QuickNeRF: Neural Radiance Fields with Hash Encoding

Yizhou Xu
Group 28, DD2424, Spring 2023
yizhoux@kth.se

Mehdi Moshtaghi
Group 28, DD2424, Spring 2023
mehdimo@kth.se

Abstract

Neural Radiance Fields (NeRF) incorporates simple MLPs to reconstruct 3D scenes from 2D images and has achieved remarkable results. However, its training process could be slow and computationally demanding. To overcome this limitation, InstantNGP introduces multi-resolution hash encoding, a novel approach in encoding positions that enables training a NeRF in seconds. In this project, we explore the efficacy of hash encoding in accelerating the NeRF training process. Through extensive experiments and analysis, we demonstrate that NeRF with hash encoding can be trained in fewer steps and with smaller models while still achieving competitive results.

1 Introduction

View synthesis is an important task in computer vision. The task is to render new views of a scene from a given set of images with their camera poses, thus requiring the reconstruction of 3D scenes from 2D images. NeRF (1) proposed a simple Multi Layer Perceptron (MLP) to complete that task, introducing two improvements to reconstruct high-resolution complex scenes: 1) Positional encoding similar to Transformer (2); 2) hierarchical volume sampling, where they used two separate MLP networks, one for coarse reconstruction with stratified sampling procedure and the other network for finer reconstruction with informed sampling based on the result of the coarse network.

Though NeRF can generate satisfactory images of new views, for each individual scene a separate model is needed and the training would take hours. To make NeRF lighter and faster, Instant Neural Graphics Primitives (InstantNGP) (3) proposed a multi-resolution hash encoding to ease the learning difficulty. It maps positions into grids with different resolutions and uses a highly efficient hash table to get the embeddings and uses interpolation to get the encoding of the position. Besides hash encoding, InstantNGP also used a fused MLP network (4) written in C++ and CUDA to further reduce the training time to seconds.

In this project, our goal is to replicate multi-resolution hash encoding proposed in InstantNGP (3) for NeRF(1) in Python and observe the effect of multi-resolution hash encoding. While it could be impossible to train the model in seconds without advanced GPUs and C++/CUDA implementation, we expect to achieve competitive results compared to NeRF (1) in relatively less training time.

2 Related Work

NeRF (1) utilized the positional encoding similar to Transformers (2) when representing positions in reconstructing 3D scenes; however, it has been established that the encoding stage plays a crucial role in improving both the efficiency and performance of the method. Some works further improved frequency encoding ideas by randomly oriented parallel wavefronts (5) and level-of-detail filtering (6). More works extended the encoding part with learnable parameters in given data structures, such

Preprint. Under review.

as grids (7; 8; 9; 10; 11) and trees (12). But many parameters are found redundant and thus result in less efficient training. Muller et al. (3) proposed hash encoding, which maps positions into hash table embeddings in different resolution levels to ensure both adaptivity and efficiency.

3 Background

3.1 NeRF

NeRF incorporates two key components, stratified sampling and positional encoding, to address crucial aspects of sampling efficiency and spatial representation. Stratified sampling is employed to achieve a balanced distribution of sampled points along the rays, across the scene, throughout the training process. It first partitions the bounded world frame into evenly spaced bins, then uniformly samples a point from within each bin. This sampling process ensures a detailed coverage of the whole scene in the long run due to the randomness.

Furthermore, NeRF leverages positional encoding as described in eqn. 1 to encode spatial information. Given that it represents scenes as continuous volumetric functions, positional encoding transforms 3D coordinates of sampled points into continuous values, facilitating their processing by neural networks. This integration enables the MLP to capture the high- and low-frequency details inherent in the scene, mitigating potential artefacts, leading to accurate rendering and synthesis of images from novel viewpoints (5).

$$enc(x) = (sin(2^0 x), sin(2^1 x), ..., sin(2^L - 1x), cos(2^0 x), cos(2^1 x), ..., cos(2^L - 1x)) \quad (1)$$

The combined application of stratified sampling and positional encoding in NeRF contributes significantly to its ability to produce high-fidelity reconstructions and realistic renderings in diverse computer vision and graphics applications.

3.2 Multi-resolution Hash Encoding

Distinct from NeRF, InstantNGP (3) partitions the bounded world frame into multiple resolution levels of voxel grids.

$$N_l = \left\lfloor N_{min}.b^l \right\rfloor \quad (2)$$

$$b = \exp\left(\frac{\ln N_{max} - \ln N_{min}}{L - 1}\right) \quad (3)$$

Where N_l denotes the resolution, geometrically increasing from a defined minimum value by the growth factor b to reach a defined maximum value.

Parameter	Symbol
Number of resolution level	L
Max. number of entries per resolution level	T
Number of dimensions per entry	F
Coarsest resolution	N_{min}
Finest resolution	N_{max}

Table 1: Hash encoding hyper-parameters

Each voxel has 2^3 integer corner vertices (add up to $(N_l + 1)^3$ corners for each resolution level in the whole grid). All corners are mapped into a fixed-size feature vector array with size T. For coarse levels where a dense grid requires fewer than T parameters, i.e., $(N_l + 1)^3 \leq T$, this mapping is 1:1. At finer levels, a sparse hash function(13) is used $h : \mathbb{Z}^d \to \mathbb{Z}^T$ to index into the array, treating it as a hash table:

$$h(x) = (\oplus_{i=1}^{d} x_i \pi_i) \mod T \quad (4)$$

where $\oplus$ denotes bit-wise XOR operation and π are unique large prime numbers, and $\pi_1 = 1, \pi_2 = 2,654,435,761, \pi_3 = 805,459,861$ are chosen, similar in the original paper.

Each sampled input coordinate from rays is scaled by each resolution level and then is discretized by rounding up and down to the voxel corners it resides.

$$x_l = xN_l \tag{5}$$

$$\text{Corresponding Voxel corners} : \lfloor x_l \rfloor, \lceil x_l \rceil \in \mathbb{Z}^3 \tag{6}$$

Lastly, the embedding vector of a sampled input point is computed at each resolution level via d-linearly interpolating its surrounding voxel corners within its hypercube. In other words, the interpolation weight is defined as $w_l := x_l - \lfloor x_l \rfloor$.

these embeddings are concatenated to form the final embedding of the point and the input to the neural network.

explicit collision handling. Sparse detail is stored in the array using gradient-based optimization and resolved through the neural network $m(y; \Phi)$ for collision resolution. The number of trainable encoding parameters θ is thus $O(T)$ and bounded by $T \cdot L \cdot F$, which in our case is always $T \cdot 16 \cdot 2$ (Table 1)."

Although in this method an explicit collision handling is not employed, it relies on gradient-based optimization to capture suitable sparse detail in the array, and the subsequent neural network for collision resolution. The number of trainable encoding parameters is therefore $O(T)$ and bounded by T, L, F. It is important to note that nearby inputs with equal integer coordinates $\lfloor x_l \rfloor$ are not considered collisions. A collision occurs only when different integer coordinates hash to the same index, and such collisions are pseudo-randomly distributed throughout space by the sparse hash function(13), and the probability of them occurring simultaneously at every level for a given pair of points is statistically unlikely.

The rest of the project is similar to the original NeRF. We refer the reader to read that for more information.

4 Implementation

In this work, we implemented Multi-resolution Hash Encoding purely in Python and PyTorch.

Our implementation is based on a vanilla NeRF implemented by nerf-pytorch (14). We implemented a smaller NeRF model, the Multi-resolution Hash Encoding and added two more evaluation criteria (SSIM and LPIPS) besides PSNR on our own. We also followed InstantNGP (3) to appropriately initialize the weights, scale the input and set the optimizer. Our implementation is inspired by HashNeRF-pytorch (15). For the spherical harmonic encoding for views, we directly use the implementation from HashNeRF-pytorch. Our implementation can be found at `https://gits-15.sys.kth.se/yizhoux/HashNeRF`.

5 Experiments

5.1 Data Preparation

In this study, we select four distinct scenes from two diverse datasets. Specifically, we choose to include LEGO, MICROPHONE, and SHIP from the NeRF synthetic dataset (1), while the ORCHID scene is sourced from the local light field fusion (llff) dataset (16).

5.2 Metrics

The following evaluation metrics are adopted.

- **Peak Signal-to-Noise Ratio** (PSNR) (higher is better) is a metric used to quantify the quality of a reconstructed image by measuring the ratio of the maximum possible signal power to the power of the distortion or noise.
- **Structural Similarity Index Measure** (SSIM) (higher is better) is a metric used to assess the similarity between two images by measuring the perceived structural information, luminance, and contrast, taking into account the human visual system characteristics.

- **Learned Perceptual Image Patch Similarity** (LPIPS) (lower is better) is a metric used to quantify the perceptual similarity between two images by leveraging a deep neural network trained on human perception to capture the visual differences at the patch level. The backbone model we use is VGG-16.

5.3 Experimental Setup

5.3.1 Models and Hyperparameters

Baseline Our baseline is vanilla NeRF without hash encoding, we will call it NeRF in the following experiments. The NeRF model is eight layers with a hidden dimension of 256. We use both a coarse model and a fine model to sample points appropriately according to (1). The fine model has the same architecture as the coarse model, and the two models are optimized simultaneously.

HashNeRF For NeRF with hash coding, we implement a smaller model. Following InstantNGP (3), the smaller NeRF model consists of two concatenated MLPs: a density MLP followed by a colour MLP. The density MLP has only one 64-dimension hidden layer and the colour MLP has two 64-dimension hidden layers. For the hash embedding, we set the hash table size as 2^{19} and the finest resolution as 65536. Other settings remain the same as InstantNGP. We use the same coarse and fine settings as the baseline. We initialize the hash embedding weights in $\mathcal{U}(-0.0001, 0.0001)$.

5.3.2 Training Details

We use Adam optimizer for both NeRF and HashNeRF. For the baseline NeRF model, we train the model with a learning rate of 5e-4. The batch size is set to 1,024, and the total number of training steps is 50,000. For HashNeRF, we use Adam optimizer with $\alpha = 0.9$ and $\beta = 0.99$. We set an L2 decay weight as 1e-6 for the model weights but not the hash embeddings. For the hash embeddings, the ϵ in the Adam optimizer is set to 1e-15 for faster convergence. The learning rate is set to 0.01 in the beginning and decays exponentially to 1e-6. We train the HashNeRF model for 10,000 steps with a batch size of 4,096. Both models are trained on a single NVIDIA RTX-2060 GPU.

5.4 Main Results

In Table 2, we report the performance comparison between NeRF models with and without hash encoding. As can be seen from the table, in all four scenes, NeRF with hash encoding (the second line in each scene) can achieve comparable results with a smaller model in fewer training steps. In some scenes the PSNR of HashNeRF doesn't outperform the baseline, but the other two metrics outperform NeRF or are at least comparable. Especially in the ORCHID scene, the LPIPS of HashNeRF outperforms the baseline significantly. Since SSIM and LPIPS are considered more similar to human visual systems, the generating results of our HashNeRF are generally comparable to NeRF while a much smaller model can be used and fewer training steps are required. We also show the final rendering result from one of the views for all scenes in Figure 1, we can see that qualitatively the generated images of HashNeRF are comparable to that of NeRF.

	Model	**Param.**	**Steps**	**PSNR ↑**	**SSIM ↑**	**LPIPS ↓**
LEGO	NeRF	1.19M	50k	**28.73**	0.937	0.073
	HashNeRF (Ours)	19.2k	10k	28.44	**0.940**	**0.067**
MICROPHONE	NeRF	1.19M	50k	**30.53**	**0.967**	**0.051**
	HashNeRF (Ours)	19.2k	10k	28.19	0.960	0.055
SHIP	NeRF	1.19M	50k	**27.34**	0.834	0.194
	HashNeRF (Ours)	19.2k	10k	26.37	0.834	**0.175**
ORCHID	NeRF	1.19M	50k	**20.57**	0.686	0.297
	HashNeRF (Ours)	19.2k	10k	17.58	**0.688**	**0.227**

Table 2: Performance comparison between NeRF models with and without hash encoding.

5.5 Comparison of Convergence

In Figure 2 and 3 we report the training curve of PSNR in the LEGO scene and the ORCHID scene. The orange curves represent HashNeRF and the blue curve represents NeRF (baseline). The figures show that HashNeRF converges much faster than vanilla NeRF, indicating that hash encoding reduces the difficulty of learning.

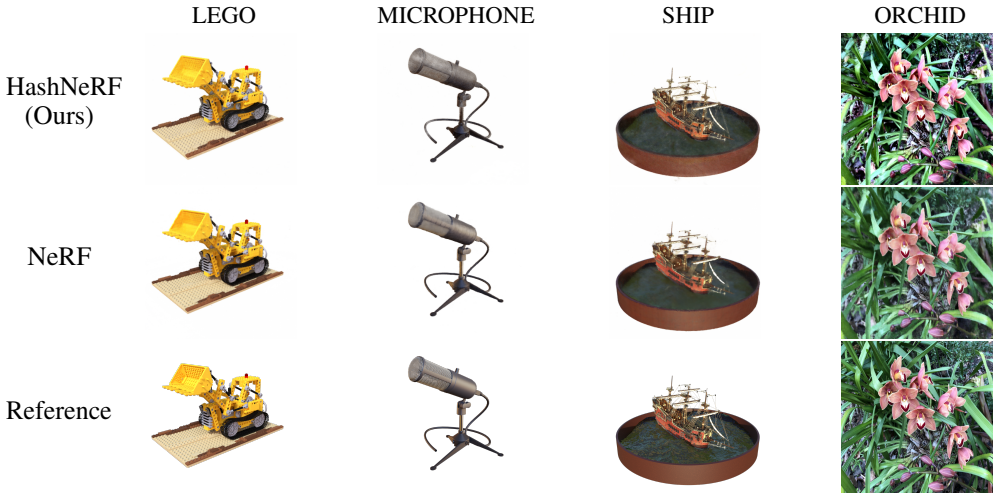

Figure 1: Comparison of final rendered figures

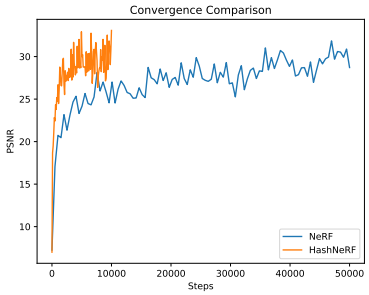

Figure 2: PSNR curve in Lego scene

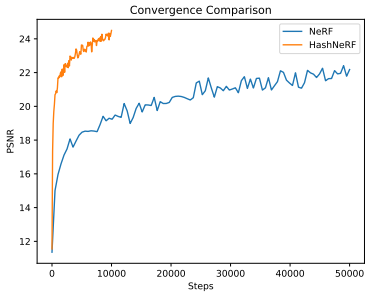

Figure 3: PNSR curve in Orchid scene

5.6 Qualitative Observation

In Figure 4, we present the qualitative comparison when the vanilla NeRF and our HashNeRF are trained for the same steps in the LEGO scene. We demonstrate the rendered figures when models are trained for 1,000, 5,000 and 10,000 steps. We can see from the figures that when trained for only 1,000 steps, NeRF can only synthesize a very blurry image while HashNeRF can produce a much clearer one. This trend continues when both models are trained for longer, suggesting that HashNeRF learns faster from a qualitative evaluation perspective.

5.7 Impact of Hash Map Size

We also investigate the impact of one of the most important hyperparameters in hash encoding: hash table size. The hash table size is expressed by the value T. We show both qualitative and quantitative results in Table 3 and Figure 5. The experiments are conducted in the LEGO scene. As can be seen from the table and the figures, a larger hash table size generally yields a better result. However, resource costs would increase exponentially, and there is not much gain when T is already large enough. So an appropriate T should be chosen to balance the performance and resources.

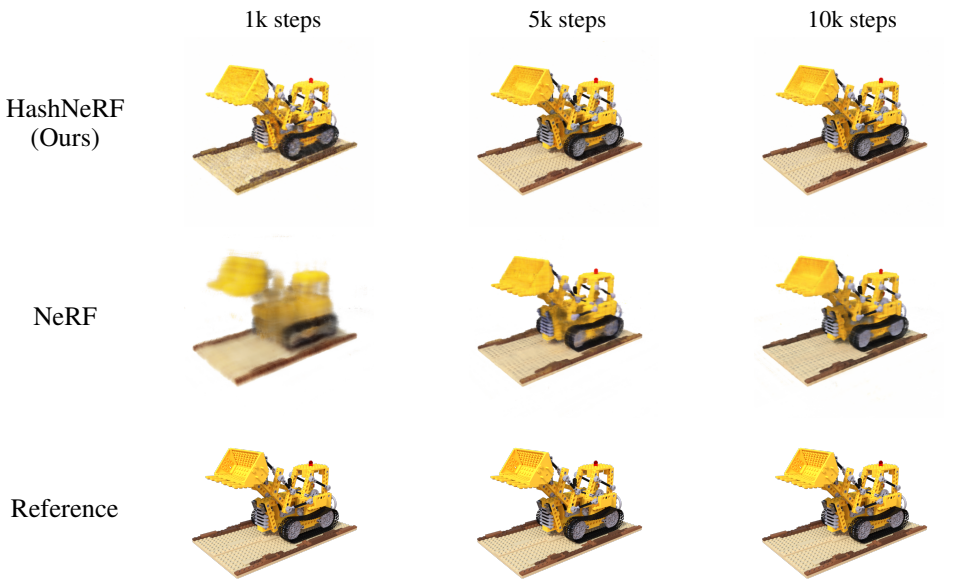

Figure 4: Comparison of generated images at different training stages.

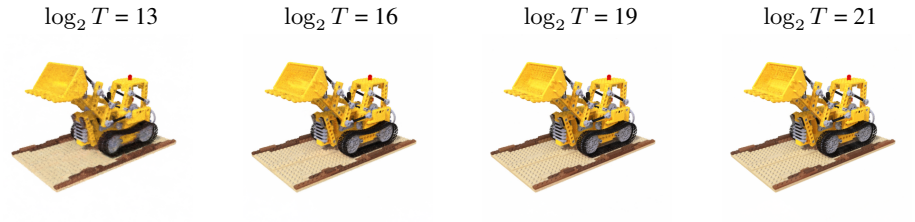

Figure 5: Comparison of different hash table sizes

$\log_2 T$	**PSNR** ↑	**SSIM** ↑	**LPIPS** ↓
13	25.87	0.878	0.141
16	27.55	0.921	0.094
19	28.45	0.940	0.067
21	**28.55**	**0.943**	**0.061**

Table 3: Performance comparison with different hash table size T.

6 Conclusion

In this project, we successfully incorporated hash encoding, as introduced by Muller et al. (2022) (3), into the NeRF framework. Our implementation demonstrated that the utilization of hash encoding in NeRF led to significantly faster convergence and required a considerably smaller model size. Moreover, we conducted an investigation into the influence of hyperparameters associated with hash encoding, such as the hash table size, which further enriched our comprehension of this encoding technique. However, due to resource constraints, our implementation was unable to surpass the performance achieved in the InstantNGP paper, and the speed of our model is much slower than that because it is a pure Python implementation.

References

[1] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," *Communications of the ACM*, vol. 65, no. 1, pp. 99–106, 2021.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[3] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," *ACM Transactions on Graphics (ToG)*, vol. 41, no. 4, pp. 1–15, 2022.

[4] T. Müller, "tiny-cuda-nn," 4 2021. [Online]. Available: https://github.com/NVlabs/tiny-cuda-nn

[5] M. Tancik, P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. Barron, and R. Ng, "Fourier features let networks learn high frequency functions in low dimensional domains," *Advances in Neural Information Processing Systems*, vol. 33, pp. 7537–7547, 2020.

[6] J. T. Barron, B. Mildenhall, M. Tancik, P. Hedman, R. Martin-Brualla, and P. P. Srinivasan, "Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 5855–5864.

[7] R. Chabra, J. E. Lenssen, E. Ilg, T. Schmidt, J. Straub, S. Lovegrove, and R. Newcombe, "Deep local shapes: Learning local sdf priors for detailed 3d reconstruction," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIX 16*. Springer, 2020, pp. 608–625.

[8] L. Liu, J. Gu, K. Zaw Lin, T.-S. Chua, and C. Theobalt, "Neural sparse voxel fields," *Advances in Neural Information Processing Systems*, vol. 33, pp. 15 651–15 663, 2020.

[9] I. Mehta, M. Gharbi, C. Barnes, E. Shechtman, R. Ramamoorthi, and M. Chandraker, "Modulated periodic activations for generalizable local functional representations," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 14 214–14 223.

[10] S. Peng, M. Niemeyer, L. Mescheder, M. Pollefeys, and A. Geiger, "Convolutional occupancy networks," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*. Springer, 2020, pp. 523–540.

[11] C. Sun, M. Sun, and H.-T. Chen, "Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 5459–5469.

[12] T. Takikawa, J. Litalien, K. Yin, K. Kreis, C. Loop, D. Nowrouzezahrai, A. Jacobson, M. McGuire, and S. Fidler, "Neural geometric level of detail: Real-time rendering with implicit 3d shapes," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 11 358–11 367.

[13] M. Teschner, B. Heidelberger, M. Müller, D. Pomerantes, and M. H. Gross, "Optimized spatial hashing for collision detection of deformable objects," in *International Symposium on Vision, Modeling, and Visualization*, 2003.

[14] L. Yen-Chen, "Nerf-pytorch," https://github.com/yenchenlin/nerf-pytorch/, 2020.

[15] Y. Bhalgat, "Hashnerf-pytorch," https://github.com/yashbhalgat/HashNeRF-pytorch/, 2022.

[16] B. Mildenhall, P. P. Srinivasan, R. Ortiz-Cayon, N. K. Kalantari, R. Ramamoorthi, R. Ng, and A. Kar, "Local light field fusion: Practical view synthesis with prescriptive sampling guidelines," *ACM Transactions on Graphics (TOG)*, 2019.