GL-NeRF: Gauss-Laguerre Quadrature for Volume Rendering

Silong Yong Yaqi Xie Simon Stepputtis Katia Sycara
School of Computer Science, Carnegie Mellon University
{silongy, yaqix, sstepput, katia}@andrew.cmu.edu

Abstract

We propose GL-NeRF, a new perspective of computing volume rendering with the Gauss-Laguerre quadrature. GL-NeRF manages to reduce the number of points needed for the "fine" network in NeRF by selecting points along the ray using the Gauss-Laguerre quadrature, which theoretically guarantees the highest algebraic degree of precision. While most of the existing works on sample efficiency for NeRF introduce extra neural networks for the purpose, GL-NeRF follows the simplest formulation with no additional neural networks. Thus, it can be seamlessly incorporated into NeRF at rendering stage without training. To the best of our knowledge, GL-NeRF is the first method that could be directly used without training to reduce rendering time and memory usage simultaneously. Our theoretical results have been empirically validated on Blender and Real Forward Facing datasets.

1. Introduction

Neural Radiance Fields (NeRFs) [20] have shown promising results for synthesizing images from novel views. The core component for NeRF's success is volume rendering, which requires approximating an integral by densely sampling points along the ray and evaluating volume density and radiance using a neural network for them. In practice, more than 100 neural network inferences are needed for precisely synthesizing the color for a single pixel, which could be redundant. Works have been done to reduce the time needed for rendering images, aiming at providing NeRF with a real-time rendering ability [6, 7, 17, 21, 35]. Despite the promising results shown by these works, they propose different approaches for achieving real-time rendering by introducing new networks, new data structures, *etc*. Therefore, each individual work requires training from scratch with a specific optimization goal. In this work, we propose a method that could be implemented in any existing NeRF-based models that require volume rendering without further training.

Our approach arises from revisiting the volume rendering integral, the key discovery is that with a simple change of variable, we can turn the integral into a pure exponentially weighted integral of color. This specific form has a Gauss quadrature (*i.e*. the Gauss-Laguerre quadrature) which best approximates it mathematically. Naturally, we propose to use the Gauss-Laguerre quadrature to directly compute the volume rendering integral, which we call GL-NeRF (Gauss Laguerre-NeRF), leading to much lower computational cost for approximating the integral and therefore lower time and memory usage.

GL-NeRF provides a different angle for computing volume rendering and has the potential to be a direct plug-in for existing NeRF-based products.

2. Preliminaries

Our work is built upon the basic NeRF pipeline with little modifications. We will cover the basic concept in NeRF, volume rendering and Gauss quadrature in this section.

2.1. NeRF and volume rendering

NeRF [20] is a powerful implicit 3D scene model for novel view synthesis. At the core of its rendering ability is volume rendering. NeRF uses coordinate-based MLP to encode the scene, assigning volume density (opacity) and radiance (color) to spatial points. When used for synthesizing new views, it casts a ray $\boldsymbol{r}(t) = \boldsymbol{o} + t\boldsymbol{d}$ through the pixel to be rendered, samples points along the ray and computes volume density and radiance for these points. These values are then aggregated together using Eq. (1) to give the pixel's color.

$$\hat{\boldsymbol{C}}(\boldsymbol{r}) = \sum_{i=1}^{N} w_i \boldsymbol{c}(\boldsymbol{r}(t_i)), \tag{1}$$

where

$$w_i = T_i(1 - exp(-\sigma(\boldsymbol{r}(t_i))\delta_i)), \tag{2}$$

$$T_i = exp(-\sum_{j=0}^{i-1} \sigma(\boldsymbol{r}(t_j))\delta_j), \tag{3}$$

t_i represents the sampled position along the ray and $\delta_i = t_{i+1} - t_i$ is the distance between two nearby sampled points.

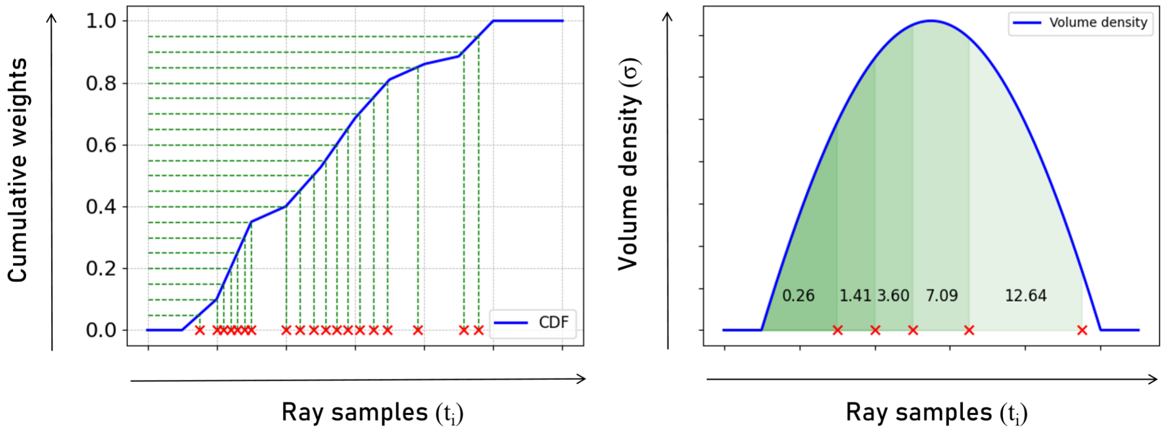

Figure 1. Comparison of point selection strategy. **Left: point selection in original NeRF.** They uniformly sample points from $U(0, 1)$ and use the CDF to inversely map these points onto the ray. With higher weights(*i.e.* the greater slope is greater) comes denser points. **Right: point selection in our method.** We choose points along the ray that satisfy the integral from zero to the point of the volume density function should be equal to the roots of Laguerre polynomials. In the figure above is an example of choosing 5 points using a 5-degree Laguerre polynomial. The number on the plot indicates the value of the integral from zero to the right boundary of the region.

Hierarchical volume sampling. Since randomly sampling along rays could fall into empty space or interior of objects that lacks the information on what the underlying scene actually looks like, NeRF proposes a two-stage hierarchical sampling strategy. It uses a coarse network to first give a rough estimation of w_i, then produces a piecewise constant PDF. Using this PDF for weighted sampling could provide points that have more visual effects along the ray. Finally, a "fine" network takes both the "coarse" samples and the "fine" samples as input and uses Eq. (1) to compute the pixel color.

2.2. Gauss quadrature

An n-point Gauss quadrature [8] is a method for numerical integration that guarantees to yield exact results for integral of polynomials of degree $2n-1$ or less, which is the highest possible precision for approximating an integral by quadrature.

Gauss-Laguerre quadrature is a variant of the Gauss quadrature for approximating integrals following the form of

$$\int_0^\infty e^{-x} f(x)dx \approx \sum_{i=1}^{n} w_i f(x_i). \tag{4}$$

In this case, the weight function is $g(x) = e^{-x}$, the integral interval is $[0, \infty)$. x_i corresponds to the root of the Laguerre polynomials. The weight can be computed explicitly.

While the computation for x_i and w_i is complicated, in practice we can use a look up table to store corresponding x_i and w_i for a given n.

3. Method

We developed our algorithm based on a simple observation of the integral for volume rendering. Eq. (1) is an approximation to the integral

$$C(\boldsymbol{r}) = \int_{t_n}^{t_f} T(t)\sigma(\boldsymbol{r}(t))\boldsymbol{c}(\boldsymbol{r}(t), d)dt, \tag{5}$$

where

$$T(t) = exp(-\int_{t_n}^{t} \sigma(\boldsymbol{r}(s))ds). \tag{6}$$

Let

$$x(t) = \int_{t_n}^{t} \sigma(\boldsymbol{r}(s))ds, \tag{7}$$

we have

$$\frac{dx}{dt} = \sigma(\boldsymbol{r}(t)). \tag{8}$$

Since $\sigma(\boldsymbol{r}(t)) \geq 0$, $x(t)$ is a monotonically non-decreasing function of t, therefore, x has a unique correspondence with t on increasing intervals. With this observation, we can do a change of variables for Eq. (5) to get

$$\begin{aligned} C(\boldsymbol{r}) &= \int_{t_n}^{t_f} T(t)\sigma(\boldsymbol{r}(t))\boldsymbol{c}(\boldsymbol{r}(t), d)dt \\ &= \int_{t_n}^{t_f} e^{-x}\boldsymbol{c}(\boldsymbol{r}(t), d)\frac{dx}{dt}dt \\ &= \int_{x(t_n)}^{x(t_f)} e^{-x}\boldsymbol{c}(\boldsymbol{r}(x), d)dx. \end{aligned} \tag{9}$$

As can be seen from Eq. (9), the integral for volume rendering is a weighted integral of $\boldsymbol{c}(\boldsymbol{r}(x), d)$ with the weight function to be $g(x) = e^{-x}$. We can extend the integral interval from $[x(t_n), x(t_f)]$ to $[0, \infty)$ since the integral between $[0, x(t_n))$ and $(x(t_f), \infty)$ are zero. Thus, we have

$$C(\boldsymbol{r}) = \int_0^\infty e^{-x}\boldsymbol{c}(\boldsymbol{r}(t(x)), d)dx, \tag{10}$$

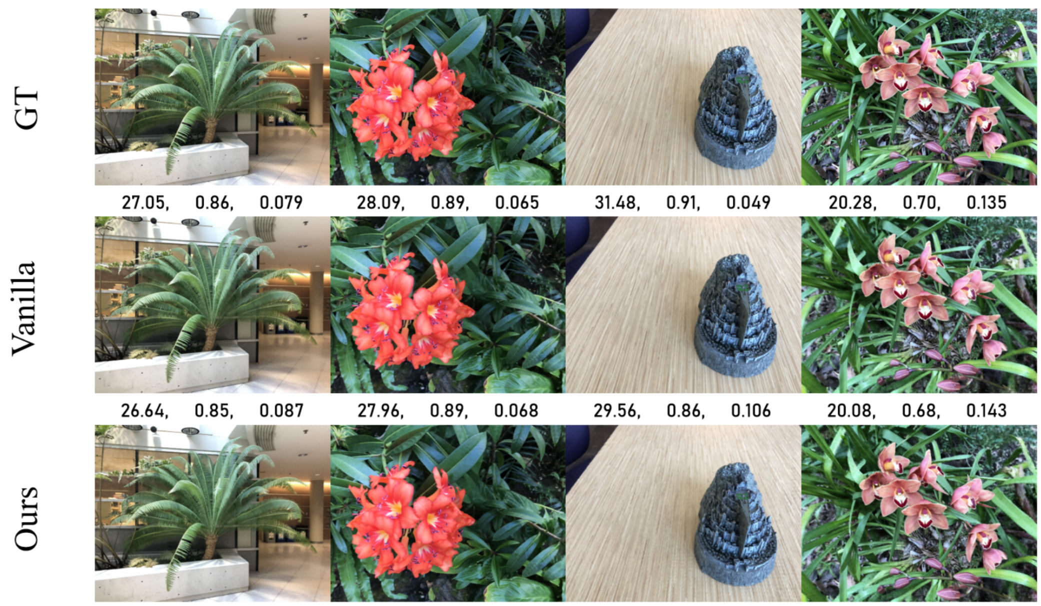

Figure 2. Qualitative comparison between GL-NeRF and vanilla NeRF on LLFF dataset, the small drop of quantitative performance doesn't affect the overall render quality. **The number in the figure represents PSNR, SSIM and LPIPS respectively for the image below them.**

Method	**LLFF**		**Blender**	
	Memory Usage	Time Usage	Memory Usage	Time Usage
Vanilla NeRF	3219.70 MB	7.54 s	4790.53 MB	18.68 s
GL-NeRF	3138.37 MB	6.32 s	3153.35 MB	9.30 s

Table 1. We compare GL-NeRF with the vanilla NeRF in terms of memory and time usage for rendering images.

a pure exponentially weighted integral with respect to the color function, which is of the exact same form as required by the Gauss-Laguerre quadrature.

3.1. Point selection with Gauss-Laguerre quadrature

Different from NeRF's sampling strategy, with the help of the Gauss-Laguerre quadrature, we can abandon the stage of "fine" sampling and replace it with a deterministic point selection strategy. Recall Eq. (11) is the integral variable for Eq. (10). This means if we want to use the Gauss-Laguerre quadrature to approximate Eq. (10), we have to choose points x_i that are the roots of the nth-degree Laguerre polynomials. Since every x_i is different and thus has a corresponding t_i following Eq. (11), we can choose t_i based on given value of x_i, as depicted in right of Fig. 1. Specifically, we want the integral Eq. (11) to be equal to the roots of an nth-degree Laguerre polynomial. Right of Fig. 1 gives an example of $n = 5$. After selecting the N_f points needed for computing the integral, we only feed these N_f points into the "fine" network instead of feeding $N_c + N_f$ points as done in the original NeRF. Here N_c stands for the coarse sample in original NeRF.

4. Experiments

4.1. Experimental setup

Datasets and evaluation metrics. We evaluate our method on the standard datasets: Blender and Real Forward Facing Dataset(LLFF) [19] as in [20]. We follow the standard training and test splits. We first use the training set to train a vanilla NeRF for each scene. Then we conduct render-only experiments with the vanilla method and our method. We plot the standard render quality evaluation metrics PSNR, SSIM [33] and LPIPS [38] with respect to the average time needed for rendering one image for each scene. We also compare the memory needed and the computation needed (in terms of FLOPS) for rendering between our method and the original NeRF.

4.2. Trade-off between render quality and time usage

We showcase that our method can be used for rendering novel views based on pretrained NeRF without further training. We plotted the quantitative metrics of GL-NeRF and original NeRF for an intuitive comparison in Fig. 3. It shows that our method achieves comparable results as the original NeRF while requiring less computation, leading

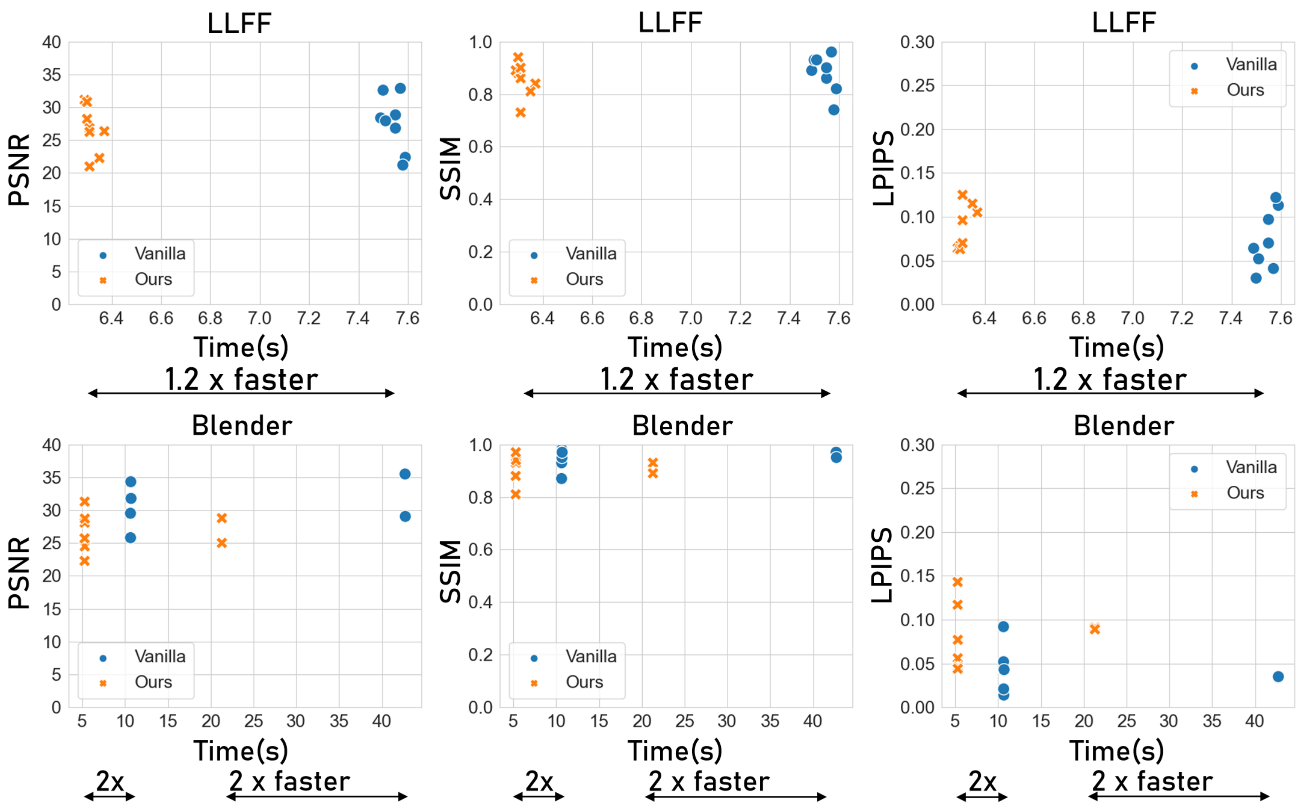

Figure 3. Comparison between GL-NeRF and original NeRF in terms of render time and quantitative metrics. Each point on the figure represents an individual scene. We showcase that with the drop of computational cost GL-NeRF provides, the average time needed for rendering one image is 1.2 to 2 times faster than the original NeRF. In the mean time, the overall performance remains almost the same despite some minor decreases.

to 1.2 to 2 times faster rendering. Fig. 3 shows that there are some minor drop in the overall render quality measured by PSNR, SSIM and LPIPS, we therefore visualize some qualitative results in Fig. 2 to show that the drops in these numbers do not have much effect on the visual quality of the images.

4.3. Efficiency in terms of time, memory and computational cost

In this section, we compare the overall computation needed for our method and the original NeRF. We theoretically compute the FLOPS for our method and the original NeRF to demonstrate the efficiency of our method. We also compare the GPU memory costs and rendering time.

FLOPS. For a single pixel in the image, the baseline method in LLFF needs to call the neural network $64 + (64 + 64) = 192$ times and in Blender it needs $128 + (128 + 64) = 320$ times, while our method only call the neural network $128 + 32 = 160$ times. We use [29] to compute the total FLOPS needed for rendering one pixel. Therefore, baseline method takes 47.435 MFLOPS for LLFF and 79.055 MFLOPS for Blender while our method remains 39.525 MFLOPS.

GPU memory costs and time needed. As can be seen from Tab. 1, GL-NeRF takes less time and memory for rendering images comparing to what it's been implemented on thanks to the reduction of computational cost.

5. Conclusion

In this paper, we propose GL-NeRF, a new perspective for computing volume rendering using the Gauss-Laguerre quadrature which guarantees the highest algebraic precision with a pre-defined point selection strategy. The main difference of GL-NeRF with other sample-efficient methods is that it requires no additional neural network for surface prediction, leading to strong flexibility (*i.e.* could be used without training). We conduct experiments to showcase that our method could achieve comparable results with the model it's been plugged into.

Acknowledgement

This work has been funded in part by the Army Research Laboratory (ARL) under grant W911NF-23-2-0007 and W911NF-19-2-0146, and the Air Force Office of Scientific Research (AFOSR) under grants FA9550-18-1-0097 and FA9550-18-1-0251.

References

[1] Relja Arandjelović and Andrew Zisserman. Nerf in detail: Learning to sample for view synthesis. *arXiv preprint arXiv:2106.05264*, 2021. 1

[2] 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. 1

[3] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5470–5479, 2022. 1

[4] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *European Conference on Computer Vision*, pages 333–350. Springer, 2022. 1

[5] Robert A Drebin, Loren Carpenter, and Pat Hanrahan. Volume rendering. *ACM Siggraph Computer Graphics*, 22(4): 65–74, 1988. 1

[6] 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. 1

[7] Stephan J Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. Fastnerf: High-fidelity neural rendering at 200fps. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14346–14355, 2021. 1

[8] Carl Friedrich Gauss. *Methodvs nova integralivm valores per approximationem inveniendi*. Dieterich, 1815. 2

[9] Walter Gautschi. *Numerical analysis*. Springer Science & Business Media, 2011. 5

[10] Peter Hedman, Pratul P Srinivasan, Ben Mildenhall, Jonathan T Barron, and Paul Debevec. Baking neural radiance fields for real-time view synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5875–5884, 2021. 1

[11] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics (ToG)*, 42(4):1–14, 2023. 1

[12] Abhijit Kundu, Kyle Genova, Xiaoqi Yin, Alireza Fathi, Caroline Pantofaru, Leonidas J Guibas, Andrea Tagliasacchi, Frank Dellaert, and Thomas Funkhouser. Panoptic neural fields: A semantic object-aware neural scene representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12871–12881, 2022. 1

[13] Andreas Kurz, Thomas Neff, Zhaoyang Lv, Michael Zollhöfer, and Markus Steinberger. Adanerf: Adaptive sampling for real-time rendering of neural radiance fields. In *European Conference on Computer Vision*, pages 254–270. Springer, 2022. 1, 2

[14] Liangchen Li and Juyong Zhang. l_0-sampler: An $l\_\{0\}$ model guided volume sampling for nerf. *arXiv preprint arXiv:2311.07044*, 2023. 1

[15] Zhengqi Li, Simon Niklaus, Noah Snavely, and Oliver Wang. Neural scene flow fields for space-time view synthesis of dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6498–6508, 2021. 1

[16] David B Lindell, Julien NP Martel, and Gordon Wetzstein. Autoint: Automatic integration for fast neural volume rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14556–14565, 2021. 1

[17] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. *Advances in Neural Information Processing Systems*, 33:15651–15663, 2020. 1

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

[19] Ben Mildenhall, Pratul P Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *ACM Transactions on Graphics (TOG)*, 38(4):1–14, 2019. 3

[20] 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. 1, 3

[21] Thomas Neff, Pascal Stadlbauer, Mathias Parger, Andreas Kurz, Joerg H Mueller, Chakravarty R Alla Chaitanya, Anton Kaplanyan, and Markus Steinberger. Donerf: Towards real-time rendering of compact neural radiance fields using depth oracle networks. In *Computer Graphics Forum*, pages 45–59. Wiley Online Library, 2021. 1, 2

[22] Julian Ost, Fahim Mannan, Nils Thuerey, Julian Knodt, and Felix Heide. Neural scene graphs for dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2856–2865, 2021. 1

[23] Martin Piala and Ronald Clark. Terminerf: Ray termination prediction for efficient neural rendering. In *2021 International Conference on 3D Vision (3DV)*, pages 1106–1114. IEEE, 2021. 1, 2

[24] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-nerf: Neural radiance fields for dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10318–10327, 2021. 1

[25] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14335–14345, 2021. 1

[26] Christian Reiser, Rick Szeliski, Dor Verbin, Pratul Srinivasan, Ben Mildenhall, Andreas Geiger, Jon Barron, and Peter Hedman. Merf: Memory-efficient radiance fields for real-

time view synthesis in unbounded scenes. *ACM Transactions on Graphics (TOG)*, 42(4):1–12, 2023. 1
[27] Yawar Siddiqui, Lorenzo Porzi, Samuel Rota Bulò, Norman Müller, Matthias Nießner, Angela Dai, and Peter Kontschieder. Panoptic lifting for 3d scene understanding with neural fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9043–9052, 2023. 1
[28] Vincent Sitzmann, Semon Rezchikov, Bill Freeman, Josh Tenenbaum, and Fredo Durand. Light field networks: Neural scene representations with single-evaluation rendering. *Advances in Neural Information Processing Systems*, 34: 19313–19325, 2021. 1
[29] Vladislav Sovrasov. ptflops: a flops counting tool for neural networks in pytorch framework, 2018. 4
[30] Matthew Tancik, Vincent Casser, Xinchen Yan, Sabeek Pradhan, Ben Mildenhall, Pratul P Srinivasan, Jonathan T Barron, and Henrik Kretzschmar. Block-nerf: Scalable large scene neural view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8248–8258, 2022. 1
[31] Mikaela Angelina Uy, Kiyohiro Nakayama, Guandao Yang, Rahul Krishna Thomas, Leonidas Guibas, and Ke Li. Nerf revisited: Fixing quadrature instability in volume rendering. *arXiv preprint arXiv:2310.20685*, 2023. 1
[32] Suhani Vora, Noha Radwan, Klaus Greff, Henning Meyer, Kyle Genova, Mehdi SM Sajjadi, Etienne Pot, Andrea Tagliasacchi, and Daniel Duckworth. Nesf: Neural semantic fields for generalizable semantic segmentation of 3d scenes. *arXiv preprint arXiv:2111.13260*, 2021. 1
[33] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004. 3
[34] Lee Westover. Interactive volume rendering. In *Proceedings of the 1989 Chapel Hill workshop on Volume visualization*, pages 9–16, 1989. 1
[35] Liwen Wu, Jae Yong Lee, Anand Bhattad, Yu-Xiong Wang, and David Forsyth. Diver: Real-time and accurate neural radiance fields with deterministic integration for volume rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16200–16209, 2022. 1
[36] 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. 1
[37] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields. *arXiv preprint arXiv:2010.07492*, 2020. 1
[38] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018. 3
[39] Shuaifeng Zhi, Tristan Laidlow, Stefan Leutenegger, and Andrew J Davison. In-place scene labelling and understanding with implicit scene representation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15838–15847, 2021. 1

GL-NeRF: Gauss-Laguerre Quadrature for Volume Rendering

Supplementary Material

A. Related works

Volume rendering. Volume rendering has been widely used in computer graphics and vision applications [5, 18, 34]. It maps a 3D scene onto 2D images by a weighted integral over the color of the points along the corresponding rays with a function of opacity (volume density) as weight. In practice, the integral is approximated using a finite sum over sampled points along the ray as derived in [18]. Implicit scene models like NeRF [20], Plenoxels [6] and 3D gaussians [11] and most of their follow-up all adopt this technique as the render pipeline. Since randomly sampling in space for approximating the integral may bring unnecessary information (*i.e*. sampling in empty space) that may cost extra computation, plenty of works aim to address that by introducing different techniques for better approximation of the component needed for volume rendering integral (*i.e*. volume density, radiance) [1, 14, 16, 21, 31, 35]. PL-NeRF [31] proposes to use piecewise linear function for approximating the volume density throughout the space, leading to fewer points needed for the "fine" stage sampling proposed by [20]. AutoInt and DIVeR [16, 35] introduce a neural network for approximating the integral of volume density instead of using Monte-Carlo sampling. DONeRF [21] reduces the sampled point needed for computing the integral by introducing a depth oracle neural network that predict surface position of the underlying scene and samples the points near the surface, which contributes the most to the visual effect in the images. Different from these previous works, Our work proposes to use the Gauss-Laguerre quadrature to directly improve the precision of the volume rendering integral itself, introduces no additional neural networks or data structures and remains in the simplest version, leading to its adaptability into any existing work that relies on volume rendering integral.

NeRFs. Neural Radiance Fields (NeRFs) have proved to be a powerful tool for novel view synthesis [20]. It uses a coordinate-based multi-layer perceptron (MLP) to represent the scene and render high-fidelity images from different views. The render is done by pixel-wise volumetric rendering [18] with density and color evaluated using the MLP on hundreds of sampled points along the ray. For modeling high-frequency information in the scene, NeRF uses positional encoding to map the input coordinates onto high-frequency bands. The success of NeRF has triggered an explosive emergence of follow-up works. There are plenty of works focusing on improving or extending the ability of NeRF towards different aspects. Aliasing along xy coordinates has been tackled [2], unbounded scenes [3, 26, 30, 37], dynamic scenes [15, 22, 24] and scenes with semantic information [12, 27, 32, 39] have been well explored and demonstrated the potential of implicit scene representation with NeRF. Nonetheless, NeRF requires plenty of time for training and rendering, blocking its way of being used for real-time rendering. The bottleneck of the computation time is the MLP used. There are two main branches of work for extending NeRF towards real-time rendering. The first branch introduces different data structure [4, 6, 7, 10, 25, 36] for scene representation. Another branch, in which our method falls, improves the sample efficiency of the model [13, 21, 23, 28] to accelerate NeRF rendering process. While previous works draw their intuition from the underlying physics perspective and thus need different formulations of the sampling strategy and different neural network architecture for predicting the surface position of the underlying scenes, we propose our method based on a mathematical observation while maintaining the overall pipeline. Benefiting from this, our work could be seamlessly incorporated into any existing NeRF-related works without further training. On the other hand, despite of being derived from the mathematical perspective, our method still intuitively satisfies the underlying physical constraints.

x_i	w_i
0.17	3.69×10^{-1}
0.90	4.19×10^{-1}
2.25	1.76×10^{-1}
4.27	3.33×10^{-2}
7.05	2.79×10^{-3}
10.76	9.08×10^{-5}
15.74	8.49×10^{-7}
22.86	1.05×10^{-9}

Table A1. Gauss-Laguerre quadrature look-up table when $n = 8$.

B. Intuitive understanding of the points selected using the Gauss-Laguerre quadrature

Since the points near the surface contribute the most to the final color of the pixel as discussed in [13, 21, 23], the optimal point selection strategy should choose points near the surface. The volume density, on the other hand, increases remarkably near the surface and remains close to zero at other areas. Therefore, the integral value of it Eq. (11) should also increases significantly near the surface and remains almost unchanged throughout the rest of

Blender		Avg.	Chair	Drums	Ficus	Hotdog	Lego	Mat.	Mic	Ship
PSNR↑	Vanilla	30.63	34.32	25.80	29.54	35.49	29.53	29.04	31.78	29.52
	Ours	29.18	32.43	24.38	26.92	33.91	29.49	27.27	31.55	27.47
SSIM↑	Vanilla	0.95	0.98	0.93	0.97	0.97	0.95	0.95	0.97	0.87
	Ours	0.93	0.97	0.91	0.94	0.96	0.95	0.92	0.97	0.84
LPIPS↓	Vanilla	0.037	0.014	0.052	0.021	0.034	0.042	0.035	0.044	0.092
	Ours	0.056	0.029	0.087	0.050	0.052	0.038	0.065	0.046	0.122
LLFF		Avg.	Fern	Flower	Fortress	Horns	Leaves	Orchid	Room	Trex
PSNR↑	Vanilla	27.62	26.82	28.37	32.59	28.83	22.38	21.20	32.87	27.93
	Ours	27.21	26.63	28.05	31.93	28.05	22.35	21.12	32.51	27.01
SSIM↑	Vanilla	0.88	0.86	0.89	0.93	0.90	0.82	0.74	0.96	0.92
	Ours	0.87	0.85	0.88	0.91	0.88	0.81	0.74	0.95	0.90
LPIPS↓	Vanilla	0.073	0.097	0.064	0.030	0.070	0.113	0.122	0.041	0.052
	Ours	0.087	0.121	0.075	0.043	0.089	0.117	0.131	0.053	0.069

Table A2. **Quantitative Results when training on Blender and LLFF Datasets.**

the space. Therefore, most of the points chosen using GL-NeRF should lie around the surface of the underlying scene.

$$x(t) = \int_{t_n}^{t} \sigma(\boldsymbol{r}(s))ds, \tag{11}$$

Consider a case when $n = 8$, we want to choose points $t_i, i = 1, 2, \ldots, 8$ such that $x(t_i)$ in Eq. (11) should be equal to the value x_i given in the look-up table Tab. A1. Notice that the first few value for x_i (say first three) are small so that they could be reached by the integral of volume density near the surface easily. These values has relatively larger weights assigned to them. Evaluating the color of these points using neural network and summing them up using the weights w_i given in Tab. A1 following Eq. (12) would contribute mostly to the pixel color. Notice that even though the last few x_i are quite large and may not be reached by Eq. (11) along the ray, their corresponding weights are so small that they almost couldn't affect the final result of the pixel color.

$$\int_0^{\infty} e^{-x} f(x)dx \approx \sum_{i=1}^{n} w_i f(x_i). \tag{12}$$

Hence, the points selected using GL-NeRF also corresponds to the points near the surface, like in previous works [13, 21, 23] that design different neural networks for estimating the surface position, but only without any additional neural networks. Therefore, thanks to the nice property of Gauss quadrature, ideally we can select the optimal points for computing volume rendering integral if the volume density estimation is oracle.

C. Gauss-Laguerre quadrature

The Gauss-Laguerre quadrature is an approximation formula for computing integrals over the semi-infinite interval $[0, +\infty)$ with the weight function e^{-x} and reads as

$$\int_0^{+\infty} e^{-x} f(x)dx \approx \sum_{k=0}^{n} w_k f(x_k). \tag{13}$$

Here $x_0, x_1, \cdots, x_n \in [0, +\infty)$ are the zeros of the Laguerre polynomial $L_{n+1} = L_{n+1}(x)$ of degree $(n+1)$:

$$L_{n+1}(x) = \frac{1}{(n+1)!} e^x \frac{d^{n+1}}{dx^{n+1}} (x^{n+1} e^{-x}),$$

for $n = -1, 0, 1, \cdots$, and the coefficients

$$w_k = \frac{1}{x_k [L'_{n+1}(x_k)]^2}, \quad k = 0, 1, 2, \cdots, n. \tag{14}$$

From the Leibniz formula, it is easy to see that $L_n(x)$ is a polynomial of degree n and the coefficient of x^n is $\frac{(-1)^n}{n!}$. In particular, we have

$$L_0 = 1, \qquad L_1 = 1 - x, \qquad L_2 = \frac{1}{2}x^2 - 2x + 1, \cdots.$$

The fundamental property of the Laguerre polynomials is

Theorem C.1. *The Laguerre polynomials $L_n = L_n(x)$ are orthogonal with respect to the weight function e^{-x}, that is,*

$$\int_0^{+\infty} e^{-x} L_n(x) L_m(x) dx = \begin{cases} 0, & n \neq m, \\ 1, & n = m. \end{cases}$$

Proof. Assume $m \leq n$ and set $g_k(x) = x^k e^{-x}$. From the Leibniz formula it follows that, for $j < k$, $g_k^{(j)}(x)$ is a product of xe^{-x} and a polynomial of degree $(k-1)$ and thereby $g_k^{(j)}(0) = 0 = g_k^{(j)}(+\infty)$ for $j < k$. Thus, we

Blender		Avg.	Chair	Drums	Ficus	Hotdog	Lego	Mat.	Mic	Ship
PSNR↑	Vanilla	30.63	34.32	25.80	29.54	35.49	29.53	29.04	31.78	29.52
	Ours	28.56	30.82	24.08	26.62	32.70	28.78	27.19	31.34	27.03
SSIM↑	Vanilla	0.95	0.98	0.93	0.97	0.97	0.95	0.95	0.97	0.87
	Ours	0.93	0.96	0.90	0.94	0.96	0.94	0.92	0.97	0.84
LPIPS↓	Vanilla	0.042	0.014	0.052	0.021	0.034	0.042	0.035	0.044	0.092
	Ours	0.070	0.050	0.098	0.055	0.059	0.047	0.070	0.044	0.135
LLFF		Avg.	Fern	Flower	Fortress	Horns	Leaves	Orchid	Room	Trex
PSNR↑	Vanilla	27.62	26.82	28.37	32.59	28.83	22.38	21.20	32.87	27.93
	Ours	26.53	26.27	28.19	31.12	26.81	22.27	20.99	30.38	26.24
SSIM↑	Vanilla	0.88	0.86	0.89	0.93	0.90	0.82	0.74	0.96	0.92
	Ours	0.85	0.84	0.88	0.89	0.86	0.81	0.73	0.93	0.89
LPIPS↓	Vanilla	0.074	0.097	0.064	0.030	0.070	0.113	0.122	0.041	0.052
	Ours	0.090	0.106	0.066	0.064	0.096	0.115	0.125	0.075	0.075

Table B3. **Render-only quantitative results on Blender and LLFF datasets.**

deduce that

$$\begin{aligned}
&n!m!\int_0^{+\infty} e^{-x}L_n(x)L_m(x)dx \\
=&\int_0^{+\infty} e^{-x}e^x g_n^{(n)}(x)e^x g_m^{(m)}(x)dx \\
=&\int_0^{+\infty} e^x g_m^{(m)}(x)dg_n^{(n-1)}(x) \\
=&g_n^{(n-1)}(x)[e^x g_m^{(m)}(x)]|_0^{+\infty} \\
&-\int_0^{+\infty}[e^x g_m^{(m)}(x)]'dg_n^{(n-2)}(x) \\
=&-g_n^{(n-2)}(x)[e^x g_m^{(m)}(x)]'|_0^{+\infty} \\
&+\int_0^{+\infty}[e^x g_m^{(m)}(x)]''dg_n^{(n-3)}(x) \\
=&\cdots\cdots \\
=&(-1)^n\int_0^{+\infty} g_n(x)[e^x g_m^{(m)}(x)]^{(n)}(x)dx
\end{aligned} \tag{15}$$

By the Leibniz formula, we have

$$\begin{aligned}
\left[e^x g_m^{(m)}(x)\right]^{(n)} &= \sum_{j=0}^{n}\frac{n!}{(n-j)!j!}(e^x)^{(n-j)}g_m^{(m+j)}(x) \\
=&e^x\sum_{j=0}^{n}\frac{n!}{(n-j)!j!}g_m^{(m+j)}(x)
\end{aligned}$$

and thereby

$$\begin{aligned}
&n!m!\int_0^{+\infty} e^{-x}L_n(x)L_m(x)dx \\
=&(-1)^n\sum_{j=0}^{n}\frac{n!}{(n-j)!j!}\int_0^{+\infty} x^n g_m^{(m+j)}(x)dx \\
=&\sum_{j=0}^{n-m}\frac{n!(-1)^{n+m+j}}{(n-j)!j!}\int_0^{+\infty}\frac{n!x^{n-m-j}}{(n-m-j)!}g_m(x)dx \\
=&\sum_{j=0}^{n-m}\frac{n!(-1)^{n+m+j}}{(n-j)!j!}\int_0^{+\infty}\frac{n!}{(n-m-j)!}x^{n-j}e^{-x}dx \\
=&(-1)^n\sum_{j=0}^{n-m}\frac{n!}{(n-j)!j!}(-1)^{m+j}\frac{n!(n-j)!}{(n-m-j)!} \\
=&(-1)^{n+m}\frac{(n!)^2}{(n-m)!}\sum_{j=0}^{n-m}\frac{(n-m)!}{j!(n-m-j)!}(-1)^j \\
=&(-1)^{n+m}\frac{(n!)^2}{(n-m)!}(1-1)^{n-m}.
\end{aligned}$$

Here the second equality is similar to that in Eq. (15) and the fourth uses

$$\int_0^{+\infty} x^{n-j}e^{-x}dx = (n-j)!.$$

This completes the proof. □

The orthogonality of the Laguerre polynomials ensures that the $L_n(x)$'s are linearly independent and $L_{n+1}(x)$ has $(n+1)$ distinct zeros $x_0, x_1, \cdots, x_n$ in $[0, +\infty)$ [1]. With the zeros, the coefficients w_k are chosen so that the following $(n+1)$ equalities

$$\int_0^{+\infty} e^{-x}x^j dx = \sum_{k=0}^{n} w_k x_k^j, \tag{16}$$

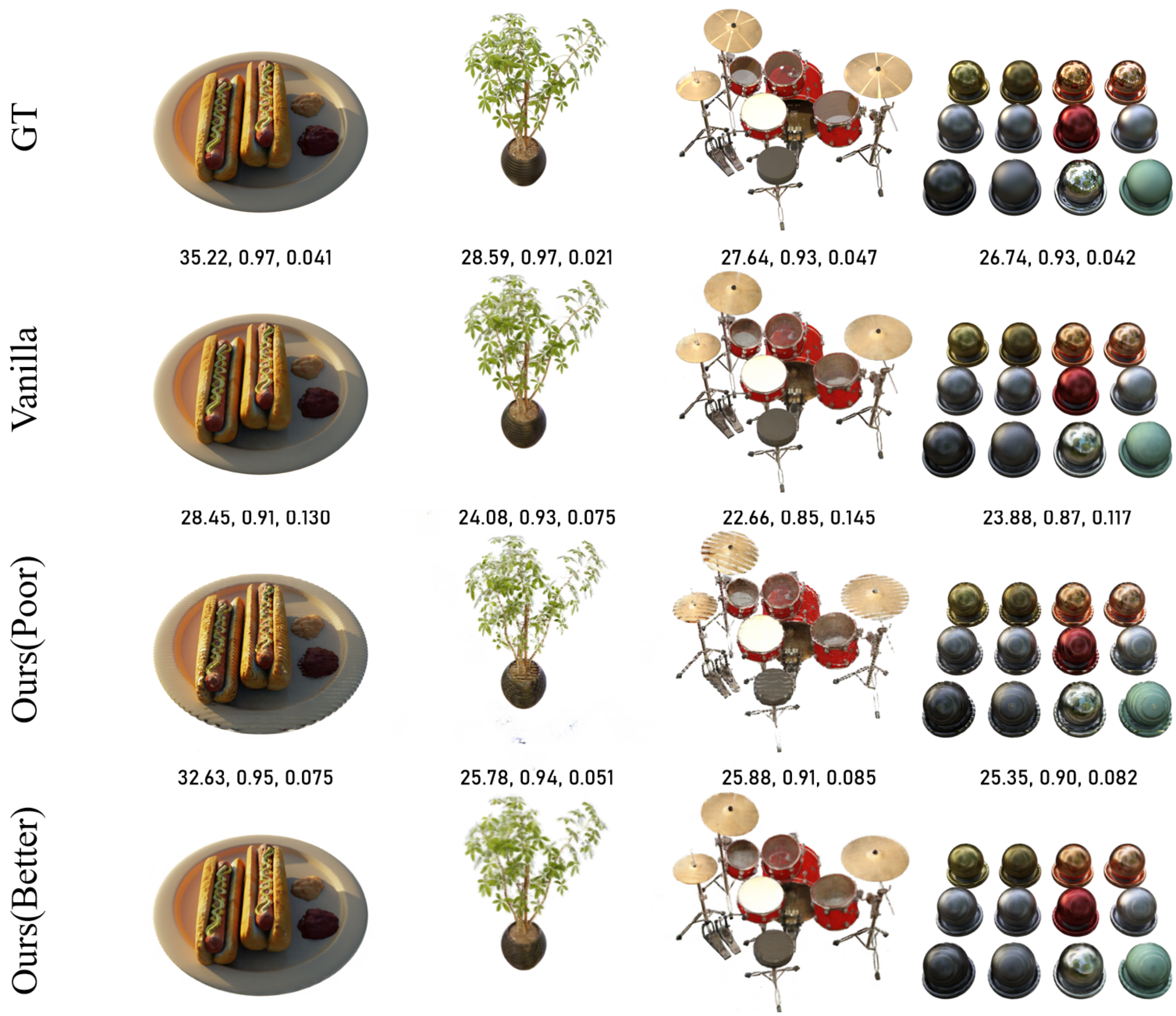

Figure B1. **Some qualitative results on Blender dataset. The number in the figure represents PSNR, SSIM and LPIPS respectively for the image below them.** Poor estimation of volume density throughout the entire space may lead to not finding points in the meaningful area, leading to the stripe-like mixture of foreground and background (row 3). By using "coarse" network for better volume density estimation, we manage to reduce the stripe-shape effect brought by white background (row 4).

hold for $j = 0, 1, \cdots, n$. This leads to a system of $(n+1)$ linear algebraic equations for the unknowns w_k and the corresponding coefficient matrix is the Vandermonde matrix $[x_k^j]_{(n+1)\times(n+1)}$. The latter is invertible since the zeros are distinct and therefore the w_k's are uniquely determined. The specific expressions of the w_k's are given in Eq. (14) [1].

It is remarkable that all the coefficients w_k are non-negative. This important property ensures the stability and convergence of the Gauss-Laguerre quadrature [1]. Moreover, we have

Theorem C.2. *The algebraic precision of the Gauss-Laguerre quadrature Eq.* (13) *is* $(2n+1)$ *exactly. Namely, "≈" in Eq.* (13) *is "=" if* $f(x)$ *is a polynomial of degree* $(2n+1)$ *and is not "=" if* $f(x)$ *is a polynomial with degree higher than* $(2n+1)$.

Proof. Notice that

$$(n+1)!L_{n+1}(x) = (-1)^{n+1}\Pi_{k=0}^{n}(x - x_k)$$

is a polynomial of degree $(n+1)$. Since

$$0 < \int_0^{+\infty} e^{-x} L_{n+1}^2(x)dx \neq 0 = \sum_{k=0}^{n} w_k L_{n+1}^2(x_k),$$

the precision is less than $2(n+1)$. On the other hand, for any polynomial $p = p(x)$ of degree $(2n+1)$ there are two polynomials of degree n such that $p(x) = q(x)L_{n+1}(x) + r(x)$. Notice that $q(x)$ can be written as a linear combination of $L_0(x), L_1(x), \cdots, L_n(x)$. Compute

$$\begin{aligned}
\sum_{k=0}^{n} w_k p(x_k) = & \sum_{k=0}^{n} w_k [q(x_k)L_{n+1}(x_k) + r(x_k)] \\
= & \sum_{k=0}^{n} w_k r(x_k) \\
= & \int_0^{+\infty} e^{-x} r(x) dx \\
= & \int_0^{+\infty} e^{-x} [q(x)L_{n+1}(x) + r(x)] dx \\
= & \int_0^{+\infty} e^{-x} p(x) dx.
\end{aligned}$$

Here the third equality is due to Eq. (16) (the choice of w_k) and the fourth is due to the orthogonality of $L_{n+1}(x)$ and $q(x)$ with respect to the weight function. Hence the proof is complete. □

For further details on the Gauss-Laguerre quadrature and for other Gauss quadratures, the interested reader is referred to the book [9].

D. Quantitative results on Blender and LLFF datasets

In this part, we present our quantitative results on Blender and LLFF datasets in Tab. B3. We also use GL-NeRF to train a neural network for comparison with the baseline in Tab. A2.

E. Failure cases

We observe that for some scenes in the Blender dataset, the rendering quality in terms of PSNR falls behind the baseline. We visualize the corresponding images and find a universal phenomenon that GL-NeRF tends to provide stripe-shape textures on the object. Recall that we have to select t that makes Eq. (11) equal to the root of Laguerre polynomials. However, t may not exist since $x(t)$ is bounded by $\max x = x(t_f) < \infty$. Therefore, if the estimation of volume density is poor, we may end up with points in the background, leading to the stripe-shape texture as shown in the third row in Fig. B1. Intuitively, during training, the majority in input to "fine" network are cluttered near the surface. Therefore, even if the estimation of volume density is small, by aggregating over all the points, the final render result could still match the ground truth color of the pixel. Smaller estimation of volume density could make the point selection strategy in GL-NeRF fail to select points near the surface and instead choose points at the camera far plane. A workaround is to use the "coarse" network trained simultaneously with the "fine" network. Since the input points to the "coarse" network are uniformly distributed in the scene, "coarse" network has to assign bigger value (than "fine" network) for the points near the surface to minimize the loss function.

This could lead to a much more reasonable estimation of volume density. We therefore turn to the "coarse" network by first querying it with "coarse" samples for volume density estimation, selecting points based on the estimation and finally rendering with the selected point from GL-NeRF. We visualize some of the results in Fig. B1, denoted as "Ours(Better)", and the stripe disappears. This suggests that GL-NeRF relies on the estimation of volume density heavily and that preciser estimation of volume density could significantly improve the performance of GL-NeRF. Despite this drawback, the results on LLFF dataset suggests that our method could still be incorporated into real-world scenes seamlessly.