Compositional Neural Scene Representations for Shadow Inference: Supplementary Material

JONATHAN GRANSKOG, ETH Zurich & NVIDIA
FABRICE ROUSSELLE, NVIDIA
MARIOS PAPAS, DisneyResearch|Studios
JAN NOVÁK, NVIDIA

ACM Reference Format:
Jonathan Granskog, Fabrice Rousselle, Marios Papas, and Jan Novák. 2020. Compositional Neural Scene Representations for Shadow Inference: Supplementary Material. *ACM Trans. Graph.* 39, 4, Article 135 (July 2020), 7 pages. https://doi.org/10.1145/3386569.3392475

1 IMAGE GENERATORS

Numerous neural image generators have been proposed in the past. Some of them generate the novel image by transforming buffers rendered from the novel view, e.g. denoising and shading U-nets [Chaitanya et al. 2017; Nalbach et al. 2017]. Other generators rely only on the neural scene representation, either directly [Eslami et al. 2018], or by extracting a 2D slice of it via ray marching [Sitzmann et al. 2019]. Our preferred approach lies somewhere in the middle, i.e. we want to leverage the neural scene representation *and* a cheap-to-compute G-buffer from the novel view.

We tested three, previously published generators adjusted to consume $\mathbf{c}_v$, $\mathbf{g}_v$, and $\mathbf{r}$ as inputs. Table 1 lists their numbers of trainable parameters and the training times. Each generator is illustrated in Figure 1, detailed in the next section, and analyzed in Section 1.2.

1.1 Implementation Details

GQN generator. The GQN generator [Eslami et al. 2018] is a probabilistic model consisting of prior and conditional densities that are parameterized by the output of deep convolutional networks. Each network is based on the recurrent convolutional DRAW model [Gregor et al. 2016], which constructs the conditional density sequentially using a convolutional LSTM core. Each instance of the core receives the camera parameters $\mathbf{c}_v$, the G-buffer $\mathbf{g}_v$, the scene representation $\mathbf{r}$, and all the other inputs (e.g. the state of the LSTM core) that Eslami et al. [2018] utilized.

We implemented the GQN generator as specified by Eslami et al. [2018]. All our experiments used twelve LSTM cells without weight sharing, which was their best-performing architecture, operating on $16 \times 16 \times 128$ states. The number of latent variables input into each cell was $16 \times 16 \times 3$. The G-buffer $\mathbf{g}_v$ is downsampled to 16×16 and concatenated to the other inputs of the cells. We also utilized the proposed simulated annealing for the standard deviation of the

Authors' addresses: Jonathan Granskog, ETH Zurich & NVIDIA, jgranskog@nvidia.com; Fabrice Rousselle, NVIDIA, frousselle@nvidia.com; Marios Papas, DisneyResearch|Studios, marios.papas@disneyresearch.com; Jan Novák, NVIDIA, jnovak@nvidia.com.

0730-0301/2020/7-ART135
https://doi.org/10.1145/3386569.3392475

Table 1. Parameters of the encoder and the different generators. The training times are for the complete model, i.e. the encoder with one of the generators, using NVIDIA Tesla V100.

	# of parameters	Training time
Pool encoder	2 000 644	
GQN generator	147 735 199	10 days
U-net generator	80 596 099	8.5 days
Pixel generator	4 199 939	8.5 days

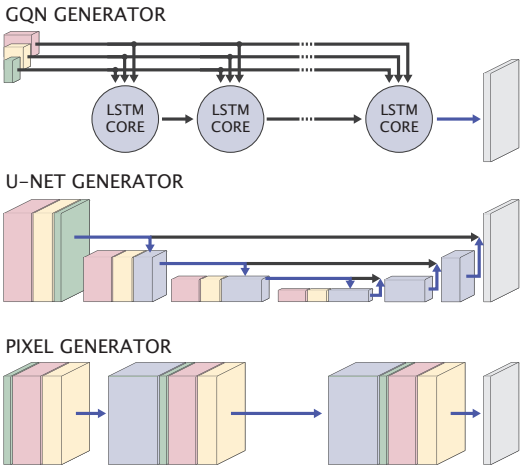

Fig. 1. Illustration of tested image generators. The green and yellow boxes represent tensors holding the G-buffer $\mathbf{g}_v$ and spatially duplicated camera parameters $\mathbf{c}_v$ for the novel view. Red boxes represent the (spatially duplicated) neural scene representation $\mathbf{r}$. Black arrows represent flow of data and blue arrows symbolize convolutions (with pooling and upsampling in the case of the U-net).

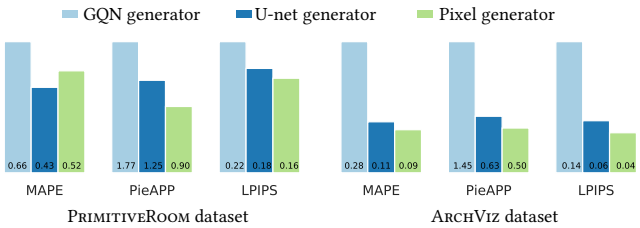

Fig. 2. Average metrics of different generators on 16 selected scenes rendered with a 64×64 resolution (lower means better).

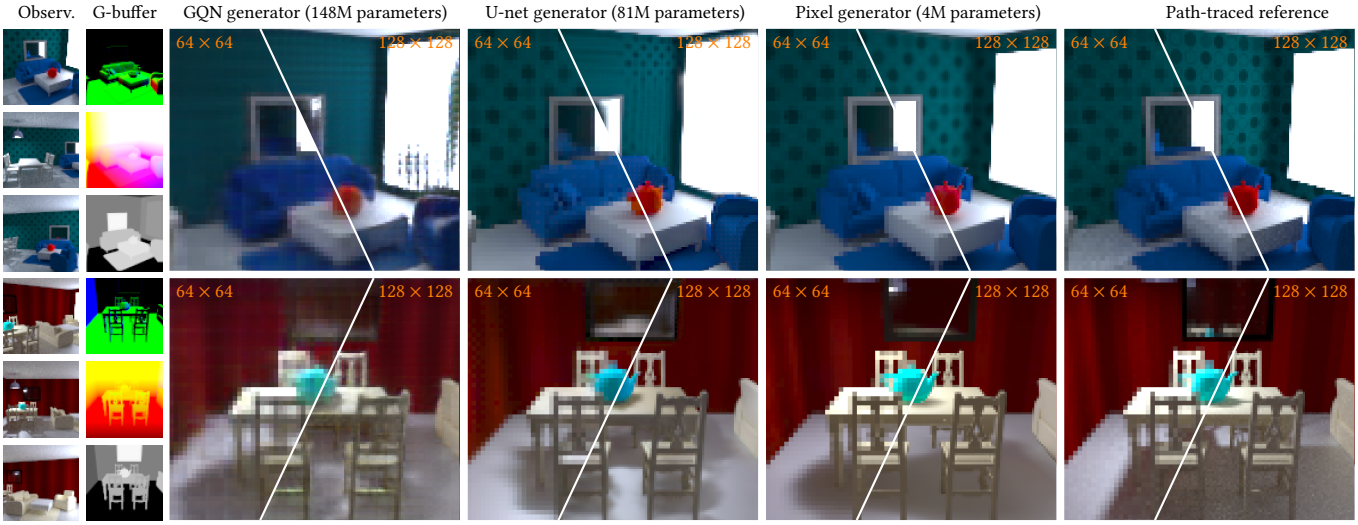

Fig. 3. Comparison of three image generators that consume (i) a neural scene representation extracted from three observations (left column; we only show the beauty image here but each observation comes with a G-buffer), and (ii) a G-buffer (second left column) containing *geometric* information of directly visible surfaces (position, normal, object ID) from the novel view of the scene. Each generator was trained with its own encoder, end-to-end, using 64×64 observations. We compare the quality of generated images with G-buffers rendered at resolutions 64×64 and 128×128; this defines the resolution of the final image. Despite being smallest (labels on top report numbers of trainable parameters), the pixel generator delivers the best results overall. The most noticeable artifacts appear on shadows and in reflections.

output normal distribution but not for the learning rate. We chose to keep the learning rate equal for all generators. We optimized the model using the proposed ELBO-based loss [Eslami et al. 2018].

U-net generator. Convolutional U-nets [Ronneberger et al. 2015] with auxiliary feature buffers [Chaitanya et al. 2017; Nalbach et al. 2017] have been applied to many image-to-image translation tasks. The main feature of a convolutional approach is the ability to extract information from a screen-space neighborhood around each pixel. The U-net architecture, specifically, consists of encoding and decoding stages that generate a very large receptive field, while still allowing activations and gradients to bypass the information bottleneck via skip connections.

The U-net consists of 7 scales and features 3×3 convolutions and ReLU activation functions. In the encoder, each level ends with 2×2 max pooling with stride 2. The first two levels output 128 and 256 depth channels respectively whereas the following five levels each use 512 depth channels. In the decoder, each level ends with a 4×4 deconvolution with stride 2. The channel counts at individual levels mirror the encoder. We use padding to ensure that the spatial resolution can be halved and doubled.

The scene representation $\mathbf{r}$ and camera parameters $\mathbf{c}_v$ are input at each level of the encoder; we concatenate the two vectors along the depth dimension and duplicate them spatially to match the resolution of the level.

Pixel generator. The pixel generator [Sitzmann et al. 2019] is a straightforward image-to-image translation model that processes each pixel independently with a multi-layer perceptron. The advantage of the pixel generator is the multi-view consistency and better handling of arbitrary output resolutions, both of which stem from the reliance on information in a single pixel only. This contrasts with convolutional approaches where the pixel neighborhood, and thus the inferred color, generally vary across views and resolutions.

We implemented the pixel generator as 1x1 convolutions organized into ten hidden layers, each with 512 output channels. The representation $\mathbf{r}$ and camera parameters $\mathbf{c}_v$ are concatenated and duplicated spatially to create a $[w \times h \times (\|\mathbf{r}\| + \|\mathbf{c}_v\|)]$ tensor, where w and h are the width and height of $\mathbf{g}_v$, respectively. This tensor is concatenated to the G-buffer $\mathbf{g}_v$ and to the output of each hidden layer.

1.2 Performance Analysis

In Figure 2, we report the average performance of individual generators measured using MAPE, PieApp, and LPIPS on a testing dataset consisting of sixteen challenging hand-picked scenes. We observe that the pixel generator is the top performing generator according to all metrics followed closely by the U-net generator. The GQN generator performed the worst; we would like to note that this approach has not been designed for image-to-image translation, which could explain its worse performance. Interestingly, the performance *anti-correlates* with the numbers of trainable parameters.

In Figure 3, we show two representative results obtained with the three generators. Each generator was trained with its own encoder end-to-end on 64×64 images from the ArchViz dataset. The figure shows images of two, previously unobserved scenes. The observations (left column) are computed for three random positions of the camera.

The GQN generator suffers from splotchy artifacts and blurry edges. It appears it did not learn to properly utilize the information in the G-buffer, which is to be expected, as it was not designed for

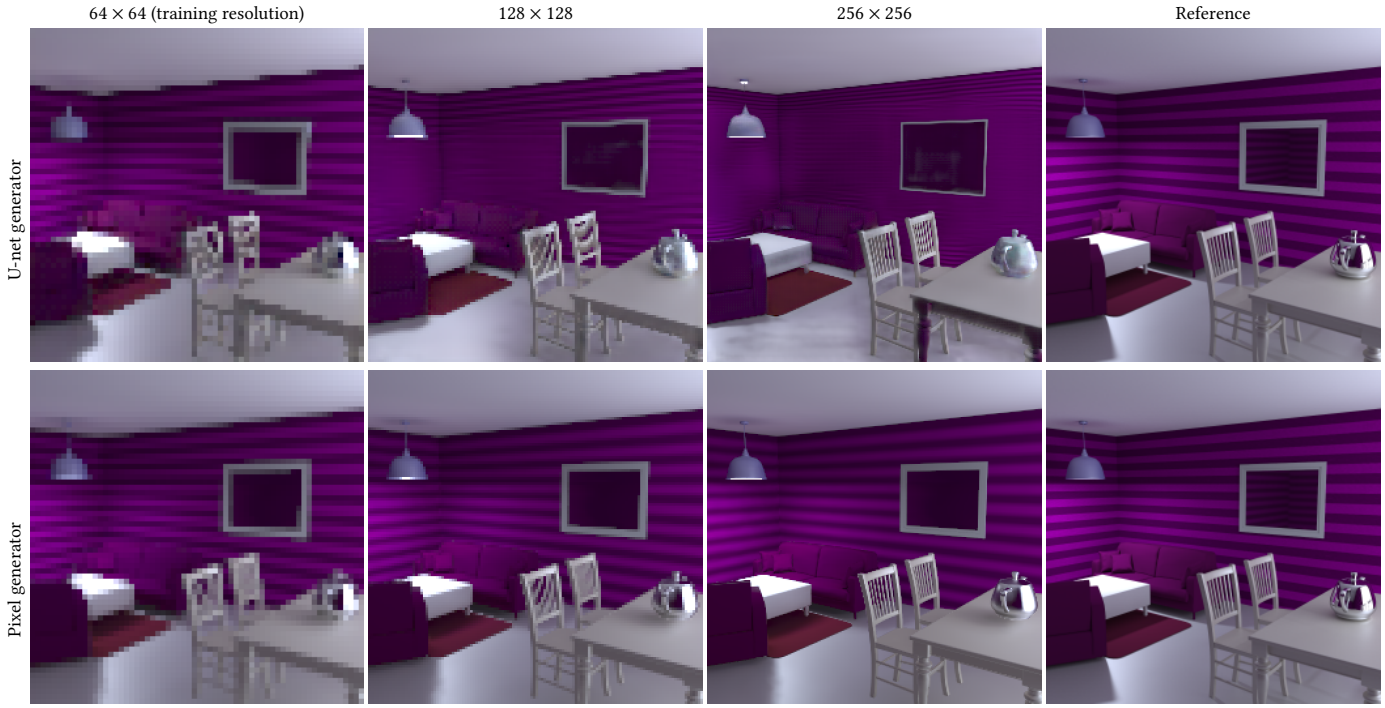

Fig. 4. Comparison of image synthesis at different target resolutions. Using the U-net and the pixel generators at the same resolution as the model was trained on (64 × 64, left column) gives results of similar quality. When the target resolution increases, the U-net generator suffers from shrinking of shadows and various other artifacts. The pixel generator scales more gracefully, with the main artifact being blurry texture detail.

image-to-image translations, but rather to generate an image given camera parameters only. The quality could potentially be further improved at the cost of longer training times.

Scaling to high resolutions. The U-net generator and the pixel generator produce generally sharp images. The most notable difference, however, surfaces when comparing outputs rendered at higher resolution than the models were trained on. We tested each generator with G-buffer $\mathbf{g}_v$ at two resolutions: 64 × 64 and 128 × 128—the resolution of the G-buffer defines the resolution of the generated image. In contrast to the U-net and GQN generators, which are both convolutional architectures, the pixel generator handles well the higher resolution (128 × 128) despite being trained with 64 × 64 G-buffers only.

As stated by Sitzmann et al. [2019], a key feature of the pixel generator is its resolution independence. This is further analyzed in Figure 4 by comparing the convolutional U-net generator and the pixel generator. The shadows and texture details synthesized by the U-net generator shrink as the resolution increases beyond what the model was trained on. In contrast, the pixel generator scales to higher resolutions gracefully. The main visual artifact is the blurry appearance of textures. In our case, this can be fixed by providing material (texture) information in the G-buffer, as shown in Figure 7.

Impact of G-buffer. In Figure 5, we show the performance of the pixel generator when no G-buffer is provided and the generator must rely on the neural representation for all information about the scene. The quality is significantly worse than in other figures.

2 DATASETS

Here we describe the procedural recipes for generating our datasets. Examples of both datasets are shown in Figure 6.

PrimitiveRoom dataset. This scene consists of a room with randomly colored walls and a set of primitive objects that have different materials. The scene is illuminated by a single spherical light source; its position is randomized but intensity is the same in all configurations. For geometry, we randomize the XZ-positions, Y-rotations, and presence of objects while avoiding intersecting geometry. The materials of objects are one of three different types: ideal mirror, glossy specular (using the GGX distribution [Walter et al. 2007] with roughness 0.5) and Lambertian diffuse with randomized hue. The walls are diffuse with a random hue.

ArchViz dataset. This dataset mimics the scenario rendering interior scenes for architectural visualization. Each scene is separated into a living room area and a dining area. The presence and spatial location of these areas is randomized. The dining area consists of a table and up to four chairs that have the same material. The sofa, the armchair, and the carpet are always diffuse with the diffuse albedo being randomly selected from a continuous color map (the sofa and the armchair have identical material). In addition to this, we randomly place a teapot on either the table, floor, or on the sofa table. The teapot can have any material. The diffuse walls feature a randomly selected pattern (one out of four exemplars) and a randomized albedo hue.

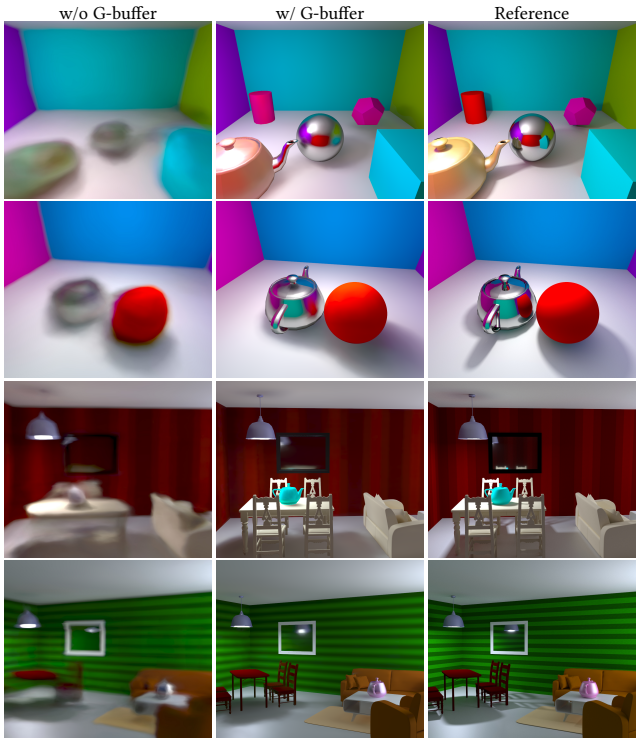

Fig. 5. Pixel generator networks can be trained without a G-buffer using a mapping from -1 to 1 across the image dimensions. However, training becomes slower and results are blurry compared to a network that receives G-buffers.

There are two light sources that illuminate the room; a ceiling light fixture aiming downwards with a randomized XZ-position, and a large emissive quad that mimics a window. Both light sources have their intensities and tints randomized between a light orange and a light blue color. To investigate quality of reflections, we also include a mirror that is placed at a random position on the back wall of the room.

HDR images. In contrast to prior works on scene representations, we train the generators to produce high dynamic-range images. We apply a $\log(\mathbf{i}+1)$ transform to each HDR input image $\mathbf{i}$ and perform the computation in log space (including the loss evaluation). The final HDR image is obtained by reverse-transforming the predicted image.

3 ADDITIONAL RESULTS

In Figure 7, we provide additional results of utilizing the neural model for synthesizing indirect illumination only. One of the objectionable artifacts are the poorly handled reflections in the teapot and the mirror on the back wall. We show how these can be improved at the cost of providing the reflection direction in the G-buffer, or tracing an extra reflection ray in Figure 8.

REFERENCES

Chakravarty R. Alla Chaitanya, Anton S. Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila. 2017. Interactive Reconstruction of Monte Carlo Image Sequences Using a Recurrent Denoising Autoencoder. *ACM Trans. Graph.* 36, 4, Article Article 98 (July 2017), 12 pages. https://doi.org/10.1145/3072959.3073601

S. M. Ali Eslami, Danilo Jimenez Rezende, Frederic Besse, Fabio Viola, Ari S. Morcos, Marta Garnelo, Avraham Ruderman, Andrei A. Rusu, Ivo Danihelka, Karol Gregor, David P. Reichert, Lars Buesing, Theophane Weber, Oriol Vinyals, Dan Rosenbaum, Neil Rabinowitz, Helen King, Chloe Hillier, Matt Botvinick, Daan Wierstra, Koray Kavukcuoglu, and Demis Hassabis. 2018. Neural scene representation and rendering. *Science* 360, 6394 (2018), 1204–1210. https://doi.org/10.1126/science.aar6170

Karol Gregor, Frederic Besse, Danilo Jimenez Rezende, Ivo Danihelka, and Daan Wierstra. 2016. Towards Conceptual Compression. In *Advances in Neural Information Processing Systems 29*. Curran Associates, Inc., 3549–3557.

O. Nalbach, E. Arabadzhiyska, D. Mehta, H.-P. Seidel, and T. Ritschel. 2017. Deep Shading: Convolutional Neural Networks for Screen Space Shading. *Comput. Graph. Forum* 36, 4 (July 2017), 65–78. https://doi.org/10.1111/cgf.13225

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Springer International Publishing, Cham, 234–241.

Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. 2019. Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 1119–1130.

Bruce Walter, Stephen R. Marschner, Hongsong Li, and Kenneth E. Torrance. 2007. Microfacet Models for Refraction through Rough Surfaces. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques (EGSR'07)*. Eurographics Association, Goslar, DEU, 195–206.

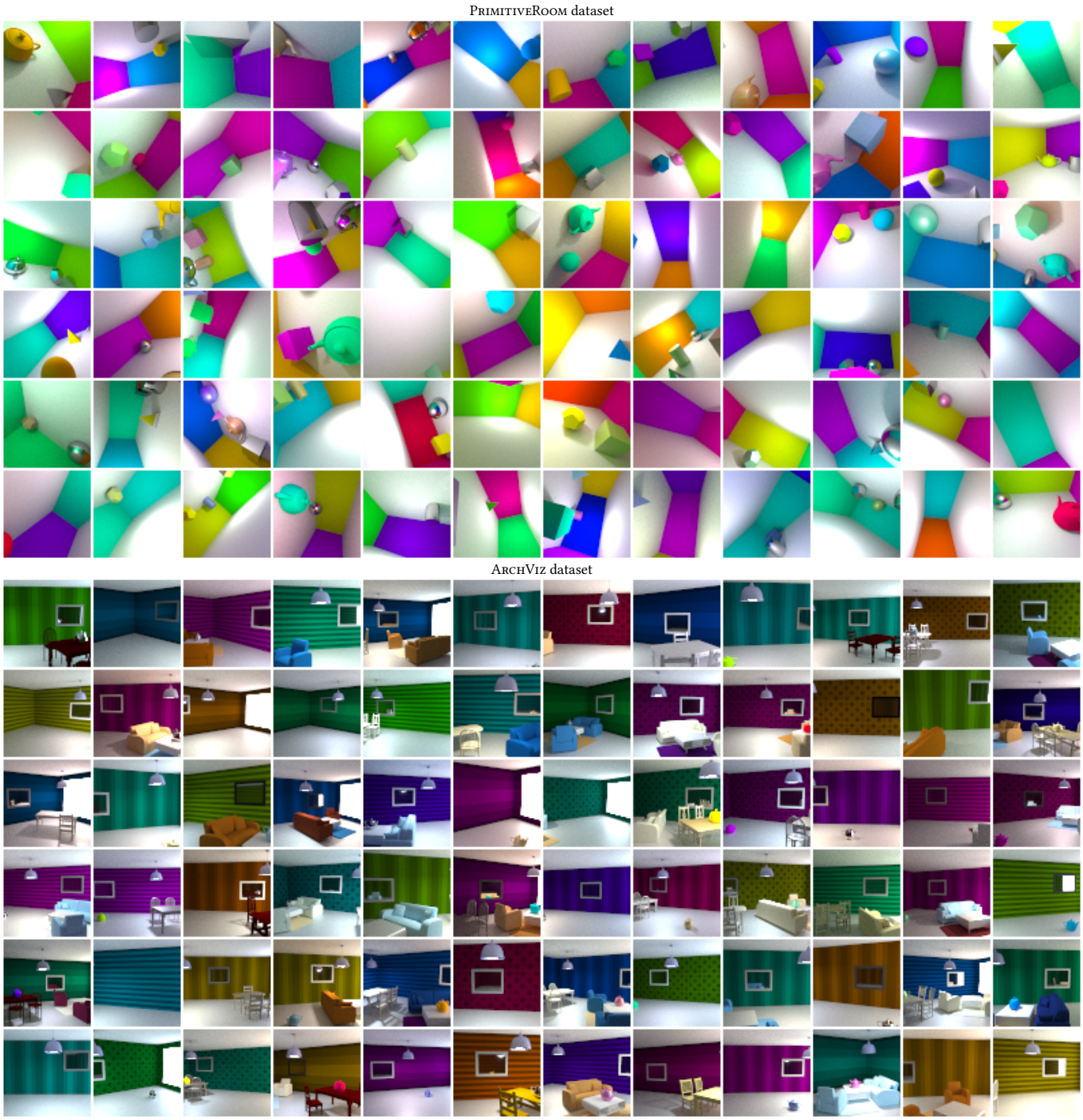

Fig. 6. Random scenes from the PrimitiveRoom and the ArchViz datasets.

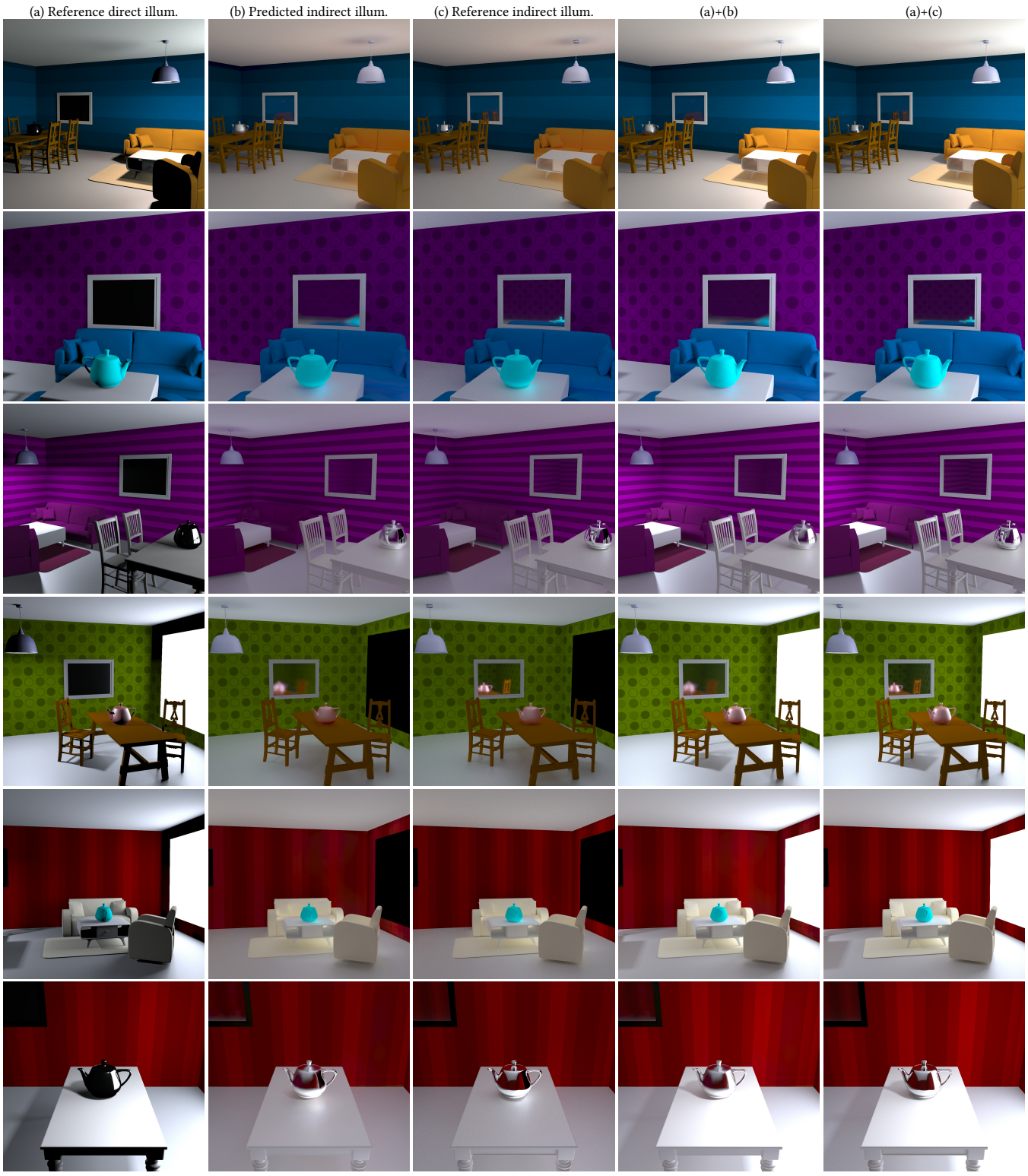

Fig. 7. Additional examples of predicting indirect illumination (b) only, and combining it with ray-traced direct illumination (a). The neural model utilizes a pixel generator that consumes (i) the neural scene representation and (ii) a G-buffer of the novel view with geometry and material information, and the direct-illumination image.

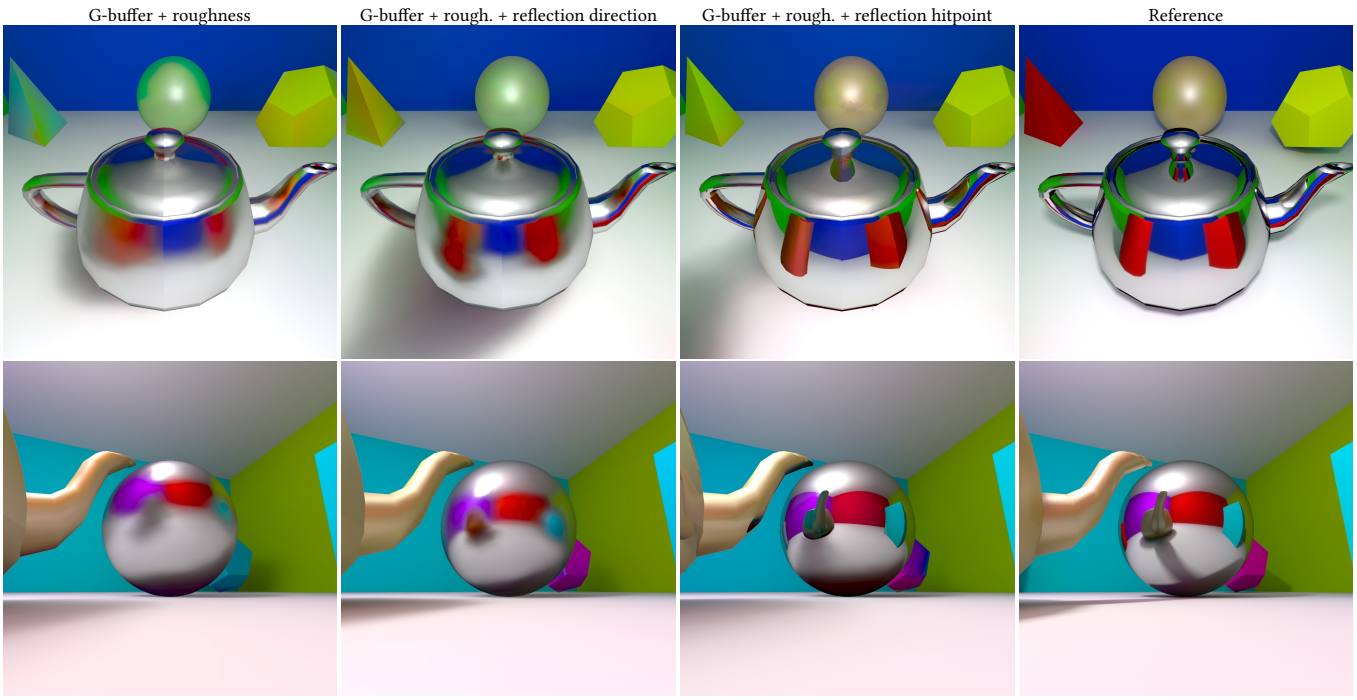

Fig. 8. Reflections can be improved by providing additional information about the reflection ray in the G-buffer. In addition to the geometry G-buffer, the generator in the first column receives also a roughness buffer. In the second column, we show results when including also the direction of the specular-reflection ray. In the third column, we further improve the results by providing the position and the normal of the surface point that the reflection ray hits; this requires tracing the reflection ray.