IBRNet: Learning Multi-View Image-Based Rendering —Supplementary Material—

Qianqian Wang[1,2] Zhicheng Wang[1] Kyle Genova[1,3] Pratul Srinivasan[1] Howard Zhou[1]
Jonathan T. Barron[1] Ricardo Martin-Brualla[1] Noah Snavely[1,2] Thomas Funkhouser[1,3]

[1]Google Research [2]Cornell Tech, Cornell University [3]Princeton University

A. Additional implementation details

Feature extraction network architecture. To render a target view, our system takes a set of source views and extracts their features using a network with shared weights. We implement the feature extraction network using a U-Net-like architecture, where the encoder is adapted from ResNet34 [1] as implemented in PyTorch [6]. We replace all Batch Normalization [2] with Instance Normalization [7] as in [3], and remove max-pooling and use instead strided convolutions. Our network is fully convolutional and accepts input images of variable size. We take a single image of size $640 \times 480 \times 3$ as an example input and present a detailed network architecture in Tab. 1. Our code and model will be made available.

Input (**id**: dimension)	Layer	Output (**id**: dimension)
0: $640 \times 480 \times 3$	7×7 Conv, 64, stride 2	**1**: $320 \times 240 \times 64$
1: $320 \times 240 \times 64$	Residual Block 1	**2**: $160 \times 120 \times 64$
2: $160 \times 120 \times 64$	Residual Block 2	**3**: $80 \times 60 \times 128$
3: $80 \times 60 \times 128$	Residual Block 3	**4**: $40 \times 30 \times 256$
5: $40 \times 30 \times 256$	3×3 Upconv, 128, factor 2	**6**: $80 \times 60 \times 128$
[**3**, **6**]: $80 \times 60 \times 256$	3×3 Conv, 128	**7**: $80 \times 60 \times 128$
7: $80 \times 60 \times 128$	3×3 Upconv, 64, factor 2	**8**: $160 \times 120 \times 64$
[**2**, **8**]: $160 \times 120 \times 128$	3×3 Conv, 64	**9**: $160 \times 120 \times 64$
9: $160 \times 120 \times 64$	1×1 Conv, 64	**Out**: $160 \times 120 \times 64$

Table 1: **Feature extraction network architecture**. 'Conv" stands for a sequence of operations: convolution, rectified linear units (ReLU) and Instance Normalization [7]. "Upconv" stands for a bilinear upsampling with certain factor, followed by a "Conv" operation with stride 1. "$[\cdot, \cdot]$" represents channel-wise concatenation of two feature maps. The residual blocks have similar structures to those in the original ResNet34 [1] design, except that the first residual block has stride equal to 2 and all Batch Normalization layers are replaced with Instance Normalization layers. The output 64-dimensional feature map will be split into two feature maps of 32 dimension, which are then used as inputs to the coarse and fine IBRNet, respectively.

IBRNet network architecture. Fig. 1 shows the detailed network architecture of IBRNet and the process of predicting the volume density and color at a single 5D location. To obtain the density at each 5D location, we first aggregate the multi-view features drawn from source views to obtain a density feature that encodes the density information for this point. We then aggregate the density information of all samples on the ray to enable visibility reasoning for better density prediction. We implement this ray-wise operation using our proposed module called "ray transformer". Specifically, we first apply positional encoding [8] to all density features on the ray, so that the network is aware of the spatial ordering for samples on the ray. We then use a single multi-head self-attention [8] layer to incorporate long-range contextual information. Separately, to obtain the view-dependent color, we predict a set of blending weights to blend the image colors drawn from source views. IBRNet is invariant to permutations of source views, and supports a variable number of source views as well as samples on a ray.

Training details. At training time, we sample points in space and project them in the source views to fetch the corresponding colors and image features. However, the projected pixel for a sample may be located outside of the image plane. In this case, we discount this source view for this sample. If a point is not projected into the image plane of any source view, we set the volume density of this point to be zero. If there are less than 3 samples on a ray that have valid density values, we ignore this ray in the loss function during training.

For pre-training, We train on eight V100 GPUs with a batch size of 3,200 to 9,600 rays depending on image resolution and the number of source views. Within each batch, we sample rays from eight different scenes. Within each scene, we sample rays randomly from a single image for training. Pre-training takes about a day to finish. For fine-tuning, the time needed is scene-dependent. It takes us about 6 hours on a single V100 GPU to achieve the reported performance for each of the *Real Forward-Facing* [4] scenes. The fine-tuning stage may be accelerated with better hyperparameters for optimization.

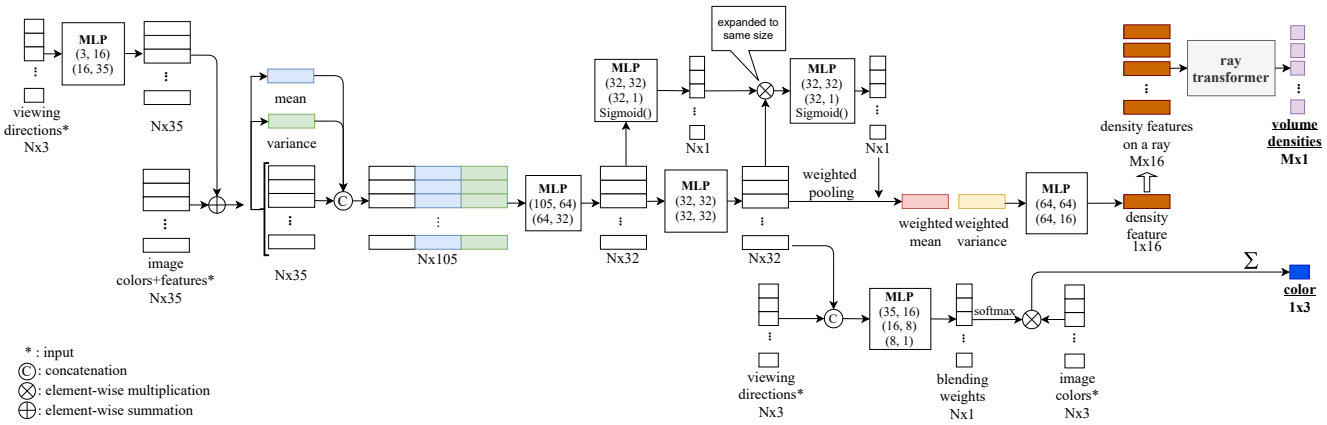

Figure 1: **IBRNet architecture**. N denotes the number of source views and M is the number of samples on a ray. For each "MLP" box, (i, j) represents a linear layer with input dimension i and output dimension j. ELU is used between each two adjacent linear layers as the non-linear activation function. When "MLP" takes in a stack of feature vectors (i.e., feature vectors from all source views), the MLP is applied to each feature vector with shared weights. "weighted pooling" computes the weighted mean and variance of the $N \times 32$ feature vector using the learned $N \times 1$ weight vector. The ray transformer module contains a single multi-head self-attention layer with the number of heads set to 4. Viewing directions shown in the figures are relative viewing directions, i.e., the viewing direction of the query ray relative to the viewing directions from source views.

Figure 2: **Qualitative comparison of our model trained with and without ray transformer.** The first and second columns show the results of our pretrained model without and with the ray transformer module, respectively. The last column shows the ground truth images. Without ray transformer, the synthesized images exhibit severe "black hole" artifacts especially near occlusion boundaries where the network fails to infer surface locations correctly. Ray transformer eliminates such artifacts by enabling more informed density prediction.

B. Additional qualitative results

Ray transformer. We provide qualitative comparison of our model trained with and without ray transformer in Fig. 2. Fig. 2 shows that our proposed ray transformer significantly improves the synthesis quality especially for challenging regions (e.g., near occlusion boundaries).

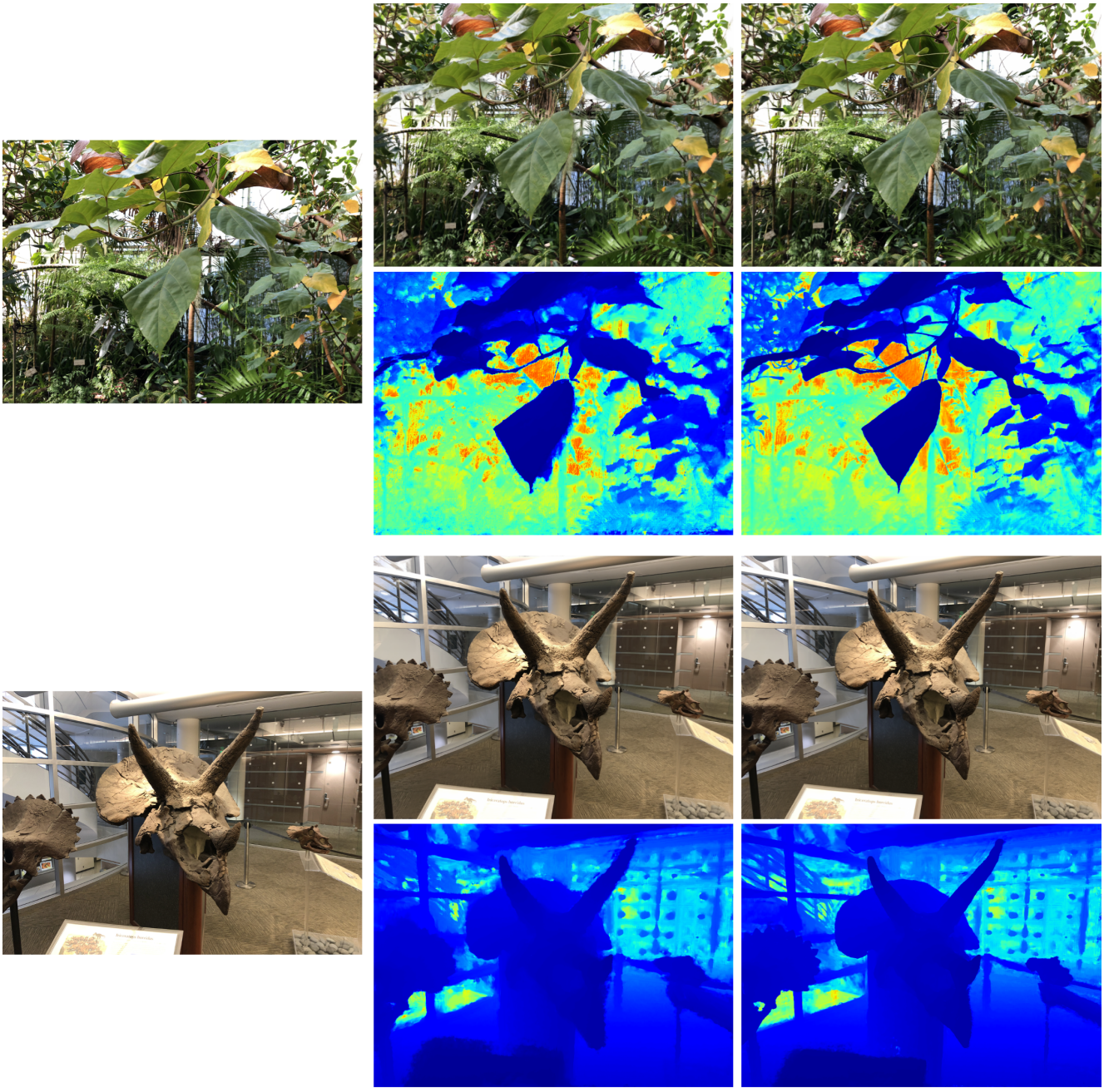

Figure 3: **Geometry Visualization**. We visualize the proxy geometry and synthesized images generated by our pretrained and fine-tuned models for two scenes *leaves* and *horns*. For each scene, the first column shows the ground truth image. The second column shows the results, i.e., synthesized image (top) and depth map (bottom), using our pretrained model. the last column shows the results of our model after finetuned on each scene.

Geometry Visualization. We visualize the proxy geometry by accumulating the predicted density values on each ray. Fig. 3 shows that our pretrained model produces reasonable proxy geometry, and the quality of the synthesized images and the underlying geometry improves when our model is fine-tuned.

Qualitative results on *Realistic Synthetic* 360° *[5].* We provide qualitative results of our pretrained and finetuned model on *Realistic Synthetic* 360° *[5]* in Fig. 4. The last column in Fig. 4 shows a challenging scenario where our method does not work well.

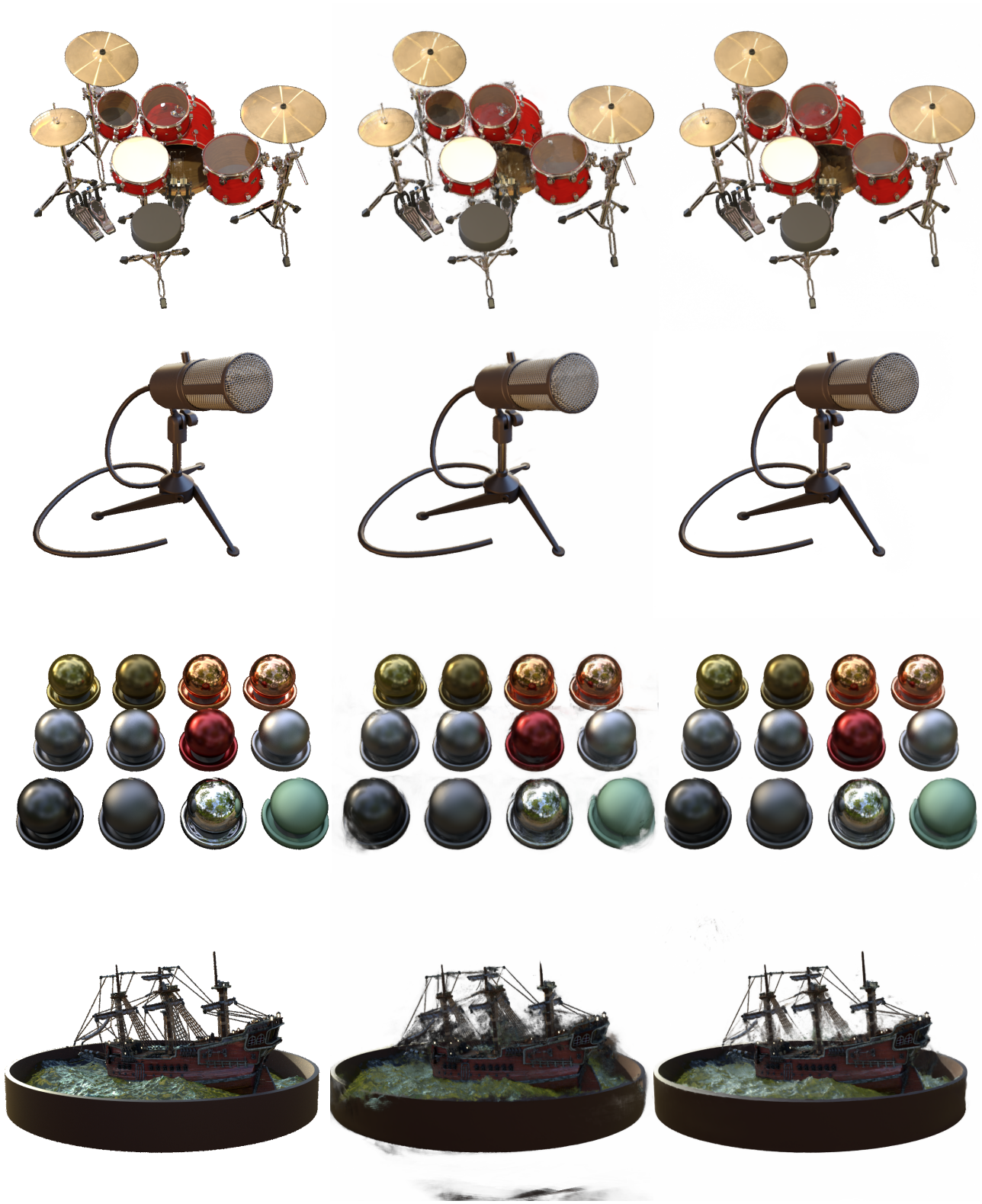

Figure 4: **Qualitative results on *Realistic Synthetic* 360° *[5].*** The first column shows the ground truth images. The second column shows the synthesized images without per-scene fine-tuning. The last column shows the synthesized images with per-scene fine-tuning. The last row is a failure case due to sparse source views and complex geometry.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CVPR*, 2016. 1

[2] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ICML*, 2015. 1

[3] Keyang Luo, Tao Guan, Lili Ju, Yuesong Wang, Zhuo Chen, and Yawei Luo. Attention-aware multi-view stereo. *CVPR*, 2020. 1

[4] Ben Mildenhall, Pratul P. Srinivasan, Rodrigo Ortiz-Cayon, Nima K. Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *ACM TOG*, 2019. 1

[5] 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. *ECCV*, 2020. 3, 4

[6] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. *NIPS-W*, 2017. 1

[7] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016. 1

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *NeurIPS*, 2017. 1