Efficient NeRF Optimization - Not All Samples Remain Equally Hard

Supplementary Materials

Juuso Korhonen[1], Goutham Rangu[1], Hamed R. Tavakoli[1], and Juho Kannala[2,3]

[1] Nokia Technologies, Finland
{juuso.korhonen, goutham.rangu, hamed.rezazadegan_tavakoli}@nokia.com
[2] Aalto University, Finland
juho.kannala@aalto.fi
[3] University of Oulu

A Additional baseline method details

We use the open-sourced PyTorch reimplementation of the Instant-NGP at https://github.com/ashawkey/nerf_template and adapt the default training script for the Mip-NeRF-360 scenes by increasing the batch size to $B = 2^{20}$, disabling the iteration based learning rate (since we test against the wall-clock training time) and downscaling the outdoor images with a factor of 8 (instead of 4 for indoor), to bring both the indoor and the outdoor images to a more even standard definition resolution. We follow the guideline of the [1] by using 7/8 images for training and 1/8 images, evenly distributed, for validation.

We note that the PyTorch version does not reimplement every feature of the official implementation [3]. Most notably the ray termination and the full-CUDA-coding are left out. This leads to the PyTorch implementation not reaching the same performance as the official implementation, which can be seen from the NeRF synthetic dataset results reported in Tab. B. We plan on a NeRFStudio implementation for more comprehensive testing.

B Additional results

B.1 Converged modeling results

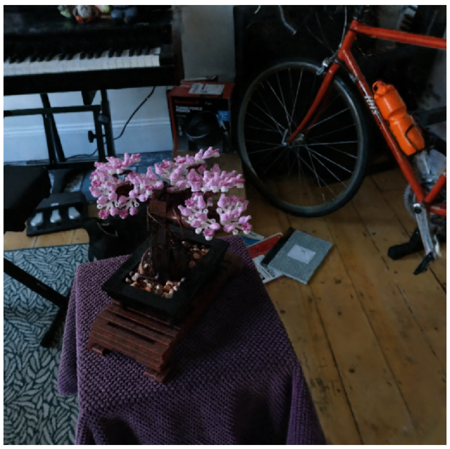
We report the converged training results for the Instant-NGP on the Mip-NeRF-360 dataset scenes in Tab. A, letting the training continue for 20 minutes. In Fig. A, we visualize free-viewpoint renderings using the repository's GUI renderer; We observe no test time artifacts caused by using our hard sample mining method during the training.

B.2 NeRF synthetic dataset results

We report the Instant-NGP results for the NeRF synthetic dataset [2] in Tab. B. We use the train split for training and evaluate the PSNR on the test split. As batch size we use $B = 2^{18}$ as mentioned in [3].

Table A: Validation dataset Peak Signal-to-Noise-Ratio when training Instant-NGP for 20 min on the Mip-NeRF-360 dataset. Results reported in format *hard sample mining* | *baseline*.

	bonsai	kitchen	room	counter	garden	bicycle	stump
20 min	30.35 | 30.03	28.27 | 27.65	29.42 | 29.05	26.09 | 26.00	27.25 | 26.84	22.66 | 22.36	23.28 | 23.46



(a) Bonsai

(b) Garden

Fig. A: Free-viewpoint test time renderings for the Instant-NGP trained with the hard sample mining.

B.3 Nerfacto results

We also did testing of our method with the Nerfacto [4] reimplementation in the repository [5], which uses the proposal sampling method for its ray sampling. We use a lower batch size ($B = 2^{18}$ instead of $B = 2^{20}$) since with the Nerfacto architecture we need to process on average 5.3 proposal samples per main NeRF network sample. Also, to make the results comparable to the ones reported in [4], we did not apply any separate downscaling for the outdoor scenes.

The results seem promising for the generalizability of our method: The dynamic memory footprint is reduced from 300 MB to 160 MB for normal iterations when using our hard sample mining (570 MB to 430 MB for iterations when sampler networks are updated). However, the measured PSNR gains per training time are not as large as with the Instant-NGP. We contribute this to two factors: The proposal sample processing, which is out-of-the-scope of our hard sample mining, (1) forms a larger part of the computation, and (2) prunes away the low-weight samples more effectively when compared to the grid sampling of Instant-NGP.

Table B: Test dataset Peak Signal-to-Noise-Ratio when training Instant-NGP on the NeRF synthetic dataset. Results reported in format *hard sample mining* | *baseline*. * denotes the official paper [3] results for 5 minutes of training.

	mic	ficus	chair	hotdog	materials	drums	ship	lego
1 min	35.18 | 31.77	32.64 | 31.32	34.48 | 32.11	36.43 | 34.29	28.42 | 26.51	25.49 | 24.72	29.12 | 26.86	34.98 | 32.00
5 min	35.76 | 34.12	32.21 | 32.21	34.87 | 33.22	36.66 | 35.15	28.69 | 27.18	25.58 | 25.17	29.53 | 27.55	35.19 | 32.74
5 min*	36.22	33.51	35.00	37.40	29.78	26.02	31.10	36.39

Table C: Validation dataset Peak Signal-to-Noise-Ratio when training Nerfacto on the Mip-NeRF-360 dataset. Results reported in format *hard sample mining* | *baseline*. We also report the results from the official paper [4] in the second section.

	bonsai	kitchen	room	counter	garden	bicycle	stump
2 min	27.88 | 27.57	27.11 | 26.42	29.15 | 28.56	24.96 | 24.60	23.37 | 23.26	22.06 | 21.98	23.92 | 23.69
4 min	29.75 | 29.46	29.44 | 29.01	30.58 | 30.18	25.95 | 25.86	24.44 | 24.30	22.96 | 22.96	24.83 | 24.69
8 min	30.82 | 30.68	30.53 | 30.24	31.46 | 31.38	26.64 | 26.63	25.22 | 25.20	23.56 | 23.67	25.49 | 25.33
12 min	31.31 | 31.18	30.96 | 30.74	31.79 | 31.83	26.94 | 26.97	25.58 | 25.61	23.84 | 23.98	25.76 | 25.63
NeRF	26.81	26.31	28.56	25.67	23.11	21.76	21.73
MipNeRF	27.13	26.47	28.73	25.59	23.16	21.69	23.10
NeRF++	29.15	27.80	28.87	26.38	24.32	22.64	24.34
MipNeRF-360	33.46	32.23	31.63	29.55	26.98	24.37	26.4
Nerfacto(5min) [4]	28.98	28.17	29.36	25.92	24.05	22.36	18.94

References

1. Barron, J.T., Mildenhall, B., Verbin, D., Srinivasan, P.P., Hedman, P.: Mip-nerf 360: Unbounded anti-aliased neural radiance fields (2022) 1
2. Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R., Ng, R.: Nerf: Representing scenes as neural radiance fields for view synthesis (2020) 1
3. Müller, T., Evans, A., Schied, C., Keller, A.: Instant neural graphics primitives with a multiresolution hash encoding. ACM Trans. Graph. **41**(4), 102:1–102:15 (2022). https://doi.org/10.1145/3528223.3530127, https://doi.org/10.1145/3528223.3530127 1, 3
4. Tancik, M., Weber, E., Ng, E., Li, R., Yi, B., Kerr, J., Wang, T., Kristoffersen, A., Austin, J., Salahi, K., Ahuja, A., McAllister, D., Kanazawa, A.: Nerfstudio: A modular framework for neural radiance field development. In: ACM SIGGRAPH 2023 Conference Proceedings. SIGGRAPH '23 (2023) 2, 3
5. Tang, J.: Torch-ngp: a pytorch implementation of instant-ngp (2022), https://github.com/ashawkey/torch-ngp 2