Unbiased Gradient Estimation for Differentiable Surface Splatting via Poisson Sampling

Jan U. Müller, Michael Weinmann and Reinhard Klein

1 Institute of Computer Science, University of Bonn, Germany
2 Department of Intelligent Systems, Delft University of Technology, Netherlands

Abstract. We propose an efficient and GPU-accelerated sampling framework which enables unbiased gradient approximation for differentiable point cloud rendering based on surface splatting. Our framework models the contribution of a point to the rendered image as a probability distribution. We derive an unbiased approximative gradient for the rendering function within this model. To efficiently evaluate the proposed sample estimate, we introduce a tree-based data-structure which employs multipole methods to draw samples in near linear time. Our gradient estimator allows us to avoid regularization required by previous methods, leading to a more faithful shape recovery from images. Furthermore, we validate that these improvements are applicable to real-world applications by refining the camera poses and point cloud obtained from a real-time SLAM system. Finally, employing our framework in a neural rendering setting optimizes both the point cloud and network parameters, highlighting the framework’s ability to enhance data driven approaches.

Keywords: Differentiable rendering · Point cloud · Multipole method · Shape recovery · Scene reconstruction

1 Introduction

Inverse rendering, i.e. the inference of scene parameters such as scene geometry, reflectance or illumination as well as imaging parameters based on observations [4], has become a central problem in Computer Vision and Graphics. A wide range of applications utilize inverse rendering; including reflectance estimation [47], object reconstruction, face remapping, body pose estimation and teeth modeling [23]. Additionally, inverse rendering can improve scene understanding in robotics [57] and can be applied to a variety of problems in Geodesy [12]. A current approach that receives a lot of attention from the research community is differentiable rendering. It describes rendering methods that provide a gradient of a rendering function with respect to scene and imaging parameters, which promises a general purpose solution to the inverse rendering problem. Intuitively, the gradient conveys information about how parameters have to be changed to match a reference observation. The potential of this approach has led to the development of a variety of differentiable rendering algorithms, categorized by rendering technique or geometry representation: differentiable light transport.
Fig. 1: Illustration of our proposed algorithm to draw per-pixel samples while still scaling linear with respect to image and point cloud size. The construction steps builds a BVH tree and computes Hermite coefficients on the GPU. Afterwards the tree is traversed for each pixel in parallel and the Hermite coefficients are used to identify points with a high contributions to the pixels. Hermite coefficients might be converted into Taylor expansions to improve performance.

Our approach relies on point-based representations, which do not require the initialization to approximate the final topology and are memory efficient compared to voxel-based representations. The method interprets a Gaussian kernel in a point’s tangent space, which was introduced in surface splatting [16, 77], as a probability density of the points’ influence on the overall image. Our method does not require a truncation of the kernels to allow for efficient rendering but instead adapts the concept of importance sampling to point cloud rendering. Without a truncation step there are fewer discontinuities in our rendering pipeline. Importantly, our stochastic interpretation allows to derive an unbiased gradient estimator which is not limited to a local region around each point but approximates the gradient over the complete image space. This allows our method to overcome the major challenge of defining an useful surrogate gradient which often requires additional regularization. Previous methods only accumulate local gradients around a surfel [61, 31] or rely on a finite-differences surrogate gradient [60]. To efficiently evaluate our sample estimate, we augment a radix-tree [22] to provide an approximation of cumulative inclusion probabilities via Hermite expansions. This yields a space partitioning scheme that is used to draw samples for each pixel while scaling linearly with instance size. An overview of this framework is presented in Figure 1.
shared-memory multiprocessors in mind, consequently maps well to GPUs and is publicly available\(^1\). Our overall contributions can be summarized as:

- We introduce a stochastic interpretation of surface splatting that abstains from kernel truncation and the resulting discontinuity. This interpretation allows us to derive an unbiased estimator for the sampling based renderer without relying on surrogates such as finite-differences \(^6\) or sub-gradients \(^6\).
- The proposed sampling algorithm used to generate the necessary samples per pixel scales near linearly with respect to the image size and number of points. By design, it allows parallelization of data processing and maps well to the constraints of shared memory multiprocessors, which allows for a GPU-accelerated implementation.
- We demonstrate that our approach yields a differentiable renderer that does not require regularization and improves the fidelity of shape reconstructions over current point-based differentiable rendering methods. We further demonstrate the framework’s flexibility by exploring its use to improve scene reconstruction and train it in conjunction with a neural network.

2 Related Works

In this section, we briefly review work on the topics of differentiable rendering and discrete Gaussian transform approximation.

**Differentiable Rendering** A large body of work exists which proposes application-specific methods that can be classified as differentiable rendering \(^2\)\(^7\)\(^3\)\(^\)\(^7\)\(^4\)\(^5\)\(^6\). However, these methods use domain-specific knowledge whereas our approach is not limited to a specific application. Another parallel research direction is differentiable simulation of light transport via computationally expensive Monte-Carlo estimation to render images and propagate the gradients \(^4\)\(^3\). Recently, this method has been extended to also provide a gradient for any scene parameter including geometry and camera parameters \(^7\)\(^3\)\(^\)\(^7\)\(^2\)\(^4\). In this work, we focus on fast shape recovery from RGB images and integration into deep neural networks, instead of using more computationally expensive differentiable light transport techniques. In the remainder of this paragraph, we focus on related work that is comparable to our method.

**Voxel, Signed distance (SDF) and Implicit Representations** Early approaches use a differentiable projection to obtain a silhouette from a voxel grid \(^6\)\(^4\)\(^7\)\(^6\)\(^5\)\(^7\). Recent approaches use differentiable ray marching through a density voxel field \(^6\)\(^4\)\(^7\)\(^6\)\(^7\)\(^3\) or sphere tracing in an opaque voxel field \(^2\)\(^0\). However, voxel-based representations require large amounts of memory in order to represent fine geometric details. Deformations of the voxel-grid have been explored to address this drawback \(^3\)\(^7\)\(^8\)\(^3\). “Neural Radiance Field” (NeRF) \(^4\)\(^1\) has been demonstrated to reproduce finer details with a smaller memory footprint and has been improved by increasing inference performance \(^1\)\(^3\)\(^5\)\(^1\)\(^6\)\(^9\), making

\(^1\) https://github.com/muellerju/unbiased-differentiable-splatting
it real-time capable \cite{42, 70} or enabling scene relighting \cite{58, 7}. Point-based representations are more memory efficient than dense voxel-grids and scale to larger scenes without workarounds. Furthermore, our point based method can be incorporated into deep neural networks as opposed to real-time “NeRF” approaches for which the integration into neural networks appears to be non-trivial since they are designed for novel view synthesis of individual scenes. In addition, our rendering pipeline does not entangle geometry and reflectance representations which significantly simplifies scene relighting.

**Mesh-based Representations** Mesh-based representations have non-continuous boundaries that can be circumvented with surrogate gradients \cite{39, 25} which require regularization to avoid object shrinkage \cite{24}. The “Soft Rasterizer” \cite{36} proposes a probabilistic approximation of the rendering pipeline and has been improved to enable optimization of diffuse shading \cite{48, 10} and to increase performance \cite{49}. Recently, flexible mesh-based differentiable rendering frameworks have been introduced \cite{50} which includes hardware accelerated deferred rendering \cite{30}. However, mesh-based differentiable renderers have been demonstrated to be unable to perform complex deformations \cite{66}. This is an inherent problem of their discrete topology, which does not exist in an unstructured point cloud. Similar to the “Soft Rasterizer” \cite{36}, our proposed method uses a probabilistic interpretation to mitigate discontinuities. However, the our sampling method avoids the quadratic scaling of the “Soft Rasterizer” and has a lower variance than mesh rendering which uses uniform sampling \cite{52}. Our implementation does not utilize any graphics hardware specific instructions, which allows it to be run on any general purpose computing hardware.

**Point-based Representations** Several point-based methods have been proposed which do not present a general purpose renderer \cite{34, 18, 54}. Smoothed boundary approaches similar to the “Soft Rasterizer” \cite{36} have only been demonstrated to render low resolution patches \cite{33} or truncate the smoothed boundary to render full images but use a sub-gradient approximation at the boundary \cite{61}. “Pulsar” \cite{31} associates a sphere with each point similar to \cite{33} but proposes a more efficient acceleration structure. In contrast to these surfel methods, our method also takes the surface normal into consideration which allows for a more accurate representation of a surface given the same number of points. “Differentiable surface splatting” (DSS) \cite{66} adapts surface splatting \cite{77} into a general differentiable renderer by introducing a surrogate gradient. In our evaluation, we will highlight that our stochastic framework yields a more faithful shape recovery compare to previous methods. Additionally, our unbiased estimator avoids the use of expensive finite-differences surrogate gradients \cite{66} which enables its application to room-scale scenes. Several works combine point-rendering with neural rendering \cite{2, 55, 29}; either by rendering pixel-sized points \cite{2, 55} or by utilizing DSS in “Point-based neural graphics” \cite{29}. Pixel-sized methods render sparse images at multiple resolutions and require a rendering network to perform hole-filling. This necessitates a per-scene training of the rendering network whereas our method allows for direct optimization of large-scale scenes. Since we improve upon results obtained with DSS and demonstrate that our method can
be used in conjunction with a neural rendering network, our framework should be a drop-in replacement into the “Point-based neural graphics” pipeline.

**Discrete Gaussian Transform Approximation** The discrete Gaussian Transform (DGT) is a finite mixture of Gaussians (whereas a Gaussian mixture model is a convex combination). “Fast Gaussian Transform” [15] proposes a “fast multipole method” (FMM) based on Hermite and Taylor expansions and a spatial grid to approximate the DGT for large numbers of source and target points in linear time. Subsequent works proposed improvements to the spatial data-structure [14, 65, 32] and generalize to larger classes of kernel functions [11, 56]. In particular, [32] proposes the use of a dual kD-tree and a coupled traversal of both trees to evaluate the DGT. More recently, task-based threading models have been proposed to improve throughput on CPUs [68] and shared memory architectures [35, 62]. Our algorithm builds on previous ideas [15, 32], however, we adapt these concepts to not only provide an approximation of DGT at the root node but potentially at every node in the source tree. Furthermore, we employ domain-specific knowledge by replacing the target-tree with a grid structure, since the pixels are uniformly distributed. Lastly, we use a BVH-tree that can be efficiently constructed on the GPU [22] to store source points, which is in contrast to previous techniques [35, 62] that only accelerate the traversal with GPUs.

### 3 Statistical Splatting with Unbiased Gradient Estimator

In our rendering algorithm we avoid the additional discontinuity that comes with truncating the Gaussian kernel introduced in “surface splatting” [77]. This removes the bias that leads to shrinkage in previous “DSS” [66]. However, a naive evaluation of the complete Gaussian kernel scales quadratic (i.e. number of pixels times number of points); to obtain an efficient renderer we approximate pixel and gradient values using unbiased sample estimates. “Surface splatting” is a resampling framework for aliasing-free point cloud rendering that associates each point with a resampling kernel \( \rho_k(x) \). This kernel is approximated by projecting a Gaussian basis function from a point’s tangent plane into screen-space. Let \( N_x \) be the set of points that describe the surface at a pixel \( x \) and \( f_k \) an attribute value for each point \( k \) (e.g. albedo color). Surface splatting computes the pixel value \( f_c(x) \) at position \( x \) as the weighted sum of pixel contributions:

\[
f_c(x) := \sum_{k \in N_x} f_k \rho_k(x) \quad \text{with} \quad \rho_k(x) := \frac{1}{|J_k^{-1}|} G_{A_k} V_k J_k^T (x - m(u_k))
\]

where \( m(u_k) \) is the projection of the point position \( u_k \) into screen space, \( J_k^{-1} \) is the Jacobian of the camera transformation and perspective projection, \( V_k \) is a diagonal matrix to control the splat sizes and \( G_A(x) \) is the multivariate Gaussian density function with covariance matrix \( A \). For more details, we refer to Zwicker et al. [77]. Previous methods [77, 78] truncate the Gaussian kernel \( \rho_k(x) \) to obtain an elliptical splat. Limiting the Gaussian kernel to have a local support
decreases the algorithm’s runtime but requires a finite-differences approach \[66\] to compute a gradient. In the following, we detail a stochastic framework used in the forward and backward pass as well as an algorithm to efficiently evaluate this framework.

**Forward Pass** First, we identify a set of candidate points which could have an effect on the pixel’s filtered attribute value without taking occlusion into consideration. We propose to draw this set of points \( \hat{\mathcal{N}}_x \) for each pixel \( x \) by sampling without replacement (SWOR) such that the inclusion probability of point \( k \) at pixel \( x \) is proportional to its influence on the pixel: \( \hat{p}_k(x) \sim f_k \rho_k(c) \). Afterwards, our method evaluates the contributions of the sampled points \( \hat{\mathcal{N}}_x \) at pixel \( x \) according to Equation 1 and resolves the occlusion in a depth filtering step to determine the set of visible points \( \mathcal{N}_x \). Our algorithm uses an alpha-blending method, which was originally introduce by Zwicker et al. \[78\], to perform the depth-filtering. The alpha-based methods sorts the points in \( \hat{\mathcal{N}}_x \) by descending distance from the camera and accumulates the contributions of a point such that its contribution is at most one minus all previous contributions. Wiles et al. \[61\] have demonstrated that alpha-blending yields an improved gradient propagation compared to the z-Buffer approximation in DSS \[66\].

**Backward Pass** Second, our novel backward pass utilizes of an unbiased estimator to compute the gradient using the samples computed during the forward pass as follows: Let \( L \) be a differentiable image loss function. The derivative of a point parameter \( \omega_k \) (e.g. position or normal) can be computed by accumulating its derivative over the complete image

\[
\frac{\partial L}{\partial \omega_k} = \sum_x \frac{\partial L}{\partial f_c(x)} \frac{\partial f_c(x)}{\partial \omega_k} .
\]  

However, at pixel \( x \) we only consider the points in \( \hat{\mathcal{N}}_x \) which have been drawn by SWOR such that \( P(k \in \hat{\mathcal{N}}_x) = \hat{p}_k(x) \). To obtain an unbiased estimator for Equation 2 we make the following observations: First, the set of pixels \( \mathcal{N}_k := \{ x \mid k \in \mathcal{N}_x \} \) that are influenced by point \( k \) contains no duplicates, since \( \hat{\mathcal{N}}_x \) was drawn with SWOR and does not have any duplicates. Second, the probability that a pixel is included in \( \mathcal{N}_k \) is equal to \( P(k \in \hat{\mathcal{N}}_x) \) by construction. Finally, note that the size of the set \( \mathcal{N}_k \) is a random variable. This allows us to use a modified Horvitz–Thompson estimator \[9\] to approximate the derivative in Equation 2 as

\[
\frac{\partial L}{\partial \omega_k} \approx \frac{1}{|\mathcal{N}_k|} \sum_{x \in \mathcal{N}_k} \hat{p}_k(x) \frac{\partial L}{\partial f_c(x)} \frac{\partial f_c(x)}{\partial \omega_k} .
\]

if \( |\mathcal{N}_k| > 0 \) and \( \partial L/\partial \omega_k = 0 \) otherwise. Note that the inclusion probability is not strictly proportional-to-size, since \( \hat{p}_k \) is not scaled according to the loss derivative \( \partial L/\partial \omega_k \). However, the gradient is dominated by the exponential decay of the Gaussian kernel \( \rho_k(x) \) that outweighs the unaccounted scaling term even for pixels close to a point.

### 3.1 Efficient Pixel-wise Sampling
Algorithm 1: Our adaption of the Sequential Poisson sampling algorithm to generate a sample for each pixel in parallel.

for all all pixels \( x \) in parallel do
\( S_x \) be an empty list of size \( s \) to store sampled indices.
Initialize \( \xi = \infty \) the upper bound of transformed random numbers.
while \( \epsilon \leq \xi \cdot c_{x,r} \cdot \sigma_r(x) \) do
Let \( n \) be the root of the tree \( T \) with point indices \( I_n \subseteq [1 : N] \).
while \( |I_n| > m \) do
Let \( n_l, n_r \) be the left and right child node of \( n \).
Set \( n \leftarrow n_l \) if \( c_{x,n_l} \sigma_{n_l}(x) > c_{x,n_r} \sigma_{n_r}(x) \) else set \( n \leftarrow n_r \).
Compute and sort \( \xi_k \) in ascending order for all \( k \in I_n \).
Merge \( I_n \) with \( S_x \) to obtain a SPS sample.
Set \( \xi \leftarrow \max_{k \in S_x} \xi_k \).
Initialize the path probability \( \beta = 1 \).
while \( n \) is not the root do
Update the capacity \( c_{x,n} \leftarrow \max\{0, c_{x,n} - \beta\} \).
Let \( n' \) be the node which shares a parent with the node \( n \).
Set \( \beta \leftarrow \beta \cdot \sigma_{n}(x) \sigma_{n}(x) + \sigma_{n'}(x) \) and \( n \leftarrow \text{parent}(n) \).
\end{algorithm

In order to evaluate our sampling-based splatting framework efficiently we introduce a tree-based sampling algorithm that serves two purposes: First, the algorithm allows to approximate the total contribution \( f_c(x) \) of all points at each pixel \( x \) (i.e. \( N_x = [1 : N] \)) which allows for proportional-to-size sampling in the forward step. Second, it allows us to draw the set \( \tilde{N}_x \) for each pixel and provide the inclusion probabilities of points within the samples.

In order to achieve these objectives, our algorithm utilizes a fast multipole method (FMM) \([15, 32]\) which was originally introduced for N-body simulations to approximate a sum of Gaussians (DGT) at multiple points in linear instead of quadratic time. The idea of FMM is that a DGT of nearby points can be approximated with a truncated Hermite expansion which provides a global approximation. This Hermite expansion can be converted into a truncated Taylor expansion, which approximates the DGT only locally but is more efficient to evaluate for multiple points:

\[
 f_c(x) \approx \sum_{0 \leq \alpha \leq n} A_{\alpha} h_\alpha \left( \frac{x - x_H}{\sqrt{2h^2}} \right) \quad \text{and} \quad f_c(x) \approx \sum_{0 \leq \beta \leq n} B_{\beta} \left( \frac{x - x_T}{\sqrt{2h^2}} \right)^\beta
\]

where \( h_\alpha = e^{-t^2} H_\alpha(t) \) is a Hermite function of the Hermite polynomial \( H_\alpha \), \( A_{\alpha} \) are the coefficients of a truncated Hermite expansion at point \( X_H \) which can be converted into coefficients \( B_{\beta} \) of a truncated Taylor expansion at point \( X_T \) and \( h \) is the Gaussian’s bandwidth parameter. For details on computing and converting the coefficients we refer to \( \text{Lee et al.}[32] \). Since FMM assumes isotropic Gaussians with bandwidth \( h \), we derive a best fit bandwidth for the multivariate Gaussians. For further details on this step we refer to the supplemental.
Fig. 2: Shape reconstruction from images using the proposed method compared to results obtained using previous methods. Our method exhibits fewer outliers, more accurate normal directions and an improved reconstruction of extremities. Images were rendered after converged optimization based on the lighting conditions used in Yifan et al. [66], which was not provided but manually recreated.

**Tree Construction** For each batch of points, we construct a bounding-volume-hierarchy (BVH) tree using a radix-sort based algorithm [22]. The algorithm enables the BVH-tree construction in near linear time on the GPU. We extend the last step of the construction in this algorithm which propagates the bounding-boxes (BB) from leaves to the root, to also compute an Hermite expansion for each node within the tree: If the nodes' BB has a side length smaller than $2h$ a Hermite expansion can be computed. The expansion point $X_H$ is chosen to be the average of the child nodes’ expansion points and the Hermite degree is increased until the approximation error is smaller than a user-defined threshold $\epsilon$. Finally, the Hermite coefficients of the children are shifted and accumulated at the new expansion point. Otherwise, the BB side length is greater or equal to $2h$ and the node is marked to have no valid Hermite expansion. The shift operation between nodes, bound for the approximation error have been established by Lee et al. [32] for the use in dual-trees. In contrast to this method, our algorithm computes
Table 1: Quantitative comparison between the reconstructed and ground-truth pointsets using the Hausdorff (HD) and Chamfer (CD) distance. Our method with non-visible point repositioning outperforms previous methods significantly. The values are an average of 10 runs with a maximum std. deviation of at most 0.0121. Further details are provided in the supplemental.
Fig. 3: Comparison between test images in the dataset by Mildenhall et al. [41] and the inference from the neural renderer which uses our method as an intermediate layer. The dataset was scaled to $256 \times 256$ pixels due to limited hardware.

by pre-computing Taylor expansions for nodes that are close to the root. In a pre-processing sub-routine to Algorithm 1 the first $k'$ threads within a block compute a Taylor expansion for the first $k'$ nodes in level order and store them in shared memory. The number of pre-computed Taylor coefficients depends on the available shared memory. This accelerates the evaluation of the CDF $\sigma_n(x)$ for frequently visited nodes during the execution of Algorithm 1. After the pre-processing step Algorithm 1 is executed and uses the following heuristic to select between the evaluation techniques:

- The thread searches via linear probing for Taylor coefficients of node $n$ in shared memory. If a local expansions exists, it is used to evaluate $\sigma_n(x)$.
- Otherwise, if node $n$ has a valid Hermite expansion, its Hermite coefficients are used to evaluate the CDF.
- If the node has neither expansion, the algorithm traverses the sub-tree and evaluates the Hermite expansions of the nodes’ children to compute the CDF.

**Implementation** Our framework is implemented as two CUDA extensions for “PyTorch” [46] where each extension combines multiple kernels. An implementation as a custom “PyTorch” extension provides flexibility when exploring the choice of optimizer, learning rate schedule or loss function. It further allows our method to be easily combined with existing neural network building blocks. The majority of the extensions are written as native CUDA kernels in CUDA-C and use the “PyTorch” C++ wrapper to provide memory management as well as a Python wrapper; only the construction of the tree is implemented using the “Thrust” library [5]. We keep track of the capacity values for each pixel and
Fig. 4: Comparison between the reference image by Bode et al. [6], a rendered image from the initial SLAM reconstruction and a rendered image after refining the scene using our method. The masked areas in the reference image are a result of calibration and are not considered in the loss, PSNR or SSIM. The pixel-wise difference between the reference image and the rendered images is also depicted (darker values correspond to a smaller error). These highlight an improved alignment of the geometry with the reference image, the removal of outliers and the restoration of sharper textures after the refinement.

node combination required in Algorithm 1 by using a hash map. This map can be substantially smaller than the number of pixels times the number of points since the algorithm only needs to store the capacity of visited nodes. Additional details regarding our implementation are provided in the supplementary.

4 Applications and Results

We provide a qualitative and quantitative comparison between our method and previous differentiable point renderer in Figure 2 and Table 1. Furthermore, we demonstrate that our method can be applied to room-scale scenes with moderate hardware requirements (Figure 4) whereas DSS is unable to do so (Figure 5). Finally, we integrate our method into a neural rendering pipeline (Figure 3). Unless otherwise specified we use the “Adam” optimizer [28] with decay-rates $\beta_1 = 0.9$ and $\beta_2 = 0.999$ to minimize a simple $L_1$-image loss with a batch size of 12 without further regularization. The learning rate is initially set to be 0.01 and reduced 5 times by a factor of 2 at regular intervals during the 300 epochs. All experiments use $m = 40$ samples per pixel, an error threshold $\epsilon = 0.01$ and are run on a Nvidia GTX 1080 with 8GB VRAM. If points have no visual contribution from any viewing direction, the point is projected into the neighbourhood.
Fig. 5: Comparison between the refinement of the SLAM reconstruction performed using either DSS or our proposed method. DSS [66] requires a downsampling by a factor of 10 to be run on the available hardware; DSS was also unable to align the pose correctly (compare the upper corner of the checkerboard).

Comparison - Image-based Shape Reconstruction

To compare our method’s ability to reconstruct shapes from images against previous point-based approaches, we use the dataset published in Yifan et al. [66]. Note that this dataset originally contained 5 objects, however, only 4 of them have been published. Based on the setup in [66], the images used for the shape reconstruction are rendered from the ground-truth point cloud with randomly sampled poses. For DSS [66] we compare against the results obtained using its published implementation [67]. For “SynSin” [61] and “Pulsar” [31] we use the implementations provided in [50] and apply them using our setup described in the previous paragraph.

The results in Figure 2 demonstrate that the proposed method is able to successfully reconstruct all objects. When comparing the results to reconstruction obtained with DSS, SynSin or Pulsar, it is apparent that the results obtained with our approach exhibit fewer outliers, holes in the surface and reconstruct smaller details more faithfully. The distances between the reconstructed and ground-truth point clouds reported in Table 1 emphasize that the improved reconstruction accuracy is not limited to the viewing direction in Figure 2. The proposed method achieves distances that are smaller by a factor of at least 2.9 and 2.8 for the Hausdorff and Chamfer distance when compared to previous methods and indicates more faithfully recovered shapes. For a additional comparisons and the runtime we refer to the supplementary.

Application - Room-scale Scene Refinement

To demonstrate that our method scales well to room-scale scenes, we utilize sequence RGB-D datasets by Steinbrucker et al. [59] and Bode et al. [6]. The latter registers RGB and depth measurements by masking RGB pixels which have no corresponding depth values. For the refinement we use point-based SLAM [26] to obtain an initial estimate for the point cloud and camera poses and optimize geometry, shading...
Fig. 6: The neural rendering framework suffers from overfitting, as indicated by the discrepancy between the training and validation loss curve (a). The neural renderer fails to reproduce certain test views (b) but is able to accurately reproduce its nearest training view; compare (c) and (d).

and camera parameters to improve the alignment between RGB images and observations using our method. Further details on this setup and results are provided in the supplementary. After the refinement, the SSIM improves by 17.33\% compared to a frame rendered directly after SLAM reconstruction. The improvements exemplified in Figure 4 are a result of an improved pose alignment between frames and an improvement of shading and geometry. We would like to highlight the improved alignment of the table edge and the checkerboard pattern as a result of improved camera poses, which allows for sharper texture details. Furthermore, outliers near the table and chair were removed in the refinement using our method. In addition to the improvements of the camera poses, albedo values, and spherical harmonics coefficients, improvements to the geometry can be observed which contribute to the lower error. In contrast to “Adop” [55], which demonstrate a similar application, our method does not require training a network and avoids this overhead. We conduct the same experiments using DSS [66]. However, we found that DSS requires to downscale the point cloud by a factor of 10 to avoid out-of-memory exceptions on the used hardware. Figure 5 demonstrates that our method allows the refinement of room-scales scenes with higher fidelity compared to DSS on the same hardware.

**Application - Neural Rendering** To demonstrate the flexibility of our framework beyond shape reconstruction/refinement, we combine it with a neural shading network and train the model on the synthetic dataset by Mildenhall et al. [41]. The design of the neural shading pipeline follows an approach similar to Lassner and Zollhöfer [31]: The shading network is a U-Net generator and the model is trained to minimize an image loss and adversarial loss based on a Patch-GAN discriminator which we adapted from Isola et al. [19]. A detailed description of the initialization, architecture and all parameters is provided in the supplementary. Figure 3 depicts the results obtained by evaluating the trained model on the test split of the dataset. The neural shading network allows the model to render fine details for which the point cloud resolution would not suffice. However, the failure case in Figure 6 indicates that the neural shading model does not generalize equally well to every camera pose. Lassner and Zollhöfer [31] report a similar problem when demonstrating “Pulsar” as part of their neural rendering model.
This suggests that overfitting is not a problem of our renderer but a limitation of neural shading models.

Scaling and Ablation To study the effect of the user-defined parameters on the shape recovery results, we use the results on the bunny dataset as a baseline. We report the effect of the per-pixel sample size \( m \) in Table 2 and the algorithm’s robustness with respect to the number of viewing direction in Table 3. While increasing the sample size improves the reconstruction, a sample size beyond 40 is not possible in our implementation since it is constrained by the amount of shared memory. We found that increasing the number of views beyond 100 does result in significant improvements on this scene. We further demonstrate that Algorithm 1 scales linearly with respect to the number of points and pixels in Figure 7. We provide a more detailed study of the runtime in the supplementary.

5 Conclusion

Our stochastic framework for differentiable point-rendering addresses the inherent bias or locality in the gradient computation of previous methods by deriving an unbiased gradient estimator. To evaluate this estimator we introduce a near linear-time algorithm to perform efficient sampling in image space. We empirically verified that our proposed method enables more faithful image-based object reconstruction without relying on regularization. Furthermore, we demonstrated the scalability and flexibility of our approach by its application for room-scale scene refinement and integrating it into a neural rendering pipeline.

Future Work Although the current implementation of the sampling algorithm is hardware-agnostic, we anticipate that the integration into a hardware-accelerated rendering pipeline may be beneficial. This is in light of recent developments in neural implicit representations [42], which have significantly improved their efficiency. We believe that reducing the hardware requirements makes methods more accessible and can yield more energy-efficient models.

Acknowledgments This work was partially funded by the DFG (German Research Foundation) KL 1142/11-2 (FOR 2535 Anticipating Human Behavior).
References


