Neural Point-Based Graphics

Kara-Ali Aliev[1], Artem Sevastopolsky[1,2], Maria Kolos[1,2], Dmitry Ulyanov[3], and Victor Lempitsky[1,2]

[1] Samsung AI Center
[2] Skolkovo Institute of Science and Technology
[3] In3D.io

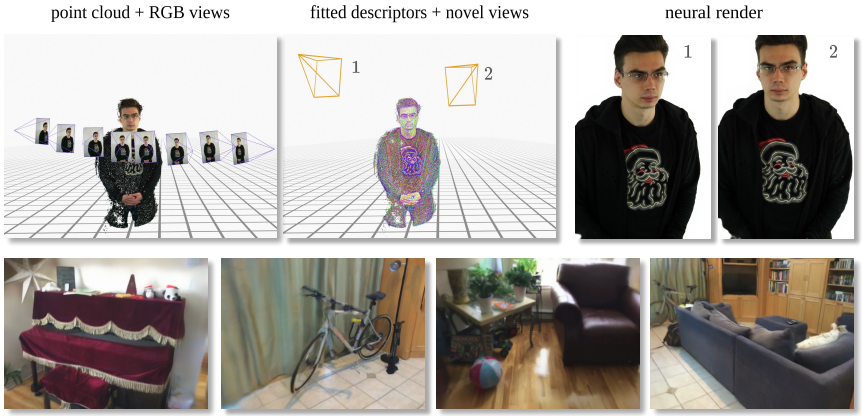


Fig. 1: Given a set of RGB views and a point cloud (top-left), our approach fits a neural descriptor to each point (top-middle), after which new views of a scene can be rendered (top-right). The method works for a variety of scenes including 3D portraits (top) and interiors (bottom).

Abstract. We present a new point-based approach for modeling the appearance of real scenes. The approach uses a raw point cloud as the geometric representation of a scene, and augments each point with a learnable neural descriptor that encodes local geometry and appearance. A deep rendering network is learned in parallel with the descriptors, so that new views of the scene can be obtained by passing the rasterizations of a point cloud from new viewpoints through this network. The input rasterizations use the learned descriptors as point pseudo-colors. We show that the proposed approach can be used for modeling complex scenes and obtaining their photorealistic views, while avoiding explicit surface estimation and meshing. In particular, compelling results are obtained for scenes scanned using hand-held commodity RGB-D sensors as well as standard RGB cameras even in the presence of objects that are challenging for standard mesh-based modeling.

Keywords: Image-based rendering, scene modeling, neural rendering, convolutional networks

1 Introduction

Creating virtual models of real scenes usually involves a lengthy pipeline of operations. Such modeling usually starts with a scanning process, where the photometric properties are captured using camera images and the raw scene geometry is captured using depth scanners or dense stereo matching. The latter process usually provides noisy and incomplete point cloud that needs to be further processed by applying certain surface reconstruction and meshing approaches. Given the mesh, the texturing and material estimation processes determine the photometric properties of surface fragments and store them in the form of 2D parameterized maps, such as texture maps [1], bump maps [2], view-dependent textures [3], surface lightfields [4]. Finally, generating photorealistic views of the modeled scene involves computationally-heavy rendering process such as ray tracing and/or radiance transfer estimation.

The outlined pipeline has been developed and polished by the computer graphics researchers and practitioners for decades. Under controlled settings, this pipeline yields highly realistic results. Yet several of its stages (and, consequently, the entire pipeline) remain brittle. Multiple streams of work aim to simplify the entire pipeline by eliminating some of its stages. Thus, image-based rendering techniques [5–8] aim to obtain photorealistic views by warping the original camera images using certain (oftentimes very coarse) approximations of scene geometry. Alternatively, point-based graphics [9–12] discards the estimation of the surface mesh and use a collection of points or unconnected disks (surfels) to model the geometry. More recently, deep rendering approaches [13–17] aim to replace physics-based rendering with a generative neural network, so that some of the mistakes of the modeling pipeline can be rectified by the rendering network.

Here, we present a system that eliminates many of the steps of the classical pipeline. It combines the ideas of image-based rendering, point-based graphics, and neural rendering into a simple approach. The approach uses the raw point-cloud as a scene geometry representation, thus eliminating the need for surface estimation and meshing. Similarly to other neural rendering approaches, it also uses a deep convolutional neural network to generate photorealistic renderings from new viewpoints. The realism of the rendering is facilitated by the estimation of latent vectors (neural descriptors) that describe both the geometric and the photometric properties of the data. These descriptors are learned directly from data, and such learning happens in coordination with the learning of the rendering network (see Fig. 2).

We show that our approach is capable of modeling and rendering scenes that are captured by hand-held RGBD cameras as well as simple RGB streams (from which point clouds are reconstructed via structure-from-motion or similar techniques). A number of comparisons are performed with ablations and competing approaches, demonstrating the capabilities, advantages, and limitations of the new method. In general, our results suggest that given the power of modern deep networks, the simplest 3D primitives (i.e. 3D points) might represent sufficient and most suitable geometric proxies for neural rendering in many cases.

2 Related work

Our approach brings together several lines of works from computer graphics, computer vision, and deep learning communities, of which only a small subset can be reviewed due to space limitations.

Point-based graphics. Using points as the modeling primitives for rendering (point-based graphics) was proposed in [9, 10] and have been in active development in the 2000s [18, 19, 11, 12]. The best results are obtained when each point is replaced with an oriented flat circular disk (a surfel), whereas the orientations and the radii of such disks can be estimated from the point cloud data. Multiple overlapping surfels are then rasterized and linearly combined using splatting operation [18]. More recently, [16] has proposed to replace linear splatting with deep convolutional network. Similarly, a rendering network is used to turn point cloud rasterizations into realistic views by [20], which rasterizes each point using its color, depth, and its semantic label. Alternatively, [21] uses a relatively sparse point cloud such as obtained by structure-and-motion reconstruction, and rasterizes the color and the high-dimensional SIFT [22] descriptor for each point.

In our work, we follow the point-based graphics paradigm as we represent the geometry of a scene using its point cloud. However, we do not use the surface orientation, or suitable disk radii, or, in fact, even color, explicitly during rasterization. Instead, we keep a 3D point as our modeling primitive and encode all local parameters of the surface (both photometric and geometric) within neural descriptors that are learned from data. We compare this strategy with the approach of [20] in the experiments.

Deep image based rendering. Recent years have also seen active convergence of image-based rendering and deep learning. A number of works combine warping of preexisting photographs and the use of neural networks to combine warped images and/or to post-process the warping result. The warping can be estimated by stereo matching [23]. Estimating warping fields from a single input image and a low-dimensional parameter specifying a certain motion from a low-parametric family is also possible [24, 25]. Other works perform warping using coarse mesh geometry, which can be obtained through multi-view stereo [17, 26] or volumetric RGBD fusion [27]. Alternatively, some methods avoid explicit warping and instead use some form of plenoptic function estimation and parameterization using neural networks. Thus, [15] proposes network-parameterized deep version of surface lightfields. The approach [28] learns neural parameterization of the plenoptic function in the form of low-dimensional descriptors situated at the nodes of a regular voxel grid and a rendering function that turns the reprojection of such desriptors to the new view into an RGB image.

Arguably most related to ours is the deferred neural rendering (DNR) system [29]. They propose to learn *neural textures* encoding the point plenoptic function at different surface points alongside the neural rendering convolutional network. Our approach is similar to [29], as it also learns neural descriptors of surface

elements jointly with the rendering network. The difference is that our approach uses point-based geometry representation and thus avoids the need for surface estimation and meshing. We perform extensive comparison to [29], and discuss relative pros and cons of the two approaches.

Texture optimization methods. Our work is also related to a class of methods that perform optimization of the color texture for mesh-based models [30–32]. Similarly to the optimization of point descriptors in our method, [30–32] optimize texture parameters using objectives that go beyond simple pixelwise color difference minimization. Likewise, one of the baselines in our comparisons performs mesh texture optimization using perceptual loss [33]. While improving significantly over simple texturing methods, texture-based optimization approaches still require the mesh to approximate the scene geometry reasonably well and may not produce plausible models when mesh reconstruction fails to recover parts of the scene.

3 Methods

Below, we explain the details of our system. First, we explain how the rendering of a new view is performed given a point cloud with learned neural descriptors and a learned rendering network. Afterwards, we discuss the process that creates a neural model of a new scene.

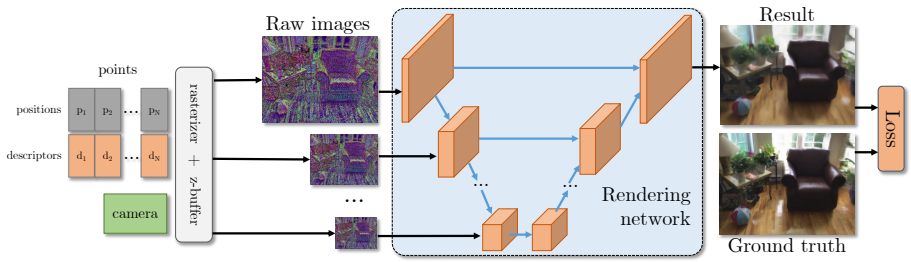


Fig. 2: An overview of our system. Given the point cloud $\mathbf{P}$ with neural descriptors $\mathbf{D}$ and camera parameters C, we rasterize the points with z-buffer at several resolutions, using descriptors as pseudo-colors. We then pass the rasterizations through the U-net-like rendering network to obtain the resulting image. Our model is fit to new scene(s) by optimizing the parameters of the rendering network and the neural descriptors by backpropagating the perceptual loss function.

3.1 Rendering

Assume that a point cloud $\mathbf{P} = \{p_1, p_2, \ldots, p_N\}$ with M-dimensional neural descriptors attached to each point $\mathbf{D} = \{d_1, d_2, \ldots, d_N\}$ is given, and its render-

ing from a new view characterized by a camera C (including both extrinsic and intrinsic parameters) needs to be obtained. In particular, assume that the target image has $W \times H$-sized pixel grid, and that its viewpoint is located in point p_0.

The rendering process first projects the points onto the target view, using descriptors as pseudo-colors, and then uses the rendering network to transform the pseudo-color image into a photorealistic RGB image. More formally, we create an M-channel *raw image* $S(\mathbf{P}, \mathbf{D}, C)$ of size $W \times H$, and for each point p_i which projects to (x, y) we set $S(\mathbf{P}, \mathbf{D}, C)[[x], [y]] = d_i$ (where $[a]$ denotes a nearest integer of $a \in \mathbb{R}$). As many points may project onto the same pixel, we use z-buffer to remove occluded points. The lack of topological information in the point cloud, however, results in hole-prone representation, such that the points from the occluded surfaces and the background can be seen through the front surface (so-called *bleeding* problem). This issue is traditionally addressed through splatting, i.e. replacing each point with a 3D disk with a radius to be estimated from data and projecting the resulting elliptic footprint of the point onto an image. We have proposed an alternative rendering scheme that does not rely on the choice of the disk radius.

Progressive rendering. Rather than performing splatting, we resort to multi-scale (progressive) rendering. We thus render a point cloud T times onto a pyramid of canvases of different spatial resolutions. In particular, we obtain a sequence of images $S[1], S[2] \dots S[T]$, where the i-th image has the size of $\frac{W}{2^i} \times \frac{H}{2^i}$, by performing a simple point cloud projection described above. As a result, the highest resolution raw image $S[1]$ contains the largest amount of details, but also suffers from strong surface bleeding. The lowest resolution image $S[T]$ has coarse geometric detailization, but has the least surface bleeding, while the intermediate raw images $S[2], \dots, S[T-1]$ achieve different detailization-bleeding tradeoffs.

Finally, we use a *rendering network* $\mathcal{R}_\theta$ with learnable parameters θ to map all the raw images into a three-channel RGB image I:

$$I(\mathbf{P}, \mathbf{D}, C, \theta) = \mathcal{R}_\theta(\, S[1](\mathbf{P}, \mathbf{D}, C),\, \dots,\, S[T](\mathbf{P}, \mathbf{D}, C)\,)\,. \tag{1}$$

The rendering network in our case is based on a popular convolutional U-Net architecture [34] with gated convolutions [35] for better handling of a potentially sparse input. Compared to the traditional U-Net, the rendering network architecture is augmented to integrate the information from all raw images (see Fig. 2). In particular, the encoder part of the U-Net contains several downsampling layers interleaved with convolutions and non-linearities. We then concatenate the raw image $S[i]$ to the first block of the U-Net encoder at the respective resolution. Such progressive (coarse-to-fine) mechanism is reminiscent to texture mipmapping [36] as well as many other coarse-to-fine/varying level of details rendering algorithms in computer graphics. In our case, the rendering network provides the mechanism for implicit level of detail selection.

The rasterization of images $S[1], \dots, S[T]$ is implemented via OpenGL. In particular U-net network has five down- and up-sampling layers. Unless noted otherwise, we set the dimensionality of descriptors to eight (M=8).

3.2 Model creation

We now describe the fitting process in our system. We assume that during fitting K different scenes are available. For the k-th scene the point cloud $\mathbf{P}^k$ as well as the set of L_k training ground truth RGB images $\mathbf{I}^k = \{I^{k,1}, I^{k,2}, \dots I^{k,L_k}\}$ with known camera parameters $\{C^{k,1}, C^{k,2}, \dots C^{k,L_k}\}$ are given. Our fitting objective $\mathcal{L}$ then corresponds to the mismatch between the rendered and the ground truth RGB images:

$$\mathcal{L}(\theta, \mathbf{D}^1, \mathbf{D}^2, \dots, \mathbf{D}^K) = \sum_{k=1}^{K} \sum_{l=1}^{L_k} \Delta \left(\mathcal{R}_\theta \left(\{S[i](\mathbf{P}^k, \mathbf{D}^k, C^{k,l})\}_{i=1}^{T} \right), \; I^{k,l} \right) , \quad (2)$$

where $\mathbf{D}^k$ denotes the set of neural descriptors for the point cloud of the k-th scene, and Δ denotes the mismatch between the two images (the ground truth and the rendered one). In our implementation, we use the perceptual loss [37, 33] that computes the mismatch between the activations of a pretrained VGG network [38].

The fitting is performed by optimizing the loss (2) over both the parameters θ of the rendering network **and** the neural descriptors $\{\mathbf{D}^1, \mathbf{D}^2, \dots, \mathbf{D}^K\}$ of points in the training set of scenes. Thus, our approach learns the neural descriptors directly from data. Optimization is performed by the ADAM algorithm [39]. Note, that the neural descriptors are updated via backpropagation through (1) of the loss derivatives w.r.t. $S(\mathbf{P}, \mathbf{D}, C)$ onto d_i.

Our system is amenable for various kinds of transfer/incremental learning. Thus, while we can perform fitting on a single scene, the results for new viewpoints tend to be better when the rendering network is fitted to multiple scenes of a similar kind. In the experimental validation, unless noted otherwise, we fit the rendering network in a two stage process. We first *pretrain* the rendering network on a family of scenes of a certain kind. Secondly, we fit (*fine-tune*) the rendering network to a new scene. At this stage, the learning process (2) starts with zero descriptor values for the new scene and with weights of the pretrained rendering network.

4 Experiments

Datasets. To demonstrate the versatility of the approach, we evaluate it on several types of real scenes. Thus, we consider three sources of data for our experiments. First, we take RGBD streams from the ScanNet dataset [40] of room-scale scenes scanned with a structured-light RGBD sensor[4]. Second, we consider RGB image datasets of still standing people captured by a mirrorless camera with high resolution (the views capture roughly 180 degrees). Finally, we consider two more scenes corresponding to two objects captured by a smartphone camera. 360° camera flights of two selected objects (a potted plant and a small figurine) of a different kind captured from a circle around the object.

[4] `https://structure.io/`

For all experiments, as per the two-stage learning scheme described in Sec. 3.2, we split the dataset into three parts, unless noted otherwise: **pretraining part**, **fine-tuning part**, and **holdout part**. The pretraining part contains a set of scenes, to which we fit the rendering network (alongside point descriptors). The fine-tuning part contains a subset of frames of a new scene, which are fitted by the model creation process started with pretrained weights of the rendering network. The holdout part contains additional views of the new scene that are not shown during fitting and are used to evaluate the performance.

For the ScanNet scenes, we use the provided registration data obtained with the BundleFusion [41] method. We also use the mesh geometry computed by BundleFusion in mesh-based baselines. Given the registration data, point clouds are obtained by joining together the 3D points from all RGBD frames and using volumetric subsampling (with the grid step 1 cm) resulting in the point clouds containing few million points per scene. We pretrain rendering networks on a set of 100 ScanNet scenes. In the evaluation, we use two ScanNet scenes '**Studio**' (scene 0), which has 5578 frames, and '**LivingRoom**' (scene 24), which has 3300 frames (both scenes are not from the pretraining part). In each case, we use every 100^{th} frame in the trajectory for holdout and, prior to the fitting, we remove 20 preceding and 20 following frames for each of the holdout frames from the fine-tuning part to ensure that holdout views are sufficiently distinct.

For the camera-captured scenes of humans, we collected 123 sequences of 41 distinct people, each in 3 different poses, by a Sony a7-III mirrorless camera. Each person was asked to stand still (like a mannequin) against a white wall for 30–45 seconds and was photographed by a slowly moving camera along a continuous trajectory, covering the frontal half of a body, head, and hair from several angles. We then remove (whiten) the backround using the method [42], and the result is processed by the Agisoft Metashape [43] package that provides camera registration, the point cloud, and the mesh by running proprietary structure-from-motion and dense multi-view stereo methods. Each sequence contains 150-200 ten megapixel frames with high amount of fine details and varying clothing and hair style. The pretraining set has 102 sequences of 38 individuals, and three scenes of three different individuals were left for validation. Each of the validation scenes is split into fine-tuning (90% of frames) and holdout (10% of frames) sets randomly.

In addition, we used a smartphone (Galaxy S10e) to capture 360° sequences of two scenes containing an **Owl** figurine (61 images) and a potted **Plant** (92 images). All frames of both scenes were segmented manually via tools from Adobe Photoshop CC software package. We split fine-tuning and holdout parts in the same manner as in People dataset.

Comparison with state-of-the-art. We compare our method to several neural rendering approaches on the evaluation scenes. Most of these approaches have a rendering network similar to our method, which takes an intermediate representation and then is trained to output the final RGB image. Unless stated otherwise, we use the network described in Section 3. It is lightweight with

Fig. 3: Comparative results on the 'Studio' dataset (from [40]). We show the textured mesh, the colored point cloud, the results of three neural rendering systems (including ours), and the ground truth. Our system can successfully reproduce details that pose challenge for meshing, such as the wheel of the bicycle.

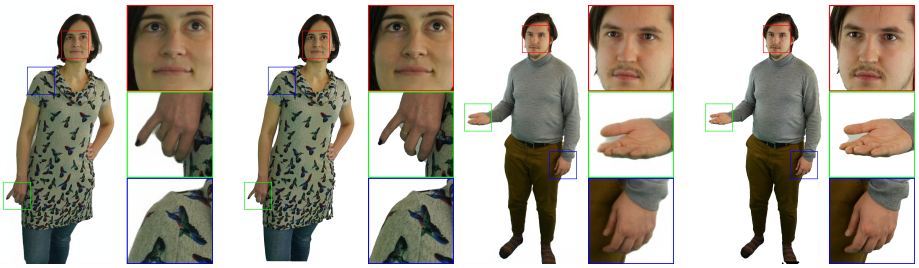
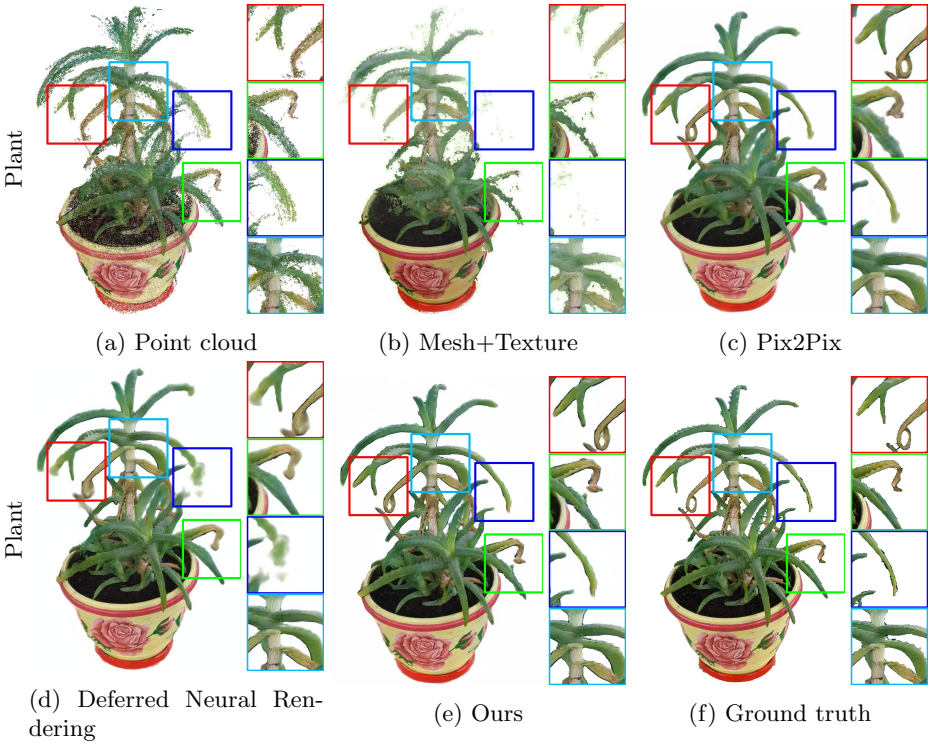


Fig. 4: Results on the holdout frames from the 'Person 1' and 'Person 2' scenes. Our approach successfully transfers fine details to new views.

1.96M parameters and allows us to render real-time, taking 62ms on GeForce RTX 2080 Ti to render a FullHD image. For all the approaches we use the same train time augmentations, particularly random 512x512 crops and 2x zoom-in and zoom-out.

The following methods were compared:

- **Ours.** During learning, we both optimize the neural descriptors and fine-tune the rendering network on the fine-tuning part.
- **Pix2Pix.** In this variant, we evaluate an ablation of our point-based system without neural descriptors. Here, we learn the rendering network that maps the point cloud rasterized in the same way as in our method. However, instead of neural descriptors, we use the color of the point (taken from the original RGBD scan/RGB image). The rendering network is then trained with the same loss as ours.
- **Pix2Pix (slow).** We observed that our method features neural descriptors which increases the number of parameters to be learned. For the sake of fair comparison, we therefore evaluated the variant of Pix2Pix with the rendering



(a) Point cloud (b) Mesh+Texture (c) Pix2Pix

(d) Deferred Neural Rendering (e) Ours (f) Ground truth

Fig. 5: Comparative results on the holdout frame from the 'Plant' scene. Our method better preserves thin parts of the scene.

network with doubled number of channels in all intermediate layers (resulting in ∼4x parameters and FLOPs).

– **Neural Rerendering in the Wild.** Following [20], we have augmented the input of the Pix2Pix method with the segmentation labels (one-hot format) and depth values. We have not used the appearance modeling from [20], since lightning was consistent within each dataset.
– **Neural Rerendering in the Wild (slow).** Same as previous, but twice larger number of channels in the rendering network. Due to the need to have meaningful segmentation labels, we have considered this and the previous methods only for ScanNet comparisons, where such labels are provided with the dataset.
– **Mesh+Texture.** In this baseline, given the mesh of the scene obtained with either BundleFusion or Metashape (depending on the dataset used), we learn the texture via backpropagation of the same loss as used in our method through the texture mapping process onto the texture map. This results in a "classical" scene representation of the textured mesh.
– **Deferred Neural Rendering (DNR).** We implemented the mesh-based approach described in [29]. As suggested, we use hierarchical neural textures

Table 1: Comparison with the state-of-the-art for all considered hold-out scenes from various sources: two scenes from ScanNet, two people captured by a professional camera, and two objects photographed by a smartphone. We assess all methods with respect to widely used perceptual metrics correlated with visual similarity of predicted and ground truth images (LPIPS, FID) and to VGG loss used in our experiments.

	ScanNet - LivingRoom			ScanNet - Studio		
Method	VGG ↓	LPIPS ↓	FID ↓	VGG ↓	LPIPS ↓	FID ↓
Pix2Pix	751.04	0.564	192.82	633.30	0.535	127.49
Pix2Pix (slow)	741.09	0.547	187.92	619.04	0.509	109.46
Neural Rerendering	751.52	0.580	206.90	634.93	0.529	119.16
Neural Rerendering (slow)	739.82	0.542	186.27	620.98	0.507	108.12
Textured mesh	791.26	0.535	152.02	690.67	0.540	97.95
Deferred Neural Rendering	**725.23**	0.492	**129.33**	603.63	0.484	84.92
Ours (splatting)	726.50	**0.485**	139.90	**591.87**	**0.470**	81.94
Ours	727.38	**0.488**	138.87	**595.24**	**0.472**	**76.73**

	People - Person 1			People - Person 2		
Method	VGG ↓	LPIPS ↓	FID ↓	VGG ↓	LPIPS ↓	FID ↓
Pix2Pix	209.16	0.1016	51.38	186.89	0.1642	114.93
Pix2Pix (slow)	204.45	0.0975	47.14	179.99	0.1566	102.62
Textured mesh	**155.37**	0.0698	60.62	163.73	0.1404	96.20
Deferred Neural Rendering	184.86	0.0659	34.41	163.13	0.1298	78.70
Ours (splatting)	186.06	0.0664	44.63	162.56	0.1174	80.60
Ours	181.11	**0.0602**	**32.63**	**161.18**	**0.1131**	**77.92**

	Plant			Owl		
Method	VGG ↓	LPIPS ↓	FID ↓	VGG ↓	LPIPS ↓	FID ↓
Pix2Pix	85.47	0.0443	52.95	34.30	0.0158	124.63
Pix2Pix (slow)	82.81	0.0422	48.89	32.93	0.0141	101.65
Textured mesh	101.56	0.0484	95.60	36.58	0.0145	141.66
Deferred Neural Rendering	77.55	0.0377	49.61	**28.12**	**0.0096**	**54.14**
Ours	**75.08**	**0.0373**	**41.67**	29.69	0.0103	78.55

with five scales (maximum 2048×2048) each having eight channels (same as the descriptor size M in our method). The rendering network is then trained with the same loss as ours. Generally, this method can be seen as the analog of our method with point-based geometric proxy replaced with mesh-based proxy.

We compare the methods on ScanNet (two scenes following pretraining on 100 other scenes), on People (two people following pretraining on 102 scenes of 38 other people), as well as on 'Owl' and 'Plant' scenes (following the pretraining on People). The quantitative results of the comparison are shown in Table 1. All comparisons are measured on the holdout parts, for which we compare the obtained and the ground truth RGB images. We stress that we keep the holdout viewpoints sufficiently dissimilar from the viewpoints of images used for fine-tuning. For all experiments *nearest train view* is defined as follows. Given a novel view, we sort train views by angle deviation from the novel view, then leave top

5% closest by angle and pick the view closest by distance. Angle proximity is more critical since we use zoom augmentation in training which compensate distance dissimilarity.

We report the value of the loss on the holdout part (VGG) as well as two common metrics (Learned Perceptial Similarity – *LPIPS* [44] and Frechet Inception Distance – *FID* [45]). We also show qualitative comparisons on the holdout set frames in Figures 3–5, where we also show the point cloud, and renderings from completely different viewpoints in Figures 6–7. Further comparisons can be found in **Supplementary video**.

Generally, both the quantitative and the qualitative comparison reveals the advantage of using *learnable neural descriptors for neural rendering*. Indeed, with the only exception (VGG metric on Person 1), Deferred Neural Rendering and Neural Point-Based Graphics, which use such learnable descriptors, outperform other methods, sometimes by a rather big margin.

The relative performance of the two methods that use learnable neural descriptors (ours and DNR) varies across metrics and scenes. Generally, our method performs better on scenes and parts of the scene, where meshing is problematic due to e.g. thin objects such as 'Studio' (Fig. 3) and 'Plant' (Fig. 5) scenes. Conversely, DNR has advantage whenever a good mesh can be reconstructed.

In support of this observations, user study via Yandex.Toloka web platform was conducted for ScanNet 'Studio' scene and 'Plant' scene. As for 'Studio', we took 300 half image size crops in total uniformly sampled from all holdout images. Labelers were asked to evaluate which picture is closer to a given ground truth crop — produced by our method or the one produced by DNR. As for 'Plant', 100 random crops of $\frac{1}{6.5}$ original image size were selected. Users have preferred Ours vs. Deferred in **49.7% vs. 50.3%** cases for 'Studio' and in **69% vs. 31%** for 'Plant'. As before, our method performs significantly better when meshing procedure fails in the presence of thin objects and yields results visually similar to DNR when the mesh artefacts are relatively rare.

Ablation study. We also evaluate the effect of some of the design choices inside our method. First, we consider the point cloud density and the descriptor size. We take an evaluation scene from ScanNet and progressively downsample point cloud using voxel downsampling and get small, medium and large variants with 0.5, 1.5, and 10 million points respectively. For each variant of point cloud we fit descriptors with sizes M equal to 4, 8 (default) and 16. Table 2 shows evaluation results. It naturally reveals the advantage of using denser point clouds, yet it also shows that the performance of our method saturates at $M = 8$. The same observation is supported by the qualitative comparison shown in Figure 8. Additionally, we investigate what is the most important add-on introduced as a part of our method. As Table 3 shows, all the features of the pipeline are helpful, while the use of learnable per-point descriptors results in the most dramatic improvement.

Fig. 6: Novel views of **People** generated by our method (*large picture*: our rendered result, *small overlaid picture*: nearest view from the fitting part). Both people are from the holdout part (excluded from pretraining).

(a) **'Owl'** and **'Plant'**. *Large picture:* render from a novel point, *small overlaid picture:* nearest train view

(b) **'Studio'**. *First row:* render from a novel point, *second row:* nearest train view

Fig. 7: Various results obtained by our method. For each of the scenes, we show the view from the nearest camera from the fine-tuning part (*nearest train view*).

Scene editing. To conclude, we show a qualitative example of creating a composite of two separately captured scenes (Fig. 9). To create it, we took the

Table 2: Dependency of the loss function values on the descriptor size and the point cloud size. Comparison is made for the 'Studio' scene of Scannet.

	Descriptor size 4		Descriptor size 8		Descriptor size 16	
Point Cloud size	VGG ↓	LPIPS ↓	VGG ↓	LPIPS ↓	VGG ↓	LPIPS ↓
small	635.74	0.543	632.39	0.505	622.06	0.508
medium	622.05	0.506	616.49	0.486	614.90	0.500
large	**610.76**	**0.509**	**609.11**	**0.485**	**611.38**	**0.488**

Table 3: Ablation study w.r.t. the add-ons of our pipeline. Comparison is made for the Person 1 fitted from scratch.

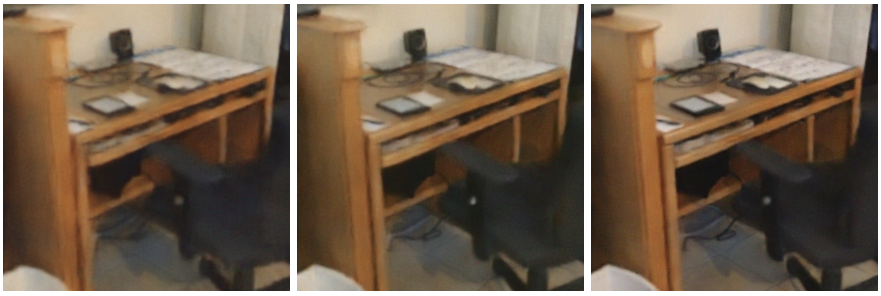
	FID ↓	LPIPS ↓	VGG ↓
Ours	197.54	**0.526**	**918.79**
Ours w/o mipmapping (1 scale input)	196.27	0.527	920.11
Ours w/ vanilla convs instead of gated	**192.92**	0.527	924.25
Ours w/ L1 loss instead of VGG loss	350.10	0.682	1053.4
Ours w/o per-point feature	476.17	0.798	1222.5

'Person 2' and the 'Plant' datasets and fitted descriptors for them while keeping the rendering network, pretrained on People, frozen. We then align the two point clouds with learned descriptors by a manually-chosen rigid transform and created the rendering.

Anti-aliasing. We have found that in the presence of camera misregistrations in the fitting set (such as ScanNet scenes), our method tends to produce flickering artefacts during camera motion (as opposed to [29] that tends to produce blurry outputs in these cases). At least part of this flickering can be attributed to rounding of point projections to the nearest integer position during rendering. It is possible to generate each of the raw images at higher resolution (e.g. 2× or 4× higher), and then downsample it to the target resolution using bilinear interpolation resulting in smoother raw images. This results in less flickering with a cost of barely noticeable blur (see **supplementary video**). Note that the increase in time complexity from such anti-aliasing is insignificant, since it does not affect the resolution at which neural rendering is performed.

5 Discussion

We have presented a neural point-based approach for modeling complex scenes. Similarly to classical point-based approaches, ours uses 3D points as modeling primitives. Each of the points in our approach is associated with a local descriptor containing information about local geometry and appearance. A rendering network that translates point rasterizations into realistic views, while taking the learned descriptors as an input point pseudo-colors. We thus demonstrate that point clouds can be successfully used as geometric proxies for neural rendering, while missing information about connectivity as well as geometric noise and



small cloud *medium* cloud *large* cloud

Fig. 8: Variation of rendering quality w.r.t. the number of points the scene. Comparison is made for a crop of a holdout image corresponding to the 'Studio' scene of Scannet.

original scene

Person 2 + Plant: point cloud composition

Person 2 + Plant: ours

Fig. 9: A novel view for the composed scenes. A point cloud from 'Plant scene was translated and rotated slightly to be placed on the left hand of the 'Person 2'. The world scale of objects was kept unchanged.

holes can be handled by deep rendering networks gracefully. Thus, our method achieves similar rendering quality to mesh-based analog [29], surpassing it wherever meshing is problematic (e.g. thin parts).

Limitations and further work. Our model currently cannot fill very big holes in geometry in a realistic way. Such ability is likely to come with additional point cloud processing/inpainting that could potentially be trained jointly with our modeling pipeline. We have also not investigated the performance of the system for dynamic scenes (including both motion and relighting scenarios), where some update mechanism for the neural descriptors of points would need to be introduced.

References

1. Blinn, J.F., Newell, M.E.: Texture and reflection in computer generated images. Communications of the ACM **19**(10) (1976) 542–547
2. Blinn, J.F.: Simulation of wrinkled surfaces. In: Proc. SIGGRAPH. Volume 12., ACM (1978) 286–292
3. Debevec, P., Yu, Y., Borshukov, G.: Efficient view-dependent image-based rendering with projective texture-mapping. In: Rendering Techniques. Springer (1998) 105–116
4. Wood, D.N., Azuma, D.I., Aldinger, K., Curless, B., Duchamp, T., Salesin, D.H., Stuetzle, W.: Surface light fields for 3d photography. In: Proc. SIGGRAPH. (2000) 287–296
5. McMillan, L., Bishop, G.: Plenoptic modeling: an image-based rendering system. In: SIGGRAPH, ACM (1995) 39–46
6. Seitz, S.M., Dyer, C.R.: View morphing. In: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, ACM (1996) 21–30
7. Gortler, S.J., Grzeszczuk, R., Szeliski, R., Cohen, M.F.: The lumigraph. In: SIGGRAPH, ACM (1996) 43–54
8. Levoy, M., Hanrahan, P.: Light field rendering. In: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, ACM (1996) 31–42
9. Levoy, M., Whitted, T.: The use of points as a display primitive. Citeseer (1985)
10. Grossman, J.P., Dally, W.J.: Point sample rendering. In: Rendering Techniques 98. Springer (1998) 181–192
11. Gross, M., Pfister, H., Alexa, M., Pauly, M., Stamminger, M., Zwicker, M.: Point based computer graphics. Eurographics Assoc. (2002)
12. Kobbelt, L., Botsch, M.: A survey of point-based techniques in computer graphics. Computers & Graphics **28**(6) (2004) 801–814
13. Isola, P., Zhu, J., Zhou, T., Efros, A.A.: Image-to-image translation with conditional adversarial networks. In: Proc. CVPR. (2017) 5967–5976
14. Nalbach, O., Arabadzhiyska, E., Mehta, D., Seidel, H., Ritschel, T.: Deep shading: Convolutional neural networks for screen space shading. Comput. Graph. Forum **36**(4) (2017) 65–78
15. Chen, A., Wu, M., Zhang, Y., Li, N., Lu, J., Gao, S., Yu, J.: Deep surface light fields. Proceedings of the ACM on Computer Graphics and Interactive Techniques **1**(1) (2018) 14
16. Bui, G., Le, T., Morago, B., Duan, Y.: Point-based rendering enhancement via deep learning. The Visual Computer **34**(6-8) (2018) 829–841
17. Hedman, P., Philip, J., Price, T., Frahm, J., Drettakis, G., Brostow, G.J.: Deep blending for free-viewpoint image-based rendering. ACM Trans. Graph. **37**(6) (2018) 257:1–257:15
18. Pfister, H., Zwicker, M., Van Baar, J., Gross, M.: Surfels: Surface elements as rendering primitives. In: Proceedings of the 27th annual conference on Computer graphics and interactive techniques, ACM Press/Addison-Wesley Publishing Co. (2000) 335–342
19. Zwicker, M., Pfister, H., Van Baar, J., Gross, M.: Surface splatting. In: Proc. SIGGRAPH, ACM (2001) 371–378
20. Meshry, M., Goldman, D.B., Khamis, S., Hoppe, H., Pandey, R., Snavely, N., Martin-Brualla, R.: Neural rerendering in the wild. In: Proc. CVPR. (June 2019)
21. Pittaluga, F., Koppal, S.J., Kang, S.B., Sinha, S.N.: Revealing scenes by inverting structure from motion reconstructions. In: Proc. CVPR. (June 2019)

22. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. International journal of computer vision **60**(2) (2004) 91–110
23. Flynn, J., Neulander, I., Philbin, J., Snavely, N.: Deepstereo: Learning to predict new views from the world's imagery. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2016) 5515–5524
24. Ganin, Y., Kononenko, D., Sungatullina, D., Lempitsky, V.S.: Deepwarp: Photorealistic image resynthesis for gaze manipulation. In: Proc. ECCV. (2016) 311–326
25. Zhou, T., Tulsiani, S., Sun, W., Malik, J., Efros, A.A.: View synthesis by appearance flow. In: Proc. ECCV. (2016) 286–301
26. Thies, J., Zollhöfer, M., Theobalt, C., Stamminger, M., Nießner, M.: Ignor: Image-guided neural object rendering. arXiv 2018 (2018)
27. Martin-Brualla, R., Pandey, R., Yang, S., Pidlypenskyi, P., Taylor, J., Valentin, J., Khamis, S., Davidson, P., Tkach, A., Lincoln, P., et al.: Lookingood: enhancing performance capture with real-time neural re-rendering. In: SIGGRAPH Asia 2018 Technical Papers, ACM (2018) 255
28. Sitzmann, V., Thies, J., Heide, F., Nießner, M., Wetzstein, G., Zollhöfer, M.: Deepvoxels: Learning persistent 3d feature embeddings. In: Proc. CVPR. (2019)
29. Thies, J., Zollhöfer, M., Nießner, M.: Deferred neural rendering: Image synthesis using neural textures. In: Proc. SIGGRAPH. (2019)
30. Zhou, Q., Koltun, V.: Color map optimization for 3d reconstruction with consumer depth cameras. ACM Trans. Graph. **33**(4) (2014) 155:1–155:10
31. Bi, S., Kalantari, N.K., Ramamoorthi, R.: Patch-based optimization for image-based texture mapping. ACM Trans. Graph. **36**(4) (2017) 106:1–106:11
32. Huang, J., Thies, J., Dai, A., Kundu, A., Jiang, C., Guibas, L.J., Niessner, M., Funkhouser, T.: Adversarial texture optimization from rgb-d scans. In: Proc. CVPR. (2020) 1559–1568
33. Johnson, J., Alahi, A., Fei-Fei, L.: Perceptual losses for real-time style transfer and super-resolution. In: Proc. ECCV. (2016) 694–711
34. Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: International Conference on Medical image computing and computer-assisted intervention, Springer (2015) 234–241
35. Yu, J., Lin, Z., Yang, J., Shen, X., Lu, X., Huang, T.S.: Free-form image inpainting with gated convolution. arXiv preprint arXiv:1806.03589 (2018)
36. Williams, L.: Pyramidal parametrics. In: Proceedings of the 10th annual conference on Computer graphics and interactive techniques. (1983) 1–11
37. Dosovitskiy, A., Brox, T.: Generating images with perceptual similarity metrics based on deep networks. In: Proc. NIPS. (2016) 658–666
38. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. CoRR **abs/1409.1556** (2014)
39. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR **abs/1412.6980** (2014)
40. Dai, A., Chang, A.X., Savva, M., Halber, M., Funkhouser, T., Nießner, M.: ScanNet: Richly-annotated 3d reconstructions of indoor scenes. In: Proc. CVPR. (2017)
41. Dai, A., Nießner, M., Zollhöfer, M., Izadi, S., Theobalt, C.: Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface reintegration. ACM Trans. Graph. **36**(3) (2017) 24:1–24:18
42. Gong, K., Gao, Y., Liang, X., Shen, X., Wang, M., Lin, L.: Graphonomy: Universal human parsing via graph transfer learning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2019) 7450–7459
43. Agisoft: Metashape software. (retrieved 20.05.2019)

44. Zhang, R., Isola, P., Efros, A.A., Shechtman, E., Wang, O.: The unreasonable effectiveness of deep features as a perceptual metric. In: CVPR. (2018)
45. Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Hochreiter, S.: Gans trained by a two time-scale update rule converge to a local nash equilibrium. In: Advances in neural information processing systems. (2017) 6626–6637