HyperReel: High-Fidelity 6-DoF Video with Ray-Conditioned Sampling Supplementary Materials

Benjamin Attal
Carnegie Mellon University

Jia-Bin Huang
Meta & UMD

Christian Richardt
Reality Labs Research

Michael Zollhöfer
Reality Labs Research

Johannes Kopf
Meta

Matthew O'Toole
Carnegie Mellon University

Changil Kim
Meta

1. Appendix Overview

Within the appendix, we provide:

1. Additional details regarding training and evaluation for static and dynamic datasets in Section 3;
2. Additional details regarding sample network design, implementation, and training in Section 4;
3. Additional details regarding keyframe-based volume design in Section 5;
4. Additional quantitative comparisons against static view synthesis approaches on the LLFF [7] and DeepView [3] datasets in Section 6;
5. Additional qualitative comparisons to Neural 3D Video Synthesis [6] on the Technicolor dataset [11] in Section 7;
6. Additional qualitative results for, **(a)** full 360 degree FoV captures and **(b)** highly refractive scenes in Section 8;

Further, we provide a full per-scene breakdown of image metrics for the Technicolor dataset in Table 3, the Neural 3D Video dataset in Table 4, and the Google Immersive Light Field Video dataset in Table 5.

2. Website Overview

Finally, in addition to our appendix, our supplemental website `https://hyperreel.github.io` contains:

1. A link to our codebase;
2. Videos of a demo running in real-time at high-resolution without any custom CUDA code;
3. Dynamic dataset results from our method on each of Technicolor ([11]), Neural 3D Video ([6]), and Google Immersive Video ([2]);
4. Qualitative results and comparisons on view-dependent static scenes from the Shiny Dataset ([18]) and the Stanford Light Field Dataset ([15]);
5. Qualitative comparison to [2].

3. Additional Training & Evaluation Details

3.1. Training Ray-Subsampling

We provide pseudo-code for our ray-subsampling scheme in Algorithm 1, which is used to enable more memory efficient training.

3.2. LPIPS Evaluation Details

For LPIPS computation, we use the AlexNet LPIPS variant for all of our comparisons in the main paper (as do all of the baseline methods).

3.3. SSIM Evaluation Details

For SSIM computation, we use the *structural_similarity scikit-image* library function, with our images normalized to the range of $[0, 1]$, and the *data_range* parameter set to 1. We note, however, that several methods either:

1. Use their own implementation of SSIM, which are not consistent with this standard implementation (e.g. R2L [14]);
2. Fail to set the *data_range* parameter appropriately, so that it defaults to the value of 2.0 (e.g. Neural 3D Video [6]).

In both of these cases, the SSIM function returns higher-than-intended values. While we believe that this inconsistency makes SSIM scores somewhat less reliable, we still report our aggregated SSIM metrics in the quantitative result tables in the main paper.

ALGORITHM 1: Training Ray-Subsampling Scheme

```
Input: Number of videos $\{N\}$, Number of frames $\{M\}$
Output: Training Rays $rays_{GT}$, Ground Truth Colors $C_{GT}$

// Initialize rays and colors
$rays_{GT} = \{\}$
$C_{GT} = \{\}$
// Iterate over all $N$ videos
for $n \in \{1, \cdots, N\}$ do
    // Iterate over all $M$ frames in video $n$
    for $m \in \{1, \cdots, M\}$ do
        // Get frame $m$ from video $n$
        $C_{n,m} = GetFrame(n, m)$
        // Get corresponding rays for this frame
        $rays_{n,m} = GetRays(n, m)$
        if $m$ is not divisible by 8 then
            // Downsample rays and colors by a factor of 4
            $C_{n,m} \leftarrow NearestNeighborDownsample(C_{n,m}, 4)$
            $rays_{n,m} \leftarrow NearestNeighborDownsample(rays_{n,m}, 4)$
            if $m$ is not divisible by 4 then
                // Downsample rays and colors by an additional factor of 2
                $C_{n,m} \leftarrow NearestNeighborDownsample(C_{n,m}, 2)$
                $rays_{n,m} \leftarrow NearestNeighborDownsample(rays_{n,m}, 2)$
            end
        end
        // Add current rays and colors to output
        $C_{GT} \leftarrow C_{GT} + C_{n,m}$
        $rays_{GT} \leftarrow rays_{GT} + rays_{n,m}$
    end
end
```

4. Sample Prediction Network Details

4.1. Additional Training Details

For both static and dynamic datasets, we use a batch size of 16,384 rays for training, an initial learning rate of 0.02 for the parameters of the keyframe-based volume, and an initial learning rate of 0.0075 for our sample prediction network. For Technicolor, Google Immersive, and all static scenes, we set the w_{TV} weight Equation 14 to 0.05 for both appearance and density, which is decayed by a factor of 0.1 every 30,000 iterations. On the other hand, w_{L1} starts at $8{\cdot}10^{-5}$ and decays to $4{\cdot}10^{-5}$ over 30,000 iterations and is only applied to the density components.

4.2. Additional Network Details

In order to make it so that the sample network outputs (primitives $G_1, \ldots, G_n$, point offsets $\mathbf{d}_1, \ldots, \mathbf{d}_n$, velocities $\mathbf{v_1}, \ldots, \mathbf{v}_n$) vary smoothly, we use 1 positional encoding frequency for the ray $\mathbf{r}$ (in both static and dynamic settings) and 2 positional encoding frequencies for the time step τ (in dynamic settings).

4.3. Forward Facing Scenes

For forward facing scenes, we first convert all rays to normalized device coordinates (NDC) [7], so that the view frustum of a "reference" camera lives within $[-1, 1]^3$. After mapping a ray with origin $\mathbf{o}$ and direction $\vec{\boldsymbol{\omega}}$ to its two-plane parameterization [5] (with planes at $z = -1$ and $z = 0$), we predict the parameters of a set of planes normal to the z-axis with our sample network. In particular, we predict $(z_1, \ldots, z_n)$, and intersect the ray with the axis-aligned planes at these distances to produce our sample points $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$. Additionally, we initialize the values $(z_1, \ldots, z_n)$ in a stratified manner, so that they uniformly span the range of $[-1, 1]$.

4.4. Outward Facing Scenes

For all other (outward facing) scenes, we map a ray to its Plücker parameterization via

$$\mathbf{r} = \textit{Plücker}(\mathbf{o}, \vec{\boldsymbol{\omega}}) = (\vec{\boldsymbol{\omega}}, \vec{\boldsymbol{\omega}} \times \mathbf{o})\,. \tag{1}$$

and predict the radii of a set of spheres centered at the origin $(r_1, \ldots, r_n)$. We then intersect the ray with each sphere to produce our sample points. We initialize $(r_1, \ldots, r_n)$ so that they range from the minimum distance to the maximum distance in the scene.

4.5. Differentiable Intersection

In both of the above cases, we make use of the implicit form of each primitive (for planes normal to the z-axis, $z = z_k$, and for the spheres centered at the origin $x^2 + y^2 + z^2 = r_k^2$) and the parameteric equation for a ray $\mathbf{o} + t_k\vec{\boldsymbol{\omega}}$, to solve for the intersection distances t_k (as is done in typical ray-tracers). The intersection distance is differentiable with respect to the primitive parameters, so that gradients can propagate from the color loss to the sample network.

4.6. Implicit Color Correction

In order to better handle multi-view datasets with inconsistent color correction / white balancing, we also output a color scale $\mathbf{c}_k^{scale}$ and shift $\mathbf{c}_k^{shift}$ from the sample prediction network for each sample point $\mathbf{x}_k$. These are used to modulate the color $L_e(\mathbf{x}_k, \vec{\boldsymbol{\omega}}, \tau_i)$ extracted from the dynamic volume via:

$$L_e(\mathbf{x}_k, \vec{\boldsymbol{\omega}}, \tau_i) \leftarrow L_e(\mathbf{x}_k, \vec{\boldsymbol{\omega}}, \tau_i) \cdot \mathbf{c}_k^{scale} + \mathbf{c}_k^{shift}. \quad (2)$$

Note that these outputs vary with low-frequency with respect to the input ray (since we use few positional encoding frequencies for the sample prediction network). Additionally, the density from the volume remains unchanged.

5. Keyframe-Based Volume Details

We initialize our keyframe-based dynamic volume within a 128^3 grid, so that each of the spatial tensor components have resolution 128×128. Our final grid size is 640^3. We upsample the volume at iterations *4,000, 6,000, 8,000, 10,000*, and *12,000*, interpolating the resolution linearly in log space.

6. Quantitative Comparisons

6.1. LLFF Dataset

The LLFF dataset [7] contains eight real-world sequences with 1008×756 pixel images. In Table 1, we compare our method to the same approaches as above on this dataset. Our approach outperforms DoNeRF, AdaNeRF, TermiNeRF, and InstantNGP but achieves slightly worse quality than NeRF. This dataset is challenging for explicit volume representations (which have more parameters and thus can more easily overfit to the training images) due to a combination of erroneous camera calibration and input-view sparsity. For completeness, we also include a comparison to R2L on the downsampled 504×378 LLFF dataset, where we perform slightly worse in terms of quality.

Table 1. **Quantitative comparisons on LLFF.** We compare our approach to others on the real-world LLFF dataset [7]. FPS is normalized per megapixel; memory in MB.

Dataset	Method	PSNR↑	FPS↑	Memory ↓
LLFF 504×378	*Single sample*			
	R2L [13]	**27.7**	—	**23.7**
	Ours (per-frame)	27.5	**4.0**	58.8
LLFF 1008×756	*Uniform sampling*			
	NeRF [7]	**26.5**	0.3	**3.8**
	Instant NGP [8]	25.6	5.3	64.0
	Adaptive sampling			
	DoNeRF [9]	22.9	2.1	4.1
	AdaNeRF [4]	25.7	**5.6**	4.1
	TermiNeRF [10]	23.6	2.1	4.1
	Ours (per-frame)	26.2	4.0	58.8

Table 2. **Quantitative comparisons to DeepView.** In addition to the comparison to NeRFPlayer, we report a comparison with DeepView [3], a variant of which is used per-frame in immersive LF video [2]. We thus compare to DeepView as a proxy for quantitative comparison. FPS normalized per megapixel.

Dataset	Method	PSNR↑	SSIM↑	LPIPS↓	FPS↑
Spaces [3]	DeepView [3]	31.60	0.965	0.085	>**100**
	Ours	**35.47**	**0.968**	**0.080**	4.0

6.2. DeepView Dataset

Unfortunately, Google's Immersive Light Field Video [2] does not provide quantitative benchmarks for the performance of their approach in terms of image quality. As a proxy, we compare our approach to DeepView [3], the method upon which their representation is built, on the static *Spaces* dataset in Table 2.

Our method achieves superior quality, outperforming DeepView by a large margin. Further, HyperReel consumes less memory per frame than the Immersive Light Field Video's baked layered mesh representation: 1.2 MB per frame vs. 8.87 MB per frame (calculated from the reported bitrate numbers [2]). Their layered mesh can render at more than 100 FPS on commodity hardware, while our approach renders at a little over 4 FPS. However, our approach is entirely implemented in vanilla PyTorch and can be further optimized using custom CUDA kernels or baked into a real-time renderable representation for better performance.

7. Qualitative Comparisons to Neural 3D [6]

We provide additional qualitative still-frame comparisons to Neural 3D Video Synthesis [6] in Figure 1.

8. Additional Results

8.1. Panoramic 6-DoF Video

In general, our method can support an unlimited FoV. We show a panoramic rendering of a synthetic 360 degree scene from our model, using spherical primitives in Figure 2.

8.2. Point Offsets for Modeling Refractions

Point offsets allow the sample network to capture appearance that violates epipolar constraints, noticeably improving quality for refractive scenes. We show a visual comparison between our approach with and without point offsets in Figure 3. More results are available on the website.

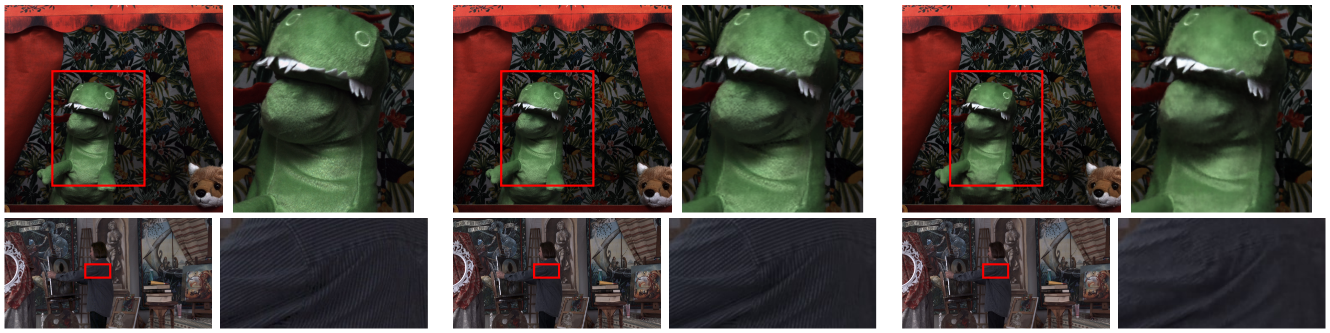

Figure 1. **Additional qualitative comparisons to Neural 3D Video Synthesis.** We show two additional qualitative comparisons against Neural 3D Video Synthesis [6] on the Technicolor dataset [11], demonstrating that our approach recovers more accurate/detailed appearance.

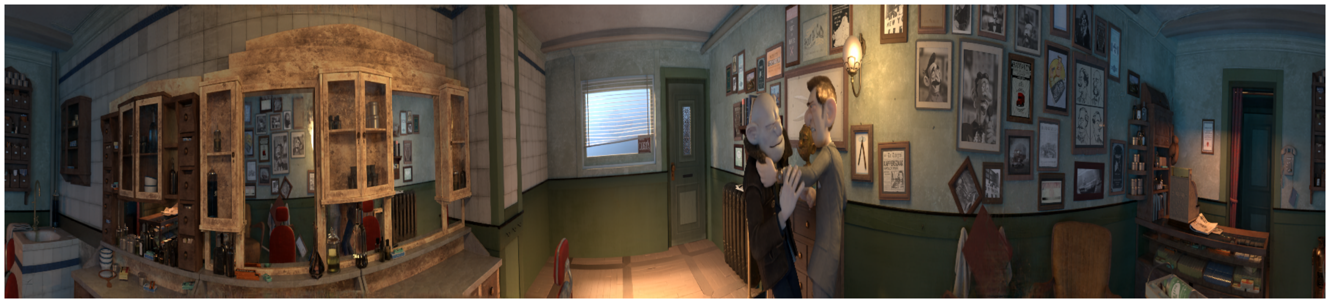

Figure 2. Example panoramic rendering from our approach applied to a synthetic scene with captures spanning a full 360 degree FoV. In this case, our sample network predicts spherical geometric primitives. The scene is one of the shots from the Blender Foundations Agent 327 open movie [1].

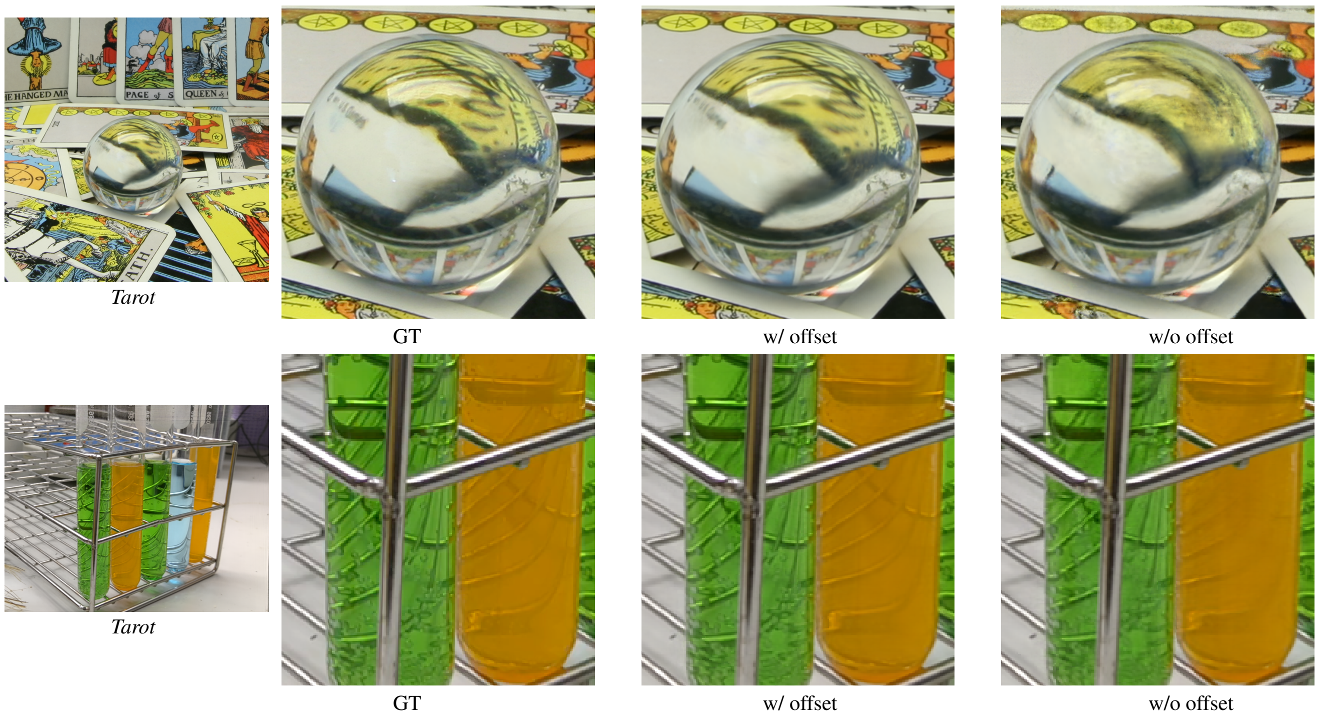

Figure 3. Comparison of our method with and without point offset on the *Tarot* sequence from the Stanford Light Field dataset [16] and *Lab* sequence from the Shiny dataset [17].

Table 3. Per-scene results from the Technicolor dataset [11]. See Section Section 3.3 for a discussion of the reliability of SSIM metrics.

Scene	PSNR↑				SSIM↑				LPIPS↓			
	Neural 3D Video [6]	Ours	Small	Tiny	Neural 3D Video [6]	Ours	Small	Tiny	Neural 3D Video [6]	Ours	Small	Tiny
Birthday	29.20	**29.99**	29.32	27.80	**0.952**	0.922	0.907	0.876	0.0668	**0.0531**	0.0622	0.0898
Fabien	32.76	**34.70**	33.67	32.25	**0.965**	0.895	0.882	0.860	0.2417	**0.1864**	0.1942	0.2233
Painter	35.95	35.91	**36.09**	34.61	**0.972**	0.923	0.920	0.905	0.1464	**0.1173**	0.1182	0.1311
Theater	29.53	**33.32**	32.19	30.74	**0.939**	0.895	0.880	0.845	0.1881	**0.1154**	0.1306	0.1739
Trains	**31.58**	29.74	27.51	25.02	**0.962**	0.895	0.835	0.773	**0.0670**	0.0723	0.1196	0.1660

Table 4. Per-scene results from the Neural 3D Video dataset [6], available only for our method and NeRFPlayer [12].

Scene	PSNR↑		SSIM↑		LPIPS↓	
	NeRFPlayer [12]	Ours	NeRFPlayer [12]	Ours	NeRFPlayer [12]	Ours
Coffee Martini	**31.534**	28.369	**0.951**	0.892	**0.085**	0.127
Cook Spinach	30.577	**32.295**	0.929	**0.941**	0.113	**0.089**
Cut Roasted Beef	29.353	**32.922**	0.908	**0.945**	0.144	**0.084**
Flame Salmon	**31.646**	28.260	**0.940**	0.882	**0.098**	0.136
Flame Steak	31.932	**32.203**	**0.950**	0.949	0.088	**0.078**
Sear Steak	29.129	**32.572**	0.908	**0.952**	0.138	**0.077**

Table 5. Per-scene results from the Google Immersive Light Field Video dataaset [2], available only for our method and NeRFPlayer [12].

Scene	PSNR↑		SSIM↑		LPIPS↓	
	NeRFPlayer [12]	Ours	NeRFPlayer [12]	Ours	NeRFPlayer [12]	Ours
01_Welder	**25.568**	25.554	**0.818**	0.790	0.289	**0.281**
02_Flames	26.554	**30.631**	0.842	**0.905**	**0.154**	0.159
04_Truck	27.021	**27.175**	**0.877**	0.848	**0.164**	0.223
09_Exhibit	24.549	**31.259**	0.869	**0.903**	0.151	**0.140**
10_Face_Paint_1	27.772	**29.305**	**0.916**	0.913	0.147	**0.139**
11_Face_Paint_2	**27.352**	27.336	**0.902**	0.879	**0.152**	0.195
12_Cave	21.825	**30.063**	0.715	**0.881**	0.314	**0.214**

References

[1] 5

[2] Michael Broxton, John Flynn, Ryan Overbeck, Daniel Erickson, Peter Hedman, Matthew DuVall, Jason Dourgarian, Jay Busch, Matt Whalen, and Paul Debevec. Immersive light field video with a layered mesh representation. *ACM Trans. Graph.*, 39(4):86:1–15, 2020. 1, 3, 6

[3] John Flynn, Michael Broxton, Paul Debevec, Matthew DuVall, Graham Fyffe, Ryan Overbeck, Noah Snavely, and Richard Tucker. DeepView: View synthesis with learned gradient descent. In *CVPR*, 2019. 1, 3

[4] Andreas Kurz, Thomas Neff, Zhaoyang Lv, Michael Zollhöfer, and Markus Steinberger. AdaNeRF: Adaptive sampling for real-time rendering of neural radiance fields. In *ECCV*, 2022. 3

[5] Marc Levoy and Pat Hanrahan. Light field rendering. In *SIGGRAPH*, 1996. 2

[6] Tianye Li, Mira Slavcheva, Michael Zollhöfer, Simon Green, Christoph Lassner, Changil Kim, Tanner Schmidt, Steven Lovegrove, Michael Goesele, Richard Newcombe, and Zhaoyang Lv. Neural 3D video synthesis from multi-view video. In *CVPR*, 2022. 1, 3, 5, 6

[7] 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. In *ECCV*, 2020. 1, 2, 3

[8] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–15, 2022. 3

[9] Thomas Neff, Pascal Stadlbauer, Mathias Parger, Andreas Kurz, Chakravarty R. Alla Chaitanya, Anton Kaplanyan, and Markus Steinberger. DONeRF: Towards real-time rendering of neural radiance fields using depth oracle networks. *Comput. Graph. Forum*, 2021. 3

[10] Martin Piala and Ronald Clark. TermiNeRF: Ray termination prediction for efficient neural rendering. In *3DV*, 2021. 3

[11] Neus Sabater, Guillaume Boisson, Benoit Vandame, Paul Kerbiriou, Frederic Babon, Matthieu Hog, Remy Gendrot, Tristan Langlois, Olivier Bureller, Arno Schubert, et al. Dataset and pipeline for multi-view light-field video. In *CVPR Workshops*, 2017. 1, 5, 6

[12] Liangchen Song, Anpei Chen, Zhong Li, Zhang Chen, Lele Chen, Junsong Yuan, Yi Xu, and Andreas Geiger. NeRFPlayer: A streamable dynamic scene representation with decomposed neural radiance fields. *TVCG*, 2023. 6

[13] Huan Wang, Jian Ren, Zeng Huang, Kyle Olszewski, Menglei Chai, Yun Fu, and Sergey Tulyakov. R2L: Distilling neural radiance field to neural light field for efficient novel view synthesis. In *ECCV*, 2022. 3

[14] Liao Wang, Jiakai Zhang, Xinhang Liu, Fuqiang Zhao, Yanshun Zhang, Yingliang Zhang, Minye Wu, Lan Xu, and Jingyi Yu. Fourier PlenOctrees for dynamic radiance field rendering in real-time. In *CVPR*, 2022. 1

[15] Bennett Wilburn, Neel Joshi, Vaibhav Vaish, Eino-Ville Talvala, Emilio R. Antúnez, Adam Barth, Andrew Adams, Mark Horowitz, and Marc Levoy. High performance imaging using large camera arrays. *ACM Trans. Graph.*, 24(3):765–776, 2005. 1

[16] Bennett Wilburn, Neel Joshi, Vaibhav Vaish, Eino-Ville Talvala, Emilio Antunez, Adam Barth, Andrew Adams, Mark Horowitz, and Marc Levoy. High performance imaging using large camera arrays. *ACM Trans. Graph.*, 24(3):765–776, 2005. 5

[17] Suttisak Wizadwongsa, Pakkapon Phongthawee, Jiraphon Yenphraphai, and Supasorn Suwajanakorn. NeX: Real-time view synthesis with neural basis expansion. In *CVPR*, 2021. 5

[18] Suttisak Wizadwongsa, Pakkapon Phongthawee, Jiraphon Yenphraphai, and Supasorn Suwajanakorn. NeX: Real-time view synthesis with neural basis expansion. In *CVPR*, 2021. 1