is directly mapped to the underlying HDF5 file in an end-user ready file format available for further data processing.

```
/t=0.0/Lake/Points/UTM32N/Labels/Frag[8x16x0] Dataset {273499}
/t=0.0/Lake/Points/UTM32N/Labels/Frag[8x17x0] Dataset {75708}
/t=0.0/Lake/Points/UTM32N/Labels/Frag[9x15x0] Dataset {108860}
/t=0.0/Lake/Points/UTM32N/Labels/Frag[9x16x0] Dataset {980038}
/t=0.0/Lake/Points/UTM32N/Labels/Frag[9x17x0] Dataset {998418}
/t=0.0/Lake/Points/UTM32N/Labels/Frag[9x18x0] Dataset {468318}
/t=0.0/Lake/Points/UTM32N/Labels/Frag[9x19x0] Dataset {901}
/t=0.0/Lake/Points/UTM32N/Labels/Frag[10x15x0] Dataset {143972}
/t=0.0/Lake/Points/UTM32N/Labels/Frag[10x16x0] Dataset {921022}
/t=0.0/Lake/Points/UTM32N/Labels/Frag[10x17x0] Dataset {1004095}
/t=0.0/Lake/Points/UTM32N/Labels/Frag[10x18x0] Dataset {857482}
/t=0.0/Lake/Points/UTM32N/Labels/Frag[10x19x0] Dataset {255736}
/t=0.0/Lake/Points/UTM32N/Labels.Undo-0000/Frag[10x19x0] Dataset {255736}
/t=0.0/Lake/Points/UTM32N/Labels.Undo-0001/Frag[10x19x0] Dataset {255736}
/t=0.0/Lake/Points/UTM32N/Labels.Undo-0002/Frag[10x19x0] Dataset {255736}
/t=0.0/Lake/Points/UTM32N/Labels.Undo-0003/Frag[10x19x0] Dataset {255736}
/t=0.0/Lake/Points/UTM32N/Labels.Undo-0004/Frag[10x19x0] Dataset {255736}
/t=0.0/Lake/Points/UTM32N/Labels.Undo-0005/Frag[10x19x0] Dataset {255736}
/t=0.0/Lake/Points/UTM32N/Labels.Undo-0006/Frag[10x17x0] Dataset {1004095}
/t=0.0/Lake/Points/UTM32N/Labels.Undo-0006/Frag[10x19x0] Dataset {255736}
/t=0.0/Lake/Points/UTM32N/Labels.Undo-0007/Frag[10x17x0] Dataset {1004095}
/t=0.0/Lake/Points/UTM32N/Labels.Undo-0007/Frag[10x19x0] Dataset {255736}
/t=0.0/Lake/Points/UTM32N/Labels.Undo-0007/Frag[9x16x0] Dataset {980038}
/t=0.0/Lake/Points/UTM32N/Labels.Undo-0007/Frag[9x17x0] Dataset {998418}
/t=0.0/Lake/Points/UTM32N/Labels.Undo-0008/Frag[9x16x0] Dataset {980038}
/t=0.0/Lake/Points/UTM32N/Labels.Undo-0008/Frag[9x17x0] Dataset {998418}
/t=0.0/Lake/Points/UTM32N/Mask/Frag[10x18x0] Dataset {857482}
/t=0.0/Lake/Points/UTM32N/Mask/Frag[10x19x0] Dataset {255736}
/t=0.0/Lake/Points/UTM32N/Mask.Undo-0000/Frag[10x19x0] Dataset {255736}
/t=0.0/Lake/Points/UTM32N/Mask.Undo-0001/Frag[10x19x0] Dataset {255736}
```

Figure 7: Partial listing of an HDF5 file in F5 layout as described in Sec. 2.2 containing both "labels" information and undo history of a fragmented point cloud using the HDF5 standard tool "h5ls". Note that fragments of the same name are of identical size.

The undo history as presented in Sec. 4.2 is stored as *Field*s in the fiber bundle model (as introduced in Sec. 2.2) and therefore subject to I/O like all other data fields. Consequently the undo history is persistently stored in an HDF5 file upon any data modification action and preserved when the application is restarted. Any data modification steps can be replayed days, weeks, years later, based on an HDF5 file containing the full history information. Fig. 7 demonstrates the effective partial structure of such an HDF5 file describing a point cloud with labels, undo information for labels, masking for labels and undo information for masking, based on the previously mentioned five-level hierarchical data organization. The length of the history is only limited by available disk space. A length of 1024 is already overkill in practical applications, using a length of 8 turned out to be sufficient to cover an active editing process.

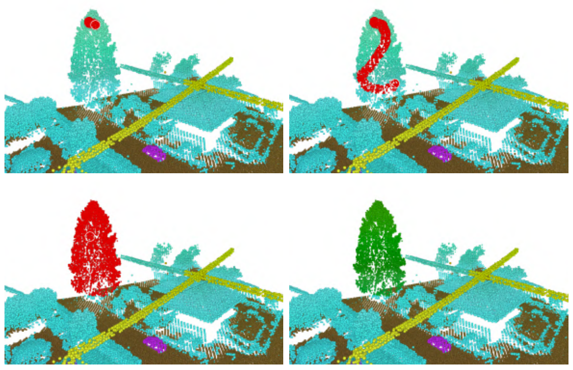


Figure 8: Example of free-style point selection (red points) and final classification assigned (green points).

5 RESULTS

In the following, we compare our implementation for editing point-clouds with that provided by two other software solutions. Similar to our software (denoted as $\boxed{3}$), one of the two software solutions is a general and widely used software toolkit for point clouds (denoted as $\boxed{1}$) allowing to import datasets from various origins including LAS format support. The other one is more specific and represents a software solution provided by a sensor manufacturer (denoted as $\boxed{2}$). We intentionally omit the actual name of the respective software packages to avoid marketing issues. The key points of the comparison are summarized in Table 1.

The comparisons were performed using a desktop PC equipped with an AMD Ryzen 7 3700X 8-Core Processor, 3.59 GHz, 64.0 GB RAM, graphic card NVIDIA GeForce RTX 3070 running under Windows 10 Pro.

5.1 Manual Selection Options

All three software solutions provide similar point selection options but with differences in their actual details such as naming. A standard selection tool is the polygon shape. The polygon shape selection must be performed vertex by vertex, and must be closed manually at the same place where the selection was started ($\boxed{1}$), or is closed automatically by simple right mouse-button click ($\boxed{2}$) respectively is already closed from the beginning ($\boxed{3}$). Further basic selection modes consist of rectangular selection mode (e.g. $\boxed{2}$), line selection mode (e.g. $\boxed{1}$), as well as a free-style selection mode ($\boxed{1}$, $\boxed{2}$, $\boxed{3}$; Fig. 8).

5.2 Subdivision of Big Point Clouds

To facilitate a quick manual editing progress as well as maintaining an overview of the editing progress especially in big point clouds, it is highly beneficial that an entire dataset can be subdivided into smaller pieces that can be edited separately from each other. This opportunity is provided in [1] and [3] but missing in [2]. For [3], Fig. 6 shows an example of a small tile (bottom image) resulting from the subtiling of a larger point cloud (top image). In Fig. 9, the active selection progress (red points) in the small tile from Fig. 6 is indicated.

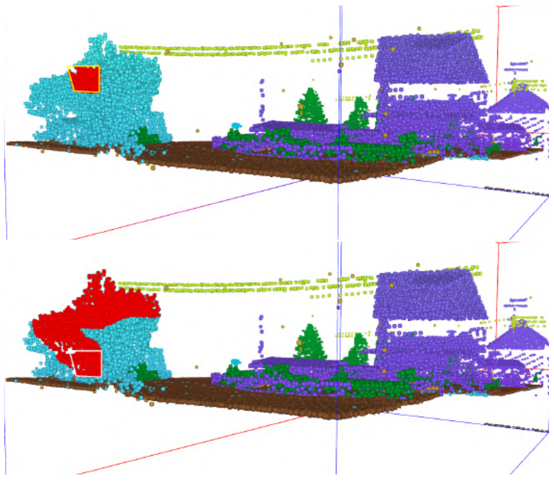


Figure 9: Active polygonal point selection progress (red points) in point cloud tile from Fig. 6.

5.3 Manual Classification Assignment

Once points were selected from a point cloud, the classification has to be assigned (Fig. 8). In one case, this is done immediately after the selection is finished ([1]). In the other case, the selection can still be edited further, e.g. expanding the selection or deselecting points ([2] and [3]). The classification is then assigned in a separate step by the user using a hotkey or button click in the respective GUI ([2] and [3]). All three software tools provide predefined classification ranges as well as the opportunity to introduce new classes according to the user's need. After assigning a class to a certain point selection, this selection is no longer maintained in all three software applications. In case of a mistakenly class assignment, [3]provides the opportunity of undo and redo operations at different levels, which are stored in the corresponding data file, and are still accessible after classification process is finished. In [1], single or multiple mistakenly class assignment(s) can be undone back in time, but these corrections are not accessible anymore after finalized classification. In [2], no undo operation is available to correct a mistakenly class assignment. Here, a renewed point selection and class assignment is required.

5.4 Display and Navigation

To facilitate the manual editing process of a point cloud, it is often required that only specific point class(es) are displayed and the point selection can be restricted to certain class(es). This is possible in all three software solutions evaluated here. In [2] it is further possible to restrict the display of points to a specified previous point selection. In [3], it is possible to simply set a certain cross-section depth and navigate back and forth through a point cloud in any desired direction just by mouse usage. In this case, points outside the defined cross-section depth are not displayed as well as not selectable for editing. In [1], a similar 3D navigation in cross-sectional view is possible but requiring a few more button clicks for specifying the navigation progress (Fig. 10). In contrast, in [2] it is only possible to display a cross-section of a certain depth throughout a point cloud based on an interactive 3D bounding box definition (Fig. 11 middle). All points inside this box are displayed afterwards (Fig. 11 bottom). For the user's orientation in a point cloud dataset, [2]provides separated 2D and 3D views. The zoom level of the 2D view defines the display area of the point cloud in 3D (Fig. 11). For re-sizing the 3D point cloud display, one need to adjust the zoom level in 2D first. Both views can be shown in parallel. In [1], the point cloud is usually displayed in several parallel 3D views, e.g. overview of entire dataset and detailed view from point cloud section for editing purpose (Fig. 10). The size of the views can be adjusted to the user's specific needs. In contrast to [1]and [2], we provide a single, quickly navigable 3D view of an entire point cloud in [3](Fig. 12).

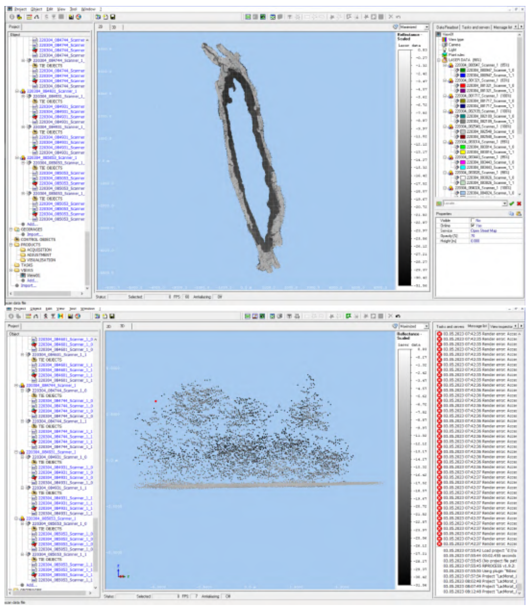


Figure 11: Solution [2]: GUI / view of an entire point cloud (top) and extracted cross-section view (bottom).

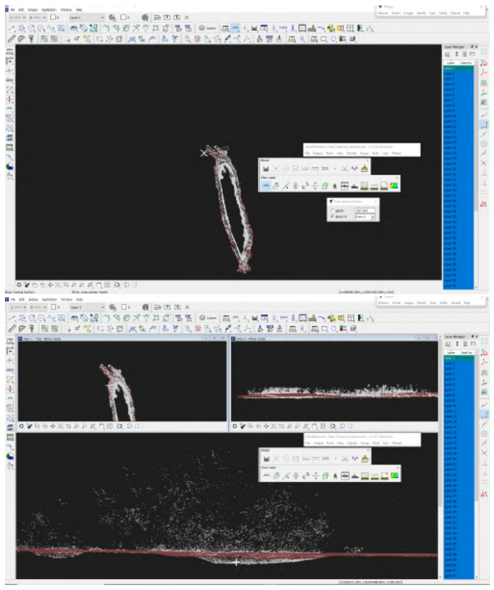


Figure 10: Solution [1]: GUI and 3D overview of an entire point cloud with indicated location of cross-sectional view (top). 3D views including detailed and cross-sectional view from area of interest (bottom).

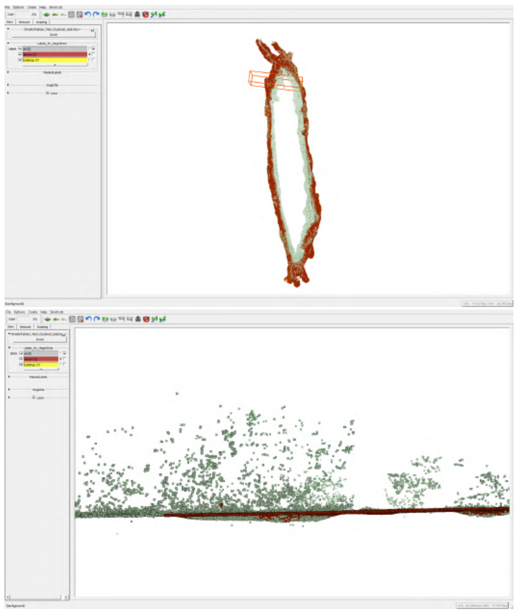


Figure 12: Solution [3] (ours): GUI and 3D view of an entire point cloud with marked sub-region (top) and cross-sectional view of sub-dataset (bottom).

6 CONCLUSION

We presented a systematic approach to allow interactive editing of big point clouds, and provided a comparison of our implementation with existing solutions. The presented methodology is based on the mathematics of fiber bundles and extends very naturally to various application scenarios, while providing high flexibility and performance at the same time. The ability to store the editing history persistently in a data file is an unique functionality offering high security on data management and modification accountability. It further supports data quality management, e.g. tracking classification errors in the editing history which became obvious during a later processing step such as terrain modeling out of classified terrain points. Manual editing of a point cloud requires combining multiple user aspects to be timely efficient. The flexible selection and class assignment modi in combination with the three-level, preservable undo/redo options of point editing stages as well as the 3D navigation and display possibilities are highly beneficial in this context. This capability increases the user's manual correction performance in general, resulting in shorter editing times in both simple and complex areas of a point cloud. A big point cloud requires quick and easy access to smaller subareas of the point cloud such that a regular subdivision for a systematic aerial editing is highly advantageous (Fig. 6 and Fig. 12) and improves productivity.

To support our findings, we performed a manual edit of the same point cloud in [1] and [3] by classifying two roofs out of the point cloud. The manual editing steps consist of a polygon-shaped point selection and class assignment to the first roof, continued by a subsequent point selection requiring 3D navigation to allow for an appropriate point selection and class assignment to the second roof. This editing was carried out by three users in both software applications, demonstrating a performance gain of about 33% for manual editing.

REFERENCES

[Alt10] Francesc Alted. Why modern cpus are starving and what can be done about it. *Computing in Science & Engineering*, 12(2):68–71, 2010.

[Alt23] Francesc Alted. Blosc, an extremely fast, multi-threaded, meta-compressor library. `https://www.blosc.org/pages/`, 2023.

[Ben04] Werner Benger. *Visualization of General Relativistic Tensor Fields via a Fiber Bundle Data Model*. PhD thesis, FU Berlin, 2004.

[Ben09] Werner Benger. Classifying data for scientific visualization via fiber bundles. In Claude Leroy and Pier-Giorgio Rancoita, editors, *ICATPP-11, Como, Italy, Oct 5-9, 2009*. World Scientific, 2009.

[BP89] David 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.

[BRH07] Werner Benger, Georg Ritter, and René Heinzl. The Concepts of VISH. In 4th *High-End Visualization Workshop, Obergurgl, Austria, June 18-21, 2007*, pages 26–39. Berlin, Lehmanns Media-LOB.de, 2007.

	[1]	[2]	[3] (ours)
Polygon selection Free-style selection	yes yes	yes yes	yes yes
Selection modification (deselect / invert / add / subtract)	no	yes	yes
User-defined classes	yes	yes	yes
Class assignment to selection	instantly	confirmative	confirmative
Undo / Redo	yes	no	yes
Persistent editing history	no	no	yes
Subdivision for systematic editing	yes	no	yes
Restrict display to specific classes	yes	yes	yes
Viewer layout	multiple 3D views for overview & detail (Fig. 10)	zoomable 2D & dependent 3D display extent (Fig. 11)	single navigable 3D view of entire point cloud (Fig. 12)
Cross-sectional navigation	additional button clicks for continuous navigation within point cloud	navigation restricted to selected cross-section extent	simple mouse usage for continuous navigation within point cloud

Table 1: Feature comparison. The faster 3D navigation in HydroVish allowed all test users to perform the manual editing in about $\approx$ 15 seconds, that was about 3-6 seconds quicker than the $\approx$ 20 seconds required for the editing in the alternative application [1] (35-40% difference).

[CZWea13] D. Chen, L. Zhang, Z. Wang, and et al. A mathematical morphology-based multi-level filter of lidar data for generating dtms. *Sci. China Inf. Sci.*, (56):1–14, 2013.

[HDF23] HDF Group. Hierarchical data format version 5, 2000-2023.

[HEA+21] S.S. Hansen, V.B. Ernstsen, M.S. Andersen, Z. Al-Hamdani, R. Baran, M. Niederwieser, F. Steinbacher, and A. Kroon. Classification of boulders in coastal environments using random forest machine learning on topo-bathymetric lidar data. *Remote Sens.*, 13(4101), 2021.

[HM90] Robert Haber and D McNabb. Visualization idioms: A conceptual model for scientific visualization systems. *IEEE Computer Society Press: Los Alamitos, CA*, pages 74—93, 01 1990.

[KP97] K. Kraus and N. Pfeifer. A new method for surface reconstruction from laser scanner data. 1997.

[LG17] B. Lohani and S. Ghosh. Airborne lidar technology: a review of data collection and processing systems. pages 567—579, 2017.

[LKS00] P. Lohmann, A. Koch, and M. Schäffer. Approaches to the filtering of laser scanner data. *Int Arch Photogrammetry Remote Sens XXXII I(B3/1)*, pages 534—541, 2000.

[Man20] A review of airborne laser bathymetry for mapping of inland and coastal waters. *Journal of Applied Hydrography*, 116(6):6–15, 2020.

[Man22] Underwater deadwood and vegetation from uav-borne topobathymetric lidar - the benefits of progress in uav and lidar sensor technology. *GIM International*, 2022.

[MWSCC09] X. Meng, L. Wang, J. Silvàn-Càrdenas, and N. Currit. A multi-directional ground filtering algorithm for airborne lidar. *ISPRS J Photogrammetry Remote Sens 64(1)*, pages 117—124, 2009.

[RBC+12] Marcel Ritter, Werner Benger, Biagio Cosenza, Keera Pullman, Hans Moritsch, and Wolfgang Leimer. Visual data mining using the point distribution tensor. Feb-Mar 2012.

[SDB+21] Frank Steinbacher, Wolfgang Dobler, Werner Benger, Ramona Baran, Manfred Niederwieser, and Wolfgang Leimer. Integrated full-waveform analysis and classification approaches for topo-bathymetric data processing and visualization in hydrovish. *PFG Journal of Photogrammetry, Remote Sensing and Geoinformation Science*, 2021.

[Sit01] G. Sithole. Filtering of laser altimetry data using a slope adaptive filter. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences XXXIV(3/4)*, pages 203—210, 2001.

[WSB+12] Gong Wei, Song Shalei, Zhu Bo, Shi Shuo, Li Faquan, and Cheng Xuewu. Multiwavelength canopy lidar for remote sensing of vegetation: Design and system performance. *ISPRS Journal of Photogrammetry and Remote Sensing*, 69:1–9, 2012.