FastNeRF: High-Fidelity Neural Rendering at 200FPS - Supplementary Material

Stephan J. Garbin* Marek Kowalski* Matthew Johnson Jamie Shotton
Julien Valentin

1. Overview

We use the supplementary material to provide more detailed results and algorithms. We urge the reader to see our video, in which we show results for all 16 scenes we tested on along with providing an intuition for our method. On a practical note, we also provide guidance on getting smooth results for anyone adapting our method to their own work.

Please note that parameters chosen throughout this work for the rendering algorithm favour the highest possible quality. It is possible to significantly increase rendering speeds by sacrificing a small amount of visual quality for real-world applications.

1.1. 'Motion' Smoothness

We notice that our method can exhibit more flickering artefacts on the NeRF 360 Synthetic dataset than the baseline *when trained using the same settings as NeRF*, which we find only noticeable in motion. These direction-dependent artefacts are due to overfitting of the view-dependent MLP and can be successfully mitigated in one of two ways (which can also be combined). Sticking with identical parameters as NeRF, we can simply choose a resolution between 16^3 and 64^3 for the direction-dependent cache, which uses interpolation at lookup by default. Alternatively, we can adjust the Fourier feature encoding of the view directions at training time. In [4], the Fourier encoding maps from $\mathbb{R}$ to $\mathbb{R}^{2L}$, where L is 10 for position, and 4 for direction. For FastNeRF, setting $L = 1$ actually works best for the NeRF 360 Synthetic dataset. Please note that we do not use these approaches when computing metrics in the text to ensure the same settings are used for all datasets, but that using them would slightly improve the results on synthetic data.

Effect of modified Fourier feature encoding on metrics: Using an 8 layer 256 hidden unit MLP for position, and a 4 layer 128 MLP for direction, we can obtain an average PSNR of $29.748dB$ for $L = 1$, $29.646dB$ for $L = 2$, $29.449dB$ for $L = 3$ on synthetic data, with smoothness of results being reflected in these numbers.

Effect of small direction cache on metrics: Using a smaller direction cache has a negligible impact on metrics, occasionally actually improving them. This shows that the direction-dependent effects required by FastNeRF are low in frequency. For the scenes where we found we needed to use a smaller cache for moving images, the PSNR differences caused by using a smaller directional cache are: *Materials* (32^3) (28.885dB $\rightarrow$28.874dB); *Drums* (16^3) (23.745dB $\rightarrow$23.836dB); *Lego* (32^3) (32.275dB $\rightarrow$32.155dB); *Mic* (32^3) (31.765dB $\rightarrow$31.667dB); *Hotdog* (32^3) (34.722dB $\rightarrow$34.644dB); *Ficus* (32^3) (27.792dB $\rightarrow$28.193dB).

1.2. Training & Detailed Results

For training, we use the same frequency encoding, noise perturbation and learning rate decay (starting at $5e-4$) as [4]. For the NeRF 360 Synthetic dataset, we use 64 and 128 samples for the coarse and fine networks, respectively. This becomes 64 and 64 for the LLFF scenes. Note that the coarse networks are discarded and not used in the caching stage - the cached density (or rather, the mesh derived from it) serves as the importance distribution during rendering. We sample 1000 random rays per gradient descent step, and use Adam [2] as our optimizer of choice, with $\beta_1 = 0.9$, $\beta_2 = 0.999$.

In the paper, we compute test metrics on a random subset of 20 images per scene from the test sets of the NeRF 360 Synthetic dataset, and all available test images for the LLFF data. We use the evaluation set only to select the best iteration, which is 300K for all scenes, indicating identical convergence trends as NeRF [4]. For completeness, we show results for all scenes using the *full* number of test images in Table 1. We note that the results and trends are in agreement with the sub-sampled test set. Please see Figure 4 and beyond for a qualitative comparison in addition to the video. We include further ablations on the number of components in Table 2 and Table 3.

1.3. Meshing & Rendering

One of the key advantages of working with sparse octrees to store the caches required by our method is that they can serve as a basis for accelerating the rendering process. Using raytracing, we can quickly terminate rays that miss

*Denotes equal contribution

Table 1. As in the main text, we compare NeRF to our method when not cached to a grid, and when cached at a high resolution in terms of PSNR, SSIM and LPIPS, and also provide the average speed in ms of our method when cached. The first 8 scenes comprise the dataset of [4] at 800^2 pixels, the last 8 scenes the LLFF dataset [3] at 504×378 pixels. We use 8 components and a 1024^3 cache for the NeRF Synthetic 360 scenes (except for the ship scene where we use 768^3), and 6 components at 768^3 for LLFF.

Scene	**NeRF**			**Ours - No Cache**			**Ours - Cache**			**Speed**
	PSNR↑	*SSIM*↑	*LPIPS*↓	*PSNR*↑	*SSIM*↑	*LPIPS*↓	*PSNR*↑	*SSIM*↑	*LPIPS*↓	
Drums	24.58dB	0.92	0.08	23.868dB	0.91	0.084	23.745dB	0.913	0.083	4.5ms
Ship	27.21dB	0.83	0.17	27.241dB	0.839	0.155	27.685dB	0.805	0.192	4.9ms
Mic	30.85dB	0.97	0.03	30.274dB	0.973	0.023	31.765dB	0.977	0.022	2.6ms
Ficus	28.86dB	0.96	0.03	27.037dB	0.948	0.034	27.792dB	0.954	0.031	3.7ms
Chair	31.16dB	0.96	0.04	30.967dB	0.96	0.032	32.322dB	0.966	0.032	2.6ms
Lego	30.41dB	0.95	0.03	31.076dB	0.955	0.023	32.275dB	0.964	0.022	5.5ms
Hotdog	34.7dB	0.97	0.03	34.21dB	0.97	0.029	34.722dB	0.973	0.031	4.9ms
Materials	28.93dB	0.94	0.03	28.567dB	0.943	0.034	28.885dB	0.947	0.034	3.9ms
Fern*	26.84dB	0.86	0.09	26.325dB	0.853	0.105	25.006dB	0.822	0.131	1.7ms
Flower*	28.32dB	0.89	0.06	28.916dB	0.912	0.043	27.012dB	0.878	0.059	1.3ms
Fortress*	32.7dB	0.93	0.03	32.946dB	0.937	0.021	31.256dB	0.913	0.043	0.8ms
Horns*	28.78dB	0.9	0.07	29.748dB	0.925	0.044	26.847dB	0.889	0.07	1.2ms
Leaves*	22.43dB	0.82	0.1	22.53dB	0.833	0.095	21.3dB	0.787	0.122	1.5ms
Orchids*	21.36dB	0.75	0.12	21.204dB	0.747	0.126	20.356dB	0.712	0.137	1.6ms
Room*	33.59dB	0.96	0.04	33.702dB	0.961	0.034	30.301dB	0.942	0.057	1.6ms
TRex*	27.74dB	0.92	0.05	28.292dB	0.933	0.04	26.204dB	0.903	0.063	1.4ms

Table 2. Influence of the number of components and grid resolution on PSNR and memory required for caching the *Horns* ('Triceratops') scene. Note how more factors also increase grid sparsity in this case. 6 components are a reasonable amount for LLFF data.

Factors	**No Cache**		256^3		384^3		512^3		768^3	
	PSNR↑	*Memory*	*PSNR*↑	*Memory*	*PSNR*↑	*Memory*	*PSNR*↑	*Memory*	*PSNR*↑	*Memory*
4	29.56dB	-	22.94dB	0.39GB	24.64dB	0.8GB	25.62dB	1.56GB	26.52dB	4.18GB
6	29.75dB	-	23.08dB	0.56GB	24.86dB	1.14GB	25.89dB	2.22GB	26.85dB	6.11GB
8	29.82dB	-	23.02dB	0.72GB	24.79dB	1.45GB	25.81dB	2.81GB	26.76dB	7.83GB

the neurally rendered object(s) all together. For the rays that hit occupied voxels, and so require integrating the volume, we can use raytracing to skip empty space efficiently, saving a significant amount of queries. This matters because caching makes our method memory instead of compute bound. Computing the inner product of the components and weights as well as tracking transmittance for each ray is fast on modern GPUs. Grid look-ups, which require reading from the GPUs RAM on the other hand, are expensive.

While we could use a hierarchical digital differential analyzer such as [5] to determine ray grid intersections, we opt to trace rays against a collision mesh derived from the volume instead, for three reasons. First, this allows us to take advantage of hardware acceleration for the BVH and ray-triangle intersections on modern hardware. Second, we can down/resample or deform the meshes easily using existing tools. Finally, meshes could be used to 'fake' shadows, bouncelight and other effects which can be composited over the neurally rendered content.

We derive the meshes by converting the density cache to a sign distance function (thresholding at 0.0), optionally downsampling it before meshing with marching cubes, and optionally simplifying the resulting geometry using standard techniques. We show the resulting geometry for the *Lego* scene in Figure 1. Note that more aggressive thresholding of the volume or remeshing is easily possible and can lead to significant performance benefits in practise as more rays can be culled early and mesh complexity reduced.

For rendering, we follow the same method as NeRF, us-

Table 3. Influence of the number of components and grid resolution on PSNR and memory required for caching the *Fortress* ('Minas Tirith') scene. Note how more factors also increase grid sparsity in this case. 6 components are a reasonable amount for LLFF data.

Factors	**No Cache**		256^3		384^3		512^3		768^3	
	PSNR↑	*Memory*	*PSNR*↑	*Memory*	*PSNR*↑	*Memory*	*PSNR*↑	*Memory*	*PSNR*↑	*Memory*
4	32.81dB	-	26.54dB	0.42GB	28.61dB	0.93GB	29.83dB	1.88GB	30.99dB	5.43GB
6	32.95dB	-	26.63dB	0.6GB	28.75dB	1.31GB	30.01dB	2.66GB	31.26dB	7.83GB
8	33.03dB	-	26.66dB	0.79GB	28.81dB	1.72GB	30.1dB	3.47GB	31.36dB	10.29GB

ing the Beer-Lambert Law [1] to model radiance extinction. Once the contribution of new samples for a ray reaches 0.001, we terminate a ray's rendering kernel. Note that this can be relaxed for greater performance. Another way to accelerate the rendering process is to visit fewer voxels based on radiance or transmittance. For all paper experiments however, we never skip occupied space, or optimise any parameters per scene.

1.4. Deep Radiance Map Analysis

As shown in Figure 2, FastNeRF can model complex appearance with specular reflections, although we observe small errors when six or less components are used for the Realistic 360 Synthetic dataset. Using more than six components gives the model more representative power, avoiding such errors. For example, point P_1 shows an incorrect specular response (despite being in shadow) when too few components are used. We can also observe greater detail as the number of components increases above eight for P_0 and P_2, although the differences are more subtle. Please note that the complexity of the direction-dependent MLP $\mathcal{F}_{dir}$ is the same throughout these experiments and applying regularisation to it is an orthogonal discussion (see Section 1.1).

Figure 3 shows the weighting of the deep radiance map components over the hemisphere in section (a) for the same scene as before. We note that this function depends only on direction and does not vary with position, unlike the deep radiance map components themselves (please also see the supplementary video at 2:34 - 2:40). With four components, the hemisphere appears divided into three base parts and one that modulates the strong specular response that is observed when the viewing direction is incident with the dominant light in the scene. Adding more components leads to greater subtlety of the learned representation, akin to adding more lobes to a traditional material model in computer graphics [10, 8].

Section (b) of Figure 3 shows that all components contribute to the final colour of an example point, P_2. More specifically, we show the mean absolute contribution of each component to the final pixel colour over the hemisphere from which the training, validation and test views are sampled. We note that these results are not cherry-picked, but illustrative of the general behaviour we are able to observe in FastNeRF reconstructions.

1.5. Comparing FastNeRF with Pruning / Quantization

Alternative to the factorisation proposed in this work, one could also consider pruning MLP weights or direct quantization of MLP weights to gain performance relative to the baseline. In [9], pruning is reported to translate into a speedup of around 11% but is highly dependent on the effectiveness of sparse indexing. For 16 bit quantization, Nvidia reports a maximal 8× increase in arithmetic throughput [6]. In contrast, our method gives speedups of at least 2600×. Hence we do not believe that optimising the neural network for performance would lead to comparable gains. Moreover, we believe it is possible to optimise our method further by removing branching behaviour in CUDA kernels, optimising for the amount of registers, or exploiting cache coherence in specific ways.

1.6. Cache size calculation

In this section we show the formulas we used to estimate cache sizes $M_{NeRF}, M_{FastNeRF}$ for NeRF and FastNeRF respectively.

$$M_{NeRF} = \alpha(s_\sigma + s_{rgb})k^3 l^2, \tag{1}$$

where s_σ, s_{rgb} are the sizes of the stored transparency and RGB values in bits and $\alpha \in \langle 0, 1\rangle$ is the inverse volume sparsity, where $\alpha = 1$ would indicate a dense volume. Just as in the main paper k, l correspond to the number of bins per dimension in the position and direction dependent caches respectively.

$$M_{FastNeRF} = \alpha\left((D \cdot s_{uvw} + s_\sigma)k^3\right) + D \cdot s_\beta l^2, \tag{2}$$

where s_{uvw}, s_β are the sizes of the stored (u_i, v_i, w_i) values and the weights β_i.

For $k = l = 1024, D = 8$, (r, g, b) stored as individual bytes and all other values stored as half-precision floats the cache size would be

$$M_{NeRF} = \alpha \cdot 5,629,499,534,213,120 \approx \alpha \cdot 5.6PB, \tag{3}$$

for NeRF and

$$M_{factor} = \alpha \cdot 53,687,091,200 + 16,777,216 \approx \alpha \cdot 54GB, \tag{4}$$

for FastNeRF.

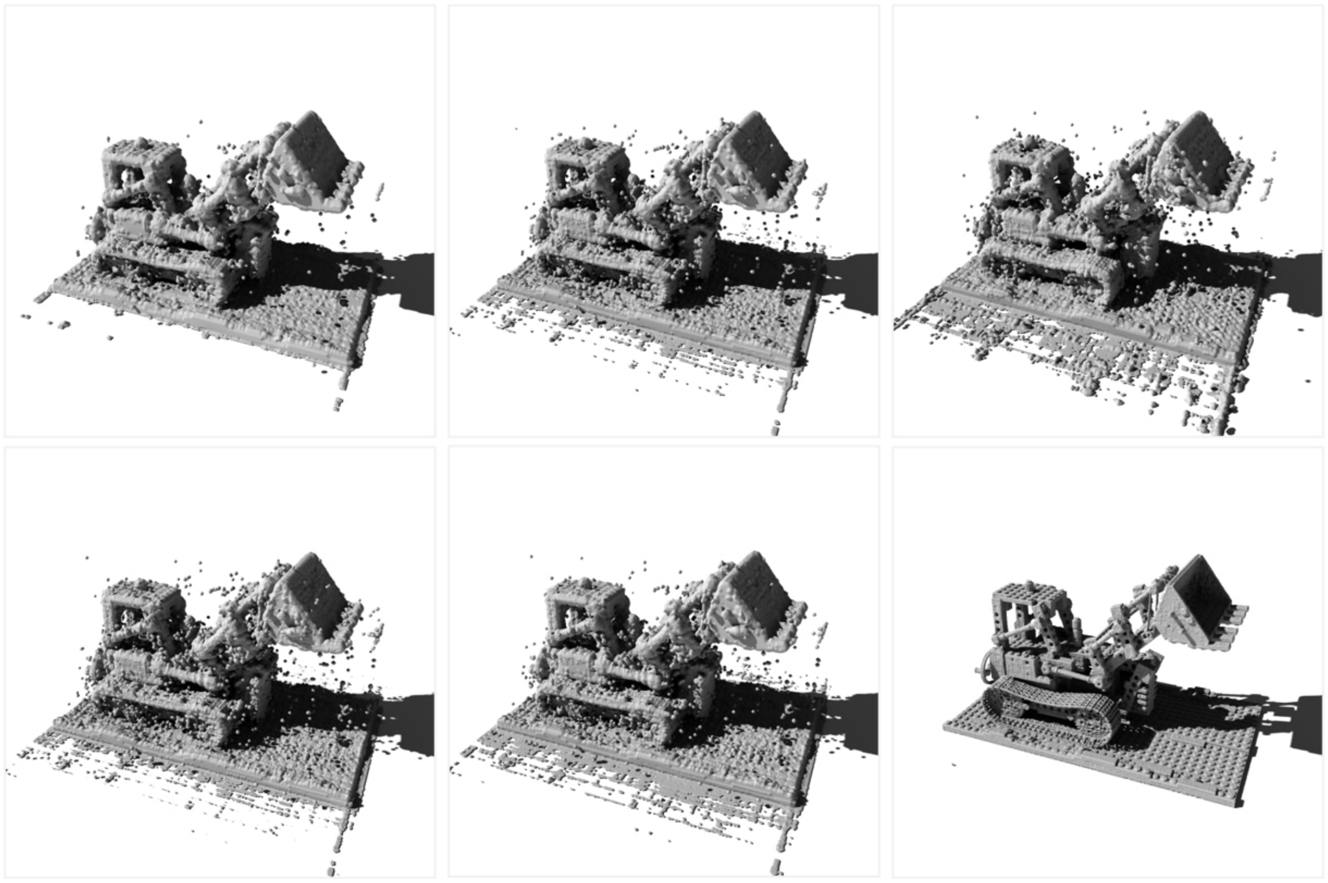

Figure 1. Collision Meshes for the *Lego* scene used for our raytraced rendering algorithm. From left to right, top to bottom, these are derived from the following grid resolutions 256^3, 384^3, 512^3, 768^3, 1024^3. Note that we threshold the density at 0, but being less conservative can give meshes good enough to compute shadows or bounce light. This is seen in the bottom-right image, where the identical volume is thresholded at 10.0 instead (best viewed zoomed in).

1.7. Details on Applications

In our proof-of-concept telepresence scenario we use a deformation field network [7] $\mathcal{F}_{deform}$ that modifies the input sample positions. This network is similar to the position-dependent network used in FastNeRF, but smaller - a 6-layer MLP with 64 units in each layer. The input is a sample position and an expression vector. The sample position is processed with positional encoding identically to how the FastNeRF input position is processed. The output is an offset that is applied to the input sample position. When training with $\mathcal{F}_{deform}$ we add an L2 regularizer that constrains $\mathcal{F}_{deform}$ to be an identity transform if a neutral expression is passed as input. We find that this solution stabilizes training and improves results.

In addition to $\mathcal{F}_{deform}$, we also use a non-trainable deformation field $\mathcal{F}_{bone}$ that moves the samples in the head region according to the movement of the 3D model bones that represent the shoulders and the neck. This additional deformation accounts for large movements of the head and the shoulders, which reduces the load on $\mathcal{F}_{deform}$ and improves quality. The two deformation fields are composed with the position-dependent network $\mathcal{F}_{pos}$ as follows:

$$\{\sigma, (\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})\} = \mathcal{F}_{pos}\left(\mathcal{F}_{deform}(\mathcal{F}_{bone}(\boldsymbol{p}, \boldsymbol{e}), \boldsymbol{e})\right), \quad (5)$$

where $\boldsymbol{e}$ is the expression vector. While we could use $\mathcal{F}_{bone}$ at test-time as well, we remove it to improve performance.

The training procedure in this scenario is identical to that used in FastNeRF but with 64 samples in the coarse stage and an additional 64 in the fine stage. The position-dependent network has 384 units in each layer, while the view-dependent network uses only 32 units and 2 layers. The view-dependent network is very small as we believe the scene's illumination is simple and fairly uniform.

The main limitation of this approach in terms of speed is the need to call $\mathcal{F}_{deform}$ for all the input samples. In order to mitigate this, at test-time we implement a grid-based sample pruner. For each position in a grid around the face region we compute the density σ

$$\sigma = \mathcal{F}_{pos}\left(\mathcal{F}_{deform}(\boldsymbol{p}, \boldsymbol{e})\right) \quad (6)$$

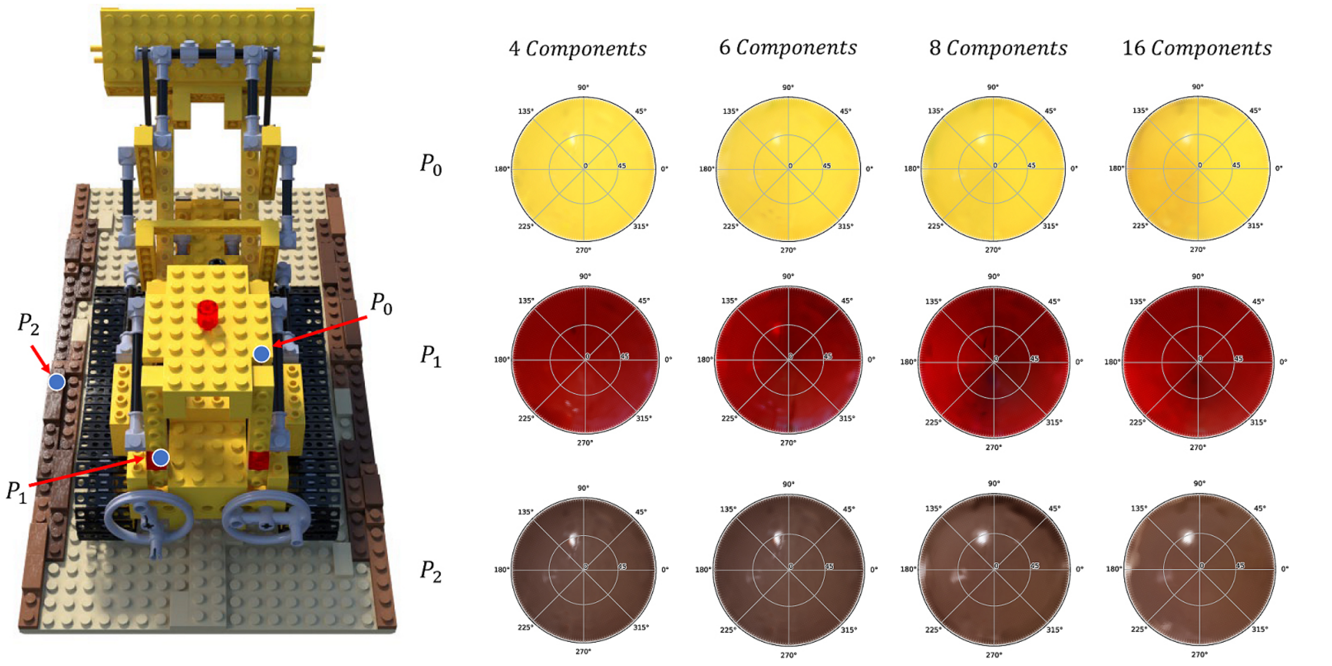

Figure 2. Final integrated pixel colours for three points, P_0, P_1, P_2, visualised with fixed camera position but varying directions over the hemisphere. We observe the presence of strong specular reflections from the directional light of the scene. These specularities are incorrectly reflected in the final colour for the occluded P_1 if less than eight components are used.

for 50 randomly chosen samples of $\boldsymbol{e}$. For each grid position we record the minimum and maximum encountered σ in a sparse volume. At test-time we use this volume to prune the inputs to $\mathcal{F}_{deform}$ where the maximum σ is 0 and where the rays would get saturated, which we estimate using the minimum density values from the volume. This approach reduces the runtime of $\mathcal{F}_{deform}$ by more than a factor of 2.

To further reduce the impact of $\mathcal{F}_{deform}$ on the runtime we only use 43 samples in the coarse stage and remove the fine stage altogether at test time. To offset this low sample count we add additional samples that are linearly interpolated between the samples generated by the deformation network

$$\boldsymbol{p}_{deform} = \mathcal{F}_{deform}(\boldsymbol{p}, \boldsymbol{e})). \tag{7}$$

The additional samples can be evaluated very cheaply as they can be looked-up in the FastNeRF cache. Note that since the deformation network changes the positions of individual samples along the rays, the rays are no longer straight. This means that in this scenario we cannot use the hardware-accelerated ray tracing procedure described in the main paper, though the pruning method described above serves a similar role.

Even though the steps above significantly reduce the runtime of $\mathcal{F}_{deform}$, we are still limited to rendering 300×300 pixel images with a reduced sample count if we want to maintain 30FPS with the deformation network. We observed that if this framerate constraint was to be disregarded, the approach described above is able to generate images at significantly better quality. Thus, we believe that a faster deformation approach remains an important goal for future work.

References

[1] Julian Fong, Magnus Wrenninge, Christopher Kulla, and Ralf Habel. Production volume rendering: Siggraph 2017 course. In *ACM SIGGRAPH 2017 Courses*, SIGGRAPH '17, New York, NY, USA, 2017. Association for Computing Machinery. 3

[2] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. 1

[3] 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 Trans. Graph.*, 38(4), July 2019. 2, 9, 10, 11, 12

[4] 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. *arXiv preprint arXiv:2003.08934*, 2020. 1, 2, 7, 8, 9

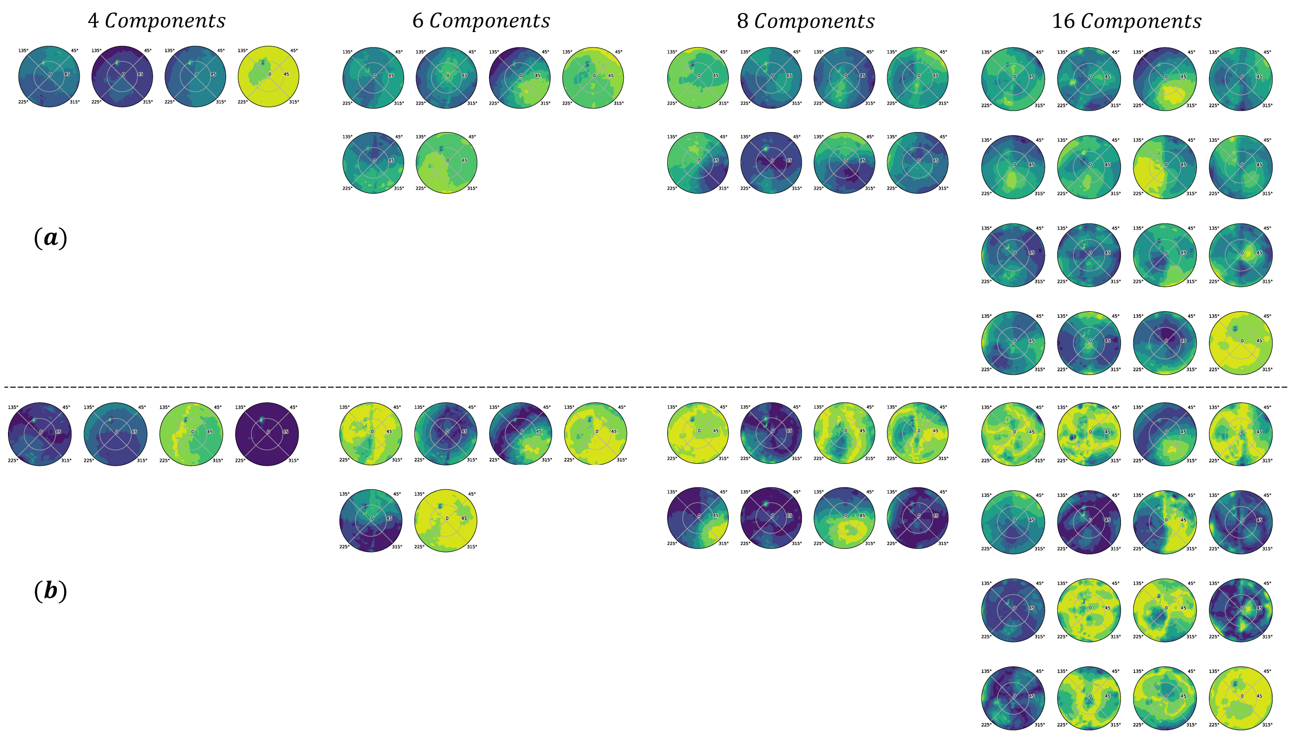

Figure 3. Visualisation of view-dependent strength of the deep radiance map components (best viewed zoomed in). (a) shows the strength of each radiance map component as a function of direction. We note that while some components are similar, there is not obvious redundancy. (b) shows the average absolute contribution of each component for the final colour of point P_2 of Figure 2 using the components shown in (a). Note that all values are normalised for display.

[5] Ken Museth. Hierarchical digital differential analyzer for efficient ray-marching in openvdb. In *ACM SIGGRAPH 2014 Talks*, SIGGRAPH '14, New York, NY, USA, 2014. Association for Computing Machinery. 2

[6] Nvidia. Deep Learning Performance Documentation - Mixed Precision Training. `https://docs.nvidia.com/deeplearning/performance/mixed-precision-training/index.html`, 2021. [Online; accessed 17-June-2021]. 3

[7] Keunhong Park, Utkarsh Sinha, Jonathan T Barron, Sofien Bouaziz, Dan B Goldman, Steven M Seitz, and Ricardo-Martin Brualla. Deformable neural radiance fields. *arXiv preprint arXiv:2011.12948*, 2020. 4

[8] Tiancheng Sun, Henrik Wann Jensen, and Ravi Ramamoorthi. Connecting measured brdfs to analytic brdfs by data-driven diffuse-specular separation. *ACM Transactions on Graphics (TOG)*, 37(6):273, 2018. 3

[9] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. Efficient processing of deep neural networks: A tutorial and survey. *CoRR*, abs/1703.09039, 2017. 3

[10] Bruce Walter, Stephen R. Marschner, Hongsong Li, and Kenneth E. Torrance. Microfacet models for refraction through rough surfaces. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques*, EGSR'07, page 195–206, Goslar, DEU, 2007. Eurographics Association. 3

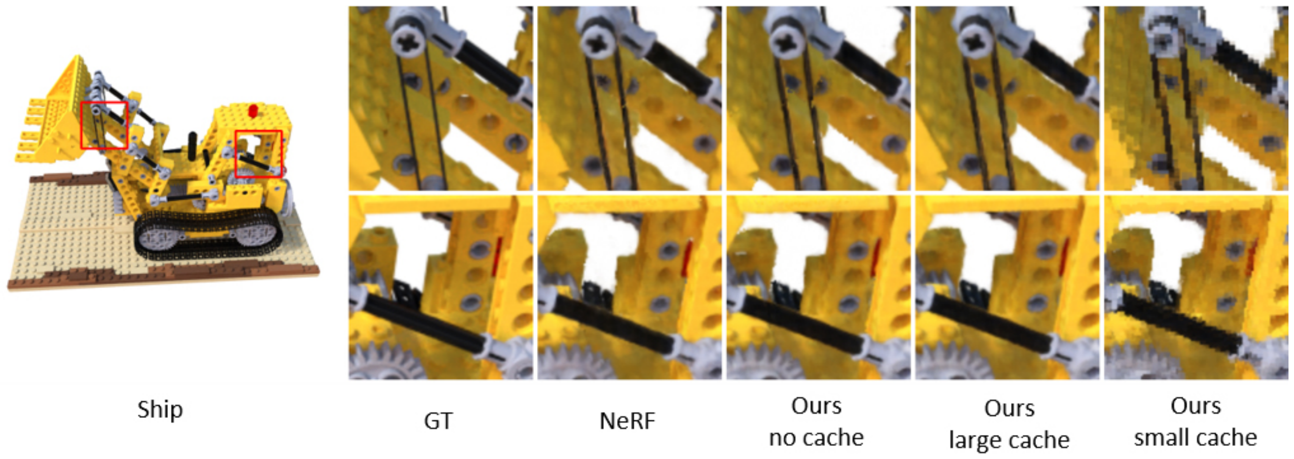

Figure 4. Qualitative comparison of our method vs NeRF on the *Lego* scene from the dataset of [4] at 800^2 pixels using 8 components. *Small cache* refers to our method cached at 256^3, and *large cache* at 1024^3.

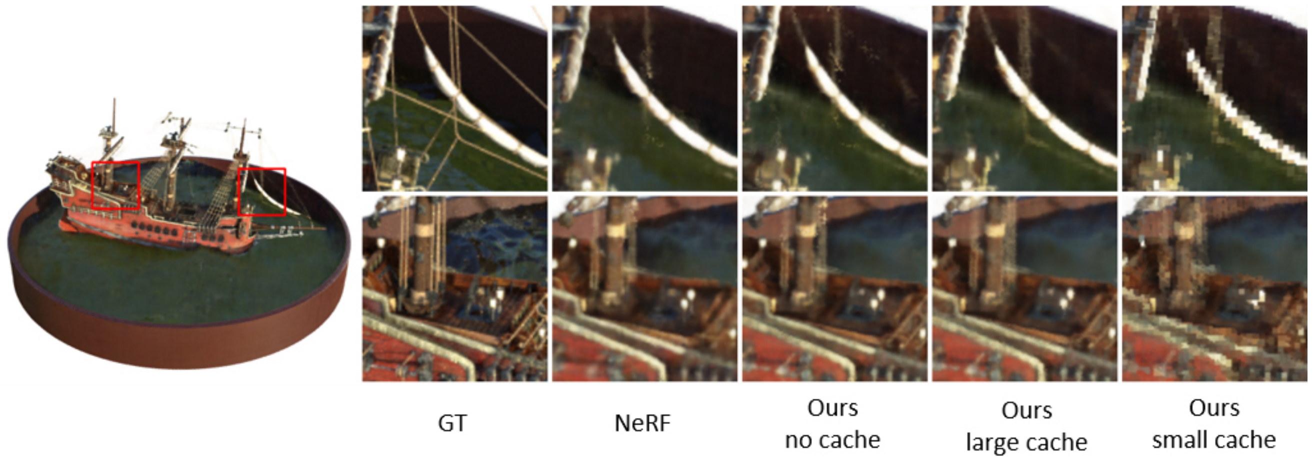

Figure 5. Qualitative comparison of our method vs NeRF on the *Ship* scene from the dataset of [4] at 800^2 pixels using 8 components. *Small cache* refers to our method cached at 256^3, and *large cache* at 1024^3.

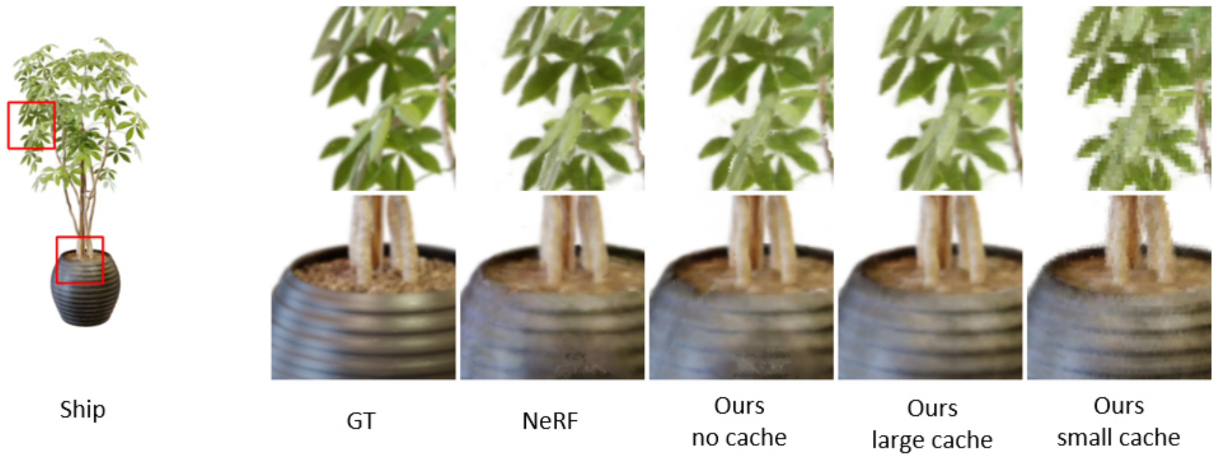

Figure 6. Qualitative comparison of our method vs NeRF on the *Ficus* scene from the dataset of [4] at 800^2 pixels using 8 components. *Small cache* refers to our method cached at 256^3, and *large cache* at 1024^3.

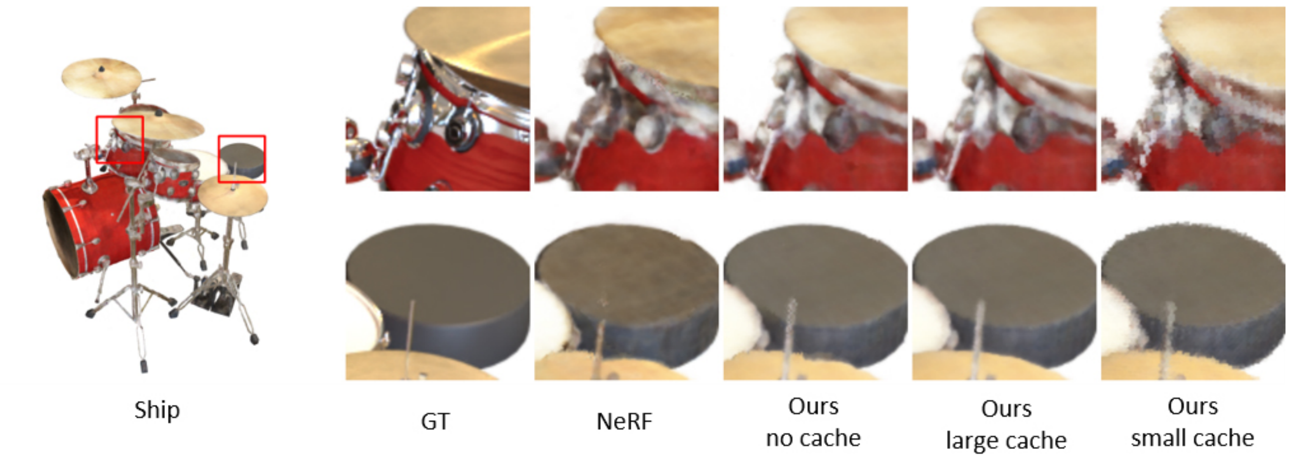

Figure 7. Qualitative comparison of our method vs NeRF on the *Drums* scene from the dataset of [4] at 800^2 pixels using 8 components. *Small cache* refers to our method cached at 256^3, and *large cache* at 1024^3.

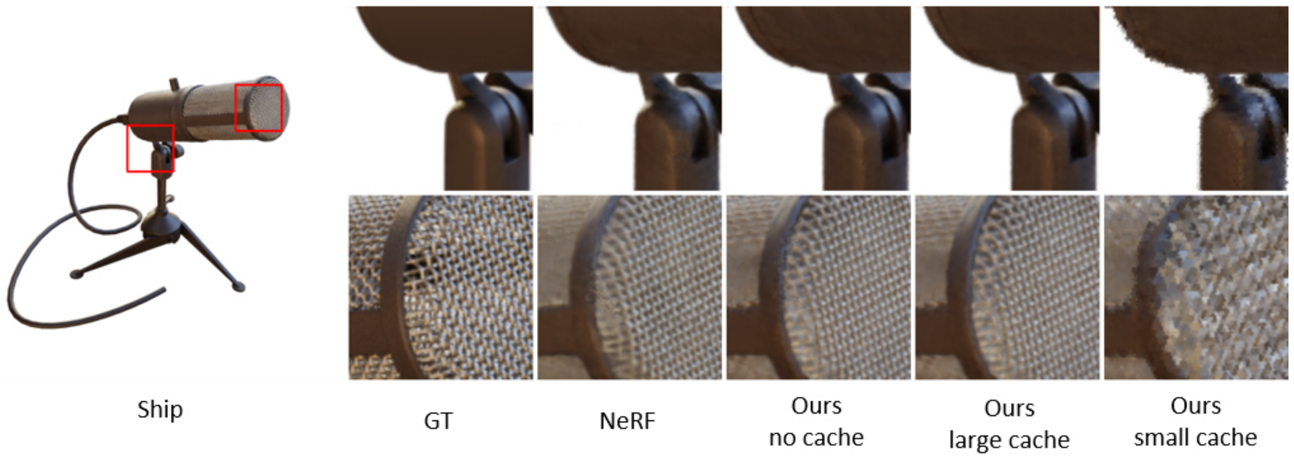

Figure 8. Qualitative comparison of our method vs NeRF on the *Mic* scene from the dataset of [4] at 800^2 pixels using 8 components. *Small cache* refers to our method cached at 256^3, and *large cache* at 1024^3.

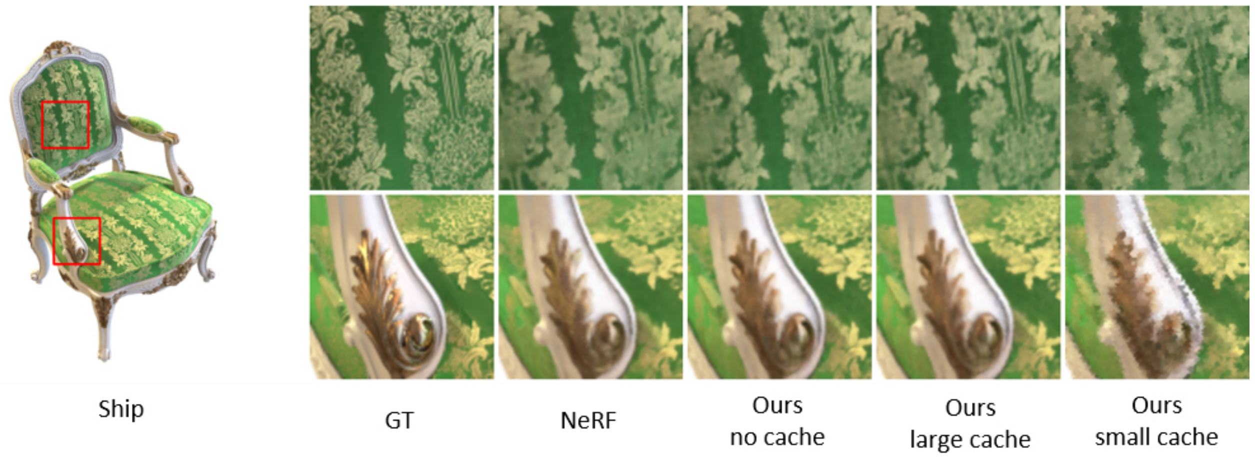

Figure 9. Qualitative comparison of our method vs NeRF on the *Chair* scene from the dataset of [4] at 800^2 pixels using 8 components. *Small cache* refers to our method cached at 256^3, and *large cache* at 1024^3.

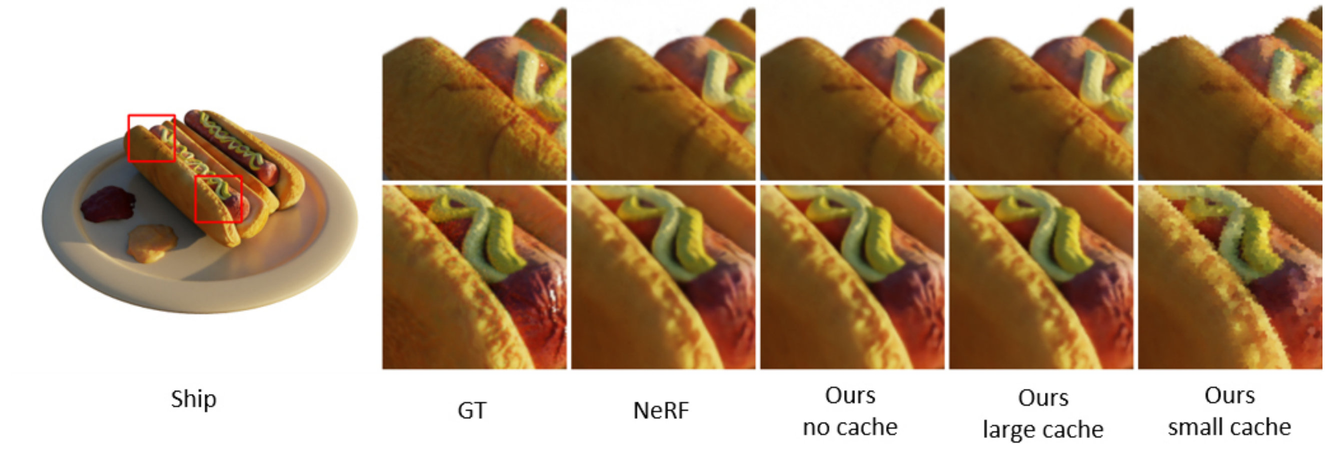

Figure 10. Qualitative comparison of our method vs NeRF on the *Hotdog* scene from the dataset of [4] at 800^2 pixels using 8 components. *Small cache* refers to our method cached at 256^3, and *large cache* at 1024^3.

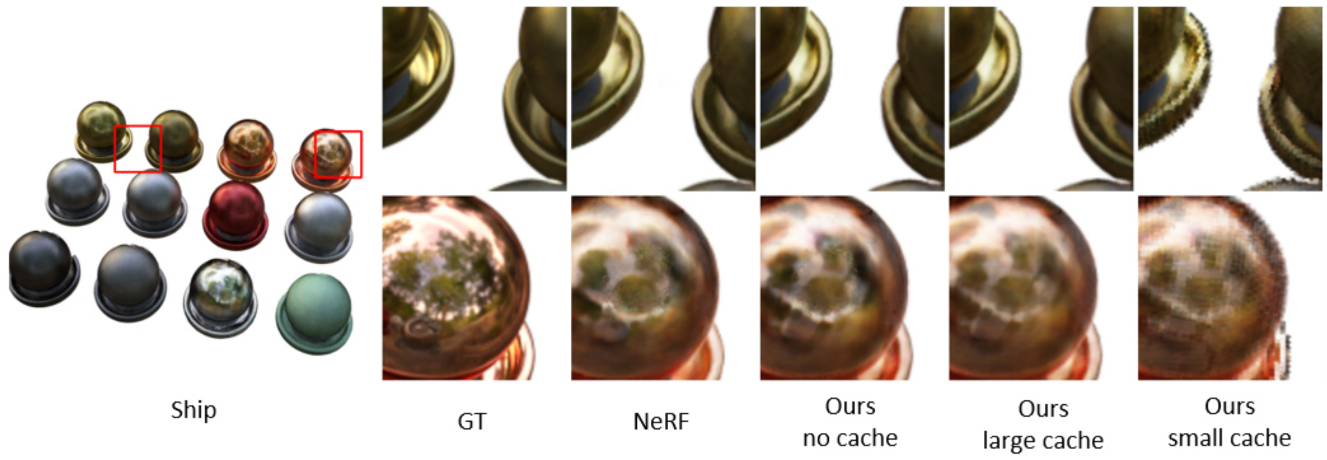

Figure 11. Qualitative comparison of our method vs NeRF on the *Materials* scene from the dataset of [4] at 800^2 pixels using 8 components. *Small cache* refers to our method cached at 256^3, and *large cache* at 1024^3.

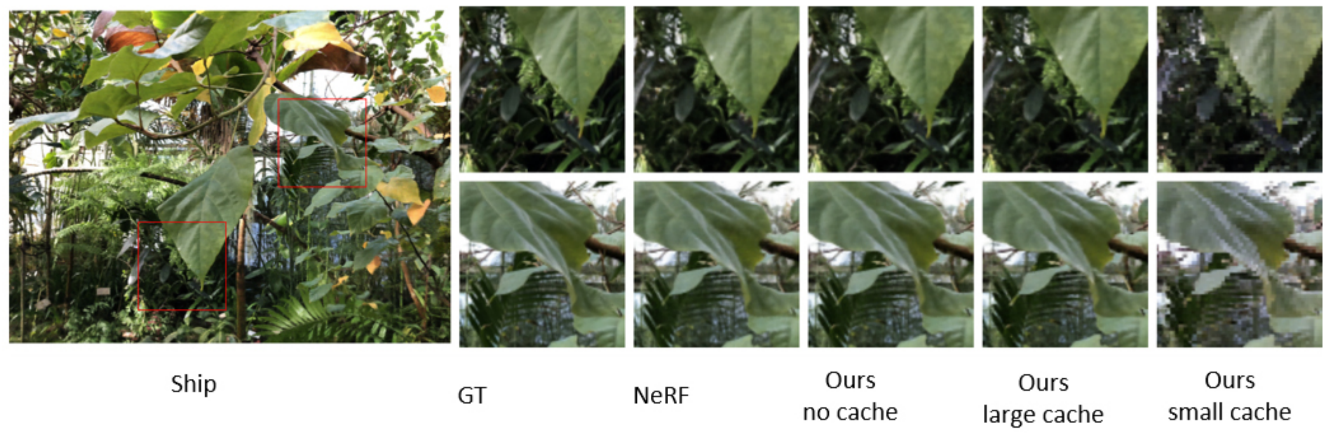

Figure 12. Qualitative comparison of our method vs NeRF on the *Leaves* scene from the dataset of [3] at 504×378 pixels using 6 factors. *Small cache* refers to our method cached at 256^3, and *large cache* at 768^3.

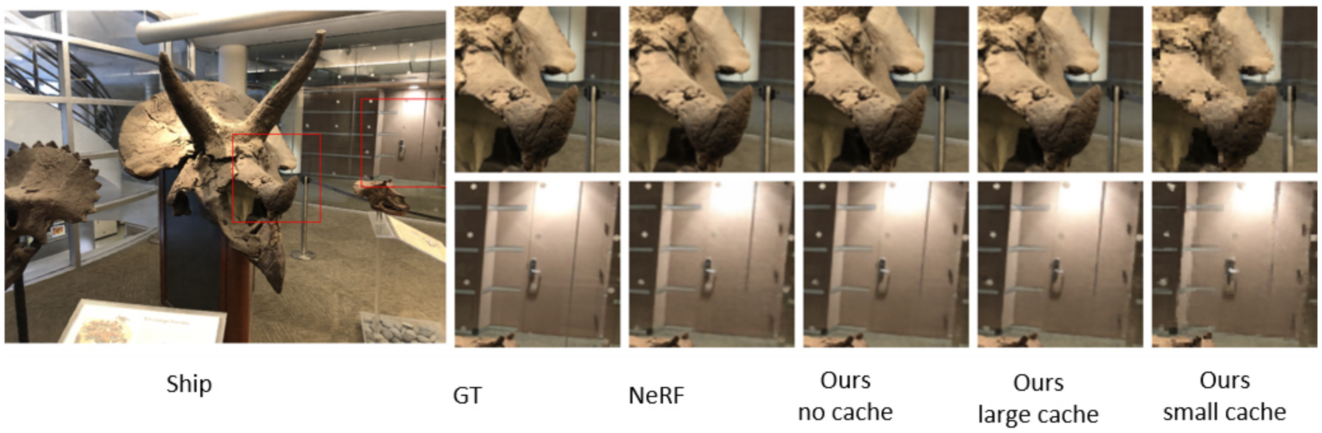

Figure 13. Qualitative comparison of our method vs NeRF on the *Horns* scene from the dataset of [3] at 504 $\times$ 378 pixels using 6 factors. *Small cache* refers to our method cached at 256^3, and *large cache* at 768^3.

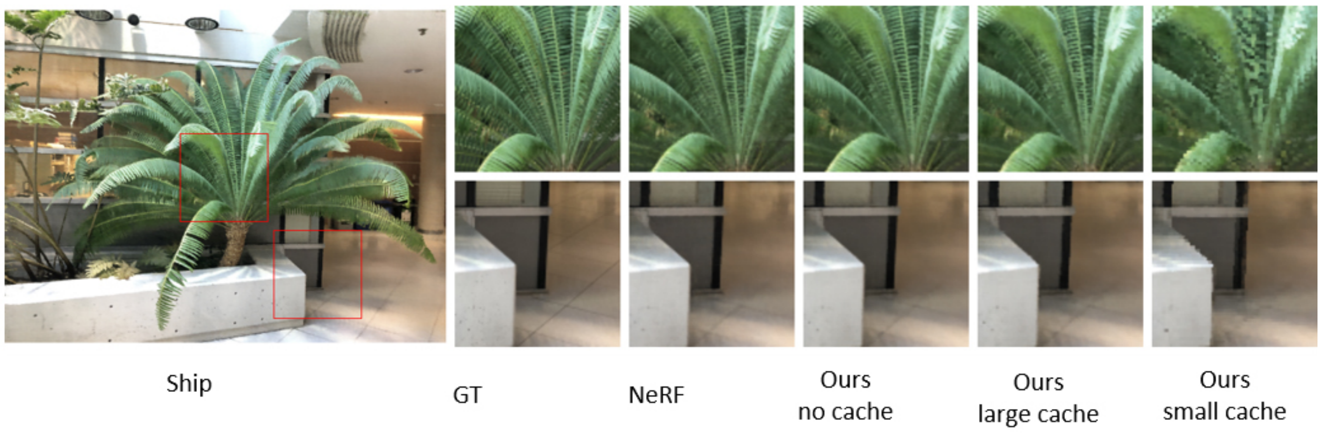

Figure 14. Qualitative comparison of our method vs NeRF on the *Fern* scene from the dataset of [3] at 504 $\times$ 378 pixels using 6 factors. *Small cache* refers to our method cached at 256^3, and *large cache* at 768^3.

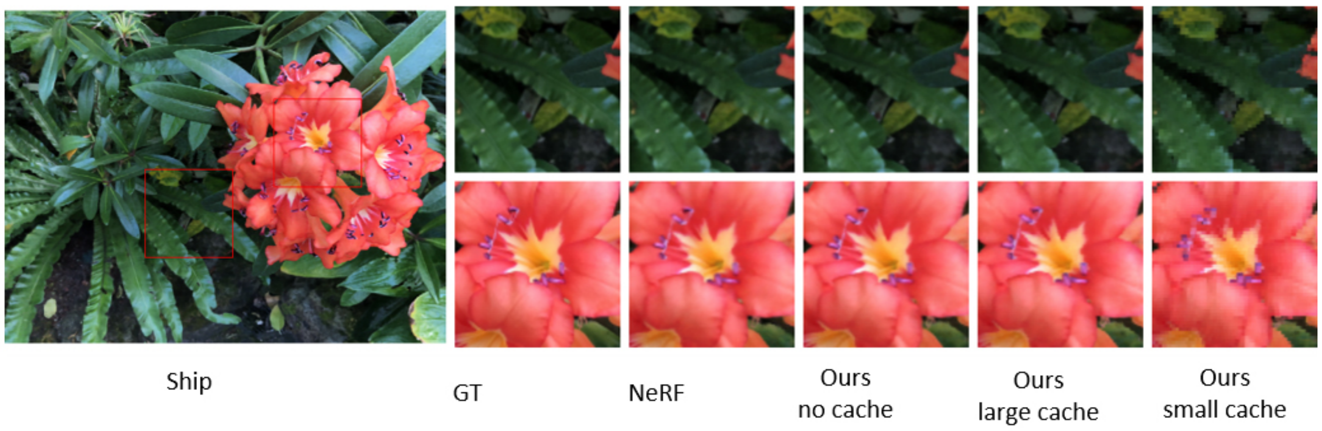

Figure 15. Qualitative comparison of our method vs NeRF on the *Flower* scene from the dataset of [3] at 504 $\times$ 378 pixels using 6 factors. *Small cache* refers to our method cached at 256^3, and *large cache* at 768^3.

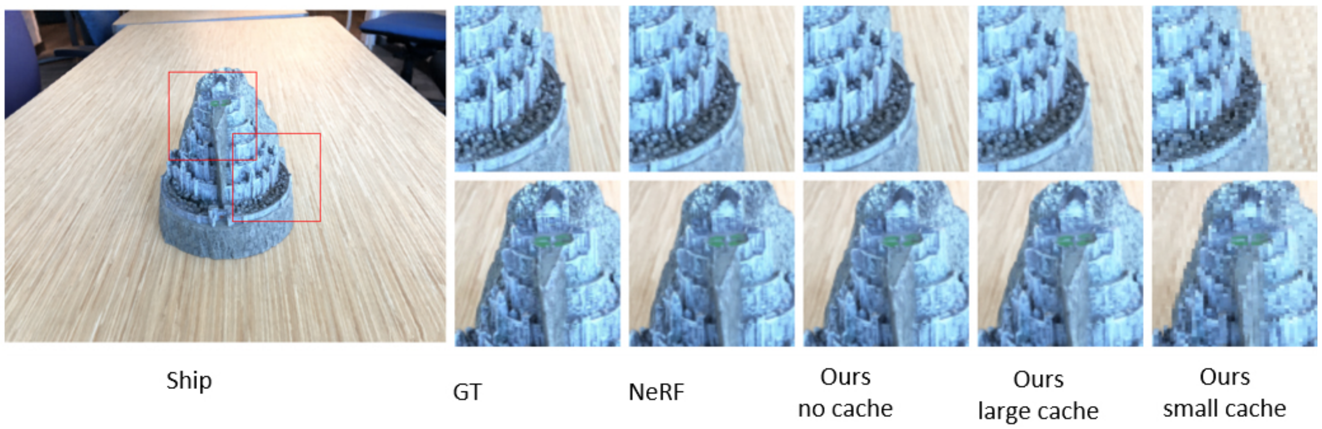

Figure 16. Qualitative comparison of our method vs NeRF on the *Fortress* ('Minas Tirith') scene from the dataset of [3] at 504×378 pixels using 6 factors. *Small cache* refers to our method cached at 256^3, and *large cache* at 768^3.

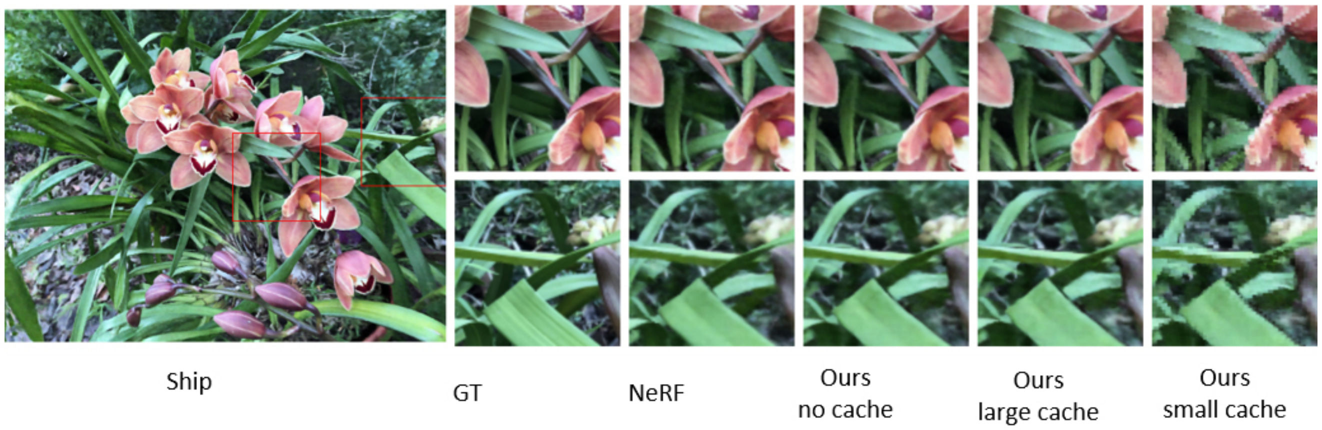

Figure 17. Qualitative comparison of our method vs NeRF on the *Orchids* scene from the dataset of [3] at 504×378 pixels using 6 factors. *Small cache* refers to our method cached at 256^3, and *large cache* at 768^3.

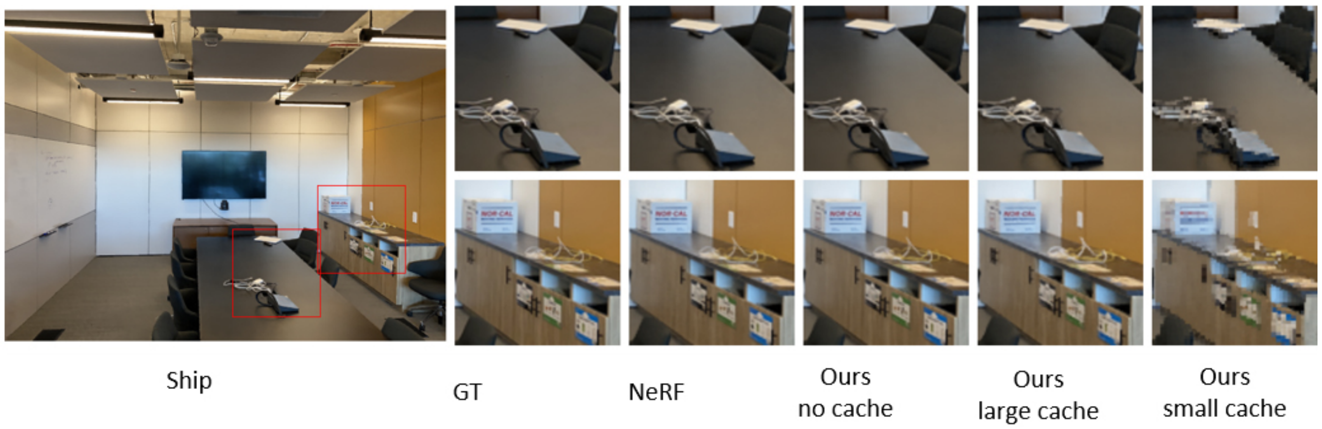

Figure 18. Qualitative comparison of our method vs NeRF on the *Room* scene from the dataset of [3] at 504×378 pixels using 6 factors. *Small cache* refers to our method cached at 256^3, and *large cache* at 768^3.

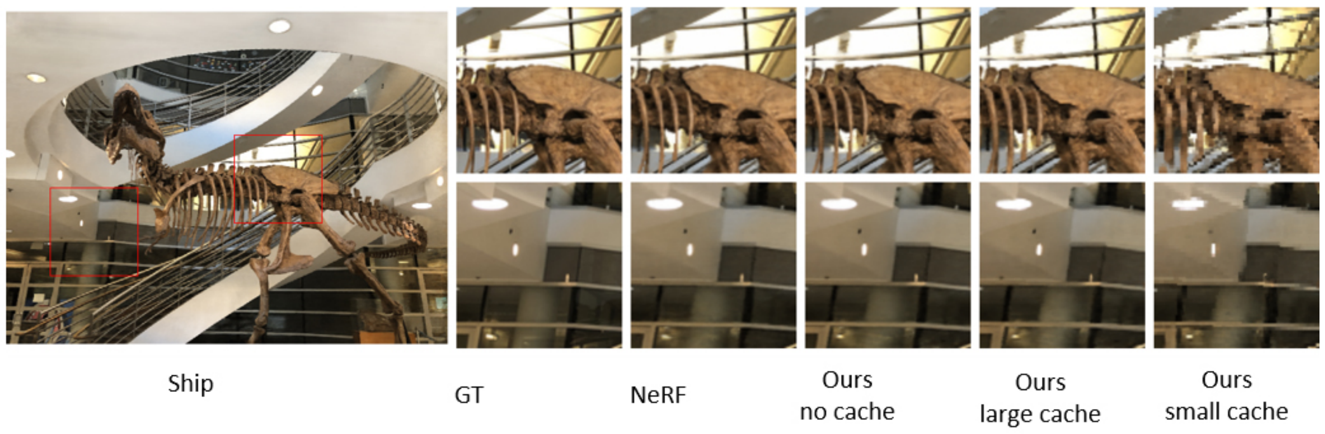

Figure 19. Qualitative comparison of our method vs NeRF on the *TRex* scene from the dataset of [3] at 504×378 pixels using 6 factors. *Small cache* refers to our method cached at 256^3, and *large cache* at 768^3.