POINTS2NERF: GENERATING NEURAL RADIANCE FIELDS FROM 3D POINT CLOUD

Anonymous authors
Paper under double-blind review

ABSTRACT

Neural Radiance Fields (NeRFs) offer a state-of-the-art quality in synthesising novel views of complex 3D scenes from a small subset of base images. For NeRFs to perform optimally, the registration of base images has to follow certain assumptions, including maintaining constant distance between the camera and the object. We can address this limitation by training NeRFs with 3D point clouds, instead of images, yet a straightforward substitution is impossible due to the sparsity of 3D clouds in the under-sampled regions which leads to incomplete reconstructions output by NeRFs. To solve this problem, here we propose an auto-encoder-based architecture that leverages a hypernetwork paradigm to transfer 3D points with the associated color values through a lower-dimensional latent space and generate weights of NeRF model. This way we are able to accommodate sparsity of 3D point clouds and fully exploit the potential of point cloud data. As a side benefit, our method offers an implicit way for representing 3D scenes and objects, that can be employed to condition NeRFs and hence generalize the models beyond objects seen during training. Empirical evaluation confirms the advantages of our method over conventional NeRFs and proves its superiority in practical applications.

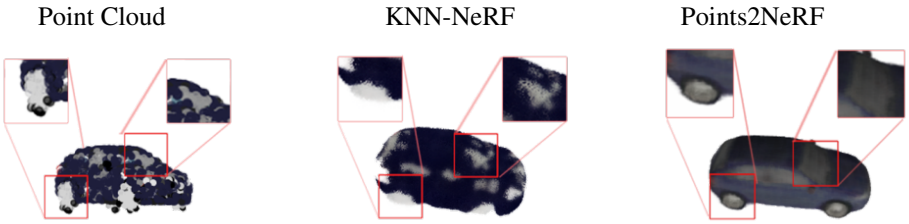

Figure 1: Conversely to other approaches, our Points2NeRF method takes a 3D point cloud with colors as an input instead of 2D images. A straightforward substitution of the input data in the existing architectures is not feasible due to the sparsity of point cloud representation (**left** image), that yields artefacts in reconstruction (baseline KNN-NeRF result in the **middle**). By extending an auto-encoder architecture with the hypernetwork paradigm, our Points2NeRF architecture is able to build internal representation of complete 3D objects which enhances the rendering quality of NeRFs (**right** image).

1 INTRODUCTION

Neural Radiance Fields (NeRFs) (Mildenhall et al., 2020) enable synthesizing novel views of complex scenes from a few 2D images with known camera positions. Based on the relations between those base images and computer graphics principles, such as ray tracing, this neural network model can render high-quality images of the scene from previously unseen viewpoints. Although, in recent years much effort was invested in improving the quality of the resulting views (Kosiorek et al., 2021) and the controllability of NeRFs (Kania et al., 2022), the robustness of those methods against various data registration challenges remains largely unchartered research area. For instance, to render high-quality views, the base images must be captured from a similar distance to the captured object, and the corresponding camera positions need to be approximately known. These practical constraints limit the

Input 3D Point Cloud Representation obtained by Points2NeRF

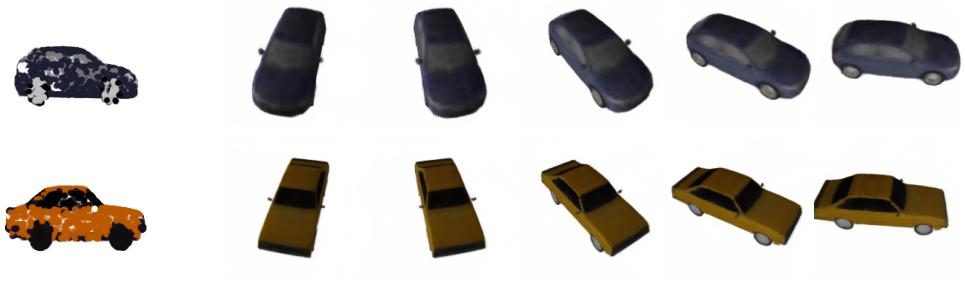

Figure 2: Our Points2NeRF approach takes a 3D point cloud with the associated color values and returns the weights of a NeRF network that reconstructs 3D objects with high fidelity and coherent coloring.

applicability of NeRFs across various applications, such as in robots deployed within noisy industrial environments.

We can address these limitations by using increasingly popular 3D capturing devices, such as LIDARs and depth cameras, to produce 3D point clouds and by feeding those point clouds to NeRFs. However, a straightforward substitution of 2D images with 3D point clouds does not produce novel views of the comparable quality. This is mainly because of the sparsity of point cloud data in the under-sampled parts of objects, *e.g.* willows and windows of a car presented in Fig. 1. As a result, the baseline solution, dubbed KNN-NeRF and described in details in Sec. 2, does not render sharp images.

To solve this problem, we propose in this work an auto-encoder-based architecture that transfers 3D point clouds through a low-dimensional latent space and outputs weights of a NeRF model. Our Points2NeRF[1] approach embodies hypernetwork paradigm as we take a 3D point cloud with the associated color values as an input and return the parameters of a target NeRF neural network. Since the implicit representation of the captured objects are built within the latent space, missing data points in under-sampled regions do not prohibit the resulting target network from synthesising high-quality views. Furthermore, during the training, our model can process multiple classes of point cloud objects, thus yielding a solution that is much more robust. In fact, the NeRF network parameters can be interpreted as a continuous parametrisation of a 3D object space. Thus, this formulation allows for conditioning NeRFs and generalizing our solution beyond 3D object classes seen during training.

To summarize, the contributions of our work are the following:

- We propose a new method dubbed Points2NeRF which adapts a hypernetwork framework to the NeRF architecture and hence allows to produce Radiance Fields from 3D point cloud.
- Our approach enables conditioning NeRFs, which, in turn, allows us to generalize the model beyond 3D objects seen in training.
- Lastly, our method offers a generative model that can represent 3D objects as NeRF parameters in a continuous manner, enabling interpolation within the object space.

2 Training NeRF based on 3D point cloud

NeRFs (Mildenhall et al., 2020) represent a scene using a fully-connected architecture. As the input, NeRF takes a 5D coordinate (spatial location $\mathbf{x} = (x, y, z)$ and viewing direction $\mathbf{d} = (\theta, \psi)$) and it outputs an emitted color $\mathbf{c} = (r, g, b)$ and volume density σ.

NeRFs trained on images. A vanilla NeRF uses a set of images for training. In such a scenario, we produce many rays passing through the image and a 3D object represented by a neural network. NeRF approximates this 3D object with a MLP network:

$$F_\Theta : (\mathbf{x}, \mathbf{d}) \rightarrow (\mathbf{c}, \sigma),$$

[1]We make our implementation available at `https://github.com/...`

and optimizes its weights Θ to map each input 5D coordinate to its corresponding volume density and directional emitted color.

The loss of NeRF is inspired by classical volume rendering (Kajiya & Von Herzen, 1984). We render the color of all rays passing through the scene. The volume density $\sigma(\mathbf{x})$ can be interpreted as the differential probability of a ray. The expected color $C(\mathbf{r})$ of camera ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ (where $\mathbf{o}$ is ray origin and $\mathbf{d}$ is direction) can be computed with an integral.

In practice, this continuous integral is numerically estimated using a quadrature. We use a stratified sampling approach where we partition our ray $[t_n, t_f]$ into N evenly-spaced bins and then draw one sample uniformly at random from within each bin:

$$t_i \sim \mathcal{U}[t_n + \frac{i-1}{N}(t_f - t_n), t_n + \frac{i}{N}(t_f - t_n)].$$

We use these samples to estimate $C(\mathbf{r})$ with the quadrature rule discussed in the volume rendering review by Max (Max, 1995):

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i(1 - \exp(-\sigma_i\delta_i))\mathbf{c}_i, \text{ where } T(t) = \exp\left(-\sum_{j=1}^{i-1}\sigma_i\delta_i\right).$$

where $\delta_i = t_{i+1} - t_i$ is the distance between adjacent samples. This function for calculating $\hat{C}(\mathbf{r})$ from the set of (c_i, σ_i) values trivially differentiable.

We then use the volume rendering procedure to render the color of each ray from both sets of samples. Our loss is simply the total squared error between the rendered and true pixel colors

$$\mathcal{L} = \sum_{\mathbf{r}\in R} \|\hat{C}(\mathbf{r}) - C(\mathbf{r})\|_2^2 \tag{1}$$

where R is the set of rays in each batch, and $C(\mathbf{r})$, $\hat{C}(\mathbf{r})$ are the ground truth and predicted RGB colors for ray $\mathbf{r}$ respectively. Contrary to the baseline NeRF (Mildenhall et al., 2020), we use only a single architecture.

NeRFs trained on point clouds Such an approach can be easily modified to train NeRFs using a 3D point cloud. We take an input point cloud $X \subset \mathbb{R}^6$ where the first three elements encode the position while the last three encode an RGB color.

We can use only rays which cross points from a 3D point cloud. Let $\mathbf{x} = (x, y, z, r, g, b) \in X$ be a point from our point cloud with color. By $\mathbf{x_p} = (x, y, z)$ we denote the coordinates of the point and by $\mathbf{x_c} = (r, g, b)$ we denote color of the point $\mathbf{x} = (\mathbf{x}_p, \mathbf{x}_c)$. The ray going through point $\mathbf{x}_p$ from origin $\mathbf{o}$ is defined by $\mathbf{r}_{\mathbf{x}_p}(t) = \mathbf{o} + t\frac{\mathbf{x}_p - \mathbf{o}}{\|\mathbf{x}_p - \mathbf{o}\|}$. In such a case as a ground true color, we use the color of the point laid on the ray instead of the pixel from the image $C(\mathbf{r}_{\mathbf{x_p}}) = \mathbf{x}_c$. Our loss is simply the total squared error between the rendered and true point colors

$$\mathcal{L} = \sum_{(\mathbf{x}_p, \mathbf{x}_c)\in X} \|\hat{C}(\mathbf{r}_{\mathbf{x}_p}) - \mathbf{x}_c)\|_2^2. \tag{2}$$

Unfortunately, such an approach cannot reconstruct the correct image since we use only a fixed number of rays. In practice, we use only rays going through the existing 3D points. In particular, the model does not see the object borders and cannot reconstruct them sharply.

KNN approach to training NeRF on point cloud To solve this problem, we can produce many different rays which are not restricted to the points from a training data. In such a scenario, we do not cross any points from the point cloud. Therefore, as a ground truth color, we can use a color of the point which is closest to our ray. It is not trivial to find such an element since our ray goes through a 3D object and crosses the front and back of the object. So the point must fulfill two main constraints. It must be as close as possible to points on the ray and the origin of the ray.

We use the K nearest neighbor (KNN) algorithm to solve the problem. Let us consider point cloud $X \subset \mathbb{R}^6$ where the first three elements encode the position while the last three encode an RGB color

and ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ (where $\mathbf{o}$ is ray origin and $\mathbf{d}$ is direction). In the first step we find $k \in \mathbb{N}$ closest elements to the origin $KNN^k_{\mathbf{o}}(X)$. Thanks to this procedure, we obtain $KNN^k_{\mathbf{o}}(X)$, which contains elements from the surface of X situated on the form of the object (according to origin $\mathbf{o}$). In practice, k is hyperaparamete. In the second step, we look for the closest element to the ray $\mathbf{r}$:

$$C(X, \mathbf{r}) = \left\{\bar{\mathbf{x}} \in \mathbf{KNN^k_o(X)} : \mathbf{d}(\bar{\mathbf{x}}_\mathbf{p}, \mathbf{r}) \leq \mathbf{d}(\mathbf{x_p}, \mathbf{r}) \text{ for all } \mathbf{x} \in \mathbf{KNN^k_o(X)}\right\}$$

where $d(\cdot, \cdot)$ is a distance from a point to a line.

When we find the closest element to the ray, we can use its color as a ground truth

$$\mathcal{L} = \sum_{\mathbf{r} \in R} \|\hat{C}(\mathbf{r}) - C_c(X, \mathbf{r})\|^\mathbf{2}_\mathbf{2} \tag{3}$$

where $C_c(X, \mathbf{r})$ is the color of closest point from X to ray $\mathbf{r}$ and $\hat{C}(r)$ is the predicted RGB colors for ray $\mathbf{r}$.

This solution, that we dub KNN-NeRF, can be used as a baseline of our model. Unfortunately, KNN-NeRFs cannot reconstruct correct shapes since point clouds in training are highly sparse. To solve this limitation, in this work we propose an auto-encoder-based architecture, which transfers the 3D point cloud into NeRF, hence mitigating the problem with sparse input data.

3 POINTS2NERF: GENERATING NEURAL RADIANCE FIELDS FROM 3D POINT CLOUD

In this section, we present our Points2NeRF model for building NeRF representations of 3D point clouds. To that end, we leverage three main components described below: the auto-encoder architecture, the NeRF representation of a 3D static scene, and the hypernetwork training paradigm. The main intuition behind our approach is the construction of an auto-encoder, which takes as an input 3D point cloud and generates the weight of the target network – NeRF. For effective NeRF training, our model requires a set of 2D images on top of the 3D point clouds captured by 3D registration devices.

Hypernetwork Hypernetworks, introduced in (Ha et al., 2016), are defined as neural models that generate weights for a separate target network solving a specific task. The authors aim to reduce the number of trainable parameters by designing a Hypernetwork with a smaller number of parameters. Making an analogy between Hypernetworks and generative models, the authors of (Sheikh et al., 2017), use this mechanism to generate a diverse set of target networks approximating the same function.

In the context of 3D objects, various methods make use of a hypernetwork to produce a continuous representation of objects (Proszewska et al., 2021; Spurek et al., 2021a; 2020; 2021b;c). HyperCloud (Spurek et al., 2020) represents a 3D point cloud as a classical MLP while in (Spurek et al., 2021c) it is represented by a Continuous Normalizing Flow (Grathwohl et al., 2018). In the case of (Proszewska et al., 2021), the authors model voxel representation by a hypernetwork architecture.

Auto-encoders for 3D Point Clouds In our approach, we use hypernetwork paradigm to aggregate information from 3D point cloud representation and produce the weights of a NeRF architecture. Moreover, such a solution allows us to create a high-resolution model of the 3D point cloud.

In Points2NeRF hypernetwork is an auto-encoder type architecture for the 3D point cloud. Let $\mathcal{X} = \{X_i\}_{i=1,\dots,n} = \{(x_i, y_i, z_i, r_i, g_i, b_i)\}_{i=1,\dots,n}$ be a given dataset containing point clouds with colors. The first three elements encode the position while the last three encode a RGB color. The objective of the autoencoder is to transport the data through a typically lower-dimensional latent space $\mathcal{Z} \subseteq \mathbb{R}^D$ while minimizing the reconstruction error. Thus, we search for an encoder $\mathcal{E} : X \to Z$ and decoder $\mathcal{D} : Z \to X$ functions, which minimize the reconstruction error between X_i and its reconstructions $\mathcal{D}(\mathcal{E}X_i)$. We use a permutation invariant encoder based on PointNet architecture (Qi et al., 2017) and a modified decoder to produce weights instead of raw points. For point cloud representation, the crucial step is to define proper reconstruction loss that can be used in the auto encoding framework. In our approach, we use NeRF cost function.

Input 3D Point Cloud Representation obtained by Points2NeRF

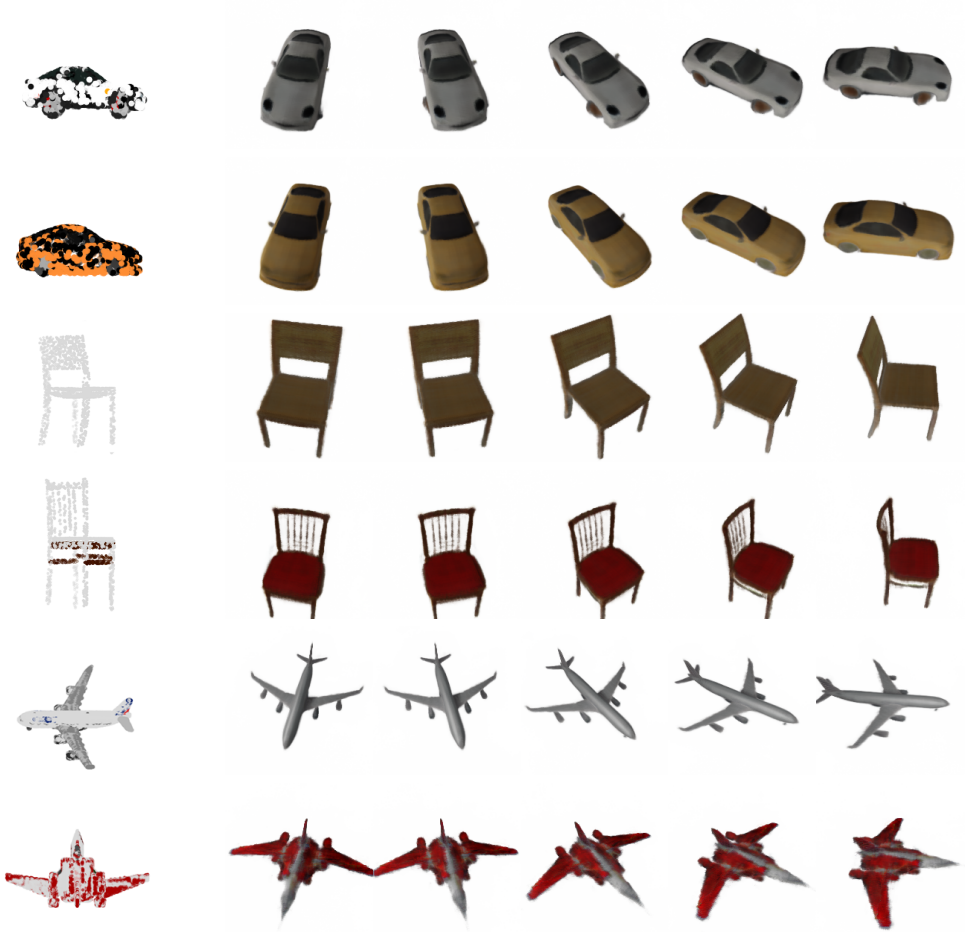

Figure 3: Reconstructions of objects obtained by Points2NeRF.

Points2NeRF In our model, we use three components: hypernetwork, autoencoder, and NeRF. The Points2NeRF model uses Hypernetwork to output weights of generative network to create NeRF representation from the 3D point cloud. More specifically, we present parameterization of the 3D objects as a function $F_\Theta : \mathbb{R}^5 \to \mathbb{R}^4$, which given location (x, y, z) and viewing direction $(\theta, \psi))$ returns a color $\mathbf{c} = (r, g, b)$ and volume density σ. Roughly speaking, instead of producing 3D objects, we would like to produce many neural networks (a different neural network for each object) that model them.

In practice, we have one neural network architecture that uses different weights for each 3D object. The target network (NeRF) is not trained directly. We use a Hypernetwork $H_\Phi : \mathbb{R}^3 \supset X \to \Theta$, which for an point-cloud $X \subset \mathbb{R}^3$ returns weights Θ to the corresponding target network (NeRF) F_Θ. Thus, a point cloud X is represented by a function:

$$F((x, y, x, \theta, \psi); \Theta) = F((x, y, x, \theta, \psi); H_\Phi(X)).$$

To use the above model, we need to train the weights Φ of the hypernetwork. For this purpose, we minimize the NeRF cost function over the training set consisting of pairs: point clouds and 2D images of the object. More precisely, we take an input point cloud $X \subset \mathbb{R}^6$ (the first three elements encode the position while the last three encode an RGB color) and pass it to H_Φ. As a result, the hypernetwork returns weights Θ to the target network F_Θ. Next, the set of 2D images is compared to the renderings generated by the target network F_Θ. As a hypernetwork, we use a permutation invariant encoder based on PointNet architecture (Qi et al., 2017) and a modified decoder to produce weight instead of raw points. The architecture of H_Φ consists of: an encoder ($\mathcal{E}$) which

Figure 4: From NeRF representation, we can extract 3D mesh with color. Mesh representation of objects produced by Points2NeRF.

is a PointNet-like network that transports the data to lower-dimensional latent space $\mathcal{Z} \in \mathbb{R}^D$ and a decoder ($\mathcal{D}$) (fully-connected network), which transfers latent space to the vector of weights for the target network. In our framework hypernetwork $H_\Phi(X)$ represents our autoencoder structure $\mathcal{D}(\mathcal{E}X)$. Assuming $H_\Phi(X) = \mathcal{D}(\mathcal{E}X)$, we train our model by minimizing the cost function given by equation (1).

We only train a single neural model (hypernetwork), which allows us to produce various functions at test time.

Figure 5: Interpolations between elements produce by Generative Points2NeRF.

Mesh representation In the paper, we represent 3D objects as Neural Radiance Fields. Such representation has few advantages over the classical one. In particular, we can obtain 3D mesh with colors.

Thanks to volume density σ, we obtain voxel representation. We can predict an inside/outside category for points from grid (x, y, z). Then we can render objects via the iso-surface extraction a method such as Marching Cubes. We can predict the color for all vertices when we have mesh representation. By using colors in the vertices of the mesh, we can add colors to the faces of the graph. We present 3D objects with colors in Fig. 4.

Figure 6: Object generated by Generative Points2NeRF.

Generative model In our model, we use autoencoder architecture in hypernetwork. Therefore it is easy to construct a generative model.

An autoencoder-based generative model is a classical auto-encoder model with a modified cost function, which forces the model to be generative, i.e., ensures that the data transported to the latent space comes from the prior distribution (typically Gaussian) (Kingma & Welling, 2013; Tolstikhin et al., 2017; Knop et al., 2020). Thus, to construct a generative auto-encoder model, we add to its cost function to measure the distance of a given sample from the prior distribution.

Variational Auto-encoders (VAE) are generative models capable of learning approximated data distribution by applying variational inference (Kingma & Welling, 2013). To ensure that the data transported to latent space $\mathcal{Z}$ are distributed according to standard normal density. We add the distance from standard multivariate normal density. By adding Kullback–Leibler divergence to our cost function to obtain a generative model, we cold Generative Points2NeRF. In Fig. 6 we present samples an in Fig. 5 interpolations obtained by Generative Points2NeRF.

4 Related works

3D objects can be represented by using various techniques, including voxel grids (Choy et al., 2016), octrees (Häne et al., 2017), multi-view images (Arsalan Soltani et al., 2017), point clouds (Achlioptas et al., 2018), geometry images (Sinha et al., 2016), deformable meshes (Girdhar et al., 2016), and part-based structural graphs (Li et al., 2017).

The above representations are discreet, a substantial limitation in real-life applications. Alternatively, we can represent 3D objects as a continuous function (Dupont et al., 2021). In practice implicit occupancy (Chen & Zhang, 2019; Mescheder et al., 2019; Peng et al., 2020), distance field (Michalkiewicz et al., 2019; Park et al., 2019) and surface parametrization (Yang et al., 2019; Spurek et al., 2020; 2021c; Cai et al., 2020) models use a neural network to parameterize a 3D object. We do not have a fixed number of voxels, points, or vertices in such a case, but we represent shapes as a continuous function.

These models are limited by their requirement of access to ground truth 3D geometry. Recently works relaxed this requirement of ground truth 3D shapes by using only 2D images. In (Niemeyer et al., 2020) authors present 3D occupancy fields. The numerical method is used to find the surface intersection for each ray. In (Sitzmann et al., 2019) propose a neural network that produces feature vector and RGB color at each continuous 3D coordinate, and propose a differentiable rendering function consisting of a recurrent neural network.

The above models are limited to simple shapes with low geometric complexity, resulting in over-smoothed renderings. To solve such a problem, NeRF model was proposed. NeRF represents a static scene as a continuous 5D function that outputs the radiance emitted in each direction and a density at each point which acts like a differential opacity controlling how much radiance is accumulated by a ray passing through the point.

The NeRF method is a state-of-the-art solution for representing 3D objects. However, the model has many different generalizations for static and dynamic scenes.

Information from point clouds was used in NeRF across different applications. In (Xu et al., 2022) authors present a novel neural scene representation Point-NeRF that models a volumetric radiance field with a neural point cloud. In NeuS (Wang et al., 2021), authors propose to represent a surface as the zero-level set of a signed distance function (SDF) and develop a new volume rendering method to train a neural SDF representation. In consequence, we obtain a novel neural surface reconstruction method. In (Ost et al., 2021), authors introduce Neural Point Light Fields that represent scenes implicitly with a light field living on a sparse point cloud. We use point clouds to produce NeRF representation in our work.

	Points2NeRF	NeRF-KNN
planes (train)	**24.83**	13.75
planes (test)	**20.45**	14.18
cars (train)	**28.14**	11.82
cars (test)	**20.86**	12.47
chairs (train)	**23.90**	10.02
chairs (test)	**17.17**	10.36

Table 1: Comparison of PSNR metric between our model and baseline KNN-NeRF trained on three classes of the ShapeNet data.

Most of such models are trained on a single scene. NeRF-VAE is a 3D scene generative model. In contrast to NeRF, such a model considers shared structure across scenes. Unfortunately, the model was trained only on simple scenes containing geometric figures. In our paper, we present Points2NeRF, which is trained on an extensive data set and can transform a 3D point cloud to NeRF representation.

5 EXPERIMENTS

In this section, we describe the experimental results of the proposed model. To our knowledge, it is the first model that obtains translation from 3D point clouds to NeRF. Therefore, it is hard to compare our results to other algorithms. In the first subsection, we show that our model produces high-quality NeRF representations of the objects by comparing our model with our baseline KNN-NeRF. In the second one, we compare our model by using voxel representation obtained from NeRF.

Methodology We used **ShapeNet** dataset to train our model. First, we sampled 2048 colored points from each object from three categories: cars, chairs, and planes. For each object: fifty 200x200 transparent background images from random camera positions.

Point to NeRF evaluation We compare the metric reported by NeRF called PSNR (*peak signal-to-noise ratio*) which is used to measure image reconstruction effectiveness.

Category	PointConv	ONet	ConvONet	POCO	Ours
cars	0.577	0.747	0.849	0.946	**0.995**
planes	0.562	0.829	0.965	**0.994**	0.952
chairs	0.618	0.730	0.939	0.985	**0.992**

Table 2: F-Score comparison between our model and baselines: PointConv(Peng et al., 2020), ONet(Mescheder et al., 2019), ConvONet(Tang et al., 2021), POCO(Boulch & Marlet, 2022). We trained our model on 2048 colored points and sampled on 3000 while other methods used 3000 to train and test.

In Tab. 1 we compare Points2NeRF and baseline solution KNN-NeRF. We present a comparison of the training and the test set. It should be highlighted that KNN-NeRF is trained separately for each object from the train and test set. The Points2NeRF is trained on the training data set and evaluated on the test set. We achieve better results in all categories on the training data and the test set.

Voxel representation NeRF networks, which are outputs of our hypernetwork can describe the occupancy of a given point in 3D space. With the marching cubes algorithm, we can obtain mesh reconstruction for a given point cloud.

To compare reconstruction with original mesh, we use Chamfer Distance, defined as the distance between two clouds of points P_1 and P_2 such that:

$$CD(P_1, P_2) = \frac{1}{2|P1|} \sum_{p_1 \in P1} \max_{p_2 \in P_2} d(p_1, p_2) + \frac{1}{2|P2|} \sum_{p_2 \in P2} \max_{p_1 \in P_1} d(p_1, p_2) \tag{4}$$

Additionally, we use F-Score metric between two clouds of points with some threshold t defined as:

$$\text{F-Score}(P_1, P_2, t) = \frac{\text{2Recall Precision}}{\text{Recall + Precision}}. \tag{5}$$

For F-Score and Chamfer Distance calculation, we sample random 3000 points from both original and reconstructed mesh, and we used threshold $t = 0.01$ to find matching points for F-Score.

Even though our loss function was not directly related to mesh but to image reconstruction, we were able to achieve competitive results, see Tab 2 and Tab 3.

Category	PointConv	ONet	ConvONet	POCO	Ours
cars	1.49	1.04	0.75	0.41	**0.16**
planes	1.40	0.64	0.34	**0.23**	0.91
chairs	1.29	0.95	0.46	0.33	**0.15**

Table 3: Chamfer Distance (multiplied by 10^2) comparison between our model and baselines: PointConv, ONet, ConvONet, POCO. We trained our model on 2048 colored points and sampled on 3000 while other methods used 3000 to train and test.

6 Conclusions

In this work, we presented a novel approach to generating NeRF representation from 3D point clouds. Our model leverages a hypernetwork paradigm and NeRF representation of the 3D scene. Points2NeRF take a 3D point cloud with the associated color values and return the weights of a NeRF network that reconstructs 3D objects from 2D images. Such representation gives several advantages over the existing approaches. First of all, we can add a conditioning mechanism to NeRFs that allows controlling the process of creating a generative model. Secondly, we can quickly obtain mesh representation with colors, which is a challenging task in 3D object rendering.

References

Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. Learning representations and generative models for 3d point clouds. In *International conference on machine learning*, pp. 40–49. PMLR, 2018.

Amir Arsalan Soltani, Haibin Huang, Jiajun Wu, Tejas D Kulkarni, and Joshua B Tenenbaum. Synthesizing 3d shapes via modeling multi-view depth maps and silhouettes with deep generative networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1511–1519, 2017.

Alexandre Boulch and Renaud Marlet. Poco: Point convolution for surface reconstruction. *arXiv:2201.01831*, 2022.

Ruojin Cai, Guandao Yang, Hadar Averbuch-Elor, Zekun Hao, Serge Belongie, Noah Snavely, and Bharath Hariharan. Learning gradient fields for shape generation. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*, pp. 364–381. Springer, 2020.

Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5939–5948, 2019.

Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *European conference on computer vision*, pp. 628–644. Springer, 2016.

Emilien Dupont, Yee Whye Teh, and Arnaud Doucet. Generative models as distributions of functions. *arXiv preprint arXiv:2102.04776*, 2021.

Rohit Girdhar, David F Fouhey, Mikel Rodriguez, and Abhinav Gupta. Learning a predictable and generative vector representation for objects. In *European Conference on Computer Vision*, pp. 484–499. Springer, 2016.

Will Grathwohl, Ricky TQ Chen, Jesse Betterncourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

Christian Häne, Shubham Tulsiani, and Jitendra Malik. Hierarchical surface prediction for 3d object reconstruction. In *2017 International Conference on 3D Vision (3DV)*, pp. 412–420. IEEE, 2017.

James T Kajiya and Brian P Von Herzen. Ray tracing volume densities. *ACM SIGGRAPH computer graphics*, 18(3):165–174, 1984.

Kacper Kania, Kwang Moo Yi, Marek Kowalski, Tomasz Trzciński, and Andrea Tagliasacchi. Conerf: Controllable neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 18623–18632, 2022.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Szymon Knop, Przemysław Spurek, Jacek Tabor, Igor Podolak, Marcin Mazur, and Stanisław Jastrzębski. Cramer-wold auto-encoder. *Journal of Machine Learning Research*, 21, 2020.

Adam R Kosiorek, Heiko Strathmann, Daniel Zoran, Pol Moreno, Rosalia Schneider, Sona Mokrá, and Danilo Jimenez Rezende. Nerf-vae: A geometry aware 3d scene generative model. In *International Conference on Machine Learning*, pp. 5742–5752. PMLR, 2021.

Jun Li, Kai Xu, Siddhartha Chaudhuri, Ersin Yumer, Hao Zhang, and Leonidas Guibas. Grass: Generative recursive autoencoders for shape structures. *ACM Transactions on Graphics (TOG)*, 36 (4):1–14, 2017.

Nelson Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, 1995.

Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4460–4470, 2019.

Mateusz Michalkiewicz, Jhony K Pontes, Dominic Jack, Mahsa Baktashmotlagh, and Anders Eriksson. Implicit surface representations as layers in neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4743–4752, 2019.

Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, pp. 405–421. Springer, 2020.

Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3504–3515, 2020.

Julian Ost, Issam Laradji, Alejandro Newell, Yuval Bahat, and Felix Heide. Neural point light fields. *arXiv preprint arXiv:2112.01473*, 2021.

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 165–174, 2019.

Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*, pp. 523–540. Springer, 2020.

Magdalena Proszewska, Marcin Mazur, Tomasz Trzciński, and Przemysław Spurek. Hypercube: Implicit field representations of voxelized 3d models. *arXiv preprint arXiv:2110.05770*, 2021.

Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 652–660, 2017.

Abdul-Saboor Sheikh, Kashif Rasul, Andreas Merentitis, and Urs Bergmann. Stochastic maximum likelihood optimization via hypernetworks. *arXiv preprint arXiv:1712.01141*, 2017.

Ayan Sinha, Jing Bai, and Karthik Ramani. Deep learning 3d shape surfaces using geometry images. In *European Conference on Computer Vision*, pp. 223–240. Springer, 2016.

Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. *Advances in Neural Information Processing Systems*, 32, 2019.

Przemysław Spurek, Sebastian Winczowski, Jacek Tabor, Maciej Zamorski, Maciej Zieba, and Tomasz Trzcinski. Hypernetwork approach to generating point clouds. In *International Conference on Machine Learning*, pp. 9099–9108. PMLR, 2020.

Przemysław Spurek, Artur Kasymov, Marcin Mazur, Diana Janik, Sławomir Tadeja, Lukasz Struski, Jacek Tabor, and Tomasz Trzciński. Hyperpocket: Generative point cloud completion. *arXiv preprint arXiv:2102.05973*, 2021a.

Przemysław Spurek, Sebastian Winczowski, Maciej Zięba, Tomasz Trzciński, and Kacper Kania. Modeling 3d surface manifolds with a locally conditioned atlas. *arXiv preprint arXiv:2102.05984*, 2021b.

Przemysław Spurek, Maciej Zieba, Jacek Tabor, and Tomasz Trzcinski. General hypernetwork framework for creating 3d point clouds. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021c.

Jiapeng Tang, Jiabao Lei, Dan Xu, Feiying Ma, Kui Jia, and Lei Zhang. Sa-convonet: Sign-agnostic optimization of convolutional occupancy networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 6504–6513, 2021.

Ilya Tolstikhin, Olivier Bousquet, Sylvain Gelly, and Bernhard Schoelkopf. Wasserstein auto-encoders. *arXiv preprint arXiv:1711.01558*, 2017.

Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. *arXiv preprint arXiv:2106.10689*, 2021.

Qiangeng Xu, Zexiang Xu, Julien Philip, Sai Bi, Zhixin Shu, Kalyan Sunkavalli, and Ulrich Neumann. Point-nerf: Point-based neural radiance fields. *arXiv preprint arXiv:2201.08845*, 2022.

Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. Pointflow: 3d point cloud generation with continuous normalizing flows. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 4541–4550, 2019.