Compositional Neural Scene Representations for Shading Inference

ANONYMOUS AUTHOR(S)
SUBMISSION ID: PAPERS_525

(a) Observations (b) G-buffer (c) Ray-traced direct illumination **(d) Inferred indirect illumination** (e) Direct + indirect illumination

Fig. 1. Our system integrates data-driven synthesis into a standard path tracing framework. Given a set of observations (a), we extract a view-independent scene representation using a neural encoder. A neural image generator then takes the scene representation together with geometric features (b) of a novel view and synthesizes indirect illumination (d). We then add it to ray-traced direct illumination (c) to obtain a high-quality, global illumination image.

We present a technique for adaptively partitioning neural scene representations. Our method disentangles lighting, material, and geometric information yielding a scene representation that preserves the orthogonality of these components, improves interpretability of the model, and allows compositing new scenes by mixing components of existing ones. The proposed adaptive partitioning respects the uneven entropy of individual components and thereby reduces the performance penalty induced by static partitioning. It also permits compressing the scene representation, which lowers its memory footprint and reduces the computation cost. Furthermore, the partitioned representation enables an in-depth analysis of existing image generators. We compare the flow of information through individual partitions, and by contrasting it to additional inputs (G-buffer), which our modified generators consume, we are able to identify the roots of undesired visual artifacts and propose one possible solution to remedy the poor performance. We also demonstrate that neural scene representations can handle scenes with moderately complex geometry and materials, if used in a complementary manner to classical representations of the scene.

CCS Concepts: • **Computing methodologies** → **Rendering**; **Neural networks**.

ACM Reference Format:
Anonymous Author(s). 2018. Compositional Neural Scene Representations for Shading Inference. *ACM Trans. Graph.* 9, 4, Article 39 (August 2018), 15 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Data-driven realistic image synthesis has recently achieved a number of notable breakthroughs, such as rendering realistic human faces [Karras et al. 2017], or high-quality relighting of photographs [Philip et al. 2019]. Remarkable achievements have been demonstrated also in data-driven simulation of light transport, where neural networks predict various radiative quantities [Hermosilla et al. 2018; Kallweit et al. 2017; Nalbach et al. 2017; Ren et al. 2013] or improve their unbiased estimation [Müller et al. 2019; Zheng and Zwicker 2019]. Common to all these is the utilization of neural networks to perform the task in its entirety, all at once. The black-box nature, however, hinders interpretability, generalization, and makes further development less intuitive.

An alternative approach to performing the synthesis at once is to introduce an intermediate neural scene representation [Eslami et al. 2018; Kulkarni et al. 2015; Sitzmann et al. 2019], by breaking the rendering task into: (i) extracting a learned scene representation, and (ii) using it to render an image. Employing the intermediate (latent) scene representation allows enforcing certain behaviors upon the model, e.g. ensuring consistency of images rendered from different views of the scene. It also presents an opportunity for increasing robustness, improving generalization, and accelerating training by injecting physically-based constraints to regularize the model; we present one step in that direction. While the scene complexity and rendering quality of these approaches may appear limited, especially when compared to state-of-the-art neural simulators and renderers, we show in Figure 1 that the quality can be increased by having the neural representation merely complement traditional rendering inputs, rather than relying on it exclusively. The key ingredient here is the end-to-end training, which makes the neural representation focus on complementary information.

We present three main contributions in this work. First, we extend the works of Eslami et al. [2018] and Sitzmann et al. [2019] with mechanisms to disentangle material, lighting, and geometric content of the scene. We do not prescribe a specific encoding between

0730-0301/2018/8-ART39
https://doi.org/10.1145/nnnnnnn.nnnnnnn

images and the latent scene representation (this shall be extracted from data), but we introduce additional constraints to *adaptively partition* the latent scene representation. The adaptive partitioning ameliorates interpretability of the representation, permits tracing the roots of poor rendering quality, and mitigates the performance penalty induced by static partitioning [Kulkarni et al. 2015]. It also permits compressing the neural scene representation and thereby accelerating the image generation.

The adaptive partitioning enables our second contribution: an analysis of generated images with respect to the extracted scene representation and the input observations. The analysis yields insight into the data-driven synthesis and guides the choice of the neural architectures and design of techniques to remedy visual artifacts and poor performance.

Our third contribution is the demonstration that disentangled neural scene representations capture sufficient information for synthesizing a wide range of illumination effects, such as shadows, color-bleeding, view-dependent highlights, and glossy reflections. We achieve this by balancing standard rendering inputs and the learned scene representation. We also demonstrate that the compositional aspect of the partitioned scene representation allows creating novel configurations, e.g. by swapping lighting and material information between scenes.

Scope. We focus on applications where the scene is known, and available as a 3D model. We draw inspiration, and architectures, from works where such model is not available; the focus is on extracting it [Eslami et al. 2018; Sitzmann et al. 2019], but we leverage these architectures to complement the strengths of classical renderers. Specifically, we use a classical renderer to obtain information that is exact and cheap, and we employ the neural renderer to approximate only the costly shading, e.g. due to global illumination.

2 RELATED WORK

Next, we review recent methods utilizing neural networks for rendering, discuss techniques for representation learning, disentanglement, and attribution, and point out hybrid renderers that bear similarities to our approach.

Neural scene representations. A recurring approach to render a scene using neural networks is to start off from a voxel grid representation, along with a camera pose and light position. This has been applied to simple shading models [Nguyen-Phuoc et al. 2018] or to a single object with global illumination [Rematas and Ferrari 2019]. Our method can also render these effects, but is not restricted to a single object, and can scale to more complex scenes as it does not incur the memory overhead of a voxel grid. Lombardi et al. [2019] reduce the memory requirements and artifacts of voxel grids, while Sitzmann et al. [2019] replace it altogether with a learned 3D scene representation obtained using a differentiable ray-marcher.

Tatarchenko et al. [2015] compress a single image into a representation for novel view synthesis. Rendering of novel views is also achieved by Thies et al. [2019] who learn neural textures of objects. However, all lighting and shading is "baked" and cannot be edited easily. Our method is designed to avoid such baking. We build upon generative query networks (GQNs) [Eslami et al. 2018], which are composed of two core components: an encoder that extracts a scene representation from multiple observations, and a generator that synthesizes the novel view. The lengthy optimization procedure of GQNs (weeks of training) led to the development of more efficient losses [Nguyen-Ha et al. 2019] and exploitation of geometric information [Tobin et al. 2019]. We take a different approach: we accelerate image generation by leveraging geometric features of the novel view (G-buffer) and using the pixel generator [Sitzmann et al. 2019] architecture instead of the original probabilistic approach utilizing LSTM cores [Eslami et al. 2018]. This significantly accelerates training (days instead of weeks) and improves accuracy.

Disentanglement and user control. GQNs [Eslami et al. 2018] extract a monolithic scene represention that does not lend itself to analysis or direct control. Multiple approaches have been explored to introduce user control, such as latent-space transformations [Nguyen-Phuoc et al. 2019; Olszewski et al. 2019] or disentanglement of the latent variables [Chen et al. 2016; Higgins et al. 2017]. Our work builds upon a disentanglement technique proposed by Kulkarni et al. [2015]: they modify a single aspect of the scene per training batch and average the activations and adjust gradients partitions encoding other (static) aspects. We employ this technique to partition lighting, material, and geometry in the representation, and extend it to obtain adaptively partitioned, compressed scene representations.

Interpretability and Attribution. An underlying goal of our work is to gain insights into data-driven image synthesis by attributing the outputs to individual components of the model. Du et al. [2019] classify interpretability as either global or local. Global interpretation, which relates a model behavior to the network parameters and structure, is facilitated by our proposed adaptive partition scheme. Local interpretability, which relates a model output to its inputs, can be achieved through pertubation-based [Wagner et al. 2019; Zeiler and Fergus 2013] and gradient-based [Ancona et al. 2018; Shrikumar et al. 2016] methods; we use the later type. While these are commonly used in the context of classification, we perform attribution on a pixel basis, that is, we attribute each pixel value to the extracted scene representation, which in turn can be related to the input observations to get a complete view of data flow.

Hybrid Renderers. Augmenting traditional renderers using data-driven techniques has a long and successful history. Early successes in data-driven lighting [Debevec 1998] and material modeling [Debevec et al. 2000; Matusik 2003] have had a profound impact on rendering pipelines, both for real-time and offline applications. Recent work has turned to deep learning techniques, for instance to model subsurface scattering [Vicini et al. 2019], approximate multiple scattering in clouds [Kallweit et al. 2017], or approximate light transport in screen space [Nalbach et al. 2017]. Similarly to these approaches, we propose to tightly integrate data-driven image synthesis in the rendering pipeline, but we do so in a more general setting allowing the model to synthesize arbitrary (residual) components of light transport, which complement existing renderers techniques. We show examples where the image generator translates a G-buffer into a shaded image, or a G-buffer w/ direct illumination into an indirectly lit image only.

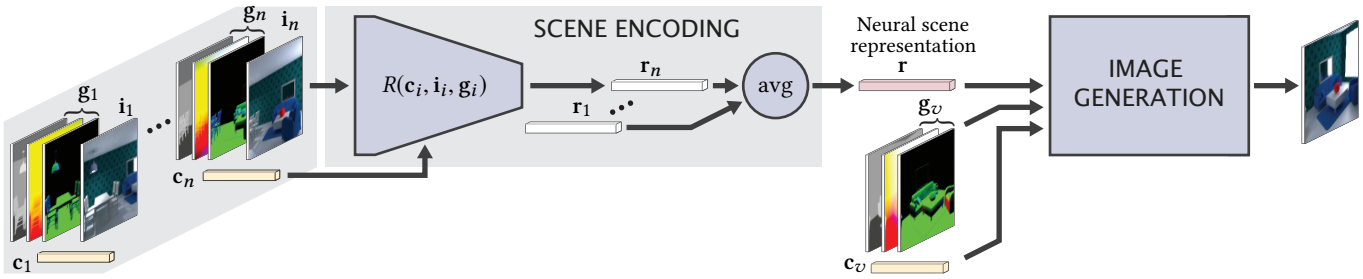


Fig. 2. In order to generate an image, we utilize a view-independent neural scene representation $\mathbf{r}$ and a G-buffer rendered using a classical renderer from a novel view v. The scene representation is extracted from n (high-quality) observations of the scene. Each i-th observation, which consists of a beauty image $\mathbf{i}_i$, G-buffer $\mathbf{g}_i$, and camera parameters $\mathbf{c}_i$, is processed using a neural encoder to produce a vector $\mathbf{r}_i$. These vectors are averaged across all observations to obtain the neural scene representation $\mathbf{r}$ (pink box). For a novel view v, the representation $\mathbf{r}$, the camera parameters $\mathbf{c}_v$, and G-buffer $\mathbf{g}_v$ are fed into an image-generating neural network to obtain an image of the scene from v.

3 BACKGROUND AND MODEL OVERVIEW

Our goal is to leverage neural scene representations to improve traditional forward-rendering tasks, in which a virtual 3D scene is rendered into a 2D image. We draw inspiration from prior works, where one neural network—scene encoder—builds a scene representation that is then passed to another network—image generator—that synthesizes a novel image of the scene [Eslami et al. 2018; Kulkarni et al. 2015; Sitzmann et al. 2019].

The individual components of our rendering system (see Figure 2) are based on previously published techniques; we do not claim novelty in their design. Our original contributions pertain to how these components are combined and optimized. More specifically, we note that the neural scene representation, if constructed as proposed by e.g. Eslami et al. [2018], lacks interpretability. Tracing down the roots of undesired artifacts is thus difficult.

In order to improve interpretability, we adjust the optimization in Section 4 to enforce (adaptive) partitioning of the scene representation, dedicating one partition to each of lighting, materials, and (macro) geometry of the scene. The partitioned scene representation allows attributing poor performance to absence of one (or more) of these properties due to their underrepresentation in the training set, or due to a loss function that is insensitive to them. Once identified, these issues can be mitigated by adjusting the optimization. To that end, we propose to add *auxiliary image generators* in Section 6.1 that rebalance the individual components of the scene representation and thereby improve the synthesis quality.

The rest of this section discusses the neural components of the system and the inputs, datasets, and loss functions used throughout the experiments.

3.1 Scene encoder

The task of the scene encoder is to extract relevant information from n scene observations $\{(\mathbf{c}_i, \mathbf{i}_i, \mathbf{g}_i)\}_{i=1..n}$ and compress it into a *neural scene representation* $\mathbf{r}$. In our case, observation i consists of a view matrix identifying the camera view $\mathbf{c}_i$, a beauty image $\mathbf{i}_i$ that contains all light transport we wish to reproduce later, and a G-buffer $\mathbf{g}_i$ that contains geometry features of visible surfaces obtained as a byproduct of rendering the beauty image.

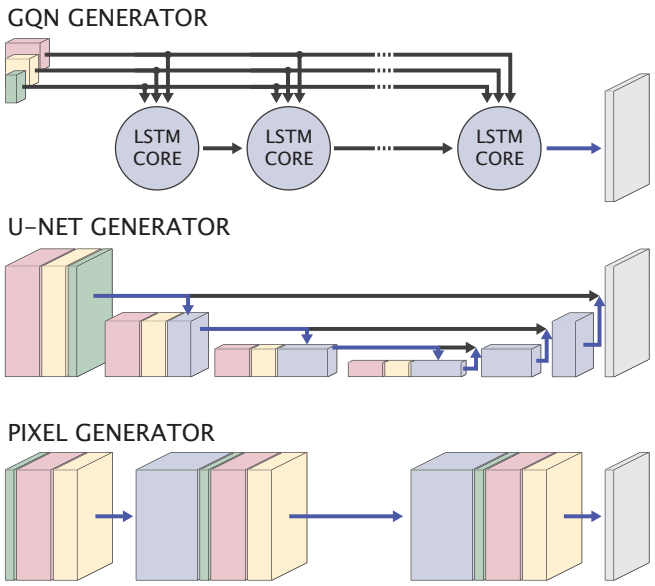


Fig. 3. Illustration of image generators compared in Figure 6. The green and yellow boxes represent tensors holding the G-buffer $\mathbf{g}_v$ and spatially duplicated camera parameters $\mathbf{c}_v$ for the novel view. Red boxes represent the (spatially duplicated) neural scene representation $\mathbf{r}$. Black arrows represent flow of data and blue arrows symbolize convolutions (with pooling and upsampling in the case of the U-net).

The main component of the encoder is a convolutional neural network R, which extracts a partial, high-dimensional scene representation $\mathbf{r}_i$ for each observation i: $\mathbf{r}_i = R(\mathbf{c}_i, \mathbf{i}_i, \mathbf{g}_i)$. We use the *pool* architecture [Eslami et al. 2018] for the encoding network R. The network takes a tensor of concatenated image buffers $(\mathbf{i}_i, \mathbf{g}_i)$ and processes them with strided convolutions. The viewpoint parameters $\mathbf{c}_i$ are shaped into a tensor and concatenated to the output tensor of one of the convolutional layers in the middle of the network. The last component of the network is a pooling layer that outputs a $1 \times 1 \times k$ vector $\mathbf{r}_i$ where k is the number of desired values in representations; see [Eslami et al. 2018] for details.

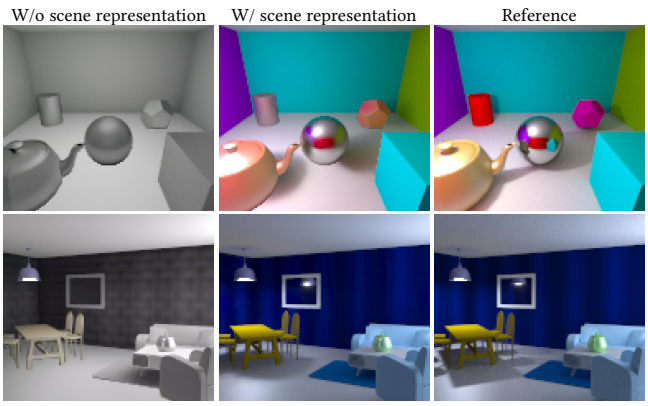


Fig. 4. In most of our experiments, the neural scene representation provides all material and lighting information, and information about geometry outside the camera frustum. Without it (left) the image generator can still synthesize images using the information in the G-buffer, but it learns to produce average appearances and lighting in the training set.

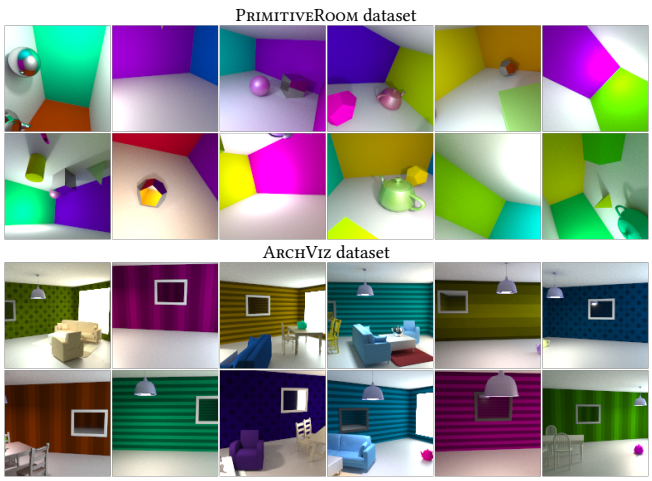


Fig. 5. Random scenes from the PrimitiveRoom and the ArchViz datasets.

All partial representations are combined to obtain a (more) complete representation of the scene using an order-independent aggregator; we use componentwise averaging: $\mathbf{r} = \text{avg}(\{\mathbf{r}_i\}_{k=1..n})$. Alternative aggregators such as summation, attention networks, and max pooling have been explored by Eslami et al. [2018], Rosenbaum et al. [2018], and Deschaintre et al. [2019], respectively.

3.2 Image Generator

The purpose of the image generator is to synthesize an image from a novel, unobserved view v of the scene. The generator receives: (i) parameters of the camera $\mathbf{c}_v$, (ii) a G-buffer $\mathbf{g}_v$ rendered using a traditional renderer from the novel view v, and (iii) a view-independent scene representation $\mathbf{r}$ extracted by the encoder.

Numerous neural image generators have been proposed in the past. Some, such as denoising and shading U-nets [Chaitanya et al. 2017; Nalbach et al. 2017], do not utilize any learned representations and reconstruct the image from image buffers rendered from the novel view. Others 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, as illustrated in Figure 3. We briefly outline their architecture here and provide details in the supplementary material.

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.

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.

We use four levels in our U-net. The G-buffer $\mathbf{g}_v$ dictates the resolution of the first and last level of the U-net. 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. In our implementation, the representation vector $\mathbf{r}$ and camera parameters $\mathbf{c}_v$ are concatenated and duplicated spatially to match the $\mathbf{g}_v$ resolution. These are then given as input to the network and concatenated to the outputs of each hidden layer.

3.3 Auxiliary Image Features

All the aforementioned generators utilize a G-buffer $\mathbf{g}_v$ rendered from the novel view. Similarly, the scene encoder operates on observations augmented with a G-buffer. This has the benefit of accelerating the optimization, improving the image quality, and allowing smaller, faster image generators to perform well. In this paper, the decision of what surface features to include in the G-buffer is primarily driven by the ease of analysis and interpretability of the

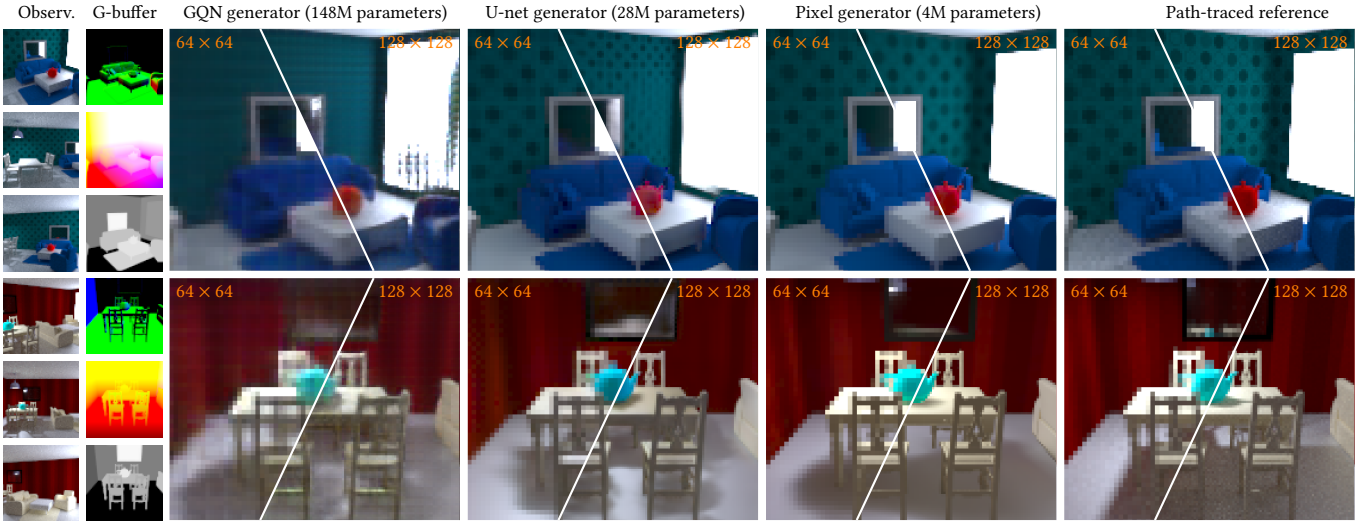


Fig. 6. Comparison of three different image generators that consume (i) a neural scene representation extracted from three observations (left column), and (ii) a G-buffer (second left column) containing *geometric* information about directly visible surfaces (position, normal, object ID). Each generator 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) and fastest to train, the pixel generator delivers the best results overall. The most noticeable artifacts appear on shadows and reflections; we analyze these in Section 5.

learned scene representation. We chose to include only *geometry* information in the G-buffer, namely world-space *positions*, *normals* and *object identifiers*. Information about materials, lighting, and invisible geometry can only be delivered to the generator through the neural scene representation. Figure 4 demonstrates the impact of the scene representation on the quality of generated images.

Such clear separation enables easy analysis of the flow of scene information. A more practical scenario, where the G-buffer also contains material and lighting information, is shown in Figure 1 and Figure 22 and discussed in Section 6.3.

3.4 Datasets and Optimization

We use two datasets in our experiments. Each dataset consists of 144k procedurally generated instances; a detailed generation recipe is provided in the supplementary material. The beauty images (see Figure 5 for examples) were rendered using path tracing and feature multi-bounce effects, such as color bleeding and mirror reflections.

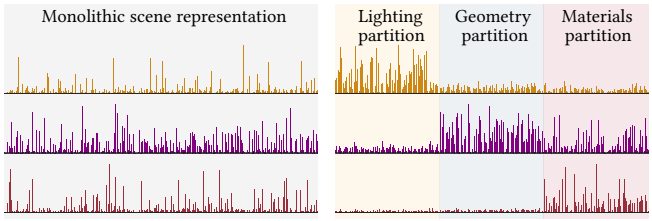
The visually simple PrimitiveRoom dataset contains rectangular rooms with a small number of geometric primitives. The primitives vary in shape, position, and rotation and feature a random instance of either a Lambertian diffuse material, a glossy material, or an ideal mirror. The scene is illuminated by a single spherical emitter.

The ArchViz dataset consists of variations of a living room with a dining area. While still rather simple, the geometry, materials, and lighting resemble realistic apartment design. The variations are again created by randomizing the placement of the luminaire and geometric primitives or varying the (textured) albedo and roughness of materials. The textures and scene objects are extracted from scenes published by Bitterli [2016].

Optimization. We trained all of our models end-to-end using the Adam optimizer [Kingma and Ba 2014] with a learning rate of 10^{-4} and minibatches consisting of 16 scenes. For each scene, we construct the neural scene representation by encoding three random observations with 64×64 resolution. The resolution of the G-buffer input into the image generator is also 64×64 during training. We use a loss comprising a pixel-wise L1 error term and a structural dissimilarity (DSSIM) term, which we empirically found to work well for the U-net and pixel generator. The weights of the terms are manually selected such that the losses are of approximately equal magnitude. For the GQN generator we rely on the ELBO-based loss utilized by Eslami et al. [2018]. We optimize all models using one million batches. This amounts to 111 training epochs and requires about 8.5, 8.5, and 10 days of training for the pixel, U-net, and GQN generators, respectively on a single NVIDIA Tesla V100 GPU.

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.

Once a model is trained, we use it to generate images of a novel, previously unseen scene in the following way. We use a traditional renderer to render the scene from three random 64×64 camera views (observations). The camera parameters, the G-buffers, and the beauty images are passed to the scene encoder to obtain a view-independent scene representation $\mathbf{r}$. To create a novel view, we pass $\mathbf{r}$ and the parameters of the view (and the G-buffer) to the image generator, which synthesizes the beauty image. The representations that we use have between 128 and 512 dimensions.



(a) No partitioning (b) Static partitioning

Fig. 7. In contrast to the monolithic scene representation (left) our partitioned scene representation (right) splits scene information into lighting (yellow), geometry (blue), and material (red) partitions. Each row shows the average *standard deviation* in each dimension of the representation when only lighting (top row), geometry and materials (middle row), or only materials (bottom row) are randomized across a large set of scenes.

3.5 Quality Assessment

In Figure 6 we show two representative results obtained with the three image generators described in Section 3.2. 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 generators are sorted according to attained overall visual quality, which, interestingly, *anti-correlates* with their sizes (reported on top of the figure).

The GQN 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 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. Unless stated otherwise, we proceed focusing primarily on the other two (image-to-image) generators that yield better results in our experiments.

Resolution scaling. The U-net generator and the pixel generator produce generally sharp images. The most notable difference, however, surfaces when comparing outputs 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. This graceful scaling is a key advantage of the pixel generator as it avoids the burden of training at all possible resolutions.

The most notable artifacts, common to all generators, are the inaccurate shadows. The exact cause of these artifacts is unclear and difficult to identify with the monolithic nature of the learned scene representation; it could stem from poorly represented geometry but also lighting. In the next section, we propose a technique for partitioning the scene representation, which allows to carry out a fine-grained analysis (Section 5) and to attribute the artifacts to (missing) information in individual partitions and the G-buffer.

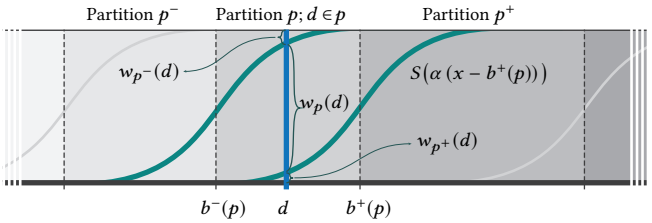


Fig. 8. We adaptively partition the scene representation and assign each dimension d to three consecutive partitions. The contribution of d to a partition is modeled using sigmoids (cyan "S"-shaped curves), which are centered at partition boundaries, and progressively sharpened during optimization.

4 ADAPTIVE DISENTANGLEMENT AND ATTRIBUTION

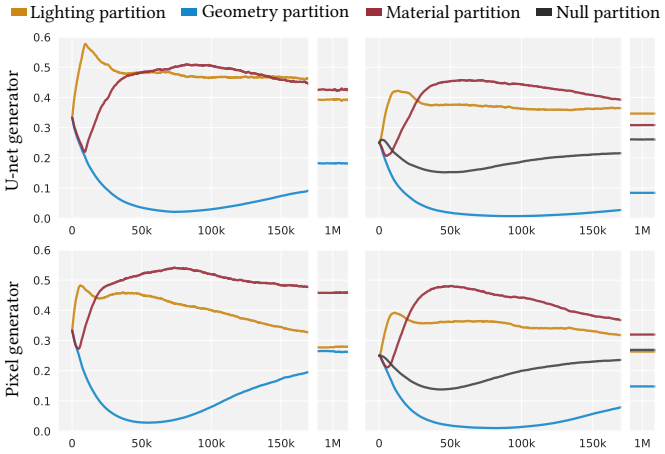
Our goal in this section is to impose a structure over the neural scene representation such that it better respects the orthogonalities between individual scene properties. This improves interpretability of the scene representation by *adaptively disentangling* the material, lighting, and geometry properties of the scene, and storing them in disjoint partitions $\mathcal{R}_M$, $\mathcal{R}_L$, and $\mathcal{R}_G$, respectively. In Section 4.1, we adapt the static partitioning approach of Kulkarni et al. [2015] and extend it to enable adaptive, compressive partitioning in Section 4.2. In Section 5, we will visualize various gradients of the partitioned representation to trace roots of visual artifacts.

4.1 Static partitioning

Denoting $\mathcal{R}$ as the latent space of the scene representations, we split $\mathcal{R}$ into three non-overlapping partitions of equal size, $\|\mathcal{R}_M\| \approx \|\mathcal{R}_L\| \approx \|\mathcal{R}_G\| \approx \|\mathcal{R}\|/3$, that will contain (most of) material, lighting, and geometry information. In order to force each partition to hold only the desired information, we adopt the algorithm of Kulkarni et al. [2015] to disentangle scene components by carefully averaging scene representations and adjusting the gradients during training.

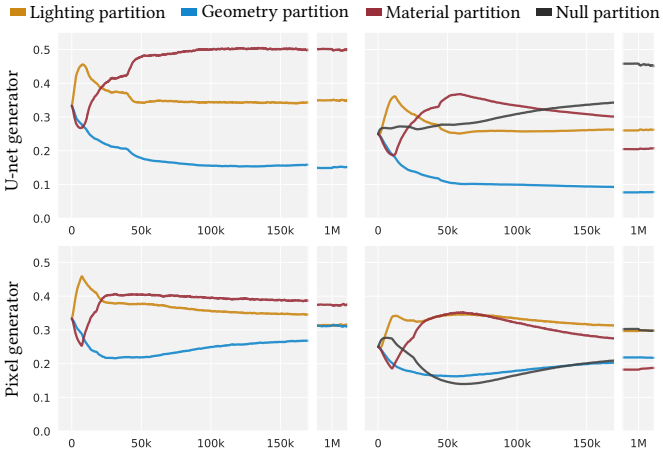
Specifically, Kulkarni et al. [2015] propose to train networks using batches where only *one* scene property varies. During forward propagation, activations in dimensions that store all the other properties are averaged; these batch-invariant properties should lead to identical activations for all entries in the batch. During backpropagation, the partial derivatives in dimensions for batch-invariant properties are adjusted; the gradient is replaced by its difference from the per-dimension mean. Using the differences nudges the encoder towards producing activations that are closer to the mean.

Example. In our case, we may decide to randomize, for instance, only materials across the batch and keep lighting and geometry identical across all batch entries. For each entry we first compute the scene representation with the scene encoder. We then modify activations in lighting and geometry partitions by averaging each dimension across the batch. During backpropagation, we compute loss gradients with respect to the scene representations; one for each entry in the batch. Then we compute the mean partial derivative for each lighting and geometry dimension across the batch, and set the derivative in those dimensions to its difference from the mean. The backpropagation then continues further to update the weights of the scene encoder.



(a) Adaptive partitioning (b) Adaptive w/ null partition (black)

Fig. 9. Adaptive partitioning distributes dimensions of the scene representation between lighting (yellow), geometry (blue), and material (red) components of the scene. The curves show relative sizes of individual partitions and how they evolve during training on the PrimitiveRoom dataset; they stabilize around 1M batches. Note how the partitions become smaller when a null partition (right, black curve) is added to compress the representation.



(a) Adaptive partitioning (b) Adaptive w/ null partition (black)

Fig. 10. Same as in Figure 9 but trained on the ArchViz dataset.

The aforementioned algorithm by Kulkarni et al. will ensure that each scene component, i.e. lighting, geometry, and materials, is stored primarily in its own partition. Figure 7 demonstrates the difference between a monolithic and partitioned neural representation. Each bar in the six charts represents the average standard deviation of activations in one dimension across many scenes, where only lighting (top), geometry and materials (middle), or only materials (bottom) are varied. We choose to vary geometry and materials jointly in the middle row since it is difficult to modify geometry without affecting materials. This is a side-effect of changing object visibility in the observation views across a batch.

4.2 Adaptive partitioning

The main drawback of a statically-partitioned representation is that it strictly defines the number of bits used to store each scene component. This constraint, which is not present in monolithic representations, is undesired. Instead of using fixed partitioning (e.g. the lighting partition occupies the first third of the representation), we allow moving the partition boundaries to adjust their sizes.

For a scene representation with k partitions, we add a trainable tuple $\mathbf{s} := (s_1, \ldots, s_k); s_i \in \mathbb{R}$, to the scene encoder, which defines the sizes of partitions. For the i-th partition, the left boundary $b^-(i)$ and the right boundary $b^+(i)$ are computed by summing up (softmaxed) size parameters of preceding partitions: $b^-(i) = \|\mathcal{R}\| \cdot \sum_{j=1}^{i-1} \sigma(\mathbf{s})_j$, and adding the size of the i-th partition: $b^+(i) = b^-(i) + \|\mathcal{R}\| \cdot \sigma(\mathbf{s})_i$, respectively. The softmax function σ ensures that the learned sizes partition unity.

To allow gradient-based optimization of $\mathbf{s}$, we use fuzzy partition boundaries. A given dimension d of the representation located within a boundary region is shared by multiple partitions: a center partition p, and left and right neighboring partitions, p^- and p^+, respectively. The contribution of dimension d to each partition is given by the following weights:

$$w_p(d) = 1 - w_{p^-}(d) - w_{p^+}(d), \tag{1}$$

$$w_{p^-}(d) = 1 - S\big(\alpha(d - b^-(p))\big), \tag{2}$$

$$w_{p^+}(d) = S\big(\alpha(d - b^+(p))\big), \tag{3}$$

where S is the sigmoid function used to model the fuzzy boundaries; see Figure 8. The parameter α controls the spread of the sigmoid; we progressively increase α during training to approach a step transition in the limit, resulting in a disjoint partitioning.

We treat the representation vector as a circular domain, where the last and first partitions are adjacent. The first partition then acts as the right neighbor of the last partition, and vice versa. All partitions have the same initial size set to $\|\mathcal{R}\|/(k+1)$.

Analysis and Discussion. As the optimization progresses, the partitions trade dimensions to best distribute the bits for storing relevant scene properties, as shown in Figures 9 and 10. Initially, the lighting partition grows at the cost of other partitions. This is likely due to the loss function being more sensitive to pixel brightness than to color hue. Once the lighting is predicted sufficiently well, the models focus on extracting material information to correctly predict colors. When converged, the material partition is typically larger than the lighting partition as lighting in our scenes can be represented with fewer dimensions.

The geometry partition initially shrinks as the generators can rely on the G-buffer (e.g. position and normal buffers). It grows back once lighting effects due to objects that are not directly visible by the camera (e.g. reflections) start dominating the loss. Comparing the U-net generator (top row) and the pixel generator (bottom row) we see that the pixel generator forces the encoder to put more information into the geometry partition. This is to be expected as it cannot rely on pixel neighborhoods like the U-net. In general, both convolutional approaches (U-net and GQN) utilized the geometry partition less in all our experiments.

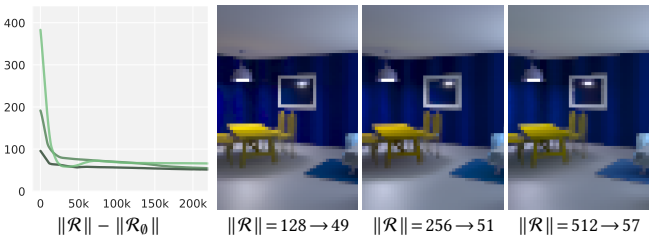


Fig. 11. The null partition allows compressing the scene representation. We plot the total size of the *non-null* partitions, $\|\mathcal{R}\| - \|\mathcal{R}_\emptyset\|$, during training for three models with 128, 256, and 512 available dimensions. The images (right) rendered after 400k iteration (with the representations shrunk to 49, 51, and 57) have comparable, but lower quality than uncompressed representation (cf. Figure 12).

Fig. 12. The lossy compression of representations from Figure 11 ($\beta = 4 \cdot 10^{-4}$) manifests as the loss of material information for the teapot.

Fig. 13. Quality of generated images as a function of β, which scales the loss term induced by the total size of the non-null partitions. The differences are best visible on the teapot and on shadows.

4.3 Null partition

Adaptive partitioning allows trading bits of the representation between partitions. However, the scene representation will always use all the available space, even in cases when it could be well-represented using fewer bits. We address this by adding a *null partition* $\mathcal{R}_\emptyset$, which does not influence the image generator, and serves only as a reservoir of unused (available) dimensions.

In order to incentivize the optimization to grow the null partition, and thereby compress the scene representation, we add a penalty term $\beta(\|\mathcal{R}\| - \|\mathcal{R}_\emptyset\|)$ to the loss function, which is proportional to the number of dimensions in all other partitions.

Analysis and Discussion. The right columns of Figures 9 and 10 show how the presence of the null partition impacts the optimization. The partition sizes at 1M iterations better represent the volume of extracted lighting, geometry, and material information than the uncompressed plots (left) as the model utilizes fewer dimensions to represent them.

The presence of the null partition removes the burden of a-priori guessing the optimal size of the neural scene representation. One can over-allocate the space conservatively and rely on the optimization to shrink or grow the non-null partitions appropriately. We confirm this in Figure 11 that plots the total number of lighting, geometry, and material dimensions in three models that initially allocate 128, 256, and 512 dimensions for the representation. All models gradually reduce the number of utilized dimensions to similar counts (49, 51, and 57 dimensions after 400k training iterations); the remaining dimensions are assigned to the null partition. The rendering quality is comparable between the three models, but lower than in the case of uncompressed representation. Figure 12 shows an inset where the compressed representations do not capture the material of the teapot. The tradeoff between the compression ratio and the loss of information can be controlled by adjusting β, as shown in Figure 13.

5 ANALYSIS AND ATTRIBUTION

The partitioning of the scene representation allows us to attribute the artifacts and poor image quality to specific information that is missing or underrepresented in the representation. In this section, we utilize the *gradient* × *input* attribution method [Shrikumar et al. 2016] to measure the sensitivity of generated images to the neural scene representation, the observations, and the G-buffer.

In cases where we compute the attribution of a collection of output values (e.g. for an image patch or an entire partition of the representation) we compute the gradient × input products for individual elements and sum their absolute values to obtain a single attribution scalar.

5.1 Attribution of Partition Activations

In Figure 14, we aggregate the attribution of partitions in the scene representation to the beauty and position channels of the observations. The false-color intensity shows the aggregate attribution of one partition to each pixel in the observations.

Activations in the lighting partition are attributed primarily to the ceiling light fixture, if visible, since its position and intensity are randomized in the dataset. In the beauty channel observations we can see that the activations are mainly attributed to regions where the emitters are directly visible. These regions contain information on the brightness of the light sources. In the position channel of the observations we see activations on the entire light fixture and the floor. We hypothesize that the structure of shadow features on the floor (beauty channel) in conjunction with their corresponding locations (position channel) contribute to the identification of the emitters positions, especially in cases when the emitters are not visible in any of the observations. The relatively weak contribution of the wall emitter in the position channel of observation 2 (right part of the image) to the lighting partition is due to its fixed position; only its intensity is randomized.

Activations in the geometry partition are attributed to regions and objects that move during training except for the walls, which are easy to identify as their positions are fixed across all training examples. Activations in regions where the light sources are directly visible indicate that the encoder might be performing some form

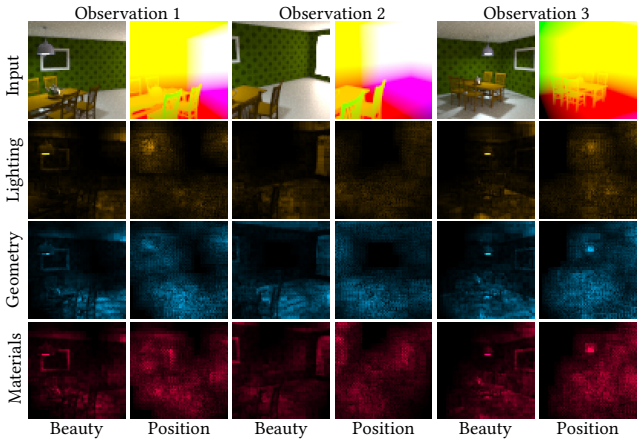


Fig. 14. Attribution of activations in the lighting, geometry, and material partition of the scene representation extracted from beauty and position channels of three observations (top row). In each row below we provide false-colored attributions aggregated over all dimensions in each partition. Bright values indicate large impact of the pixel on the partition.

of shape from shading or shape from shadow calculations. However, such behavior can also be attributed to extracting the emitters geometric positions.

Activations in the material partition are attributed to regions with randomized materials (walls, mirror frame, table, chairs, and the teapot), but also to the emitters. This is because the intensity and position of the emitters is needed to correctly extract material parameters without baking illumination effects into them.

5.2 Attribution of Generated Colors

In Figure 15 we visualize the attribution (gradient × input) of predicted colors to the neural scene representation (middle) and to the observations (left columns) that the representation was extracted from. We use the pixel generator to obtain the generated image.

Reflections. The orange patch focuses on a reflection of a light fixture seen through a mirror surface. Since the corresponding pixels of the G-buffer contain information only about the mirror, and no information about the reflected objects, the generator needs to rely on the scene representation to produce colors of the reflected light fixture. Note that the reflection is in neither of the observations, but the lighting fixture itself is directly observed and the corresponding pixels in the observations have large impact on the reflection; their attribution is high. This indicates that the network is capable of synthesizing view-dependent reflections.

The red patch is placed on a mirror teapot, which reflects large portions of the scene, including the light fixture. The colors in the red patch are thus attributed to all partitions of the representation as reflections require all scene components. In the observations the colors are strongly attributed to the light source as it illuminates the reflected surfaces, but also to the teapot itself when visible.

Materials. The blue patch focuses on the textured wall. The generated colors should depend on the incident illumination and the material of the wall. This is confirmed by the representation attribution, where the lighting partition (yellow background) and material partition (red background) have high attribution values, and also by the observations, where the attribution is high on the light fixture and the walls. Since all textures used during training were periodic, the texture pattern can be detected from any region of the wall, although some regions are preferred over others; we attribute this to biases in the stochastic placement of the camera.

Shadows. The green patch focuses on a shadow boundary due to the table and the light fixture. The representation attributions show that the lighting and geometry partitions are mainly responsible for the synthesis of this shadow region. The material partition is not attributed because we used the same glossy gray material for all floors in the ArchViz dataset. The floor material is thus likely embedded in the generator and the encoder was not forced to extract it. As expected, the shadow boundary is attributed to the light fixture and the pixel region around the table and the chairs in the observations.

5.3 Attribution to Representation and G-buffer

We now study whether the generator relies more on the scene representation or the G-buffer.

In the top row of Figure 16 we demonstrate that this balance is specific to the architecture of the generator by studying the quality of synthesized shadows. The scene contains a cylindrical shadow caster. In the left configuration, the cylinder is in the camera frustum of the novel view and therefore captured by the G-buffer. The generators can thus rely on the representation, but also on the G-buffer to identify the shape and position of the shadow caster. In the right configuration, the caster is outside of the frustum and its position and shape can only be inferred from the neural scene representation.

The bar charts below the images attribute the predicted colors in the red patches to (i) the partitions of the representation (top stacked bars) and (ii) the position and normal channels of the G-buffer (bottom stacked bars). The bar charts indicate that the pixel generator relies primarily on the neural scene representation; the geometry partition (blue bar) is the largest. In contrast, the U-net generator barely utilizes the geometry partition and relies significantly more on the G-buffer (green and black bars). This poses a problem when the shadow caster is outside the camera frustum: the G-buffer contains no information about the caster and the U-net generator is unable to predict the shadow accurately.

The results of this comparison are not entirely unexpected, considering that convolutional architectures, which source information from pixel neighborhoods, will rely more on the G-buffer in our case. Pixel generators, which access one pixel at a time only, force the encoder to put more information about the geometry into the scene representation (as already suggested by Figures 9 and 10) and their performance is thus less impacted by the content of the G-buffer.

The other drawback of the convolutional U-net is the artificial shrinking of shadows in cases where the novel view resolution is larger than the one the model was trained on (cv. Figure 6, bottom image).

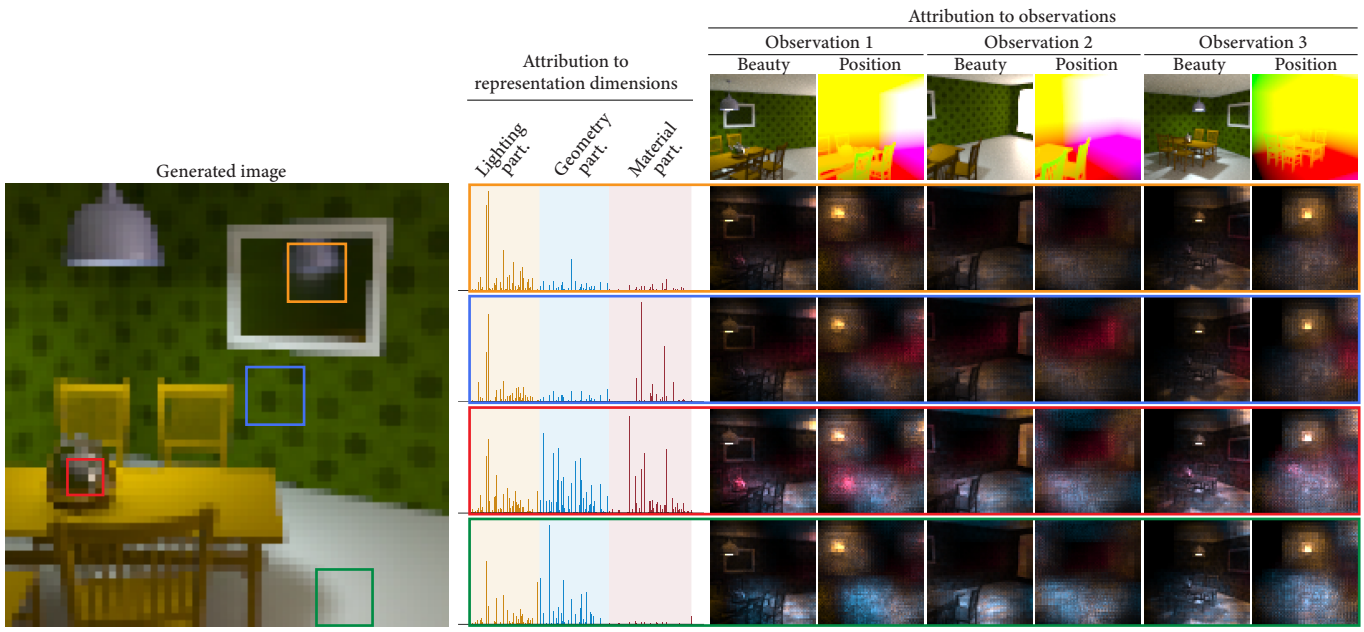


Fig. 15. We attribute the colors of four patches of generated image (left) to dimensions in the neural scene representation (middle), and to the beauty and position channels of the three observations (right) that the representation was extracted from. Each bar in the attribution histograms shows the contribution of one dimension of the representation to the patch; the bar height corresponds to the aggregate attribution. On the right, we propagate the attribution through each partition to individual observations and their channels. A blue tint, for instance, indicates that an observation pixel contributed geometry information (through the geometry partition) to the generated patch; the brightness corresponds to the aggregate attribution.

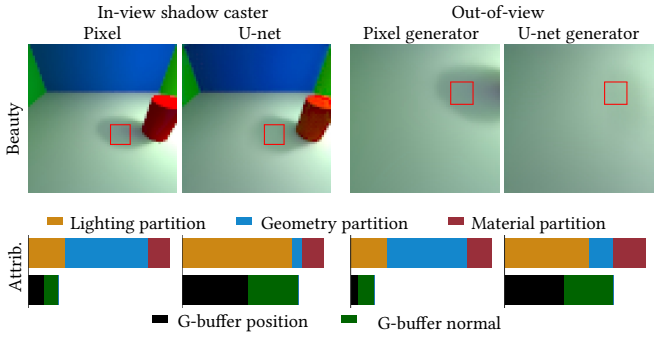


Fig. 16. Shadow synthesis in a scene configuration where the shadow caster is visible from the novel view (left pair) and when the shadow caster is out of the view (right pair). Both pixel and U-net generators are able to synthesize shadows of visible shadow casters. When the shadow caster is not visible, the U-net is unable to synthesize a shadow since it relies primarily on the G-buffer. The bar charts attribute colors in the red square to individual partitions and components of the G-buffer.

Summary. The analysis in this section revealed poor performance of direct illumination shadows. Figure 16 illustrated the importance of encoding a sufficient amount of geometric information in the geometry partition of the representation, especially in the case of U-net generators; these tend to rely on the G-buffer too much and fail to synthesize shadows due to out-of-view shadow casters.

6 ADDITIONAL IMPROVEMENTS AND RESULTS

In this section, we describe a mechanism for increasing the amount of specific information in the scene representation, and present additional results and comparisons.

6.1 Auxiliary Shadow Generator

We employ an auxiliary generator [Philip et al. 2019] that is trained concurrently with the main generator, but focuses on the task of synthesizing shadows. Specifically, the generator is trained to produce grayscale images of (partial) visibility of light sources. The loss function of the generator is the same as the loss function of the main generator; except we evaluate it on reference shadow images. The losses are summed together.

Adding the auxiliary generator can be interpreted as changing the loss function, but it has the key benefit of impacting *only* the scene encoder of the original model; it keeps the original generator (and its loss) intact. Since both generators use the same scene representation during training, the auxiliary generator merely steers the scene encoder to extract geometry and lighting information.

Figure 17 shows examples of improved shadow synthesis of a pixel generator trained with an auxiliary shadow generator. Figure 18 confirms the expected growth of the geometry partition to accommodate additional geometric information. The lighting partition also marginally increases. This comes at the cost of the material partition. As a result, materials tend to be captured with lower accuracy. This can be observed on some of the objects (e.g. prediction of material roughness is less accurate). Adding an auxiliary generator thus does not necessarily lead to better overall results—quantitative

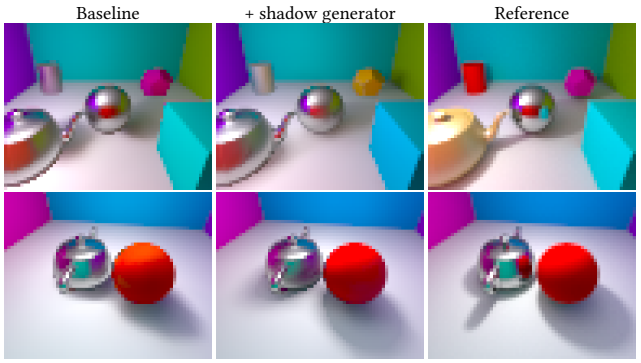


Fig. 17. Adding an auxiliary shadow (pixel) generator steers the encoder to focus more on geometric and lighting information, which improves the synthesis of shadows, albeit at the cost of encoding materials.

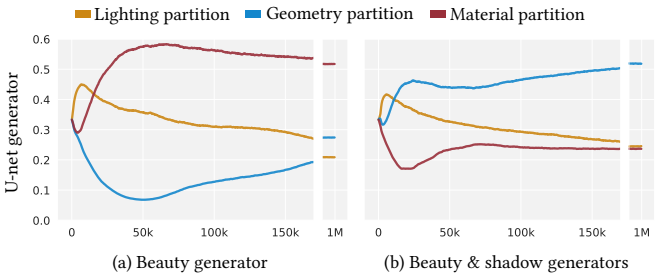


Fig. 18. A model with a single pixel generator (left) can be steered towards extracting more geometry information (blue) by adding an auxiliary shadow generator (right). At 1M iterations the auxiliary generator resulted in the geometry partition to grow to 52% (instead of 27% in (a)) at the cost of the material partition, which shrunk to 24% (instead of 52% in (a)).

metrics remained unaffected in this case—but rebalances the focus of the encoder.

6.2 Quantitative Evaluation

For each dataset, we hand-picked 16 scenes with difficult shading and several objects. These scenes were not used during training. We compare the performance of different models on these scenes after one million training iterations using three different error functions: mean absolute percentage error (MAPE), PieAPP [Prashnani et al. 2018] and LPIPS [Zhang et al. 2018]. For PieAPP and LPIPS metrics we first tonemap the high dynamic range images into low dynamic ranges using gamma correction, quantize to 8-bits precision per channel and clip values above 255.

In Figure 19 we report the average metrics across the 16 scenes for models with: (i) a monolithic scene representation, (ii) statically partitioned representation, and (ii) our adaptively partitioned representation. The static partitioning typically yields the worst results as the static sizes do not reflect on the entropy of individual scene components and over-constrain the problem. Our adaptive partitioning scheme relaxes the constraints and recovers some of the lost performance of the, typically best-performing monolithic representation.

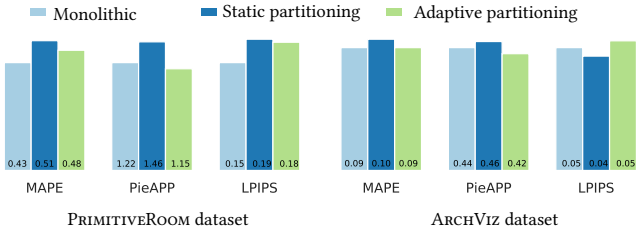


Fig. 19. Average metrics of the pixel generator model with no partitioning, static partitioning, and our adaptive partitioning (higher means better).

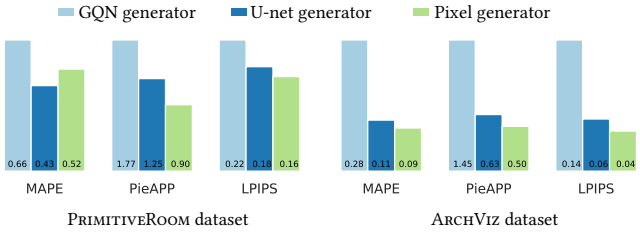


Fig. 20. Average metrics of different generators on 16 selected scenes rendered with a 64×64 resolution (higher means better).

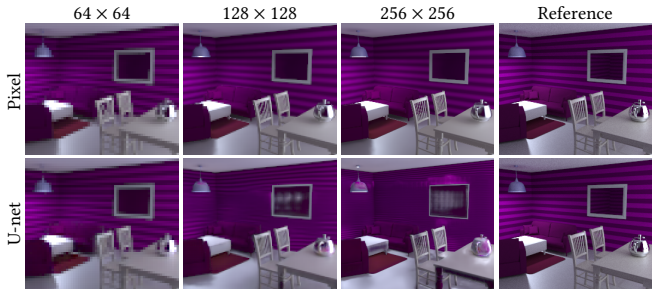
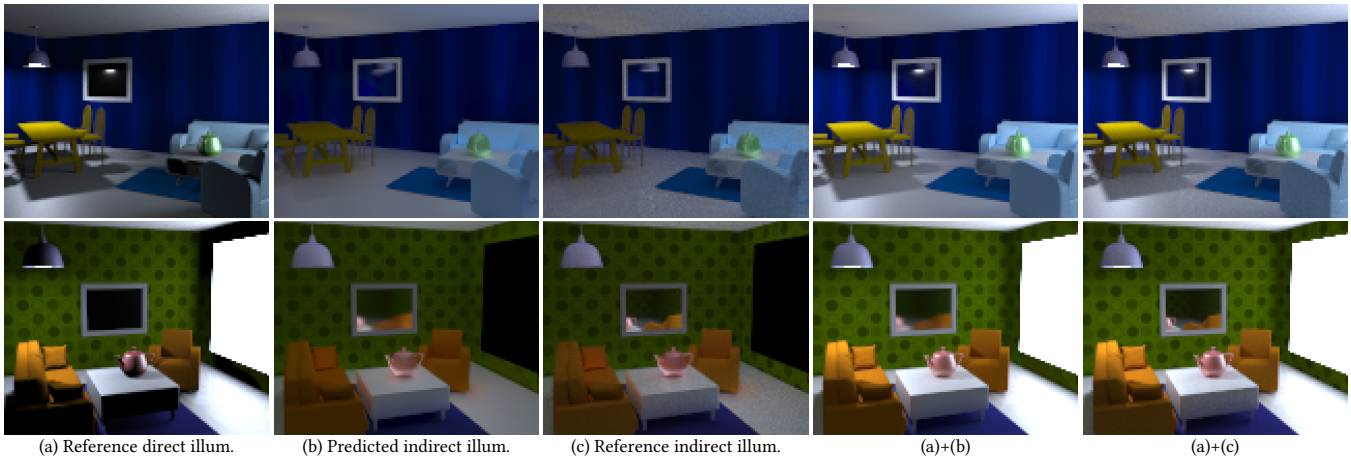


Fig. 21. Pixel generator stability across resolution changes. For each generator, we used one neural scene representation obtained from observations with 64×64 resolution. The pixel and U-net generators were trained to synthesize 64×64 images, and then used at higher resolution. For the pixel generator, visual differences are due to the varying resolution of its input G-buffers, whereas the U-net generator suffers from shrinking shadows as the resolution increases and various other artifacts.

In Figure 20 we report average metrics of the GQN, U-net and pixel generators. 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 placed last; we would like to note that this approach has not been designed for image-to-image translation, which could explain its worse performance. These results confirm our qualitative evaluation in Section 3.5.

As stated by Sitzmann et al. [2019], a key feature of the pixel generator is its resolution independence. We confirm this statement in Figure 21, which compares the outputs of the pixel and U-net generators at various resolutions, when both where trained to generate images at a fixed 64×64 resolution. Note, in particular, how



(a) Reference direct illum. (b) Predicted indirect illum. (c) Reference indirect illum. (a)+(b) (a)+(c)

Fig. 22. Synthesizing indirect illumination. We provide a pixel generator with the scene representation, G-buffers and the direct illumination buffer, leaving only the indirect illumination to be synthesized. The observations from which the scene representation was extracted consisted of the beauty buffer and the G-buffers, but did not include the direct illumination. Results show that interreflections are correctly synthesized by the generator, for instance between the teapot and the table (top and bottom row) or the pillow and the sofa (bottom row). Interreflections between distant surfaces, such as the yellow tint of the ceiling (top row) due to light bouncing on the table underneath, are also synthesized.

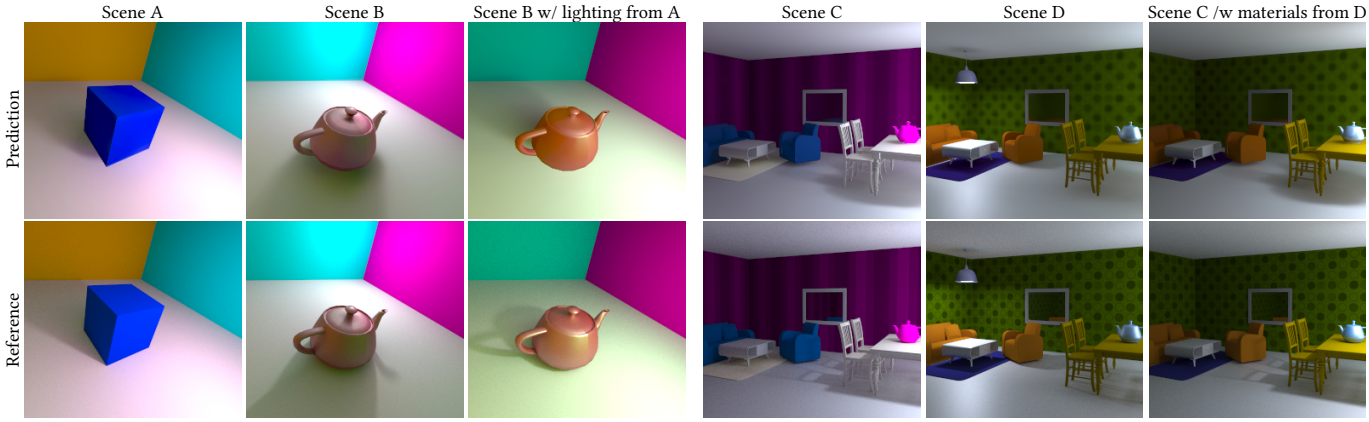


Fig. 23. Generated examples (top row) where lighting and material representation partitions are transferred across scenes. On the left, we show a relighting example obtained after transferring the lighting partition of Scene A to the lighting partition of Scene B. On the right, we show another example obtained after replacing the material partition of Scene C with the one from Scene D. The bottom row contains corresponding rendered references for each configuration.

the shadows generated by the pixel generator are stable across resolutions. In contrast, shadows synthesized by the U-net generator shrink as the resolution increases. The wall texture synthesized by the pixel generator remains blurry at higher resolutions, since training was done on 64×64 observations, but becomes severely distorted with the U-net generator.

6.3 Modeling Indirect Illumination

One shortcoming of our current data-driven image synthesis pipeline is that high frequency variations, as well as fine features, cannot be reproduced with sufficient fidelity. This limitation is not unique to our method and a common solution, used for instance in the irradiance caching algorithm [Ward et al. 1988], is to synthesize only indirect illumination as it tends to vary more smoothly.

For this application, we provide the direct illumination buffer of the novel view to the generator (in addition to the other channels of the G-buffer), leaving only the indirect illumination to be synthesized. However, the direct illumination buffer is not provided with the observations. This way, we enforce the scene encoder to separate out the indirect component to enable its synthesis by the generator. In this context, our framework is capable of producing highly convincing results, as illustrated in Figure 22. We used a pixel generator to ensure that indirect illumination could not be inferred using image-space approximation [Nalbach et al. 2017]. Our two test scenes feature interreflection between the teapot and the table that is correctly synthesized. Similarly, the predicted interreflections between the pillows and the sofa (bottom row) match the reference. Lastly, indirect illumination over long distances, such as the yellow tint on the ceiling (top row) due to light bouncing off the yellow table underneath, is also synthesized by our framework.

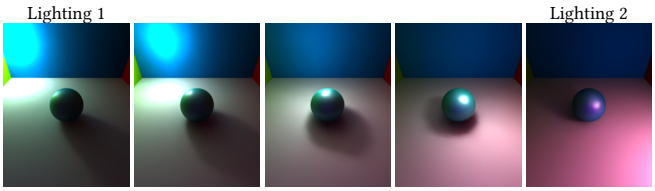


Fig. 24. Lighting partition interpolation, where the interpolation endpoints were from extracted from lighting partitions of two other scenes, shows that indirect color bleeding and specular highlights are correctly synthesized even though the material partition remains constant. This proves that the model has learned to properly separate materials from lighting.

6.4 Relighting and Material Transfer

In this section we exploit the compositionality of our neural scene representation to perform scene edits directly in the latent representation space. The semantic partitioning into lighting, geometry, and material components allows applications such as material transfer across scenes, or relighting.

In Figure 23 we transfer material and lighting components between different scene representations, which were constructed from observations of different scenes. The scenes differ in more aspects than the one being transferred. This makes the transfer tasks challenging for monolithic representations, that do not have a clear separation of individual components. On the left-hand side, Scene A and Scene B differ in lighting, materials, and geometry. On the right-hand side, Scene C and Scene D have differences in lighting and materials (geometry is identical).

We begin with a relighting application where we replace the lighting partition in the representation of Scene B with the lighting partition from Scene A. We then feed the composited representation with the G-buffer of Scene B into the pixel generator. The generated image is shown in the third column (top); reference renderings are included in the bottom row. We observe that the backlit teapot is now lit from the front and the highlights and shadows are repositioned correctly.

On the right-hand side, we present a material transfer application, where we substitute materials of Scene C with materials from Scene D. Analogously to before, we replace the material partition of Scene C with the corresponding partition from Scene D, and pass the composite to the pixel generator. The resulting image on the right shows that material properties have been transferred accurately, their appearance adapted to new illumination conditions, and they impact indirect illumination as expected; the ceiling and the floor have slightly different color tint.

In both transfers shown in Figure 23, we can see that the generated images of novel scene configurations retain the quality of images of the original scenes.

In Figure 24, we perform a linear interpolation between two lighting configurations to demonstrate that highlights and shadows move gradually, as if the light source was moving, instead of blending intensities of two light sources. Please see the supplementary video for additional animations.

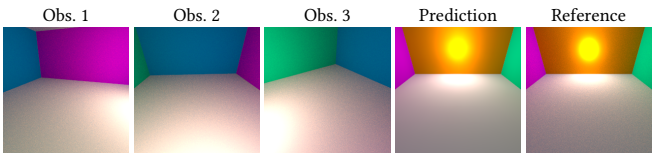


Fig. 25. Despite the orange wall is not visible in the observations, the model is capable of extracting its color from indirect lighting in the observations.

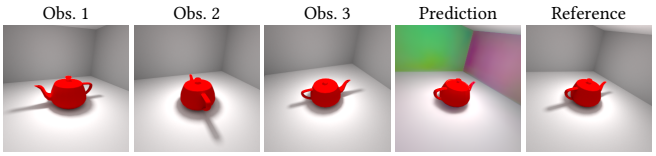


Fig. 26. An example of poor generalization of the model to a scene with "unknown" materials; gray color was never used for the walls in the training dataset, only for the floor and the ceiling. The model incorrectly maps the gray color of the walls in the observations to shades of green and pink.

7 DISCUSSION AND FUTURE WORK

Difficult cases and generalization. Figure 25 shows a challenging case for the encoder: the observations do not capture the orange wall that is viewed in the novel view. Nevertheless, the encoder is capable of extracting its material parameters from the indirect illumination on the floor. On the other hand, if the model is presented with configurations that it did not encounter during training, such as the gray walls in Figure 26, it is prone to projecting these onto configurations experienced during training rather than generalizing gracefully.

Implicit versus explicit disentanglement. It has been shown that neural scene encoders can naturally produce representations that are disentangled; see e.g. the scene algebra experiments demonstrated by [Eslami et al. 2018]. We observed this in our experiments as well. For instance, changing a single texture and computing the difference in scene representations had significant values only in one dimension (see the supplemental video). This one dimension in the representation fully controlled the pattern of the wall texture. This suggests that the image generator developed a parametric texture synthesizer, which is controlled by (at least) the identified dimension. Interestingly though, sweeping through the meaningful range of values in this dimension produced only a subset of the textures used in the training dataset. The remaining patterns were produced by activations in other dimensions. Explicit partitioning and disentanglement thus still provides more direct and exact control.

Validity of design decisions. We made the deliberate choice to use classical rendering algorithms only for a very specific set of image buffers. Our primary desire was to *not* provide any material and lighting information in the G-buffer, such that we can better analyze the inner workings of the model. Only geometry was captured by both, the G-buffer and the neural scene representation, which allowed contrasting the performance of the U-net and pixel generators in Figure 16.

Utilizing the classical renderer to obtain additional surface parameters, such as albedo and roughness, or intersections of specular reflection rays improved the quality in our experiments. Figures 1 and 22 demonstrated the benefits of including direct illumination renders in the G-buffer. One must however ensure that the model does not drift towards using the G-buffer exclusively and that the information is appropriately present also in the latent scene representation, e.g. by using auxiliary generators.

The other decision, which is worth revisiting in the future, is the choice to extract the scene representation from a set of image buffers (observations). The observations have the benefit of including all the visual effects, which we strive to synthesize, but other forms of input, e.g. voxel representations, unstructured sets of radiance estimates, light fields, or textual descriptions could suit particular applications better.

8 CONCLUSION

We presented a method for augmenting classical rendering with neural scene representations. We extended the approach of [Eslami et al. 2018] by allowing it to use a G-buffer (for both the observations and the novel view), and combined it with the pixel generator [Sitzmann et al. 2019] and U-net generator [Ronneberger et al. 2015]; these two adjustments enabled obtaining good-quality results after only days of training.

We then adopted the approach of Kulkarni et al. [2015] to partition the representation, and extended it to permit sizing the partitions adaptively according to the entropy of each scene component. By adding an null partition we enabled (lossily) compressing the representation. The adaptive partitioning provided means for gaining new insights in the data flow of the model. Specifically, it allowed us to attribute the resulting colors to individual pixels in the observations and, importantly, reasoning whether a pixel in an observation impacted the final color through the lighting, material, or geometry partition. Based on these insights, we proposed adding an auxiliary generator to steer the representation extraction towards missing or underrepresented information; the subsequent analysis of partition sizes confirmed that the auxiliary generator had the intended impact. Yet, the results generated by models presented in this paper suffer from visual artifacts if used to synthesize all shading. We thus investigated a scenario, where the neural renderer synthesized colors due to indirect illumination only; the obtained images featured quality sufficient for many practical applications.

Addressing the quality, however, should still remain a priority for future work as artifacts may not go away by merely increasing the size of networks and datasets. We believe that additional constraints and principles of light transport will be necessary to yield realistically looking images. Understanding of the model's inner workings will be paramount for injecting additional constraints; our work presents an important step in that direction.

REFERENCES

Marco Ancona, Enea Ceolini, Cengiz Öztireli, and Markus Gross. 2018. Towards better understanding of gradient-based attribution methods for Deep Neural Networks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=Sy21R9JAW

Benedikt Bitterli. 2016. Rendering resources. https://benedikt-bitterli.me/resources/.

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

Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. 2016. InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. In *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett (Eds.). Curran Associates, Inc., 2172–2180. http://papers.nips.cc/paper/6399-infogan-interpretable-representation-learning-by-information-maximizing-generative-adversarial-nets.pdf

Paul Debevec. 1998. Rendering Synthetic Objects into Real Scenes: Bridging Traditional and Image-Based Graphics with Global Illumination and High Dynamic Range Photography. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '98)*. Association for Computing Machinery, New York, NY, USA, 189–198. https://doi.org/10.1145/280814.280864

Paul Debevec, Tim Hawkins, Chris Tchou, Haarm-Pieter Duiker, Westley Sarokin, and Mark Sagar. 2000. Acquiring the Reflectance Field of a Human Face. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '00)*. ACM Press/Addison-Wesley Publishing Co., USA, 145–156. https://doi.org/10.1145/344779.344855

Valentin Deschaintre, Miika Aittala, Fredo Durand, George Drettakis, and Adrien Bousseau. 2019. Flexible SVBRDF Capture with a Multi-Image Deep Network. *Computer Graphics Forum* 38, 4 (2019), 1–13. https://doi.org/10.1111/cgf.13765 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13765

Mengnan Du, Ninghao Liu, and Xia Hu. 2019. Techniques for Interpretable Machine Learning. *Commun. ACM* 63, 1 (Dec. 2019), 68–77. https://doi.org/10.1145/3359786

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 arXiv:https://science.sciencemag.org/content/360/6394/1204.full.pdf

Karol Gregor, Frederic Besse, Danilo Jimenez Rezende, Ivo Danihelka, and Daan Wierstra. 2016. Towards Conceptual Compression. In *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett (Eds.). Curran Associates, Inc., 3549–3557. http://papers.nips.cc/paper/6542-towards-conceptual-compression.pdf

Pedro Hermosilla, Sebastian Maisch, Tobias Ritschel, and Timo Ropinski. 2018. Deep-learning the Latent Space of Light Transport. *cgfegsr* 38 (2018), 207–217.

Irina Higgins, Loïc Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew M Botvinick, Shakir Mohamed, and Alexander Lerchner. 2017. beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework. In *ICLR*.

Simon Kallweit, Thomas Müller, Brian Mcwilliams, Markus Gross, and Jan Novák. 2017. Deep Scattering: Rendering Atmospheric Clouds with Radiance-Predicting Neural Networks. *ACM Trans. Graph.* 36, 6, Article Article 231 (Nov. 2017), 11 pages. https://doi.org/10.1145/3130800.3130880

Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. 2017. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196* (2017).

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980* (June 2014).

Tejas D. Kulkarni, William F. Whitney, Pushmeet Kohli, and Joshua B. Tenenbaum. 2015. Deep Convolutional Inverse Graphics Network. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2 (NIPS'15)*. MIT Press, Cambridge, MA, USA, 2539–2547. arXiv:https://arxiv.org/pdf/1503.03167.pdf https://dl.acm.org/doi/10.5555/2969442.2969523

Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. 2019. Neural Volumes: Learning Dynamic Renderable Volumes from Images. *ACM Trans. Graph.* 38, 4, Article Article 65 (July 2019), 14 pages. https://doi.org/10.1145/3306346.3323020

Wojciech Matusik. 2003. *A data-driven reflectance model*. Ph.D. Dissertation. Massachusetts Institute of Technology.

Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. 2019. Neural Importance Sampling. *ACM Trans. Graph.* 38, 5, Article 145 (Oct. 2019), 19 pages. https://doi.org/10.1145/3341156

Oliver Nalbach, Elena Arabadzhiyska, Dushyant Mehta, Hans-Peter Seidel, and Tobias Ritschel. 2017. Deep Shading: Convolutional Neural Networks for Screen-Space Shading. 36, 4 (2017).

Phong Nguyen-Ha, Lam Huynh, Esa Rahtu, and Janne Heikkilä. 2019. Predicting Novel Views Using Generative Adversarial Query Network. *CoRR* abs/1904.05124 (2019). arXiv:1904.05124 http://arxiv.org/abs/1904.05124

Thu Nguyen-Phuoc, Chuan Li, Lucas Theis, Christian Richardt, and Yong-Liang Yang. 2019. HoloGAN: Unsupervised Learning of 3D Representations From Natural Images.

In *The IEEE International Conference on Computer Vision (ICCV)*.

Thu H Nguyen-Phuoc, Chuan Li, Stephen Balaban, and Yongliang Yang. 2018. RenderNet: A deep convolutional network for differentiable rendering from 3D shapes. In *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.). Curran Associates, Inc., 7891–7901. http://papers.nips.cc/paper/8014-rendernet-a-deep-convolutional-network-for-differentiable-rendering-from-3d-shapes.pdf

Kyle Olszewski, Sergey Tulyakov, Oliver Woodford, Hao Li, and Linjie Luo. 2019. Transformable Bottleneck Networks. *The IEEE International Conference on Computer Vision (ICCV)* (Nov 2019).

Julien Philip, Michaël Gharbi, Tinghui Zhou, Alexei Efros, and George Drettakis. 2019. Multi-view Relighting Using a Geometry-Aware Network. *tog* 38, 4 (July 2019). http://www-sop.inria.fr/reves/Basilic/2019/PGZED19

Ekta Prashnani, Hong Cai, Yasamin Mostofi, and Pradeep Sen. 2018. PieAPP: Perceptual Image-Error Assessment Through Pairwise Preference. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Konstantinos Rematas and Vittorio Ferrari. 2019. Neural Voxel Renderer: Learning an Accurate and Controllable Rendering Tool. arXiv:cs.CV/1912.04591

Peiran Ren, Jiaping Wang, Minmin Gong, Stephen Lin, Xin Tong, and Baining Guo. 2013. Global Illumination with Radiance Regression Functions. *ACM Trans. Graph.* 32, 4, Article 130 (July 2013), 12 pages. https://doi.org/10.1145/2461912.2462009

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.

Dan Rosenbaum, Frederic Besse, Fabio Viola, Danilo J. Rezende, and S. M. Ali Eslami. 2018. Learning models for visual 3D localization with implicit mapping. arXiv:cs.CV/1807.03149

Avanti Shrikumar, Peyton Greenside, Anna Shcherbina, and Anshul Kundaje. 2016. Not Just a Black Box: Learning Important Features Through Propagating Activation Differences. arXiv:cs.LG/1605.01713

Vincent Sitzmann, Michael Zollhoefer, and Gordon Wetzstein. 2019. Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 1119–1130. http://papers.nips.cc/paper/8396-scene-representation-networks-continuous-3d-structure-aware-neural-scene-representations.pdf

Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. 2015. Single-view to Multi-view: Reconstructing Unseen Views with a Convolutional Network. *CoRR* abs/1511.06702 (2015). arXiv:1511.06702 http://arxiv.org/abs/1511.06702

Justus Thies, Michael Zollhöfer, and Matthias Nieundefinedner. 2019. Deferred Neural Rendering: Image Synthesis Using Neural Textures. *ACM Trans. Graph.* 38, 4, Article Article 66 (July 2019), 12 pages. https://doi.org/10.1145/3306346.3323035

Joshua Tobin, Wojciech Zaremba, and Pieter Abbeel. 2019. Geometry-Aware Neural Rendering. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 11555–11565. http://papers.nips.cc/paper/9331-geometry-aware-neural-rendering.pdf

Delio Vicini, Vladlen Koltun, and Wenzel Jakob. 2019. A Learned Shape-Adaptive Subsurface Scattering Model. *ACM Trans. Graph.* 38, 4, Article Article 127 (July 2019), 15 pages. https://doi.org/10.1145/3306346.3322974

Jorg Wagner, Jan Mathias Kohler, Tobias Gindele, Leon Hetzel, Jakob Thaddaus Wiedemer, and Sven Behnke. 2019. Interpretable and Fine-Grained Visual Explanations for Convolutional Neural Networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Gregory J. Ward, Francis M. Rubinstein, and Robert D. Clear. 1988. A Ray Tracing Solution for Diffuse Interreflection. In *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '88)*. Association for Computing Machinery, New York, NY, USA, 85–92. https://doi.org/10.1145/54852.378490

Matthew D Zeiler and Rob Fergus. 2013. Visualizing and Understanding Convolutional Networks. arXiv:cs.CV/1311.2901

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. 2018. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Quan Zheng and Matthias Zwicker. 2019. Learning to Importance Sample in Primary Sample Space. *Computer Graphics Forum* 38, 2 (2019), 169–179. https://doi.org/10.1111/cgf.13628