High-Performance Graphics (2025)
A. Knoll and C. Peters (Editors)

LidarScout: Direct Out-of-Core Rendering of Massive Point Clouds

P. Erler[1] and L. Herzberger[1] and M. Wimmer[1] and M. Schütz[1]

[1]TU Wien, Austria

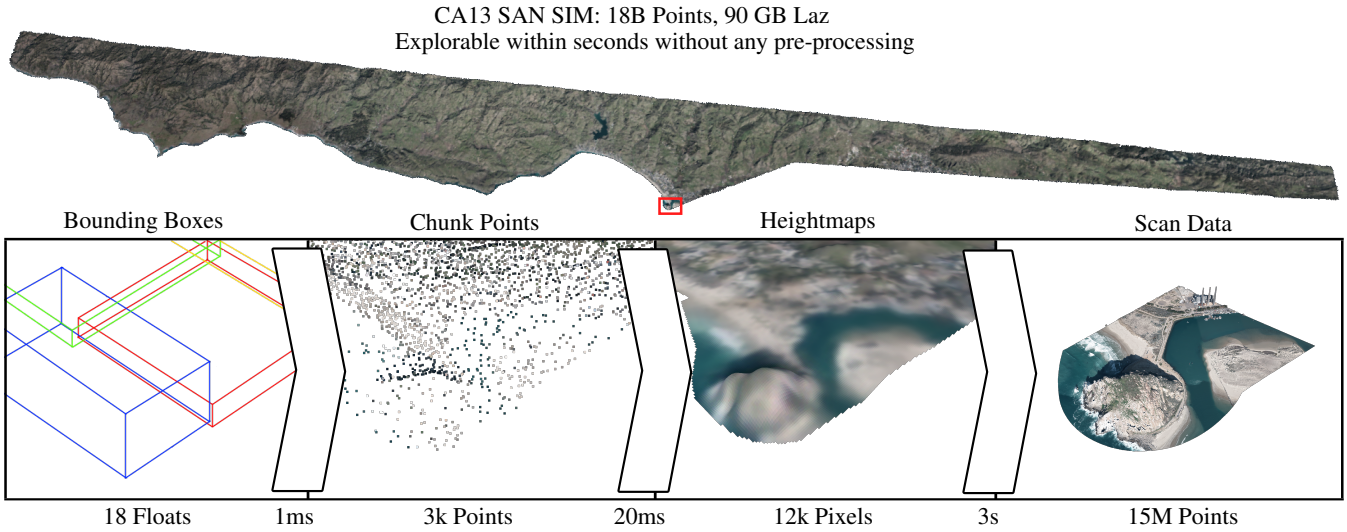


Figure 1: *We present* LIDARSCOUT*, a method to explore huge, compressed point clouds within seconds. The example shown here contains the Morro Bay area focusing on Morro Rock, rendered with three heightmaps of 640m x 640m.*

Abstract

Large-scale terrain scans are the basis for many important tasks, such as topographic mapping, forestry, agriculture, and infrastructure planning. The resulting point cloud data sets are so massive in size that even basic tasks like viewing take hours to days of pre-processing in order to create level-of-detail structures that allow inspecting the data set in their entirety in real time. In this paper, we propose a method that is capable of instantly visualizing massive country-sized scans with hundreds of billions of points. Upon opening the data set, we first load a sparse subsample of points and initialize an overview of the entire point cloud, immediately followed by a surface reconstruction process to generate higher-quality, hole-free heightmaps. As users start navigating towards a region of interest, we continue to prioritize the heightmap construction process to the user's viewpoint. Once a user zooms in closely, we load the full-resolution point cloud data for that region and update the corresponding height map textures with the full-resolution data. As users navigate elsewhere, full-resolution point data that is no longer needed is unloaded, but the updated heightmap textures are retained as a form of medium level of detail. Overall, our method constitutes a form of direct out-of-core rendering for massive point cloud data sets (terabytes, compressed) that requires no preprocessing and no additional disk space.

Source code, executable, pre-trained model, and dataset are available at:
https://github.com/cg-tuwien/lidarscout

CCS Concepts

• ***Computing methodologies*** → ***Point-based models; Mesh models; Neural networks; Reconstruction;***

DOI: 10.2312/hpg.20251170

1. Introduction

Many fields require huge terrain scans, including archeology, infrastructure, bathymetry, agriculture, forestry, flood and landslide prediction, geology, climate research, and many more. Improvements in laser scanners and frequent scanning operations (e.g., 3DEP [Sur18] and AHN [Ned23]) result in country-wide data sets comprising hundreds of billions to trillions of points. These are typically stored in a compressed format (LAZ) and can amount to terabytes. Visualizing these data sets requires out-of-core level-of-detail (LOD) structures that allow loading only those tiny subsets necessary for a given viewpoint. However, generating these structures takes hours to days of preprocessing. For example, AHN2, a point cloud of the entire Netherlands, comprises 640 billion points, and constructing an LOD structure took Martinez-Rubi [MRVv*15] 15 days of processing.

With these huge amounts of data, tasks like viewing become non-trivial. Such supposedly simple tasks include having a quick overview, searching for obvious problems like outliers and noise, finding the relevant files for a specific region, and transferring the data. Compression can reduce transfer and storage problems, but makes other tasks even slower. With LIDARSCOUT, we reduce the time it takes to visualize massive data sets from days down to seconds. After a user drops the data set into the application, we quickly read all tiles' bounding boxes as the only global operation. Afterward, we efficiently pick a sparse subsample of the compressed point cloud, generate rough heightmaps, refine them with a small neural network, and render them with a CUDA-based software rasterizer, all prioritizing the user's current viewpoint. When zooming in further, we stream the high-resolution scan data and update the hightmaps.

Our main contributions are:

1. An interactive point cloud viewer for massive terrain scans that requires no pre-processing and no additional disk space.
2. Efficient extraction of a sparse subsample from compressed point clouds (LAZ).
3. A method to predict high-quality heightmaps from this sparse subsample.

2. Related Work

Related work includes fast point-based rendering approaches, particularly of massive data sets, and surface reconstruction, especially those related to constructing heightmaps from sparse point samples. We also briefly explore neural rendering methods, as several point-based neural methods reconstruct high-quality images from sparse point samples, a problem similar to constructing heightmaps from sparse subsamples.

Surface Reconstruction Surface reconstruction aims to recover the underlying object that a point cloud was sampled from. Most surface reconstruction methods work in full 3D to generate a mesh or distance field. Only a few works target 2.5D and directly output heightmaps.

Being one of the first 3D reconstruction methods, BallPivot [BMR*99] runs from point to point in a point cloud, connecting them with edges. Screened Poisson Surface Reconstruction [KH13] is likely the most popular non-data-driven reconstruction method to date, despite requiring normals for fitting an indicator field to the points. BallMerge [POEM24] uses Voronoi balls to recognize the inside and outside of large scans. PPSurf [EFPH*24] is a recent data-driven method predicting a signed-distance field from unoriented points. Many 3D reconstruction methods aim at single solids and tend to fail in open scenes.

Reconstructing heightmaps from point clouds has seen little work in recent years. However, the field of depthmap estimation is very active, and many results can be adapted to heightmaps. Moving Least Squares [LS81] interpolates a smooth surface from scattered data points. The now classic approach for heightmap reconstruction is to generate a Delaunay triangulation and interpolate it linearly. This is done, e.g., in Las2Dem of the popular LASTools [rG15] suite, which was used for SWISSS3D [Swi20], for example. Las2Dem also deals with multiple laser returns falling into the same texel. However, triangulation approaches suffer from the fundamental problem of interpolating within slender triangles. Closely related, most 2.5D reconstruction methods that work on depthmaps are in the context of single-view 3D reconstruction. Recent survey papers [RSL*24, MRC*22] describe the advances of this field from depth-cue-based methods via machine learning and hand-crafted features to deep learning.

Overall, surface reconstruction from point clouds has seen significant advances in recent years, especially on the data-driven side. However, heightmap reconstruction for aerial LIDAR scans has been neglected.

Aerial LIDAR Storage The LAS and LAZ formats are specifically targeted towards aerial laser scanning and thus the two most commonly used formats for massive, country-wide aerial point cloud scans. LAZ is a compressed form of LAS that specifically takes advantage of common patterns in point clouds for efficient and lossless compression [Ise13]. For example, LAZ predicts the next position of a point based on the differences of the previous five points and then entropy encodes the difference between prediction and true position, which takes advantage of the fact that laser scanners observe points in a line-wise and thus predictable fashion.

Point-Based Rendering Levoy and Whitted [LW85] proposed points as a meta-primitive that all other surface primitives can be converted to, or to be directly generated from procedural functions. Since then, point cloud rendering has become a widely popular field due to the vast amount of point samples generated by laser scanners, and the resulting need for higher performance and better quality [KB04]. Surfels [PZVBG00] and EWA-Splatting [ZPVBG02] introduced high-quality rendering approaches for point-based primitives. Botsch et al. [BHZK05] propose an efficient GPU-based implementation for high-quality splatting. Günther et al. [GKLR13] and Schütz et al. [SKW22] improve the rendering performance via compute-based solutions that are faster than the triangle-oriented hardware pipeline. Schütz et al. [SMOW20] introduce a progressive rendering approach that renders random subsets each frame and converges towards the true result over the coarse of several frames. 3D Gaussian Splat-

ting [KKLD23] proposes 3-dimensional Gaussians as a geometric primitive for 3D reconstruction and novel-view-synthesis. While some of the referenced works deal with point-based geometry that comprises position as well as orientation and size/scale, our method is targeted toward point clouds from aerial LIDAR scans, whose geometry is solely described by positions.

Point-Based Levels of Detail Rendering massive data sets requires LOD structures that reduce memory usage and increase performance. QSplat [RL00] proposes a bounding-sphere hierarchy as a means to render large splat models. Sequential Point Trees [DVS03] introduces a similar hierarchy but sequentializes it into an array that allows efficient rendering on the GPU by invoking a draw call for a subset of the array, replacing fine-grained hierarchy traversal by a draw call to a batch of data. Instant Points [WS06] suggests a nested octree that allows for view-dependent LOD. Wand et al. [WBB*08] and Modifiable Nested Octree (MNO) [SW11] propose modifiable structures that enable efficient selection and deletion on large point data sets. Potree [SOW20] and Lidarserv [BDSF22] improve the LOD construction performance of MNOs. Lidarserv and SimLOD [SHW24] both propose incremental LOD construction algorithms that build and display the LOD structure while additional points are loaded. The former focuses on live-capture of point clouds, while the latter focuses on GPU-accelerated LOD construction that can build the structure as fast as points can be streamed from disk.

The largest LOD construction study for point clouds that we are aware of was made by Martinez-Rubi et al. – 640 billion points converted to an MNO in 15 days [MRVv*15].

Neural Point Cloud Rendering Neural rendering uses a neural network to synthesize images for given parameters, rather than generating and rasterizing geometry. Tewari et al. [TFT*20] give an overview in their STAR. Neural Point Cloud Rendering was just a side-note in 2020, with Neural Point-Based Graphics [ASK*20] being the only mention. First, they compute feature vectors for the given points. Then, they rasterize them as high-dimensional points in multiple resolutions. Finally, they feed these raw images into a U-Net [RFB15], which outputs a rendering. Since then, many methods have been proposed [KPLD21, NKH*21, WWG*21, RFS22, RALB22, YGL*23, HFF*23, FRFS24, HFK*24], improving various aspects of the approach. These methods share one drawback: they are made for relatively dense point clouds, typically produced by photogrammetry, and many require camera poses, which are not available in our application.

3. Method

LIDARSCOUT consists of five stages: Quickly loading a sparse subsample of the entire point cloud; generating heightmaps with textures; refining them for the user's viewpoint; loading full-res data in close-up viewpoints and updating heightmaps with full-res data to retain a medium level of detail; and rendering them with a CUDA software rasterizer.

3.1. Data and Data Structures

Massive aerial LIDAR data sets are typically distributed using the compressed LAZ format (e.g., OpenTopography [Pac13], AHN5 [Ned23], 3DEP [Sur18], etc.). For this paper's evaluation, we selected three point clouds from the USA, one from New Zealand, and one from Switzerland, as shown in Table 1. The sizes range from 1.6 billion points (ID15_BUNDS) to 262 billion points (Gisborne+Addendum). The full Switzerland data set would exceed Gisborne (estimated 18 TB Zipped LAS), but due to lack of storage we selected a subset of 18.7 GB. In the context of aerial LIDAR scans, a point's geometry is solely defined by its position, unlike other point-based primitives such as Surfels or Gaussian Splats. In many cases, they also lack colors, which is why we include the data set of Switzerland as a representative example.

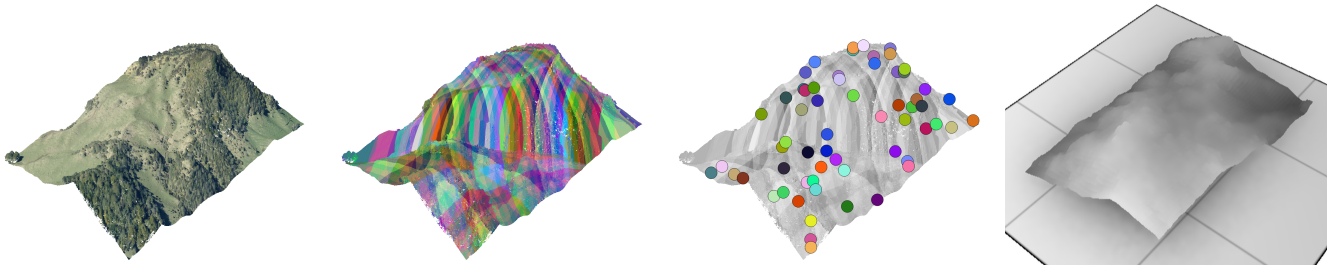
In this paper, we will regularly refer to tiles, chunks, chunk points, and patches, as illustrated in Figure 2. **Tiles** correspond to individual LAZ files. Massive LIDAR data sets are almost always organized in such rectangular tiles, typically covering several hundreds to a thousand meters, and storing several millions of points. **Chunks** are an important concept in the LAZ compression algorithm. LAZ compresses points in chunks of 50 000 points. The first point in each chunk is uncompressed, and subsequent points must be decoded sequentially. Multiple chunks may be decompressed in parallel. **Chunk Points** refer to the first uncompressed point in each chunk. These are important in our approach since they are the only ones we can quickly access without the need to set up an expensive arithmetic decoder. **Patches** are quadratic 640 meter x 640 meter regions for which we reconstruct a 64 x 64 pixel heightmap from the surrounding sparse chunk points.

3.2. Rapidly Creating an Overview for Billions of Points

In the first stage, we initialize an overview of the entire data set with a sparse subsample, which poses two challenges: File I/O is optimized for loading sequential data instead of random subsamples. Also, massive data sets are typically compressed sequentially, which further limits our ability to access random sparse subsets.

On modern SSDs, the first challenge is addressed by investigating their 4 kB random access performance. Similar to accessing RAM [Dre07] or GPU memory [Har13], SSDs are also optimized for coalesced access to a range of bytes rather than a few individual bytes. In case of SSDs, access is typically grouped into sectors of 4 kB, i.e., fetching a single point from disk is about as fast as accessing all points in a sector. However, only the first is uncompressed and thus easily accessible. Modern SSDs are capable of reading about 1 million random 4 kB sectors per second, so we are theoretically able to load a subsample of 1 M points of an arbitrarily large data set – an arguably sufficiently large subset for an overview viewpoint on a 2-megapixel monitor. For example, the model shown in Figure 1 depicts the heightmap model that was constructed from that data set's $\frac{18B}{50k} = 360k$ chunk points.

The second challenge – the industry standard LAZ compression format for massive aerial LIDAR data – puts a limit on the way we can load the initial sparse sample. LAZ uses arithmetic coding to sequentially compress points one after the other, and in turn we also need to decompress them sequentially [Ise13]. Fortunately, points



(a) *Tile* **(b)** *Chunks* **(c)** *Chunkpoints* **(d)** *Patches*

Figure 2: *Massive LIDAR data is stored in rectangular tiles. Tiles store data in compressed chunks of 50k points. Points in a tile are typically stored by timestamp, in this case indicating circular scanning patterns. Chunk points refer to the uncompressed first point of each chunk. Patches denote 640x640 meter (10m/pixel) heightmaps, created from the rapidly loaded chunk points.*

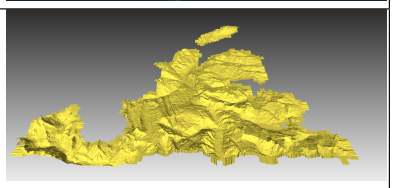
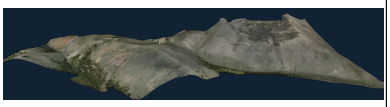
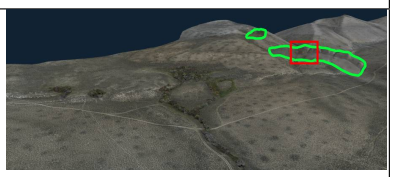
Data set	Image
CA13 OpenTopography 17.7 B Points 80 GB LAZ 20 $Points/m^2$ LIDAR (linear) Screenshot: 56M points	
SWISSS3D Swisstopo 700 M Points (of 714 B) 18.7 GB LAS (of 18 TB) 17 $Points/m^2$ No colors LIDAR (linear)	
BUND_BORA OpenTopography 6.5 B Points 40 GB LAZ 574.22 $Points/m^2$ Photogrammetry	
ID15_BUNDS OpenTopography 1.6 B Points 14.8 GB LAZ 387.24 $Points/m^2$ Photogrammetry	
GISBORNE+Addendum OpenTopography 95 + 167 B points 850 + 1 550 GB LAZ 30.25 $Points/m^2$ LIDAR (circular) Screenshot: 248M points	

Table 1: *Data sets used for* LIDARSCOUT. *The table shows close-up screenshots, the entire map with green outline, and the close-up's area marked as red box.*

are compressed in chunks of typically 50 k points, and the first point in each chunk remains uncompressed, thus we are able to quickly load a sparse subsample made of every 50 000th point. Since LAZ is variable-rate compressed, byte locations of each uncompressed chunk point must be obtained by first reading the file's chunk table.

After all chunk points are loaded, we have a sparse subsample of the entire data set that is sufficiently dense in an overview perspective. When zooming in, holes between points will appear. Since massive aerial LIDAR data sets constitute 2.5D data until one zooms in closely, we propose to fill these holes by constructing high-quality heightmaps from the sparse set of chunk points. We divide the entire map into patches, covering 640x640 meters each, and for each patch, we construct a 64x64 pixel textured heightmap.

3.3. Interpolated Heightmaps

The goal of this part is to convert a region of chunk points to rough, textured heightmaps, which we will later refine with a neural network. For any patch of 64x64 pixels (10m/pixel) we first construct two 96x96 pixel heightmaps using nearest-neighbor (NN) and linear interpolation of the triangulated chunk points. The additional padding is applied to avoid seams between adjacent learned heightmaps.

We receive $\mathcal{P}_{ms}$, the relevant chunk points for the current patch, from a grid-based data structure. We transform these points from model space to patch space as follows: $\mathcal{P} = (\mathbf{p} - \mathbf{c})/r, \mathbf{p} \in \mathcal{P}_{ms}$, where $\mathbf{c} \in \mathbb{R}^2$ is the 2D patch center and $r \in \mathbb{R}$ is the padded patch radius. The triangulation and interpolation for the heightmaps and textures work as described in Algorithm 1, omitting minor implementation details and optimizations for clarity. At the core, we fill a face map indicating which triangle of the triangulation covers which pixel. This is mostly done by performing Flood-Fill from the triangle centroids. Due to rasterization, some pixels of very slender triangles may be disconnected. We catch those in a second Flood-Fill step, started at every pixel inside the bounding box of the triangle. The Flood-Fill works in an 8-connected neighborhood. It fills pixels if they are inside a triangle by checking their barycentric coordinates. After collecting the triangle IDs, we interpolate linearly, again with barycentric coordinates. This is only possible inside the convex hull of the triangulation. Therefore, we fall back to NN interpolation on the outside using a KD-Tree of $\mathcal{P}$ for the necessary speed. The NN interpolation uses the same KD-Tree for all pixels.

Algorithm 1 Patch-Space Chunk Points to Heightmaps

Require: Chunk points $P = \{p_i\}$ (with $p_i \in [-1,1]^2$), height values $h = \{h_i\}$, (optional) color $rgb = \{c_i\}$, grid resolution res
Ensure: Heightmaps $\texttt{hm}_{\text{nn}}$, $\texttt{hm}_{\text{lin}}$, mask `face_map`

// *Initialization*
$N \leftarrow res^2$
$K \leftarrow \text{KDTree}(P)$
Add corner padding points (with NaN values) to P, h and rgb
$\mathcal{T} \leftarrow \text{DelaunayTriangulate}(P)$
$G \leftarrow$ generate regular grid of $res \times res$ points over $[-1,1]^2$
Initialize mask array $\texttt{face\_map}[1..N] \leftarrow -2$
Allocate $\texttt{hm}_{\text{nn}}[1..N]$ and $\texttt{hm}_{\text{lin}}[1..N]$
// *FloodFill from Triangle Centroids*
for all triangles T_i in $\mathcal{T}$ **do**
  $c_i \leftarrow$ centroid of T_i
  $g_{\text{id}} \leftarrow$ index of G-grid point closest to c_i
  FloodFill(g_{id}, i, `face_map`, $\mathcal{T}$, G, res)
end for
// *FloodFill from Disconnected Rests*
for all triangles T_i in $\mathcal{T}$ **do**
  $B_i \leftarrow$ bounding box of T_i intersected with $[-1,1]^2$
  for all grid points g_j in B_i **do**
    if $\texttt{face\_map}[j] = -2$ **and** $g_j \in T_i$ **then**
      FloodFill(j, i, `face_map`, $\mathcal{T}$, G, res)
    end if
  end for
end for
// *Remove Face IDs of Padding Triangles*
for all $j = 1 \ldots N$ with $\texttt{face\_map}[j] \geq 0$ **do**
  $i \leftarrow \texttt{face\_map}[j]$
  if any vertex of triangle $\mathcal{T}[i]$ is a padding vertex **then**
    $\texttt{face\_map}[j] \leftarrow -1$
  end if
end for
// *Linear Interpolation in Convex Hull*
for all j with $\texttt{face\_map}[j] \geq 0$ **do**
  $i \leftarrow \texttt{face\_map}[j]$
  $T \leftarrow \mathcal{T}[i]$
  bary $\leftarrow$ barycentric coordinates of $G[j]$ in T
  $\texttt{hm}_{\text{lin}}[j] \leftarrow$ interpolate h at triangle vertices of T via bary
  Optionally: color rgb via barycentric interpolation
end for
// *Linear Interpolation outside Convex Hull*
for all j with $\texttt{face\_map}[j] = -1$ **do**
  $\texttt{hm}_{\text{lin}}[j] \leftarrow$ nearest neighbor interpolation on h via K at $G[j]$
  Optionally: color rgb via nearest-neighbor interpolation
end for
// *NN Interpolation*
for all $j = 1 \ldots N$ **do**
  $\texttt{hm}_{\text{nn}}[j] \leftarrow$ nearest neighbor interpolation on h via K at $G[j]$
end for
// *Finish Up*
Remove padding points from P, h, rgb if needed
return $\texttt{hm}_{\text{nn}}$, $\texttt{hm}_{\text{lin}}$, `face_map`

3.4. Learned Heightmaps

The linearly interpolated patches could be used directly for rendering. However, discontinuities and interpolation across slender triangles reduce the visual quality. Therefore, we employ a small neural network that produces accurate and visually pleasing interpolations.

The input of our CNN consists of two 96x96 heightmaps and two 96x96x3 RGB textures, both linearly and NN interpolated. It outputs a 64x64 heightmap and a 64x64x3 RGB texture. The 16 additional texels in each direction give the network context information, so it can produce smooth patch seams. Redundancy in the outputs could smooth the seams further but is not necessary in practice. We batch inference calls to avoid kernel overhead and context switches, increasing efficiency.

Since we need to keep every step of our method local, the model must not depend on the global context. Therefore, we normalize the given heights to patch space with $z = (z_{ms} - c_z)/r$, where z_{ms} is a height in model space (meters above sea level) and c_z is the height of the patch center. This normalization also helps avoid numerical problems in the network. Since the patch centers are created from a grid on the XY-plane, we take c_z from the linearly interpolated heightmap's center. This means that the network cannot know how far above sea level a patch is, which makes it more general but also makes color estimation more difficult.

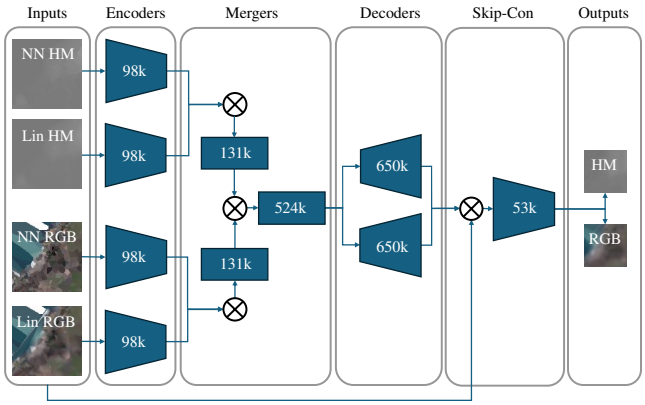


Figure 3: LidarScout *architecture. The network predicts clean heightmaps and textures from rough ones using a combination of encoders, fully-connected layers, and decoders.*

Our network architecture (see Section 3) is inspired by the U-Net [RFB15], which is also used in Neural Point-Based Graphics (NPBG) [ASK*20]. Unlike NPBG, we encode each input independently with several convolution layers, which increases the model size. However, since we work with dense inputs, we do not need gated convolutions, making our network smaller. Next, we merge the produced feature vector with separate, fully connected layers in two steps. Then, we decode the feature vector with two separate CNNs to heightmap and texture. In contrast to NPBG, we only have one skip connection that concatenates the original inputs and the decoder outputs along their channel dimension. Another CNN reduces this tensor again to the final number of channels. Finally, we take the center 64x64 region for further usage in rendering. In

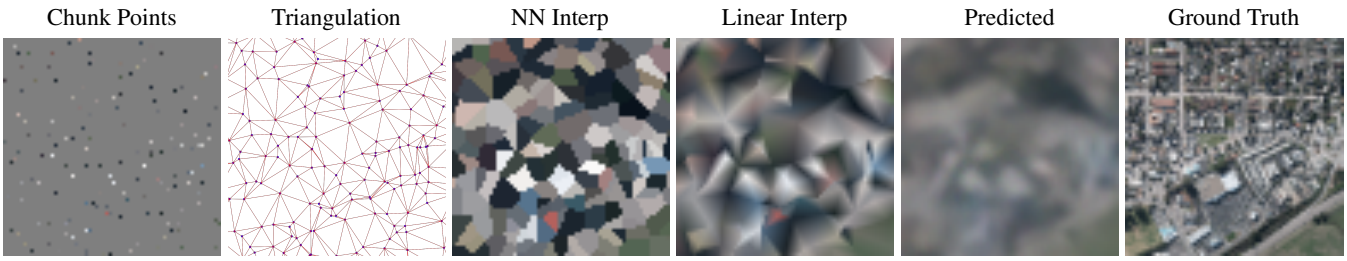


Figure 4: *Example patch with inputs, processing steps, and network prediction.*

total, the network has 2.5M learnable parameters, 500k more than NPBG. Figure 4 shows one example patch with its chunk points, triangulation, interpolations, prediction, and ground truth. The GT cannot be reached from the available sparse chunk points. Compared to linear interpolation, our network prediction is smoother, usually more accurate, and provides better transitions for our implicit level of detail. Note the red outlier at the middle bottom, which interpolation and simple smoothing preserve, but our CNN manages to ignore.

We train LIDARSCOUT only on CA13 and SWISSS3D, while we evaluate it on all the data sets described in Section 3.1 to show its generalizability. For each data set, we generate our ground-truth data for training and evaluation from these scans. We choose the patch centers randomly from the entire point cloud. 7000 of them are for training, 3000 for evaluation. For CA13 and SWISSS3D, we split the patch centers by the 70-percentile of their x coordinates into train and test sets. For each patch center, we sample a heightmap and a texture by taking the mean of all points that fall onto a texel, which also removes most of the original scanning noise. To simulate the chunk points of LAZ, we randomly select a 50 000th of all points.

We train our network with MSE loss, supervised by the ground-truth patches. We set loss elements corresponding to NaN elements in the GT (gaps in the original scans) to zero. The heightmap and texture losses are clipped to $(0.0 \dots 1.0)$ and averaged. We tried loss functions that put more weight on tile seams or gradients, but they did not make a noticeable difference. L1 loss and SSIM produce very similar, smoothed results, and LPIPS loss creates stripe artifacts. The bad results with LPIPS are likely due to a low number of top-down landscape images in their data sets and our images having a very low resolution. Our optimizer is AdamW (lr = 0.0001, betas = (0.9, 0.999), eps = 1e-5, weight decay = 1e-2, amsgrad = False). With a step scheduler (gamma = 0.1, steps at 25 and 50 epochs), we train for 75 epochs. The training on CA13 and SWISSS3D takes about 25 minutes. Training is done in Python using PyTorch, and the inference in C++/CUDA using LibTorch.

3.5. Full-Resolution Tiles

Upon navigating close to the surface, we additionally load and display full-resolution point data from tiles with a sufficiently large screen-space bounding box. In our test data sets, tiles typically hold about 1 to 50 million points. Close-range viewpoints such as in Figure 6 may require loading about 50-300 million points, but nowadays, compute-based brute-force software rasterizers are capable of rendering up to two billion points in real time [SKW22]. Tiles that are no longer in focus are unloaded to free memory for other tiles. However, we update the heightmap textures to preserve some information for medium zoom levels.

3.6. Rendering

We need to render chunk points for any patch whose heightmap is not yet ready, followed by rendering textured heightmaps, and eventually by rendering the full-resolution point cloud data upon zooming in close to a tile. Rendering of points and heightmaps was implemented in a CUDA-based software rasterizer.

For points, we use the approach by Schütz et al. [SKW22]: We launch one thread-block comprising 256 threads for each chunk of 50k points. The threads iterate through all points, projecting them to screen and encoding their depth and color value into a 64 bit integer. We then use a 64 bit atomicMin to evaluate the point with the smallest depth value for each pixel. Afterward, a screen-space resolve pass extracts the color value from the least significant bits of each pixel's 64 bit depth and color value, and stores the result in an OpenGL texture for display.

Heightmaps are rendered with a custom CUDA-based triangle-rasterizer: We invoke a cooperative kernel with 64 threads per block. Each block processes 32 triangles at a time, projects them to screen, and computes the screen-space bounding box. For any triangle that covers less than 1024 pixels, a single thread of the group iterates over the pixels, evaluates the barycentric coordinates. If they indicate that the pixel is inside the triangle, it draws the fragment using the same 64 bit atomic-min logic as the point rasterizer. If a triangle is larger than 1024 pixels, it is added to a queue. After the block finishes rendering the smaller triangles in one thread per triangle, it continues to render the large triangles utilizing all 64 threads for each triangle, one after the other. The thread blocks continue to loop until all triangles of all heightmaps are rendered.

4. Results

We evaluate two use cases for our method and compare with several baselines: exploring large point clouds quickly and estimating the surface accurately from local subsamples.

4.1. Exploring Large Point Clouds

In order to display massive data sets that do not fit in GPU memory, state-of-the-art solutions require creating LOD

Table 2: *Time to construct an LOD structure in Potree vs. time to completely finish each stage in* LidarScout. *After loading all tiles' metadata, loading the chunk points and the stages of heightmap construction operate progressively, so users can already navigate the data set before they are finished.*

Data set	Potree	LidarScout		
		Tiles	Chunk Points	Heightmaps
CA13	28m44s	0.02s	1.9s	23s
Gisborne	19h56m	0.10s	12.8s	159s
Gisborne+Add	-	0.19s	53.0s	701s

structures in a preprocessing step. Potential solutions include Entwine [Ent21], Potree [SOW20], MassivePotreeConverter [MRVv*15], and Lidarserv [BDSF22]. We will study the performance of LidarScout in comparison to Potree, the fastest of the related approaches. For rendering, we use this test system: RTX 4090; AMD Ryzen 9 7950X 16-Core; Crucial T700 4TB PCIe Gen5.

4.1.1. Case Study: CA13 (17.7 billion points).

As shown in Table 2, Potree takes 28m44s until LODs are constructed and the data set can be explored. LidarScout takes 0.02s to load the metadata of 2336 tiles and another 1.9s until all chunk points are loaded. Heightmap construction takes 23s to finish, but construction prioritizes the current viewpoint so users do not need to wait to see meaningful results.

4.1.2. Case Study: Gisborne (95 billion points).

LidarScout takes 12.8s until a subsample of 1.9 million chunk points has been loaded in order to display an overview of the entire data set. Heightmaps are then constructed based on the user's current viewpoint. In comparison, users would traditionally have to wait 19h56m to construct an LOD structure before being able to explore the data set. The LOD structure that was constructed by Potree required 1.7TB of additional disk space.

4.1.3. Case Study: Gisborne+Addendum (262 billion points)

We were not able to evaluate Potree's performance due to lack of additional disk space for the constructed LOD data. Extrapolating from Gisborne without Addendum, Potree would presumably require 2 days and 6 hours to finish LOD construction.

Although LidarScout takes 53s to load the chunk points of the entire overview, users can already start exploring the data set as soon as the tile metadata is loaded. Chunkpoints are loaded progressively so users may navigate to already prepared regions, and a list of files/tiles allows users to zoom towards specific tiles, which are then loaded in full resolution even before all chunk points are loaded or heightmaps are generated.

4.2. Surface Reconstruction

Few methods for surface reconstruction are applicable to our use case. Recent and popular global reconstruction methods, such as BallMerge [POEM24], Ball Pivot [BMR*99], Screened Poisson Surface Reconstruction [KH13], and PPSurf [EFPH*24] perform poorly with our chunk points. Please see the supplementary material for images. Figure 5 shows a qualitative comparison of the most relevant local reconstruction methods. Linear interpolation has hard discontinuities and noisy colors (see also Fig. 4). Cubic interpolation suffers from overshooting, causing very bright and dark spots, and sometimes peaks of several hundred meters. High-Quality Splatting [BHZK05] either creates blobby structures, smoothed stairs, or gaps due to the fixed-size kernel. NPBG [ASK*20] with our inputs is close to LidarScout in quality but distorts colors sometimes.

Table 3: *Root Mean Square Error (lower is better) of predicted heightmaps. Best results are in bold, second best underlined.*

Data set	Cubic	HQSplat	Linear	LidarScout
CA13	4.36±0.72	5.47±0.69	4.17±0.62	**3.81±0.55**
SwissS3D	11.60±2.14	12.58±2.16	9.26±1.44	**8.56±1.39**
Bund_Bora	1.40±0.82	1.97±0.31	**1.00±0.60**	1.16±0.17
ID15_Bunds	**0.65±0.19**	1.73±0.26	0.89±0.23	1.22±0.21
Gisborne_A	11.38±4.71	7.43±0.93	6.97±1.12	**6.55±0.75**
Gisborne_B	11.32±1.77	6.83±0.41	7.16±0.35	**6.26±0.34**
Gisborne_C	7.41±1.95	6.40±0.48	5.51±0.46	**5.09±0.33**
Mean	6.87±1.76	6.06±0.75	4.99±0.69	**4.66±0.53**

Table 4: *Peak Signal-To-Noise Ratio (higher is better) of predicted textures. Best results are in bold, second best underlined.*

Data set	Cubic	HQSplat	Linear	LidarScout
CA13	66.08±1.26	69.83±1.14	69.38±1.14	**70.76±1.09**
Bund_Bora	69.57±1.11	**78.42±1.19**	73.29±0.97	77.45±0.86
ID15_Bunds	67.60±3.44	**78.79±0.80**	75.02±0.53	77.89±0.87
Gisborne_A	66.98±1.15	71.30±1.03	70.80±0.95	**72.24±0.77**
Gisborne_B	64.54±0.52	70.86±0.63	68.67±0.54	**71.46±0.48**
Gisborne_C	65.84±0.67	70.47±0.68	69.67±0.57	**71.22±0.52**
Mean	66.77±1.36	73.28±0.91	71.14±0.79	**73.50±0.77**

Quantitative Comparison Tables 3 and 4 show the performance of LidarScout on all data sets. We report the average over all patches in the test sets. We use Root Mean Squared Error (RMSE) in meters to compare heightmap quality and Peak Signal-to-Noise Ratio (PSNR) in dB for textures.

Our most important baseline is also used as input to our network: linearly interpolated heightmaps and textures. We perform Delaunay triangulation and linear interpolation with barycentric coordinates on the local chunk points, as described in Section 3.3. For such a simple baseline, it is surprisingly good. It is also an approximation for Rapidlasso's Las2Dem [rG15], which takes care of multiple returns per texels in addition. However, this refinement does not make a noticeable difference with our sparse chunk points. We also compare with a cubic Clough-Tocher interpolation implemented in SciPy [SA11], which is accurate in many cases but occasionally overshoots. Lastly, we compare to High-Quality Splatting [BHZK05] by treating each chunk point as a large, fixed-size splat, and using a Gaussian blending function to obtain a

Table 5: *Ablation study main results. Best results are in bold, second best underlined. Note that the **Extra Data** variant was partially trained on the test data, so its metrics are positively biased. Please see the supplementary material for the full tables.*

Variant	NPBG	DCTNet	Raster	HM Only	NN only	Lin only	Extra Data	LIDARSCOUT
Heights RMSE [m] ↓	4.87±0.64	4.82±0.56	14.91±3.30	4.64±0.55	4.84±0.55	4.71±0.55	**4.56±0.52**	4.66±0.53
Colors PSNR [DB] ↑	73.30±0.77	73.20±0.77	60.98±1.07	NA	73.50±0.77	73.53±0.76	**74.16±0.78**	73.50±0.77

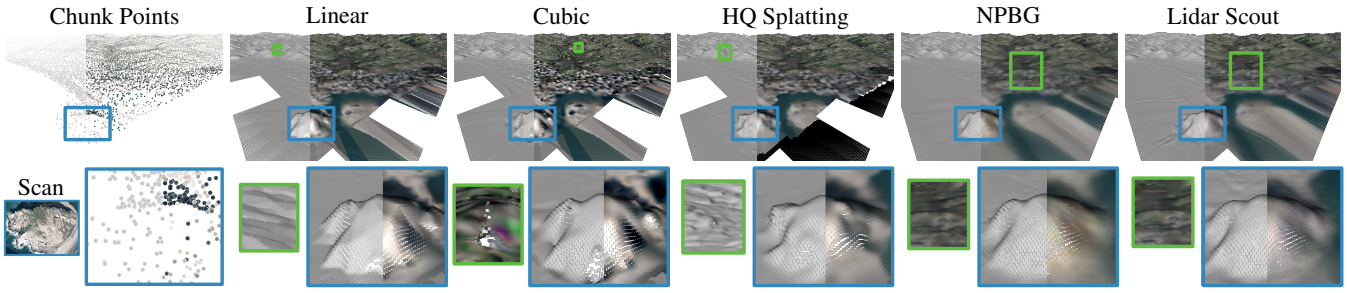


Figure 5: *Qualitative comparison of the Morro Rock region in* CA13.

smooth transition of heights and colors between overlapping splats. LIDARSCOUT performs significantly better than the baselines except for the much denser BUND_BORA and ID15_BUNDS data sets.

Computation Time and Memory Consumption Reconstruction was evaluated on an NVIDIA RTX 3090 and an AMD Ryzen 7 3700X 8-Core. The reconstruction of one patch in our C++/CUDA framework takes around 20 ms. Single-threaded triangulation and sampling take 16 ms, inference and buffer copies 4 ms. Batched inference requires copying data to make the input heightmaps and textures contiguous in memory, which means a small overhead. In any case, the timings (see supplementary material) show that batching always pays off.

Ablation Table 5 shows an ablation that empirically validates our design choices, comparing different network architectures and inputs. Other architectures like **NPBG** [ASK*20] and **DCTNet** [ZZX*22] perform worse than ours. **Raster**izing points and filling unknown pixels with zeros does not work well as input for our image-based network. This means that dense inputs are necessary for viable quality. The model draws information from both nearest-neighbor (**NN**) and **lin**ear interpolation inputs, especially for the heights. **Lin**ear-only generalizes better across point densities, while **NN**-only is better with similar densities. Combining them is a step towards the best of both. Omitting RGB inputs (**HM only**) has a negligible impact on heightmap quality. Adding **Extra Data** (ID15_BUNDS and GISBORNE in addition to CA13 and SWISSS3D) improves the quality significantly. Note that only BUND_BORA is completely unseen, making a fair comparison difficult for this variant. Please see the supplementary material for detailed statistics.

4.3. Discussion and Limitations

Lidarserv [BDSF22] and SimLOD [SHW24] are prior work that follow similar goals: Exploring large data sets without the need to preprocess and wait. Lidarserv specifically aims to enable displaying arbitrarily large point clouds directly during capture, and is able to construct out-of-core LOD structures at rates of up to around 1.8 million points per second. SimLOD aims to visualize large point cloud files quickly and is capable of loading industry standard LAS files at rates of up to 300 million points per second, or compressed LAZ files at up to 30 million points per second. Both display points immediately as they are streamed, without the need to wait until processing is finished. A major difference in our approach is that we aim to display massive data sets in their entirety in a matter of seconds and prioritize more detailed reconstructions towards the user's viewpoint, while the prior works operate on local regions in undefined order without priorization. SimLOD is further limited to data sets that fit in memory, i.e., about 800 million points per 24GB of memory. Our approach, on the other hand, rapidly displays arbitrarily large data sets but lacks level-of-detail structures that would further improve rendering performance, especially for previously visited regions. In the future, we would like to integrate and expand SimLOD's incremental LOD construction in order to build an out-of-core system that is capable of rendering arbitrarily large point clouds with instant overviews of the entire data set, and priorization towards the current viewpoint.

Although we trained LIDARSCOUT only on CA13 and SWISSS3D, it generalizes well to other regions of the world. The model has mostly seen colors typical for deserts, small cities, and beaches from the Morro Bay region in California, USA. Nonetheless, it can still reconstruct the colors of the lush vegetation of Gisborne, New Zealand, as shown in Table 4 and Figure 6. Furthermore, it generalizes well across scan patterns that may affect the sparse subsample. It was trained on the line-wise scanning patterns of the CA13 aerial LIDAR, but it has no issues with the circular scanning patterns of GISBORNE.

Linear and cubic interpolation are competitive on the much denser photogrammetry point clouds of ID15_BUNDS and BUND_BORA for predicting heights. However, only HQ-Splatting

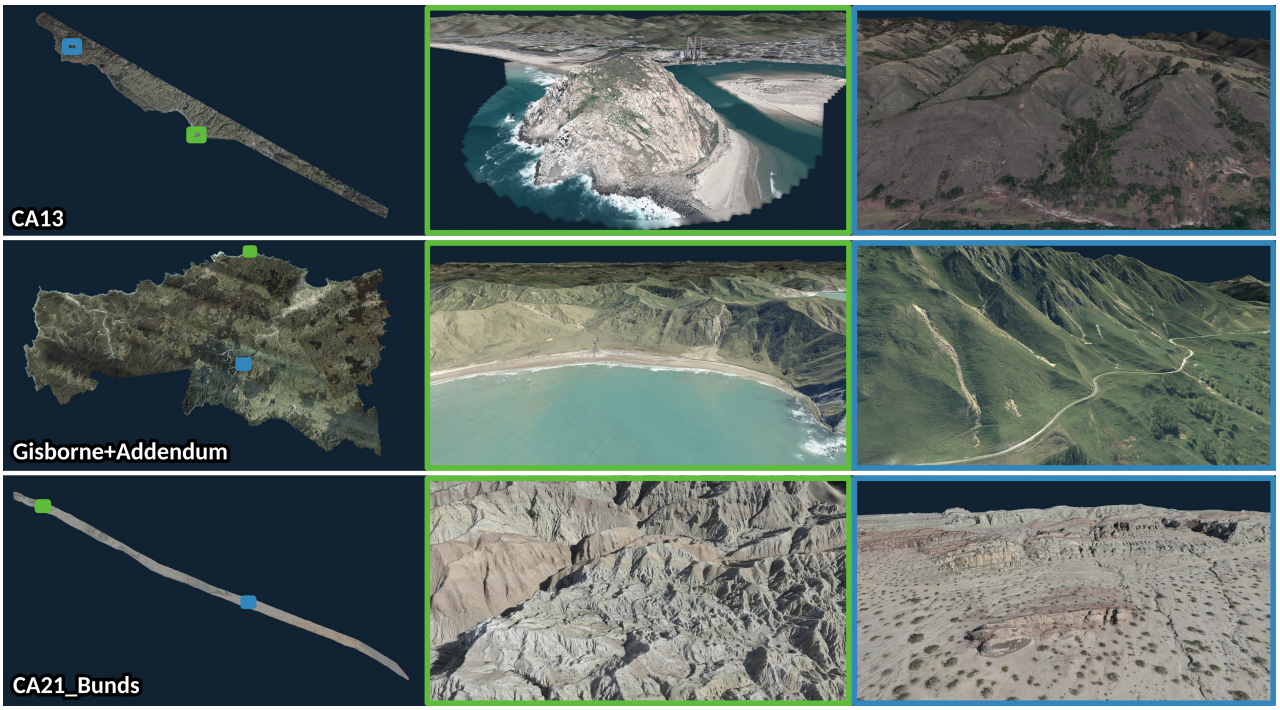


Figure 6: *Screenshots of CA13 (17.7B points, 90GB), Gisborne (262B points, 2.4TB) and CA21_Bunds (8.4B points, 96GB) made with* LIDARSCOUT.

can compete with our method when estimating colors. This indicates that our method does not generalize too well from point densities of around 20 $points/m^2$ in CA13 to almost 600 $points/m^2$ in BUND_BORA and fails to use the available information. Adapting the patch size solves this issue.

5. Conclusion

LIDARSCOUT is the first point cloud viewer that allows exploring terabytes of compressed LIDAR scans within seconds without any pre-processing. We present fast 2.5D surface reconstruction from sparse, local subsamples with minimal overhead. Our neural network outperforms the current industry standard of triangulation and linear interpolation. In the future, the local chunk points or heightmaps could be streamed from a server for only the required regions, which would drastically reduce data storage on the user side and data transfer costs for everyone.

Our approach is potentially applicable to any data that allows at least partial random access with some learnable patterns, such as huge photographs, volumetric data from CT or MRT scans, weather forecasts, and astronomy simulations. The neural network can be adapted for annotation, segmentation, and classification tasks. With the latter, for example, it could take the number of returns and other extra point properties to detect vegetation. Extending LIDARSCOUT to 3D for large urban and indoor scans should be possible by combining our persistent heightmaps with screen-space approaches like ADOP [RFS22] and TRIPS [FRFS24].

6. Acknowledgements

The authors wish to thank following data set providers: *Bunds at el.* and *Open Topography* for the Bund_Bora [BDG*19] and ID15_Bunds [BDG*20] data sets; *PG&E* and *Open Topography* for CA13 [Pac13]; The *Ministry of Business, Innovation and Employment* and *Toitū Te Whenua Land Information New Zealand* and *Open Topography* for Gisborne [MoBE24]; The *São Paulo City Hall (PMSP)* and *Open Topography* for São Paulo [(PM17]; The *Bundesamt für Landestopografie swisstopo* for swissSURFACE3D [Swi20].

We thank Paul Guerrero, Pedro Hermosilla, and Adam Celarek for their valuable inputs. Further, we thank Stefan Ohrhallinger for running reconstructions with BallMerge [POEM24].

This research has been funded by WWTF project *ICT22-055 - Instant Visualization and Interaction for Large Point Clouds*.

References

[ASK*20] ALIEV K.-A., SEVASTOPOLSKY A., KOLOS M., ULYANOV D., LEMPITSKY V.: Neural point-based graphics. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXII 16* (2020), Springer, pp. 696–712. 3, 5, 7, 8

[BDG*19] BUNDS M., DUROSS C., GOLD R., REITMAN N., TOKE N., BRIGGS R., PERSONIUS S., JOHNSON K., LAJOIE L., UNGERMAN B., MATHESON E., ANDREINI J., LARSEN K.: Lost river fault zone near borah peak, idaho, 2019. Distributed by OpenTopography, Accessed 2025-01-17. doi:https://doi.org/10.5069/G9222RWR. 9

[BDG*20] BUNDS M., DUROSS C., GOLD R., REITMAN N., TOKE

N., BRIGGS R., UNGERMAN B., , MATHESON E.: Lost river fault at doublespring pass rd, idaho 2015, 2020. Distributed by OpenTopography, Accessed: 2025-01-17. doi:https://doi.org/10.5069/G9TH8JWV. 9

[BDSF22] BORMANN P., DORRA T., STAHL B., FELLNER D. W.: Real-time Indexing of Point Cloud Data During LiDAR Capture. In *Computer Graphics and Visual Computing (CGVC)* (2022), Vangorp P., Turner M. J., (Eds.), The Eurographics Association. doi:10.2312/cgvc.20221173. 3, 7, 8

[BHZK05] BOTSCH M., HORNUNG A., ZWICKER M., KOBBELT L.: High-quality surface splatting on today's gpus. In *Proceedings Eurographics/IEEE VGTC Symposium Point-Based Graphics, 2005.* (2005), IEEE, pp. 17–141. 2, 7

[BMR*99] BERNARDINI F., MITTLEMAN J., RUSHMEIER H., SILVA C., TAUBIN G.: The ball-pivoting algorithm for surface reconstruction. *IEEE transactions on visualization and computer graphics 5*, 4 (1999), 349–359. 2, 7

[Dre07] DREPPER U.: What every programmer should know about memory. *Red Hat, Inc 11*, 2007 (2007), 2007. 3

[DVS03] DACHSBACHER C., VOGELGSANG C., STAMMINGER M.: Sequential point trees. *ACM Trans. Graph. 22*, 3 (2003), 657–662. 3

[EFPH*24] ERLER P., FUENTES-PEREZ L., HERMOSILLA P., GUERRERO P., PAJAROLA R., WIMMER M.: Ppsurf: Combining patches and point convolutions for detailed surface reconstruction. In *Computer Graphics Forum* (2024), vol. 43, Wiley Online Library, p. e15000. 2, 7

[Ent21] Entwine, 2021. https://entwine.io/, Accessed 2021.04.13. 7

[FRFS24] FRANKE L., RÜCKERT D., FINK L., STAMMINGER M.: Trips: Trilinear point splatting for real-time radiance field rendering. In *Computer Graphics Forum* (2024), Wiley Online Library, p. e15012. 3, 9

[GKLR13] GÜNTHER C., KANZOK T., LINSEN L., ROSENTHAL P.: A gpgpu-based pipeline for accelerated rendering of point clouds. *J. WSCG 21* (2013), 153–161. 2

[Har13] HARRIS M.: How to access global memory efficiently in cuda c/c++ kernels. NVIDIA Technical Blog, 2013. Accessed 2025.01.17. URL: https://developer.nvidia.com/blog/how-access-global-memory-efficiently-cuda-c-kernels/. 3

[HFF*23] HARRER M., FRANKE L., FINK L., STAMMINGER M., WEYRICH T.: Inovis: Instant novel-view synthesis. In *SIGGRAPH Asia 2023 Conference Papers* (2023), pp. 1–12. 3

[HFK*24] HAHLBOHM F., FRANKE L., KAPPEL M., CASTILLO S., STAMMINGER M., MAGNOR M.: Inpc: Implicit neural point clouds for radiance field rendering. *arXiv preprint arXiv:2403.16862* (2024). 3

[Ise13] ISENBURG M.: Laszip: lossless compression of lidar data. *Photogrammetric engineering and remote sensing 79*, 2 (2013), 209–217. 2, 3

[KB04] KOBBELT L., BOTSCH M.: A survey of point-based techniques in computer graphics. *Computers & Graphics 28*, 6 (2004), 801–814. 2

[KH13] KAZHDAN M., HOPPE H.: Screened poisson surface reconstruction. *ACM Transactions on Graphics (ToG) 32*, 3 (2013), 1–13. 2, 7

[KKLD23] KERBL B., KOPANAS G., LEIMKÜHLER T., DRETTAKIS G.: 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics 42*, 4 (July 2023). URL: https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/. 3

[KPLD21] KOPANAS G., PHILIP J., LEIMKÜHLER T., DRETTAKIS G.: Point-based neural rendering with per-view optimization. In *Computer Graphics Forum* (2021), vol. 40, Wiley Online Library, pp. 29–43. 3

[LS81] LANCASTER P., SALKAUSKAS K.: Surfaces generated by moving least squares methods. *Mathematics of computation 37*, 155 (1981), 141–158. 2

[LW85] LEVOY M., WHITTED T.: The use of points as a display primitive, 1985. URL: http://www.graphics.stanford.edu/papers/points/. 2

[MoBE24] MINISTRY OF BUSINESS I., EMPLOYMENT T. T. W. L. I. N. Z. L.: Gisborne, new zealand 2023, 2024. Released under Creative Commons CC BY 4.0 by NIWA, Collected by Landpro, distributed by OpenTopography and LINZ, Accessed: 2025-01-17. doi:https://doi.org/10.5069/G9MK6B34. 9

[MRC*22] MASOUMIAN A., RASHWAN H. A., CRISTIANO J., ASIF M. S., PUIG D.: Monocular depth estimation using deep learning: A review. *Sensors 22*, 14 (2022), 5353. 2

[MRVv*15] MARTINEZ-RUBI O., VERHOEVEN S., VAN MEERSBERGEN M., SCHUTZ M., VAN OOSTEROM P., GONCALVES R., TIJSSEN T.: Taming the beast: Free and open-source massive point cloud web visualization. In *Capturing reality: The 3rd, laser scanning and LIDAR technologies forum* (2015), MacDonald A., (Ed.), s.n., pp. 23–25. geen ISBN; Capturing Reality 2015, Salzburg, Austria ; Conference date: 23-11-2015 Through 25-11-2015. 2, 3, 7

[Ned23] NEDERLAND A. H.: Eerste deel van ahn 5 is beschikbaar! https://www.ahn.nl/eerste-deel-van-ahn-5-is-beschikbaar, 2023. [Accessed 17-01-2025]. 2, 3

[NKH*21] NGUYEN P., KARNEWAR A., HUYNH L., RAHTU E., MATAS J., HEIKKILA J.: Rgbd-net: Predicting color and depth images for novel views synthesis. In *2021 International Conference on 3D Vision (3DV)* (2021), IEEE, pp. 1095–1105. 3

[Pac13] PACIFIC GAS & ELECTRIC COMPANY: Pg&e diablo canyon power plant (dcpp): San simeon and cambria faults, ca, airborne lidar survey, 2013. Distributed by OpenTopography. doi:https://doi.org/10.5069/G9CN71V5. 3, 9

[(PM17] (PMSP) S. P. C. H.: Sao paulo, brazil lidar survey 2017, 2017. Distributed by OpenTopography and LINZ, Accessed: 2025-06-07. doi:https://doi.org/10.5069/G9NV9GD1. 9

[POEM24] PARAKKAT A. D., OHRHALLINGER S., EISEMANN E., MEMARI P.: Ballmerge: High-quality fast surface reconstruction via voronoi balls. In *Computer Graphics Forum* (2024), Wiley Online Library, p. e15019. 2, 7, 9

[PZVBG00] PFISTER H., ZWICKER M., VAN BAAR J., GROSS M.: Surfels: Surface elements as rendering primitives. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (2000), pp. 335–342. 2

[RALB22] RAKHIMOV R., ARDELEAN A.-T., LEMPITSKY V., BURNAEV E.: Npbg++: Accelerating neural point-based graphics. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2022), pp. 15969–15979. 3

[RFB15] RONNEBERGER O., FISCHER P., BROX T.: U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18* (2015), Springer, pp. 234–241. 3, 5

[RFS22] RÜCKERT D., FRANKE L., STAMMINGER M.: Adop: Approximate differentiable one-pixel point rendering. *ACM Transactions on Graphics (ToG) 41*, 4 (2022), 1–14. 3, 9

[rG15] RAPIDLASSO GMBH: Generating Spike-Free Digital Surface Models from LiDAR - rapidlasso GmbH — rapidlasso.de. https://rapidlasso.de/generating-spike-free-digital-surface-models-from-lidar/, 2015. [Accessed 09-01-2025]. 2, 7

[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* (USA, 2000), SIGGRAPH '00, ACM Press/Addison-Wesley Publishing Co., p. 343–352. 3

[RSL*24] RAJAPAKSHA U., SOHEL F., LAGA H., DIEPEVEEN D., BENNAMOUN M.: Deep learning-based depth estimation methods from

monocular image and videos: A comprehensive survey. *ACM computing surveys 56*, 12 (2024), 1–51. 2

[SA11] SCIPY AUTHORS: CloughTocher2DInterpolator x2014; SciPy v1.15.1 Manual — docs.scipy.org. https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.CloughTocher2DInterpolator.html, 2011. [Accessed 17-01-2025]. 7

[SHW24] SCHÜTZ M., HERZBERGER L., WIMMER M.: Simlod: Simultaneous lod generation and rendering for point clouds. *Proceedings of the ACM on Computer Graphics and Interactive Techniques 7*, 1 (2024), 1–20. 3, 8

[SKW22] SCHÜTZ M., KERBL B., WIMMER M.: Software rasterization of 2 billion points in real time. *Proceedings of the ACM on Computer Graphics and Interactive Techniques 5*, 3 (July 2022), 1–17. URL: https://www.cg.tuwien.ac.at/research/publications/2022/SCHUETZ-2022-PCC/, doi:10.1145/3543863. 2, 6

[SMOW20] SCHÜTZ M., MANDLBURGER G., OTEPKA J., WIMMER M.: Progressive real-time rendering of one billion points without hierarchical acceleration structures. *Computer Graphics Forum 39*, 2 (2020), 51–64. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13911, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13911, doi:https://doi.org/10.1111/cgf.13911. 2

[SOW20] SCHÜTZ M., OHRHALLINGER S., WIMMER M.: Fast out-of-core octree generation for massive point clouds. *Computer Graphics Forum 39*, 7 (Nov. 2020), 1–13. URL: https://www.cg.tuwien.ac.at/research/publications/2020/SCHUETZ-2020-MPC/, doi:10.1111/cgf.14134. 3, 7

[Sur18] SURVEY) U. U. G.: Changes in usgs lidar data distribution announced. https://www.usgs.gov/news/technical-announcement/3d-elevation-program-distributing-lidar-data-laz-format, 2018. [Accessed 17-01-2025]. 2, 3

[SW11] SCHEIBLAUER C., WIMMER M.: Out-of-core selection and editing of huge point clouds. *Computers & Graphics 35*, 2 (2011), 342–351. 3

[Swi20] SWISSTOPO: swisssurface3d raster: Das hoch aufgelöste oberflächenmodell der schweiz. https://backend.swisstopo.admin.ch/fileservice/sdweb-docs-prod-swisstopoch-files/files/2023/11/14/24d12399-72b8-4000-8544-235023c4369f.pdf, 2020. [Accessed 09-01-2025]. 2, 9

[TFT*20] TEWARI A., FRIED O., THIES J., SITZMANN V., LOMBARDI S., SUNKAVALLI K., MARTIN-BRUALLA R., SIMON T., SARAGIH J., NIESSNER M., ET AL.: State of the art on neural rendering. In *Computer Graphics Forum* (2020), vol. 39, Wiley Online Library, pp. 701–727. 3

[WBB*08] WAND M., BERNER A., BOKELOH M., JENKE P., FLECK A., HOFFMANN M., MAIER B., STANEKER D., SCHILLING A., SEIDEL H.-P.: Processing and interactive editing of huge point clouds from 3d scanners. *Computers & Graphics 32*, 2 (2008), 204 – 220. 3

[WS06] WIMMER M., SCHEIBLAUER C.: Instant Points: Fast Rendering of Unprocessed Point Clouds. In *Symposium on Point-Based Graphics* (2006), Botsch M., Chen B., Pauly M., Zwicker M., (Eds.), The Eurographics Association. 3

[WWG*21] WANG Q., WANG Z., GENOVA K., SRINIVASAN P. P., ZHOU H., BARRON J. T., MARTIN-BRUALLA R., SNAVELY N., FUNKHOUSER T.: Ibrnet: Learning multi-view image-based rendering. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (2021), pp. 4690–4699. 3

[YGL*23] YOU M., GUO M., LYU X., LIU H., HOU J.: Learning a locally unified 3d point cloud for view synthesis. *IEEE Transactions on Image Processing* (2023). 3

[ZPVBG02] ZWICKER M., PFISTER H., VAN BAAR J., GROSS M.: Ewa splatting. *IEEE Transactions on Visualization and Computer Graphics 8*, 3 (2002), 223–238. 2

[ZZX*22] ZHAO Z., ZHANG J., XU S., LIN Z., PFISTER H.: Discrete cosine transform network for guided depth map super-resolution. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (2022), pp. 5697–5707. 8