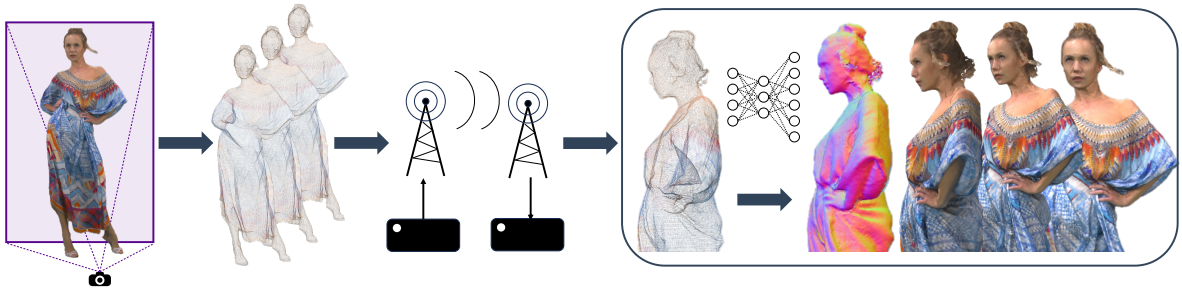
2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW) | 979-8-3503-6547-4/24/$31.00 ©2024 IEEE | DOI: 10.1109/CVPRW63382.2024.00584

Low Latency Point Cloud Rendering with Learned Splatting

Yueyu Hu[1] Ran Gong[2] Qi Sun[1] Yao Wang[1]
[1]Tandon School of Engineering, New York University [2]Tsinghua University
{yyhu, qisun, yaowang}@nyu.edu rangong41@gmail.com



Abstract

Point cloud is a critical 3D representation with many emerging applications. Because of the point sparsity and irregularity, high-quality rendering of point clouds is challenging and often requires complex computations to recover the continuous surface representation. On the other hand, to avoid visual discomfort, the motion-to-photon latency has to be very short, under 10 ms. Existing rendering solutions lack in either quality or speed. To tackle these challenges, we present a framework that unlocks interactive, free-viewing and high-fidelity point cloud rendering. We train a generic neural network to estimate 3D elliptical Gaussians from arbitrary point clouds and use differentiable surface splatting to render smooth texture and surface normal for arbitrary views. Our approach does not require per-scene optimization, and enable real-time rendering of dynamic point cloud. Experimental results demonstrate the proposed solution enjoys superior visual quality and speed, as well as generalizability to different scene content and robustness to compression artifacts.

1. Introduction

Point cloud is a versatile 3D representation that can be directly acquired by various sensors such as LiDAR, multi-view, or RGB-D cameras without heavy processing, thus enabling real-time capturing and streaming. It is also more flexible than polygonal mesh when representing non-manifold geometry. These benefits have led to growing deployment of point cloud for applications such as culture heritage, autonomous driving, immersive visual communications, and VR/AR. Its importance is also evidenced by the MPEG point cloud compression standardization activities since 2014 [6, 19]. However, rendering a point cloud given a user's viewpoint is uniquely challenging: Unlike meshes, the point representation does not provide explicit surface information, making the rendering suffer from point sparsity, geometric irregularity, and sensor noise [10]. On the other hand, to avoid visual discomfort, experimental studies have demonstrated that the motion-to-photon (MTP) latency should not exceed 10 ms [12, 17]. These challenges and demands have become the main roadblocks of the aforementioned point-cloud-enabled applications, especially for scenarios where a dynamic point cloud needs to be captured, streamed, and rendered in real time.

Various solutions have been presented to render point clouds as images from arbitrary viewpoints, including point splatting [4, 20, 35] and ray tracing [5]. However, existing methods either require heavy computation (e.g., [5]), far exceeding the MTP threshold (see Table 1), or generate blurred images that miss details [14] or have holes [20] (see Figure 5). An alternative solution is cloud-based rendering using a powerful server in the cloud [13, 26]. This, however, requires the total latency including the round-trip transmission between the user and the server, rendering at the server, and (de)compression of the rendered view, to be completed within the MTP threshold, necessitating ex-

2160-7516/24/$31.00 ©2024 IEEE
DOI 10.1109/CVPRW63382.2024.00584

tremely low-latency communication links. Even when the server is positioned close to the network edge, the typical delay in today's access networks is still prohibitively large to meet the MTP requirement.

We develop a point cloud rendering framework that enables high visual quality, six degrees of freedom (6-DoF) viewing, and satisfies the MTP requirement with consumer-grade computers at the user end. Our solution does not require per-scene neural network training nor heavy surface reconstruction computation. Our method leverages the 3D surface representation using 3D elliptical Gaussians and the corresponding differentiable 3D splatting renderer [15]. We train a light-weight 3D sparse convolutional neural network called Point-to-Ellipsoid (P2ENet) to transfer each point in the colored point cloud into an ellipsoid. The ellipsoids are then splatted to render a frame at the most current viewpoint. The differentiable renderer enables us to train the P2ENet to optimize the rendering quality. The 3D Gaussian representation enables high-quality rendering. The P2ENet also derives a normal vector with each Gaussian, enables the generation of a normal map beyond a rendered image, and, therefore, unlocks practical applications such as relighting and meshing.

We evaluate our method with both high- and low-density point clouds, as well as a variety of scenes of both dynamic human in actions and outdoor objects. Experimental results show that our method can render high-quality and hole-less images faster than 100 FPS after an initial delay of less than 30 ms. It is also robust to point cloud capturing and compression noise. Given the affordability of RGB-D capturing devices, we hope the research enables the high quality real-time streaming and rendering of live captured 3D scenes from remote studios, and interactive VR/AR applications among multiple remote participants. To this aim, we will release our code to the public upon acceptance. In summary, we make the following main contributions:

- a generalizable neural network P2ENet that transforms point clouds to 3D Gaussian representations without per-scene training;
- an end-to-end framework for low-latency and high-fidelity point cloud rendering;
- the representation enabled high-quality normal maps for practical applications such as meshing and relighting;

2. Related Work

2.1. Point Cloud Rasterization

Rasterization is a common and efficient method for rendering. However, since points do not naturally have spatial dimensions, they are converted to oriented disk [20] or 3D Gaussians [35], which are then rasterized to pixels. Generally, with screen-space alpha blending by the projected Gaussian kernels, the surface splatting methods [3, 32, 36] render smoother surface geometries and visually pleasing texture when the points are dense (i.e. the surface is adequately sampled). Although the initially proposed 3D Gaussian representation in [35] is general and can represent an arbitrary tilted ellipsoid through a non-diagonal covariance matrix, earlier methods considered only isotropic Gaussians (i.e. a diagonal covariance matrix with equal diagonal elements). The variance of the isotropc Gaussian kernel was either set to a global constant and determined by global point density, or spatially varying depending on the local covariance of point coordinates. To ensure that points in the front surface are correctly occluding points in the back surface, a pixel-level visibility check is usually required [32]. The later work in [7] proposed an approach to estimate elliptical parameters from a point cloud and further demonstrated that this kind of method can be used for real-time rendering of point clouds. However, due to the diversity in point clouds and existence of capturing and quantization noise, it is hard to determine the optimal variance or elliptical parameters, leading to either holes in the rendered image or blurry texture. This motivates our idea of using a neural network to estimate the optimal elliptical parameters and the displacement for each point by analyzing the local and global structure of the colored point cloud.

2.2. Learning-based Renderer

Neural networks are capable of learning complex mapping from inputs to outputs. With a differentiable renderer [2, 18, 29], neural networks can be end-to-end trained with supervision on the rendered images. Recent works leveraging a differentiable version of the 3D Gaussian splatting renderer have demonstrated capability of inverse rendering, geometry editing [29], and novel-view synthesis from multi-view images [2, 15, 23]. Pursuing a different direction, [5] presents a point cloud renderer based on a differentiable ray tracer. By introducing a transformer to estimate the intersection between a camera ray and a local set of points within a cylinder of the ray, the method achieves state-of-the-art rendering quality and enables relighting by simultaneously predicting the surface normal at the intersection point. However, the ray-tracing process, which involves inference of a complex transformer when shading every pixel, is too slow for real-time rendering.

Our work is inspired by 3D Gaussian Splatting (3DGS) [15] which demonstrates that 3D Gaussians can be used to represent smooth surface through per-scene optimization. However, our research is fundamentally orthogonal to the application scenarios of 3DGS and Neural Point-Based Graphics (NBPG) [2], which focus on novel view synthesis from multiview images and rely on these images as inputs to generate the point cloud. In contrast, our work is concerned with applications when only pre-captured point cloud data are available and need to be streamed and ren-

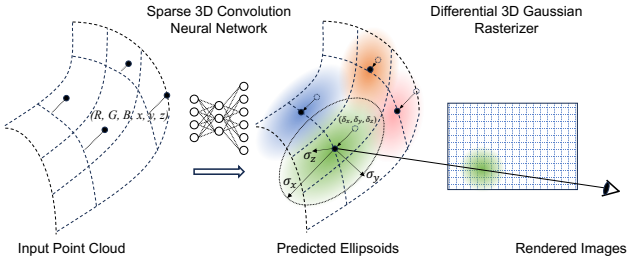


Figure 1. Rendering by estimating elliptical parameters from point cloud using sparse 3D convolutional neural network.

dered in real time. Furthermore, 3DGS requires per scene optimization for each individual frame. Therefore, 3DGS cannot support our target application. In comparison, once trained, our model can be used for real-time rendering of various point cloud datasets with not only humans but also natural scenes.

2.3. Alternative Image-Based 3D Representation

Beyond explicit representation, such as point cloud or mesh, a 3D environment can also be represented in the image space, such as panoramas, light fields, or unstructured multi-view images. Transforming those data to a given viewpoint requires a view synthesis approach by tackling dis-occlusion and relighting challenges. Example solutions include multi-layer image inpainting [16, 24], or neural radiance fields [15, 18]. As an orthogonal scope of work, we focus on rendering the 3D point cloud data with high quality and speed.

3. Method

3.1. Adaptive Surface Splatting Using Elliptical Gaussians

We are given color points sampled from the visible surface in a 3D scene. Each point consists of a 3D coordinate $\mathbf{p} = (x, y, z)$ and an RGB color $\mathbf{c} = (r, g, b)$. The collection of the points form a frame of 3D point cloud $P = \{(\mathbf{p}_i, \mathbf{c}_i) | i = 1, \cdots, N\}$, and multiple frames form a point cloud video that captures a dynamic scene. These point clouds are usually captured by RGB-D cameras, LiDAR, or reconstructed from multi-view images, and are usually preprocessed to remove noise and outliers.

Given the point cloud, our goal is to render the 3D scene into 2D images from arbitrary views. We mainly address two technical problems. First, points are originally without spatial dimensions. To render smooth textured surface from different views out of points, we need to convert zero-dimensional points into 3D primitives with spatial volumes. Second, since the point clouds may be unevenly spaced and contain quantization noise introduced in compression, we need to adjust the coordinates of the primitives accordingly. Generally, we require the generated primitives to, 1) approximate the surface; 2) avoid visibility leakage; 3) produce smooth texture.

Previous work shows that 3D elliptical Gaussians have the capability to represent a scene and render smooth texture with hole-free surface [15]. Inspired by this, we propose to estimate an ellipsoid (a 3D Gaussian with an arbitrary covariance matrix) from each point, serving as the rendering primitive. For each input point $((\mathbf{p}, \mathbf{c}))$ in the original point cloud P, we estimate a Gaussian center offset $\delta = (\delta_x, \delta_y, \delta_z)$ and a covariance matrix $\Sigma \in \mathbb{R}^{3\times3}$, in effect transforming the point into a 3D Gaussian $\mathcal{G}(\mathbf{p} + \delta, \Sigma)$. Specifically, as in [15] the covariance matrix is parameterized as $\Sigma = \mathbf{R}^T\mathbf{S}^T\mathbf{S}\mathbf{R}$, where $\mathbf{R}$ is a rotation matrix and $\mathbf{S} = \text{diag}(\sigma_x, \sigma_y, \sigma_z)$ is a diagonal matrix. This parameterization guarantees Σ to be positive semi-definite. The rotation matrix $\mathbf{R}$ is calculated from a quaternion $\mathbf{q} = (q_w, q_x, q_y, q_z)$, which is also estimated by the neural network. The neural network also estimates an opacity value o for each point. Combined with the original color of the point, the final 3D primitive is $< \mathcal{G}(\mathbf{p} + \delta, \Sigma), o, c >$, consisting of 11 parameters: $\delta_x, \delta_y, \delta_z, \sigma_x, \sigma_y, \sigma_z, q_w, q_x, q_y, q_z, o$ to be estimated by the neural network.

We render the 3D Gaussians by splatting them onto the screen space and then rasterize. In practice, we set a threshold of three times the standard deviation of the Gaussian to determine the boundaries of the splats. In the rasterization, one pixel may be covered by multiple splats. Following [15], We use alpha blending to combine the colors of the splats. Let x be the screen-space coordinate of a pixel and $U = < \mathcal{G}(\mathbf{p}_k + \delta_k, \Sigma_k), o_k, c_k > | k = 1, \cdots, K$ be the set of splats that cover the pixel, sorted by the depth (*i.e.* the z coordinate after transforming $\mathbf{p}_k + \delta_k$ into the camera space). The rendered color at x is

$$C = \sum_{k=1}^{K} T_k \alpha_k c_k, \text{with } T_k = \prod_{j=1}^{k-1}(1 - \alpha_j), \tag{1}$$

where c_k is the original color of the k-th point. We obtain the opacity α_k of the k-th splat by projecting the 3D Gaussian onto the screen space and evaluate the Gaussian density at the projected point, weighted by the estimated opacity value o_k,

$$\alpha_k = o_k \cdot e^{-\frac{1}{2}\mathbf{d}_k^T \Sigma_{k(s)}^{-1} \mathbf{d}_k}, \tag{2}$$

where $\Sigma_{k(s)}$ denote the screen space covariance matrix and $\mathbf{d}_k$ is the distance between the pixel point and the Gaussian center in the screen space. It has been shown in [34] that, given the viewing transformation $\mathbf{W}$ with camera rotation $\mathbf{T}$ and translation $\mathbf{t}$,

$$\mathbf{W} = \begin{bmatrix} \mathbf{T} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}, \tag{3}$$

$\Sigma_{k(s)}$ can be approximated as,

$$\Sigma_{k(s)} = \mathbf{JT}\Sigma_k\mathbf{T}^T\mathbf{J}^T, \text{with } \mathbf{J} = \begin{bmatrix} \frac{f_x}{z}, 0, -\frac{xf_x}{z^2} \\ 0, \frac{f_y}{z}, -\frac{yf_y}{z^2} \end{bmatrix}, \quad (4)$$

where f_x and f_y are the focal lengths of the camera, and x, y, z are the camera-space coordinate of the projected point.

To enable re-lighting of the rendered images, we also estimate the surface normal of each point n_k. The surface normal is estimated by the same neural network as three additional output channels for each point. In the scenario that requires re-lighting, we first render the surface normal to 2D pixels by subsituting c_k in Eq. (1) with n_k, and then use the rendered surface normal to calculate the shading. We then conduct late shading with a pixel shader.

3.2. Neural Elliptical Parameter Estimation

Unlike the work in [15] which estimates the elliptical parameters from multi-view images using a per-scene optimized neural network, we employ a feed-forward 3D sparse convolutional neural network to estimate the elliptical parameters and surface normal from a given point cloud. The neural network is based on Minkowski Engine [8], which allows efficient 3D voxelized convolutions on sparse data like point clouds. However, the input point clouds may not be originally voxelized. Hence, we need to voxelize the point clouds before feeding them into the neural network. We design a voxelization method that is adaptive to the density of the point cloud and at the same time preserves the information. We first determine a scaling factor such that after scaling, the point cloud has an average density of 1 point per voxel. We then voxelize the scaled point cloud with a voxel size of 1. Since there will be coordinates offset after the voxelization, we also calculate the coordinate residuals, and make it as part of the input feature to the neural network. This ensures that the geometric information is kept. Finally each point we feed into the neural network has the attributes $\{\mathbf{p} = (x, y, z), \mathbf{c} = (r, g, b)$, and $\mathbf{r} = (\delta_x, \delta_y, \delta_z)\}$, namely the absolute coordinates, the color, and the coordinate residuals, respectively.

Approximating the underlying surface represented by the point cloud requires local and global information of the point cloud. To learn the global and local features of the point cloud, we use a UNet-like architecture with 3D convolutions, where the downsampling in the encoder allows the neural network to extract global features from points that are far away from the center point. The network architecture is shown in Fig. 2. We use the Minkowski Engine pooling method to downsample the sparse voxel grid. It simply takes every $2 \times 2 \times 2$ voxels and aggregate to one voxel. The new voxel conceptually lies in the center of the original $2 \times 2 \times 2$ voxels and has the mean feature of those voxels. If some of these voxels are not occupied, only non-empty voxels will take part in the average calculation. In our architecture, there are 3 downsampling layers in the encoder, effectively enlarging the voxel size by a factor of 8^3.

The upsampling in the decoder is done by transposed convolution. In an upsampling layer, each non-empty voxel in the input will generate $2 \times 2 \times 2$ voxels in the output by transposed 3D convolution. Because the corresponding voxel grid in the encoder may not be fully occupied, we prune the output voxel grid with the ground truth occupancy of the corresponding voxel grid in the encoder. This guarantees the same geometry of two sparse voxel grid coming from the transposed convolution and the skip connection from the encoder. Symmetrically, the decoder also has 3 layers, with the last layer having the same set of points as the original point cloud. Finally, for each occupied point in the final layer, the network predicts the set of 3D Gaussian parameters $< \mathcal{G}(\mathbf{p} + \delta, \Sigma), o, c, \mathbf{n} >$ from the features of that point in the decoder using a shared MLP.

One of the advantage of using sparse 3D convolution to process voxelized point cloud is that it allows efficient handling of large point clouds with millions of point on GPU. Alternatives like DGCNN [25] or PointNet++ [21] involve nearest neighbor search, which is memory and time consuming on large scale point clouds. In our experiments, we are able to process high density point clouds in real-time on a single RTX 4090 GPU.

3.3. Model Training

We train the neural network using the THuman 2.0 dataset [30]. This dataset includes meshes of 3D captured human subjects. The human subjects are captured in real-time using a multi-view system with three RGB-D cameras. After that, high-quality texture meshes are reconstructed from the capture and provided as assets.

The training of our system requires input point clouds and the ground truth images from arbitrary viewing directions. We synthesize the training pairs from the meshes. The ground truth rendered RGB images and surface normal maps are directly rasterized from the mesh with random cameras. We obtain the input point clouds for training by mixing both the quantized and non-quantized point clouds sampled from the mesh. We normalize the vertices of the training meshes to be within a bounding box of size $2 \times 2 \times 2$ centered at the origin $(0, 0, 0)$. We then use the Poisson Disk [31] algorithm to sample 800K points from each mesh. To obtain the quantized point clouds, we apply a scaling factor of 512 to the points within the $2 \times 2 \times 2$ bounding box, and round the scaled coordinates into 10 bit integers. The quantized point clouds are scaled back to be aligned with the non-quantized ones, together forming the training inputs.

To simulate real-world scenarios where the point cloud

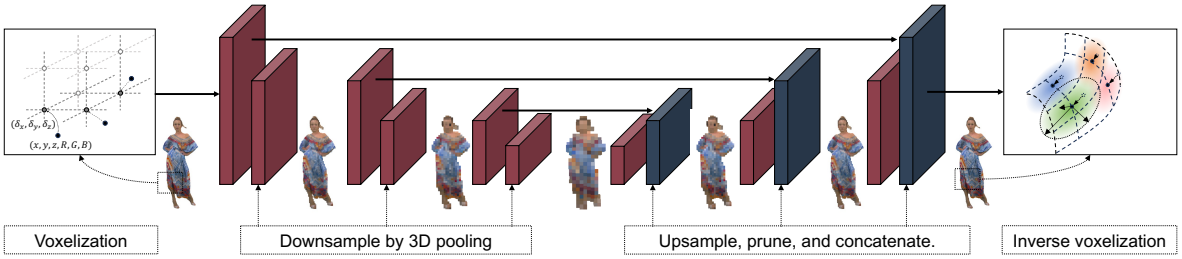


Figure 2. The UNet-like architecture with 3D convolutions. The network takes a point cloud as input and predicts the elliptical parameters and surface normal for each point.

may have sparser areas due to capturing limitation or lossy compression, we randomly downsample the point cloud for data augmentation during training. For quantized point coordinate x, we get the downsampled version $\hat{x}$ by $\hat{x} = \lfloor \alpha x \rceil / \alpha, = \alpha \in [0.25, 1]$. For non-quantized point clouds with N points, we randomly choose βN points out of the original point set, where $\beta \in [0.125, 1]$.

We randomly place virtual cameras around the scene and calculate the following loss function between the ground truth rasterized RGB images I and normal maps $\mathbf{n}$ generated from the mesh data, with the differential spaltting results $\hat{I}$ and $\hat{\mathbf{n}}$, as,

$$\begin{aligned} \mathcal{L} &= w_1 ||I - \hat{I}||_1 + w_2 \mathcal{L}_n, \\ \mathcal{L}_n &= ||\mathbf{n} \times \hat{\mathbf{n}}||_2 + w_3 \min\left\{||\mathbf{n} - \hat{\mathbf{n}}||_2, ||\mathbf{n} + \hat{\mathbf{n}}||_2\right\}. \end{aligned} \tag{5}$$

In experiments, we use $w_1 = 1, w_2 = 10$, and $w_3 = 0.1$.

4. Evaluation

4.1. Experimental Settings

Dataset To evaluate our methods on various contents and different point cloud qualities, we conduct the experiments with the following datasets:

- **THuman 2.0** [30]. The THuman 2.0 dataset contains textured meshes of human subjects, captured and reconstructed in real-time by three RGBD camaras. We train on the training split in this dataset with 250 meshes, and evaluate on the testing split with 8 unseen subjects.
- **8iVFB** [9]. This is the standard testing dataset for MPEG point cloud compression standardization. It contains high-quality dynamic voxelized point cloud videos of human in action, with each point cloud frame containing 700K to 900K points.
- **BlendedMVS** [28]. This dataset provides textured meshes of outdoor scenes, captured and reconstructed from a multi-view system. We use this dataset to evaluate the generalizability of our method to various content.
- **CWIPC** [22]. This dataset provides real-time captured raw point cloud from a multi-camera setting, specifically for social XR applications. Each frame contains $\tilde{1}$M points. Since the production of the point cloud does not include surface construction, the point clouds have discontinuous surface and inaccurate point coordinates. We evaluate on this dataset to show the capability of method to be used for live streaming.

Baselines for Comparison The main benefit of our method is to jointly provide speed and fidelity. We compare with the following methods in terms of quality and latency. Considering the total latency from the point cloud to final images into account, we categorize these methods into two groups. The first group includes **offline** methods targeting at high fidelity without considering speed.

- **Per-scene Optimized Surface Splatting** [15]. This method builds a collection of 3D Gaussians from multiview images and then rasterize the Gaussians to render arbitrary views. Note that this method requires per-scene optimization with known multi-view images. We generate these multi-view supervision images by rendering 144 images from the mesh with a virtual camera following a spiral trajectory. Because of the per-scene optimization requirement, it cannot be applied for real-time rendering. We recognize that this method is developed for 3D scene reconstruction from multi-view images and then novel-view generation, rather then for rendering a point cloud. We use it as a benchmark for the best possible quality we may obtain by building 3D Gaussians from a point cloud.
- **Pointersect** [5]. A state-of-the-art point cloud rendering method utilizing a transformer to calculate ray-point-cloud intersection for ray tracing. Despite the good quality, ray tracing is slow and hence cannot be used for real-time rendering.
- **Poisson surface reconstruction** [14]. Rendering a point cloud by first reconstructing a water-tight triangle mesh and then rasterize.

The second group includes **real-time** methods that can render within the MTP constraint:

- **OpenGL**. Each point is converted to a 1 pixel wide square in the screen space and rasterized to pixels. We use the packaged implementation provided in Open3D [33].
- **Surface splatting using the same isotropic Gaussian representing each point**. We use the same splatting renderer as our proposed method but we use a diagonal co-

variance matrix with isotropic variance for all three axes, determined globally using average point density.

For datasets that have mesh representations, we measure the fidelity of the rendered images from the point cloud compared to the ground truth obtained from the original scene mesh, using quality metrics PSNR and MS-SSIM [27]. In addition to the original point cloud, we also evaluate these methods in the scenario where the point clouds are lossily compressed for streaming. We use the standard G-PCC [11] to compress the point clouds at different bit-rate. For other datasets without meshes, we present visual comparisons for selected point clouds.

4.2. Fidelity, Latency, and Robustness to Compression Artifacts

We first evaluate the method's capability of rendering high-quality point clouds sampled from a smooth surface. From the testing split of the THuman 2.0 dataset, we create two categories of point cloud for each mesh: 1) **High-quality** non-quantized point cloud sampled from the mesh surface using Poisson Disk [31]. We sample 800K points from each individual mesh asset; 2) **Compact** point cloud uniformly sampled from the mesh, with 280K points on average, with coordinates quantized to a 10-bit depth. (see more details in Sec. 3.3). Quantization and down-sampling of points are common tools used for compression of point cloud data codec [11], necessary for efficient point cloud data storage and delivery.

We render point clouds from 12 different viewing angles, forming a circle trajectory surrounding the subject. We report the PSNR and MS-SSIM between the rendered views and the ground truth views rasterized from the meshes. For this evaluation, we render images with a 512×512 resolution.

Usually a point cloud rendering method consists of a two-stage procedure: 1) constructing primitives from the point cloud (*preprocess*), and 2) render from a camera view by rasterization. In real-time video streaming applications, it is important for the method to finish preprocessing faster than the content frame rate, in addition to have a rendering time that is within the MTP threshold. Hence we also compare these two latencies among different methods. The latencies are measured on a computer with an Intel i7-9700K CPU and an NVIDIA RTX 4090 GPU.

As shown in Table 1, our method demonstrates higher rendering quality than all compared methods, with more than 4dB improvement in PSNR than real-time baselines ("Global parameters" and "OpenGL"), for both "high-quality" and "compact" data. Our method is also substantially more robust to quantization noise and reduced point density, compared to all other methods. Whereas the PSNR only reduced from 34.1 dB for the unquantized 800K data to 33.8 dB for the quantized 280K data, some of the baseline methods suffer a reduction of 2 to 3 dB. We attribute the robustness of our method to training of the neural network using both unquantized and quantized data at randomized point densities. The model can generate accurate Gaussian parameters even when the point cloud is compressed.

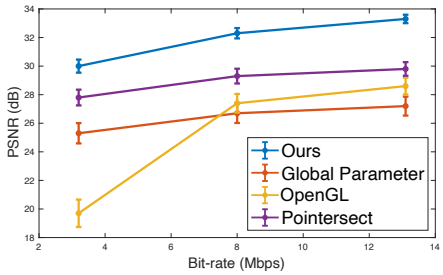


Figure 3. Average PSNR of rendered views from THuman 2.0 *compact* point clouds compressed to different bit-rates by G-PCC.

We visualize the rendering results for a sample compact point cloud in Figure 5. As shown, despite the high rendering latency, Poisson mesh reconstruction still has noticeable blurring artifacts, and Pointersect produces noisy edges due to inaccurate intersection calculation. Among the real-time methods, OpenGL fails to reproduce hole-less surface and leads to visible holes and gaps. 3D Gaussians with global parameters produce inaccurate geometry and noisy edges. Our method generally produces better visual quality than both offline and real-time methods. Please refer to the supplementary material for results with high-quality non-quantized point clouds.

Our method also enjoys very fast rendering speed once the Gaussian parameters are inferenced using the P2ENet, comparable to the real-time baselines, shorter than the MTP threshold. The preprocessing time (for inferencing the Gaussian parameters) is under 30 ms, which should be acceptable for most applications. Therefore, our rendering method enables the streaming and free viewing of a 30 fps point cloud video after an initial delay of 30 ms (considering only the preprocessing delay). Given the rendering time of under 1 ms, the proposed method enables a display frame rate of more than 100 fps. Since the rendering resolution is irrelevant to the preprocessing, our method is capable of rendering at even higher resolution at high frame-rate, *e.g.* 1K$\times$1K at 1.1 ms and 2K$\times$2K at 2.0 ms, per frame. Note that such parallel inference-rendering framework is compatible with most consumer grade computers with an integrated GPU on the CPU chip and a discrete GPU in the graphics card. The discrete GPU can be used for inference, while the integrated GPU can be used for rendering.

To more thoroughly evaluate the robustness of the rendering methods to compression, we use the standard point cloud codec G-PCC [1] to further compress the "Compact" versions of the point clouds at different bit-rates, and use different renderers to render the decoded point clouds. Note that G-PCC compresses each frame independently

Table 1. Average PSNR (dB) and MS-SSIM scores of rendered views from original point clouds in the THuman 2.0 testing set. We also report preprocessing (P) and rendering (rasterization or ray-tracing) (R) latencies.

Method	Compact (280K, quantized)				High-quality (800K, non-quantized)			
	PSNR ↑	MS-SSIM ↑	Latency (P) ↓	Latency (R) ↓	PSNR	MS-SSIM	Latency (P) ↓	Latency (R) ↓
Pointersect	30.8	0.9926	<1 ms	1 s	33.7	**0.9954**	< 1ms	1 s
Poisson	28.5	0.9739	19 s	2 ms	28.7	0.9748	40s	2 ms
Global Parameter	28.6	0.9863	<1 ms (190 ms)[1]	< 1 ms	30.3	0.9924	<1 ms (570 ms)[1]	< 1ms
OpenGL	29.2	0.9903	2 ms (190 ms)[1]	2 ms[2]	29.2	0.9903	3 ms (570 ms)[1]	3 ms[2]
Ours	**33.8**	**0.9952**	27 ms	<1 ms	**34.1**	**0.9954**	70 ms	1 ms
Per-Scene 3D GS	<u>34.5</u>	0.9946	> 5 min	< 1 ms	<u>34.2</u>	0.9942	> 7min	< 1 ms

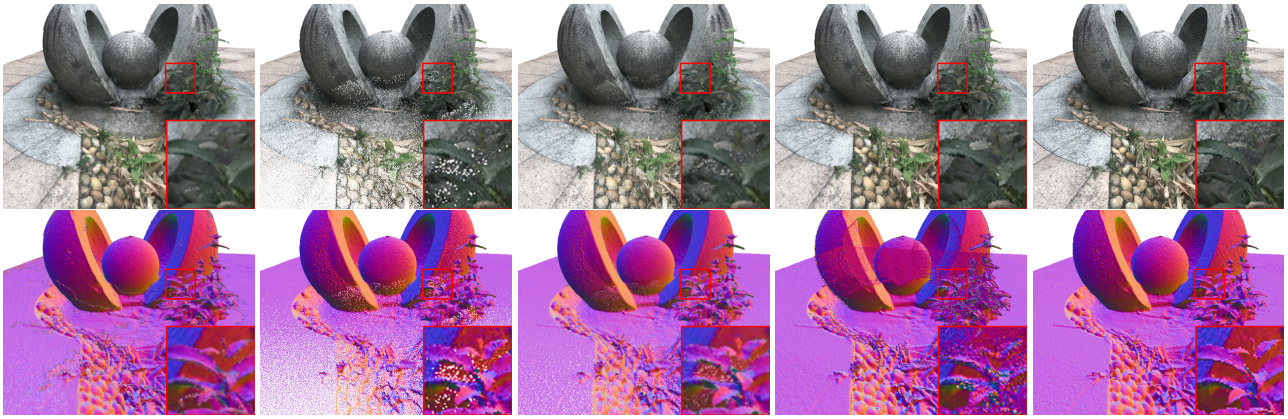


Figure 4. Comparison of rendering results of a point cloud (1.7M points) in the BlendedMVS dataset. Quality metrics are shown in format (PSNR, MS-SSIM) and calculated from the rasterization results of the ground truth mesh. The insets visualize local details with 3× zooming.

and achieves different rates by rescaling and quantizing the point coordinates (which has the effect of reducing the point density) plus additional color quantization. As shown in Figure 3, our method consistently achieves higher rendering quality at different bit-rate levels. The robustness of our rendering method to compression, in addition to its fast speed, makes it more suitable for streaming applications.

4.3. Generalizability

Although our model is trained on the THuman dataset with only high-quality 3D human scans, it generalize to different types of point clouds including outdoor scenes in the BlendedMVS dataset and noisy point clouds in the CWIPC dataset. We show rendering results with the BlendedMVS dataset in Figure 4. Please refer to the supplementary material for results with the CWIPC dataset. Despite the discrepancy in contents and capturing quality between the tested point clouds and training samples, our model still produces comparable quality among the best rendering methods. However, since our training data do not simulate misalignment capturing artifacts and severe noise, it has limitations in handling these quality degradations. We therefore leave it for an aspect of future work.

[1]The method does not directly provide surface normal. In the bracket is the time for point cloud normal estimation by Open3D on CPU.

[2]Implemented on CPU by Open3D.

5. Limitations and Future Work

Our pre-trained 3D sparse CNN (Section 3.2) allows us to estimate the elliptical Gaussian attributes from point clouds, without the per-scene training in current view synthesis methods [15, 18]. It also ensures consistent real-time speed. However, different scene content types also exhibit varied image quality metrics. For example, the human-body-based point clouds show elevated quality than natural scenes and we observe that our method tends to produce over smoothed images when the point clouds are sparse. Our model also needs to be further finetuned for scenes with a particular irregularity in point density caused by specific capturing setup for better robustness. On the other hand, since our method is optimized to produce high quality renders from each frame of the point cloud video, and does not specifically optimize for temporal consistency, the temporal jittering caused by point cloud capturing can be still observed in the rendered video.

In the future, we will invest data augmentation approaches that balances various scene types and noise levels.

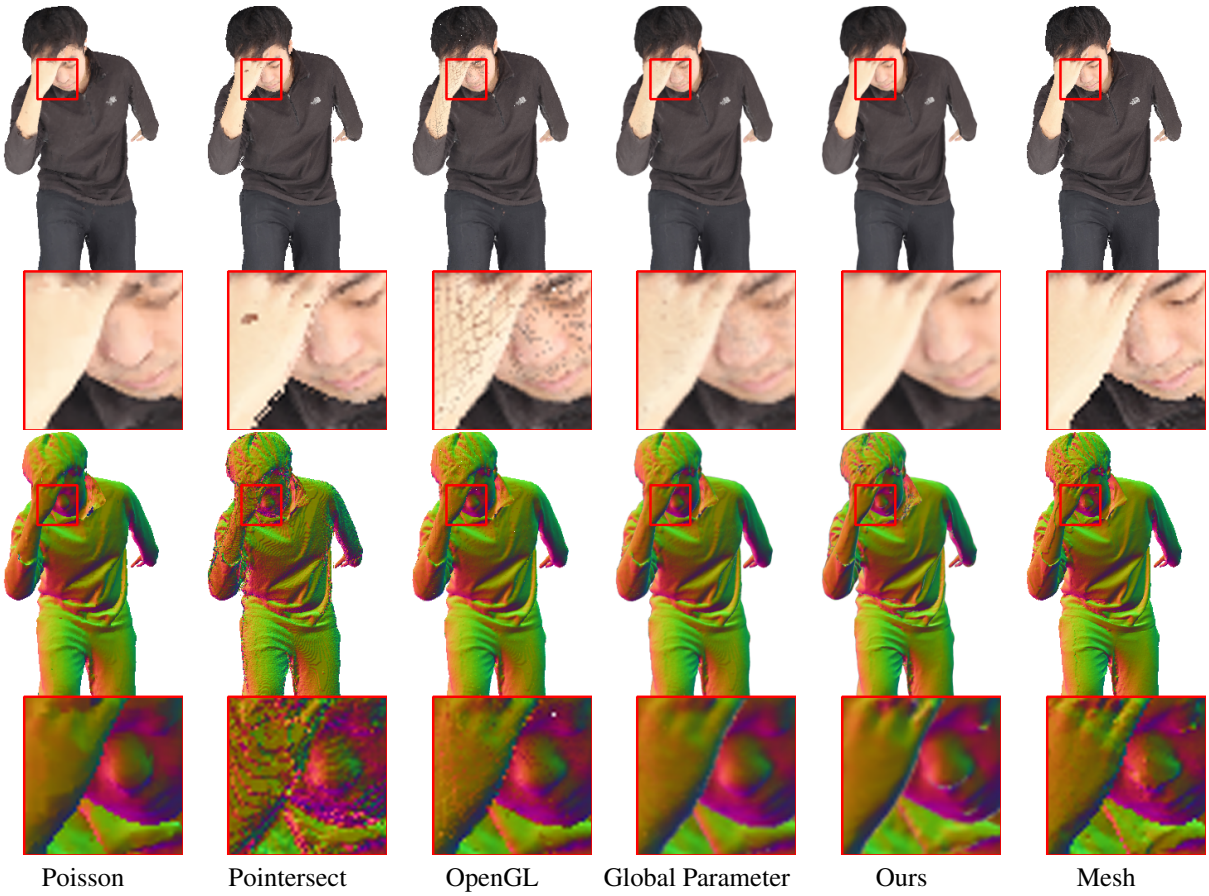


Figure 5. Rendering results in the *compact* setting of a point cloud in the Thuman 2.0 Dataset. The first row shows the rendered RGB views. The second row shows the surface normal. The insets visualize local details with $4\times$ zooming.

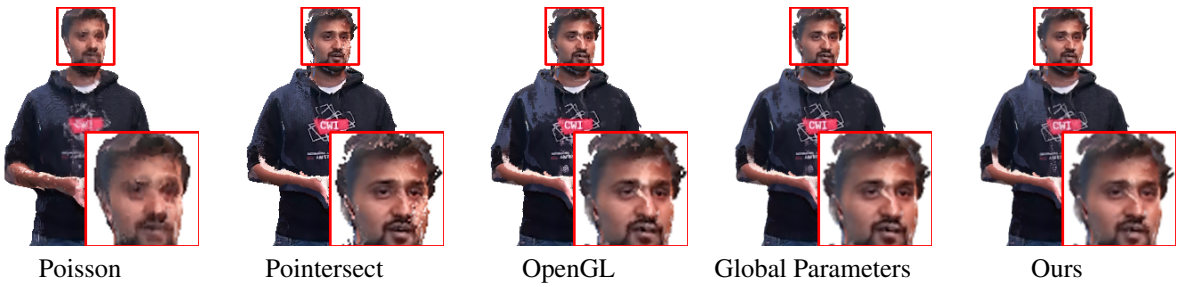


Figure 6. Rendering results from noisy raw point clouds in the CWIPC dataset. The insets are with $2\times$ and $3\times$ zooming, respectively.

We also plan to include temporal coherence constraints in model training, and further make the neural network generate denser 3D Gaussians for areas of complex texture, to improve render quality in both spatial and temporal domains.

6. Conclusion

In this paper, we present an end-to-end and learning-based framework that addresses the challenging dilemma between speed and quality in point cloud rendering. To this end, we leveraged a differentiable, splatting-in-the-loop approach that can generate fine-grained geometric and textural details through a learnt 3D sparse neural network. Extensive comparisons with a broad spectrum of datasets and alternative solutions demonstrated the effectiveness of the method. We hope the research to contribute a new building block for enabling the flexible point cloud as a promising medium toward high-fidelity interactive computer graphics, VR/AR, and immersive visual communications, while passing the required MTP for user-centric applications.

References

[1] Mpeg g-pcc tmc13, 2023. Accessed on Jan 24, 2024. 6

[2] Kara-Ali Aliev, Artem Sevastopolsky, Maria Kolos, Dmitry Ulyanov, and Victor Lempitsky. Neural point-based graphics. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXII 16*, pages 696–712. Springer, 2020. 2

[3] Mario Botsch, Michael Spernat, and Leif Kobbelt. Phong splatting. In *Proceedings of the First Eurographics conference on Point-Based Graphics*, pages 25–32, 2004. 2

[4] Giang Bui, Truc Le, Brittany Morago, and Ye Duan. Point-based rendering enhancement via deep learning. *The Visual Computer*, 34:829–841, 2018. 1

[5] Jen-Hao Rick Chang, Wei-Yu Chen, Anurag Ranjan, Kwang Moo Yi, and Oncel Tuzel. Pointersect: Neural rendering with cloud-ray intersection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8359–8369, 2023. 1, 2, 5

[6] Anthony Chen, Shiwen Mao, Zhu Li, Minrui Xu, Hongliang Zhang, Dusit Niyato, and Zhu Han. An introduction to point cloud compression standards. *GetMobile: Mobile Computing and Communications*, 27(1):11–17, 2023. 1

[7] H Childs, T Kuhlen, and F Marton. Auto splats: Dynamic point cloud visualization on the gpu. In *Proc. Eurographics Symp. Parallel Graph. Vis.*, pages 1–10, 2012. 2

[8] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3075–3084, 2019. 4

[9] Eugene d'Eon, Harrison Bob, Taos Myers, and Philip A. Chou. 8i voxelized full bodies - a voxelized point cloud dataset. In *ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG) input document WG11M40059/WG1M74006*, 2017. 5

[10] Peter Eisert, Oliver Schreer, Ingo Feldmann, Cornelius Hellge, and Anna Hilsmann. Volumetric video–acquisition, interaction, streaming and rendering. In *Immersive Video Technologies*, pages 289–326. Elsevier, 2023. 1

[11] Danillo Graziosi, Ohji Nakagami, Shinroku Kuma, Alexandre Zaghetto, Teruhiko Suzuki, and Ali Tabatabai. An overview of ongoing point cloud compression standardization activities: Video-based (v-pcc) and geometry-based (g-pcc). *APSIPA Transactions on Signal and Information Processing*, 9:e13, 2020. 6

[12] Adam Grzelka, Adrian Dziembowski, Dawid Mieloch, Olgierd Stankiewicz, Jakub Stankowski, and Marek Domański. Impact of video streaming delay on user experience with head-mounted displays. In *2019 Picture Coding Symposium (PCS)*, pages 1–5. IEEE, 2019. 1

[13] Anton S Kaplanyan, Anton Sochenov, Thomas Leimkühler, Mikhail Okunev, Todd Goodall, and Gizem Rufo. Deepfovea: Neural reconstruction for foveated rendering and video compression using learned statistics of natural videos. *ACM Transactions on Graphics (TOG)*, 38(6):1–13, 2019. 1

[14] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, page 0, 2006. 1, 5

[15] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42 (4), 2023. 2, 3, 4, 5, 7

[16] Kai-En Lin, Zexiang Xu, Ben Mildenhall, Pratul P Srinivasan, Yannick Hold-Geoffroy, Stephen DiVerdi, Qi Sun, Kalyan Sunkavalli, and Ravi Ramamoorthi. Deep multi depth panoramas for view synthesis. In *European Conference on Computer Vision*, pages 328–344. Springer, 2020. 3

[17] Katerina Mania, Bernard D Adelstein, Stephen R Ellis, and Michael I Hill. Perceptual sensitivity to head tracking latency in virtual environments with varying degrees of scene complexity. In *Proceedings of the 1st Symposium on Applied Perception in Graphics and Visualization*, pages 39–47, 2004. 1

[18] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021. 2, 3, 7

[19] Ohji Nakagami, Sebastien Lasserre, Sugio Toshiyasu, and Marius Preda. White paper on g-pcc. In *ISO/IEC JTC 1/SC 29/AG 03 N0111*, 2023. 1

[20] Hanspeter Pfister, Matthias Zwicker, Jeroen Van Baar, and Markus Gross. Surfels: Surface elements as rendering primitives. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 335–342, 2000. 1, 2

[21] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017. 4

[22] Ignacio Reimat, Evangelos Alexiou, Jack Jansen, Irene Viola, Shishir Subramanyam, and Pablo Cesar. Cwipc-sxr: Point cloud dynamic human dataset for social xr. In *Proceedings of the 12th ACM Multimedia Systems Conference*, pages 300–306, 2021. 5

[23] Darius Rückert, Linus Franke, and Marc Stamminger. Adop: Approximate differentiable one-pixel point rendering. *ACM Transactions on Graphics (ToG)*, 41(4):1–14, 2022. 2

[24] Ana Serrano, Incheol Kim, Zhili Chen, Stephen DiVerdi, Diego Gutierrez, Aaron Hertzmann, and Belen Masia. Motion parallax for 360 rgbd video. *IEEE Transactions on Visualization and Computer Graphics*, 25(5):1817–1827, 2019. 3

[25] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (tog)*, 38(5):1–12, 2019. 4

[26] Yujie Wang, Praneeth Chakravarthula, Qi Sun, and Baoquan Chen. Joint neural phase retrieval and compression for energy- and computation-efficient holography on the edge. *ACM Trans. Graph.*, 41(4), 2022. 1

[27] Zhou Wang, Eero P Simoncelli, and Alan C Bovik. Multiscale structural similarity for image quality assessment. In

The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003, pages 1398–1402. Ieee, 2003. 6

[28] Yao Yao, Zixin Luo, Shiwei Li, Jingyang Zhang, Yufan Ren, Lei Zhou, Tian Fang, and Long Quan. Blendedmvs: A large-scale dataset for generalized multi-view stereo networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1790–1799, 2020. 5

[29] Wang Yifan, Felice Serena, Shihao Wu, Cengiz Öztireli, and Olga Sorkine-Hornung. Differentiable surface splatting for point-based geometry processing. *ACM Transactions on Graphics (TOG)*, 38(6):1–14, 2019. 2

[30] Tao Yu, Zerong Zheng, Kaiwen Guo, Pengpeng Liu, Qionghai Dai, and Yebin Liu. Function4d: Real-time human volumetric capture from very sparse consumer rgbd sensors. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5746–5756, 2021. 4, 5

[31] Cem Yuksel. Sample elimination for generating poisson disk sample sets. In *Computer Graphics Forum*, pages 25–32. Wiley Online Library, 2015. 4, 6

[32] Yanci Zhang and Renato Pajarola. Deferred blending: Image composition for single-pass point rendering. *Computers & Graphics*, 31(2):175–189, 2007. 2

[33] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3d: A modern library for 3d data processing. *arXiv preprint arXiv:1801.09847*, 2018. 5

[34] Matthias Zwicker, Hanspeter Pfister, Jeroen Van Baar, and Markus Gross. Ewa volume splatting. In *Proceedings Visualization, 2001. VIS'01.*, pages 29–538. IEEE, 2001. 3

[35] Matthias Zwicker, Hanspeter Pfister, Jeroen Van Baar, and Markus Gross. Surface splatting. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 371–378, 2001. 1, 2

[36] Matthias Zwicker, Jussi Rasanen, Mario Botsch, Carsten Dachsbacher, and Mark Pauly. Perspective accurate splatting. In *Proceedings-Graphics Interface*, number CONF, pages 247–254, 2004. 2