A Review of Point-NeRF

Julián Alvarez de Giorgi
Institut Polytechnique de Paris
Palaiseau, France
julian.alvarez@telecom-paris.fr

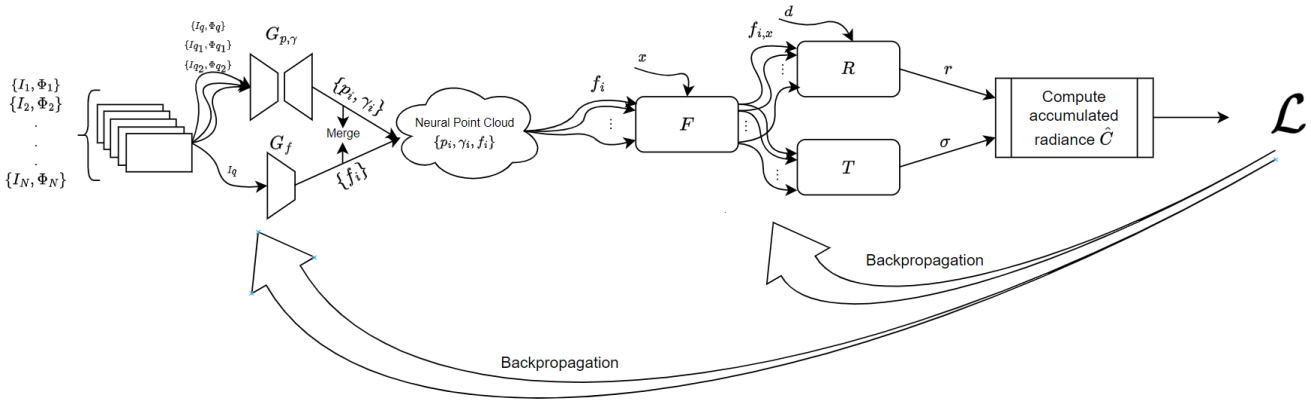

Figure 1. Conceptual diagram of the model structure.

Abstract

This is a report of Point-NeRF [11], was carried out as a final project of the NPM3D (Nuages de Points et Modélisation 3D) course of the MVA Master of the ENS Paris-Saclay. The goal is to have a complete understanding of how Point-NeRF works and to study its capabilities and limitations by exploring new data.

Keywords: Neural Radiance Field, Point-NeRF, Computer Vision Survey, Novel View Synthesis, Neural Rendering, Volume Rendering, 3D Reconstruction

1 Introduction

Point-NeRF (Point-based Neural Radiance Fields) is a variation of the original Neural Radiance Fields (NeRF) model. Point-NeRF, instead of modeling the vector field with an MLP, it utilize a neural 3D point cloud with associated neural features to model intrinsically the radiance field. The use of point clouds enables to do a local approximation of the scene geometry, around the meaningful data, avoiding encoding the entire scene space. This representation can model more complex scene content than MLPs which have limited network capacity. Moreover, Point-NeRF takes advantage of deep multi-view models that can reconstruct a 3D scene fast. This allows us to initialize these reconstructions and optimize on the point cloud to improve the definition. The usage of these networks pre-trained on a diversity of scenes significantly improves the training time of point-NeRF with respect to NeRF, in the order of 3 times faster.

Furthermore, Point-NeRF efficiently renders scenes by aggregating neural point features near surfaces using a ray marching-based rendering pipeline, this "smart" sampling of points near the surface and not uniformly throughout space is a clear improvement over the base nerf approach.

The authors proposed a multi-view model based on a MVS-Net [3], and a 2D CNN architecture for the neural point cloud generation. They also provide the pre-trained model trained on DTU [4] dataset. However, the Poin-Nerf approach is agnostic to the model choice, the authors show that it behaves well also with COLMAP [7], regardless of the sparsity of the COLMAP point clouds.

In section 2, NeRF rendering technique with ray marching is explained, and then in section 3 the point cloud synthesis is detailed, further in section 4, the regression from the point cloud is drawn, meanwhile in section 5 the training and optimization steps are shown, later on section 6 some original results are shown, to then discuss in section 7, and finally in section 8 the Conclusions are drawn.

2 NeRF rendering, ray marching.

First, let's introduce the rendering approach presented in NeRF. As mentioned, in NeRF, the scene is represented as a vector field, learned by an MLP. Given a point 3D point $\mathbf{x}$ in the scene, and a viewing direction $\mathbf{d}$ (represented by a 3D unitary vector), the MLP outputs a density σ, and a radiance

r, i.e.:

$$\sigma, \mathbf{r} = MLP(\gamma_{10}(\mathbf{x}), \gamma_4(\mathbf{d}))^1 \quad (1)$$

How σ must only depend on the evaluation point **x**, and not on the viewing direction, the MLP is actually divided into 2 MLPs where the first one only takes the point **x** and outputs the σ along with a feature vector, which is then concatenated to **d** and passed as input to the second MLP which will output the radiance **r**. [2]

The scene is then reconstructed with what is called ray marching. Basically, a virtual ray is launched from the focal point, through the pixel, and into the scene, (this can be precisely calculated using the camera parameters) samples shading points $\{x_j, j = 1, ..., M\}$, along the ray, collecting its respective radiance and density $\{(\sigma_j, r_j), i = 1, ..., M\}$, for each image pixel. From it, we calculate the accumulated radiance $\hat{C}$ as follows:

$$\hat{C} = \sum_M \tau_j (1 - e^{-\sigma_j \Delta_j}) r_j$$
$$\tau_j = e^{-\sum_{t=1}^{j-1} \sigma_t \Delta_t} \quad (2)$$

This accumulated radiance is an estimator of the value of the respective pixel. τ_j represents the volume transmittance up to point j, which stands for the percentage of the light transmitted over the incident light. Δ_t is the distance between adjacency shading points.

Finally, the rendering loss is the following:

$$\mathcal{L}_{render} = \sum_{p \in \mathcal{R}} ||\hat{C}_p - C_p||_2^2 \quad (3)$$

Where p is the ray of each pixel in $\mathcal{R}$ which is the batch, typically one image. C_p is the ground truth, the value of the pixel.

NeRF implements a hierarchical sampling of the shading points over the ray, it first samples uniformly the shading points, equidistant over all the space, and then with a representation of the PDF distribution of the transmittance, carries a second sampling on this distribution, allocating more samples near surfaces. Point-NeRF on the other hand, does direct an smart sampling of the shading points, focusing on the space near the surface, this is a huge advantage over NeRF.

3 Nueral Point Cloud.

As explained in the introduction, first an initialization of the neural point cloud is made, this could be with the proposed method, explained in 3.2, or with any MVS method that outputs a point cloud. Then, the neural point cloud is optimized to achieve a detailed scene representation and achieve high-quality rendering.

3.1 Neural Point Cloud Definition

As explained earlier, the Radiance Field is intrinsically coded in the neural point cloud. This cloud is defined as follows, $P = \{(p_i, f_i, \gamma_i), i = 1, ..., N\}$, where p_i is the 3D position of the point i, f_i is its neural feature vector, and $\gamma_i \in [0, 1]$ represents how likely is the point to be near a surface of the scene (where the information is).

3.2 Neural Point Cloud Synthesis

The goal is to have a good initialization of the Neural Point Cloud from the set of images and its parameters $\{I_q, \Phi_q\}$. To do so, the authors divided in 2 networks the work, a first MVSNet [3] based network $G_{p,\gamma}$, which to give the point confidence γ_i and the point locations p_i, meanwhile a 2D CNN G_f will predict the point features f_i. For each image, a point cloud is created from both networks, at the end all point clouds are merged to create the final neural point cloud. To better understand refer to the Fig. 2

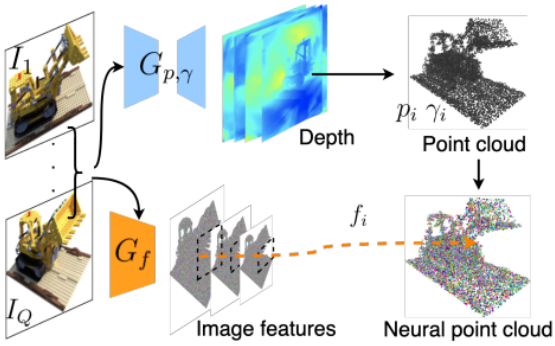

Figure 2. Conceptual diagram of the Neural Point Cloud Synthesis process. The image was taken from the original Point-Nerf paper [11]

3.2.1 Point Location and Confidence. The MSVNet-based network network takes an imput image I_q with its camara parameters Φ_q and outputs a point cloud for each image. 3D points are generated by a cost-volume based 3D CNN, it builds a plane-swept by wraping 2D image features from neighboring views $I_{q_1}, \Phi_{q_1}, I_{q_2}, \Phi_{q_2}, ..$, according to the camera pose. Typically 2 neighboring viewpoints. The variance is computed and passed through a U-Net [6] architecture from which a depth probability volume is regressed using. A depth map is computed by linearly combining per-plane depth values weighted by their corresponding probabilities. The depth map is finally unprojected to 3D space to obtain a

[1] $\gamma_L(x) = (sin(2^0\pi x), cos(2^0\pi x), ..., sin(2^{L-1}\pi x), cos(2^{L-1}\pi x))$ is a positional encoding, which maps the inputs to a higher dimension using high-frequency functions, this enable the MLP to easily leverage high-frequency variations on the imput

[2] In the original paper of NeRF [5], they called the output of the second MLP the view-dependent RGB color, which they noted as **c**. However, in Point-NeRF [11], they have called it the view-dependent radiance, denoted by **r**, this is why I've chosen to keep this notation to be consistent.

point cloud $\{p_1, ..., p_{N_q}\}$ for each viewpoint q. To compute the confidence γ_i at a specific point p_i in the scene, tri-linear sampling is employed to interpolate the probabilities from the neighboring voxels in the depth probability volume. This is resumed as:

$$\{p_i, \gamma_i\} = G_{p,\gamma}(I_q, \Phi_q, I_{q_1}, \Phi_{q_1}, I_{q_2}, \Phi_{q_2}, ...) \tag{4}$$

3.2.2 Point Features. To predict the point features, a 2D CNN was implemented, a VGG [8] architecture with three downsampling layers. It takes an image I_q and extract a 2D feature map for each resolution, which are aligned with the points of the point clouds from $G_{p,\gamma}$, predicting like that the point features f_i.

$$\{f_i\} = G_f(I_q) \tag{5}$$

3.2.3 Merging Neural Point Clouds. With the presented networks we created a neural point cloud for each image I_q. Finally, we merged all of the pointclouds to build the final neural point cloud representation of the scene.

4 Point-NeRF Regression.

The rendering of Point-Nerf is the same as NeRF, explained in section 2. We need now a method to extract the radiance r and density σ from the neural point cloud. To do so, for each shading point only the 8 nearest neural points neighbors are taken into account. An efficiently query neural point neighbors inspired by the CAGQ [10] point query is implemented, this allows to place the shading points in the region of interest (high density of points) and like that avoid sampling the empty spaces. This is a huge advantage of Point-Nerf, available thanks to the usage of neural point clouds, which offer a good prior of the scene geometry. This process is abstracted as:

$$(\sigma, r) = Point - NeRF(x, d, p_1, f_1, \gamma_1, ..., p_8 f_8, \gamma_8) \tag{6}$$

For this propose, three different MLPs are implemented, a per-point feature processor MLP F, a view-dependent radiance regress MLP R, and a density regress MLP T.

4.1 Per-Point feature processor MLP F

This MLP takes the relative position of the neural point $x - p_i$ and it's feature vector f_i and outputs a feature vector relative to the shading point $f_{i,x}$ of dimension 128. The realtive position makes the network invariant to translation, which is more suited for the task. This is synthetised as:

$$f_{i,x} = F(f_i, x - p_i)^3 \tag{7}$$

[3] As in the classical NeRF a high-frequency positional encoding was implemented, here it also implemented on the point features f_i and the relative distance $x - p_i$.

Those features compress all the 3D information of the scene locally around the shading point, and will allow us to regress from them the radiance and density at the shading point.

4.2 View-dependent radiance regress MLP R

A single feature is computed doing an aggragation of the point features as follows:

$$f_x = \sum_i \gamma_i \frac{\omega_i}{\sum \omega_i} f_{i,x}, \quad where\ \omega_i = \frac{1}{||p_i - x||} \tag{8}$$

This gives more importance to closer points and to points that are more likely to be near a scene surface (γ_i). Then, as before, a high-frecuency positional encoding is applied to the viewing direction d to embedded as a feature of dimension 24, this is passed with the feature f_x to the MLP to regress the radiance:

$$r = R(f_x, d) \tag{9}$$

4.3 Density Regress MLP T

Here we follow the same path as for the radiance, we compute the density for each neural point, and then do an aggregation to compute the shading point density;

$$\begin{aligned} &\sigma_i = T(f_{i,x}) \\ &\sigma = \sum_i \sigma_i \gamma_i \frac{\omega_i}{\sum \omega_i}, \quad where\ \omega_i = \frac{1}{||p_i - x||} \end{aligned} \tag{10}$$

5 Training

As in NeRF, we use the same L2 rendering loss presented in 2. This allows end-to-end training, optimizing all the networks at the same time. Anyway, they've done it smarter than that and divided the training into 3 steps, an MVS-Net training step, an end-to-end training step, and finally a per-scene optimization stage. All the training steps were done on the DTU [4] dataset.

5.1 MVS-Net training.

First, the MVS-Net was trained alone. The DTU dataset provides the ground truth point cloud with normal information, Screened Poisson Surface Reconstruction (SPSR) was used to generate the mesh surface and then render the mesh to each viewpoint, generation the depth maps for ground truth.

5.2 End-To-End training.

In this section we train the full pipeline end-to-end with the L2 rendering loss using ADAM optimizer, creating the render views as explained in the section 2.

5.3 Per-Scene Optimization.

This is the most important step, the goal is to build a neural point cloud that compresses well all the scene information

and trains the rendering networks to well-retrieve the radiance and density from it. Although the initial point clouds gives fairly good results, is not sufficient for achieving high-quality rendering. The point clouds could contain holes and many outliers, especially if we are using another model as COLMAP. Hence, two additional techniques were added to the Per-scene optimization to overcome this issue. Those are point pruning and point growing.

At this stage, an additional sparsity loss over γ_i is used to enforce probabilities being close to 1 or 0.

$$\mathcal{L}_{sparse} = \frac{1}{|\gamma|} \sum_{\gamma_i} log(\gamma_i) + log(1 - \gamma_i) \tag{11}$$

Then, the whole loss in this step is the following:

$$\mathcal{L} = \mathcal{L}_{render} + a\mathcal{L}_{s}parse \tag{12}$$

They have used $a = e^{-3}$.

5.3.1 Point-Pruning. To eliminate artifacts and outliers, every point with $\gamma_i < 0.1$ is eliminated every 10K iterations.

5.3.2 Point-Growing. The goal is to cover missing parts of the scenes, this issue appears frequently on COLMAP and on big texturized regions. To do so we have to recover the missing information, this is done by imputing new neural points. On the ray marching algorithm, we take the shading point with the highest opacity along the ray x_{j_g}:

$$\alpha_i = (1 - e^{-\sigma_j \Delta_j}) \quad j_g = argmax_j \alpha_i$$

α_i is the opacity of the shading point, and ϵ_i is the distance to the nearest neural point. The shading point x_{j_g} is added to the neural point cloud if $\alpha_{j_g} > T_{opacity}$ and $\epsilon_{j_g} > T_{dist}$. These grantees that the new neural point is near a surface (high opacity) and is not that close to other neural points (we don't care about zones already densely sampled). As point-pruning, this is carried out each 10K iterations.

6 Results

Unfortunately, due to the limited access to GPU, and time constraints, I couldn't test the model. How I couldn't test the model on my own, to see qualitatively the improvement over NeRF, I'm going to show the images shown on the original paper.

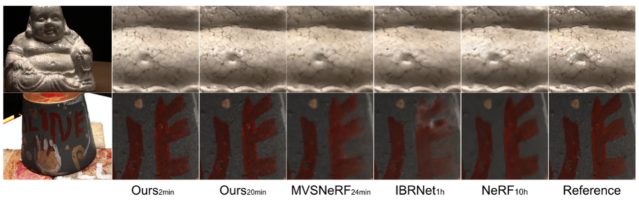

Figure 3. Comparison of different render methods and Point-NeRF (Ours). The image was taken from the original Point-Nerf paper [11]

In figure 3 we can confirm that the Point-Nerf outperforms the other method and is much faster.

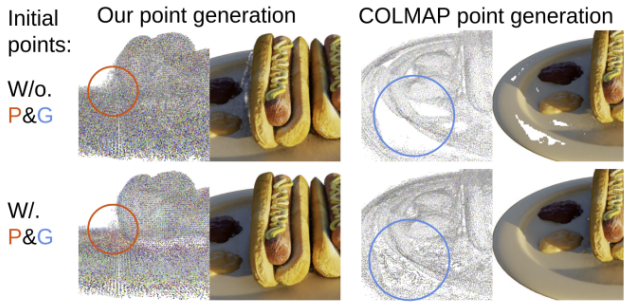

Figure 4. Comparison of point clouds and render results of both methods the MVS develop on the original paper and COLMAP, also the effect of pruning and growing on the per-scene optimization step is shown. The image was taken from the original Point-Nerf paper [11]

From Fig. 4, we see how the MVS model developed by the authors outperforms COLMAP and builds a better initialization. However, the most important here is to see the effect of the pruning and growing on the per-scene optimization step eliminates the defects and we end up with similar rendering results, no matter the initialization.

7 Discussion

I think the key idea of Point-NeRF is quitting to the MLP intrinsic representation of the scene, and adding a middle step by encoding the scene in a neural point cloud. This permits subdividing the problem into specified networks, the neural point cloud generators $G_{p,\gamma}$ and G_f, and the radiance regress networks F, R, and T. This change represents a substantial change, allowing a kind of transfer learning, pre-training either the cloud generator networks or incorporating other pre-trained models. This greatly accelerates the optimization time per scene, because it is not the same to learn a 3D scene if there is already a general knowledge of what a scene is. In the other case, we must first learn about the world, what a scene is, what each coordinate represents, and then focus on the specific scene.

The use of point clouds as an intermediate representation of the scene constitutes many advantages in terms of optimization. Point clouds adapt well to details, allowing to obtain high-frequency information in the scene, and their plasticity allows to model the scene locally and avoid empty areas. The use of point clouds also made it possible to study a point in the scene locally, using only a neighborhood of points.

Furthermore, as I mentioned before, point-NeRF is agnostic to the generative model of point cloud, allowing improvements with the use of better models. In addition, per-scene

optimization can be used as a method to improve the quality of the point cloud.

7.1 Possible improvements

A straightforward and simple time-saving improvement is to implement ray termination. Ray termination cuts off the ray when we reach a region where it is very opaque (a threshold has to be determined).

Another possible change I can think of is to pass the neural point features through a GAT-inspired attention layer, first presented in [9] and improved in [1]. What this will do is transform the neural features to be more aware of their neighbors. If we have the neural point features $\{f_1, f_2, ..., f_n\}$, we would transform them to $\{f'_1, f'_2, ..., f'_n\}$ as follows:

$$\alpha_{ij} = softmax(LeakyReLU(\mathbf{a}^T.[\mathbf{W}f_i||\mathbf{W}f_j])) \quad (13)$$

Where $\mathbf{a}$ and $\mathbf{W}$ are learned, α_{ij} are the attention coefficients, used to do the weighted average of the neural features:

$$f'_i = \sigma\left(\sum_j \alpha_{ij}\mathbf{W}f_j\right) \quad (14)$$

The attention layer could be used before F or after. I think this can help to synthesize better the local information of that area, I find it interesting to try.

8 Conclusion

The operation of Point-NeRF was correctly detailed, all its advantages and some hypothesizing about possible improvements were explained.

However, the project is incomplete. Due to several difficulties with the installation of all the necessary packages and later problems with GPU access, it was impossible for me to test Point-Nerf. The original idea was to test it, then test it with a new nerf-synthetic scene, and finally see how it behaved with other MVS-images to point-clouds networks like COLMAP [7] or MESHROOM [2].

References

[1] Shaked Brody, Uri Alon, and Eran Yahav. 2021. How Attentive are Graph Attention Networks? *CoRR* abs/2105.14491 (2021). arXiv:2105.14491 https://arxiv.org/abs/2105.14491

[2] Carsten Griwodz, Simone Gasparini, Lilian Calvet, Pierre Gurdjos, Fabien Castan, Benoit Maujean, Gregoire De Lillo, and Yann Lanthony. 2021. AliceVision Meshroom: An open-source 3D reconstruction pipeline. In *Proceedings of the 12th ACM Multimedia Systems Conference - MMSys '21*. ACM Press. https://doi.org/10.1145/3458305.3478443

[3] Po-Han Huang, Kevin Matzen, Johannes Kopf, Narendra Ahuja, and Jia-Bin Huang. 2018. DeepMVS: Learning Multi-view Stereopsis. *CoRR* abs/1804.00650 (2018). arXiv:1804.00650 http://arxiv.org/abs/1804.00650

[4] Rasmus Jensen, Anders Dahl, George Vogiatzis, Engil Tola, and Henrik Aanæs. 2014. Large Scale Multi-view Stereopsis Evaluation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 406–413. https://doi.org/10.1109/CVPR.2014.59

[5] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. *CoRR* abs/2003.08934 (2020). arXiv:2003.08934 https://arxiv.org/abs/2003.08934

[6] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. *CoRR* abs/1505.04597 (2015). arXiv:1505.04597 http://arxiv.org/abs/1505.04597

[7] Johannes L. Schönberger, Enliang Zheng, Jan-Michael Frahm, and Marc Pollefeys. 2016. Pixelwise View Selection for Unstructured Multi-View Stereo. In *Computer Vision – ECCV 2016*, Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (Eds.). Springer International Publishing, Cham, 501–518.

[8] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv e-prints*, Article arXiv:1409.1556 (Sept. 2014), arXiv:1409.1556 pages. https://doi.org/10.48550/arXiv.1409.1556 arXiv:1409.1556 [cs.CV]

[9] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. arXiv:1710.10903 [stat.ML]

[10] Qiangeng Xu, Xudong Sun, Cho-Ying Wu, Panqu Wang, and Ulrich Neumann. 2019. Grid-GCN for Fast and Scalable Point Cloud Learning. *CoRR* abs/1912.02984 (2019). arXiv:1912.02984 http://arxiv.org/abs/1912.02984

[11] Qiangeng Xu, Zexiang Xu, Julien Philip, Sai Bi, Zhixin Shu, Kalyan Sunkavalli, and Ulrich Neumann. 2022. Point-nerf: Point-based neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5438–5448.

A Online Resources

The code is available at https://github.com/JulianAlvarezdeGiorgi/Point-NeRF.

Received 3 February 2024