GH-NeRF: Neural Radiance Fields with Gaussian Distributed Spherical Harmonics

Abstract

Since the proposal of Neural Radiance Field (NeRF), novel view synthesis has achieved remarkable progress. However, NeRF requires a large amount of time and GPU training resources. At the same time, due to its design limitations, it is very likely to cause aliasing artifacts in the images. Our method, which we called "GH-NeRF", is a NeRF-like model which uses 3D multivariate Gaussian random variables along with Spherical Harmonics to accelerate the training process by 25% and also, tackle the problem of excessively blurring or alias of the original NeRF implementation, lowering the error rate by with 6.4% relative to NeRF on LLFF dataset.

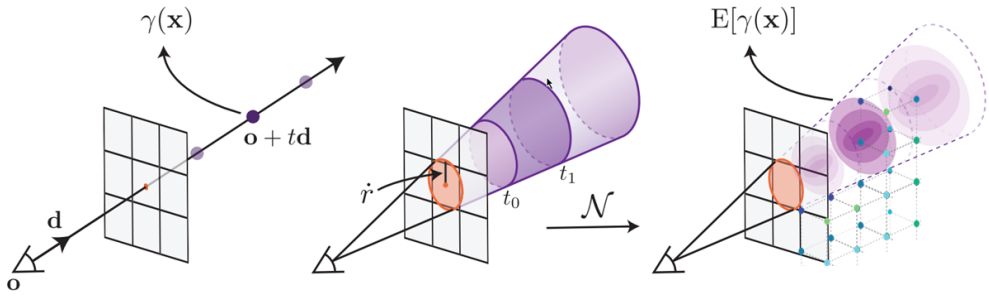
1 Introduction

Neural volumetric representations such as neural radiance fields (NeRF) [5] have emerged as a compelling strategy for learning to represent 3D objects and scenes from images to render photorealistic novel views. Although NeRF and its variants have demonstrated impressive results across a range of view synthesis tasks, NeRF's rendering model is flawed in a manner that can cause excessive blurring and aliasing. To deal with the mentioned problem, Jonathan *et al.* proposes Mip-NeRF [1], by approximating the 3D *conical frustum* with a multivariate Gaussian.

Another issue is that the training time of the model is too long. Jonáš Kulhánek *et al.* proposed a transformer architecture [3] to directly train the model end-to-end. Recent advances in NeRF such use of Spherical Harmonics to accelerate the rendering process [6], and even get rid of the neural network training part [2]. Moreover, to deal with the case where camera views maybe not be enough for the training process, there has been work using multiplane images (MPI) [7] along with an encoder-decoder architecture to solve the problem.

In this paper, we take inspiration from Spherical Harmonics used in physical science in solving partial differential equations in many scientific fields. Our method, which we call GH-NeRF extends NeRF to simultaneously represent the prefiltered radiance field for a continuous space of scales. The input of GH-NeRF is a 3D Gaussian that represents the region over which the radiance field should be integrated, and the output is a corresponding Spherical Harmonics which would then be used to calculate the RGB of the pixel.

As illustrated in Figure 1, we can then render a prefiltered pixel by querying GH-NeRF at intervals along a cone, using Gaussians that approximate the conical frustums corresponding to the pixel in the same way as Mip-NeRF [1]. Different from other works, the output of our model is the expectation of the cone's Spherical Harmonics, which would later be used to calculate the pixel's color.



(a) NeRF (b) GH-NeRF

Figure 1: NeRF (a) samples points x along rays that are traced from the camera center of projection through each pixel then encodes those points with a positional encoding (PE) γ to produce a feature $\gamma(x)$.
GH-NeRF (b) instead reasons about the 3D conical frustum defined by a camera pixel. These conical frustums are then featured with our integrated positional encoding (IPE), which works by approximating the frustum with a multivariate Gaussian and then computing the (closed form) integral $E[\gamma(x)]$ over the positional encodings of the coordinates within the Gaussian.

2 Related Works

2.1 NeRF

NeRF uses the weights of a multilayer perceptron (MLP) to represent a scene as a continuous volumetric field of particles that block and emit light. NeRF renders each pixel of a camera as follows: A ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ is emitted from the camera's center of projection o along the direction d such that it passes through the pixel. A sampling strategy is used to determine a vector of sorted distances $\mathbf{t}$ between the camera's predefined near and far planes t_n and t_f. For each distance $t_k \in \mathbf{t}$, we compute its corresponding 3D position along the ray $\mathbf{x} = \mathbf{r}(t_k)$, then transform each position using a positional encoding:

$$\gamma(\mathbf{x}) = \left[\sin(\mathbf{x}), \cos(\mathbf{x}), \ldots, \sin\left(2^{L-1}\mathbf{x}\right), \cos\left(2^{L-1}\mathbf{x}\right)\right]^{\mathrm{T}} \tag{1}$$

This is simply the concatenation of the sines and cosines of each dimension of the 3D position x scaled by powers of 2 from 1 to 2^{L-1}, where L is a hyperparameter. Then, the positional encoding of each ray position $\gamma(\mathbf{r}(t_k))$ is provided as input to an MLP parameterized by weights Θ, which outputs a density τ and an RGB color c:

$$\forall t_k \in \mathbf{t}, [\tau_k, \mathbf{c}_k] = \mathrm{MLP}(\gamma(\mathbf{r}(t_k)); \Theta) \tag{2}$$

The MLP also takes the view direction as input, which is omitted from the notation for simplicity. These estimated densities and colors are used to approximate the volume rendering integral using numerical quadrature, as per Max [4]:

$$\begin{array}{c} \mathbf{C}(\mathbf{r}; \Theta, \mathbf{t}) = \sum_k T_k \left(1 - \exp\left(-\tau_k \left(t_{k+1} - t_k\right)\right)\right) \mathbf{c}_k \\ \text{with} \quad T_k = \exp\left(-\sum_{k'<k} \tau_{k'} \left(t_{k'+1} - t_{k'}\right)\right), \end{array} \tag{3}$$

Moreover, the original NeRF adopts a "coarse-to-fine" network strategy. It trains two separate MLPs Θ^c and Θ^f with the loss function to be:

$$\min_{\Theta^c, \Theta^f} \sum_{\mathbf{r} \in \mathcal{R}} \left(\left\| \mathbf{C}^*(\mathbf{r}) - \mathbf{C}\left(\mathbf{r}; \Theta^c, \mathbf{t}^c\right) \right\|_2^2 + \left\| \mathbf{C}^*(\mathbf{r}) - \mathbf{C}\left(\mathbf{r}; \Theta^f, \mathbf{t}^c \cup \mathbf{t}^f\right) \right\|_2^2 \right) \tag{4}$$

2.2 Spherical Harmonic

In mathematics and physical science, spherical harmonics are special functions defined on the surface of a sphere. The Spherical Harmonics (SH) form a complete orthogonal basis of functions $\mathbf{S}^2 \to \mathbf{C}$. For $\ell \in \mathbf{N}$ and $m \in \{-\ell, \cdots, \ell\}$, the SH function of degree l and order m is defined as:

$$Y_\ell^m(\theta, \phi) = \sqrt{\frac{2\ell+1}{4\pi}\frac{(\ell-m)!}{(\ell+m)!}} P_\ell^m(\cos\theta) e^{im\phi} \tag{5}$$

where $P_\ell^m(\cos\theta)$ are the associated Legendre polynomials. A real basis of spherical harmonics $Y_\ell^m : S^2 \to \mathbb{R}$.

$$Y_\ell^m(\theta, \phi) = \begin{cases} \sqrt{2}\,(-1)^m\, \Im[Y_\ell^{|m|}] & \text{if } m < 0 \\ Y_\ell^0 & \text{if } m = 0 \\ \sqrt{2}\,(-1)^m\, \Re[Y_\ell^m] & \text{if } m > 0. \end{cases} \tag{6}$$

SHs have been a popular low-dimensional representation for spherical functions and have been used to model Lambertian surfaces or even glossy surfaces.

3 Method

Here we use a neural network to estimate the Spherical Harmonics coefficients:

$$c(\mathbf{d}; \mathbf{k}) = \sigma(\sum_{\ell=0}^{\ell_{\max}} \sum_{m=-\ell}^{\ell} k_\ell^m Y_\ell^m(\mathbf{d})) \tag{7}$$

where $\sigma(\cdot)$ is the sigmoid function to guarantee the output within the range $[0, 1]$.

Different from the original NeRF, the input of the network is the integrated positional encoding (IPE) used same in Mip-NeRF[1]:

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 & 2 & 0 & 0 & & 2^{L-1} & 0 & 0 \\ 0 & 1 & 0 & 0 & 2 & 0 & \dots & 0 & 2^{L-1} & 0 \\ 0 & 0 & 1 & 0 & 0 & 2 & & 0 & 0 & 2^{L-1} \end{bmatrix}^T, \; \gamma(\mathbf{x}) = \begin{bmatrix} \sin(\mathbf{P}\mathbf{x}) \\ \cos(\mathbf{P}\mathbf{x}) \end{bmatrix} \tag{8}$$

where $\mathbf{x}$ is a multivariate r.v. such that $\mathbf{x} \sim \mathcal{N}(\mu, \mathbf{\Sigma})$, with:

$$\begin{gathered} \boldsymbol{\mu} = \mathbf{o} + \mu_t \mathbf{d}, \quad \mathbf{\Sigma} = \sigma_t^2 \left(\mathbf{d}\mathbf{d}^{\mathrm{T}}\right) + \sigma_r^2 \left(\mathbf{I} - \frac{\mathbf{d}\mathbf{d}^{\mathrm{T}}}{\|\mathbf{d}\|_2^2}\right) \\ \mu_t = t_\mu + \frac{2t_\mu t_\delta^2}{3t_\mu^2 + t_\delta^2}, \quad \sigma_t^2 = \frac{t_\delta^2}{3} - \frac{4t_\delta^4\left(12t_\mu^2 - t_\delta^2\right)}{15\left(3t_\mu^2 + t_\delta^2\right)^2} \\ \sigma_r^2 = \dot{r}^2 \left(\frac{t_\mu^2}{4} + \frac{5t_\delta^2}{12} - \frac{4t_\delta^4}{15\left(3t_\mu^2 + t_\delta^2\right)}\right) \end{gathered} \tag{9}$$

Here we use $t_\mu = (t_0 + t_1)/2, t_\delta = (t_1 - t_0)/2$ to avoid floating-point errors during computation.

3.1 Model

Aside from SH-coefficients, GH-NeRF behaves similarly to NeRF mentioned in Section 2.1. Figure 2 shows the architecture of GH-NeRF. As for the implementation details, we use Layer Normalization after each fully-connected layer to stabilize the training process.

Moreover, GH-NeRF still adopts a "coarse-to-fine" network strategy, but with a single MLP (here $\lambda = 0.1$).

$$\min_{\Theta} \sum_{\mathbf{r} \in \mathcal{R}} \left(\lambda \cdot \|\mathbf{C}^*(\mathbf{r}) - \mathbf{C}\left(\mathbf{r}; \Theta, \mathbf{t}^c\right)\|_2^2 + \left\|\mathbf{C}^*(\mathbf{r}) - \mathbf{C}\left(\mathbf{r}; \Theta, \mathbf{t}^c \cup \mathbf{t}^f\right)\right\|_2^2\right) \tag{10}$$

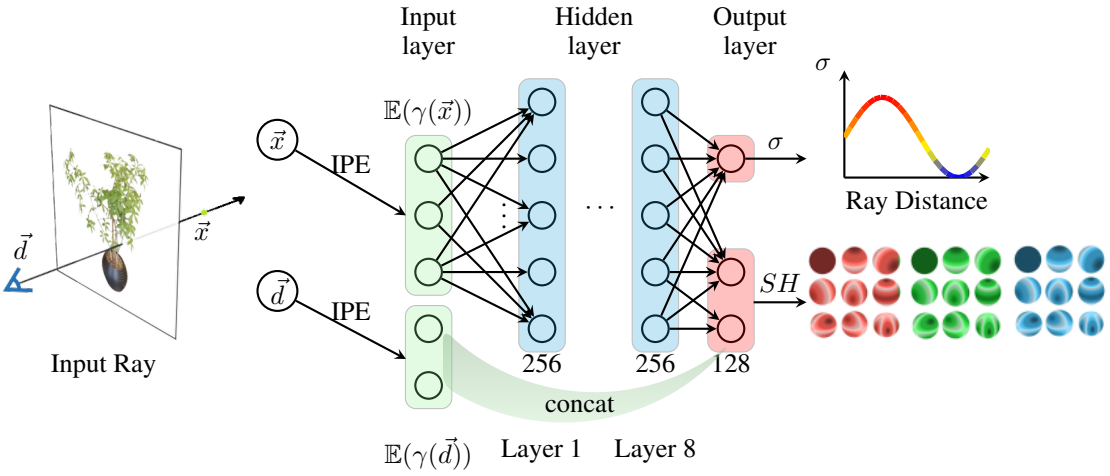


Figure 2: **Architecture of GH-NeRF:** We first do integrated positional encoding for $\vec{x}$ and $\vec{d}$ individually, where $\vec{x} = \mathbf{r}(t_k)$. Then, $\mathbb{E}(\gamma(\vec{x}))$ is passed through 8 fully-connected ReLU layers, each with 256 channels. An additional layer outputs the volume density σ (which is rectified using a ReLU to ensure that the output volume density is nonnegative). We will also concatenate the last hidden layer (with 128 channels) with the IPE of view direction $\vec{d}$ (showed by the light green area) to output the estimated Spherical Harmonics coefficients for RGB respectively.

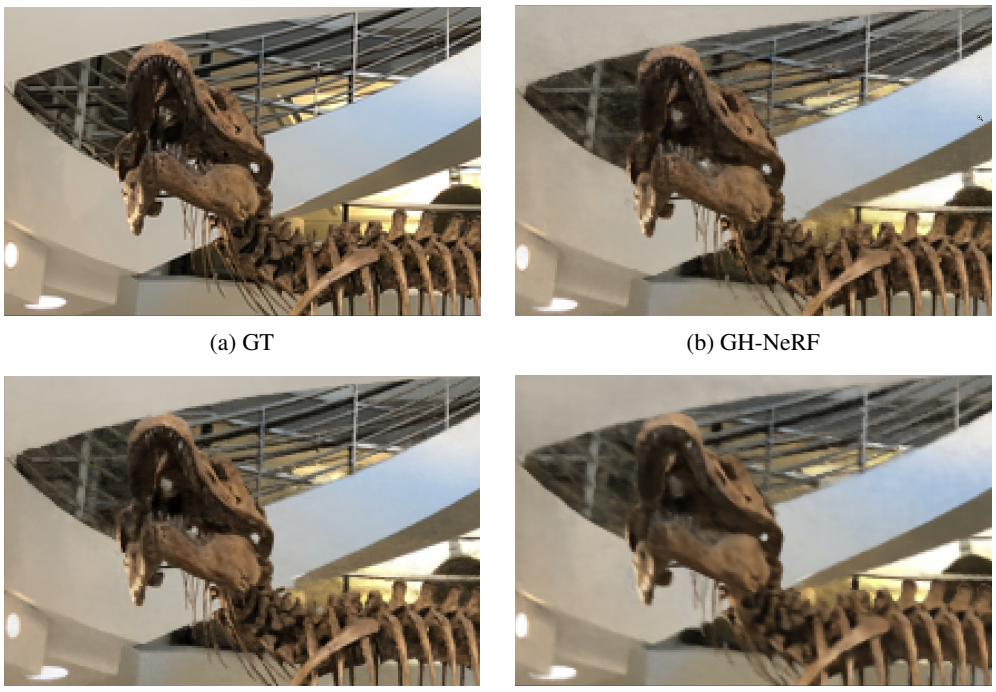
4 Experiments

Table 1: **Quantitative results on the NeRF-LLFF dataset scene: TREX**

Method	TRAIN PSNR↑	TEST PSNR↑	SSIM ↑	LPIPS↓	Time (hours)
NeRF	28.3	25.98	0.914	0.156	3.17
Mip-NeRF	30.62	28.19	0.947	0.109	2.22
GH-NeRF	31.07	27.65	0.942	0.117	2.37
GH-NeRF coarse	29.23	/	/	/	2.37
GH-NeRF LayerNorm	31.33	28.7	0.954	0.088	3.13

Tabel 1 shows a quantitative comparison of GH-NeRF and its ablations against NeRF and several NeRF variants on the test set of our multiscale Blender dataset. We use one GTX3090(24GB) to train all models with 2×10^5 epochs and provide the training time in the last column. See the text for details (light green: best; light yellow: second best).

Compared to the original NeRF which runs nearly 3.17 hours, GH-NeRF is 25.2% faster than it and gets the second best PSNR during the training phase and reduced TEST PSNR error rate 6.4% compared to NeRF. If we want high resolution pictures, GH-NeRF with Layer Normalization achieved the best performance in any metric, reducing error rates roughly 10.5% relative to NeRF, despite the fact that its running time is slower that others.



(a) GT (b) GH-NeRF

(c) Mip-NeRF (d) NeRF

Figure 3: On the trex scene of LLFF dataset, compared to the ground truth, GH-NeRF achieved anti-aliasing and GH-NeRF achieves better lighting and shading effects compared to Mip-NeRF.

Meanwhile, as Figure 3 shows, the GH-NeRF tackled the problem of excessive blurring or alias of the original NeRF implementation. Moreover, the GH-NeRF can also be used for 3D reconstruction and depth-estimation as Figure 4 shows.

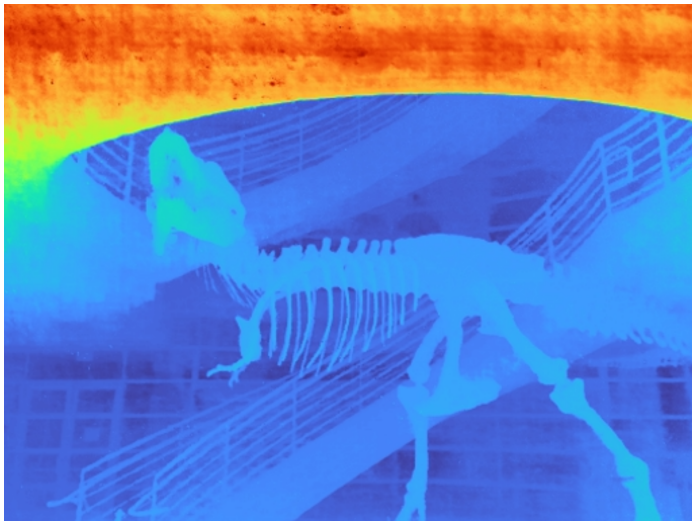


Figure 4: Depth estimation achieved by GH-NeRF on LLFF trex scene

5 Future Work

Although we have significantly reduced the training time of NeRF, it must be acknowledged that the existing training time is still quite long and costly. Additionally, for different real-world scenarios,

retraining the entire model each time incurs significant overhead. We hope that in the future, we can effectively address the aforementioned issues.

6 Conclusions

We have presented GH-NeRF, a NeRF-like model which uses 3D multivariate Gaussian random variables along with Spherical Harmonics to address the inherent aliasing of NeRF. NeRF works by casting rays, encoding the positions of points along those rays, and training separate neural networks at distinct scales. In contrast, GH-NeRF casts cones, encodes the positions and sizes of conical frustums, and trains a single neural network that models the scene at multiple scales. Compared to the original NeRF, GH-NeRF is 25.2% faster and reduces error rate 6.4% compared to NeRF.

Hope that the general techniques presented here will be valuable to other researchers working to improve the performance of raytracing-based neural rendering models.

References

[1] Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5855–5864, 2021.

[2] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5501–5510, 2022.

[3] Jonáš Kulhánek, Erik Derner, Torsten Sattler, and Robert Babuška. Viewformer: Nerf-free neural rendering from few images using transformers. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XV*, pages 198–216. Springer, 2022.

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

[5] 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. *Communications of the ACM*, 65(1):99–106, 2021.

[6] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. Plenoctrees for real-time rendering of neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5752–5761, 2021.

[7] Tinghui Zhou, Richard Tucker, John Flynn, Graham Fyffe, and Noah Snavely. Stereo magnification: Learning view synthesis using multiplane images. *arXiv preprint arXiv:1805.09817*, 2018.