Temporal Interpolation Is All You Need for Dynamic Neural Radiance Fields Supplementary Materials

Sungheon Park, Minjung Son, Seokhwan Jang, Young Chun Ahn, Ji-Yeon Kim, Nahyup Kang
Samsung Advanced Institute of Technology (SAIT)
{sh2019.park, minjungs.son, swan.jang, ychun.ahn, jiyeon31.kim, nahyup.kang}@samsung.com

A. Network Details

The detailed structures of the networks for the neural and grid representation are explained in this section.

A.1. Neural Representation

The detailed network structure of the neural representation is illustrated in Fig. 1. Input 3D position $\mathbf{x}$ and the embedding vector $\mathbf{z}_t$ are encoded via the positional encoding used in [1]. We set the maximum frequency levels of the positional encoding to 8 for $\mathbf{x}$ and 3 for $\mathbf{z}_t$. We adopted windowed positional encoding from [3], which weights the frequency bands of the positional encoding using a window function.

After the feature vector $\mathbf{v}(\mathbf{x}, t)$ is extracted, it is fed into the template NeRF which consists of 8-layer MLPs with hidden size of 256 with ReLU activations, and one additional layer with hidden size of 128 for RGB color estimation. View direction and optional appearance code are also used as inputs for the RGB color estimation. We used the appearance code only for the DyNeRF dataset where an embedding vector is assigned to each camera.

Our implementation of the neural representation is based on the code from [3] which is built using JAX [5].

A.2. Grid Representation

As depicted in Fig. 2, the NeRF MLP of the grid representation consists of 3-layer network with hidden size of 128 for density estimation, and one additional layer with hidden size of 128 for RGB color estimation. View direction is encoded using spherical harmonics of degree 4 following the implementation of [2]. ReLU activations are used in all layers.

Our implementation of the grid representation is based on the code from [2] which is implemented using C++ and CUDA.

B. Training Details

Hyperparameter settings and training details for each dataset are described in this section. For all experiments, network parameters are optimized using ADAM optimizer [6].

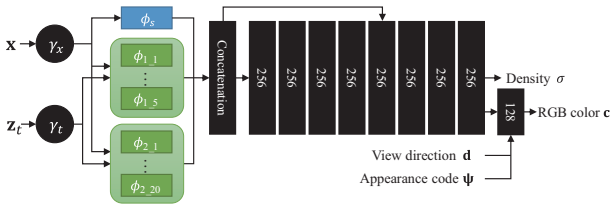


Figure 1. The network architecture of the neural representation. Input 3D position ($\mathbf{x}$) and the embedding vector for time t ($\mathbf{z}_t$) are fed to the network after the positional encoding [1] (γ_x and γ_t). The template NeRF consists of 8 layers of MLP with hidden size of 256 and one additional layer for RGB color estimation.

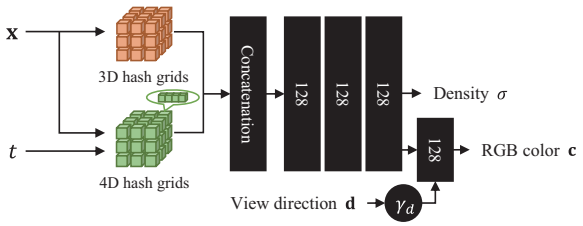


Figure 2. The network architecture of the grid representation. The extracted feature vector is fed to the template NeRF that has 3 layers of MLP with hidden size of 128 and one additional layer for RGB color estimation. View direction $\mathbf{d}$ is encoded using spherical harmonics as in [2].

B.1. Neural Representation

We maintained two template NeRFs for optimization as in [1], one of which is trained from the sampled points that uses stratified sampling, and the other is trained using importance sampling for ray sampling strategy. We used 8 V100 GPUs or 4 A100 gpus to train the neural representation model. Each minibatch contains inputs that are sampled from 6,144 rays. It took approximately one day for training in D-NeRF and HyperNeRF datasets, and two days

in DyNeRF datasets.

D-NeRF For D-NeRF datasets, initial learning rate is set to 0.002 and exponentially decayed to 0.0002 for 300,000 iterations. All training images are resized to 400×400 following the implementation of [4]. The smoothness weight λ is set to 0.01.

HyperNeRF For HyperNeRF datasets, initial and final learning rate are set to 0.001 and 0.0001 respectively. The smoothness weight λ is set to 0.001. Training images are scaled by 0.25 and 0.5 for *vrig* dataset and *interp* dataset respectively.

DyNeRF The training images are downsized to 1K resolution (1000×750). We set the initial and final learning rate to 0.001 and 0.00001 respectively. The smoothness weight λ is set to 0.01, and the network is optimized for 600,000 iterations.

B.2. Grid Representation

The initial learning rate is set to 0.01 and is multiplied by 0.33 for every 10,000 iterations starting 20,000 iterations. As in [2], we maintain the occupancy grid which is used to speed up training and rendering. To save the occupancy information of a whole sequence to a single occupancy grid, we assign random time frame value in addition to the input 3D points when querying the occupancy of the grid. Moreover, we adjust the decay weight of the values in the occupancy grid to 0.99 and set the threshold for culling to 0.0001.

C. Derivation of the Smoothness Term

We elaborate on the derivation of the smoothness term used in the grid representation, which is discussed in Section 3.3 of the main text. Applying the smoothness constraint to adjacent frames, as in the neural representation, the smoothness term becomes

$$L_s = \|\mathbf{v}_d(\mathbf{x}, t) - \mathbf{v}_d(\mathbf{x}, t+1)\|_2^2. \tag{1}$$

Here, we assume that $\mathbf{x}$ is located on the 3D grid point without loss of generality, and we assume that the time frames t and $t+1$ lie between two grid points of time t_a and t_b ($t_a < t_b$). Then, the output feature vector becomes a linear interpolation of the feature vectors of the two grid points, *i.e.*,

$$\mathbf{v}_d(\mathbf{x}, t) = s h_4(\mathbf{x}, t_a) + (1-s) h_4(\mathbf{x}, t_b), \tag{2}$$

where $s = \frac{t_b - t}{t_b - t_a}$. Changing t from t to $t+1$ makes the weight s decrease. Let ϵ denote the decrease in the weight, i.e.,

$$\frac{t_b - (t+1)}{t_b - t_a} = \frac{t_b - t}{t_b - t_a} - \frac{1}{t_b - t_a} = s - \epsilon. \tag{3}$$

, where $\epsilon = \frac{1}{t_b - t_a}$.

Then, the feature vector at time $t+1$ can be calculated as

$$\mathbf{v}_d(\mathbf{x}, t+1) = (s-\epsilon) h_4(\mathbf{x}, t_a) + (1-s+\epsilon) h_4(\mathbf{x}, t_b). \tag{4}$$

Substituting Eq. (2) and Eq. (4) to Eq. (1) yields

$$L_s = \epsilon^2 \|h_4(\mathbf{x}, t_a) - h_4(\mathbf{x}, t_b)\|_2^2. \tag{5}$$

When the grid resolution is fixed, ϵ is proportional to $\frac{1}{n_f}$. Hence, we can obtain the smoothness term as the following form:

$$L_s = \frac{1}{{n_f}^2} \|h_4(\mathbf{x}, t_a) - h_4(\mathbf{x}, t_b)\|_2^2. \tag{6}$$

In the case that t and $t+1$ do not belong to the same grid, similar derivation reach to the same conclusion as Eq. (6) except that t_a and t_b are not adjacent but the closest grid points that satisfies $t_a \le t \le t+1 \le t_b$. In practice, imposing the smoothness term only on adjacent grids is enough to improve performance. We applied Eq. (6) to every grid point that is used for feature vector calculation during the training, so the smoothness term can be assigned to a single grid point multiple times in a single iteration.

D. Evaluation Details

LPIPS metric may vary depending on which backbone network is used. We reported LPIPS using VGGNet for D-NeRF dataset and AlexNet for DyNeRF dataset. FLIP is calculated using weighted median for the DyNeRF dataset. We inferred those settings by implementing previous works or using publicly available code, and comparing them with our method.

When evaluating *interp* dataset, the feature vector $\mathbf{v}_t$ is interpolated instead of interpolating the embedding vector $\mathbf{z}_t$. Since temporal interpolation of feature vectors is repeatedly occurred during the training, it is natural to interpolate the feature vectors rather than the embedding vectors to generate features of intermediate frames. This strategy improves overall performance by 0.5dB in PSNR compared to the embedding vector interpolation strategy.

In all experiments, network models are optimized and evaluated per sequence except the *flame_salmon* sequence in DyNeRF dataset which is separated to four sequences of 100 frames to have the same frames with the other sequences in the dataset. We provide detailed per-scene quantitative results for each dataset. From Tab. 1 to Tab. 3, the per-sequence performance of the neural representation models are presented. Tab. 4 represents the per-sequence results of the grid representation on D-NeRF dataset.

Sequence	PSNR	SSIM	LPIPS	AVG
HellWarrior	25.40	0.953	0.0682	0.0349
Mutant	34.70	0.983	0.0226	0.0100
Hook	28.76	0.960	0.0496	0.0237
BouncingBalls	43.32	0.996	0.0203	0.0040
Lego	25.33	0.943	0.0413	0.0307
T-Rex	33.06	0.982	0.0212	0.0112
StandUp	36.27	0.988	0.0159	0.0074
JumpingJacks	35.03	0.985	0.0249	0.0098
Mean	32.73	0.974	0.0330	0.0142

Table 1. Per-sequence quantitative results of the neural representation model on D-NeRF dataset.

vrig			*interp*		
Sequence	PSNR	SSIM	Sequence	PSNR	SSIM
Broom	20.48	0.685	Teapot	26.53	0.933
3D Printer	20.38	0.678	Chicken	27.99	0.940
Chicken	21.89	0.869	Fist	29.74	0.933
Peel Banana	28.87	0.965	Fist	29.74	0.933
			Slice Banana	28.39	0.923
			Lemon	31.31	0.948
Mean	24.35	0.866	Mean	28.67	0.940

Table 2. Per-sequence results of the neural representation model on HyperNeRF dataset.

Sequence	PSNR	LPIPS	FLIP
coffee_martini	27.48	0.1143	0.1456
cook_spinach	33.12	0.0699	0.1262
cut_roasted_beef	33.63	0.0695	0.1221
flame_salmon_1	27.66	0.1127	0.1468
flame_salmon_2	26.91	0.1239	0.1464
flame_salmon_3	27.05	0.1191	0.1558
flame_salmon_4	26.72	0.1410	0.1493
flame_steak	33.11	0.0560	0.1398
sear_steak	33.24	0.0576	0.1396
Mean	29.88	0.0960	0.1413

Table 3. Per-sequence quantitative results of the neural representation model on DyNeRF dataset.

E. Ablation Studies

E.1. Neural Representation

To find the optimal structure of the network, we conducted experiments on various settings of the neural representation models. We changed number of MLPs for feature extraction, number of levels, and application of the smoothness term. All models are tested on $3dprinter$ sequence of *vrig* dataset, and the results are shown in Tab. 5. Two-level architecture with $n_0 = 5$, $n_1 = 20$ showed the best performance. The performance becomes worse in three-level architectures. Thus, using large number of networks does not guarantee performance improvements. In the case that the time slot between MLPs is too small, the network is optimized in only a few frames which prevents from learning meaningful features while imposing additional computational burden.

Sequence	PSNR	SSIM	LPIPS	AVG
HellWarrior	24.33	0.936	0.1088	0.0466
Mutant	32.04	0.977	0.0374	0.0152
Hook	27.63	0.949	0.0859	0.0322
BouncingBalls	34.52	0.973	0.0633	0.0154
Lego	25.16	0.935	0.0618	0.0364
T-Rex	31.21	0.974	0.0445	0.0176
StandUp	33.29	0.983	0.0315	0.0125
JumpingJacks	30.51	0.968	0.0590	0.0211
Mean	29.84	0.962	0.0615	0.0230

Table 4. Per-sequence quantitative results of the grid representation model on D-NeRF dataset.

Method	n_0	n_1	n_2	Smooth	PSNR	MS-SSIM
NeRF + Time	-	-	-	-	21.05	0.847
Ours-NN	2	-	-	X	20.64	0.839
	2	5	-	X	21.06	0.849
	5	20	-	X	21.15	0.850
	5	20	-	O	**21.73**	**0.864**
	2	10	25	O	21.15	0.849
	5	20	50	O	21.12	0.848

Table 5. Ablation studies on network architectures of the neural representation. *3dprinter* sequence from $vrig$ dataset is used for evaluation.

E.2. Grid Representation

We examined the effect of hash table size and number of grid levels. When small number of grid levels are used, as showin in Fig. 3, the model converges faster, but shows slightly worse performance. Optimal performance is achieved when number of levels are set to 12. On the other hand, the performance tends to degrade when large hash table is used as observed in Fig. 4. Not only for slow training speed, large hash table seems also inefficient to learn compact representations of the scene although further study including various real-scene data would be needed.

E.3. Effectiveness of the Static Features

We showed qualitative results that validate the effectiveness of the static features in Fig. 5. While one may think the quantitative improvement for the neural representation seems marginal, as it can be seen in Fig. 5 left, the model trained with static feature recovers fine details (*e.g.* the texture of a stone and dirt). For the grid representation (Fig. 5 right), severe artifacts exist in static regions when only dynamic features are used.

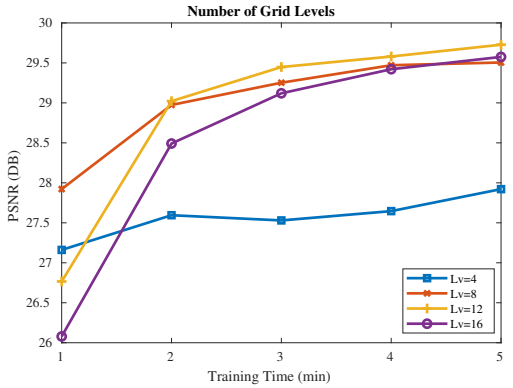


Figure 3. The performance of the grid representation model when different number of levels are used.

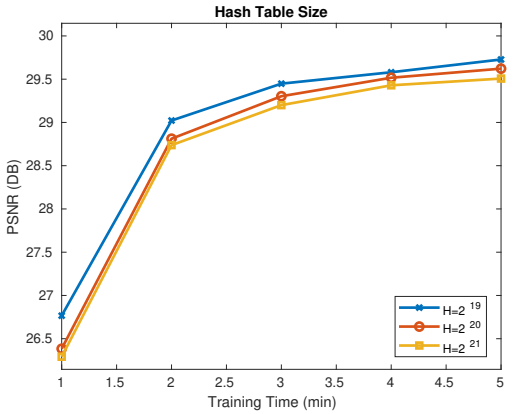


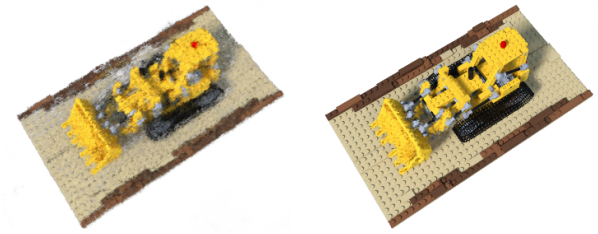
Figure 4. The performance of the grid representation model with various hash table size.

F. Additional Qualitative Results

We conducted an additional experiment on the sequence containing significant topological variations, *split-cookie* and *espresso* from the HyperNeRF dataset. Figure 6 shows qualitative results of the neural representation model on the sequence. We also included the depth estimation results at the corner of each image to ensure that 3D geometry is correctly estimated. Our method does not suffer from topological variations since it does not have any assumption about the shape topology, which verifies the flexibility of our approach.

Lastly, we provide a video clip which summarize qualitative results of our method. We demonstrate the rendering results from novel viewpoints that are not included in training or testing data. First, rendered images of D-NeRF dataset using the neural representation models are shown. Next, rendered images of HyperNeRF dataset is shown and compared to the results from HyperNeRF. Then, we demonstrate the result on DyNeRF dataset. For the grid representation, we first show training progress of the grid representation model on D-NeRF datasets. It can be observed that the grid representation model can learn rough 3D structures within 30 seconds. Lastly, we demonstrate interactive demo which performs real-time rendering using the model trained for 8 minutes.

(a) dynamic only (b) dynamic+static



(c) dynamic only (d) dynamic+static

Figure 5. The effect of the static features on the neural representation ((a),(b)) and the grid representation ((c),(d)).

References

[1] 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

[2] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, July 2022. 1, 2

[3] Keunhong Park, Utkarsh Sinha, Peter Hedman, Jonathan T Barron, Sofien Bouaziz, Dan B Goldman, Ricardo Martin-Brualla, and Steven M Seitz. Hypernerf: A higher-dimensional representation for topologically varying neural radiance fields. *arXiv preprint arXiv:2106.13228*, 2021. 1

[4] 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. 2

[5] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye

Figure 6. Qualitative results of the neural representation model on the sequences containing topological variations.

Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. 1

[6] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 1