A Point-Based and Image-Based Multi-Pass Rendering Technique for Visualizing Massive 3D Point Clouds in VR Environments

Sören Discher
Hasso Plattner Institute
University of Potsdam
Prof.-Dr.-Helmert-Straße 2-3
14482 Potsdam, Germany
soeren.discher@hpi.de

Leon Masopust
Hasso Plattner Institute
University of Potsdam
Prof.-Dr.-Helmert-Straße 2-3
14482 Potsdam, Germany
leon.masopust@student.hpi.de

Sebastian Schulz
Hasso Plattner Institute
University of Potsdam
Prof.-Dr.-Helmert-Straße 2-3
14482 Potsdam, Germany
sebastian.schulz@student.hpi.de

Rico Richter
Hasso Plattner Institute
University of Potsdam
Prof.-Dr.-Helmert-Straße 2-3
14482 Potsdam, Germany
rico.richter@hpi.de

Jürgen Döllner
Hasso Plattner Institute
University of Potsdam
Prof.-Dr.-Helmert-Straße 2-3
14482 Potsdam, Germany
juergen.doellner@hpi.de

ABSTRACT

Real-time rendering for 3D point clouds allows for interactively exploring and inspecting real-world assets, sites, or regions on a broad range of devices but has to cope with their vastly different computing capabilities. Virtual reality (VR) applications rely on high frame rates (i.e., around 90 fps as opposed to 30 - 60 fps) and show high sensitivity to any kind of visual artifacts, which are typical for 3D point cloud depictions (e.g., holey surfaces or visual clutter due to inappropriate point sizes). We present a novel rendering system that allows for an immersive, nausea-free exploration of arbitrary large 3D point clouds on state-of-the-art VR devices such as HTC Vive and Oculus Rift. Our approach applies several point-based and image-based rendering techniques that are combined using a multi-pass rendering pipeline. The approach does not require to derive generalized, mesh-based representations in a pre-processing step and preserves precision and density of the raw 3D point cloud data. The presented techniques have been implemented and evaluated with massive real-world data sets from aerial, mobile, and terrestrial acquisition campaigns containing up to 2.6 billion points to show the practicability and scalability of our approach.

Keywords
Virtual reality, 3D point clouds, Real-time rendering

1 INTRODUCTION

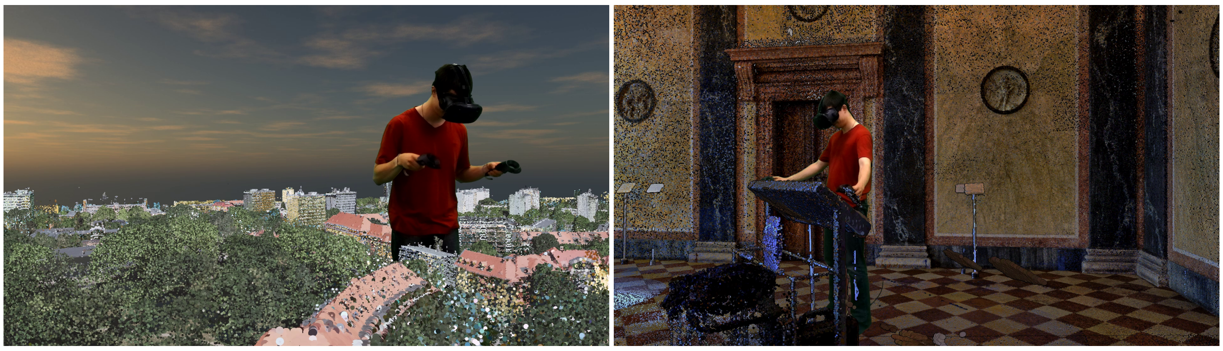
Virtual reality (VR) devices, for example Oculus Rift[1] or HTC Vive[2], open up new ways to present digital 3D models on standard consumer hardware, granting users the perception of being physically present in a 3D virtual environment [1, 2]. In general, the corresponding 3D models can be designed and modeled for a particular purpose (e.g., game environment) or can be derived by captured data from real-world sites or assets (e.g., building models).

For complex sites, e.g., buildings with a highly detailed interior, or large areas, e.g., cities and landscapes, manually modeling 3D contents is neither time efficient nor cost efficient due to the required effort [26]. As a remedy, there is a strong demand for methods and techniques that (1) automatically and efficiently capture real-world sites of arbitrary size and complexity with high precision and that (2) directly integrate the resulting 3D contents into VR applications without having to sacrifice any captured details.

In recent years, automatically capturing real-world sites by means of 3D point clouds has become increasingly cost efficient and time efficient due to technological advances in remote and in-situ sensing technology [11]. As an example, active and passive sensing technology,

[1] https://www.oculus.com/rift/
[2] https://www.vive.com

https://doi.org/10.24132/JWSCG.2018.26.2.2



a *Airborne scan of a city.* b *Terrestrial indoor scan.*

Figure 1: Examples of massive 3D point clouds being immersively visualized using our rendering system and an HTC Vive. Supported interaction techniques include measuring of distances as well as rotating and scaling of the rendered data.

such as *light detection and ranging* (LiDAR), radar, or aerial and digital cameras, provides precision rates of up to a few centimeters or millimeters [17, 27]. By attaching those sensors to moving vehicles, such as cars or unmanned aircraft systems (UAS), large areas can be covered within few hours, resulting in massive data sets containing hundreds of gigabytes of raw data [21, 31].

Large unstructured collections of 3D points, called *3D point clouds*, can be directly used as interactively explorable models by combining *level-of-detail (LoD)* concepts, out-of-core strategies, and external memory algorithms [32, 14]: State-of-the-art rendering systems are capable of handling enormous amounts of 3D point cloud data, e.g., billions of points for a non-immersive inspection and visualization on a multitude of devices with vastly different CPU and GPU capabilities [24, 7]. However, they typically focus on non-immersive applications, carefully balancing the trade-off between rendering quality and performance [36]. In VR applications additional challenges are raised:

- **Stereo rendering**. To generate a stereoscopic image, each scene has to be rendered for two displays simultaneously.
- **High rendering quality**. Visual artifacts such as visible holes between neighboring points or visual clutter tend to be more noticeable on VR displays, can easily break the immersion [37] and, therefore, need to be fixed.
- **High frame rates of 90 fps**. Nausea, i.e., the feeling of motion sickness, typically occurs when the motion-to-photon-latency, i.e., the time required for the depicted images to update after a physical movement by the user, becomes too high. As a remedy, the built-in displays of VR devices such as Oculus Rift or HTC Vive operate at 90 Hz [39]. Hence, frames have to be rendered at a considerably higher speed compared to non-immersive applications, for which frame rates between 30 and 60 fps are usually sufficient.

For these reasons, applications for VR devices frequently have to reduce the precision and density of the data, either by thinning the respective point clouds [29] or by converting them into generalized 3D meshes [3].

In this paper, we present a rendering system (Section 3) that allows for an interactive, immersive, and nausea-free visualization of arbitrary large 3D point clouds on state-of-the-art VR devices (Fig. 1). To that end, we combine selected rendering techniques for 3D point clouds that can be roughly sorted into two categories: *Performance optimization techniques* that speed up the rendering pipeline (Section 4) and *image optimization techniques* that improve the overall image quality (Section 5). All techniques have been implemented and are evaluated for two massive, real-world data sets. We end with a conclusion and an overview of future work.

2 RELATED WORK

3D point clouds are widely used in a variety of geospatial [9] and non-geospatial applications [38], used in diverse areas such as building information modeling [28], urban planning and development [25] or the digital preservation of cultural heritage [24]. As fundamental geospatial data representation, 3D point clouds provide highly detailed geometry information about a site that can be further extracted and leveraged by applying point-based analysis algorithms [6, 8]. Although this not being the focus, our approach provides efficient means to visualize results of those analyses, e.g., by applying different per-point color schemes (Fig. 3).

Presentation and interactive visualization of 3D point clouds have to cope with the corresponding massive amount of data, which generally exceeds available CPU and GPU capabilities [15]. To render massive data sets with billions of points, out-of-core rendering concepts and spatial data structures are required to decouple rendering efforts from data management such as

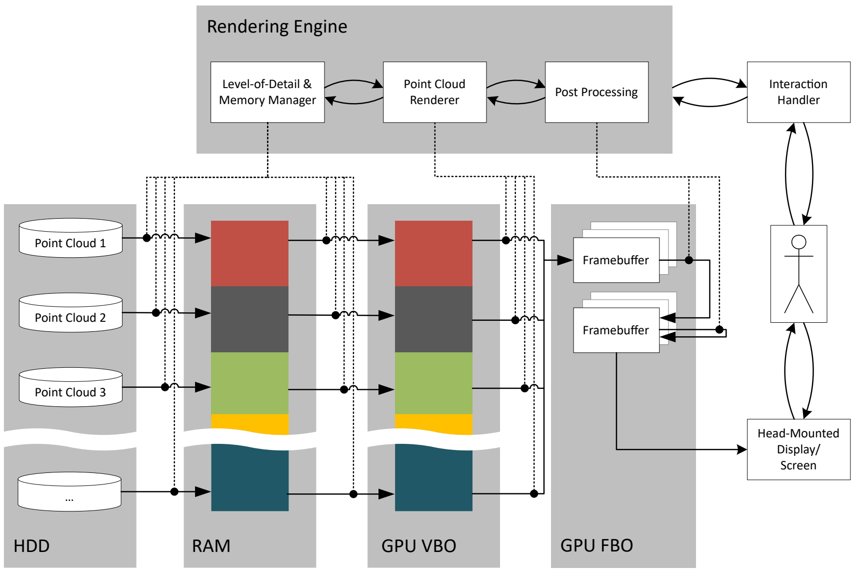


Figure 2: Overview of the rendering pipeline and data flow between hard disk drive (HDD), random-access memory (RAM), vertex buffer objects (VBO), and frame buffer objects (FBO). Different 3D point clouds are rendered separately but share a single memory budget.

quadtrees [13], octrees [12], or kd-trees [14] to subdivide 3D point clouds into small, representative subsets that are suitable for real-time rendering. Out-of-core approaches and web-based rendering concepts are frequently combined. For example, a central server infrastructure can be used to organize and distribute the corresponding 3D point clouds, which limits workload and data traffic on client side [36, 7]. While those approaches allow to visualize massive data sets on client devices with vastly different hardware and graphics capabilities, they generally provide neither visual quality nor rendering performance as required by an immersive visualization.

Real-time rendering is based on performance optimization techniques: While techniques such as view frustum culling and detail culling can be easily applied to 3D point clouds, occlusion, backface, and portal culling are designed with mesh-based geometry and closed surfaces in mind [1]. Due to the unstructured nature of 3D point clouds those techniques require adaptation before being applicable to point-based rendering. Our rendering system implements occlusion culling based on the reverse painter's algorithm [19]. We decided against adapting backface and portal culling as both techniques require specific knowledge or preprocessed information about a 3D point cloud (e.g., per-point normals, semantic information) that might not always be available. Performance optimization techniques specifically for VR applications have been discussed by [39]. Some of those techniques, such as the hidden mesh or the single-pass stereo rendering, are implemented and evaluated by our rendering system.

Visual optimization techniques for 3D point clouds are discussed by several authors, an overview is given

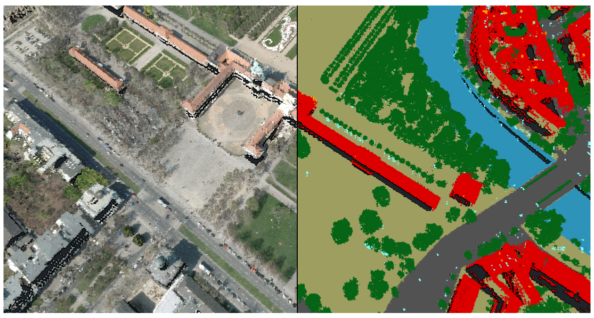


Figure 3: Different color schemes can be applied at runtime. Left: RGB colors extracted from aerial imagery. Right: Colorization based on surface categories.

by [15]. Visual clutter and holes between neighboring points can be addressed by applying appropriate size, orientation, and color schemes to each point [36, 32]. While leading to good visual results, those techniques also raise the computational cost due to calculating point sizes in object space – either in a pre-processing step [4] or during rendering [30]. As an alternative that scales better for massive data sets, visual artifacts can be eliminated via post-processing using image-based rendering techniques, e.g., to fill holes ([10], [33]), to blur visual clutter [22], or to emphasize depth cues [5, 23]. In the context of VR applications Schütz [37] introduces the usage of point cloud mipmaps as well as multisampling for a reduction of z-fighting and softer edges, which we also evaluate in this paper.

3 SYSTEM OVERVIEW

Our implementation of the rendering system is based on a multi-pass rendering pipeline (Fig. 2) that can

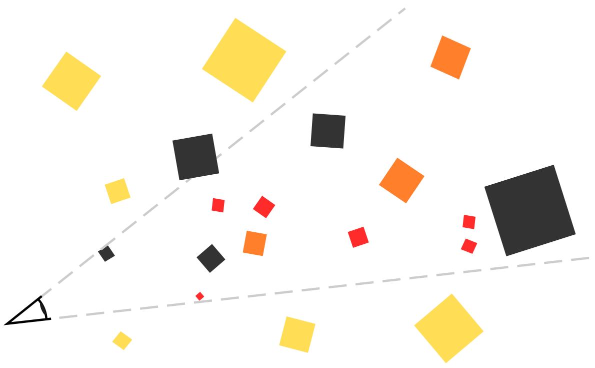


Figure 4: Culling techniques used to reduce the amount of points to be rendered: View frustum culling (yellow), occlusion culling (orange), detail culling (red).

be divided into three distinct stages: Data subset selection, point cloud rendering, and image-based post-processing.

3.1 Level-of-Detail and Data Subset Selection

Instead of rendering every point of a given data set, we determine a *representative subset* of points that can be managed by available CPU and GPU capabilities. Those subsets are determined on a per-frame basis using two major criteria (Fig. 4): First, points outside the current view frustum are excluded as they would not be visible anyway (i.e., *view frustum culling*). Second, points are aggregated based on their spatial position to accommodate for the perspective distortion resulting in areas farther away from the current view position to appear smaller on screen (i.e., *detail culling*). To provide an efficient access to representative data subsets, the 3D point cloud is hierarchically subdivided using a kd-tree, i.e., a binary tree whose splitting planes can be freely positioned alongside the respective coordinate axes. This allows for minimal tree traversal times during rendering as the resulting tree structures are guaranteed to be balanced independently of the data's spatial distribution. For each 3D point cloud a separate kd-tree is generated in a preprocessing step. A flexible memory budget is defined to limit the amount of points that can be rendered per frame. While each 3D point cloud is rendered separately, the memory budget is shared among them. As the performance may vary based on scene complexity and applied rendering techniques, the memory budget is adjusted dynamically to guarantee 90 fps at any time.

3.2 Point Cloud Rendering

Selected data subsets are rendered into so-called g-buffers [34], i.e., specialized frame buffer objects (FBO) that combine multiple 2D textures for, e.g., color, depth, or normal values. This provides efficient means to apply varying post-processing effects that

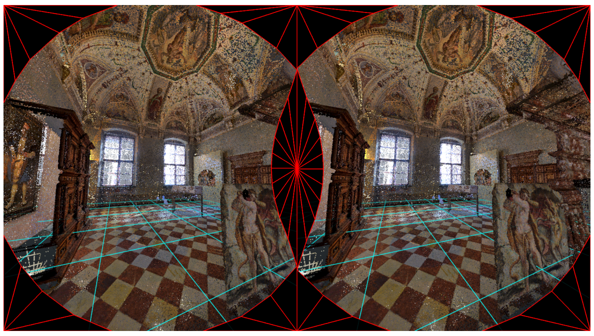


Figure 5: A separately rendered mesh serves as a mask to discard fragments beyond the visible area of an VR device's screens early on.

improve the visual quality of the final image being displayed on the VR device. Furthermore, different rendering techniques for 3D point clouds can be configured, selected, and combined at runtime, allowing to dynamically adjust a 3D point cloud's appearance (e.g., size and color scheme applied to each point) (Fig. 3) as well as the overall rendering performance (Section 4).

3.3 Image-Based Post-Processing

The rendering pipeline's final stage operates recursively on the previously generated g-buffers, allowing to configure and combine several image-based rendering techniques. As an example, rendering techniques for hole-filling, blurring, anti-aliasing as well as edge detecting and highlighting can be efficiently combined to improve the visual quality of the rendering (Section 5).

3.4 Interaction Handling

An interaction handler is responsible for managing user interactions and for updating the visualization accordingly. Users may (1) change view position and angle, (2) configure and select applied rendering techniques and color schemes, (3) measure distances between points, or (4) scale and rotate rendered 3D point clouds.

4 PERFORMANCE OPTIMIZATION TECHNIQUES

To further improve the performance of our rendering system on state-of-the-art VR devices, we have implemented and evaluated three rendering techniques: Hidden mesh rendering, reverse painter's algorithm, and single-pass stereo rendering.

4.1 Hidden Mesh Rendering

Due to the radially symmetric distortion produced by the lenses of an VR device, the actually visible area of the built-in screens is restricted to a circular area (Fig. 5). To prevent unnecessary fragment shader operations, fragments outside that area are discarded early, using a separately rendered mesh representing the hidden parts of the screen as a mask that is evaluated using early fragment testing [39].

4.2 Reverse Painter's Algorithm

As a GPU-based *occlusion culling* technique (Fig. 4), the reverse painter's algorithm [19] describes efficient means to prevent occluded fragments from being unnecessarily processed by the fragment shader. Based on early fragment testing, scene objects should be rendered in order of their distance to the view position for the technique to have a measurable effect. Calculating such an order on a per-point basis would be inefficient. As each point belongs to a specific node of the kd-tree however, we can instead perform that calculation on a per-node basis, considering only those nodes that have been selected for rendering.

4.3 Single-Pass Stereo Rendering

VR devices require to render all view dependent items from two different views representing the left and right eye, respectively. *Single-pass stereo rendering* aims to reduce the *CPU overhead* by rendering both views in a single render pass [20]. To that end, the frame buffer size is doubled, assigning each half to one eye. Instanced rendering is used to avoid duplicated draw calls. It duplicates each point and applies the corresponding view transform at the vertex shader stage. To minimize the probability of points spilling over into the opposite half of the frame buffer, we apply a heuristic that shrinks points close to the border. Preventing such artifacts completely would require to discard affected fragments explicitly, which would be incompatible to early fragment testing as required by the techniques presented above.

5 IMAGE OPTIMIZATION TECHNIQUES

The immersiveness of a virtual scene is negatively affected by any kind of visual artifacts or inconsistencies one would not expect in the real-world, such as aliasing, z-fighting, and insufficient or missing depth cues [1]. In 3D point cloud depictions, the most noticeable artifacts arise when points representing a continuous surface are sized inappropriately, resulting in either a holey appearance of those surfaces or visual clutter due to overlapping points (Fig. 8a+b). We aim to minimize such artifacts by applying the following rendering techniques: Adaptive point sizes, paraboloid rendering, image-based post-processing (i.e., edge highlighting, blurring, filling), and multisampling.

5.1 Adaptive Point Sizes

The different nodes of LoD data structures exhibit noticeable differences regarding the point density. Thus, assigning all points a uniform size results in either holes between neighboring points or overlaps and visual clutter. Schütz addresses that issue by adjusting each point's size based on the maximum LoD within its local neighborhood [36]. While we also adjust point sizes adaptively, our technique operates on a per-node instead of a per-point basis, thus, avoiding the need for a separate render pass to calculate each point's LoD. In that regard, our technique is similar to the one proposed by Scheiblauer [35]. However, we use inherently balanced kd-trees in favor of octrees. For each node, we determine its deepest descendant that has been selected for rendering. The adaptive point size for that descendant is then applied to all of its ancestors. Furthermore, we calculate point sizes based on a node's bounding box rather than its LoD since nodes of the same LoD might still feature drastically different point densities. While our technique drastically and effectively reduces holes and overlaps, it does not exclude those artifacts entirely. For example, if nodes selected for rendering form a heavily unbalanced tree, some points might be rendered too small (Fig. 8c). We fill the resulting holes via post-processing.

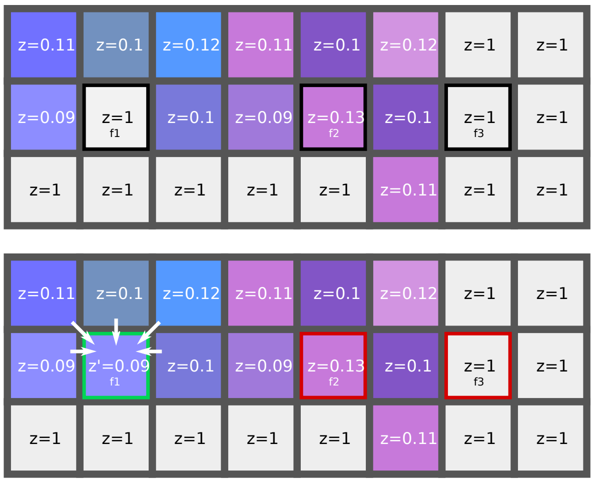


Figure 6: Fragment *f1* is detected as a hole based on depth differences to its neighbors and gets assigned the minimum depth value within its neighborhood; *f2* and *f3* remain unchanged as they fail the distance threshold and the minimum number of neighbors, respectively.

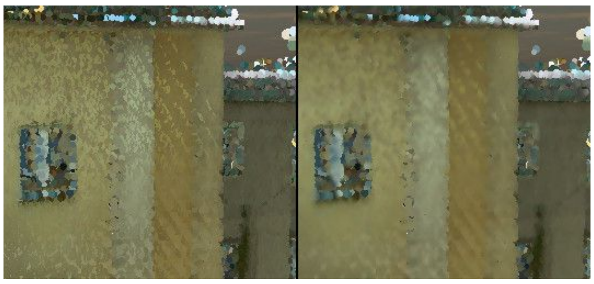


Figure 7: Contrasting color values can be harmonized using blurring to smooth aliasing and z-fighting.

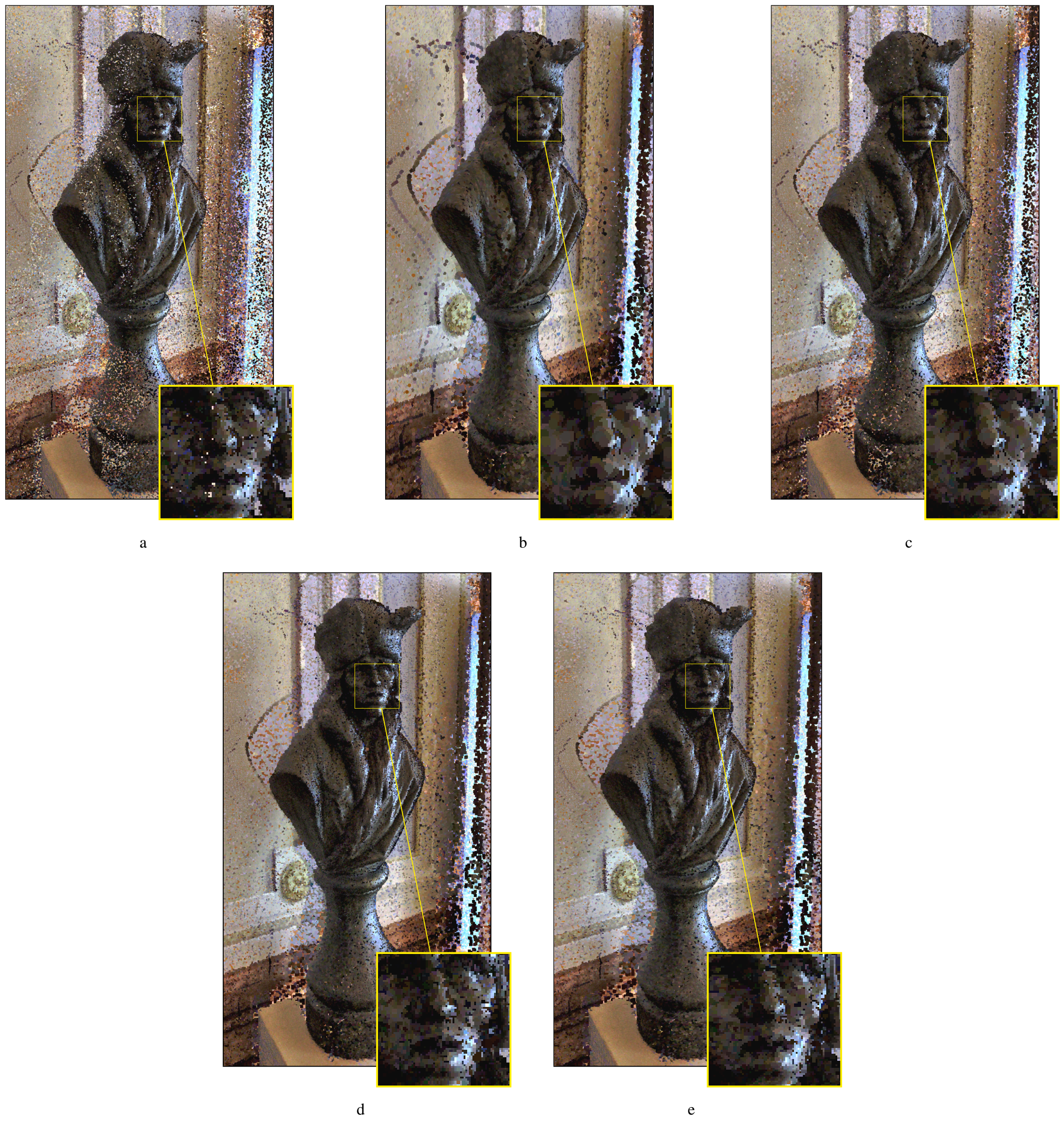


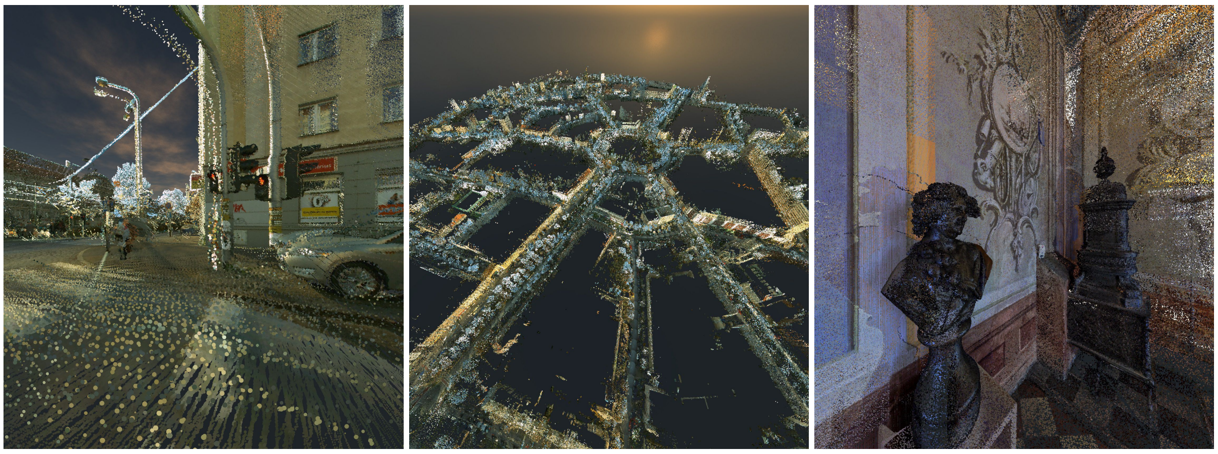
Figure 8: Incorrectly sized points may lead to a holey appearance (a – point size of 1px) or visual clutter (b – point size of 5px). An adaptive point size strikes a balance between both artifact types, but does not eliminate them completely (c). This can be minimized by applying paraboloid rendering (d – diameter of 5px) or filling (e – 5x5 filter kernel and point size of 1px).

5.2 Paraboloid Rendering

Paraboloid rendering is a technique introduced by Schütz [36] that aims to further reduce visual clutter by rendering points not as flat, screen-aligned disks but as paraboloids oriented towards the view position. By adding a depth offset to fragments based on their distance to the corresponding point's center, undesired occlusions are drastically reduced (Fig. 8d). As this technique requires us to modify depth values at the fragment shader stage however, it is incompatible with early fragment testing and thus for most of the techniques discussed in Section 4.

5.3 Post-Processing

We use several post-processing techniques to further improve the visual quality: Screen space ambient occlusion (SSAO) [23] and eye-dome lighting (EDL) [5] add depth cues and highlight silhouettes, blurring [22] smoothes aliasing and z-fighting (Fig. 7). Furthermore, we fill remaining holes between points representing the same surface (Fig. 8e). To that end, we adapt the tech-



a *Pedestrian view of a mobile mapping scan.* b *Birds-eye view of a mobile mapping scan.* c *Close-up view of a terrestrial indoor scan.*

Figure 9: Scenes used during the performance evaluation.

nique presented by Dobrev et al. [10], applying two one-dimensional filter kernels instead of a single two-dimensional one for a performance speed up. The filter kernel checks a pixel's neighborhood for significant depth differences and overwrites corresponding pixels with interpolated values from those neighbors being closest to the view position (Fig. 6).

5.4 Multisampling

A technique to further smooth aliasing and reduce z-fighting would be multisampling, which provides a smoother color transition between neighboring fragments by sampling them several times. While this technique also reduces the visibility of outliers, we ultimately opted against it as it would require us to render fragments several times, thus, drastically affecting the performance, especially when combined with post-processing effects.

6 PERFORMANCE EVALUATION

We have implemented the presented rendering system using C++, OpenGL, GLSL, and OpenVR[3]. The test system featured an Intel Core i7-5820K CPU, 16 GB main memory (DDR4, 1200 MHz), a GeForce GTX 980 with 4096 MByte device memory(GDDR5, driver version 390.77) as well as an HTC Vive as the output device. Measurements on an Oculus Rift lead to comparable, slightly better results due to the tighter view frustum. The test data sets comprised a mobile mapping scan of an urban area (2.6 billion points) and a terrestrial indoor scan of an individual site (1.5 billion points). The performance evaluation was conducted for three different scenes (Fig. 9): A close up and a zoomed out view of the urban area (Scene 1 and 2) as well as a close up view of the individual site (Scene 3). We disabled the dynamic memory budget, which guarantees the constant framerate of at least 90 fps, for the evaluation to ensure the comparability of the measured values.

[3] https://github.com/ValveSoftware/openvr

Both *hidden mesh* and *reverse painter's* algorithm improve the rendering performance. However, their effectiveness varies, depending on the number of affected fragments (Table 1). *Single-pass stereo rendering* proved to be less effective as the primary rendering bottleneck is the GPU, not the CPU. On the contrary, the technique even slows the rendering pipeline as view frustum culling needs to be combined for both eyes, thus notably increasing the amount of unnecessarily rendered points per side. Regarding image optimization techniques, *paraboloid rendering* and *multisampling* -as expected- significantly reduces the rendering performance (Table 2) and thus should only be used, if the z-fighting becomes too prominent and significantly affects the immersion. On the other hand, post-processing effects and adaptive point sizes only have a moderate performance impact. While combining all post-processing techniques would amount to a significant performance drop, doing so will hardly be necessary. As an example, *EDL* and *SSAO* aim for similar effects, whereas *blurring* will only be noticeable in specific scenes, e.g., if color values of neighboring points are inconsistent due to an erroneous capturing process.

7 CONCLUSIONS AND FUTURE WORK

We have presented a point-based and image-based multi-pass rendering technique that allows for visualizing massive 3D point clouds on VR devices in non immersion-breaking quality (i.e., reducing visual artifacts) and at nausea-avoiding frame rates (i.e., around 90 fps). The multi-pass approach offers many degrees of freedom for graphics and application design because the applied rendering techniques can be selected and configured at runtime. We envision

	Scene 1	Scene 2	Scene 3
#Rendered points	19.8M	6.9M	11.6M
Default	15.93ms	9.23ms	12.15ms
Hidden Mesh	15.59ms	9.19ms	11.87ms
Reverse Painter's	12.95ms	9.27ms	11.11ms
Single-Pass Stereo	17.48ms	9.82ms	13.54ms

Table 1: Average rendering performance of performance optimization techniques in ms/frame. All test runs include view frustum and detail culling. Dynamic memory budget was disabled to ensure comparability of measured values.

	Scene 1	Scene 2	Scene 3
#Rendered points	19.8M	6.9M	11.6M
Default	12.82ms	9.21ms	10.77ms
Adaptive Pt. Size	13.88ms	9.48ms	12.46ms
SSAO		+ 2.67 ms	
EDL		+ 0.32 ms	
Filling		+ 1.07 ms	
Blurring		+ 2.17 ms	
Multisampling	17.91ms	10.14ms	16.94ms
Paraboloids Def.	12.72ms	10.77ms	10.26ms
Paraboloids	15.17ms	18.45ms	15.62ms

Table 2: Average rendering performance of image optimization techniques in ms/frame. For paraboloids, hidden mesh rendering and the reverse painter's algorithm were deactivated and an oversized point size (5 px) was used. Dynamic memory budget was disabled to ensure comparability of measured values.

the presented approach to be highly beneficial for applications in the fields of digital documentation, preservation, and presentation of natural and cultural heritage as it allows users to remotely explore and inspect *digital twins* of endangered or hardly accessible sites in a much more immersive way than existing solutions [24]. In building information modeling or urban planning and development, it facilitates planning processes by providing efficient means to integrate additional, mesh-based geometry such as 3D floor plans or building models into the generated stereoscopic 3D point cloud depictions. Tests on data sets with up to 2.6 billion points show the feasibility and scalability of our rendering system.

Future work could focus on performance improvements by distributing the stereo rendering across two separate GPUs as proposed by [40]. To support hardware that is not specifically designed for VR, we plan to integrate web-based rendering concepts for thin clients [18, 16]. Using a centralized server to generate and distribute stereoscopic images would support VR applications on mobile devices with limited CPU and GPU capabilities. In addition, many applications require more sophisticated interaction techniques such as placing annotations or directly manipulating data subsets. We plan to investigate how such interaction techniques can be integrated into the presented rendering system.

ACKNOWLEDGEMENTS

We thank Felix Thiel for his contributions to the implementation. Data has been provided by *virtualcitySYSTEMS*, *SHH Sp. z.o.o*, and *illustrated architecture*.

8 REFERENCES

[1] Akenine-Möller, T., Haines, E., Hoffman, N. Real-time rendering (4th ed.). CRC Press, 2018.

[2] Berg, L.P., Vance, J.M. Industry use of virtual reality in product design and manufacturing: a survey. Virtual reality 21, No. 1, pp.1–17, 2017.

[3] Berger, M., Tagliasacchi, A., Seversky, L., Alliez, P., Guennebaud, G., Levine, J., Sharf, A., Silva, C. A survey of surface reconstruction from point clouds. Computer Graphics Forum 36, No.1, pp. 301–329, 2017.

[4] Botsch, M., Kobbelt, L. High-quality point-based rendering on modern GPUs. In Proc. Pacific Graphics, pp. 335–343, 2003.

[5] Boucheny, C. Interactive Scientific Visualisation of Large Datasets: Towards a Perception-based Approach. PhD thesis, Université Joseph Fourier, 2009.

[6] Boulch, A., Saux, B.L., Audebert, N. Unstructured point cloud semantic labeling using deep segmentation networks. In Proc. 3DOR, pp. 17–24, 2017.

[7] Butler, H., Finnegan, D.C., Gadomski, P.J., Verma, U.K. Plas.io: Open Source, Browser-based WebGL Point Cloud Visualization. In AGU Fall Meeting Abstracts, 2014.

[8] Chen, D., Wang, R., Peethambaran, J. Topologically aware building rooftop reconstruction from airborne laser scanning point clouds. IEEE TGRS 55, No. 12, pp. 7032-7052, 2017.

[9] Cura, R., Perret, J., Paparoditis, N. A scalable and multi-purpose point cloud server (PCS) for easier and faster point cloud data management and processing. ISPRS P& RS 127, pp. 39–56, 2017.

[10] Dobrev, P., Rosenthal, P., Linsen, L. An image-space approach to interactive point cloud rendering including shadows and transparency. Computer Graphics and Geometry 12, No.3, pp. 2–25, 2010.

[11] Eitel, J.U., Höfle, B., Vierling, L.A., Abellán, A., Asner, G.P., Deems, J.S., Glennie, C.L., Joerg, P.C., LeWinter, A.L., Magney, T.S., Mandlburger,

G. Beyond 3-D: The new spectrum of lidar applications for earth and ecological sciences. Remote Sensing of Environment 186, pp. 372–392, 2016.

[12] Elseberg, J., Borrmann, D., Nüchter, A. One billion points in the cloud–an octree for efficient processing of 3D laser scans. ISPRS P & RS 76, pp. 76–88, 2013.

[13] Gao, Z., Nocera, L., Wang, M., Neumann, U. Visualizing aerial LiDAR cities with hierarchical hybrid point-polygon structures. In Proc. GI, pp. 137–144, 2014.

[14] Goswami, P., Erol, F., Mukhi, R., Pajarola, R., Gobbetti, E. An efficient multi-resolution framework for high quality interactive rendering of massive point clouds using multi-way kd-trees. The Visual Computer 29, No. 1, pp. 69–83, 2013.

[15] Gross, M., Pfister, H. (Eds.). Point-based graphics. Morgan Kaufmann, 2011.

[16] Gutbell, R., Pandikow, L., Coors, V., Kammeyer, Y. A framework for server side rendering using OGC's 3D portrayal service. In Proc. Web3D, pp. 137–146, 2016.

[17] Hämmerle, M., Höfle, B., Fuchs, J., Schröder-Ritzrau, A., Vollweiler, N., Frank, N. Comparison of kinect and terrestrial lidar capturing natural karst cave 3-d objects. IEEE GRSL 11, No. 11, pp. 1896–1900, 2014.

[18] Hagedorn, B., Thum, S., Reitz, T., Coors, V., Gutbell, R. OGC 3D Portrayal Service 1.0, OGC Implementation Standard 1.0, Open Geospatial Consortium, 2017.

[19] Hughes, J.F., van Dam, A., McGuire, M., Sklar, D.F., Foley, J.D., Feiner, S.K., Akeley, K. Computer graphics: principles and practice (3rd ed.), Addison-Wesley Professional, 2013.

[20] Johansson, M. Efficient stereoscopic rendering of building information models (BIM). JCGT 5, No.3, 2016.

[21] Langner, T., Seifert, D., Fischer, B., Goehring, D., Ganjineh, T., Rojas, R. Traffic awareness driver assistance based on stereovision, eye-tracking, and head-up display. In Proc. IEEE ICRA, pp. 3167–3173, 2016.

[22] Lukin, A. Tips & tricks: Fast image filtering algorithms. In Proc. GraphiCon, pp. 186–189, 2016.

[23] Mittring, M. Finding next gen: Cryengine 2. In ACM SIGGRAPH courses, pp. 97–121, 2007.

[24] Martinez-Rubi, O., de Kleijn, M., Verhoeven, S., Drost, N., Attema, J., van Meersbergen, M., van Nieuwpoort, R., de Hond, R., Dias, E., Svetachov, P. Using modular 3D digital earth applications based on point clouds for the study of complex sites. Intl. Journal of Digital Earth 9, No. 12, pp. 1135–1152, 2016.

[25] Musialski, P., Wonka, P., Aliaga, D.G., Wimmer, M., Gool, L.V., Purgathofer, W. A survey of urban reconstruction. Computer Graphics Forum 32, No. 6, pp. 146–177, 2013.

[26] Nebiker, S., Bleisch, S., Christen, M. Rich point clouds in virtual globes–A new paradigm in city modeling?. Computers, Environment and Urban Systems 34, No. 6, pp. 508–517, 2010.

[27] Ostrowski, S., Jozkow, G., Toth, C., Vander Jagt, B. Analysis of point cloud generation from UAS images. ISPRS Annals 2, No. 1, pp. 45–51, 2014.

[28] Pătrăucean, V., Armeni, I., Nahangi, M., Yeung, J., Brilakis, I., Haas, C. State of research in automatic as-built modelling. Advanced Engineering Informatics 29, No. 2, pp. 162–171, 2015.

[29] Peters, R., Ledoux, H. Robust approximation of the Medial Axis Transform of LiDAR point clouds as a tool for visualisation. Computers & Geosciences 90, pp. 123–133, 2016.

[30] Preiner, R., Jeschke, S., Wimmer, M. Auto Splats: Dynamic Point Cloud Visualization on the GPU. In Proc. EGPGV, pp. 139–148, 2012.

[31] Remondino, F., Spera, M.G., Nocerino, E., Menna, F., Nex, F., Gonizzi-Barsanti, S. Dense image matching: comparisons and analyses. In Proc. DigitalHeritage, pp. 47–54, 2013.

[32] Richter, R., Discher, S., Döllner, J. Out-of-core visualization of classified 3d point clouds. 3D Geoinformation Science, pp. 227–242, 2015.

[33] Rosenthal, P., Linsen, L. Image-space point cloud rendering. In Proc. CGI, pp. 137–143, 2008.

[34] Saito, T. and Takahashi, T. Comprehensible Rendering of 3-D Shapes. In Proc. SIGGRAPH Computer Graphics, pp. 197–206, 1990.

[35] Scheiblauer, C., Pregesbauer, M. Consolidated Visualization of Enormous 3D Scan Point Clouds with Scanopy. In Proc. CHNT, pp. 242–247, 2011.

[36] Schütz, M. Potree–Rendering Large Point Clouds in Web Browsers. Master thesis, Technische Universität Wien, 2016.

[37] Schütz, M. Massive Time-Lapse Point Cloud Rendering in Virtual Reality. Presentation at SIGGRAPH, 2016.

[38] Sitek, A., Huesman, R.H., Gullberg, G.T. Tomographic reconstruction using an adaptive tetrahedral mesh defined by a point cloud. IEEE T-MI 25, No. 9, pp. 1172–1179, 2006.

[39] Vlachos, A. Advanced VR Rendering. Presentation at GDC, 2015.

[40] Vlachos, A. Advanced VR Rendering Performance. Presentation at GDC, 2016.