Neural Point Cloud Rendering via Multi-Plane Projection

Peng Dai1    Yinda Zhang2    Zhuwen Li3    Shuaicheng Liu1    Bing Zeng1

1University of Electronic Science and Technology of China    2Google LLC    3Nuro Inc

Abstract

We present a new deep point cloud rendering pipeline through multi-plane projections. The input to the network is the raw point cloud of a scene and the output are image or image sequences from a novel view or along a novel camera trajectory. Unlike previous approaches that directly project features from 3D points onto 2D image domain, we propose to project these features into a layered volume of camera frustum. In this way, the visibility of 3D points can be automatically learnt by the network, such that ghosting effects due to false visibility check as well as occlusions caused by noise interferences are both avoided successfully. Next, the 3D feature volume is fed into a 3D CNN to produce multiple layers of images w.r.t. the space division in the depth directions. The layered images are then blended based on learned weights to produce the final rendering results. Experiments show that our network produces more stable renderings compared to previous methods, especially near the object boundaries. Moreover, our pipeline is robust to noisy and relatively sparse point cloud for a variety of challenging scenes.

1. Introduction

Rendering is on high demand for many graphics and vision applications. To produce high quality rendering, various physical understanding of the scene has to be established, such as scene geometry [1], scene textures [2], materials [3], illuminations [4], all of which require tremendous efforts to obtain. After the construction of rendering essentials, a photo-realistic view of the modeled scene is generated through expensive rendering process such as ray tracing [5] and radiance simulation [6].

Image based rendering (IBR) techniques [7, 8, 9], alternatively, try to render a novel view based on the given images and their approximated scene geometries through image warping [10] and image inpainting [11]. The scene structure approximation often adopts simplified forms such that the rendering process becomes relatively cheaper than physically based renderings. However, IBR requires the novel view to stay close to the original views in order to avoid rendering artifacts. Instead of purely based on images, point-based graphics (PBG) [12, 13, 14] simplify the scene structures by replacing the surface mesh with point cloud or surfels [15], such that the heavy geometry constructions can be avoid.

On the other hand, many deep learning approaches show strong capability in inpainting [16], refining [17], and even constructing images from only a few indications [18, 19]. Such capabilities can be considered as a complement to IBR, namely neural IBR [20], to overcome the challenging during image synthesize. For examples, the empty regions caused by view change can be compensated through high quality inpainting [21].

Recently, combining advantages of simplified geometry representation and neural capabilities become a new trend, yielding neural rendering methods [22, 23, 20, 20, 24]. It directly learns to render end-to-end, which bypass complicated intermediate representations. Previous work use mostly 3D volume as representation [25]. However, the memory complexity of 3D volume is cubic, and thus these approaches are not scalable and usually only work for small...
2. Related Work

2.1. Model-based rendering

Model based rendering requires the construction of 3D models, such as multi-view structure-from-motion for point cloud recovery [29], surface reconstruction and meshing [32]. When performance is preferred, ray tracing [5] is used to simulate the transmission of light in the space, such that can better interact with the environment, e.g., geometry [33], material [34], BRDF [35], lighting [36], and produce more realistic rendering. However, each estimation step is prone to errors, which leads to the render artifacts. Moreover, such methods not only require a lot of knowledge of the scene, but also are notoriously slow.

2.2. Image-based rendering

Image based rendering (IBR) [7, 8, 9, 37] aims to produce a novel view from given images through warping [10] and blending [34], which is a computationally efficient approach compared with classical rendering pipeline. Multiple view geometry [29] is applied for camera parameter estimation or some variant that bypasses the 3D reconstruction [33], such as adopting epipolar constraints [38]. Recently, deep learning has been proved to be more effective in replacing the warping and blending of traditional approaches in the IBR pipeline [39, 20]. However, the quality of rendered novel view still depends heavily on the distribution of existing views, sparse samples or large viewpoint drift would produce unsatisfactory results. Adopting light field cameras is one solution to alleviate such problems [40].

2.3. Deep image synthesis

Deep methods for 2D image synthesize has achieved very promising results, such as autoencoders [41], PixelCNN [42] and image-to-image translation [43]. The most exciting results are produced based on generative adversarial networks [44, 18]. Most of the generators adopt encoder-decode architecture with skip connections to facilitate feature propagation [45]. However, these approaches cannot be directly applied to the rendering task, as the underlying 3D structures cannot be exploited for 2D image translations.

2.4. Neural rendering

Recently, deep learning are used to renovating the rendering pipeline. Nvidia [47] uses deep learning to denoise a relatively fast low-quality rendering. More fundamentally, many successes have been achieved by neural rendering that directly learn representation from input and produce desired output, such as DeepVoxel [25] and Neural Point Based Graphics (NPG) [28]. Most of them rely on a 3D volume intermediate representation. The DeepVoxel cannot render large scenes, such as room environment. The most related work is NPG [28]; it also proposes to render images from

objects. Recently, there is a trend to build neural rendering pipeline from 3D point cloud, which is more scalable for relatively larger scenes. However, 3D point cloud often contains strong noise due to both depth measurements [26] and camera calibrations [27], which interference the visibility check when projected onto 2D image plane and results in jarring artifacts if a series of images are generated along a given 3D camera trajectory. On the other hand, this type of methods often require a lot of points for a reliable z-buffer check as well as the requirement of full coverage of all camera viewpoints. Even though the memory usage is linear with the number of points, the huge number still cause unaffordable memory and storage. Aliev et al. [28] proposed a neural point-based graphics approach that directly project 3D geometry onto the 2D plane for neural descriptor encoding, which not only ignores the visibility check but also suffers from noise interferences, resulting ghosting artifacts as well as strong temporal jitters.

In this paper, we propose a novel deep point cloud rendering pipeline through multi-plane projection, which is more robust to depth noise and can work with relatively sparse point cloud. In particular, instead of directly projecting features from 3D points onto 2D image domain using perspective geometry [29], we propose to project these features into a layered volume in the camera frustum. By doing so, all the features from points in the camera’s field-of-view are maintained, thus useful features are not occluded accidentally by other points due to noisy interferences. The 3D feature volume are then fed into a 3D CNN to produce multiple layers of images, which corresponds to different space depths. The layered images are blended subsequently according to learned weights. In this way, the network can fix the point cloud errors in the 3D space rather than work on projected 2D images where visibility has already lost. In addition, the network can pick information accurately from the projected full feature volume to facilitate the rendering.

Extensive experiments evaluated on the popular dataset, such as ScanNet [30] and Matterport 3D [31], show that our model produces more temporal coherent rendering results comparing to previous methods, especially near the object boundary. Moreover, the system can learn effectively from more points, but still perform reasonably well given relatively sparse point cloud.

To recap, we propose a deep learning based method to render images from point cloud. Our contributions are:

- 3D points are projected to a layered volume such that occlusions and noises can be handled appropriately.
- Not only the rendered single view is superior in terms of image quality, but also the rendered image sequences are temporally more stable.
- Our system works reasonably well with respect to relatively sparse point cloud.
point cloud, by projecting learned features from points to the 2D image plane according to perspective geometry, and train a 2D CNN to produce the color image. The method learns complete and view-dependent appearance, however, it suffers from visibility verification problem due to directly 3D projection, and is also sensitive to point cloud noises. We project 3D points to a layered 3D volume to overcome such problems.

3. Method

Our deep learning framework receives a point cloud representation of a scene and generates photo-realistic images from an arbitrary camera viewpoint. The overview of the framework is illustrated in Fig. 2. The whole framework consists of two modules: multi-layer based voxelization and multi-layer rendering. The multi-layer based voxelization module divides the 3D space of the camera view frustum into voxels w.r.t. image dimensions and a pre-defined number of depth layers. The voxels then aggregates features of points inside it with geometric rules. The 3D feature volume is then fed into the multi-layer rendering module, which is a 3D CNN, to generate one color image plus a blending weight per layer on the depth dimension of the volume. The final output is a weighted blending of multi-layer images. It is worth noting that the point cloud feature representation and the network are jointly optimized in an end-to-end fashion. The remaining part of this section describes details of these two modules.

3.1. Learnable point cloud features

Our input is a 3D point cloud representation of the scene. To be sufficient for rendering, each 3D point should contain both position and appearance feature. The position feature is obtained from 3D reconstruction, and one simple way to collect appearance feature is to keep the RGB value from the corresponding image pixel. However, one point may show different RGB intensities when observed from different views due to view-depend effects (e.g., reflection and highlight). To solve this problem, we learn a 8-dimensional vector as the appearance feature jointly with the network parameters. To this end, we update this feature by propagating gradient to the input [28, 46], such that the appearance feature can be automatically learned from data. Since the object appearance is often view-depend, we further take view direction between camera position and point clouds in 3D space into consideration. Thus, we concatenate the normalized view direction of the point as an additional feature vector to each point, following [28]. Note that this feature is not trainable as the point position.

3.2. Multi-layer based voxelization

Layered voxels on camera frustum. The image dimension is denoted as $H \times W$. With a known camera projection matrix, each pixel is lifted to 3D space to form a frustum with near and far planes specified by the minimum and maximum depth of the projected point cloud. The frustum is further divided into $P$ small ones along the $z$-axis uniformly as shown in Fig. 3(a). As a result, we will obtain $P \times H \times W$ frustum voxels.

Feature aggregation The next step is to aggregate feature from the point cloud into the camera frustum volume. Since one frustum voxel may contain multiple 3D points, we need an efficient and effective way to aggregate features. Aliev et al. [28] propose to take the feature from the point closest to camera along the ray, however, this is not robust against geometry error and may result in temporal jitters or accidentally wrong occlusions. In contrast, we maintain all the...
depth plane coordinate in volume. Specifically, the feature in a voxel with point feature to nearby voxels according to the distance to volume. To achieve sub-voxel performance, we vote each features available in the camera frustum thanks to the 3D weights are applied to aggregate multiple points into a new point. Each point on the basis of these two distance. Finally, the blend weight on direction parallel and vertical to the image plane. Further, we can define the blend weight between points and minimum depth value in this voxel in vertical direction. Distance.

To illustrate, we take one frustum voxel for example. In order to aggregate points in one frustum voxel, we calculate two kinds of distance. \( D_1(p, h, w) \) is the distance between points and grid center in parallel direction and \( D_2(p, h, w) \) calculates depth difference between points and minimum depth value in this voxel in vertical direction. Further, we can define the blend weight \( w_i(p, h, w) \) of each point on the basis of these two distance. Finally, the blend weights are applied to aggregate multiple points into a new point.

features available in the camera frustum thanks to the 3D volume. To achieve sub-voxel performance, we vote each point feature to nearby voxels according to the distance to the voxel center. Specifically, the feature in a voxel with coordinate in volume \((p, h, w)\) is calculated as:

\[
F(p, h, w) = \frac{\sum_i w_i(p, h, w) \times F_i(p, h, w)}{\sum_i w_i(p, h, w)},
\]

\[
w_i(p, h, w) = (1 - D_1(p, h, w))^a \times \left( \frac{1}{1 + D_2(p, h, w)} \right)^b,
\]

where \(F_i\) is the feature of the \(i\)th point in a voxel, and \(w_i(p, h, w)\) is the blending weight of point \(i\) to voxel \((p, h, w)\). \(D_1\) is the distance between the point projection on image to the corresponding pixel center of voxel \((p, h, w)\), and \(D_2\) is the depth difference between point and minimum depth point in this voxel. Parameter \(a\) and \(b\) controls the blending weight on direction parallel and vertical to the image plane. When \(b \to +\infty\), it turns into Aliev et al. [28] where a point is picked via z-buffer.

3.3. Multi-layer rendering

We adopt the U-Net-like 3D convolutional neural network as the back-bone network. The 3D convolution effectively exploit information from neighboring pixel and depth, which naturally handle projection error caused by geometry noise. In addition to that we also adopt dilated convolution in the last layer of encoder (left part of U-Net) to capture more image context. As the output of our network, we predict multi-layer RGB images plus their blending weights. The final output image \(I\) is obtained by

\[
I = \sum P \cdot \alpha_p,
\]

where \(P\) indicates a layer, and \(I_p\) and \(\alpha_p\) are the corresponding layer predictions. Please refer to supplementary material for more details of the network architecture.

3.4. Loss function

To measure the difference between network prediction and the ground truth image, we use perceptual loss [23, 48], which we found practically works better than other common candidates, e.g. \(\ell_1\), \(\ell_2\), SSIM. In particular, we use feature vector from ‘input’, ‘conv1-2’, ‘conv2-2’, ‘conv3-2’, ‘conv4-2’, ‘conv5-2’ layers from a VGG-19 pre-trained on ImageNet dataset [49]. The perceptual loss is defined as a weighted sum of the \(\ell_1\) loss on each feature maps, with the weight following Chen et al. [23]. Note that VGG-19 is not updated during training.

3.5. Feature optimization.

Inspired by Thies et al. [46] and Aliev et al. [28], the appearance feature on each point can be updated via back-propagation. Note that the aggregated voxel features are a weighted combination of point features, and thus gradient on the point feature using the chain rule is \(-l_r \times w_i(p, h, w) \times g_i(p, h, w)\), where \(g_i(p, h, w)\) is gradient derived from the loss function and \(l_r\) indicates the learning rate.

4. Experiments

We evaluate our framework on various datasets and show qualitative and quantitative results. Particularly, we test the system robustness against noise in data that heavily degrades performance of previous methods.

4.1. Datasets.

ScanNet [30] contains RGBD scans of indoor environments. We follow the training and testing split of Aliev et al. [28]. In particular, one frame is picked from every 100 frames for testing (e.g., frame 100, 200, 300...). The rest of frames are used for training. To avoid including frames that are too similar to the training set, the neighbors (with in 20 frames) of every testing frame are removed from training. Regarding the scene point cloud, we randomly lift 15% pixels from the depth map into the 3D space to create a point cloud, which contains around 50 million points per scene. We then simplify it using volumetric sampling, leading to 8.9 million points per scene in average.
Datasets | ScanNet | Matterport 3D
--- | --- | ---
Methods | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓
pix2pix [43] | 19.247 | 0.731 | 0.429 | 14.964 | 0.530 | 0.675
Neural point based graphic [28] | **22.911** | **0.840** | **0.245** | 17.931 | 0.622 | 0.597
Ours + direct render | 22.259 | 0.818 | **0.230** | 17.833 | 0.601 | 0.610
Ours | **22.813** | **0.835** | **0.234** | **18.09** | **0.649** | **0.534**

| Table 1. PSNR, SSIM and LPIPS values on ScanNet and Matterport 3D datasets.

4.3. Rendering results

We first compare our method to two competitors, Neural Point-based Graphic (NPG) [28] and Pix2Pix [43]. Specifically, NPG is a deep rendering approach that project 3D point features onto 2D image planes via a z-buffer and run 2D convolutions for neural rendering. They adopt U-Net like structure with gate convolution [21]. Since the authors of [28] did not release the code, we implemented their method and achieved similar performance on the same testing cases. The Pix2Pix is an image to image translation framework [43]. The network takes projected colored point cloud and is trained to produce the ground truth. Compared to NPG, this baseline does not save a feature per point.

4.3.1 Quantitative comparisons

We use standard metric, Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index (SSIM), to measure the rendering quality. Since these two metrics may not necessarily reflect the visual quality, we also adopt a human perception metric, Learned Perceptual Image Patch Similarity (LPIPS) [51]. Table 1 reports the comparison on two datasets. Our method is significantly better than Pix2Pix on both dataset with a large improvement margin on all the metrics. When compared to NPG, our method outperforms on Matterport3D, and is comparable on Scan-Net, where NPG achieves better PSNR and SSIM, while our LPIPS is higher. Some examples from both dataset are shown in Fig. 5. Note that it has been mentioned in their own paper that NPG is optimized for pixel-wise color accuracy at the cost of sacrificing the temporal consistency. In contrast, our result is free such jittering, especially obvious at depth boundaries. Please refer to the supplementary video for visual comparisons.

4.4. Data preparation and training details

For each scene, our network is trained for 21 epochs over 1,925 images on average, using Adam optimizer [50]. During training, $l_r$ is initialized as 0.01, which will be decreased every 7 epochs, and the learning rate decreasing follows $0.01 \rightarrow 0.005 \rightarrow 0.001$. $(P, H, W)$ are set to $(32, 480, 640)$ for ScanNet dataset and $(32, 512, 640)$ for Matterport 3D dataset according to the image resolution provided by the datasets. Point feature dimension is set as 11 and initialized as 0.5 (5 dimensions) + RGB (3 dimensions) + viewpoint direction (3 dimensions), note that only the first 8 dimensions will be updated. The hyperparameters $a, b$ in Eq. 2 are set to 1. The training process is accomplished on one GeForce 1080 Ti, which takes on average 41.5 hours per scene.

Direct render To verify if learning point cloud feature is necessary. We train our model taking only the point RGB value as the feature, which is referred to as ‘direct render’. The results are displayed in Fig. 4. As seen, the direct render without point feature is more blurry (e.g., the sofa) and lack of specular components (e.g., the ball). This indicates that point feature helps to encode material related information and support view-dependent components. We also report the quantitative numbers in Table 1 as ‘Ours+direct render’.
4.3.2 Qualitative comparisons

Figure 5 shows some visual comparisons of our method with NPG and Pix2Pix on ScanNet and Matterport 3D datasets. The first two rows show two scenes of ScanNet while the third and forth rows show two scenes of Matterport 3D. Fig. 5 (a) shows point cloud. The point cloud is noisy and incomplete. Fig. 5 (b) shows the result of NPG. For its ScanNet results, we notice some incorrect places, e.g., black stripe on the laptop screen of first scene, and missing details of shelf of second scene (Please zoom in for details). For its Matterport results, the missing of details become more serious, e.g., the floor textures is missing in the forth example. Fig. 5 (c) shows the result of pix2pix. It generates strange curves for ScanNet while introduce weird textures for Matteport result. Fig. 5 (d) shows our results. As seen, our result is free from such problems.

4.4. Robustness and stability

Robustness to noise point cloud In practice, the point cloud are usually noisy, and the rendering model needs to tolerant such noises to produce robust results. When objects are close to each other, noisy depth may result in wrong z-buffer such that the correct points are occluded. This is especially harmful for methods that rely on 2D projection of the point cloud, such as NPG and Pix2Pix. In contrast, our method maintains all related point features in the camera render’. It is observed that ours with learner features outperforms the direct render approach on all metrics.
Figure 6. Restore adversely occluded objects. (a) Noisy point clouds with objects adversely occluded. (b) pix2pix method. (c) Results generated through neural point based graphic method [28] from 2D point clouds images. Can’t efficiently restore occluded objects. (d) Our results generated from 3D neural render. Restore adversely occluded objects. (e) Ground truth.

Figure 7. SSIM and PSNR trend when point clouds density decreased.

Robustness to varying densities Theoretically, our method can support arbitrarily large scene since the point features can be stored on hard drive. However for efficient rendering, it is more desirable to keep point features in memory and render from relatively sparse point cloud to save both the memory and the computational cost for point projection. Unfortunately, sparse point cloud may result in incomplete z-buffer such that the occluded background shows up in the image. NPG proposes to assign each point a square size during the projection according to its depth to mitigate this issue, but may not be sufficient. Fig. 8 shows a qualitative comparison to NPG on the same scene with different point density. With less points, NPG reveals more background due to incomplete z-buffer, while our method still maintain the chair. Fig. 7 further shows the quantitative comparison. As seen, while both methods perform worse with fewer points, the metrics of our method drops relatively slower, which means using camera frustum is more robust against the varying point density.

Temporal consistency 3D camera frustum also helps to improve the temporal consistency. 2D projection based methods may project very different points for the same 3D location to very close camera viewpoints. This is because the order of the points in the z-buffer may dramatically change between slightly different camera views. Consequently, the rendering of the same 3D location may use features from different points and thus cause jittering artifacts.

We perform a user study to compare the temporal consis-
Figure 8. Different sparsity of point clouds. The number of point clouds varies from column to column, and different row shows results generated from different method. By analysis, our method can work at relatively sparse point clouds.

Figure 9. The user study result on preference for videos synthesized by different methods (i.e. ours, NPG, direct render and pix2pix.)

5. Conclusion

In this work, we propose a method which synthesizes novel view images from 3D point clouds. Instead of directly project features from 3D points onto 2D image domain, we projected these features into a layered volume of camera frustum, such that the visibility of 3D points can be naturally maintained. Through experiments, our method is robust to point clouds noise and generates flicker-less videos.

In the future, we will explore novel view synthesis from point clouds in multiple views, considering sharing points can provide potential information, such as optical flow which can be utilized for further temporal consistency and flickers removal by enforcing shared points should generate similar predictions. What’s more, applying interpolation in different depth layers maybe further improve the robust to sparse point clouds.
References

[1] Angela Dai, Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Christian Theobalt. Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface reintegration. ACM Trans. Graphics, 36(3):24, 2017.

[2] James F Blinn. Simulation of wrinkled surfaces. In ACM SIGGRAPH computer graphics, volume 12, pages 286–292, 1978.

[3] Oliver Wang, Prabath Gunawardane, Steve Scher, and James Davis. Material classification using brdf slices. In Proc. CVPR, pages 2805–2811, 2009.

[4] Steven J Gortler, Radek Grzeszczuk, Richard Szeliski, and Gregory J Ward. The radiance lighting simulation and rendering system. In Proceedings of the 21st annual conference on Computer graphics and interactive techniques, pages 459–472. ACM, 1994.

[5] Marc Levoy and Pat Hanrahan. Light field rendering. In Proc. of conference on Computer graphics and interactive techniques, pages 31–42, 1996.

[6] Gregory J Ward. The radiance lighting simulation and rendering system. In Proceedings of the 21st annual conference on Computer graphics and interactive techniques, pages 459–472. ACM, 1994.

[7] Leonard McMillan and Gary Bishop. Plenoptic modeling: An image-based rendering system. In Proc. of conference on Computer graphics and interactive techniques, pages 31–42, 1996.

[8] Leif Kobbelt and Mario Botsch. A survey of point-based techniques in computer graphics. Computers & Graphics, 28(6):801–814, 2004.

[9] Jeffrey P Grossman and William J Dally. Point sample rendering. In Rendering techniques 98, pages 181–192. 1998.

[10] Kirill Kuropkov, Mario Botsch, and Matthias Nießner. Deepvoxels: Learning persistent 3d feature embeddings. In Proc. CVPR, pages 2437–2446, 2019.

[11] Katja Wolff, Changil Kim, Henning Zimmer, Christopher Schroers, Mario Botsch, Olga Sorkine-Hornung, and Alexander Sorkine-Hornung. Point cloud noise and outlier removal for image-based 3d reconstruction. In International Conference on 3D Vision (3DV), pages 118–127, 2016.
[31] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. *arXiv preprint arXiv:1709.06158*, 2017.

[32] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2mesh: Generating 3d mesh models from single rgb images. In *Proc. ECCV*, pages 52–67, 2018.

[33] Noah Snavely, Steven M Seitz, and Richard Szeliski. Photo tourism: exploring photo collections in 3d. In *ACM Trans. Graphics*, volume 25, pages 835–846, 2006.

[34] Paul Debevec, Yizhou Yu, and George Borshukov. Efficient view-dependent image-based rendering with projective texture-mapping. In *Rendering Techniques*, pages 105–116. 1998.

[35] Dan B Goldman, Brian Curless, Aaron Hertzmann, and Steven M Seitz. Shape and spatially-varying brdfs from photometric stereo. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 32(6):1060–1071, 2009.

[36] Yannick Hold-Geoffroy, Kalyan Sunkavalli, Sunil Hadap, Emiliano Gambaretto, and Jean-François Lalonde. Deep outdoor illumination estimation. In *Proc. CVPR*, pages 7312–7321, 2017.

[37] Steven M Seitz and Charles R Dyer. View morphing. In *Proc. of conference on Computer graphics and interactive techniques*, pages 21–30, 1996.

[38] Amit Goldstein and Raanan Fattal. Video stabilization using epipolar geometry. *ACM Trans. Graphics*, 31(5):126, 2012.

[39] John Flynn, Ivan Neulander, James Philbin, and Noah Snavely. Deepstereo: Learning to predict new views from the world’s imagery. In *Proc. CVPR*, pages 5515–5524, 2016.

[40] Nima Khademi Kalantari, Ting-Chun Wang, and Ravi Ramamoorthi. Learning-based view synthesis for light field cameras. *ACM Trans. Graphics*, 35(6):193, 2016.

[41] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[42] Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In *Proc. NIPS*, pages 4790–4798, 2016.

[43] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proc. CVPR*, pages 1125–1134, 2017.

[44] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proc. CVPR*, pages 2223–2232, 2017.

[45] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241, 2015.

[46] Justus Thies, Michael Zollhöfer, and Matthias Nießner. Deferred neural rendering: Image synthesis using neural textures. *arXiv preprint arXiv:1904.12356*, 2019.

[47] Chakravarty R Alla Chaitanya, Anton S Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila. Interactive reconstruction of monte carlo image sequences using a recurrent denoising autoencoder. *ACM Trans. Graphics*, 36(4):98, 2017.

[48] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *Proc. ECCV*, pages 694–711, 2016.

[49] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proc. CVPR*, pages 248–255. Ieee, 2009.

[50] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[51] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proc. CVPR*, pages 586–595, 2018.
Supplementary Materials for
“Neural Point Cloud Rendering via Multi-Plane Projection”

We provide details about network architecture, evaluation details, and results on various datasets.

1. Network Architecture

The network architecture is provided in Table 1, including kernel size, stride, kernel dilation, activation function, and feature channels. In short, our network is a UNet with short-cut connection.

2. Evaluation Details

Our test set consists of four scenes from ScanNet [3] and two scenes from Matterport3D [2]. The scene IDs chosen from ScanNet are ‘scene0000_00’, ‘scene0010_00’, ‘scene0016_00’, ‘scene0024_00’, and the first and fourth scenes are also adopted in NPG [1]. The scene IDs chosen from Matterport3D are ‘17DRP5sb8fy’ and ‘29hnd4uzFmX’.

We calculate PSNR and SSIM in standard way by using the function provided by Matlab.

3. More Results

3.1. Multi-Plane Image and Weight

For the network outputs, we produce multiple layers of images with their blending weights. Figure 1 visualizes one of the output example. As can be seen, the blending weights roughly reflects the depth of the scene since each layer corresponds to a specific depth in camera coordinate. Regarding the color image, all the images seems to maintain the scene layout reasonably well, while for each pixel the layer with the correct depth are likely to provide the right color.

3.2. More Results on ScanNet

In this section, we show more results on ScanNet dataset in Figure 2 and Figure 3. Our method is better than pix2pix in terms of the visual quality. Compared to NPG [1], we perform especially better at the location with sparsity and occlusion.

3.3. More Results on Matterport3D

We also show more results on Matterport3D dataset in Figure 4 and Figure 5. Our results are significantly better than all the other methods. Our camera frustum based point projection is more robust compared to z-buffer based projection [1] when the point cloud is sparse, which is especially true on large scenes from Matterport3D. As a result, our renders are more complete and reflect correct occlusion.

3.4. Video for Temporal Consistency

To demonstrate the temporal coherence, we provide video sequences on the same trajectories from NPG and our method. We compare video temporal consistency synthesized by different methods (e.g. ours, pix2pix, neural point based graphic, direct render.) in different scenes.

References

[1] Kara-Ali Aliev, Dmitry Ulyanov, and Victor Lempitsky. Neural point-based graphics. arXiv preprint arXiv:1906.08240, 2019.
[2] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. arXiv preprint arXiv:1709.06158, 2017.
[3] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In Proc. CVPR, pages 5828–5839, 2017.
| Layers   | kernel size | stride | dilation | channel_in | channel_out | actv   | inputs                        |
|----------|-------------|--------|----------|------------|-------------|--------|-------------------------------|
| conv0    | 1×1×1       | 1×1×1  | 1        | 11         | 16          | lrelu  | neural point features         |
| conv1    | 3×3×3       | 1×1×1  | 1        | 16         | 32          | lrelu  | conv0                         |
| maxpool1 | 2×2×2       | 2×2×2  | -        | 32         | 32          | -      | conv1                         |
| conv2    | 3×3×3       | 1×1×1  | 1        | 32         | 32          | lrelu  | maxpool1                      |
| maxpool2 | 2×1×1       | 2×1×1  | -        | 32         | 32          | -      | conv2                         |
| conv3    | 3×3×3       | 1×1×1  | 1        | 32         | 64          | lrelu  | maxpool2                      |
| maxpool3 | 2×2×2       | 2×2×2  | -        | 64         | 64          | -      | conv3                         |
| conv4    | 3×3×3       | 1×1×1  | 1        | 64         | 64          | lrelu  | maxpool3                      |
| maxpool4 | 2×1×1       | 2×1×1  | -        | 64         | 64          | -      | conv4                         |
| conv5    | 3×3×3       | 1×1×1  | 1        | 64         | 128         | lrelu  | maxpool4                      |
| maxpool5 | 2×2×2       | 2×2×2  | -        | 128        | 128         | -      | conv5                         |
| conv6    | 1×3×3       | 1×1×1  | 2        | 128        | 128         | lrelu  | maxpool5                      |
| up1      | 2×2×2       | 2×2×2  | 1        | 128        | 128         | -      | conv6                         |
| conv7    | 3×3×3       | 1×1×1  | 1        | 128        | 128         | lrelu  | up1+conv5                     |
| up2      | 2×1×1       | 2×1×1  | 1        | 128        | 64          | -      | conv7                         |
| conv8    | 3×3×3       | 1×1×1  | 1        | 128        | 64          | lrelu  | up2+conv4                     |
| up3      | 2×2×2       | 2×2×2  | 1        | 64         | 64          | -      | conv8                         |
| conv9    | 3×3×3       | 1×1×1  | 1        | 128        | 64          | lrelu  | up3+conv3                     |
| up4      | 2×1×1       | 2×1×1  | 1        | 64         | 32          | -      | conv9                         |
| conv10   | 3×3×3       | 1×1×1  | 1        | 64         | 32          | lrelu  | up4+conv2                     |
| up5      | 2×2×2       | 2×2×2  | 1        | 32         | 32          | -      | conv10                        |
| conv11   | 3×3×3       | 1×1×1  | 1        | 64         | 32          | lrelu  | up5+conv1                     |
| rgbs     | 1×1×1       | 1×1×1  | 1        | 32         | 3           | -      | conv11                        |
| blend weights | 1×1×1 | 1×1×1  | 1        | 32         | 1           | softmax | conv11                       |

Table 1. Details of network architecture, where **actv** is the activation function. And **up1-5** are transposed convolutions.
Figure 1. One example of multi-plane RGB images and blending weights. Left to right, up to down.
Figure 2. More results on ScanNet. Our results are better than other methods, especially at the location with sparsity and occlusion. Zoom in for details.
Figure 3. More results on ScanNet. Our results are better than other methods, especially at the location with sparsity and occlusion. Zoom in for details.
Figure 4. More results on Matterport3D. Our method obtains higher performance than other baselines on this challenging dataset.
Figure 5. More results on Matterport3D. Our method obtains higher performance than other baselines on this challenging dataset.