

# INPC: Implicit Neural Point Clouds for Radiance Field Rendering

## Supplementary Material

### Outline

To facilitate navigation of this document, we provide an overview of its content:

- Sec. 1 details our implementation to ensure reproducibility.
- Sec. 2 discusses additional ablations for further insights w.r.t. our algorithmic and architectural design choices.
- Sec. 3 presents per-scene metrics and visual comparisons.
- Sec. 4 describes our perceptual experiment in detail.

### 1. Implementation

We believe that the flexibility of our implicit point cloud could prove to be useful for future work. Here, we try to support such work by providing all details of our implementation.

#### 1.1. Overview

We implement our method using PyTorch and create reusable PyTorch extensions using C++/CUDA for our differentiable rasterization module and multiple – otherwise slightly slower – stages of our sampling process. Furthermore, we use multiple libraries that provide efficient implementations for parts of our pipeline. For the multi-resolution hash grid as well as the background MLP, we use the implementation provided with the *tiny-cuda-nn* (TCNN) framework [16]. As our optimizer, we use the fused ADAM [11] implementation provided by the NVIDIA Apex library. To reduce computation time of the CNN, we also make use of PyTorch’s automatic mixed precision package and scale FP16 gradients by a fixed factor of 128 during the backward pass.

#### 1.2. Pre-Processing Details

**Lens Distortion.** For the Tanks and Temples dataset [13], the original images are subject to slight lens distortions. While our rendering supports radial and tangential distortion, we use COLMAP [20, 21] to extract undistorted images after camera calibration to ensure fair comparison against the selected baselines. Specifically, 3DGS [10] only works with a pinhole camera model. Note that the undistorted images are used by all of the compared methods, including INPC.

**Scene Normalization.** We follow Barron et al. [2] and apply a world-space transformation to all camera poses so that they fit inside a cube  $[-1, 1]^3$ . We apply the same transformation to the SfM point cloud if it is used. For all scenes, we set the values for near and far plane to 0.01 and 100 respectively.

#### 1.3. Optimization Details

For ADAM, we use  $\beta_1 = 0.9$ ,  $\beta_2 = 0.99$ ,  $\epsilon = 10^{-15}$ , and disable weight decay. The learning rates for all parameters are exponentially decayed during the optimization. For the hash grid and MLP parameters, we decay the learning rate from  $1e-2$  to  $3e-4$ , while CNN parameter learning rates are decayed from  $3e-4$  to  $5e-5$ .

When using differentiable tone mapping as proposed by Rückert et al. [19], we follow the authors and only activate the sub-modules for exposure and camera response. Similar to Rückert et al. [19], these parameters are updated with a fixed learning rate of  $5e-4$  and  $1e-3$  respectively. However, we use a warm-up strategy [1] where the learning rate cosine-delayed by multiplying with a factor between 0.01 and 1 during the first 5,000 iterations. This is slightly different from the original version that simply disabled optimization of tone mapping parameters during the first few iterations.

#### 1.4. Point Probability Octree Details

**Initialization.** For each scene, we define  $b_{\min}, b_{\max} \in \mathbb{R}^3$  as the axis-aligned bounding box of the SfM point cloud and enlarge the result by 10% for increased robustness. If no SfM point cloud is available, we instead use the smallest axis-aligned box enclosing the viewing frusta of all training cameras. Given a desired initial resolution  $R$  and the aforementioned bounding box of the scene, we initialize our octree-based data structure as a regular grid of cube-shaped voxels. These 3D voxels are the initial leaf nodes of  $\mathcal{P}$ . For each leaf node, we store its center  $c_i \in \mathbb{R}^3$  in world space, current subdivision level  $l_i \in \mathbb{N}_0$ , sampling probability  $p_i \in [0, 1]$ , and subdivision “probability”  $q_i \in [0, 1]$ . We limit values of  $R$  to powers of 2 and can thus initialize the initial subdivision level as  $\log_2(N)$  for all leaf nodes. To incorporate a point cloud  $\mathcal{I}$  as a prior, we modify initial sampling probability: Assuming  $|\mathcal{I}_i|$  is the number of points inside the volume of the  $i$ -th leaf node and  $Q$  is the 0.95-quantile of these point counts, we compute the initial probability as  $\frac{|\mathcal{I}_i|}{Q}$ , clipping values with 0.1 and 1.0 for increased robustness.

**Ensuring Sample Count.** For our viewpoint-specific sampling, a major difficulty comes from dealing with those leaves whose volume intersects the faces of the viewing frustum. For these, it is possible that positions generated by our sampling scheme lie outside the frustum. To this end, we re-sample until the desired number of samples is reached. Note that our re-weighting scheme for sampling probabilities does not account for partial visibility of voxels. This means simply placing points inside the visible volume of each leaf

would slightly bias sampling towards partially visible leaves. We avoid this by re-sampling the viewpoint-specific probability distribution for affected samples repeatedly until all generated samples are visible. Notably, our multisampling approach allows us to omit this step during inference, which speeds up sampling.

**Depth Calculation.** For our view-specific re-weighting scheme, we look to obtain a view-specific depth value  $d_i \in [0, 1]$  for each leaf node. To this end, we adjust the distance of each leaf center to the image plane, i.e., the view-space depth value  $z_{\text{view}}$ :

$$d_i = \max\left(\frac{|z_{\text{view}} - \text{near}|}{\text{far}}, \tau_d\right). \quad (1)$$

Note that  $|z_{\text{view}} - \text{near}|$  accounts for the fact that  $z_{\text{view}}$  may be smaller than "near" for partially visible leaves (see the previous paragraph). To prevent extreme cases of oversampling close to the camera, we use a threshold  $\tau_d = 10^{-8}$  in all experiments.

## 1.5. TCNN Details

For the multi-resolution hash grid and the accompanying MLP, we use the implementation from the TCNN framework [16]. We tested different configurations using those employed by prior works [3, 17, 26] as a starting point. Our resulting configuration is shown in Listing 1. It is similar to the one used by Zip-NeRF [3], but we increase the hashmap size from  $2^{21}$  to  $2^{23}$  (see Sec. 2 for ablations). For the background MLP we also use the fully-fused implementation from the TCNN framework [16]. It has four hidden layers with 128 neurons each and uses ReLU activations. For each pixel, we encode its normalized viewing direction using spherical harmonics of degree three before inputting into the MLP. Note that in TCNN this means using 'degree': 4 and converting input directions to  $[0, 1]^3$ .

## 1.6. Bilinear Splatting Details

Prior work on point rasterization has demonstrated that software implementations are a) very fast [22, 23] compared to

hardware implementations, e.g., GL\_POINTS, and b) allow for backward pass implementations suited for gradient-based optimization [14, 19]. Motivated by these insights, we set out to design our own rasterization module in an effort to adapt it to the needs of our rendering pipeline.

Part of why MLP-based radiance fields works so well is the fully differentiable color accumulation of multiple samples for a given pixel. While NeRF uses a volume rendering-based formulation, recent successful point-based methods [7, 10] use conventional  $\alpha$ -blending, which is similarly favorable for gradient-based optimization. We deem the extra cost of sorting points worthy with respect to the superior optimization properties.

The second consideration is the selection of points blended into each pixel. Typically, each point is projected onto a single pixel using the extrinsic and intrinsic camera matrices. Using this approach, the unstructured and disconnected nature of point clouds may cause large holes in the resulting image. To this end, we follow the common practice of optimizing a hole-filling CNN alongside the rest of our model, which helps a great deal but causes other downstream problems such as temporal instabilities. However, other approaches for addressing the issue exist. Among these, splatting is a well-established technique in rasterization-based rendering where the same point may influence multiple pixels at once. Furthermore, splatting ameliorates the problems arising from the need of discrete pixel coordinates for drawing each point. While, mathematically speaking, a point is always projected onto a single pixel, giving the same weight to points projected near pixel boundaries and those projected into the center of a pixel can be problematic for gradient-based optimization. We therefore splat each point into the four closest pixels and achieve this by downweighting its opacity using bilinear interpolation before blending it into each pixel (see main paper). During  $\alpha$ -blending, we stop accumulation for a pixel once the remaining transmittance falls below  $10^{-4}$ . The full algorithm, which we implement in CUDA to achieve accelerated rendering times, can be written as shown in Algorithm 1. We use the NVIDIA CUB library for the exclusive sum as well as the global radix sorting of key/value pairs. As we identify global memory accesses to

**Listing 1 Hash grid configuration** for the *tiny-cuda-nn* implementation [16].

```
encoding_config = {
    'otype': 'Grid',
    'type': 'Hash',
    'n_levels': 10,
    'n_features_per_level': 4,
    'log2_hashmap_size': 23,
    'base_resolution': 16,
    'per_level_scale': 2.0,
    'interpolation': 'Linear'
}
network_config = {
    'otype': 'FullyFusedMLP',
    'activation': 'ReLU',
    'output_activation': 'None',
    'n_neurons': 64,
    'n_hidden_layers': 1
}
```

**Algorithm 1 GPU rasterization with bilinear splatting and  $\alpha$ -blending**

```
Input:  $C$ : Camera model
Input:  $P, A, F$ : Point positions, opacities, and features
Output:  $I$ : Image
1: function RASTERIZE( $C, P, A, F$ )
2:    $P' \leftarrow \text{PROJECTANDCULL}(P, C)$ 
3:    $A', K, N \leftarrow \text{BILINEARSPLATTING}(P', A)$   $\triangleright$  Splat  $\alpha$ , keys, counts
4:    $L \leftarrow \text{EXCLUSIVESUM}(N)$   $\triangleright$  Splat indices
5:    $\text{RADIXSORTPAIRS}(K, L)$   $\triangleright$  Global sort
6:    $I \leftarrow \text{BLENDSORTEDSPLATS}(A', F, L, N)$   $\triangleright$  Per-pixel  $\alpha$ -blending
7:   return  $I$ 
8: end function
```



be the main performance bottleneck for our blending kernel, we implement it using CUDA’s warp-level primitives distributing the workload for each pixel over a full warp of 32 threads. The blending weight computation required for updating  $\mathcal{P}$  as well as the spherical harmonics computations are fused into this pipeline. During optimization, we store the bilinear interpolation weights, per-pixel point counts, and sorted key/index buffers for the backward pass. To compute per-splat gradients we process them in the same order as in the forward pass. We obtain a per-point gradient by combining its four splat gradients in a weighted sum according to the respective bilinear interpolation weights, i.e., the analytical derivative of the bilinear splatting process. For the probability field updates, we also need to combine per-splat blending weights of each point. Interestingly, we observe that the use of a weighted sum according to the respective bilinear interpolation weights outperforms the theoretically correct approach of a simple sum in all our experiments. We believe this is because the weighted sum causes the resulting per-point blending weight to more closely resemble the expected visibility of future samples from the leaf from which the point was sampled.

## 1.7. CNN Details

For the rendering network, we use a standard three-layer U-Net architecture with 64 initial filters, GELU activations, average pooling for downsampling, and bilinear interpolation followed by a point-wise convolution for upsampling. One important change is the introduction of a single residual block based on *Fast Fourier Convolution (FFC)* [5]. We use the authors’ implementation with  $\alpha_{in}, \alpha_{out} = 0.75$  and no further modifications. To avoid having to crop or pad when concatenating tensors for the employed skip-connections in the expansive path of the U-Net, we pad the target width and height to a multiple of  $2^{L-1}$ , where  $L$  is the number of layers in the U-Net, before initiating our rendering pipeline. Note that this requires adjusting the principal point to allow for renderings that – after removing the padded pixels – can properly be compared to ground truth images for loss or quality metric computation.

## 2. Additional Ablations

In Tab. 1, we show additional model ablations computed by running various versions of our 8M configuration on the five outdoor scenes from the Mip-NeRF360 dataset [2]. For all configurations, we also include model size as well as optimization time and inference frame rate measured on the *Bicycle* scene ( $1237 \times 822$  pixels). To avoid any confusion, we want to mention that we re-computed the results of the baseline configuration *Ours* (8M) for the ablation studies. As our method produces marginally different results in every run, the results are not exactly the same as in Tab. 2. This also applies to the corresponding tables from the main pa-

Configuration	LPIPS↓	SSIM↑	PSNR↑	Train (hrs)↓	Render (fps)↑	Size (GB)↓
A) No D-SSIM Loss	0.204	0.729	25.27	4.24	9.8	1.1
B) No VGG Loss	0.238	0.754	25.29	3.43	9.8	1.1
C) No Weight Decay	0.210	0.744	25.22	3.69	9.8	1.1
D) No Subdivision	0.508	0.406	19.15	4.07	9.9	0.88
E) No Bilinear Splatting	0.243	0.708	24.36	4.14	11.7	1.1
F) No Multisampling	0.201	0.740	25.02	4.25	26.3	1.1
G) No SfM Prior	0.197	0.753	25.29	4.25	9.8	1.1
H) No Background Model	0.197	0.752	25.28	4.24	9.9	1.1
I) No FFC Block	0.207	0.749	25.32	4.15	10.0	1.1
J) No Post-Processing	0.277	0.718	24.32	4.05	5.6	1.1
$\mathcal{P}$ : No Viewpoint Bias	0.328	0.618	23.23	3.83	10.1	1.0
$\mathcal{P}$ : No Depth Re-weighting	0.226	0.721	24.72	4.25	9.8	1.1
$\mathcal{P}$ : $d_i \rightarrow d_i^2$	0.210	0.740	25.04	4.25	9.8	1.1
$\mathcal{P}$ : No Size Re-weighting	0.198	0.753	25.27	4.25	9.8	1.1
$\mathcal{P}$ : $\lambda_l = 2$	0.363	0.557	22.01	4.25	9.8	1.1
$\mathcal{A}$ : No Space Contraction	0.242	0.702	24.59	4.45	10.2	1.1
$\mathcal{A}$ : Hashmap Size $2^{21}$	0.203	0.750	25.23	3.57	10.6	0.52
$\mathcal{A}$ : Hashmap Size $2^{22}$	0.201	0.751	25.34	3.76	10.4	0.73
$\mathcal{A}$ : No View-Dependence	0.205	0.743	24.80	4.13	11.6	1.1
$\mathcal{A}$ : SH Degree 1	0.196	0.753	25.32	4.16	10.4	1.1
$\mathcal{A}$ : 3 SG Lobes	0.197	0.753	25.26	4.58	8.2	1.1
$\mathcal{A}$ : 4 SG Lobes	0.197	0.755	25.26	4.85	7.8	1.1
$\mathcal{A}$ : 1 ASG Lobe	0.200	0.751	25.16	4.69	8.4	1.1
U-Net: 32 Initial Filters	0.206	0.750	25.23	3.84	10.6	1.1
U-Net: 2 Layers	0.207	0.745	24.88	4.23	9.8	1.1
Ours (4M)	0.228	0.733	24.93	3.61	17.5	1.1
Ours (8M)	0.192	0.761	25.53	4.25	9.8	1.1

Table 1. Model ablations computed on the five Mip-NeRF360 outdoor scenes.

per. The reason for these slightly different results lies in the fact that the PyTorch modules we use for the CNN use fast implementations provided by the CUDA Deep Neural Network (cuDNN) API. For floating point numbers, applying associative operations in different order might slightly change the result. Many cuDNN kernels do not ensure identical ordering of operations for every kernel launch to obtain faster computation times. Therefore, the slight differences in results can not be prevented by, e.g., using a fixed random seed – which we are doing anyways.

**Viewpoint-Specific Sampling.** We show the impact of omitting/changing parts of our view-specific re-weighting scheme used when sampling our point probability octree  $\mathcal{P}$ . Without any re-weighting, i.e., sampling the whole scene for every viewpoint, we observe a significant drop in quality of the reconstruction. We further observe that the depth term  $1/d_i$  is more important than the size term  $1/2^{l_i \cdot \lambda_l}$ . While intuition may suggest that the depth term should be squared for a perspective camera model to not overemphasize on the background, we show that this leads to worse results. This is caused by the use of a CNN for hole-filling, as it can easily fill slightly larger holes close to the camera but may not correctly produce clean background with less samples. As a result, placing more samples in the background is beneficial for our method. Along the same lines, we expected  $\lambda_l = 2$  to work best, as it most closely resembles the relative size of leaves when projected onto the 2D image plane. How-

ever, this performs much worse than using  $\lambda_l = 0.5$ , which we think is due to the fact that for better observed regions leaves are more likely to get subdivided. Thus, overemphasizing smaller leaves leads to more samples being placed into well-reconstructed areas, which improves results.

**Appearance Field.** Our appearance field  $\mathcal{A}$  requires querying with positions in  $[0, 1]^3$ , for which we employ spherical contraction [2]. If we instead use the bounding box of  $\mathcal{P}$  for the normalization, we observe a less detailed reconstruction of the well-observed foreground. We further show that using smaller hashmaps for the multi-resolution hash grid has a modest impact on speed and quality but a major impact on model size. Note that the impact on speed mostly depends on the amount of available L2 cache on the used GPU, which was extensively discussed by Müller et al. [17]

**View-Dependence.** We also analyze different representations for capturing view-dependent effects. Not modeling view-dependence at all produces the worst results and using one spherical harmonics (SH) degree less impacts quality only on a few scenes. Alternatives to SH such as spherical Gaussians (SG) [26, 27] and anisotropic spherical Gaussians (ASG) [25] perform worse in our tests. For SG, we encode the lobe width inside the length of the mean vector, which results in 25 and 32 parameters per point with three and four lobes respectively. For ASG, we use a 6D feature vector representing a rotation matrix [29] for each lobe and use the basis vectors of this matrix to model the anisotropic extent. This results in 15 parameters per point for a configuration with a single ASG lobe. We would like to emphasize that we find the ASG representation in particular much more elegant and compact as it uses more than  $2\times$  less features per point and is theoretically less likely to overfit to the training views. However, we observe that it is much harder to optimize using gradient descent compared to SH.

**U-Net.** Lastly, we analyze the impact of using smaller U-Net configurations w.r.t. speed and image quality. Using less initial filters considerably speeds up training and rendering at the cost of image quality. In contrast, using only two layers barely has any impact on speed but reduces image quality significantly.

**Other.** We show effect of using even less samples (4M) and observe much faster training and rendering with only slightly reduced image quality. Here, we also want to highlight that for ablation J, which uses no post-processing but 16M samples instead, the rendering speed is lower than with 8M samples and post-processing enabled. Reducing the number of samples to 8M results in 2.51 hours training time and 10.9 fps. In combination, these two observations suggest finding a method for hole-filling that works with fewer samples without requiring a CNN could lead to a remarkably powerful point-based representation in the future.

### 3. Per-Scene Results

**Mip-NeRF360.** [2] We show per-scene image quality metrics and visual comparisons in Tab. 2 and Fig. 1 respectively. We follow the common practice of using  $4\times$  downsampling for the five outdoor scenes and  $2\times$  downsampling for the four indoor scenes. Consequently, reconstructions and evaluation use images with roughly one megapixel (MP) resolution for outdoor and 1.5 MP resolution for indoor scenes. Importantly, we directly use the downsampled images provided with the dataset as we observe that these are of considerably higher quality compared to the output of commonly used downsampling implementations, e.g., those provided by OpenCV, PIL, and PyTorch/Torchvision. In our tests, the used downsampling algorithm had a non-negligible influence on results. When using the downsampled images Barron et al. [2] obtained using ImageMagick, we notice more accurate recovery of fine details but up to 0.5 db lower PSNR values. Intuitively this makes sense: A better downsampling algorithm leads to more details that *can* be recovered but also more details that *must* be recovered to achieve good metrics. However, we suggest that a detailed study on the influence of using different downsampling algorithms in the context of novel view synthesis could be interesting.

**Tanks and Temples.** [13] We show per-scene image quality metrics and visual comparisons in Tab. 3 and Fig. 2 respectively. We evaluate on all eight scenes of the *intermediate* set and use no downsampling, which results in images that have a resolution of roughly two MP. As the Tanks and Temples dataset does not include camera parameters, we estimate them using COLMAP [20, 21] and use the resulting calibration for all methods/experiments.

**About LPIPS and the VGG Loss.** The most commonly used version of the Learned Perceptual Image Patch Similarity (LPIPS) [28] metric uses a VGG-16 [24] architecture, where 16 quantifies the number of convolution layers in the employed neural network. When inspecting our loss function in combination with our quantitative evaluation, it is easy to assume that our LPIPS results are so good specifically because we use a VGG-based loss term ( $\mathcal{L}_{\text{VGG}}$ ). We highlight that – similar to prior works [7, 15, 19] – our VGG loss uses the VGG-19 [24] architecture. This is a key difference compared to both the original version of the loss by Johnson et al. [8] and the used LPIPS metric. Our ablation study (see Tab. 1) clearly shows that using the VGG loss improves results w.r.t. all three reported quality metrics. To further validate that exploitation of similarities in the VGG-16 and VGG-19 architectures is not a deciding factor for our improved results, we include all three available versions of LPIPS (LPIPS<sub>vgg</sub>/LPIPS<sub>alex</sub>/LPIPS<sub>squeeze</sub>) in the per-scene result tables (Tab. 2 and 3). Our method achieves the best average performance for all versions of LPIPS on both datasets.

LPIPS <sub>vgg</sub> ↓ on Mip-NeRF360 [2]										
Method	Bicycle	Flowers	Garden	Stump	Treehill	Bonsai	Counter	Kitchen	Room	Average
Instant-NGP [17]	0.478	0.466	0.289	0.474	0.496	0.258	0.368	0.249	0.340	0.380
ADOP [19]	0.250	0.361	0.203	0.305	0.354	0.223	0.264	0.221	0.241	0.259
TRIPS [7]	0.223	0.318	0.183	0.309	0.308	0.153	0.206	0.154	0.197	0.213
3DGS [10]	0.229	0.366	0.118	0.244	0.367	0.253	0.262	0.158	0.289	0.254
Zip-NeRF [3]	0.228	0.309	0.127	0.236	0.281	0.196	0.223	0.134	0.238	0.219
Ours	0.161	0.212	0.086	0.173	0.215	0.137	0.184	0.123	0.187	0.164
Ours (16M)	0.171	0.220	0.090	0.190	0.226	0.149	0.187	0.126	0.195	0.173
Ours (8M)	0.183	0.246	0.098	0.210	0.242	0.163	0.212	0.134	0.202	0.188
Ours (pre-ex.)	0.220	0.254	0.113	0.278	0.276	0.156	0.224	0.137	0.200	0.207
SSIM↑ on Mip-NeRF360 [2]										
Method	Bicycle	Flowers	Garden	Stump	Treehill	Bonsai	Counter	Kitchen	Room	Average
Instant-NGP [17]	0.513	0.485	0.706	0.591	0.544	0.904	0.816	0.856	0.870	0.698
ADOP [19]	0.665	0.494	0.741	0.666	0.556	0.818	0.769	0.737	0.839	0.723
TRIPS [7]	0.704	0.502	0.773	0.681	0.591	0.899	0.845	0.850	0.883	0.778
3DGS [10]	0.770	0.602	0.869	0.774	0.637	0.938	0.905	0.921	0.913	0.814
Zip-NeRF [3]	0.772	0.637	0.863	0.788	0.674	0.952	0.905	0.929	0.929	0.828
Ours	0.805	0.668	0.886	0.826	0.702	0.954	0.912	0.932	0.932	0.847
Ours (16M)	0.796	0.655	0.884	0.811	0.695	0.950	0.912	0.933	0.930	0.841
Ours (8M)	0.783	0.645	0.876	0.790	0.688	0.942	0.896	0.927	0.926	0.830
Ours (pre-ex.)	0.731	0.617	0.852	0.711	0.645	0.941	0.880	0.923	0.920	0.802
PSNR↑ on Mip-NeRF360 [2]										
Method	Bicycle	Flowers	Garden	Stump	Treehill	Bonsai	Counter	Kitchen	Room	Average
Instant-NGP [17]	22.21	20.68	25.14	23.47	22.42	30.69	26.69	29.48	29.71	25.61
ADOP [19]	22.60	19.68	24.85	24.18	20.99	24.33	23.09	23.61	25.97	23.54
TRIPS [7]	23.47	19.44	25.38	24.17	22.04	28.71	27.00	27.66	29.07	25.94
3DGS [10]	25.25	21.52	27.41	26.55	22.49	31.98	28.69	30.32	30.63	27.20
Zip-NeRF [3]	25.85	22.33	28.22	27.35	23.95	34.79	29.12	32.36	33.04	28.56
Ours	26.26	22.41	28.52	27.71	24.16	33.89	29.38	31.81	32.89	28.56
Ours (16M)	26.09	22.19	28.34	27.44	24.20	33.29	29.20	31.61	32.91	28.36
Ours (8M)	25.77	22.12	28.09	26.85	24.05	31.90	28.39	30.92	32.36	27.83
Ours (pre-ex.)	24.45	21.60	26.78	25.35	23.41	31.72	26.69	30.47	31.22	26.85
LPIPS <sub>alex</sub> ↓ on Mip-NeRF360 [2]										
Method	Bicycle	Flowers	Garden	Stump	Treehill	Bonsai	Counter	Kitchen	Room	Average
Instant-NGP [17]	0.389	0.385	0.221	0.342	0.468	0.123	0.244	0.137	0.206	0.279
ADOP [19]	0.166	0.280	0.132	0.208	0.278	0.129	0.177	0.151	0.164	0.177
TRIPS [7]	0.138	0.218	0.103	0.205	0.218	0.073	0.124	0.102	0.123	0.133
3DGS [10]	0.165	0.340	0.074	0.151	0.318	0.132	0.161	0.099	0.170	0.179
Zip-NeRF [3]	0.161	0.213	0.085	0.129	0.223	0.092	0.129	0.082	0.133	0.139
Ours	0.124	0.169	0.057	0.112	0.163	0.070	0.123	0.083	0.116	0.113
Ours (16M)	0.137	0.183	0.060	0.131	0.173	0.080	0.125	0.087	0.120	0.122
Ours (8M)	0.152	0.213	0.067	0.161	0.214	0.098	0.153	0.096	0.129	0.143
Ours (pre-ex.)	0.170	0.207	0.078	0.192	0.232	0.090	0.156	0.095	0.128	0.150
LPIPS <sub>squeeze</sub> ↓ on Mip-NeRF360 [2]										
Method	Bicycle	Flowers	Garden	Stump	Treehill	Bonsai	Counter	Kitchen	Room	Average
Instant-NGP [17]	0.260	0.269	0.137	0.254	0.285	0.105	0.185	0.112	0.155	0.196
ADOP [19]	0.104	0.179	0.091	0.139	0.155	0.084	0.111	0.096	0.103	0.112
TRIPS [7]	0.089	0.131	0.072	0.141	0.123	0.049	0.080	0.065	0.078	0.084
3DGS [10]	0.108	0.246	0.051	0.112	0.213	0.127	0.134	0.078	0.148	0.135
Zip-NeRF [3]	0.095	0.136	0.053	0.102	0.117	0.085	0.093	0.061	0.103	0.094
Ours	0.073	0.094	0.037	0.079	0.090	0.050	0.082	0.057	0.076	0.071
Ours (16M)	0.081	0.104	0.040	0.093	0.096	0.058	0.084	0.059	0.079	0.077
Ours (8M)	0.091	0.122	0.045	0.111	0.113	0.069	0.104	0.064	0.086	0.089
Ours (pre-ex.)	0.102	0.119	0.049	0.133	0.120	0.061	0.103	0.063	0.082	0.092

Table 2. **Per-scene image quality metrics** for the Mip-NeRF360 dataset [2] separated into outdoor and indoor scenes. The three best results are highlighted in **green** in descending order of saturation.



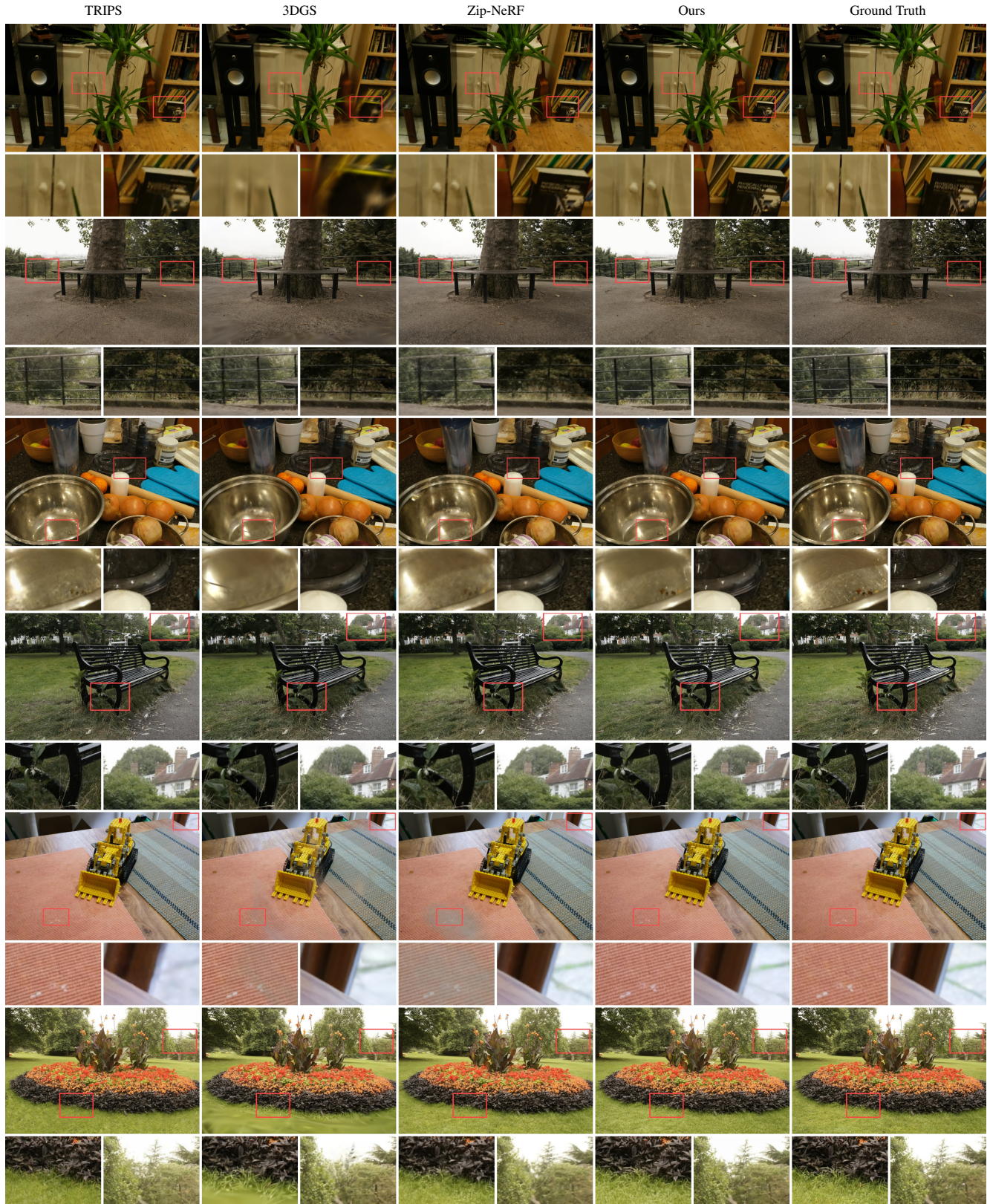


Figure 1. **Visual comparisons** for multiple scenes of the Mip-NeRF360 dataset [2].



LPIPS <sub>vgg</sub> ↓ on Tanks and Temples [13]									
Method	Family	Francis	Horse	Lighthouse	M60	Panther	Playground	Train	Average
Instant-NGP [17]	0.413	0.439	0.458	0.439	0.367	0.355	0.547	0.487	0.438
ADOP [19]	0.203	0.233	0.201	0.242	0.225	0.219	0.231	0.302	0.236
TRIPS [7]	0.176	0.266	0.182	0.277	0.204	0.191	0.222	0.267	0.229
3DGS [10]	0.236	0.344	0.239	0.291	0.244	0.241	0.291	0.320	0.276
Zip-NeRF [3]	0.172	0.270	0.181	0.281	0.212	0.217	0.251	0.279	0.233
Ours	0.115	0.227	0.134	0.217	0.182	0.178	0.172	0.287	0.189
Ours (16M)	0.125	0.241	0.146	0.236	0.239	0.220	0.195	0.356	0.220
Ours (8M)	0.141	0.253	0.159	0.275	0.304	0.267	0.215	0.398	0.252
Ours (pre-ex.)	0.174	0.292	0.207	0.265	0.263	0.232	0.237	0.421	0.261

SSIM↑ on Tanks and Temples [13]									
Method	Family	Francis	Horse	Lighthouse	M60	Panther	Playground	Train	Average
Instant-NGP [17]	0.729	0.812	0.733	0.759	0.810	0.840	0.550	0.666	0.737
ADOP [19]	0.807	0.860	0.842	0.782	0.835	0.859	0.785	0.667	0.802
TRIPS [7]	0.849	0.879	0.871	0.792	0.862	0.884	0.771	0.768	0.831
3DGS [10]	0.871	0.901	0.889	0.834	0.901	0.910	0.834	0.791	0.866
Zip-NeRF [3]	0.893	0.918	0.909	0.835	0.905	0.908	0.846	0.813	0.878
Ours	0.905	0.915	0.914	0.833	0.903	0.912	0.878	0.769	0.878
Ours (16M)	0.902	0.906	0.909	0.822	0.884	0.896	0.860	0.718	0.862
Ours (8M)	0.895	0.906	0.905	0.804	0.848	0.870	0.852	0.687	0.846
Ours (pre-ex.)	0.864	0.879	0.877	0.808	0.864	0.884	0.826	0.664	0.833

PSNR↑ on Tanks and Temples [13]									
Method	Family	Francis	Horse	Lighthouse	M60	Panther	Playground	Train	Average
Instant-NGP [17]	21.47	23.96	18.45	21.17	24.87	26.45	18.52	19.72	21.82
ADOP [19]	24.29	21.92	23.87	18.28	23.66	25.47	22.58	15.66	21.69
TRIPS [7]	24.03	20.06	23.45	18.09	25.52	27.73	24.10	18.79	22.62
3DGS [10]	25.05	27.64	24.18	21.76	27.82	28.35	25.65	21.69	25.27
Zip-NeRF [3]	28.05	29.55	27.67	22.31	28.86	28.84	26.62	22.10	26.75
Ours	28.52	26.85	27.73	21.83	27.77	28.82	26.81	19.12	25.93
Ours (16M)	28.05	26.61	27.10	21.40	26.55	27.55	25.61	18.11	25.12
Ours (8M)	27.92	26.09	26.86	21.08	24.56	25.99	26.04	17.81	24.54
Ours (pre-ex.)	26.20	24.23	24.53	20.96	25.22	26.43	25.15	16.98	23.71

LPIPS <sub>alex</sub> ↓ on Tanks and Temples [13]									
Method	Family	Francis	Horse	Lighthouse	M60	Panther	Playground	Train	Average
Instant-NGP [17]	0.337	0.295	0.387	0.313	0.259	0.245	0.475	0.393	0.338
ADOP [19]	0.142	0.122	0.138	0.180	0.148	0.128	0.157	0.236	0.158
TRIPS [7]	0.114	0.138	0.115	0.187	0.124	0.101	0.133	0.181	0.140
3DGS [10]	0.152	0.187	0.144	0.197	0.165	0.146	0.231	0.237	0.182
Zip-NeRF [3]	0.105	0.125	0.099	0.174	0.125	0.118	0.175	0.171	0.136
Ours	0.073	0.113	0.072	0.152	0.136	0.114	0.145	0.241	0.131
Ours (16M)	0.087	0.126	0.085	0.176	0.192	0.160	0.174	0.323	0.165
Ours (8M)	0.105	0.139	0.099	0.222	0.266	0.213	0.196	0.371	0.201
Ours (pre-ex.)	0.128	0.182	0.148	0.200	0.205	0.151	0.200	0.385	0.200

LPIPS <sub>squeeze</sub> ↓ on Tanks and Temples [13]									
Method	Family	Francis	Horse	Lighthouse	M60	Panther	Playground	Train	Average
Instant-NGP [17]	0.251	0.224	0.302	0.245	0.207	0.194	0.351	0.310	0.261
ADOP [19]	0.090	0.085	0.091	0.118	0.100	0.088	0.099	0.162	0.106
TRIPS [7]	0.076	0.134	0.080	0.144	0.086	0.071	0.086	0.136	0.105
3DGS [10]	0.120	0.180	0.122	0.164	0.137	0.127	0.175	0.196	0.153
Zip-NeRF [3]	0.074	0.104	0.073	0.131	0.094	0.091	0.117	0.126	0.101
Ours	0.050	0.093	0.052	0.112	0.091	0.080	0.083	0.166	0.091
Ours (16M)	0.058	0.102	0.060	0.126	0.133	0.113	0.101	0.222	0.114
Ours (8M)	0.071	0.111	0.071	0.159	0.183	0.148	0.116	0.263	0.140
Ours (pre-ex.)	0.083	0.132	0.102	0.139	0.141	0.110	0.114	0.273	0.137

Table 3. **Per-scene image quality metrics** for the Tanks and Temples dataset [13]. The three best results are highlighted in **green** in descending order of saturation.



Figure 2. **Visual comparisons** for multiple scenes of the Tanks and Temples dataset [13].



## 4. Perceptual Experiment

The objective of our perceptual experiment is to compare, from an observer’s perspective, the novel view results of our method and Zip-NeRF [3] (the method achieving the best quality metrics’ scores among the compared-against methods). This necessitates the presentation of a multitude of stimuli, the distinctions between which are frequently nuanced. It is also crucial to acknowledge that the quality of the results we seek to quantify cannot be accurately represented on a linear scale [9]. This underscores the limitations of ranking methods in this context. Accordingly, the paired comparisons technique was selected for use in this experiment. This technique requires participants to view two images simultaneously and then select the one that more closely aligns with the task question. This resulted in a two-alternatives forced-choice (2AFC) preference task, in which each result was compared to the corresponding other method’s results for the same image.

### 4.1. Experimental Design

**Stimuli.** Consistently with the analysis of the main paper, the stimuli were selected from the well established datasets Mip-NeRF360 [2] and Tanks and Temples [13]. In order to maintain the number of trials of the experiment under a reasonable number that allows for a single participant to perform a complete test while maintaining the necessary level of attention, we selected a set of three novel views per scene, for all scenes in both considered datasets. The criteria for selection aimed to cover the maximum possible diversity in image attributes (i.e., background/foreground objects, recurring texture, scene composition, lighting, faces/people, lines/clear edges, view angle/focus depth, textual elements) for a given scene. For those scenes where the attributes were not sufficiently sampled, we extended the number of considered novel views (i.e., *Room*: 4, *Lighthouse*: 4, *M60*: 5). We then gathered the corresponding results of both Zip-NeRF [3] and our method, for a total of 55 paired comparisons (see Fig. 3).

**Participants.** A total of 17 participants were included in the study, with an age range of 22 to 55 years and a gender distribution of 4 females, 1 non-available, and 12 males. All participants reported normal or corrected-to-normal vision. Additionally, the majority of participants indicated an intermediate level of proficiency in computer graphics and image processing, as measured on a scale of 1 to 4, with 1 indicating no experience and 4 indicating advanced proficiency.

**Procedure.** The participants performed the experiment in-situ one at a time. Before starting the experiment, each participant signed an informed consent form. They were then given an explanation on the experimental task and provided with the options of asking any questions. However, they were neither informed of the research question behind the experiment, nor of the nature (e.g., captured vs. generated)

of the images they were presented with. The participant was asked to sit roughly 50 cm in front of a 27” UHD monitor (at a resolution of  $3840 \times 2160$ ) in a semi-dark room. After the experimenter gathered the participant’s demographic data, the participant was presented with a screen describing once again the instructions. Before the experimenter left the room, the participants were given another chance to ask any further questions. Each trial started by presenting two images side by side over a black background. Participants were not explicitly instructed to focus their attention on anything in particular, but to just select which of the two images they found more appealing. The participants were able to enter their answers by clicking on the desired image without having any restriction regarding the time to make their choice; once a response was entered, the next trial started. Both the order of the pairs of stimuli, and their position on the screen (left vs. right) was fully randomized, with each participant receiving a different random order. The experiment was controlled by Psychophysics Toolbox Version 3.0.15 (PTB-3) [4, 12], and the mean time to complete it was of 21 minutes.

### 4.2. Analysis

To assess not only the efficacy of each method, but also the consistency of participant responses, we employed the linked-paired comparison design [6]. Accordingly, the methods are ranked according to the number of times they are preferred over the other method. The total number of votes a pair of stimuli received is displayed in Fig. 3, while the votes over datasets and over all stimuli are displayed in Tab. 4. These results favorably rank our method over Zip-NeRF in all scenarios. In order to ascertain the true meaning of this ranking, a significance test of the score differences is performed. In order to achieve this objective, it is necessary to identify a value, designated as  $R'$ , which represents the minimum variance-normalized range of scores within each group. This necessitates the computation of  $R'$  such that  $P[R \geq R'] \leq \alpha$ , where the confidence level, set at 0.01, is represented by  $\alpha$ . Subsequently, in accordance with the methodology proposed by Herbert A. David [6], we can derive  $R'$  from the following equation:

$$P(W_{t,\alpha} \geq \frac{2R' - 0.5}{\sqrt{mt}}), \quad (2)$$

where  $m$  is the number of participants,  $t$  is the number of methods to be compared, and  $W_{t,\alpha}$  has been previously tabulated by Pearson and Hartley [18]. In our case,  $W_{2,0.01} = 3.645$ , which leads to  $R'_{Exp} = 10.87691$  for our experiment.

Since the score differences between all ranked groups exceed the aforementioned  $R'$ , we can conclude that they are all statistically significant. Thus, the ranking creates two distinguishable groups within our experimental context.

Dataset	Method	#Votes	Ranking
Mip-NeRF360 [2]	Ours	331	1 <sup>st</sup>
	Zip-NeRF [3]	145	2 <sup>nd</sup>
Tanks & Temples [13]	Ours	318	1 <sup>st</sup>
	Zip-NeRF [3]	141	2 <sup>nd</sup>
All Stimuli	Ours	649	1 <sup>st</sup>
	Zip-NeRF [3]	286	2 <sup>nd</sup>

Table 4. Perceptual ranking of the compared methods in our experiment with 17 participants. All rankings are statistically significant.

## References

- [1] Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. Mip-NeRF: A multiscale representation for anti-aliasing neural radiance fields. In *Int. Conf. Comput. Vis.*, pages 5835–5844, 2021. 1
- [2] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-NeRF 360: Unbounded anti-aliased neural radiance fields. In *IEEE/CVF Conf. Comput. Vis. Pattern Recog.*, pages 5460–5469, 2022. 1, 3, 4, 5, 6, 9, 10, 14
- [3] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Zip-NeRF: Anti-aliased grid-based neural radiance fields. In *Int. Conf. Comput. Vis.*, pages 19640–19648, 2023. 2, 5, 7, 9, 10, 14
- [4] David H. Brainard. The psychophysics toolbox. *Spat. Vis.*, 10(4):433–436, 1997. 9
- [5] Lu Chi, Borui Jiang, and Yadong Mu. Fast Fourier convolution. In *Adv. Neural Inform. Process. Syst.*, pages 4479–4488. Curran Associates, Inc., 2020. 3
- [6] Herbert Aron David. *The method of paired comparisons*. Charles Griffin, London, 1963. 9
- [7] Linus Franke, Darius Rückert, Laura Fink, and Marc Stamminger. TRIPS: Trilinear point splatting for real-time radiance field rendering. *Comput. Graph. Forum*, 43(2), 2024. 2, 4, 5, 7
- [8] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *Eur. Conf. Comput. Vis.*, pages 694–711. Springer International Publishing, 2016. 4
- [9] M. G. Kendall and B. Babington Smith. On the method of paired comparisons. *Biometrika*, 31(3-4):324–345, 1940. 9
- [10] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkuehler, and George Drettakis. 3D Gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4), 2023. 1, 2, 5, 7
- [11] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Int. Conf. Learn. Represent.*, 2015. 1
- [12] Mario Kleiner, David Brainard, Denis Pelli, Allen Ingling, Richard Murray, Christopher Broussard, and Frans Cornelissen. What’s new in psychtoolbox-3? *Perception*, 36(14): 1–16, 2007. 9
- [13] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and Temples: Benchmarking large-scale scene reconstruction. *ACM Trans. Graph.*, 36(4), 2017. 1, 4, 7, 8, 9, 10, 14
- [14] Christoph Lassner and Michael Zollhöfer. Pulsar: Efficient sphere-based neural rendering. In *IEEE/CVF Conf. Comput. Vis. Pattern Recog.*, pages 1440–1449, 2021. 2
- [15] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *IEEE/CVF Conf. Comput. Vis. Pattern Recog.*, pages 105–114, 2017. 4
- [16] Thomas Müller. tiny-cuda-nn, 2021. 1, 2
- [17] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4), 2022. 2, 4, 5, 7
- [18] E. S. Pearson and H.O. Hartley. *Biometrika Tables for Statisticians*. Cambridge University Press, 3 edition, 1966. 9
- [19] Darius Rückert, Linus Franke, and Marc Stamminger. ADOP: Approximate differentiable one-pixel point rendering. *ACM Trans. Graph.*, 41(4), 2022. 1, 2, 4, 5, 7
- [20] Johannes L. Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *IEEE/CVF Conf. Comput. Vis. Pattern Recog.*, pages 4104–4113, 2016. 1, 4
- [21] Johannes L. Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise view selection for unstructured multi-view stereo. In *Eur. Conf. Comput. Vis.*, pages 501–518. Springer International Publishing, 2016. 1, 4
- [22] Markus Schütz, Bernhard Kerbl, and Michael Wimmer. Software rasterization of 2 billion points in real time. *Proc. ACM Comput. Graph. Interact. Tech.*, 5(3), 2022. 2
- [23] Markus Schütz, Bernhard Kerbl, and Michael Wimmer. Rendering point clouds with compute shaders and vertex order optimization. *Comput. Graph. Forum*, 40(4):115–126, 2021. 2
- [24] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Int. Conf. Learn. Represent.*, 2015. 4
- [25] Kun Xu, Wei-Lun Sun, Zhao Dong, Dan-Yong Zhao, Run-Dong Wu, and Shi-Min Hu. Anisotropic spherical gaussians. *ACM Trans. Graph.*, 32(6), 2013. 4
- [26] Lior Yariv, Peter Hedman, Christian Reiser, Dor Verbin, Pratul P. Srinivasan, Richard Szeliski, Jonathan T. Barron, and Ben Mildenhall. BakedSDF: Meshing neural SDFs for real-time view synthesis. In *ACM SIGGRAPH Conference Papers*. Association for Computing Machinery, 2023. 2, 4
- [27] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. PlenOctrees for real-time rendering of neural radiance fields. In *Int. Conf. Comput. Vis.*, pages 5732–5741, 2021. 4
- [28] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *IEEE/CVF Conf. Comput. Vis. Pattern Recog.*, pages 586–595, 2018. 4
- [29] Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. On the continuity of rotation representations in neural networks. In *IEEE/CVF Conf. Comput. Vis. Pattern Recog.*, pages 5738–5746, 2019. 4



Ours

Zip-NeRF



*bicycle-01*

Total Votes: Ours - **15** Zip-NeRF - 2



*bicycle-03*

Total Votes: Ours - **15** Zip-NeRF - 2



*bicycle-14*

Total Votes: Ours - **14** Zip-NeRF - 3



*bonsai-01*

Total Votes: Ours - **15** Zip-NeRF - 3



*bonsai-03*

Total Votes: Ours - **12** Zip-NeRF - 5



*bonsai-15*

Total Votes: Ours - **4** Zip-NeRF - **13**



*counter-01*

Total Votes: Ours - **12** Zip-NeRF - 5

Ours

Zip-NeRF



*family-04*

Total Votes: Ours - **16** Zip-NeRF - 1



*family-08*

Total Votes: Ours - **17** Zip-NeRF - 0



*family-17*

Total Votes: Ours - **4** Zip-NeRF - **13**



*francis-05*

Total Votes: Ours - **9** Zip-NeRF - 8



*francis-23*

Total Votes: Ours - **16** Zip-NeRF - 1



*francis-37*

Total Votes: Ours - **17** Zip-NeRF - 0



*playground-03*

Total Votes: Ours - **16** Zip-NeRF - 1



Ours



Zip-NeRF



counter-14

Total Votes: Ours - 10 Zip-NeRF - 7



counter-19

Total Votes: Ours - 3 Zip-NeRF - 14



stump-00

Total Votes: Ours - 7 Zip-NeRF - 10



stump-04

Total Votes: Ours - 8 Zip-NeRF - 9



stump-08

Total Votes: Ours - 13 Zip-NeRF - 4



flowers-01

Total Votes: Ours - 16 Zip-NeRF - 1



flowers-13

Total Votes: Ours - 15 Zip-NeRF - 2

Ours



Zip-NeRF



playground-11

Total Votes: Ours - 12 Zip-NeRF - 5



playground-38

Total Votes: Ours - 16 Zip-NeRF - 1



m60-02

Total Votes: Ours - 1 Zip-NeRF - 16



m60-17

Total Votes: Ours - 3 Zip-NeRF - 14



m60-27

Total Votes: Ours - 7 Zip-NeRF - 10



m60-29

Total Votes: Ours - 3 Zip-NeRF - 14



m60-37

Total Votes: Ours - 6 Zip-NeRF - 11



Ours



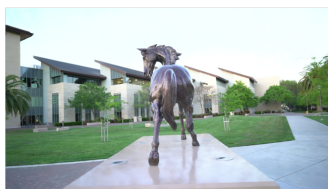
Zip-NeRF



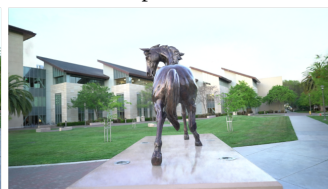
flowers-20

Total Votes: Ours - 15 Zip-NeRF - 2

Ours



Zip-NeRF



horse-02

Total Votes: Ours - 13 Zip-NeRF - 4



treehill-00

Total Votes: Ours - 17 Zip-NeRF - 0



horse-13

Total Votes: Ours - 16 Zip-NeRF - 1



treehill-07

Total Votes: Ours - 16 Zip-NeRF - 1



horse-14

Total Votes: Ours - 15 Zip-NeRF - 2



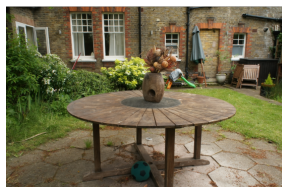
treehill-14

Total Votes: Ours - 12 Zip-NeRF - 5



train-05

Total Votes: Ours - 11 Zip-NeRF - 6



garden-02

Total Votes: Ours - 13 Zip-NeRF - 4



train-19

Total Votes: Ours - 8 Zip-NeRF - 9



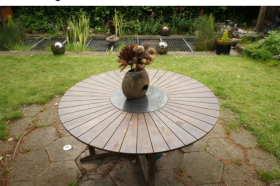
garden-09

Total Votes: Ours - 13 Zip-NeRF - 4



train-21

Total Votes: Ours - 14 Zip-NeRF - 3



garden-15

Total Votes: Ours - 13 Zip-NeRF - 4





Figure 3. **Experimental Stimuli.** All images generated with Zip-NeRF [3] and our method used as stimuli for our perceptual experiment. The two columns of the left belong to the results on the Mip-NeRF360 dataset [2], while the two on the right are from the Tanks and Temples dataset [13]. Under each pair of stimuli, we report the scene name, frame number, and total votes gathered per method (max=17).