CEIG – Spanish Computer Graphics Conference (2019)
A. Jarabo and D. Casas (Editors)

View-dependent Hierarchical Rendering of Massive Point Clouds through Textured Splats

Marc Comino, Antonio Chica, Carlos Andujar

ViRVIG, Computer Science Department, Universitat Politècnica de Catalunya, Barcelona, Spain

Abstract

Nowadays, there are multiple available range scanning technologies which can capture extremely detailed models of real-world surfaces. The result of such process is usually a set of point clouds which can contain billions of points. While these point clouds can be used and processed offline for a variety of purposes (such as surface reconstruction and offline rendering) it is unfeasible to interactively visualize the raw point data. The most common approach is to use a hierarchical representation to render varying-size oriented splats, but this method also has its limitations as usually a single color is encoded for each point sample. Some authors have proposed the use of color-textured splats, but these either have been designed for offline rendering or do not address the efficient encoding of image datasets into textures. In this work, we propose extending point clouds by encoding their color information into textures and using a pruning and scaling rendering algorithm to achieve interactive rendering. Our approach can be combined with hierarchical point-based representations to allow for real-time rendering of massive point clouds in commodity hardware.

CCS Concepts

•***Computing methodologies*** → ***Rendering; Texturing; Point-based models;*** *Image-based rendering;*

1. Introduction

State-of-the-art Lidar equipment can acquire more than one million samples per second and several billion points per scan [CACB17]. The high resolution and extremely accurate depth measurements of terrestrial Lidar scanners make them very suitable for digitizing buildings and sites in archaeology, cultural heritage and architecture applications. Unfortunately, the resulting high-resolution point clouds do not fit in main memory and cannot be rendered without resorting to out-of-core acceleration techniques only available in high-end point-based rendering tools [MRVVM*15].

Acceleration techniques commonly include the use of GPU for splat rendering [PJW12], visibility culling [RL00, KTB07], and hierarchical representations to quickly retrieve suitable points at different levels of details [RL00, GM04, GEM*13, RDD15, DRD18]. Although these methods do scale well with huge point datasets, maintaining an ideal one-sample per pixel ratio during interactive navigation is extremely challenging due to the out-of-core nature of these algorithms. When exploring buildings and urban models (in contrast to e.g. a single digitized statue), fast camera movements during navigation might require loading a substantial amount of point data from disk/network, which adds latency and often results in some frames showing point splats covering large screen regions. Since most point-based rendering approaches store a single color per point sample, high-frequency color details (which are key in many urban models) are not reproduced until all the required point data has been moved to main memory.

In this paper we present a hierarchical point-based rendering method that employs textured splats at varying resolution as rendering primitive. The use of textured splats is not new, but existing approaches using color textures [SSLK13, BLMD18] do not target real-time rendering, but the off-line creation of high-quality views for image-based localization.

The key ingredient of our approach is a representation of the color data for each individual scan. Combined with a hierarchical representation of the point cloud geometry, our approach allows for efficient and scalable rendering of point clouds through textured splats. The major benefit of textured splats is that they allow rendering frames with much less (and larger) primitives while still maintaining high-quality images and preserving high-resolution image details. As a consequence, textured splats allow for more aggressive level-of-detail simplification with little impact on image quality. Since texture coordinates can be computed directly from point sample coordinates, our approach can be combined with most hierarchical point-based rendering approaches. The texture encoding assumes high-resolution 3D scans are available from a few scanner locations (a typical situation when acquiring buildings with terrestrial Lidar equipment).

The rest of the paper is organized as follows. Section 2 reviews previous work on point-based rendering. Section 3 describes our

DOI: 10.2312/ceig.20191203

approach, detailing the encoding of color data through panoramic images suitable for textured splats. We present results with a massive Lidar point cloud in Section 4 and discuss future research avenues in Section 5.

2. Previous Work

Point-based representations have been gaining popularity in recent years due to a variety of reasons. For example, they offer more flexibility for representing surfaces, in comparison with meshes, and they are also the natural output of both Lidar and Structure-from-Motion based scanning technologies. One of the first rendering approaches for point clouds was proposed by Zwicker et al. [ZPVBG01]. It consists on rendering one oriented surface splat for each point. The overlapping of the splats, along with a carefully designed anti-aliasing scheme, guarantees a hole free visualization. Rosenthal and Linsen [RL08] introduced an alternative approach that renders the point cloud first and then filters the resulting color, depth and normal channels to fill any holes left. The result is reliable enough that edge detection algorithms can extract silhouettes and feature lines.

To optimally render a point cloud using surface splatting, it is necessary to perform a sub-sampling process first. The algorithm of Wu and Kobbelt [WK04] applies this transformation taking into account the particular geometrical properties of circular and elliptical splats. A global optimization scheme computes the minimum set of splats necessary to cover the surface completely, ensuring that the approximation error is below a threshold provided by the user.

Point clouds resulting from scans cannot be rendered instantly, since many algorithms assume a certain point density or the availability of a correct normal computed at each point. Wimmer and Scheiblauer [WS06] proposed a technique for massive point model visualization by combining two data structures. First, a memory optimized version of sequential point trees allows rendering point sets sequentially on the GPU. Second, nested octrees are used to achieve an out-of-core solution. The combined result allows for the direct exploration of a point cloud, without the need for post-processing.

Preiner et al. [PJW12] proposed another visualization method based on splats interactively generated on the GPU. It relies on a fast GPU-based algorithm for estimated tangent planes and they claim they are able to process point clouds of about 10 million points. However, nowadays technology can easily acquire point clouds with one or two order of magnitude more points. For this reason, hierarchical rendering and visibility culling techniques are needed. Katz et al. [KTB07] present their HPR operator which is able to determine the visibility of points, to avoid rendering occluded ones.

However, most authors have focused on out-of-core hierarchical approaches for rendering. QSplat, introduced by Rusinkiewicz and Levoy [RL00], starts by pre-computing a bounding sphere tree which is stored out-of-core. This structure is used at run time to produce points at different levels of details as well as to perform visibility queries. Follow-up approaches focus on further exploiting GPU capabilities for hierarchical point rendering. Layered point clouds [GM04], consist on rendering by refining multi-resolution blocks guided by the contribution of each point measured as their projected size. Finally, Goswami et al. [GEM*13] explore using multi-way kd-trees as the hierarchical structure for managing the point cloud information.

Nevertheless, all of the previous approaches are designed to accurately reproduce the geometric properties of point clouds at render time, but do not take into account other properties such as color. When points representing lower levels of detail are rendered as splats, they are rendered with uniform properties across the surface of the splat (e.g. with an uniform color), even if the original signal had a higher frequency. Because of this, there is also a family of image-based rendering approaches which exploit the implicit 2.5D structure of point clouds that have been captured using Lidar technology. Benedetto et al. [DBGBR*14] propose exploring virtual environments by moving between specific points where a panoramic view has been generated. When moving between points, a pre-computed video sequence is played emulating the motion. More recently, Comino et al. [CACB17] proposed converting point clouds to 360-degree panoramas encoding properties such as geometry, normals and color. These are fed into tessellation shaders which generate the scene geometry to allow interactive inspection. However, tessellation is fixed at a certain level and is not refined for closer views. This means that part of the geometric information is lost.

A few recent works do use textured splats [SSLK13, BLMD18] but to the best of our knowledge these approaches do not pursue real-time rendering but the generation of high-quality views. Sibbing et al. [SSLK13] present different point-based rendering techniques for terrestrial laser scan data. Their major goal is not real-time point-based rendering, but the creation of high-quality views for image-based localization. The authors combine point data from a few laser scan locations with large collections of registered images to create new views from which local image features can be extracted and matched to those of the query image. Since local feature extractors used for correspondences (e.g., SIFT) are only tolerant to small perspective changes, new views substantially improve 3D localization results. Their main contribution is the use of image completion techniques to fill holes and better preserve color gradients.

In the same spirit, Bui et al. [BLMD18] also target the creation of high-quality images from co-registered high-resolution photographs. They use these photographs to train a deep neural network for super-resolution. The proposed network takes a splat rendering image as input and generates the corresponding high-quality image, which can be used for higher recognition rates in image-based localization. Unfortunately, the generator network is quite deep (up to 80 layers) and cannot be used to render all frames in real-time. For real-time navigation, they suggest to do super-resolution on several synthetic views selected manually, and use these views as textures for textured splats, but no further details are provided. None of these approaches do consider level-of-details nor the amount of memory required for storing the image datasets.

Arikan et al. [APS*14] also address the problem of rendering points clouds accompanied by a set of high-resolution photographs. Their approach generates meshes by rendering point-based depth maps from the image cameras. Each of the meshes is textured by

Figure 1: *Stretch-invariant polar capped color map for a given point cloud.*

all the input images. Since meshes overlap, for each screen pixel the algorithm has to decide which fragment should be shown.

In our approach, we combine the panorama representation proposed by Comino et al. [CACB17] with an adaptive splat rendering algorithm, achieving both color and geometry accuracy. We employ a very simple, but effective, hierarchical representation, although our approach could be combined with any of the pre-existing hierarchical out-of-core rendering schemes.

3. Our Approach

3.1. Representation of point locations and color data

We consider that a given scene S has been scanned from n different locations $\mathbf{s}_1,...,\mathbf{s}_k,...,\mathbf{s}_n$ yielding point clouds $C_1,..,C_k,..C_n$. We also assume that there is an available set of geometric transformation matrices resulting from the 3D registration of these point clouds.

A spherical representation P of a point cloud C_k, generated from $\mathbf{s}_k$, is always a valid discrete representation of C_k because occlusions in P are identical to occlusions in C_k. Comino et al. [CACB17] propose to represent a point cloud using three images (color, normal and depth maps) where each pixel represents the cylindrical plane-chart projection of a point. This projection is similar to the one used by terrestrial Lidar scanners to determine the ray direction of the infrared pulses used for depth measurements. However, the biggest flaw of this projection is that the sample density is not uniform across the unit sphere.

Let ψ, θ represent latitude/longitude angles on a unit sphere. Following Snyder et al. [SM01], we propose storing the color information associated to each point cloud C_k by generating one stretch-invariant polar capped map. This map has the advantage of minimizing the distortion across every region of the unit sphere, obtaining a more uniform sample density. In particular, we use a plane-chart cylindrical projection for points whose spherical coordinate ψ is in the range of $[-\pi/4,\pi/4]$ and two azimuthal equidistant polar caps, one for points with $\psi \in [-\pi/2,-\pi/4)$ and another one for points with $\psi \in (\pi/4,\pi/2]$, as shown in Figure 1. One polar capped map t_k is generated for each scan location $\mathbf{s}_k$, which is used as texture for point cloud C_k. Hence, a total of n texture maps are computed.

We define a forward map from texture coordinates (s,t) to 3D points on the unit sphere as follows. Let u_b be $6s-0.5$, v_b be $t-0.5$ and $r_b = \sqrt{u_b^2+v_b^2}$. For the bottom cap ($s \leq 1/6$), the direction (x,y,z) encoded by a texel (s,t) is given by

$$\begin{cases} x = (u_b/r_b)\sin(\pi r_b/2) \\ y = (v_b/r_b)\sin(\pi r_b/2) \\ z = -\cos(\pi r_b/2) \end{cases} \quad (1)$$

Similarly, for the top cap ($1/6 < s \leq 2/6$) we define $u_t = 6s-1.5$, $v_t = t-0.5$ and $r_t = \sqrt{u_t^2+v_t^2}$, and get the direction (x,y,z) by

$$\begin{cases} x = (u_t/r_t)\sin(\pi r_t/2) \\ y = (v_t/r_t)\sin(\pi r_t/2) \\ z = \cos(\pi r_t/2) \end{cases} \quad (2)$$

Finally, for the band across the Equator ($s > 2/6$), we set $\psi = t\pi/2-\pi/4$, $\theta = (1.5s-0.5)2\pi$ and get the direction using a simple equirectangular projection:

$$\begin{cases} x = \cos(\psi)\cos(\theta) \\ y = \cos(\psi)\sin(\theta) \\ z = \sin(\psi) \end{cases} \quad (3)$$

Given a 3D point $\mathbf{p} \in C_k$, we can compute its texture coordinates by using the inverse mapping. Let $(\bar{x},\bar{y},\bar{z})$ be the vector obtained by normalizing the vector from $\mathbf{s}_k$ to $\mathbf{p}$, and let ψ be $\arcsin(\bar{z})$. The texture coordinates for P can be computed as follows.

For the bottom cap ($\psi \leq -\pi/4$), $r = 2\pi\arccos(\bar{z})$ and

$$\begin{cases} s = (\bar{x}r/\sin(\pi r/2)+0.5)/6 \\ t = (\bar{y}r/\sin(\pi r/2)+0.5)/6 \end{cases} \quad (4)$$

Similarly, for the top cap ($\psi > \pi/4$)

$$\begin{cases} s = (\bar{x}r/\sin(\pi r/2) + 1.5)/6 \\ t = (\bar{y}r/\sin(\pi r/2) + 0.5)/6 \end{cases} \quad (5)$$

Finally, for the Equator ($-\pi/4 < \psi <= \pi/4$) we set $\theta = \arctan(\bar{y}, \bar{x})$ and get the texture coordinates using

$$\begin{cases} s = (\theta/(2\pi) + 0.5)/1.5 \\ t = (\psi + \pi/4)/(2\pi) \end{cases} \quad (6)$$

As opposed to Comino et al. [CACB17], where both geometry and color data is encoded in texture maps, we propose storing geometry as regular (x,y,z) data and use a texture map only for color data. A first advantage is that keeping the original point coordinates we avoid the precision loss due to using 16 bits to encode depth in an image format such as PNG. On the other hand, storing color information as texture maps we are able to greatly reduce the disk memory footprint. Lossy image compression formats can be used to achieve further compression, since usually color compression artifacts are tolerable. Note that texture coordinates are not stored since they can be computed directly from the point coordinates at run time, which in turn makes our approach orthogonal to hierarchical arrangements of point samples.

Another advantage of using textures to store color data is that users can use image processing software to edit them (e.g. removing advertising from a building facade photograph). Otherwise, it is a rather tedious task to edit the color directly in the point cloud. The biggest advantage of using this approach though is that we are no longer constrained to using a single color across a splat rendering primitive. The fragment shader can compute different texture coordinates for each fragment generated by the splat, greatly improving the rendering results.

Besides RGB color data, the texture might also include an alpha mask indicating the existence of a sample along the scanning direction associated to each pixel. This is specially convenient in outdoor scenes for which some scanning directions have no depth measurement. The alpha channel allows to trim the shape of large splats to match the shape of the underlying point samples.

3.2. Hierarchical Rendering

Many of the existing hierarchical rendering algorithms for point clouds are based on traversing out-of-core hierarchical structures. The level of detail for a point is usually chosen by determining its contribution measured by its screen-projected area. Nevertheless, interactively exploring large areas potentially means loading and unloading a large number of points from disk to GPU. In this scenario, rendering a different element (raw point or higher level representative) for each screen pixel can become too expensive.

We design our algorithm taking inspiration from the field of botanical rendering. Rendering leaves in botanical trees is strongly connected to rendering points. Both elements are unorganised, rendered in large amounts, and greatly impact the realism of a scene.

Figure 2: *Renders of a point cloud using simple splats (left) and our method (right). We are using a tenth of the original points, increasing the splatting radius to fill the holes. Regular splats appear completely flat and the high frequency detail is lost. Our method assigns a texture coordinate to each fragment and thus we are able to reproduce the high frequency features for both color (rows 1 through 3) and normal data.*

There is a family of methods [CHPR07, NPDD11] which aim at reducing the rendering cost of leaves by pruning part of them and scaling the rest to preserve the overall appearance.

When using any of these hierarchical point-based rendering algorithms, some of the raw points are replaced by higher level representatives. This would be the equivalent to *pruning*. Moreover, if the resulting set of elements is not enough to generate enough fragments to cover the whole surface, then we would upscale the splats, which would be the equivalent to *scaling*. Nevertheless, using regular scaled splats yields poor results, as the splat properties (color, normal) are constant across the splat surface (Figure 2, left column). We propose using our point cloud representation, with associated color maps, as a tool to improve any of the existing hierarchical rendering approaches and to enable more aggressive pruning of the cloud.

We propose drawing each point $\mathbf{p} \in C_k$ as an oriented quad. The fragment shader receives its interpolated position and the corresponding position $\mathbf{s}_k$. Per-fragment texture coordinates are computed using Equations 4-6. Texture coordinates can be used to retrieve fragment properties such as color and normals. The alpha channel of the color texture can be used to discard fragments outside the silhouette of the model and the distance to the point center can be used to generate circular or ellipsoidal splats. Figure 2 shows a comparison of our algorithm against using flat splats. We can recover high frequency color and normal detail rendering a smaller number of samples with larger radii.

4. Results

In order to validate our method, we implemented a very simple hierarchical rendering algorithm for point clouds. We start by generating a discretization of our scene using a uniform grid. The resulting cells are stored out-of-core. We also produce multiple simplified versions for each cell by decimating the number of points by a factor of 10 for each decreasing level of detail. These are kept in main memory.

When interacting with the application, we traverse the cells overlapping the view frustum in front-to-back order (to benefit from early depth culling) and we dynamically select the most appropriate level of detail for each cell, based on the distance from the cell center to the view point. We manually tune the scaling factors for the points at each level of detail in order to achieve visually pleasing results. Despite the use of textured splats already leads to highly-detailed rendered images, when the interaction is stopped and the view point is held still for a few seconds, we load from disk and display the cells containing the original points.

For each point, we render a textured splat and we use the alpha channel of the texture to preserve its original silhouette.

We have tested our algorithm on an Ubuntu Workstation with an Intel Core i7-4790K CPU and a GeForce GTX 970. The test dataset included 31 different 3D scans from a singular XIX century market. This market has an extent of 5,214 m^2 on a city block of about 15,876 m^2. A Leica ScanStation P20 was used to digitize key parts of the building. The raw dataset included 31 ASCII files containing information on 3.487.095.733 points and requiring a total of 157.3 GB. For our tests we used the point clouds acquired from 4 different outdoor locations, amounting to roughly 150 million points. We chose the outdoor scans to better illustrate the preservation of fine silhouette details when large splats are trimmed with the alpha mask. Using the interactive inspection mode, we achieve framerates of up to 600 fps for far-away views and from 300 to 600 fps for closer ones. See supplemental material for a video recorded in real-time. When the interaction is stopped it can take several seconds to load and display the finest level of detail. Fortunately, the use of textured splats makes this last refinement less critical than with classical single-colored splats.

Figure 3 shows multiple renders of this scene, with different resolution levels. Note that even for the level with the most aggressive pruning (keeping one point out of one hundred samples) the final render quality is good for areas with an approximately uniform sampling density.

A video showing a demo of the application can be found in https://www.cs.upc.edu/~virtual/CEIG19/video.mp4

5. Conclusions and future work

We have presented a technique for rendering massive point cloud models using a multiresolution approach. Our method encodes color information (and possibly other properties) in panoramic textures. This allows selecting a point cloud subset such that the number of points displayed is much smaller than the number of fragments covered by the model. The scaling of these points eliminates the holes that could occur between points, while the textures help to represent the color information that would be lost when discarding points for rendering. Consequently using our textured splats results in a sharper image than that obtained by combining a greater number of simple splats.

A possible line of future work would be to research different methods for splat generation. The way points are divided into clusters for simplification affects the shape of the final splats. Because of this, we plan to test other strategies for generating clusters comparing their effects in the visualization of the model.

Acknowledgments

This work has been partially funded by the Spanish Ministry of Economy and Competitiveness and FEDER under grant TIN2017-88515-C2-1-R, and the Spanish Ministry of Education, Culture and Sports PhD grant FPU14/00725.

References

[APS*14] ARIKAN M., PREINER R., SCHEIBLAUER C., JESCHKE S., WIMMER M.: Large-scale point-cloud visualization through localized textured surface reconstruction. *IEEE transactions on visualization and computer graphics 20*, 9 (2014), 1280–1292. 2

[BLMD18] BUI G., LE T., MORAGO B., DUAN Y.: Point-based rendering enhancement via deep learning. *The Visual Computer 34*, 6-8 (2018), 829–841. 1, 2

[CACB17] COMINO M., ANDÚJAR C., CHICA A., BRUNET P.: Error-aware construction and rendering of multi-scan panoramas from massive point clouds. *Computer Vision and Image Understanding 157* (2017), 43–54. 1, 2, 3, 4

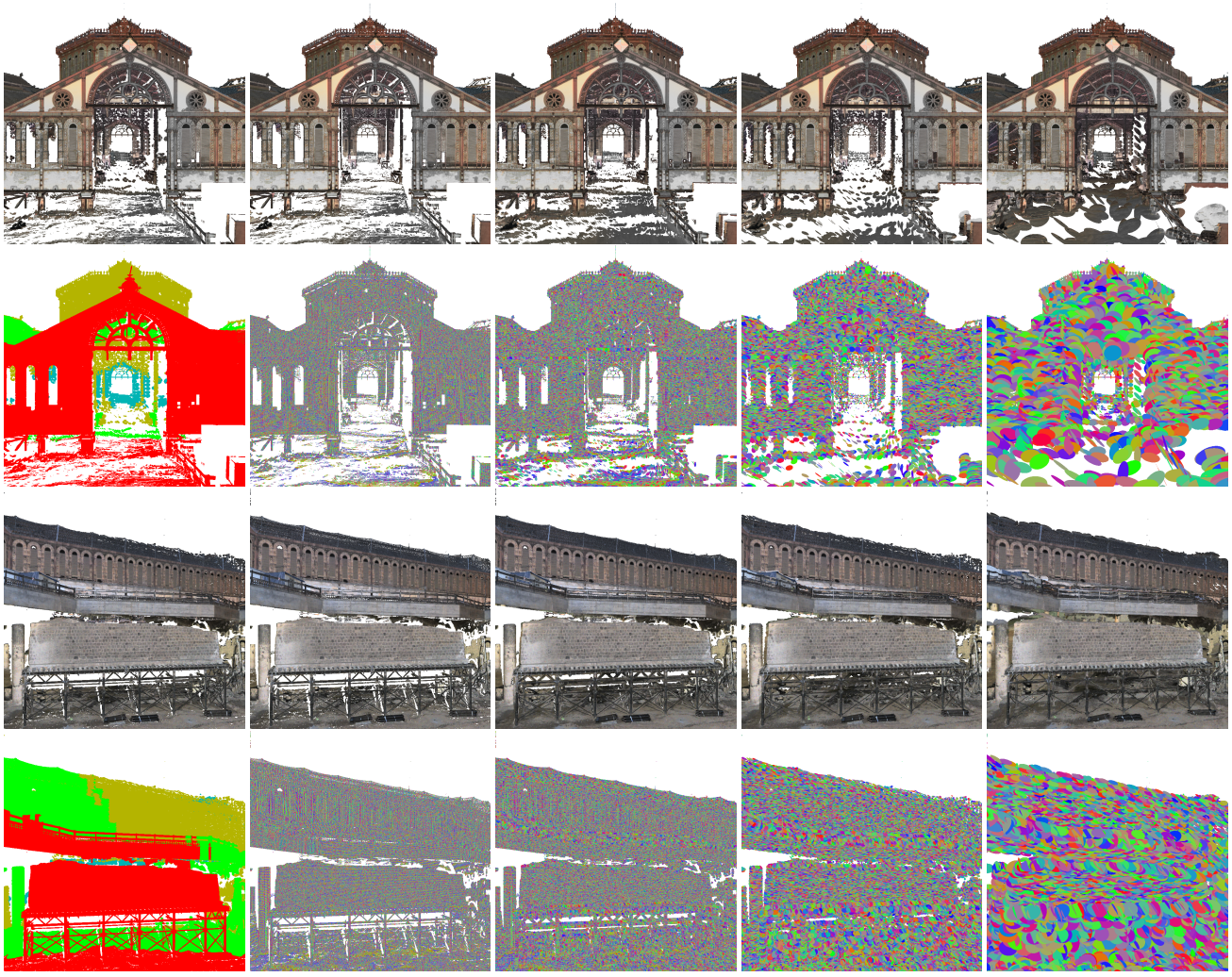

Figure 3: *Renders of point clouds using multiple levels of detail. The first column shows the render using our algorithm. The color scheme indicates original points (red) and decreasing levels of detail (green, yellow, blue). Columns 2 through 5 shows the results of rendering the same scenes choosing a single level of detail. Column 2 shows a render with only the original points, column 3 shows a render with a tenth of the original points, column 4 uses a hundredth and column 5 uses a thousandth.*

[CHPR07] COOK R. L., HALSTEAD J., PLANCK M., RYU D.: Stochastic simplification of aggregate detail. In *ACM Transactions on Graphics (TOG)* (2007), vol. 26, ACM, p. 79. 5

[DBGBR*14] DI BENEDETTO M., GANOVELLI F., BALSA RODRIGUEZ M., JASPE VILLANUEVA A., SCOPIGNO R., GOBBETTI E.: Exploremaps: Efficient construction and ubiquitous exploration of panoramic view graphs of complex 3D environments. *Computer Graphics Forum 33*, 2 (2014), 459–468. 2

[DRD18] DISCHER S., RICHTER R., DÖLLNER J.: A scalable webgl-based approach for visualizing massive 3d point clouds using semantics-dependent rendering techniques. In *Proceedings of the 23rd International ACM Conference on 3D Web Technology* (2018), Web3D '18, pp. 19:1–19:9. 1

[GEM*13] 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*, 1 (2013), 69–83. 1, 2

[GM04] GOBBETTI E., MARTON F.: Layered point clouds. In *Eurographics Symposium on Point Based Graphics* (2004), vol. 227, pp. 113–120. 1, 2

[KTB07] KATZ S., TAL A., BASRI R.: Direct visibility of point sets. *ACM Transactions on Graphics (TOG) 26*, 3 (2007), 24. 1, 2

[MRVVM*15] MARTINEZ-RUBI O., VERHOEVEN S., VAN MEERSBERGEN M., VAN OOSTEROM P., GONÁALVES R., TIJSSEN T., ET AL.: Taming the beast: Free and open-source massive point cloud web visualization. In *Capturing Reality Forum 2015, 23-25 November 2015, Salzburg, Austria* (2015), The Servey Association. 1

[NPDD11] NEUBERT B., PIRK S., DEUSSEN O., DACHSBACHER C.: Improved model- and view-dependent pruning of large botanical scenes. *Computer Graphics Forum 30*, 6 (2011), 1708–1718. 5

[PJW12] PREINER R., JESCHKE S., WIMMER M.: Auto splats: Dynamic point cloud visualization on the gpu. In *Eurographics Symposium on Parallel Graphics and Visualization* (2012), pp. 139–148. 1, 2

[RDD15] RICHTER R., DISCHER S., DÖLLNER J.: Out-of-core visu-

alization of classified 3d point clouds. In *3D Geoinformation Science*. Springer, 2015, pp. 227–242. 1

[RL00] RUSINKIEWICZ S., LEVOY M.: Qsplat: A multiresolution point rendering system for large meshes. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (2000), ACM Press/Addison-Wesley Publishing Co., pp. 343–352. 1, 2

[RL08] ROSENTHAL P., LINSEN L.: Image-space point cloud rendering. In *CGI 2008 Conference Proceedings* (2008), pp. 136âĂŞ–143. 2

[SM01] SNYDER J., MITCHELL D.: Sampling-efficient mapping of spherical images. *Polar 19* (2001), 0–29. 3

[SSLK13] SIBBING D., SATTLER T., LEIBE B., KOBBELT L.: Sift-realistic rendering. In *2013 International Conference on 3D Vision-3DV 2013* (2013), IEEE, pp. 56–63. 1, 2

[WK04] WU J., KOBBELT L.: Optimized sub-sampling of point sets for surface splatting. *Computer Graphics Forum 23*, 3 (2004), 643–652. 2

[WS06] WIMMER M., SCHEIBLAUER C.: Instant points: Fast rendering of unprocessed point clouds. In *Proceedings Symposium on Point-Based Graphics 2006* (2006), Eurographics, Eurographics Association, pp. 129–136. 2

[ZPVBG01] ZWICKER M., PFISTER H., VAN BAAR J., GROSS M.: Surface splatting. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), ACM, pp. 371–378. 2