

# ADOP: Approximate Differentiable One-Pixel Point Rendering

Darius Rückert<sup>1</sup>, Linus Franke<sup>1</sup> and Marc Stamminger<sup>1</sup>

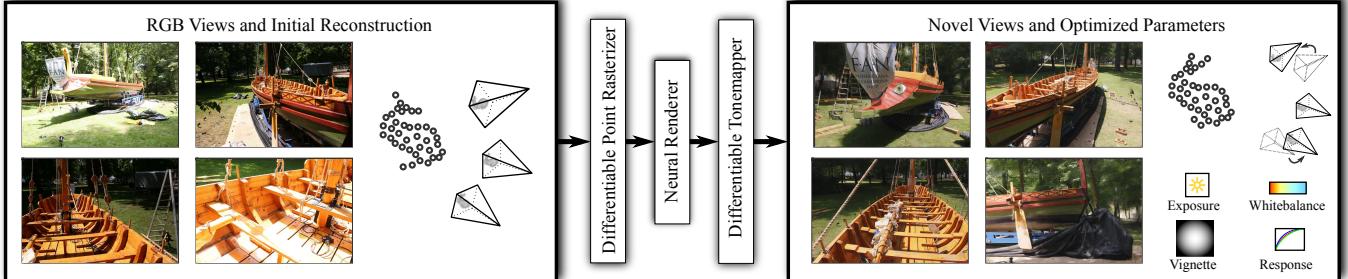


Fig. 1: Given a set of RGB images and an initial 3D reconstruction (left), our inverse rendering approach is able to synthesize novel frames and optimize the scene’s parameters (right). This includes structural parameters like point position and camera pose as well as image settings such as exposure time and white balance.

**Abstract**—We present a novel point-based, differentiable neural rendering pipeline for scene refinement and novel view synthesis. The input are an initial estimate of the point cloud and the camera parameters. The output are synthesized images from arbitrary camera poses. The point cloud rendering is performed by a differentiable renderer using multi-resolution one-pixel point rasterization. Spatial gradients of the discrete rasterization are approximated by the novel concept of *ghost geometry*. After rendering, the neural image pyramid is passed through a deep neural network for shading calculations and hole-filling. A differentiable, physically-based tonemapper then converts the intermediate output to the target image. Since all stages of the pipeline are differentiable, we optimize all of the scene’s parameters i.e. camera model, camera pose, point position, point color, environment map, rendering network weights, vignetting, camera response function, per image exposure, and per image white balance. We show that our system is able to synthesize sharper and more consistent novel views than existing approaches because the initial reconstruction is refined during training. The efficient one-pixel point rasterization allows us to use arbitrary camera models and display scenes with well over 100M points in real time.

## I. INTRODUCTION

Synthesizing realistic virtual environments is one of the most researched topics in computer graphics and computer vision. An important decision is how 3D shapes should be encoded and stored in memory. Users usually choose between triangle meshes, voxel grids, implicit functions, and point clouds [1], [2]. Each representation has different advantages and disadvantages. For efficient rendering of opaque surfaces, triangle meshes are commonly chosen. Voxel grids are often used in volume rendering while implicit functions can be used to precisely describe nonlinear analytical surfaces. Point clouds, on the other hand, have the advantage of being easy to use because no topology

has to be considered, which makes them a good candidate for intermediate output or as a representation in a scientific context. In the early 2000s, point cloud rendering, especially with point splatting, has been a well researched field in computer graphics [3]. In parallel, image-based rendering techniques gained increasing interest [4]. Based on a coarse, reconstructed 3D model, as well as a registered set of images of an object, novel views are synthesized. These approaches suffer from imprecision in the input, for example, ghosting artifacts appear if the geometry contain holes or the input images are not perfectly aligned. Neural image based rendering approaches such as [5] use neural networks to remove these artifacts and can generate photo-realistic novel views of unprecedented quality. Aliev et al. [6] show that this is also possible by pairing a traditional point rasterizer with a deep neural network. This is especially beneficial in the field of 3D reconstruction because dense point clouds are often the initial output. An unnecessary and potentially erroneous triangulation can therefore be skipped and the reconstructed scene directly visualized.

Our approach (see Fig. 1) builds on Aliev et al.’s [6] pipeline and improves it in various ways. In particular, we add a physical, differentiable camera model and a differentiable tone mapper, and provide a formulation for better approximation of the spatial gradient of one-pixel point rasterization. This differentiable pipeline allows us to not only optimize the neural point features, but also to correct imprecisions from the input during the training stage. So our system adjusts the camera pose and camera model based on the visual loss from the neural rendering network and estimates per-image exposure and white-balance values paired with a vignetting model and sensor response curve per camera. Based on this cleaned and corrected input, rendering quality is significantly improved. Further, our method is able

<sup>1</sup> Visual Computing Lab, University of Erlangen-Nuremberg, Germany  
darius.rueckert@fau.de

to synthesize arbitrary HDR and LDR setups and to correct under- or overexposed views. Finally, the number of parameters inside the deep neural network can be significantly decreased because brightness and color changes are now handled separately by a physically correct sensor model. Rendering performance is therefore increased and overfitting artifacts are reduced. In summary, the contributions of our work are:

- An end-to-end trainable point-based neural rendering pipeline for scene refinement and visualization.
- A differentiable rasterizer for one-pixel point splats using the concept of ghost geometry.
- A differentiable physically-based tonemapper that models lens and sensor effects of digital photography.
- A stochastic point discarding technique for efficient multi-layer rendering of large point clouds.
- An open source implementation<sup>1</sup> of the proposed method, which is easy to use and adapt to new problems.

## II. RELATED WORK

### A. Point-Based Rendering

Point-based rendering has been a topic of interest in computer graphics research for some time [7]. Over the past decades two orthogonal approaches have been emerged [3]. The first major approach is the rendering of points as oriented discs, which are usually called *splats* or *surfels* [8], with the radius of each disc being precomputed from the point cloud density. To get rid of visible artifacts between neighboring splats, the discs are rendered with a Gaussian alpha-mask and then combined by a normalizing blend function [9]–[11]. Further extensions to surfel rendering exist, for example, Auto Splat, a technique to automatically compute oriented and sized splats [12]. In recent years, deep learning-based approaches have been presented that improve image quality of surfel rendering, especially along sharp edges [13], [14].

The second major approach to point-based graphics is point sample rendering [15], where points are rendered as one-pixel splats generating a sparse image of the scene. Using iterative [16] or pyramid-based [15], [17], [18] hole-filling approaches, the final image is reconstructed as a post processing step. To reduce aliasing in moving scenes, points with a similar depth value can be blended during rendering [19], [20]. It has been shown that software implementations [20], [21] outperform hardware accelerated rendering through GL\_POINTS [22]. Recently developed approaches replace traditional hole-filling techniques with deep neural networks to reduce blurriness and better complete large holes [23], [24].

### B. Novel View Synthesis

Traditional novel view synthesis, which is closely related to image-based rendering (IBR), relies on the basic principle of warping colors from one frame to another. One approach is to use a triangle-mesh proxy to directly warp the image colors to a novel view [4], [25], [26]. However, imperfect

geometry leads to ghosting artifacts around silhouettes. This can be improved by replacing hand-crafted heuristics in the classic IBR pipeline with deep neural networks [5], [27]–[29]. If no proxy geometry is available, learning-based approaches have been developed that create a multi plane image representation [30]–[33] or directly estimate the required warp-field [34]–[36]. Novel view synthesis can also be performed by reconstructing a 3D model of the scene and rendering it from novel view points. Thies et al. [37] learn a neural texture on a triangle mesh which can be rendered using traditional rasterization. The rasterized image is then converted to RGB by a deep neural network. Other approaches use ray-casting to automatically learn a voxel grid [38]–[40] or an implicit function [41], [42]. It has also been shown that point clouds are suitable geometric proxies for novel view synthesis [23], [24]. *Neural Point-based Graphics* (NPBG) [6], which is closely related to our method, renders a point cloud with learned neural descriptors in multiple resolutions. These images are then used to reconstruct the final output image by a deep neural network. NPBG is very flexible as it doesn't require a textured triangle mesh as proxy and shows impressive results on public datasets. Similar to NPBG, Dai et al. show that rendering multiple depth layers of a point cloud can also be used to synthesize novel views [43].

### C. Inverse Rendering

Inverse rendering and differentiable rendering have been a topic of research for some time. However, major breakthroughs have only been made in recent years due to improved hardware and advancements in deep learning [44]. Its application can be challenging though: Traditional triangle rasterization with depth-testing has no analytically correct spatial derivative [44], [45]. Available systems therefore either approximate the gradient [45]–[48] or approximate the rendering itself using alpha blending along edges [49]–[51]. Volume raycasting on the other hand is differentiable by accumulating all voxel values along the ray [44]. This has been used by multiple authors to build volumetric reconstruction systems [52]–[54] or predict the signed distance field of an object [55]. Instead of a voxel grid, an implicit function can also be used inside a differentiable volumetric raycasting framework [56]–[59]. Mildenhall et al. show with *Neural Radiance Fields* (NeRF) that a deep neural network can be trained by volumetric raycasting to store the view-dependent radiance of an object. Due to the impressive results of NeRF, multiple extensions and improvements have been published in the following years [60]–[63]. Inverse rendering has also been proposed for point cloud rasterization [44], [64]. The spatial gradients of the points can be approximated in different ways. This includes gradient computation along silhouettes [65], gradients for Gaussian splatting [66]–[68], and gradient approximation using a temporary 3D volume [69]. A different approach is taken by Lassner et al. who render the points as small spheres instead of splats [70]. Our differentiable point rendering approach is similar to the differentiable splatting techniques of [66], [68]. However we

<sup>1</sup><https://github.com/darglein/ADOP>

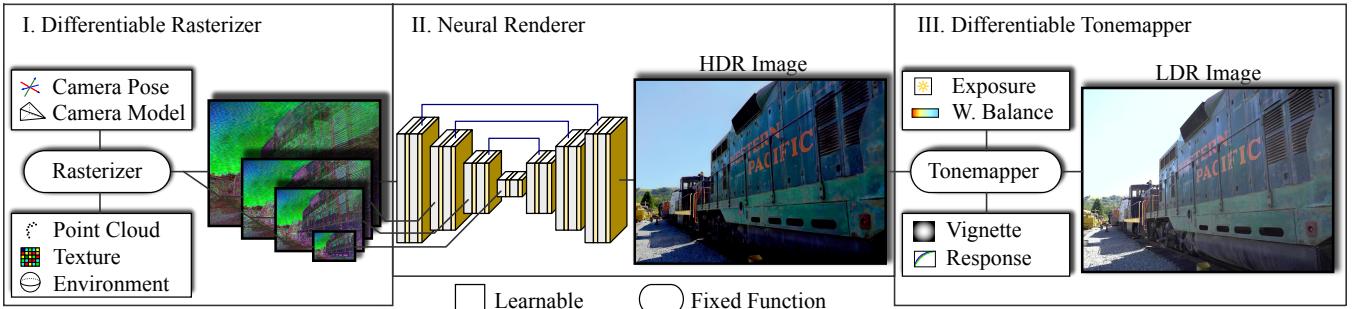


Fig. 2: Overview of our point-based HDR neural rendering pipeline. The scene, consisting of a textured point cloud and an environment map, is rasterized into a set of sparse neural images in multiple resolutions. A deep neural network reconstructs an HDR image, which is then converted to LDR by a differentiable physically-based tonemapper. All parameters in the rectangular boxes, as well as the neural network can be optimized simultaneously during the training stage.

render only one pixel per point, which allows our approach to be multiple magnitudes more efficient than competing methods [70].

### III. PIPELINE OVERVIEW

Our complete end-to-end trainable neural rendering pipeline is presented in Figure 2. As input, we use the novel frame’s camera parameters, a point cloud where each point is assigned to a learned neural descriptor, and an environment map. The output is an LDR image of the scene from the given novel viewpoint. This output image is compared to the ground truth, for example a photograph, and the loss is backpropagated through our rendering pipeline. Since all steps are differentiable, we can simultaneously optimize the scene structure, network parameters, and the sensor model.

The first step in the pipeline is the differentiable rasterizer (Figure 2 left). It renders each point as a one-pixel-sized splat by projecting it to image space using the camera parameters. If the point passes the fuzzy depth-test [20] its descriptor is blended into the neural output image. All pixels that have not been colored by a point are filled by the background color sampled from a spherical environment map. Since we render points as one-pixel-sized splats, the output image might be very sparse depending on the spatial resolution of the point cloud and the camera distance. Therefore we render multiple layers in different scales and let the neural rendering module manage occlusion and illumination. Further details on the differential rasterizer are presented after this overview in Section IV.

The neural renderer (Figure 2 middle) takes the multi-resolution neural image to produce a single HDR output image. It consists of a four layer fully convolutional U-Net with skip-connections, where the lower resolution input images are concatenated to the intermediate feature tensors. Downsampling is performed using average-pooling and the images are upsampled by bilinear interpolation. As convolution primitives, we use *gated convolutions* [71] which have been initially developed for hole-filling tasks and are therefore well suited for sparse point input. Overall the network architecture is similar to Aliev et al. [6] with one less layer and a few modifications to enable HDR imaging. First,

we remove the batch-normalization layers, as they normalize the mean and standard deviation of intermediate images to fixed values. The total sensor irradiance is therefore lost and cannot be propagated from a 3D point to the final image. Additionally, we store the neural point descriptors logarithmically if the scene’s radiance range is sizable (larger than 1 : 400). Otherwise the neural descriptors are stored linearly. For logarithmic descriptors, we convert them to linear space during rasterization so that the convolution operations only use linear brightness values.

The last step in the pipeline (Figure 2 right) is the learnable tone mapping operator, which converts the rendered HDR image to LDR. Our tonemapper mimics the physical lens and sensor properties of digital cameras. It is therefore best suited for LDR image captures of smartphones, DSLR cameras, and video cameras. A detailed description of the tonemapper is presented in Section V.

### IV. DIFFERENTIABLE ONE-PIXEL POINT RENDERING

As already mentioned in the overview, our differentiable rasterizer renders multiple resolutions of a textured point cloud using one-pixel-sized splats. Formally speaking, the resolution layer  $l \in \{0, 1, \dots, L-1\}$  of the neural image  $I$  is the output of the render function  $\Phi_l$

$$I_l = \Phi_l(C, R, t, x, n, E, \tau), \quad (1)$$

where  $C$  is the camera model,  $(R, t)$  the camera pose,  $x$  the point position,  $n$  the point normal,  $E$  the environment map, and  $\tau$  the neural texture. In the following section, we provide an in-depth explanation of the render function  $\Phi_l$ . After that we show how the gradients of each input parameter can be computed. Finally, further optimizations are presented to improve gradient robustness and rendering efficiency.

#### A. Forward

The forward pass of our point rasterizer can be broken down into three major steps, which are projection, occlusion check, and blending. The first step is the projection of each world point  $x$  into the image-space of layer  $l$ . Using the camera model  $C$  and the rigid transformation from world to



Fig. 3: One-pixel point rendering with fuzzy depth testing and threshold  $\alpha = 0$  on the left and  $\alpha = 0.01$  on the right. Geometric aliasing is significantly reduced due to blending of points with similar z-values.

camera-space  $(R, t)$ , we define this projection as the function  $P$ :

$$P_l(C, R, t, x) = \frac{1}{2^l} C(Rx + t) \quad (2)$$

The real valued result of  $P$  is then converted to pixel coordinates by rounding it to the nearest integer. A specific world point  $x_i \in \mathbb{R}^3$  is therefore projected to the pixel coordinates  $p_i \in \mathbb{Z}^2$  of layer  $l$  by

$$p_i = \left\lfloor P_l(C, R, t, x_i) \right\rfloor. \quad (3)$$

Note that the rounding operation makes the projection discrete, which requires us to approximate its derivative (see Section IV-B). After all points have been projected to image-space, we discard them if they fail at least one of the following three conditions:

$$\text{Bounds Test: } p_{ix} \in [0, w] \wedge p_{iy} \in [0, h] \quad (4)$$

$$\text{Normal Culling: } (Rn_i)^T \cdot \frac{Rx_i + t}{|Rx_i + t|} > 0 \quad (5)$$

$$\text{Depth Test: } z \leq (1 + \alpha) \min_z(p_i) \quad (6)$$

The last condition (6) is the fuzzy depth-test as described in [20]. A point passes the fuzzy depth-test if its z-value is smaller than or equal to the scaled minimum depth value at that pixel. If the camera model does not provide a valid z-value, we instead use the distance between the camera and the 3D point. A large value of  $\alpha$  in Eq. (6) increases the number of points that pass the depth-test. This leads to a smooth image but also can introduce artifacts when background points are merged into the foreground. In practice we use  $\alpha = 0.01$  as suggested by [20].

After the bounds check (4), normal culling (5), and fuzzy depth-test (6), for each pixel  $(l, u, v)$  we obtain a list of points  $\Lambda_{l,u,v}$ . If a pixel is not hit by any point, we sample the output color from the environment map  $E$  using the inverse camera projection  $C^{-1}$ . Otherwise, we sample the texture  $\tau$  of every point and write the mean into the output image. The blending function  $B_l$  can therefore be separately written as:

$$I_l(u, v, \Lambda) = \begin{cases} E(R^T C^{-1}(u, v)) & \Lambda_{l,u,v} = \emptyset \\ \frac{1}{|\Lambda_{l,u,v}|} \sum_{p \in \Lambda_{l,u,v}} \tau(p) & \text{else} \end{cases} \quad (7)$$

Figure 3 shows two color images that have been rendered using the one-pixel point rasterization technique described in

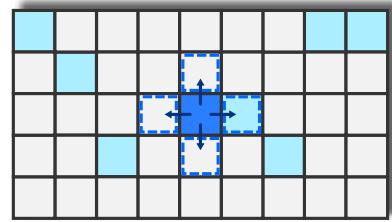


Fig. 4: Pixel lookup for the spatial gradient computation of the center (dark blue) point. The white pixels are from the background and the teal pixels are other rasterized points.

this section. We can see that with  $\alpha = 0.01$  (right image) the aliasing is significantly reduced and for example the letters are easier to read.

### B. Backward

The backward pass of our point rasterizer first computes the partial derivatives of the render function (1) w.r.t. its parameters.

$$\frac{\delta \Phi}{\delta C}, \frac{\delta \Phi}{\delta R}, \dots \quad (8)$$

Using the chain rule, we can then compute the parameter's gradient w.r.t. the loss and pass it to the optimizer. The difficult part of this calculation are the partial derivatives of the structural parameters  $C, R, t$  and  $x$ . This will be explained in the following section. The derivatives of the texture  $\tau$  and environment map  $E$  are straightforward and will not be detailed in this work. The interested reader is encouraged to study the accompanied source code.

The problem of deriving  $\Phi$  w.r.t. the structural parameters is the rounding operation Eq. (3) of our one-pixel point rasterizer. This prevents an exact derivation of the blending function (7) and forces us to approximate  $\frac{\delta I}{\delta p}$  by finite differences with a step size of  $h = 1$ . As visualized in Figure 4, we compute this approximation by virtually shifting  $p = (u, v)$  one pixel in each direction. The induced intensity change of these shifts are

$$\frac{\delta I}{\delta u} \Big|_{p=(u+1,v)}, \quad \frac{\delta I}{\delta u} \Big|_{p=(u-1,v)}, \quad \dots \quad (9)$$

from which we can approximate the gradient for  $u$  (equally for  $v$ ) at the desired location with

$$\frac{\delta I}{\delta u} \Big|_{p=(u,v)} \approx \frac{1}{2} \left( \frac{\delta I}{\delta u} \Big|_{p=(u-1,v)} + \frac{\delta I}{\delta u} \Big|_{p=(u+1,v)} \right). \quad (10)$$

To compute each induced change of Eq. (9), multiple cases have to be considered. As shown in Figure 4, the neighboring elements of  $p$  can either be background pixels or other rasterized points. If the neighbor is a point, again three difference cases have to be considered. Firstly, the virtually shifted point can be completely behind the neighbor in which case no intensity change would be induced. Secondly, the shifted point can be in front of the neighbor replacing the old color with the point's texture. Lastly, the neighbor point can have a similar depth value according to the fuzzy depth

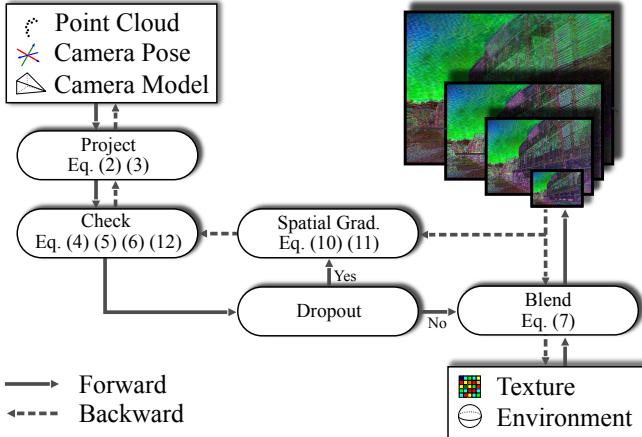


Fig. 5: Forward and backward pass of our differentiable point rasterizer using *ghost geometry*. The gradients for point position, camera pose, and camera model are only computed for points that have been discarded by the dropout module (*ghost points*). The remaining points are blended into the neural image and contribute to the gradient of the texture and environment map.

test (6), which causes us to have to compute the change of the blend function (7) if  $p$  is added to the neighboring  $\Lambda$ . In summary, the four cases to compute local spatial gradients are (with  $(i, j) = (u \pm 1, v \pm 1)$ ):

$$\frac{\delta I}{\delta u} \Big|_{p=(i,j)} = \begin{cases} \tau(u, v) - I_l(i, j) & \Lambda_{l,u,v} = \emptyset \\ 0 & z > (1 + \alpha) \min_z(u, v) \\ \tau(u, v) - I_l(i, j) & z(1 + \alpha) < \min_z(u, v) \\ \frac{|\Lambda_{i,j}|I_l(i,j) + \tau(u,v)}{1+|\Lambda_{i,j}|} - I_l(i,j) & \text{else} \end{cases} \quad (11)$$

### C. Ghost Gradients

Using Eq. (10) and (11) we can now compute the image-space gradient  $\frac{\delta I}{\delta p}$ . Assuming that the camera model is differentiable, the desired partial derivatives (8) are computed with the chain rule. However, after implementing a direct bundle adjustment with perceptual loss, we found that the gradients in Eq. (11) are not accurate. In some extreme cases, we even observed divergence of the optimization. We believe that this problem comes from high variations inside perceptual loss paired with the large step size  $h = 1$  in gradient computation. Other work may have found similar results as they mention that gradient computation is not viable for one-pixel points [67] and has to be skipped [70].

However, we propose a trick to significantly improve the gradient quality and make differentiable one-pixel point rasterization viable. The idea is that a large  $h$  is usually only feasible if the current solution is far away from the optimum. Once the optimization converges to the solution,  $h$  has to be reduced for a better gradient approximation. Since we cannot reduce  $h$  in one-pixel point rasterization, we

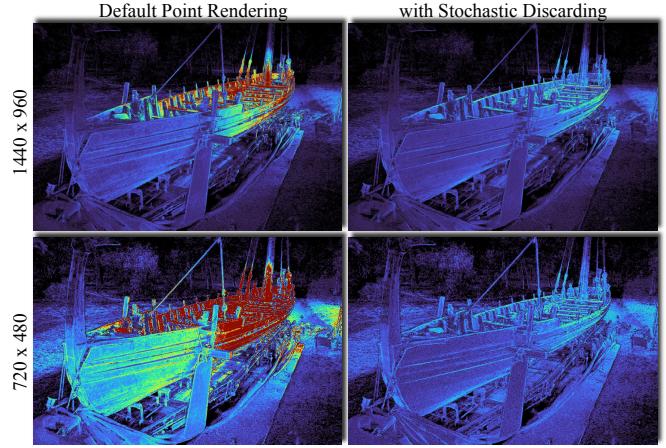


Fig. 6: The number of blended points per pixel with original rendering [20] (left) and our stochastic discarding technique (right). Especially in low resolution layers (bottom), our approach can significantly reduce the number of blended points with a total speed-up of over 20%.

apply this concept by artificially worsening the image using *ghost geometry*. As shown in Figure 5, a dropout layer is inserted before the blending stage, which divides the point cloud into two sets. The first set is blended normally and contributes to the output image. These points are used to compute the texture gradient  $\frac{\delta \Phi}{\delta \tau}$ , though we do not compute any spatial gradients for them. The second set, which we call the *ghost points* is not used in the forward pass. They are only rendered during backpropagation and the spatial gradient (11) is evaluated at the projected positions. We further analyze *ghost points* in the evaluation (Section VI-B) and show that they improve the convergence by around 20%.

### D. Stochastic Point Discarding

During further performance analysis we found that especially at the small resolution layers, hundreds of points can pass the fuzzy depth-test of a single pixel. To reduce this number to a reasonable range, we apply a stochastic point-discarding technique similar to [72]. As a first step, we compute the world-space radius  $r_{world}$  for each point inside its local neighborhood. This radius is projected into screen-space during rendering, which gives a rough approximation of the circular splat-size  $r_{screen}$ . Based on the splat-size we can then discard points stochastically to obtain a desired number of points per pixel. For example, if we want to render roughly one point per pixel and we have computed the screen-space splat size as one quarter of a pixel, then we want to discard this point by a probability of  $\frac{3}{4}$ . We achieve this by assigning each point a uniform random value  $\beta \in [0, 1]$  and render it only if the following condition succeeds:

$$\frac{r_{screen}}{\sqrt{1-\beta}} > \frac{1}{\gamma} \quad (12)$$

The parameter  $\gamma$  roughly controls how many points are rendered per pixel. For example, if  $\gamma = 1$  all points with  $r_{screen} > 1$  are rendered because  $\sqrt{1-\beta}$  is always less than

one. Only if a point covers less than one pixel, Eq. (12) starts discarding points based on the sampled  $\gamma$ . In our experiments, we use  $\gamma = 1.5$  for all datasets and scenes. The effect of stochastic discarding is visualized in Figure 6 by coloring each pixel based on the number of blended points. A blue color means that only a few points have been blended, while a red color indicates that over 100 points contribute to that pixel. Using our stochastic discarding technique (right images), the point density is close to uniform over the image.

### E. Implementation Details

To implement the forward pass efficiently, we mainly follow the work of [20]. They propose a three-pass point rasterizer that outperforms traditional point rendering with `GL_POINTS`. After a depth-only render pass, the color is accumulated for all points that pass the fuzzy depth-test. During accumulation, we track the number of points per pixel so we can compute (7) without writing the set  $\Lambda_{u,v}$  to memory. To further improve efficiency, we have adopted the proposed blocked-morton-shuffle on the point cloud and use optimized local reductions inside the rasterization kernels. A side-effect of not explicitly storing  $\Lambda_{u,v}$  is that during backpropagation all points have to be rendered again. However, this re-rendering only marginally impacts performance as the total backpropagation time is dominated by the neural networks.

To further improve the pose estimation of our inverse rendering pipeline, we follow state-of-the-art SLAM systems [73]–[75] that optimize the rigid transformation  $(R,t) \in SE(3)$  in tangent-space. Linearized pose increments are expressed as the Lie-algebra elements  $x \in \mathfrak{se}(3)$  and applied to a transformation by

$$(R', t') = \exp(x) \cdot (R, t), \quad (13)$$

where  $\exp(x)$  is the exponential map from  $\mathfrak{se}(3)$  to  $SE(3)$ . To implement tangent-space optimization in Torch [76] [77], we create a tangent tensor for the pose which is updated by the optimizer. After each step, the tangent is applied to the original pose (13) and reset to zero.

## V. DIFFERENTIABLE TONE MAPPING

The HDR output image of the neural renderer is converted to LDR by a learnable tonemapping operator. Our tonemapper mimics the physical lens and sensor properties of digital photo cameras. For cameras directly supporting HDR imaging, a more straightforward adaption can be used. The first tone mapping stage applies brightness correction to the HDR image  $I_{HDR}$  using the estimated exposure value  $EV_i$  of image  $i$ .

$$I_e = \frac{I_{HDR}}{\log_2(EV_i)} \quad (14)$$

If image meta information is available, we initialize  $EV_i$  using Eq. (15), where  $f$  is the f-number of the optical system,  $t$  the exposure time in seconds,  $S$  the ISO arithmetic speed rating, and  $\bar{EV}$  the mean exposure value of all images.

Otherwise, if no meta information is available, we initialize  $EV_i$  to zero for all images.

$$EV_i = \log_2\left(\frac{f_i^2}{t_i}\right) + \log_2\left(\frac{S_i}{100}\right) - \bar{EV} \quad (15)$$

After brightness correction, we compensate for a changing white balance by estimating the white point  $(R_i^w, G_i^w, B_i^w)$  for each image. The white balance is applied using Eq. (16)

$$I_w = I_e \odot \begin{bmatrix} \frac{1}{R_i^w} & \frac{1}{G_i^w} & \frac{1}{B_i^w} \end{bmatrix}^T, \quad (16)$$

where  $\odot$  is the element-wise multiplication of each texel with the vector. If no prior knowledge is available the white point is initialized to  $(1, 1, 1)$ . During optimization, we keep  $G_i^w$  constant so that the white point can not change the total image brightness.

As a next step, we model the *vignette* effect of digital cameras, which is a radial intensity falloff due to various optical and sensor-specific effects. The model we use is the polynomial vignette model of [78], which first computes the distance  $r$  of a pixel  $p$  to the vignette center  $c_v$ . Then, a polynomial is evaluated to compute a linear scaling factor from  $r$ . For better stability, we normalize  $p$  and  $c_v$  to be in the range  $[0, 1]$ . The coefficients  $a_i$  are initialized to zero and  $c_v$  is initialized to the image center.

$$r^2 = |p - c_v|^2 \quad (17)$$

$$I_v = I_w \cdot (1 + a_2 r^2 + a_4 r^4 + a_6 r^6) \quad (18)$$

The last stage in the tone mapping operator maps the linear RGB values to non-linear image intensities. This mapping consists of applying the camera response function (CRF) [79] and an optional color space conversion from RGB to sRGB. To simplify the optimization we combine both operations into a single function  $R$ , which is implemented using a 1-D texture for every color channel.

$$I_{ldr} = R(I_v) \quad (19)$$

To guide the optimization towards a plausible response function, we initialize  $R(x)$  to  $x^{0.45}$  and add the following additional constraints based on [80]:

$$\begin{aligned} R(0) &= 0 \\ R(1) &= 1 \\ R''(x) &= 0 \end{aligned}$$

These constraints ensure that the whole intensity range is covered and the response function is smooth. Overexposed and underexposed pixels are clamped to the maximum and minimum output values of 1 and 0. This matches digital camera behavior, however we found that an end-to-end training of the rendering pipeline is not reliable when implementing the clamping. Depending on the random network initialization, it is possible that the whole image is over/under exposed generating a zero gradient over the complete image. Inspired by the LeakyReLU activation function [81], we

	Geometry	# Layers	Forward	Backward	Forward	Backward	Forward	Backward	Forward	Backward
Synsin	Disc Splats	1	856.63	14.14	1692.63	17.15	3859.1	31.16	7342.05	25.73
Pulsar	Spheres	1	45.18	4.8	78.29	7.66	124.72	13.45	209.17	17.2
GL_POINTS	1-Pixel Points	1	0.3	×	0.54	×	1.09	×	1.85	×
Ours	1-Pixel Points	1	0.83	0.62	1.09	0.78	1.59	1.39	2.33	2.34
Ours	1-Pixel Points	4	1.4	0.77	1.82	1.24	2.52	1.74	3.65	2.8
<b>Ours + Stoc. Disc.</b>	<b>1-Pixel Points</b>	<b>4</b>	<b>1.28</b>	<b>0.66</b>	<b>1.66</b>	<b>1.07</b>	<b>2.15</b>	<b>1.66</b>	<b>3.08</b>	<b>2.64</b>
# Points			1,348,406		2,570,810		5,400,615		10,283,243	

TABLE I: Forward and backward render-time in milliseconds of a  $1920 \times 1080$  image on a RTX 2080 Ti. In comparison to other differentiable renderers, our approach is around two magnitudes more efficient. The highlighted row is our approach with stochastic point discarding (see Section IV-E) that we use for scene refinement and novel-view synthesis.

define a separate response function  $R_t$  during training that leaks small values instead of clamping them.

$$R_t(x) = \begin{cases} \alpha x & x < 0 \\ R(x) & 0 \leq x \leq 1 \\ \frac{-\alpha}{\sqrt{x}} + \alpha + 1 & 1 < x \end{cases} \quad (20)$$

The last term of Eq. (20) asserts that the maximum leaked value is  $(1 + \alpha)$ . This is important in HDR scenes, because an overexposure of multiple magnitudes should not create a large gradient. In practice we use  $\alpha = 0.01$ , which results in a maximum image intensity of  $R_t(\infty) = 1.01$ .

## VI. EXPERIMENTS

We have conducted several experiments to verify the effectiveness of our method. In Section VI-A, we compare the runtime of our forward and backward one-pixel point rasterization to other differentiable rendering systems. Then, we perform an ablation study on ghost gradients (Section VI-B) and show how our inverse rendering pipeline is able to align images to a point cloud (see Section VI-C). Finally, we evaluate the use-case of novel-view synthesis on various datasets including HDR scenes (Section VI-D and VI-E). For further experiments and evaluation of temporal stability we refer to the supplementary video at:

<https://youtu.be/WJRyu1JUtVw>

### A. Runtime Performance

Runtime performance has been a limiting factor for differentiable rendering systems in the past [44]. Most software rasterization techniques exceed the 100 ms barrier even for small scenes and render resolutions [70]. This limits their usefulness in real-world applications such as 3D reconstruction from high-resolution photographs. Currently, the two most performant differentiable rendering methods that are able to process point-cloud data are Synsin [67] and Pulsar [70]. Synsin, which is the default point render engine of PyTorch3D [82], splats each point to a disk and blends the  $K$  nearest points of each pixel into the output image. Pulsar converts each point to sphere and blends them with a similar approach as Soft Rasterizer [50]. Both methods are fully differentiable, meaning that the point position and color can be optimized during rendering.

Table I shows the measured GPU frame-time for Synsin, Pulsar, our approach, and OpenGL’s default point rendering with GL\_POINTS. These timings only include the rasterization itself without the neural network and tonemapper described in the previous sections. For our method, we also include the rendering time of four layers in different resolutions. This is a more fair comparison to the other methods because all four layers are required for the neural rendering network. The output of Synsin and Pulsar is more complete and therefore a single layer can already be successfully used. The right most columns of Table I show the forward and backward render time of a  $1920 \times 1080$  image for a point cloud with around 10M points. Both Synsin and Pulsar are not real-time capable at such dimensions with forward timings of 7342 ms and 209 ms respectively. Our approach takes two magnitudes less time than Pulsar with a combined rendering time of 3.65 ms for all four layers. This result is expected though because previous work has shown that software one-pixel point rendering can outperform hardware rasterization techniques [20]. Point splatting and sphere rendering is inherently more complex because each point effects multiple output pixels.

If we enable stochastic discarding (see Section IV-E), the rendering performance is further increased. The largest gains are achieved for multi-resolution rendering of large point clouds. For example, generating a multi-layer image of the cloud with 10.4M points takes 19% less time with our stochastic discarding approach. However, if only a single image layer is required, the speedup due to stochastic discarding reduces to 3%, as points are only discarded if they fill less than one pixel in the output image. The pixels inside the low resolution layers are much larger and therefore more points are discarded. In comparison to native GL\_POINTS rendering, our approach is only slightly slower (by about 26%). This is an impressive result, because we have implemented a three-pass blending approach with fuzzy depth-test as described in Section IV-E. The GL\_POINTS reference implementation in Table I uses a single pass without blending and standard GL\_LESS depth test.

### B. Ghost Gradient Ablation Study

In Section IV-C, we have proposed the novel concept of ghost gradients for differentiable one-pixel point rendering. The basic idea is that spatial gradients are only computed for

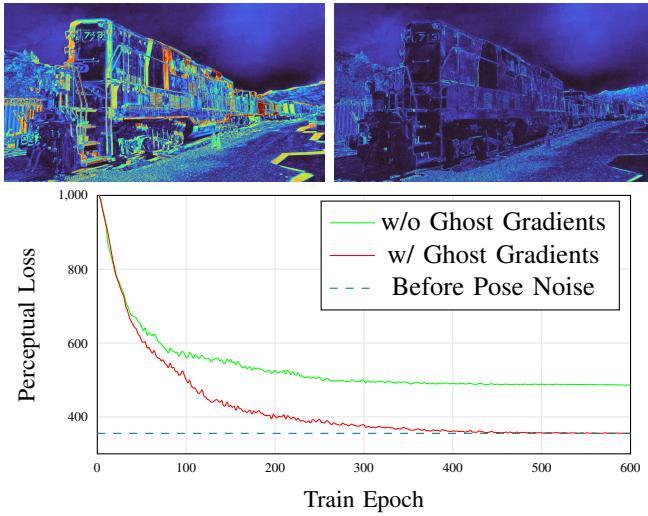


Fig. 7: Image to point cloud alignment after adding an artificial rotational and translational error to each camera pose. The top images show the pixel error before the first iteration and after the last iteration. Using ghost gradients, our systems converges to the initial solution before the noise has been added. Without ghost gradients, 20% of the images were not aligned correctly resulting in a larger perceptual loss.

points that have been discarded by the dropout module. We have claimed that ghost gradients improve gradient accuracy and increase robustness during scene refinement. In the following, we provide an ablation study to confirm this claim.

To this end, we first train our pipeline on one of the multi-view stereo scenes from the tanks and temples dataset [83]. We are now able to accurately synthesize each image of the input sequence with an average perceptual loss of 355. Then we randomly pick 30 images and add Gaussian-distributed positional and rotational noise to the camera pose. The standard deviation of the Gaussian noise is 15 mm and  $1^\circ$  respectively. The neural rendering of these images is now off by 5 to 30 pixels and the average perceptual loss of these images is 1030. Finally, we setup an optimizer that refines the camera pose based on the perceptual loss for 600 epochs. One time with ghost gradients enabled and one time with ghost gradients disabled. All other parameters, e.g. the neural texture, are kept constant during this optimization. The results of this experiment are presented in Figure 7. The images at the top illustrate the pixel error between the synthesized image and the ground truth before and after pose optimization. The graph at the bottom shows the mean perceptual error during training. We can clearly see that with ghost gradients the perceptual loss converges towards the initial solution before adding positional and rotational noise. The images are therefore correctly aligned to the scene. Without ghost gradients (green line in Figure 7) the optimization converges to a worse local optimum with a mean perceptual loss of 486. In this local optimum, only 80% of the 30 test images were aligned correctly. The remaining 20% have been stuck in a local minimum and sometimes

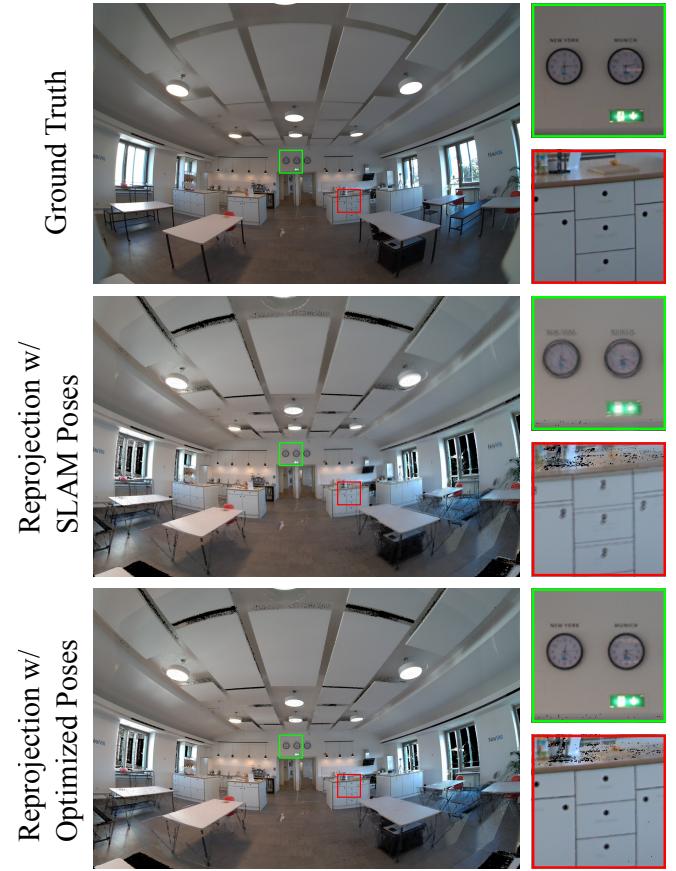


Fig. 9: The initial camera pose estimates of the SLAM-System are slightly misaligned w.r.t. the LiDAR point cloud. Reprojecting the pixel color of several source views into a target view produces ghosting artifacts (center row). Our system is able to optimize the camera poses resulting in almost pixel perfect reprojections (bottom row).

moved even further away from the correct solution. We have repeated this experiment five times with different random seeds for the pose noise. In each run, the ghost gradient optimization outperformed traditional gradient computation. This shows that ghost gradients indeed improve robustness for differentiable one-pixel point rendering. From here on, we enable ghost gradients for all further experiments.

### C. Image to Point Cloud Alignment

Another application of our differentiable rendering pipeline is the alignment of camera images to point-clouds that were reconstructed by external devices. NavVis provided us a dataset captured by the VLX mobile scanning platform [84]. This platform consists of a high performance LiDAR scanner and four  $5472 \times 3648$  pixel fisheye cameras. Their reconstruction software is able to combine multiple laser scans into a consistent point cloud and provides camera pose estimates for panorama generation and point cloud coloring. However, the image to point cloud registration is not perfect. Small errors during the SLAM-based tracking and vibrations due to the hand-held operation result in pose errors in the scale of millimeters. If we then use these

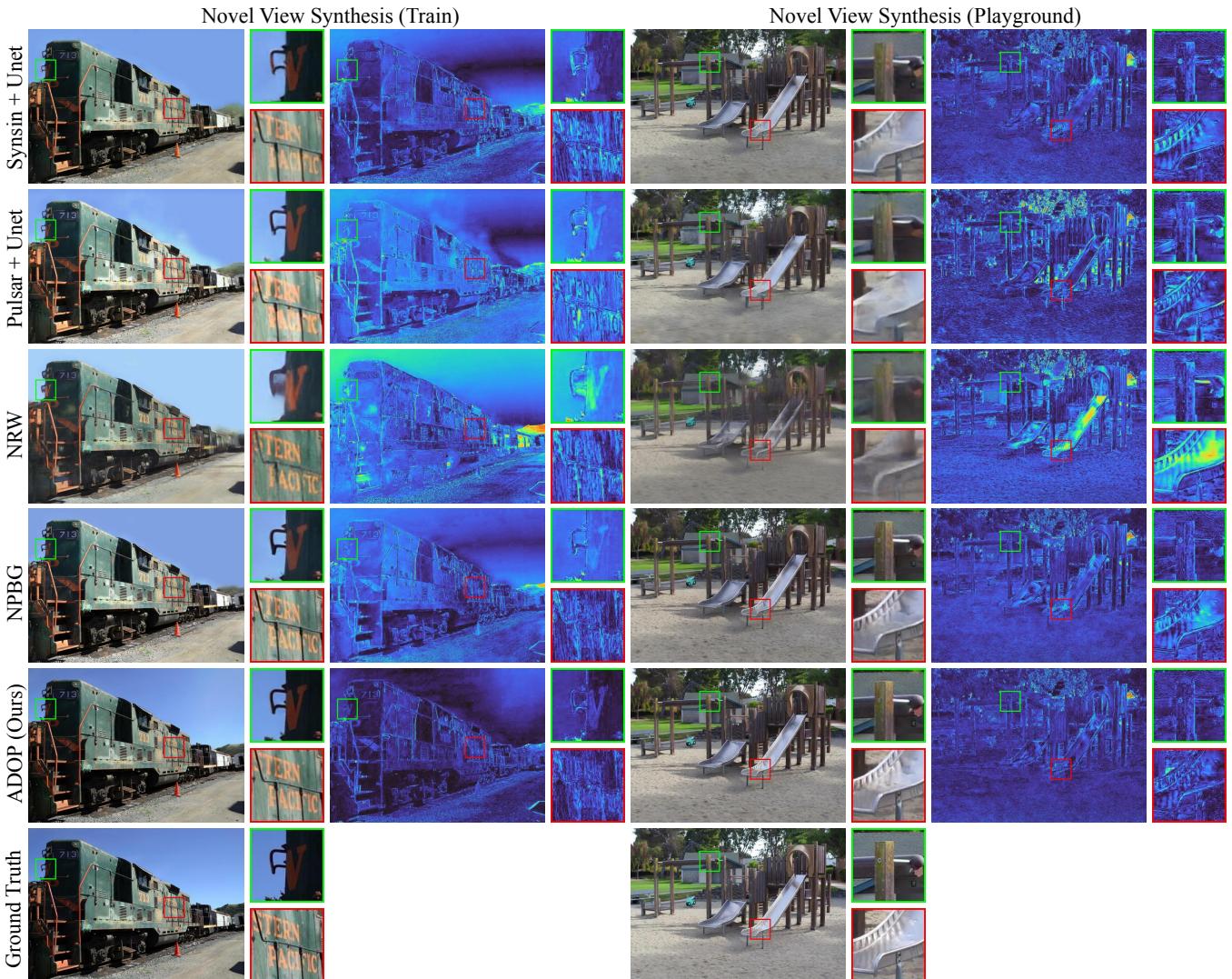


Fig. 8: Comparative results of novel-view synthesis on the *Train* and *Playground* scene from the tanks and temples dataset [83]. The ground truth reference image is placed in the last row. The blueish images illustrate the per-pixel error after comparing the generated image to the ground truth. All methods show great result on both views however our approach preserves the most detail and provides a better color and brightness accuracy. A quantitative evaluation of this task can be found in the table below.

Method	Train			Playground			M60			Lighthouse		
	VGG ↓	LPIPS ↓	PSNR ↑	VGG ↓	LPIPS ↓	PSNR ↑	VGG ↓	LPIPS ↓	PSNR ↑	VGG ↓	LPIPS ↓	PSNR ↑
Syntsin + Unet	706.0	0.3853	16.97	521.3	0.3198	21.81	564.3	0.2904	20.16	679.7	0.3655	15.41
Pulsar + Unet	677.9	0.3418	17.78	661.2	0.3849	20.00	508.2	0.2403	21.42	587.1	0.3112	17.82
NRW	817.5	0.4552	14.44	632.8	0.4110	19.47	741.2	0.4476	16.96	709.5	0.3835	14.88
NPBG	553.6	0.2290	16.92	399.0	0.1892	23.04	385.7	0.1576	<b>23.81</b>	524.5	0.2270	16.38
ADOP (ours)	<b>475.1</b>	<b>0.1899</b>	<b>20.24</b>	<b>374.4</b>	<b>0.1710</b>	<b>24.00</b>	<b>385.6</b>	<b>0.1567</b>	23.78	<b>409.3</b>	<b>0.1700</b>	<b>21.07</b>

TABLE II: Quantitative evaluation of novel view synthesis on four scenes of the tanks and temples dataset. The values in this table represent the mean loss over all test images. All approaches have been trained using the VGG preceptual loss. Our method (last row) achieves the best scores on the *Train*, *Playground*, and *Lighthouse* scenes. On the indoor-scene *M60*, the results of our approach are almost identical to NPBG. The other differentiable rendering systems, Syntsin and Pulsar, show consistently worse results than NPBG and ADOP.

poses for precise operation, such as reprojecting the color of neighboring source images into a target view, small ghosting artifacts can be observed (see Figure 9 center row). We train our neural rendering pipeline on this office dataset, which is composed of 688 images and 73M points, to synthesize all captured views. During training, our system also optimizes the camera pose of each frame. The refined poses are then used to reproject the colors into the same target frame as before (see Figure 9 bottom row). It can be seen that the ghosting artifacts are mostly eliminated and the synthesized image is a lot sharper. This experiment shows that the proposed method is able to perform pixel-perfect alignment of fisheye camera images to a point cloud of a LiDAR scanner. To our knowledge, no other available differentiable renderer can fulfill this task, as they assume a pinhole camera model or are not able to handle a point cloud with 73M points.

#### D. Novel View Synthesis

In addition to scene refinement, our method can be used to synthesize novel views on multi-view stereo datasets. We show in this section that our approach outperforms state-of-the-art neural rendering pipelines that operate on point-cloud input data. In particular, we provide a qualitative and quantitative evaluation of novel view synthesis on four scenes of the tanks and temples dataset [83]. These scenes are *Train*, *Lighthouse*, *Playground*, and *M60*. They consist of 300-350 images in Full-HD resolution captured by a high-end video camera. We use COLMAP [85] to reconstruct the dense point-cloud as well as the camera extrinsics and intrinsics. The number of points in the datasets are in order 10M, 8M, 12M, and 11M. For evaluation, 5% of the input frames are put aside for testing. The remaining images are used to train each pipeline.

The methods we compare against are NPBG [6], NRW [24], Pulsar [70], and Synsin [67]. The latter two are general differentiable rendering front-ends that have been adapted by us to the novel view synthesis problem. We use them to render the point-geometry into a neural image that is then passed through a deep neural network for the final output. They can be seen similar to our approach and NPBG with the difference of using sphere and splat-based rendering instead of a multi-layer one-pixel point rendering. Pulsar and Synsin only support pinhole cameras. Therefore we train them on the undistorted images and camera parameters provided by COLMAP. During evaluation the synthesized undistorted images are distorted again and compared to the ground truth. NPBG, NRW, and our approach are trained on the original distorted images. These methods use one-pixel point rendering, which natively supports all camera models.

In Figure 8, we show two of the test frames synthesized by all of the previously described methods. The bottom row is the ground truth image, which has not been seen during training. Column two and four visualize the per-pixel error of the rendered image in comparison to the ground truth. A dark blue color means the error is small, while a brighter, reddish color indicates a large pixel error. When comparing

the result images, we can see that Synsin, NPBG, and our approach can synthesize the reference frame very well. The output of Pulsar and NRW is slightly worse as indicated by the blurriness of the image. A detailed look reveals that the renderings of our methods preserve color, brightness, and sharpness better than the other approaches. The text on the train has slightly more contrast and the reflections on the playground slide are closer to the reference reflections. However, we also note that a visual inspection is insufficient because especially the results of NPBG are very close to ours. Therefore, we also provide a quantitative evaluation in Table II. This table shows the mean VGG loss [86], LPIPS loss [87], and peak signal-to-noise ratio (PSNR) over all test images. All approaches were trained by minimizing the VGG loss. On three of the four scenes, our method achieves the best results for all three quality metrics. We believe that the reasoning behind these results are twofold. First, Train, Playground, and Lighthouse, were captured outdoors on a sunny day. The dynamic range is therefore large and a physically-based tonemapper is required to abstract color response and exposure changes. The other approaches, which do not have a learnable tonemapper, encode brightness changes directly into the neural network. Novel views are then synthesized incorrectly based on the training frames instead of the novel view's image settings. The second reason why our systems outperforms the competitors is the input scene refinement during training. As mentioned earlier, we optimize all available parameters including camera pose, camera model, and point position. Hence, we can correct small errors of the initial reconstruction, which further improves sharpness due to a more accurate point projection. These two reasons also explain why the difference in loss of our method to NPBG is small on the *M60* scene. This scene was captured indoors and the initial COLMAP reconstruction is already very precise. The effect of our tonemapper, as well as, the scene refinement is therefore reduced. However, our approach still achieves the best scores compared to the other point-based neural rendering approaches.

#### E. HDR Neural Rendering

Capturing the high-dynamic range (HDR) of real-world scenes with consumer-grade photo equipment is a challenging task. The exposure value (EV) has to be adapted for each shot because the radiance difference of points in shadow regions and points that are directly lit by sunlight is too large for digital camera systems. This means that on the one hand the small EV required for dark regions overexposes bright areas. On the other hand, a large EV enables us to capture the bright regions but eliminates the contrast in shadowed areas. To test the performance of our pipeline on HDR scenes, we fit it to our *roman vessel* dataset. It consists of 742 image captured with a Canon EOS 5D camera with a resolution of  $5760 \times 3840$  pixels. The camera was set to auto-exposure, which adapts the exposure time based on the received brightness at the image center. All other image related settings such as, aperture, ISO-Speed, and white balance are set constant for all frames. In Figure

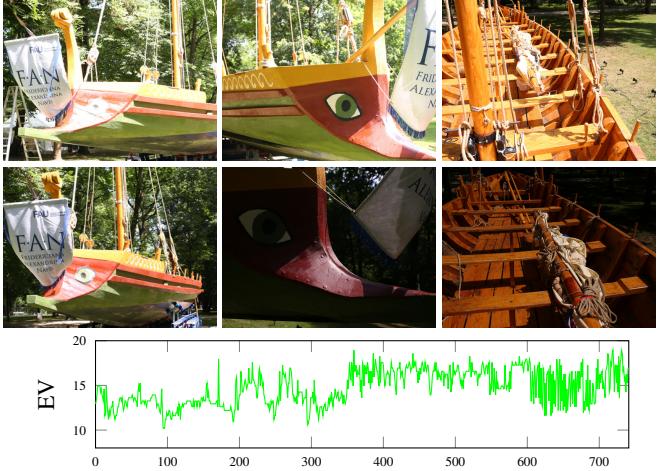


Fig. 10: Sample images of the *roman vessel* dataset. The frames were captured using the auto-exposure setting on the camera. The exposure value (EV) for each image is plotted below the images.

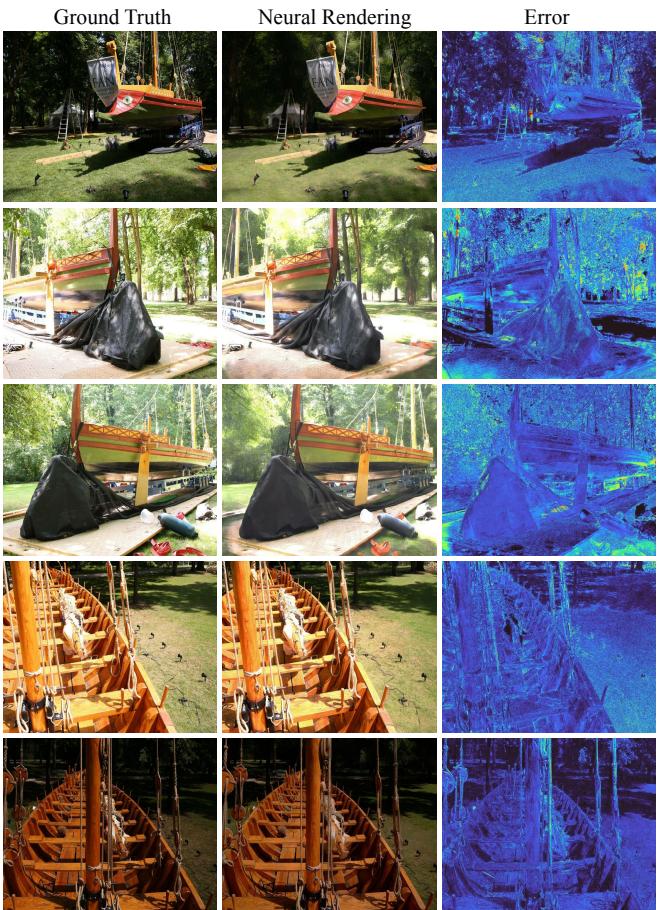


Fig. 11: Novel views synthesized on the roman vessel dataset. The images are rendered using the same exposure value as the reference photograph. In the right column, the per pixel error is visualized.

10, six samples from the dataset are shown. It can be seen that the EV can differ even for similar capture locations resulting in darkened or brightened images. Below the image samples, the exposure value is plotted for every frame. The difference between the smallest and largest EV is 8.7, which corresponds to a factor of  $2^{8.7} = 426.67$ .

To reconstruct a 3D point cloud of the roman vessel we use the photogrammetry software COLMAP [85]. The output are the camera extrinsics, intrinsics, and a point cloud with 53 million points. Using this initial reconstruction we train our rendering pipeline to synthesize an HDR image of the scene, which is then tone-mapped to one of the ground truth images. The exposure value used by the tonemapper is set to the real EV from the EXIF meta data and kept constant during training. To allow a high radiance variation inside the texture, we store it logarithmically and fit it using the ADAM optimizer [88]. The learning rate of the logarithmic texture is reduced by a factor of 10 compared to the linear texture. All other optimization settings are identical to the previous experiment (Section VI-D). We have trained the system for 300 epochs on half resolution. This takes around 12 hours on a single GPU workstation equipped with a NVidia Titan V.

For evaluation purposes, we have removed 20 randomly selected frames from the training set and let our system synthesize them from the estimated pose. The exposure value of the test frames, which is stored in the image metadata, is passed to the tone mapper. Figure 11 shows a few test frames with the ground truth in the left column, the synthesized view in the center, and the per pixel error plot on the right. Small artifacts can be found when inspecting the error map. For example, in row two and three, the paint's reflectance on the hull is not modeled correctly. It is rendered too dark compared to the ground truth as if it is a diffuse material. Besides that, the exterior and interior of the vessel is reconstructed very well with an average LPIPS loss of 0.254. Our system is able to understand HDR scenes and can model camera-specific effects. For example, row four and five show a similar part of the boat captured with different exposure times.

The optimized tonemapper (TM) therefore closely resembles the physical and optical properties of the digital camera that has been used during capturing. However, these images often look unpleasant in HDR scenes because the dynamic range of consumer-grade digital cameras is quite low. An idea to improve visual appearance is to replace the learned TM at inference time by a filmic tone mapper [89]. Filmic tonemappers are state-of-the-art in real-time graphics [90] because they better resemble human perception than digital sensors. The result of replacing the TM at inference time is shown in Figure 12. On the left side you can see the ground truth images. On the right side, the neural renderings with replaced TM are shown. The dark area, for example, in the shadow side of the vessel has significantly more contrast without overexposing the bright wood inside the boat. The colors also look more natural in the rendered images because the filmic TM slightly reduces color saturation.

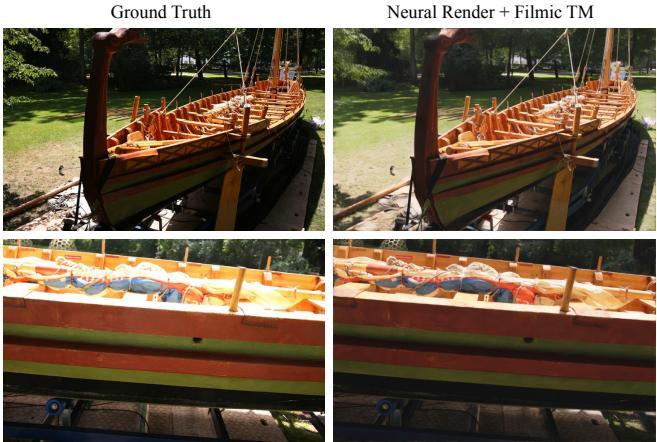


Fig. 12: At inference time, we can replace the learned tone mapper (TM) of the digital camera by a filmic TM. This gives the rendered images a more natural look, because the filmic TM can display a bigger dynamic range.

## VII. LIMITATIONS

In the previous section we have shown that our method can achieve impressive results in various tasks. However, during our experiments we have also found some limitations. One limitation is that due to the vast amount of different parameters, the search of suitable hyper parameters is non-trivial. We have to balance learning rates of the texture color, structural parameters, tone mapping settings, and neural network weights. An extensive grid-search was necessary to find viable settings that work well for all of our scenes. These hyper parameters are now included in the source code distribution and should also fit well to novel scenes.

Another limitation is that the optimization of point position is not stable for moderate to large learning rates. Our pipeline therefore requires a reasonable initial point cloud, for example, by a multi view stereo system or a LiDaR scanner. We believe that this problem is caused by the gradient approximation during rasterization. It works well for camera model and pose optimization because the spatial gradient of thousands of points are averaged in one optimizer step. For the positional point-gradients however, only a single approximate gradient is used to update its coordinates. A very low learning rate is therefore required to average the point gradient over time.

Finally, due to the one-pixel point rendering, holes may appear when the camera is moved too close to an object or the point cloud is very sparse. This happens because the neural network architecture can only fill holes up to a certain size threshold. In our experiments, we diminish this problem by artificially increasing the point density. However, this is not a universally usable solution as in a free view environment the user can still move the camera arbitrarily close to a surface. Future work should be conducted here, for example one could try to dynamically generate new points during magnification that have interpolated neural descriptors.

## VIII. CONCLUSION

We have presented a novel differentiable neural point-based rendering pipeline. Each point is projected into image-space and its neural descriptor is blended into a multi-resolution neural image. This image is processed by a deep convolutional neural network to generate an HDR image of the scene. A differentiable tonemapper converts the HDR image to LDR. We can train this pipeline end-to-end using images, a point cloud, and the camera parameters as input. As all stages are differentiable, the parameters that shall be optimized can be chosen freely, e.g. the camera pose, camera model, or texture color. Additionally, all differentiable camera models can be used because the point rasterizer fills only one pixel per point. In our experiments, we first show that our approach achieves a higher efficiency than state-of-the-art differentiable renderers. After that we present various applications for ADOP. We can, for example, align camera images to a LiDaR point cloud or synthesize novel frames from an initial MVS reconstruction. Our pipeline can also be fitted to HDR scenes captured with large differences in exposure time. By replacing the tonemapper we can then generate images with more natural and pleasant colors than the ground truth. The complete source code of the proposed method will be released on:

<https://github.com/darglein/ADOP>

## REFERENCES

- [1] D. Shin, C. C. Fowlkes, and D. Hoiem, “Pixels, voxels, and views: A study of shape representations for single view 3d object shape prediction,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 3061–3069.
- [2] R. Rostami, F. S. Bashiri, B. Rostami, and Z. Yu, “A survey on data-driven 3d shape descriptors,” in *Computer Graphics Forum*, vol. 38, no. 1. Wiley Online Library, 2019, pp. 356–393.
- [3] L. Kobbelt and M. Botsch, “A survey of point-based techniques in computer graphics,” *Computers & Graphics*, vol. 28, no. 6, pp. 801–814, 2004.
- [4] G. Chaurasia, S. Duchene, O. Sorkine-Hornung, and G. Drettakis, “Depth synthesis and local warps for plausible image-based navigation,” *ACM Transactions on Graphics (TOG)*, vol. 32, no. 3, pp. 1–12, 2013.
- [5] G. Riegler and V. Koltun, “Free view synthesis,” in *European Conference on Computer Vision*. Springer, 2020, pp. 623–640.
- [6] K.-A. Aliev, A. Sevastopolsky, M. Kolos, D. Ulyanov, and V. Lempiitsky, “Neural point-based graphics,” in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXII 16*. Springer, 2020, pp. 696–712.
- [7] M. Levoy and T. Whitted, *The use of points as a display primitive*. Citeseer, 1985.
- [8] M. Zwicker and M. Pauly, “Point-based computer graphics,” *Computers & Graphics*, vol. 28, no. ARTICLE, pp. 799–800, 2004.
- [9] M. Alexa, M. Gross, M. Pauly, H. Pfister, M. Stamminger, and M. Zwicker, “Point-based computer graphics,” in *ACM SIGGRAPH 2004 Course Notes*, 2004, pp. 7–es.
- [10] H. Pfister, M. Zwicker, J. Van Baar, and M. Gross, “Surfels: Surface elements as rendering primitives,” in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, 2000, pp. 335–342.
- [11] M. Zwicker, H. Pfister, J. Van Baar, and M. Gross, “Surface splatting,” in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 2001, pp. 371–378.
- [12] R. Preiner, S. Jeschke, and M. Wimmer, “Auto splats: Dynamic point cloud visualization on the gpu.” in *EGPGV@ Eurographics*, 2012, pp. 139–148.

- [13] G. Bui, T. Le, B. Morago, and Y. Duan, “Point-based rendering enhancement via deep learning,” *The Visual Computer*, vol. 34, no. 6, pp. 829–841, 2018.
- [14] Z. Yang, Y. Chai, D. Anguelov, Y. Zhou, P. Sun, D. Erhan, S. Rafferty, and H. Kretzschmar, “Surfelgan: Synthesizing realistic sensor data for autonomous driving,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.
- [15] J. P. Grossman and W. J. Dally, “Point sample rendering,” in *Eurographics Workshop on Rendering Techniques*. Springer, 1998, pp. 181–192.
- [16] P. Rosenthal and L. Linsen, “Image-space point cloud rendering,” in *Proceedings of Computer Graphics International*. Citeseer, 2008, pp. 136–143.
- [17] R. Pintus, E. Gobbetti, and M. Agus, “Real-time rendering of massive unstructured raw point clouds using screen-space operators,” in *Proceedings of the 12th International conference on Virtual Reality, Archaeology and Cultural Heritage*, 2011, pp. 105–112.
- [18] R. Marroquim, M. Kraus, and P. R. Cavalcanti, “Efficient point-based rendering using image reconstruction,” in *PBG@ Eurographics*, 2007, pp. 101–108.
- [19] M. Botsch, A. Hornung, M. Zwicker, and L. Kobbelt, “High-quality surface splatting on today’s gpus,” in *Proceedings Eurographics/IEEE VGTC Symposium Point-Based Graphics, 2005*. IEEE, 2005, pp. 17–141.
- [20] M. Schütz, B. Kerbl, and M. Wimmer, “Rendering Point Clouds with Compute Shaders and Vertex Order Optimization,” *Computer Graphics Forum*, 2021.
- [21] C. Günther, T. Kanzok, L. Linsen, and P. Rosenthal, “A gpgpu-based pipeline for accelerated rendering of point clouds,” 2013.
- [22] D. Shreiner, B. T. K. O. A. W. Group, et al., *OpenGL programming guide: the official guide to learning OpenGL, versions 3.0 and 3.1*. Pearson Education, 2009.
- [23] F. Pittaluga, S. J. Koppal, S. B. Kang, and S. N. Sinha, “Revealing scenes by inverting structure from motion reconstructions,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.
- [24] M. Meshry, D. B. Goldman, S. Khamis, H. Hoppe, R. Pandey, N. Snavely, and R. Martin-Brualla, “Neural rerendering in the wild,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 6878–6887.
- [25] P. Debevec, Y. Yu, and G. Boshkov, “Efficient view-dependent ibr with projective texture-mapping,” in *EG Rendering Workshop*, vol. 4, no. 11, 1998.
- [26] H. Shum and S. B. Kang, “Review of image-based rendering techniques,” in *Visual Communications and Image Processing 2000*, vol. 4067. International Society for Optics and Photonics, 2000, pp. 2–13.
- [27] P. Hedman, J. Philip, T. Price, J.-M. Frahm, G. Drettakis, and G. Brostow, “Deep blending for free-viewpoint image-based rendering,” *ACM Transactions on Graphics (TOG)*, vol. 37, no. 6, pp. 1–15, 2018.
- [28] G. Riegler and V. Koltun, “Stable view synthesis,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 12216–12225.
- [29] J. Thies, M. Zollhöfer, C. Theobalt, M. Stamminger, and M. Nießner, “Image-guided neural object rendering,” in *8th International Conference on Learning Representations*. OpenReview.net, 2020.
- [30] P. P. Srinivasan, R. Tucker, J. T. Barron, R. Ramamoorthi, R. Ng, and N. Snavely, “Pushing the boundaries of view extrapolation with multiplane images,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 175–184.
- [31] R. Tucker and N. Snavely, “Single-view view synthesis with multiplane images,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 551–560.
- [32] T. Zhou, R. Tucker, J. Flynn, G. Fyffe, and N. Snavely, “Stereo magnification: Learning view synthesis using multiplane images,” *arXiv preprint arXiv:1805.09817*, 2018.
- [33] B. Mildenhall, P. P. Srinivasan, R. Ortiz-Cayon, N. K. Kalantari, R. Ramamoorthi, R. Ng, and A. Kar, “Local light field fusion: Practical view synthesis with prescriptive sampling guidelines,” *ACM Transactions on Graphics (TOG)*, vol. 38, no. 4, pp. 1–14, 2019.
- [34] Y. Ganin, D. Kononenko, D. Sungatullina, and V. Lempitsky, “Deepwarp: Photorealistic image resynthesis for gaze manipulation,” in *European conference on computer vision*. Springer, 2016, pp. 311–326.
- [35] T. Zhou, S. Tulsiani, W. Sun, J. Malik, and A. A. Efros, “View synthesis by appearance flow,” in *European conference on computer vision*. Springer, 2016, pp. 286–301.
- [36] J. Flynn, I. Neulander, J. Philbin, and N. Snavely, “Deepstereo: Learning to predict new views from the world’s imagery,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 5515–5524.
- [37] J. Thies, M. Zollhöfer, and M. Nießner, “Deferred neural rendering: Image synthesis using neural textures,” *ACM Transactions on Graphics (TOG)*, vol. 38, no. 4, pp. 1–12, 2019.
- [38] T. Nguyen-Phuoc, C. Li, S. Balaban, and Y.-L. Yang, “Rendernet: A deep convolutional network for differentiable rendering from 3d shapes,” *arXiv preprint arXiv:1806.06575*, 2018.
- [39] V. Sitzmann, J. Thies, F. Heide, M. Nießner, G. Wetzstein, and M. Zollhöfer, “Deepvoxels: Learning persistent 3d feature embeddings,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2437–2446.
- [40] J.-Y. Zhu, Z. Zhang, C. Zhang, J. Wu, A. Torralba, J. B. Tenenbaum, and W. T. Freeman, “Visual object networks: Image generation with disentangled 3d representation,” *arXiv preprint arXiv:1812.02725*, 2018.
- [41] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, “Deepsdf: Learning continuous signed distance functions for shape representation,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 165–174.
- [42] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger, “Occupancy networks: Learning 3d reconstruction in function space,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4460–4470.
- [43] P. Dai, Y. Zhang, Z. Li, S. Liu, and B. Zeng, “Neural point cloud rendering via multi-plane projection,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.
- [44] H. Kato, D. Beker, M. Morariu, T. Ando, T. Matsuoka, W. Kehl, and A. Gaidon, “Differentiable rendering: A survey,” *arXiv preprint arXiv:2006.12057*, 2020.
- [45] M. L. Loper and M. J. Black, “Opendr: An approximate differentiable renderer,” in *European Conference on Computer Vision*. Springer, 2014, pp. 154–169.
- [46] H. Kato, Y. Ushiku, and T. Harada, “Neural 3d mesh renderer,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 3907–3916.
- [47] H. Kato and T. Harada, “Learning view priors for single-view 3d reconstruction,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9778–9787.
- [48] K. Genova, F. Cole, A. Maschinot, A. Sarna, D. Vlasic, and W. T. Freeman, “Unsupervised training for 3d morphable model regression,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8377–8386.
- [49] H. Rhodin, N. Robertini, C. Richardt, H.-P. Seidel, and C. Theobalt, “A versatile scene model with differentiable visibility applied to generative pose estimation,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 765–773.
- [50] S. Liu, T. Li, W. Chen, and H. Li, “Soft rasterizer: A differentiable renderer for image-based 3d reasoning,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 7708–7717.
- [51] W. Chen, H. Ling, J. Gao, E. Smith, J. Lehtinen, A. Jacobson, and S. Fidler, “Learning to predict 3d objects with an interpolation-based differentiable renderer,” *Advances in Neural Information Processing Systems*, vol. 32, pp. 9609–9619, 2019.
- [52] S. Tulsiani, T. Zhou, A. A. Efros, and J. Malik, “Multi-view supervision for single-view reconstruction via differentiable ray consistency,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2626–2634.
- [53] P. Henzler, N. J. Mitra, and T. Ritschel, “Escaping plato’s cave: 3d shape from adversarial rendering,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 9984–9993.
- [54] S. Lombardi, T. Simon, J. Saragih, G. Schwartz, A. Lehrmann, and Y. Sheikh, “Neural volumes: Learning dynamic renderable volumes from images,” *arXiv preprint arXiv:1906.07751*, 2019.
- [55] Y. Jiang, D. Ji, Z. Han, and M. Zwicker, “Sdfdiff: Differentiable rendering of signed distance fields for 3d shape optimization,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1251–1261.

- [56] S. Liu, S. Saito, W. Chen, and H. Li, “Learning to infer implicit surfaces without 3d supervision,” *arXiv preprint arXiv:1911.00767*, 2019.
- [57] M. Niemeyer, L. Mescheder, M. Oechsle, and A. Geiger, “Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 3504–3515.
- [58] S. Liu, Y. Zhang, S. Peng, B. Shi, M. Pollefeys, and Z. Cui, “Dist: Rendering deep implicit signed distance function with differentiable sphere tracing,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 2019–2028.
- [59] S. Zakharov, W. Kehl, A. Bhargava, and A. Gaidon, “Autolabeling 3d objects with differentiable rendering of sdf shape priors,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 12224–12233.
- [60] K. Zhang, G. Riegler, N. Snavely, and V. Koltun, “Nerf++: Analyzing and improving neural radiance fields,” *arXiv preprint arXiv:2010.07492*, 2020.
- [61] A. Yu, V. Ye, M. Tancik, and A. Kanazawa, “pixelnerf: Neural radiance fields from one or few images,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 4578–4587.
- [62] R. Martin-Brualla, N. Radwan, M. S. Sajjadi, J. T. Barron, A. Dosovitskiy, and D. Duckworth, “Nerf in the wild: Neural radiance fields for unconstrained photo collections,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 7210–7219.
- [63] V. Sitzmann, J. Martel, A. Bergman, D. Lindell, and G. Wetzstein, “Implicit neural representations with periodic activation functions,” *Advances in Neural Information Processing Systems*, vol. 33, 2020.
- [64] C.-H. Lin, C. Kong, and S. Lucey, “Learning efficient point cloud generation for dense 3d object reconstruction,” in *proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.
- [65] Z. Han, C. Chen, Y.-S. Liu, and M. Zwicker, “Drwr: A differentiable renderer without rendering for unsupervised 3d structure learning from silhouette images,” *arXiv preprint arXiv:2007.06127*, 2020.
- [66] W. Yifan, F. Serena, S. Wu, C. Öztireli, and O. Sorkine-Hornung, “Differentiable surface splatting for point-based geometry processing,” *ACM Transactions on Graphics (TOG)*, vol. 38, no. 6, pp. 1–14, 2019.
- [67] O. Wiles, G. Gkioxari, R. Szeliski, and J. Johnson, “Synsin: End-to-end view synthesis from a single image,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 7467–7477.
- [68] G. Kopanas, J. Philip, T. Leimkühler, and G. Drettakis, “Point-based neural rendering with per-view optimization,” in *Computer Graphics Forum*, vol. 40, no. 4. Wiley Online Library, 2021, pp. 29–43.
- [69] E. Insafutdinov and A. Dosovitskiy, “Unsupervised learning of shape and pose with differentiable point clouds,” *arXiv preprint arXiv:1810.09381*, 2018.
- [70] C. Lassner and M. Zollhofer, “Pulsar: Efficient sphere-based neural rendering,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021, pp. 1440–1449.
- [71] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. S. Huang, “Free-form image inpainting with gated convolution,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 4471–4480.
- [72] E. Enderton, E. Sintorn, P. Shirley, and D. Luebke, “Stochastic transparency,” *IEEE transactions on visualization and computer graphics*, vol. 17, no. 8, pp. 1036–1047, 2010.
- [73] J. Engel, V. Koltun, and D. Cremers, “Direct sparse odometry,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 3, pp. 611–625, 2017.
- [74] R. Mur-Artal and J. D. Tardós, “Orb-slam2: An open-source slam system for monocular, stereo, and rgbd cameras,” *IEEE transactions on robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.
- [75] D. Rückert and M. Stamminger, “Snake-slam: Efficient global visual inertial slam using decoupled nonlinear optimization,” in *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2021, pp. 219–228.
- [76] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, “Caffe: Convolutional architecture for fast feature embedding,” in *Proceedings of the 22nd ACM international conference on Multimedia*, 2014, pp. 675–678.
- [77] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al., “Pytorch: An imperative style, high-performance deep learning library,” *Advances in neural information processing systems*, vol. 32, pp. 8026–8037, 2019.
- [78] D. B. Goldman, “Vignette and exposure calibration and compensation,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 12, pp. 2276–2288, 2010.
- [79] M. D. Grossberg and S. K. Nayar, “Determining the camera response from images: What is knowable?” *IEEE Transactions on pattern analysis and machine intelligence*, vol. 25, no. 11, pp. 1455–1467, 2003.
- [80] P. E. Debevec and J. Malik, “Recovering high dynamic range radiance maps from photographs,” in *ACM SIGGRAPH 2008 classes*, 2008, pp. 1–10.
- [81] B. Xu, N. Wang, T. Chen, and M. Li, “Empirical evaluation of rectified activations in convolutional network,” *arXiv preprint arXiv:1505.00853*, 2015.
- [82] N. Ravi, J. Reizenstein, D. Novotny, T. Gordon, W.-Y. Lo, J. Johnson, and G. Gkioxari, “Accelerating 3d deep learning with pytorch3d,” *arXiv preprint arXiv:2007.08501*, 2020.
- [83] A. Knapitsch, J. Park, Q.-Y. Zhou, and V. Koltun, “Tanks and temples: Benchmarking large-scale scene reconstruction,” *ACM Transactions on Graphics (ToG)*, vol. 36, no. 4, pp. 1–13, 2017.
- [84] NavVis, “Vlx reality capture,” 2021.
- [85] J. L. Schonberger and J.-M. Frahm, “Structure-from-motion revisited,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4104–4113.
- [86] J. Johnson, A. Alahi, and L. Fei-Fei, “Perceptual losses for real-time style transfer and super-resolution,” in *European conference on computer vision*. Springer, 2016, pp. 694–711.
- [87] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, “The unreasonable effectiveness of deep features as a perceptual metric,” in *CVPR*, 2018.
- [88] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.
- [89] J. Hable, “Filmic tonemapping for real-time rendering,” in *Siggraph 2010 Color Course*, 2010.
- [90] Epic Games, “Unreal engine,” 2019. [Online]. Available: <https://www.unrealengine.com>