

---

# Supplementary Material

## NerfBaselines: Consistent and Reproducible Evaluation of Novel View Synthesis Methods

---

1 In the Supplementary Material, we extend Section 3 from the main paper by providing the full  
2 Method API in Section A.1. We describe the video released as part of the Supplementary Material  
3 in Section A.2. We extend the reproducibility comparison from Section 4 of the main paper in  
4 Section A.3. We show screenshots of the web platform in Section A.4. We give the instructions  
5 on how to use NerfBaselines and reproduce the results in Section A.5. Finally, we provide detailed  
6 licenses for integrated methods in Section A.6.

### 7 **A.1 Method API**

8 Every method implements the following interface:

- 9 • `constructor(train_dataset?, checkpoint?)`: The constructor takes as its inputs the (optional) training dataset instance (a set of images and camera parameters) or the (optional) checkpoint. At least one of the two has to be provided.
- 10 • `train_iteration`: Performs one training step (using the train dataset), updating the parameters.
- 11 • `save(path)`: Saves the current checkpoint.
- 12 • `render(cameras, embeddings?)`: Renders the 3D scene using the list of camera parameters with an optional list of camera embeddings if the method supports appearance conditioning.
- 13 • `get_info` and `get_method_info`: Returns information about the trained model and the base method, respectively.
- 14 • (optional) `optimize_embeddings(dataset)`: Optimizes the appearance embeddings on the dataset (if the method supports it).
- 15
- 16
- 17
- 18
- 19

### 20 **A.2 Video**

21 In the attached video, we compare Gaussian Splatting [4], Mip-Splatting [9], Zip-NeRF [2], Instant-  
22 NGP [7], and NerfStudio [8]. We generate trajectories such that we start from regions close to the  
23 training trajectory, then we move further from the scene center and close to geometry to visualize  
24 how different methods handle these viewpoint changes. We show the results on two scenes from  
25 the Mip-NeRF 360 dataset [1] (stump and kitchen), and two scenes from the Tanks and Temples  
26 dataset [?] (temple and lighthouse). Notice, how 3DGS behaves well close to the training trajectory  
27 but has blank spots outside (where the geometry is missing). On the other hand, NeRFs (especially  
28 NerfStudio [8]) is better able to extrapolate to less visible regions further from the scene center.

### 29 A.3 Reproducing paper results

30 To extend the data reported in the main paper, we further give the detailed PSNR and SSIM scores for  
 31 Blender [6] and Mip-NeRF 360 [1] datasets. From the results, we can see that SSIM behaves similarly  
 32 to PSNR with mostly smaller differences (except for the Tetra-NeRF run on the Blender/ship scene).  
 33 However, LPIPS (VGG) [10] numbers are very different from the papers. There are two reasons: **1)**  
 34 Multi-NeRF [1] and 3DGS [4] codebases both have a bug in LPIPS computation, where the images  
 35 which should be normalized to  $[-1, 1]$  range is kept in the  $[0, 1]$  range. This makes the errors smaller.  
 36 If we also change our evaluation not to normalize to  $[-1, 1]$ , we reproduce their results. Since these  
 37 codebases are the basis for most other methods, the bug is shared for other methods as well. **2)** A  
 38 smaller reason present in the case of NerfStudio is that we evaluate all metrics on images in the  
 39 `uint8` range (to be reproducible from the stored image). While this change does not have much  
 40 influence on PSNR and SSIM, LPIPS is more sensitive to these changes.

PSNR $\uparrow$	lego materials	drums mic	ficus ship	hotdog chair
Instant NGP [7]	35.64/36.39/-2.05% 28.95/29.78/-2.77%	24.57/26.02/-5.55%	30.29/33.51/-9.62%	37.01/37.40/-1.03%
TensoRF [3]	36.49/36.46/+0.09% 30.08/30.12/-0.14%	26.01/26.01/-0.00% 34.85/34.61/+0.68%	34.06/33.99/+0.19% 30.69/30.77/-0.26%	37.49/37.41/+0.20% 35.72/35.76/-0.11%
Tetra-NeRF [5]	33.93/34.75/-2.36% 28.75/29.30/-1.88%	24.99/25.01/-0.09% 34.54/35.49/-2.68%	32.37/33.31/-2.82% 31.06/31.13/-0.22%	35.80/36.16/-0.98% 34.17/35.05/-2.52%
Zip-NeRF [2]	35.81/34.84/+2.79% 30.98/31.66/-2.15%	25.90/25.84/+0.23% 35.90/35.15/+2.14%	34.73/33.90/+2.44% 32.30/31.38/+2.94%	37.98/37.14/+2.27% 35.75/34.84/+2.62%
Gaussian Splatting [4]	35.70/35.78/-0.24% 30.01/30.00/+0.02%	26.15/26.15/-0.01% 35.49/35.36/+0.36%	34.79/34.87/-0.22% 30.85/30.80/+0.15%	37.64/37.72/-0.21% 35.84/35.83/+0.04%

  

SSIM $\uparrow$	lego materials	drums mic	ficus ship	hotdog chair
Instant NGP [7]	0.981/-/ 0.944/-/	0.930/-/ 0.989/-/	0.972/-/ 0.892/-/	0.982/-/ 0.984/-/
TensoRF [3]	0.983/0.983/+0.01% 0.952/0.952/+0.04%	0.936/0.937/-0.05% 0.988/0.988/+0.03%	0.982/0.982/+0.04% 0.894/0.895/-0.14%	0.982/0.982/+0.04% 0.984/0.985/-0.06%
Tetra-NeRF [5]	0.972/0.987/-1.57% 0.941/0.968/-2.77%	0.927/0.947/-2.08% 0.987/0.993/-0.60%	0.977/0.989/-1.22% 0.896/0.994/-9.83%	0.978/0.989/-1.09% 0.977/0.990/-1.35%
Zip-NeRF [2]	0.983/0.980/+0.29% 0.967/0.969/-0.19%	0.948/0.944/+0.38% 0.992/0.991/+0.11%	0.987/0.985/+0.19% 0.937/0.929/+0.83%	0.987/0.984/+0.25% 0.987/0.983/+0.36%
Gaussian Splatting [4]	0.982/-/ 0.959/-/	0.953/-/ 0.991/-/	0.987/-/ 0.904/-/	0.985/-/ 0.987/-/

  

LPIPS <sub>VGG</sub> $\downarrow$	lego materials	drums mic	ficus ship	hotdog chair
Instant NGP [7]	0.020/-/ 0.069/-/	0.109/-/ 0.016/-/	0.031/-/ 0.136/-/	0.037/-/ 0.023/-/
TensoRF [3]	0.022/0.018/+23.33% 0.059/0.058/+1.31%	0.076/0.073/+4.16% 0.021/0.015/+39.93%	0.029/0.022/+30.50% 0.141/0.138/+2.44%	0.033/0.032/+2.16% 0.027/0.022/+22.86%
Tetra-NeRF [5]	0.036/-/ 0.076/-/	0.087/-/ 0.022/-/	0.032/-/ 0.129/-/	0.040/-/ 0.029/-/
Zip-NeRF [2]	0.019/0.019/-0.63% 0.040/0.032/+26.50%	0.054/0.050/+8.08% 0.008/0.007/+16.14%	0.014/0.015/-6.00% 0.114/0.091/+24.84%	0.023/0.020/+12.70% 0.017/0.017/+0.24%
Gaussian Splatting [4]	0.019/-/ 0.043/-/	0.044/-/ 0.008/-/	0.013/-/ 0.130/-/	0.026/-/ 0.015/-/

Table 1: **Blender results comparing PSNR, SSIM, and LPIPS (VGG) obtained via NerfBaselines with those reported in the original papers.** We show the current PSNR/original PSNR/relative difference in %. The darker the color, the larger the difference. The differences for Instant NGP [7] are larger because the paper used a black background (instead of the default white). For LPIPS (VGG), the differences are larger because the compared codebases do not normalize input images to the correct range.

	garden kitchen	bicycle bonsai	flowers counter	treehill room	stump
PSNR $\uparrow$					
Mip-NeRF 360 [1]	27.00/26.98/+0.07% 32.08/32.23/-0.48%	24.34/24.37/-0.12% 33.48/33.46/+0.05%	21.74/21.73/+0.04% 29.51/29.55/-0.15%	22.89/22.87/+0.10% 31.58/31.63/-0.14%	26.41/26.40/+0.04%
NerfStudio [8]	25.96/26.47/-1.91% 29.96/30.29/-1.10%	23.61/24.08/-1.94% 30.52/32.16/-5.11%	21.12/-/- 26.80/27.20/-1.48%	22.85/-/- 30.56/30.89/-1.06%	25.75/24.78/+3.91%
Zip-NeRF [2]	28.18/28.20/-0.05% 32.39/32.50/-0.34%	25.87/25.80/+0.26% 34.67/34.46/+0.61%	22.34/22.40/-0.25% 28.90/29.38/-1.63%	24.01/23.89/+0.51% 32.95/32.65/+0.93%	27.32/27.55/-0.82%
Gaussian Splatting [4]	27.37/27.41/-0.16% 31.36/30.32/+3.43%	25.20/25.25/-0.17% 32.10/31.98/+0.36%	21.60/21.52/+0.36% 28.97/28.70/+0.93%	22.46/22.49/-0.15% 31.43/30.63/+2.62%	26.48/26.55/-0.28%
Mip-Splatting [9]	27.48/27.76/-1.02% 31.12/31.55/-1.36%	25.30/25.72/-1.64% 32.18/32.31/-0.39%	21.64/21.93/-1.31% 29.04/29.16/-0.39%	22.64/22.98/-1.47% 31.55/31.74/-0.60%	26.52/26.94/-1.54%
SSIM $\uparrow$					
Mip-NeRF 360 [1]	0.813/0.813/+0.01% 0.919/0.920/-0.11%	0.688/0.685/+0.41% 0.940/0.941/-0.08%	0.583/0.583/-0.01% 0.893/0.894/-0.09%	0.632/0.632/-0.08% 0.912/0.913/-0.07%	0.747/0.744/+0.37%
NerfStudio [8]	0.756/0.774/-2.28% 0.881/0.890/-0.96%	0.567/0.599/-5.34% 0.909/0.933/-2.61%	0.512/-/- 0.829/0.843/-1.66%	0.546/-/- 0.880/0.896/-1.82%	0.694/0.662/+4.90%
Zip-NeRF [2]	0.863/0.860/+0.37% 0.929/0.928/+0.14%	0.775/0.769/+0.78% 0.951/0.949/+0.24%	0.637/0.642/-0.80% 0.904/0.902/+0.21%	0.675/0.681/-0.93% 0.927/0.925/+0.24%	0.789/0.800/-1.35%
Gaussian Splatting [4]	0.866/0.868/-0.17% 0.927/0.922/+0.52%	0.765/0.771/-0.79% 0.941/0.938/+0.30%	0.604/0.605/-0.23% 0.907/0.905/+0.25%	0.629/0.638/-1.34% 0.917/0.914/+0.31%	0.768/0.775/-0.87%
Mip-Splatting [9]	0.869/0.875/-0.71% 0.927/0.933/-0.68%	0.766/0.780/-1.74% 0.941/0.948/-0.71%	0.604/0.623/-3.01% 0.907/0.916/-0.94%	0.635/0.655/-3.11% 0.917/0.928/-1.14%	0.770/0.786/-2.04%
LPIPS <sub>VGG</sub> $\downarrow$					
Mip-NeRF 360 [1]	0.190/0.170/+11.59% 0.155/0.127/+22.06%	0.328/0.301/+9.10% 0.211/0.176/+19.82%	0.370/0.344/+7.63% 0.253/0.204/+24.18%	0.379/0.339/+11.93% 0.267/0.211/+26.38%	0.300/0.261/+14.76%
NerfStudio [8]	0.249/0.235/+5.95% 0.200/0.190/+5.03%	0.454/0.422/+7.46% 0.249/0.197/+26.27%	0.434/-/- 0.338/0.314/+7.77%	0.492/-/- 0.314/0.296/+5.98%	0.350/0.380/-8.01%
Zip-NeRF [2]	0.127/0.118/+7.85% 0.133/0.116/+15.07%	0.227/0.208/+8.91% 0.195/0.173/+12.77%	0.310/0.273/+13.41% 0.224/0.185/+20.83%	0.281/0.242/+16.11% 0.237/0.196/+20.88%	0.234/0.193/+21.44%
Gaussian Splatting [4]	0.123/0.103/+19.23% 0.155/0.129/+19.93%	0.239/0.205/+16.66% 0.253/0.205/+23.48%	0.366/0.336/+8.88% 0.257/0.204/+26.01%	0.379/0.317/+19.41% 0.286/0.220/+30.19%	0.251/0.210/+19.66%
Mip-Splatting [9]	0.124/0.103/+20.72% 0.155/0.113/+37.07%	0.241/0.206/+17.12% 0.253/0.173/+46.24%	0.371/0.331/+12.19% 0.258/0.179/+43.87%	0.379/0.320/+18.37% 0.286/0.192/+48.85%	0.253/0.209/+21.18%

Table 2: **Mip-NeRF 360** results comparing PSNR, SSIM, and LPIPS (VGG) obtained via NerfBaselines with those reported in the original papers. We show the current PSNR/original PSNR/relative difference in %. The darker the color, the larger the difference. For PSNR and SSIM, in most cases, the difference is  $< 1\%$ . For LPIPS (VGG), the differences are larger because the compared codebases (except for NerfStudio) do not normalize input images to the correct range.

### Mip-NeRF 360

Mip-NeRF 360 is a collection of four indoor and five outdoor object-centric scenes. The camera trajectory is an orbit around the object with fixed elevation and radius. The test set takes each n-th frame of the trajectory as test views.

Web: <https://jonbarron.info/mipnerf360/>

Paper: [Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields](#)

Authors: Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, Peter Hedman

Method	PSNR	SSIM	LPIPS	Time	GPU Mem.
Zip-NeRF	28.516	0.828	0.138	5h 30m 49s	26.19 GB
Mip-NeRF 360	27.670	0.792	0.196	7h 29m 42s	126.99 GB
Mip-Splatting	27.498	0.815	0.183	25m 1s	10.96 GB
Gaussian Splatting	27.439	0.814	0.180	22m 45s	11.12 GB
garden	27.396	0.866	0.076	29m 5s	15.64 GB
360	26.500	0.766	0.176	57m 44s	13.74 GB

a) Dataset results view

### Zip-NeRF

Zip-NeRF is a radiance field method which addresses the aliasing problem in the case of hash-grid based methods (NGP-based). Instead of sampling along the ray it samples along a spiral path - approximating integration along the frustum.

Web: <https://jonbarron.info/zipnerf/>

Paper: [Zip-NeRF: Anti-Aliased Grid-Based Neural Radiance Fields](#)

Authors: Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, Peter Hedman

### Mip-NeRF 360

Mip-NeRF 360 is a collection of four indoor and five outdoor object-centric scenes. The camera trajectory is an orbit around the object with fixed elevation and radius. The test set takes each n-th frame of the trajectory as test views.

Scene	PSNR	SSIM	LPIPS	Time	GPU Mem.
average	28.516	0.828	0.138	5h 30m 49s	26.19 GB

Blender

a) Method results view

Figure 1: **Web platform.** Shows the ranking of the current set of integrated methods. It enables downloading of the checkpoints and predictions, and for some methods, it provides an online viewer.

## 41 A.4 Web platform

42 To keep track of the current state of the art (SoTA), we release a web platform (<https://jkuhanek.com/nerfbaselines>). The web platform shows results on all individual scenes for all methods, enables comparing methods, and allows users to download checkpoints and predictions for the datasets. Example screenshots can be seen in Figure 1.

## 46 A.5 Use instructions

47 **Installation.** Before installing NerfBaselines, Python 3.7+ must be installed on the host system. We recommend using either conda or venv to separate NerfBaselines from system packages. After Python is ready, install the nerfbaselines pip package on your host system by running: `pip install nerfbaselines`. Now, `nerfbaselines cli` can be used to interact with NerfBaselines. However, at least one supported backend must be installed before any method can be used. At the moment there are the following backends implemented:

- 53 • **docker:** Offers good isolation, requires docker (with NVIDIA container toolkit) to be installed and the user to have access to it (being in the docker user group). In order to install it, please follow the instructions at <https://github.com/NVIDIA/nvidia-container-toolkit>
- 54 • **apptainer:** Similar level of isolation as docker, but does not require the user to have privileged access. To install the backend, please follow instructions at <https://apptainer.org/docs/admin/main/installation.html>.
- 55 • **conda** (default): Does not require docker/apptainer to be installed, but does not offer the same level of isolation and some methods require additional dependencies to be installed. Also, some methods are not implemented for this backend because they rely on dependencies not found on conda. To install conda, we recommend following instructions at <https://github.com/conda-forge/miniforge> to install the miniforge distribution of conda.
- 56 • **python:** Will run everything directly in the current environment. Everything needs to be installed in the environment for this backend to work.

67 Additionally, all backends require NVIDIA GPU drivers to be installed to access the GPUs. For NerfBaselines commands, the backend can be set either via the `--backend <backend>` argument or using the `NERFBASELINES_BACKEND` environment variable.

70 **Training.** To start the training, use the following command: `nerfbaselines train --method <method> --data external://<dataset>/<scene>`, where `<method>` can be e.g., `nerfacto`, `zipnerf`, `instant-ngp`, ... (for the full list, run `nerfbaselines train --help`). The `<dataset>` can be one of the following: `mipnerf360`, `blender`, `tanksandtemples`. Similarly, `<scene>` is the scene name in lowercase. The training script will automatically download the dataset and start the training. The training will also run the evaluation and output the metrics computed on the test set.

77 **Other commands.** The resulting checkpoint can be used in the viewer (`nerfbaselines viewer --checkpoint <checkpoint> --data external://<dataset>/<scene>`), to rerun the rendering (`nerfbaselines render --checkpoint <checkpoint> --data external://<dataset>/<scene>`), or to render a camera trajectory (`nerfbaselines render-trajectory --checkpoint <checkpoint> --trajectory <trajectory> --output <output>.mp4`). The full list of available command can be seen by running `nerfbaselines --help`.

## 84 **A.6 License**

85 The NerfBaselines project is licensed under the MIT license. Each implemented method is licensed  
86 under the license provided by the authors of the method. For the currently implemented methods, the  
87 following licenses apply:

- 88 • NerfStudio: Apache 2.0
- 89 • Instant-NGP: custom, research purposes only
- 90 • Gaussian-Splatting: custom, research purposes only
- 91 • Mip-Splatting: custom, research purposes only
- 92 • Gaussian Opacity Fields: custom, research purposes only
- 93 • Tetra-NeRF: MIT, Apache 2.0
- 94 • Mip-NeRF 360: Apache 2.0
- 95 • Zip-NeRF: Apache 2.0
- 96 • CamP: Apache 2.0

## 97 **References**

- 98 [1] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-nerf 360:  
99 Unbounded anti-aliased neural radiance fields. In *CVPR*, 2022.
- 100 [2] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Zip-nerf:  
101 Anti-aliased grid-based neural radiance fields. In *ICCV*, pages 19697–19705, 2023.
- 102 [3] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. TensorRF: Tensorial radiance fields. In  
103 *ECCV*, 2022.
- 104 [4] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for  
105 real-time radiance field rendering. *ACM TOG*, 2023.
- 106 [5] Jonas Kulhanek and Torsten Sattler. Tetra-nerf: Representing neural radiance fields using tetrahedra. In  
107 *ICCV*, pages 18458–18469, 2023.
- 108 [6] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng.  
109 NeRF: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020.
- 110 [7] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives  
111 with a multiresolution hash encoding. *ACM TOG*, 2022.
- 112 [8] Matthew Tancik, Ethan Weber, Evonne Ng, Ruilong Li, Brent Yi, Terrance Wang, Alexander Kristoffersen,  
113 Jake Austin, Kamyar Salahi, Abhik Ahuja, et al. Nerfstudio: A modular framework for neural radiance  
114 field development. In *ACM TOG*, 2023.
- 115 [9] Zehao Yu, Anpei Chen, Binbin Huang, Torsten Sattler, and Andreas Geiger. Mip-splatting: Alias-free 3d  
116 gaussian splatting. In *CVPR*, 2024.
- 117 [10] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable  
118 effectiveness of deep features as a perceptual metric. In *CVPR*, 2018.