Neural Point Cloud Rendering via Multi-Plane Projection

Peng Dai 1* Yinda Zhang 2* Zhuwen Li 3* Shuaicheng Liu 1† Bing Zeng 1

1University of Electronic Science and Technology of China 2Google Research 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 planes of images w.r.t. the space division in the depth directions. The multi-plane 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 [9], scene textures [3], materials [44], illuminations [47], 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 [13] and radiance simulation [45].

Image based rendering (IBR) techniques [16, 29, 32], alternatively, try to render a novel view based on the given images and their approximated scene geometries through image warping [30] and image inpainting [23]. 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) [28, 18, 17] simplify the scene structures by replacing the surface mesh with point cloud or surfels [35], such that the heavy geometry constructions can be avoid.

On the other hand, many deep learning approaches show strong capability in inpainting [31], refining [36], and even constructing images from only a few indications [34, 2]. 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 [48].

Recently, combining advantages of simplified geometry representation and neural capabilities become a new trend, yielding neural rendering methods [4, 7, 20, 20, 33]. It directly learns to render end-to-end, which bypass complicated intermediate representations. Previous work used mostly 3D volume as representation [39]. However, the memory complexity of 3D volume is cubic, and thus these
approaches are not scalable and usually only work for small 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 [46] and camera calibrations [49], which interfere the visibility check when projected onto 2D image plane and results in jittering 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. [1] 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 [19], 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 planes 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 [8] and Matterport 3D [6], 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 performs reasonably well given relatively sparse point cloud.

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

- 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.

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 [19], surface reconstruction and meshing [43]. When performance is preferred, ray tracing [13] is used to simulate the transmission of light in the space, such that can better interact with the environment, e.g., geometry [40], material [10], BRDF [14], lighting [22], 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) [16, 29, 32, 38] aims to produce a novel view from given images through warping [30] and blending [10], which is a computationally efficient approach compared with classical rendering pipeline. Multiple view geometry [19] is applied for camera parameter estimation or some variant that bypasses the 3D reconstruction [40], such as adopting epipolar constraints [15]. Recently, deep learning has been proved to be more effective in replacing the warping and blending of traditional approaches in the IBR pipeline [12, 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 [26].

2.3. Deep image synthesis

Deep methods for 2D image synthesize has achieved very promising results, such as autoencoders [21], PixelCNN [42] and image-to-image translation [24]. The most exciting results are produced based on generative adversarial networks [51, 34]. Most of the generators adopt encoder-decode architecture with skip connections to facilitate feature propagation [37]. 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 [41]. Nvidia [5] 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 [39] and Neural Point Based Graphics(NPG) [1]. 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 [1]; it also proposes to render images from 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
3.1. Overview

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-plane based voxelization and multi-plane rendering. The multi-plane 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 planes. The voxels then aggregates features of points inside it with geometric rules. The 3D feature volume is then fed into the multi-plane rendering module, which is a 3D CNN, to generates one color image plus a blending weight per plane on the depth dimension of the volume. The final output is a weighted blending of multi-plane 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.

3.2. Learnable point cloud features

Our input is a 3D point cloud representation of the scene. To be sufficient for rendering, each 3D point should contains 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 [1, 41], 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 [1]. Note that this feature is not trainable as the point position.

3.3. Multi-plane 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. [1] 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 features available in the camera frustum thanks to the 3D volume. To achieve sub-voxel performance, we vote each aggregate points in one frustum voxel, we calculate two kinds of distance. $D_{1}(p,h,w)$ is the distance between points and pixel 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.

Figure 3. 3D space voxelization and aggregation. (a) Multi-plane voxelization. Point clouds under camera view are restricted into a frustum region specified by the minimum and maximum depth planes, and such region can be uniformly split into small frustum voxels according to image size($H,W$) and predefined number of planes $P$, where $l$ indicates length of frustum. (b) Aggregation. 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 pixel 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.

$$F_{(p,h,w)} = \frac{\sum w_{i}^{2}(p,h,w) \times F_{(p,h,w)}}{\sum w_{i}^{2}(p,h,w)},$$  

(1)

$$w_{i}^{2}(p,h,w) = (1-D_{1}^{2}(p,h,w))^{a} \times \frac{1}{1 + D_{2}^{2}(p,h,w)}^{b},$$  

(2)

where $F_{i}$ is the feature of the $i$th point in a voxel, and $w_{i}^{2}(p,h,w)$ is the blending weight of point $i$ to voxel $(p,h,w)$. $D_{1} \in (0,1/\sqrt{2})$ is the distance between the point projection on image to the corresponding pixel center of voxel $(p,h,w)$, and $D_{2} \in (0,l)$ is the depth difference between point and minimum depth point in this voxel. Parameters $a$ and $b$ control the blending weight on direction parallel and vertical to the image plane. When $b \to +\infty$, it turns into Aliev et al. [1] where a point is picked via z-buffer. The motivation of Eq. 2 is to assign larger weights to points closer to camera or pixel center. We found it work well empirically. Other formulations reflecting similar property could also work properly.

3.4. Multi-plane rendering

We adopt the U-Net-like [37] 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-plane RGB images plus their blending weights. The final output image $I$ is obtained by

$$I = \sum_{p} I_{p} \cdot \alpha_{p},$$  

(3)

where $p$ indicates a plane, and $I_{p}$ and $\alpha_{p}$ are the corresponding plane predictions. Please refer to supplementary material for more details of the network architecture.

3.5. Loss function

To measure the difference between network prediction and the ground truth image, we use perceptual loss [7, 25], 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 [11]. The perceptual loss is defined as a weighted sum of the $\ell_{1}$ loss on each feature maps. Specifically, the loss function is defined as follows:

$$L(P_{f}, \theta) = \sum_{l} \lambda_{l} ||\Phi_{l}(I_{g}) - \Phi_{l}(f(P_{f}; \theta))||_{1},$$  

(4)

where $P_{f}$ represents point features, $I_{g}$ is ground-truth image, $\theta$ is network parameters, $f$ is our point cloud renderer, $\phi_{l}$ is a set of VGG-19 layers and $\lambda_{l}$ is weight used to balance different layers.

3.6. Feature optimization

Inspired by Thies et al. [41] and Aliev et al. [1], 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 $-\nabla_{l} \times w_{i}^{2}(p,h,w) \times \theta_{(p,h,w)}$, where $\theta_{(p,h,w)}$ is gradient derived from the loss function and $l_{i}$ 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 [8] contains RGBD scans of indoor environments. We follow the training and testing split of Aliev et al. [1]. 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 on average.

Matterport 3D [6] contains RGBD panoramas captured at multiple locations in indoor scenes. Each panorama is composed of 18 regular RGBD images viewing toward different directions. For each scene, we randomly pick 1/100 of the views for testing and leave the others for training. Note that overall this dataset is more challenging due to the sparse point cloud and large variation of camera viewpoints.

4.2. Data preparation and training details

For each scene, our network is trained for 21 epochs over 1,925 images on average, using Adam optimizer [27]. 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, and \( \lambda_1 \) in Eq. 4 following Chen et al. [7]. The training process is accomplished on one GeForce 1080 Ti, which takes on average 41.5 hours per scene.

4.3. Rendering results

We first compare our method to two competitors, Neural Point-based Graphic (NPG) [1] and Pix2Pix [24]. 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 [48]. Since the authors of [1] 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 [24]. 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.

**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’. It is observed that ours with learnt features outperforms the direct render approach on all metrics.

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) [50]. 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 ScanNet, 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.

![Figure 4. Advantage of neural descriptor. (a) displays results generated by directly rendering. (b) Ours results using neural descriptor. Without the assistant of neural descriptor, the final point clouds rendering results are blurry. Please zoom in for details.](image-url)
---|---|---
| Methods | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ |
| pix2pix [24] | 19.247 | 0.731 | 0.429 | 14.964 | 0.530 | 0.675 |
| Neural point based graphic [1] | 22.911 | 0.840 | 0.245 | 17.931 | 0.622 | 0.597 |
| Ours + direct render | 22.259 | 0.818 | 0.290 | 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.

Figure 5. Rendering quality on ScanNet and Matterport 3D datasets. We compare our proposed method with pix2pix and neural point based graphic on two indoor scene datasets. By analyzing the generated image in novel view, our proposed method achieves better performance. Please zoom in for details.

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 labtop 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
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 [1] 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.

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 frustum volume and allows network to infer correctly. Fig. 6 shows two comparisons on cases with noisy depth. NPG and Pix2Pix either completely miss the correct objects or produce a mixture of foreground and background.

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 in-complete 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 imcomplete 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 relatives 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.
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.

We perform a user study to compare the temporal consistency against NPG, Pix2Pix and direct render. We render 4 videos of 4 different scenes with respect to each method. During user study, each time, we present 4 videos of 4 approaches and ask the subject to pick the best one. As we have 4 scenes, a user will pick 4 times. 20 users are invited, accumulating to 80 picks in total. The participants are required to only judge the temporal consistency. Statistical results are shown in Fig. 9. Our method received 61 picks, which indicates our video is apparently better than other methods in terms of temporal consistency. Please refer to the supplementary files for these videos.

5. Conclusion

In this work, we have proposed 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. Optical flow can be utilized for enforcing temporal consistency given additional observations. Flickers can be removed by enforcing constraints on similar predictions from shared points. In addition, applying interpolation in different depth planes could further improve the robustness against sparse point clouds.

Acknowledgement: This research was supported in part by National Natural Science Foundation of China (NSFC, No.61872067, No.61720106004), in part by Research Programs of Science and Technology in Sichuan Province (No.2019YFH0016).
References


