Out-of-core Visualization of Classified 3D Point Clouds

Rico Richter and Sören Discher and Jürgen Döllner

Abstract 3D point clouds represent an essential category of geodata used in a variety of geoinformation applications and systems. We present a novel, interactive out-of-core rendering technique for massive 3D point clouds based on a layered, multi-resolution kd-tree, whereby point-based rendering techniques are selected according to each point's classification (e.g., vegetation, buildings, terrain). The classification-dependent rendering leads to an improved visual representation, enhances recognition of objects within 3D point cloud depictions, and facilitates visual filtering and highlighting. To interactively explore objects, structures, and relations represented by 3D point clouds, our technique provides efficient means for an instantaneous, ad-hoc visualization compared to approaches that visualize 3D point clouds by deriving mesh-based 3D models. We have evaluated our approach for massive laser scan datasets of urban areas. The results show the scalability of the technique and how different configurations allow for designing task and domain-specific analysis and inspection tools.

1 Introduction

In-situ and remote sensing technology (e.g., airborne, mobile, or terrestrial laser scanning and photogrammetric approaches) allows for efficient and automatic creation of digital representations of spatial environments such as cities and landscapes (Leberl et al, 2010; Lafarge and Mallet, 2012). These 3D point clouds are commonly used as an input data for applications, systems, and workflows to derive mesh-based 3D models (Arikan et al, 2013; Beutel et al, 2010) such as for sites, buildings, terrain, and vegetation. These models for example, can be used to create and maintain virtual 3D city models (Lafarge and Mallet, 2012; Kolbe, 2009), which can be applied in urban planning and development, environmental monitoring, disaster and risk management, and homeland security (Coutinho-Rodrigues et al, 2011). Applications and systems using massive 3D point clouds are faced by increasing availability (e.g., for whole countries), density (e.g., 400 points per m^2), and capturing frequency (e.g., once a year). However, they are limited due to their processing strategies that generally do not scale and limited storage capacities. As a remedy they frequently have to reduce the precision and density of the data. To process and analyze large datasets such as massive 3D point clouds out-of-core or external memory algorithms have been designed (Livny et al, 2009; Nebiker et al, 2010;

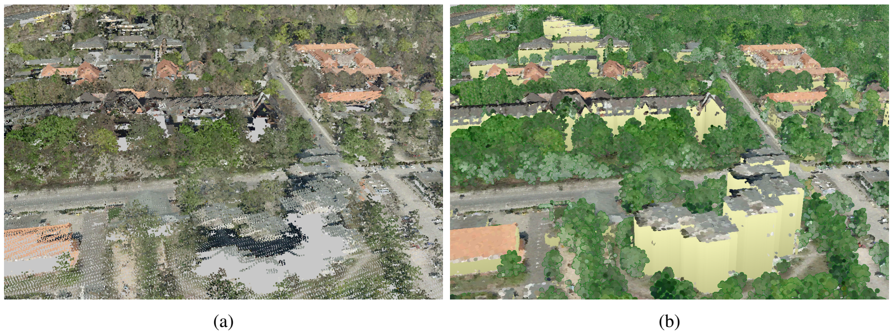


Fig. 1 (a) Example of a massive 3D point cloud rendered in a uniform way by *GL_POINTS* primitives and textured by aerial photography. (b) Same scene rendered by class-specific point-based techniques: different object classes can be better distinguished, holes on façades are filled, and visual clutter in the background is reduced.

Ganovelli and Scopigno, 2012; Rodríguez and Gobbetti, 2013). For the inspection and visualization of such datasets out-of-core real-time rendering systems enable an interactive exploration by using specialized spatial data structures and Level-of-Detail (LoD) concepts (Gobbetti and Marton, 2004; Wimmer and Scheiblauer, 2006; Richter and Döllner, 2010; Goswami et al, 2013). These systems generally render all points in a "uniform way" that does not take into account characteristics of different object classes, such as vegetation, building, terrain, street, or water. For example, building façades generally exhibit lower point density in contrast to roofs and terrain. A uniform rendering, therefore, results in gaps between neighboring façade points (Fig. 1), complicating their perception as a continuous surface. If points are rendered by the point primitives of the underlying rendering system (e.g., OpenGL's *GL_POINTS*) they are not scaled according to the camera distance making it difficult to correctly estimate depth differences and leading to visual artifacts due to overlapping of points close to each other. In addition, a uniform rendering does not differentiate between surface characteristics such as planar (e.g., terrain), structured (e.g., roof structures), and fuzzy areas (e.g., vegetation), complicating the visual identification and categorization of objects and structures by the user.

We report how the visualization of massive 3D point clouds can be improved based on object class information. Such information are computed with point cloud classification approaches (Lodha et al, 2007; Carlberg et al, 2009; Richter et al, 2013), which typically analyze the 3D point cloud topology, i.e., geometric relationships between points such as connectivity, local flatness, normal distribution, and orientation. We present a novel rendering approach that uses precomputed per-point attributes, such as object class information, color information, and topologic information to adapt the appearance of each point, i.e., its color, size, orientation, and shape. Different photorealistic, non-photorealistic, and solid point-based rendering techniques matching different surface characteristics are selected according to each point's classification. The class-specific rendering techniques can be configured at runtime according to the application and aim of the presentation. To filter and highlight points of specific object classes, focus + context visualization techniques, e.g., interactive and static lenses (Vaaraniemi et al, 2012; Trapp et al, 2008) can be applied. Interactive visualization of massive 3D point clouds, that exceed available memory resources and rendering capabilities, is achieved by storing points in a layered, multi-resolution kd-tree providing an object class specific subdivision of the data.

This paper is structured as follows: Section 2 discusses previous work. The system architecture is described in Section 3, focusing on point-based rendering techniques and the multi-pass rendering approach. Section 4 introduces the out-of-core rendering visualization based on the layered, multi-resolution kd-tree. In Section 5 we evaluate the performance of our system for massive datasets of urban areas. Section 6 gives conclusions and outlines future research directions.

2 Related Work

A general overview of point-based rendering is given by Gross and Pfister (2007). Several rendering techniques aim for a photorealistic and, thus, solid visualization of 3D point clouds without holes in the surface (Sibbing et al, 2013; Yu and Turk, 2013). These techniques commonly represent points as splats, i.e., oriented flat disks (Botsch et al, 2005; Zwicker et al, 2001), spheres, or particles. To visualize closed surfaces, an adequate size and orientation have to be applied to each point (Kim et al, 2012). These attributes can be calculated in a preprocessing step (Wu and Kobbelt, 2004) or on a per-frame basis as proposed by Preiner et al (2012). However, these techniques are difficult to apply for aerial 3D point clouds because of varying point densities, e.g., on horizontal and vertical structures, as well as on fuzzy and planar areas. In addition, it is difficult to combine these techniques with out-of-core rendering techniques for 3D point clouds because the point density varies depending on the LoD.

Non-photorealistic rendering techniques for 3D point clouds have been proposed by Goesele et al (2010) and Xu et al (2004). We extended the silhouette highlighting technique of Xu et al and added it to our set of rendering techniques. Olson et al (2011) show how the complete set of silhouette points of a surface can be calculated instant. However, that information comes with the cost of an additional preprocessing step.

Out-of-core rendering systems for 3D point clouds have been presented in (Gobbetti and Marton, 2004; Wimmer and Scheiblauer, 2006; Richter and Döllner, 2010; Goswami et al, 2013). These systems use LoD data structures that aggregate or generalize points solely based on spatial attributes. This is not applicable for our purpose because we need to separate points according to their object class at any time during rendering to apply object class specific rendering techniques as well as to render only selected object classes.

Point cloud classification of airborne laser scans has been discussed by several authors in recent years. Identification of building, terrain, and vegetation points is usually achieved by computing and weighting certain features (e.g., normal distribution, surface variation, horizontality) that describe the topology of the local neighborhood of a point (Zhou and Neumann, 2008; Lodha et al, 2007). An alternative to that approach is to use attributes specific to the respective scanning technology (Yunfei et al, 2008) (e.g., intensity of returning signals) or information that can be derived from additional geodata covering the same surface area (Kaminsky et al, 2009) (e.g., aerial images, infrastructure maps). In this contribution we compute object class information for each point in a preprocessing pass with a hybrid approach introduced by Richter et al (2013) that considers topologic features and additional per-point attributes.

In general, object class information is used to extract mesh-based 3D models (Zhou and Neumann, 2012) for specific categories such as vegetation, building, or terrain models. However, it is rarely used to enhance the visual quality of a 3D point cloud directly - aside from adapting the colorization of the points. A more advanced rendering approach that does take semantics into account was presented by Gao et al (2012). They aim for a solid, hole-free visualization of airborne laser scans by resampling terrain segments and by applying a solid rendering style. The purpose of this approach is quite similar to ours. However, our approach supports a larger variety of rendering styles that may be applied to arbitrary object classes at runtime. In addition, the preprocessing in our system is less demanding because we do not differentiate between roof and building points.

3 Class-Specific Point-Based Rendering

Our point-based rendering approach uses object class, color, and topologic information on a per-point basis to individualize the appearance of each point. Different point-based techniques are integrated by a multi-pass rendering technique responsible for the final image synthesis.

3.1 Data Characteristics

For a given raw 3D point cloud we compute per-point attributes in a preprocessing step. These attributes include the following:

- **Color**. Color or color-infrared values can be extracted from aerial images, ideally captured at the same point in time as the 3D point cloud. These values are generally used for a colorization, e.g., when a photorealistic and natural appearance of the points is required.
- **Object class information**. This attribute denotes to which surface category a point belongs. Typical object classes are vegetation, building, terrain, and water, which can be derived by analyzing the 3D point cloud topology, i.e., local neighborhood of a point. A more detailed subdivision of terrain (e.g., infrastructure, land use) or building points (e.g., commercial, residence) can be made by taking into account additional map data (e.g., infrastructure maps) (Richter et al, 2013).
- **Surface normal**. Per-point normals approximate the surface of the local point proximity. They can be computed efficiently by analyzing the local neighborhood of a point (Mitra and Nguyen, 2003) and are used to orientate the point primitive according to the represented surface.

- **Horizontality**. This attribute indicates how vertical the surface normal of a point is oriented, i.e., points representing horizontal surfaces (e.g., flat building roofs) feature higher values than points on vertical surfaces (e.g., building façades) (Zhou and Neumann, 2008). The horizontality can be used for a colorization to accentuate detailed object structures (e.g., roof elements).
- **Global height**. This attribute describes the height of a point in relation to all other points that belong to the same object class. Colorizing points based on their global height emphasizes height differences for different objects belonging to the same object class (e.g., trees with different heights).
- **Local height**. The local height describes the height of a point in relation to all points belonging to the same object class in the point's proximity. Using local heights for a colorization allows to highlight edges and differences in the structure of an object (e.g., roof ridges and smokestacks).

All attributes can be used to adapt the appearance of a point, i.e., its color, size, orientation and shape, at run-time. The color of a point can be chosen based on its color value, object class, topology attributes (i.e., surface normal, horizontality, global, or local height), or a combination of these. The orientation of a point can either correspond to its surface normal, the current view direction or a defined uniform vector. In addition, size and shape type of a point can be set dependent on its object class.

3.2 Point-Based Rendering Techniques

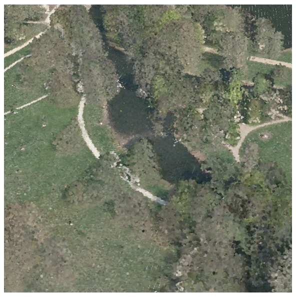
To efficiently render 3D point clouds, the Graphics Processing Unit (GPU) supports point primitives, such as *GL_POINTS* in OpenGL. However, these primitives have a fixed size in pixels (Shreiner et al, 2013) (e.g., Fig. 1 (a) uses a size of 3 pixel), i.e., their size in object space varies according to their perspective depth. Depending on the view position undersampling, i.e., holes between neighboring points (Fig. 1 (a) - bottom), or oversampling, i.e., visual clutter due to overlapping points (Fig. 1 (a) - top), occurs.

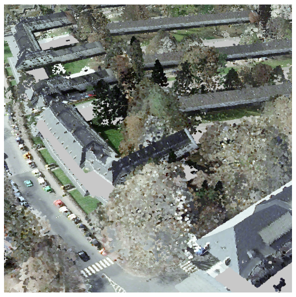
3.2.1 Point Splats

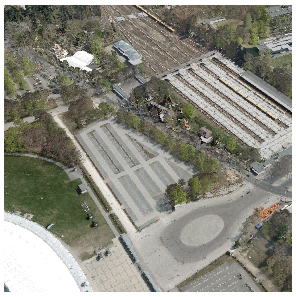
To avoid undersampling and oversampling due to changing view positions, the point splats technique renders each point as an opaque disk defined in object space that can be oriented alongside the surface normal (Rusinkiewicz and Levoy, 2000; Botsch et al, 2005). The on-screen size depends on the current view position and angle, ensuring a perspective correct visualization (Fig. 2 (a–f, i)). However, the perception of depth differences between overlapping points that are colored homogeneously (e.g., points belonging to the same object class), is generally limited.

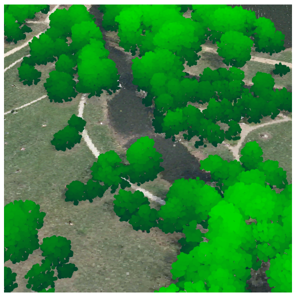
3.2.2 Point Spheres

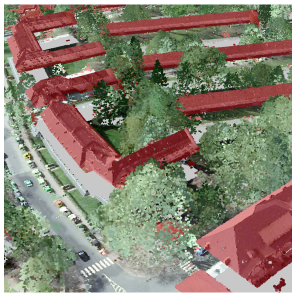
We implemented this point-based rendering technique to emphasize the three-dimensional character of a point. The proposed point spheres extend the original

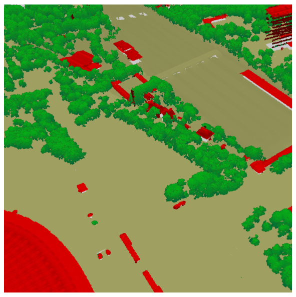

(a) Point Splats; aerial image colors.

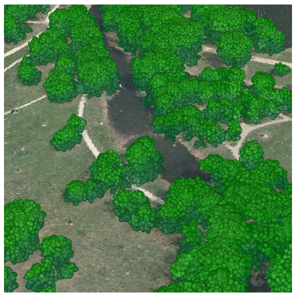

(b) Point Splats; aerial image colors.

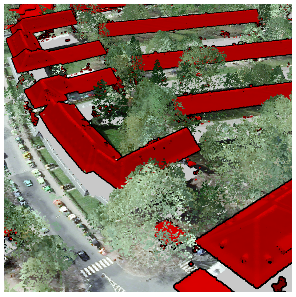

(c) Point Splats; aerial image colors.

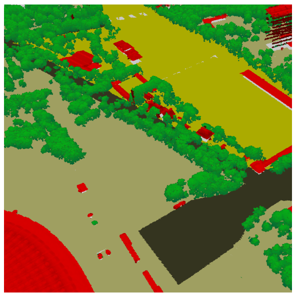

(d) Points Splats; global height.

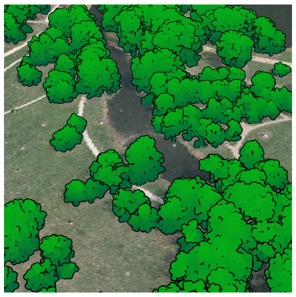

(e) Point Splats; aerial image colors and object class information.

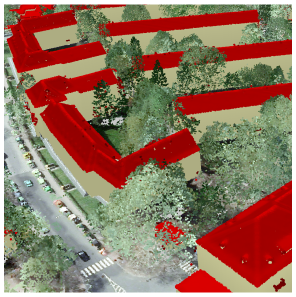

(f) Point Splats; global height.


(g) Point Spheres; local height.


(h) Silhouette Rendering; horizontality.


(i) Point Splats; object class information.


(j) Silhouette Rendering; local height.


(k) Solid Rendering; horizontality.

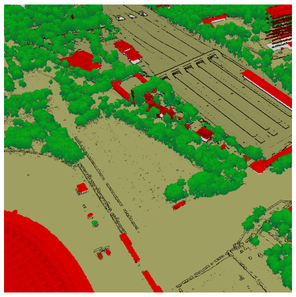

(l) Silhouette Rendering; global height.

Fig. 2 Examples of massive 3D point clouds rendered with different rendering setups for vegetation (left), buildings (middle), and terrain (right).

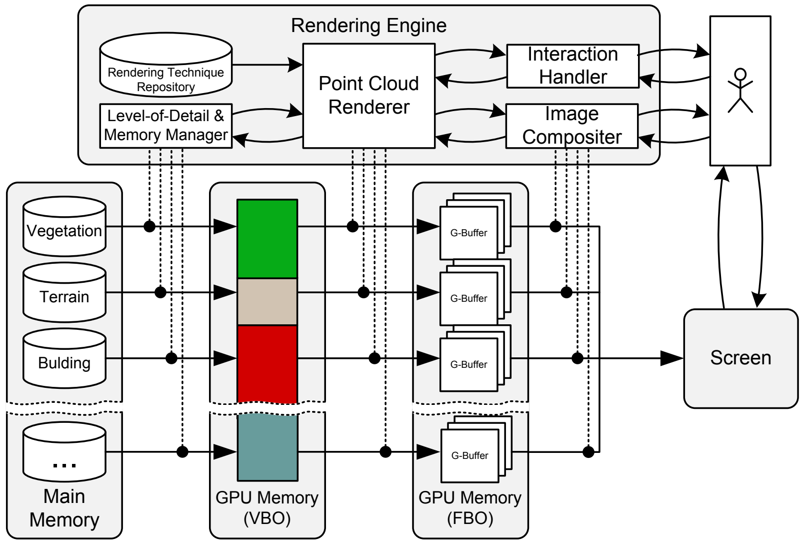


Fig. 3 Schematic overview of our class-specific point-based rendering system. Categorized by object classes, points are transferred to GPU memory and rendered into separate G-Buffers that are composed to synthesize the final image.

splat concept by rendering points as hemispheres instead of flat disks that are always facing the view position and, thus, look like spheres (Rusinkiewicz and Levoy, 2000). These hemispheres are created by (1) adding an offset to each depth value of the rendered fragment and by (2) shading each fragment. The depth offset as well as the shading color can be determined by projecting the fragment onto a plane defined by the corresponding splat and by calculating the projected distance of the fragment to the center of the splat. Point spheres are well suited for non-planar and fuzzy surfaces, such as vegetation (Fig. 2 (g)).

3.2.3 Silhouette Rendering

Point-based silhouettes highlight and abstract silhouettes and distinctive surface structures (e.g., depth differences). This technique extends the splat rendering approach and was originally proposed by Xu et al (2004). Similar to the rendering of point spheres, color and depth of each fragment depend on its projected distance to the center of the splat. In addition, the splat is divided into an inner and an outer part. Fragments in the outer part represent the silhouette and are rendered with an increased depth value and a distinct color. As a result, depth discontinuities between overlapping points exceeding a given depth offset are highlighted (Fig. 2 (h, j, l)).

3.2.4 Solid Rendering

We developed this point-based rendering technique to render buildings with solid and hole-free façades. As the point density on façades in airborne laser scans is

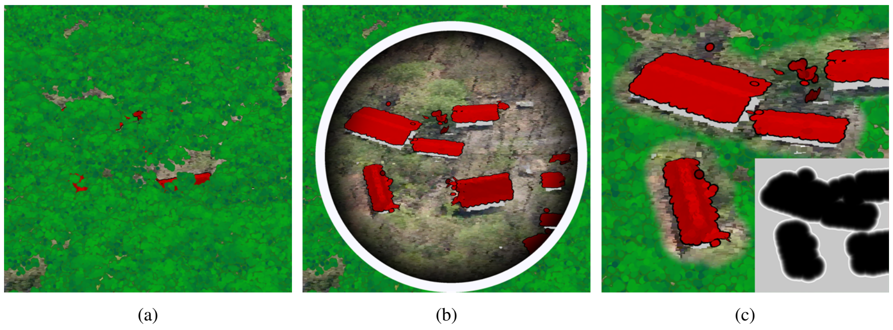


Fig. 4 Examples of focus + context visualization for classified 3D point clouds. (a) Regular visualization with buildings partially occluded by vegetation. (b) Interactive focus + context lens. (c) Static focus + context lenses positioned around building points.

very low in contrast to horizontal structures, the efficient identification of building segments is limited because other structures behind a building are visible through the façade (Gao et al, 2012). To overcome this, we use a second rendering pass to fill the area below roof points with new primitives. The *geometry shader* is used to render (1) a point-based splat, sphere or silhouette equal to the rendering techniques presented above and (2) a quad that imitates the façade below a point. The quad width is equal to the point size used in (1) whereas the height depends on the point's distance to the terrain level. All quads are aligned to the view direction and have the same color or height-based color gradient to create a solid façade look (Fig. 2 (k)).

3.3 Image Compositing

To combine different point-based rendering techniques, we use multi-pass rendering utilizing G-Buffers for image-based compositing (Saito and Takahashi, 1990) (Fig. 3). G-buffers are specialized frame buffer objects (FBO) that store multiple 2D textures for color, depth or normal values. Per object class we have one rendering pass. The results are stored in G-Buffers that are combined by the final rendering pass. This compositing pass allows to implement rendering techniques for focus + context visualization (Vaaraniemi et al, 2012; Trapp et al, 2008) such as interactive lenses (Fig. 4 (b)). Moreover, object class specific visibility masks, i.e., static lenses, can be computed and applied during the rendering to highlight occluded structures (Fig. 4 (c)). Point-based rendering techniques can be independently selected, combined and configured at run-time to adjust the appearance of each object class.

4 Out-of-Core Rendering

The interactive visualization of massive 3D point clouds exceeding available memory resources and rendering capabilities demands for out-of-core rendering techniques that combine LoD concepts, spatial data structures, and external memory

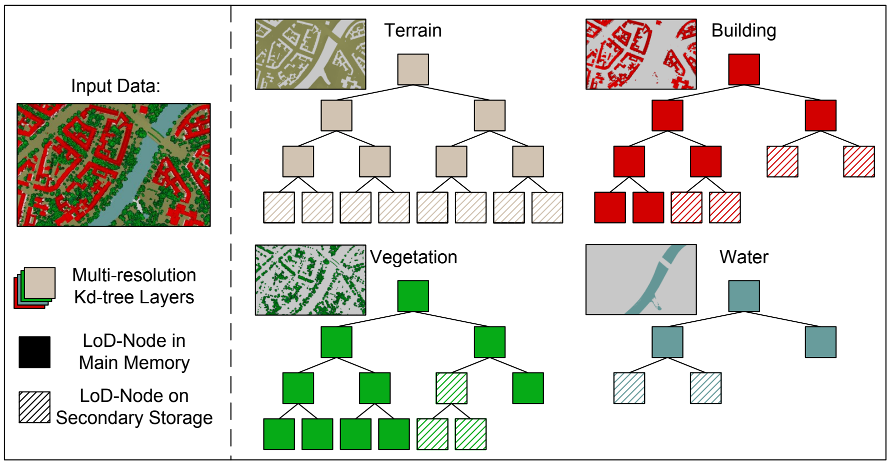


Fig. 5 Schematic overview showing the structure of our layered, multi-resolution kd-tree. For each object class a separate multi-resolution kd-tree is maintained.

algorithms. We developed a *layered, multi-resolution kd-tree* for massive 3D point clouds that have been attributed with object class information. It is characterized by the following properties:

- Object class specific subdivision of the data to enable a selective access and visualization (e.g., only building points).
- Adaptive multi-resolution LoDs to preserve a defined rendering budget (e.g., 30 frames per second).
- Efficient and adaptive memory management (e.g., by using equal-sized LoD chunks).
- Object class specific LoD selection to fulfill different requirements for specific rendering techniques (e.g., varying point densities).

4.1 Layered Multi-Resolution Kd-Tree

Most spatial data structures use kd-tree, quadtree, or octree derivations to arrange 3D point clouds in a preprocessing step (Rusinkiewicz and Levoy, 2000; Gobbetti and Marton, 2004; Wimmer and Scheiblauer, 2006; Richter and Döllner, 2010; Goswami et al, 2013). The construction of quadtrees and octrees can be performed faster in contrast to kd-trees because there is no need to sort the points. However, the use of quadtrees and octrees for irregular and sparse distributed data, e.g., airborne laser scans, results in tree nodes with a varying number of points. Out-of-core memory management has to implement efficient caching and memory swapping mechanisms that benefit from *equal-sized* data chunks. For that reason, we decided to use kd-trees to arrange the data. All points belonging to the same object class are arranged in a sub-tree consisting of nodes with an equal number of points (Fig. 5).

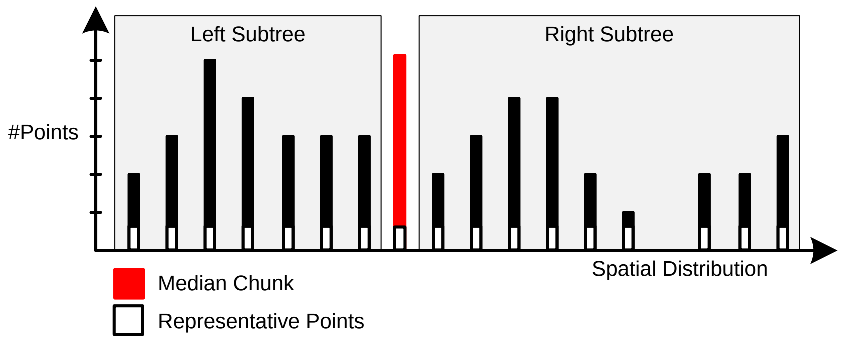


Fig. 6 Illustration of the histogram-based construction of the kd-tree to reduce preprocessing times for massive 3D point clouds.

Each of these nodes corresponds to a LoD for a spatial area with the root node representing the overall expansion of the 3D point cloud and child nodes subdividing the area of their parent node. Each point is stored only once in the tree, and all nodes together are equal to the input 3D point cloud.

4.1.1 Construction

The layered, multi-resolution kd-tree is constructed in a preprocessing step. It can be stored on secondary storage and therefore applied for arbitrary sized 3D point clouds. First, the given 3D point cloud is subdivided based on object classes. Second, for each object class the corresponding points are arranged in a multi-resolution kd-tree. The construction of a kd-tree with an equal number of points per node, i.e., a *balanced kd-tree*, is implemented by a multi-pass histogram-based approach that avoids a time-consuming sorting of the entire data for each tree level. In a first pass, we iterate over the 3D point cloud to fill a histogram that describes the spatial distribution and extent of the data. Similar to a voxel grid, the histogram organizes points into a number of equal-sized spatial chunks. For each chunk, the number of points belonging to the respective area and a representative point are stored (Fig. 6). Based on the number of points per chunk and the spatial extent of the histogram, a median chunk can be determined that contains the median point required to construct the kd-tree. A second iteration over the 3D point cloud is used to fill up the current node with representative points (i.e., to create a LoD) and to assign all points to the left or right part of the tree. Only points belonging to the median chunk need to be sorted to determine the exact median element. The median element for the split is chosen so that the number of points to the left is a multiple of the number of the points stored per node. This is important to construct a balanced kd-tree with equal sized nodes with exception of one leaf node. The out-of-core construction process subdivides point data on the file system until data chunks can be processed in main memory.

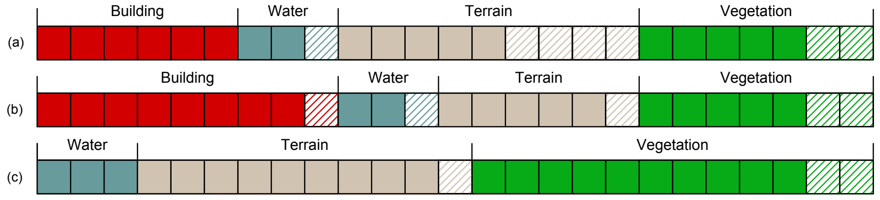


Fig. 7 Illustration of an exemplary GPU memory usage that is balanced during rendering according to memory requirements of LoD nodes that belong to different object classes. (a) to (b) illustrate how unused memory is assigned to other object classes. (b) to (c) illustrate the balancing process when the visualization of one object class (e.g., building) is disabled.

4.2 Layered Kd-Tree Rendering

The rendering process can be divided into three stages that are performed per frame. The first stage is responsible for the data provision, caching, and transferring of points from secondary storage to main memory as well as from main memory to GPU memory using the layered, multi-resolution kd-tree. The second stage applies one of our point-based rendering techniques (Sec. 3) to all points belonging to the respective object class. The last stage seamlessly combines all class-specific rendering results into one final image (Sec. 3.3).

At first, the root nodes of all class-specific sub-trees are loaded into main memory. Each chunk is equal to a LoD node and is mapped into a vertex buffer object (VBO) resident in GPU memory. The VBO is divided into equal sized chunks that can store exactly one LoD node. The layered, multi-resolution kd-tree is used to determined LoD nodes that need to be transferred to or can be removed from the VBO. The decision to add or remove a LoD node from memory depends on the projected node size (PNS). Therefore, the bounding sphere of the node is projected into screen space, and the number of covered pixels is compared to the number of points per node (Richter and Döllner, 2010). The threshold applied to the PNS depends on the point-based rendering technique, available memory, and computing capability of the GPU. Each object class has its own memory budget (Fig. 7) and is balanced permanently during the rendering process because the amount of memory required by an object class may vary due to the following reasons:

- Only a small number of points belonging to an object class is visible during the exploration.
- Visualization of certain object classes is disabled.
- Close up views require a high point density for an object class (e.g., for buildings).

Object classes can be rendered with different LoDs because the required number of points for an appropriate rendering result depends on the structure. For example, buildings may require to be rendered with more points due to detailed roof structures in contrast to terrain or vegetation that can be rendered with less points. To ensure a hole-free surface, the lower point density can be compensated by using larger primitives, e.g., splats for terrain or spheres for vegetation.

Table 1 Characteristics of the datasets used to evaluate the performance of the presented point-based rendering approach.

	Dataset 1	**Dataset 2**	**Dataset 3**
Point Density	10 pts/m^2	28 pts/m^2	100 pts/m^2
Number Points	5 Billion	7.1 Billion	80 Billion
Data Size	112 GB	159 GB	1788 GB

Table 2 Rendering performance in frames per second (fps) using the proposed out-of-core rendering approach. Each dataset is evaluated for a close and a far perspective.

	Dataset 1		**Dataset 2**		**Dataset 3**	
	Far	Close	Far	Close	Far	Close
#Rendered Points in Million	2.32	0.50	3.42	0.85	4.85	1.04
GL_POINTS	86.39	378.07	60.02	246.12	40.24	194.35
Point Splats	51.84	214.32	32.27	138.67	23.01	108.63
Point Spheres	49.57	203.81	28.31	133.72	22.35	107.07
Silhouette Rendering	46.07	195.65	26.66	127.38	22.18	106.97
Solid Rendering	27.32	100.13	20.22	63.78	18.74	59.45
Combination 1 (Fig. 2, row 3)	40.51	200.97	27.33	128.73	22.45	107.75
Combination 2 (Fig. 2, row 4)	33.28	126.31	22.21	80.80	19.90	68.47

5 Results and Applications

We have evaluated the presented system and all implemented point-based rendering techniques with three massive 3D point clouds containing up to 80 billion points (Table 1). For implementation we used C++, OpenGL, GLSL, and OpenSceneGraph. Measurements and tests were performed on an Intel Xeon CPU with 3.20 GHz, 12 GB main memory, and a NVIDIA GeForce GTX 770 with 2 GB device memory.

As shown in Fig. 8, interactive frame rates can be achieved for each rendering technique as long as the overall number of rendered points does not exceed a certain threshold (e.g., 6 million points for the solid rendering approach). The highest frame rate could be observed for *GL_POINTS*, which was expected since these primitives are supported natively by the GPU. Point Spheres as well as our solid and silhouette rendering approach extend the concept of Point Splats and increase the computational effort during rendering. Consequently, lower frame rates were achieved when using these techniques for rendering as opposed to Point Splats. Furthermore, the performance for Point Spheres is higher than for Point Silhouettes due to a more hardware demanding shading implementation (e.g., conditional branching).

Since the proposed out-of-core rendering approach limits the number of rendered points by dynamically selecting them, arbitrarily large datasets with varying point densities can be rendered in real-time as well (Table 2).

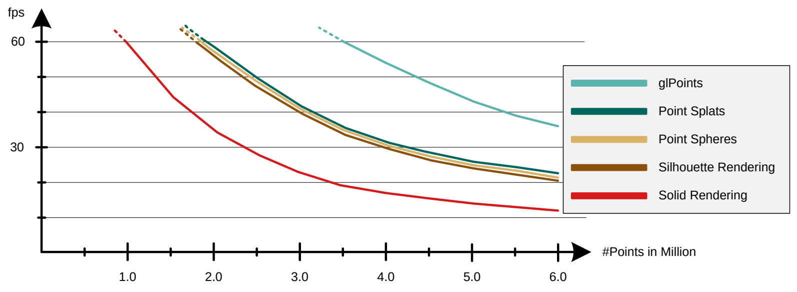


Fig. 8 Rendering performance in frames per second (fps) using different sized subsets of the datasets from Table 1.

6 Conclusions and Future Work

We have shown that out-of-core rendering for massive 3D point clouds can be improved by using point-specific attributes such as topologic or semantic information. In particular, object class information can be used to select specialized point-based rendering techniques that take into account class-specific surface characteristics (e.g., solid, planar, non-planar, fuzzy). In addition, it enables focus + context techniques, e.g., lenses for filtering and highlighting. This way we can improve the visual appearance and facilitate recognition of objects within 3D point clouds. Furthermore, our approach offers many degrees of freedom for graphics and interaction design. This approach also allows us to dissolve occlusion and enable a task-specific interactive exploration. The proposed layered, multi-resolution kd-tree enables in addition to a spatial data selection an object class specific selection of LoDs. Hence, memory and processing resources can be used economically and adaptively. In future work, we plan to integrate point-based rendering techniques that enable a per-frame reconstruction of object surfaces (Preiner et al, 2012), e.g., for terrain or roof points. In addition, we want to combine 3D point clouds from aerial scans with data from mobile and terrestrial scans to increase the number of available object classes.

7 Acknowledgements

This work was funded by the Federal Ministry of Education and Research (BMBF), Germany within the InnoProfile Transfer research group 4DnD-Vis (`www.4dndvis.de`) and the Research School on Service-Oriented Systems Engineering of the Hasso Plattner Institute. We would like to thank virtualcitySYSTEMS for providing datasets.

References

Arikan M, Schwärzler M, Flöry S, Wimmer M, Maierhofer S (2013) O-snap: Optimization-based Snapping for Modeling Architecture. ACM Transactions on Graphics 32(1):6:1–6:15

Beutel A, Mølhave T, Agarwal P (2010) Natural neighbor interpolation based grid DEM construction using a GPU. In: 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp 172–181

Botsch M, Hornung A, Zwicker M, Kobbelt L (2005) High-quality surface splatting on today's GPUs. In: Eurographics Symposium on Point-Based Graphics, pp 17–24

Carlberg M, Gao P, Chen G, Zakhor A (2009) Classifying Urban Landscape in Aerial Lidar Using 3D Shape Analysis. In: 16th IEEE International Conference on Image Processing, pp 1701–1704

Coutinho-Rodrigues J, Simão A, Antunes C (2011) A GIS-based multicriteria spatial decision support system for planning urban infrastructures. Decision Support Systems 51(3):720–726

Ganovelli F, Scopigno R (2012) OCME: Out-of-Core Mesh Editing Made Practical. Computer Graphics and Applications 32(3):46–58

Gao Z, Nocera L, Neumann U (2012) Visually-complete aerial LiDAR point cloud rendering. In: 20th International Conference on Advances in Geographic Information Systems, pp 289–298

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(6):815–826

Goesele M, Ackermann J, Fuhrmann S, Haubold C, Klowsky R, Steedly D, Szeliski R (2010) Ambient point clouds for view interpolation. ACM Transactions on Graphics 29(4):95:1–95:6

Goswami P, Erol F, Mukhi R, Pajarola R, Gobbetti E (2013) An efficient multi-resolution framework for high quality interactive rendering of massive point clouds using multi-way kd-trees. The Visual Computer 29(1):69–83

Gross M, Pfister H (2007) Point-based Graphics. Morgan Kaufmann Publishers Inc.

Kaminsky R, Snavely N, Seitz S, Szeliski R (2009) Alignment of 3D point clouds to overhead images. In: Computer Vision and Pattern Recognition Workshops, pp 63–70

Kim HJ, Öztireli AC, Gross M, Choi SM (2012) Adaptive surface splatting for facial rendering. Computer Animation and Virtual Worlds 23(3-4):363–373

Kolbe TH (2009) Representing and Exchanging 3D City Models with CityGML. In: 3D geo-information sciences, chap 2, pp 15–31

Lafarge F, Mallet C (2012) Creating Large-Scale City Models from 3D-Point Clouds: A Robust Approach with Hybrid Representation. International Journal of Computer Vision 99(1):69–85

Leberl F, Irschara A, Pock T, Meixner P, Gruber M, Scholz S, Wiechert A (2010) Point Clouds: Lidar versus 3D Vision. Photogrammetric Engineering and Remote Sensing 76(10):1123–1134

Livny Y, Kogan Z, El-Sana J (2009) Seamless patches for GPU-based terrain rendering. The Visual Computer 25(3):197–208

Lodha SK, Fitzpatrick DM, Helmbold DP (2007) Aerial Lidar Data Classification using AdaBoost. In: Sixth International Conference on 3-D Digital Imaging and Modeling (3DIM), pp 435–442

Mitra NJ, Nguyen A (2003) Estimating surface normals in noisy point cloud data. In: 19th Annual Symposium on Computational Geometry, pp 322–328

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

Olson M, Dyer R, Zhang H, Sheffer A (2011) Point set silhouettes via local reconstruction. Computers & Graphics 35(3):500–509

Preiner R, Jeschke S, Wimmer M (2012) Auto Splats: Dynamic Point Cloud Visualization on the GPU. In: Proceedings of Eurographics Symposium on Parallel Graphics and Visualization, pp 139–148

Richter R, Döllner J (2010) Out-of-core Real-time Visualization of Massive 3D Point Clouds. In: 7th International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa, pp 121–128

Richter R, Behrens M, Döllner J (2013) Object class segmentation of massive 3D point clouds of urban areas using point cloud topology. International Journal of Remote Sensing 34(23):8408–8424

Rodríguez M, Gobbetti E (2013) Coarse-grained multiresolution structures for mobile exploration of gigantic surface models. In: SIGGRAPH Asia Symposium on Mobile Graphics and Interactive Applications, pp 4:1–4:6

Rusinkiewicz S, Levoy M (2000) QSplat: A multiresolution point rendering system for large meshes. In: ACM SIGGRAPH, pp 343–352

Saito T, Takahashi T (1990) Comprehensible Rendering of 3-D Shapes. SIGGRAPH Computer Graphics 24(4):197–206

Shreiner D, Sellers G, Kessenich JM, Licea-Kane BM (2013) OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.3, 8th edn. Addison-Wesley

Sibbing D, Sattler T, Leibe B, Kobbelt L (2013) SIFT-Realistic Rendering. International Conference on 3D Vision pp 56–63

Trapp M, Glander T, Buchholz H, Döllner J (2008) 3D Generalization Lenses for Interactive Focus + Context Visualization of Virtual City Models. In: 12th International Conference on Information Visualisation, pp 356–361

Vaaraniemi M, Freidank M, Westermann R (2012) Enhancing the visibility of labels in 3D navigation maps. In: Lecture Notes in Geoinformation and Cartography, pp 23–40

Wimmer M, Scheiblauer C (2006) Instant points: Fast rendering of unprocessed point clouds. In: Eurographics Symposium on Point-Based Graphics, pp 129–137

Wu J, Kobbelt L (2004) Optimized Sub-Sampling of Point Sets for Surface Splatting. Computer Graphics Forum 23(3):643–652

Xu H, Nguyen MX, Yuan X, Chen B (2004) Interactive Silhouette Rendering for Point-Based Models. Eurographics Symposium on Point-Based Graphics pp 13–18

Yu J, Turk G (2013) Reconstructing surfaces of particle-based fluids using anisotropic kernels. ACM Transactions on Graphics 32(1):5:1–5:12

Yunfei B, Guoping L, Chunxiang C, Xiaowen L, Hao Z, Qisheng H, Linyan B, Chaoyi C (2008) Classification of LIDAR point cloud and generation of DTM from LIDAR height and intensity data in forested area. In: International Society for Photogrammetry and Remote Sensing Congress, pp 313–318

Zhou QY, Neumann U (2008) Fast and Extensible Building Modeling from Airborne Lidar Data. In: 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp 1–8

Zhou QY, Neumann U (2012) 2.5D Building Modeling by Discovering Global Regularities. In: Computer Vision and Pattern Recognition, pp 326–333

Zwicker M, Pfister H, van Baar J, Gross MH (2001) Surface splatting. In: ACM SIGGRAPH, pp 371–378