Compositional Neural Scene Representations for Shading Inference

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

We present a technique for adaptively partitioning neural scene represen-
tations. 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
permits compressing the scene representation to lower its memory footprint
and potentially reduce the evaluation cost of the model. Furthermore, the
partitioned representation enables an in-depth analysis of existing image
generators. We compare the flow of information through individual parti-
tions, and by contrasting it to the impact of additional inputs (G-buffer), we
are able to identify the roots of undesired visual artifacts, and propose one
possible solution to remedy the poor performance. We also demonstrate the
benefits of complementing traditional forward renderers by neural represen-
tations and synthesis, e.g. to infer expensive shading effects, and show how
these could improve production rendering in the future if developed further.

CCS Concepts: • Computing methodologies → Rendering. Neural net-
works.

Additional Key Words and Phrases: rendering, neural networks, neural scene
representations, disentanglement, attribution

ACM Reference Format:
Compositional Neural Scene Representations for Shading Inference. ACM
Trans. Graph. 39, 4, Article 135 (July 2020), 13 pages. https://doi.org/10.1145/
3386569.3392475

1 INTRODUCTION

Data-driven realistic image synthesis has recently achieved a number
of notable breakthroughs, such as rendering realistic human faces [Karras et al. 2018], or high-quality relighting of photographs
[Philip et al. 2019]. Remarkable achievements have been demonstrat-
ed also in data-driven simulation of light transport, where neu-
ral networks predict various radiative quantities [Hermosilla et al. 2019; 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 net-
works 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. 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. While the scene
complexity and rendering quality of these approaches may appear limited at present, it is foreseeable that these will improve in the
future, especially if the neural renderer merely augments classical
rendering pipelines. The key advantage of the neural approach
is the end-to-end training, which allows the neural representation
to carry information that is complementary to classical inputs and
tailored to the task at hand. We carry out our investigations in one
such scenario, where a classical rasterizer determines directly visible
objects and the neural renderer infers their appearance.

We present two distinct contributions in this article. First, we ex-
tend the works of Eslami et al. [2018] and Sitzmann et al. [2019] with
mechanisms to disentangle material, lighting, and geometric con-
tent of the scene. We do not prescribe a specific encoding between
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

This is the author’s version of the work. It is posted here for your personal use. Not for
redistribution. The definitive Version of Record was published in ACM Transactions on
Graphics.

© 2020 Association for Computing Machinery.
0730-0301/2020/7-ART135
https://doi.org/10.1145/3386569.3392475

ACM Trans. Graph., Vol. 39, No. 4, Article 135. Publication date: July 2020.
the roots of poor rendering quality, and allows compressing the neural scene representation. 

Our second contribution, enabled by the adaptive partitioning, is an analysis of generated images with respect to the extracted scene representation and the encoded observations. The analysis yields valuable insights into the inner mechanisms of the neural model and guides the design of techniques for remedying objectionable artifacts and poor performance. Specifically, attributing pixel colors in problematic regions to individual partitions revealed lack of geometric information in the representation; an issue that we address by adding an auxiliary image generator to steer the model towards capturing more geometry.

While the primary focus in this article is on disentanglement and compositionality, we also tease with an example of how neural scene representations could augment classical game and film renderers in the future. We use a ray tracer to render an image with direct illumination, and combine it with an indirect-illumination image synthesized by the neural image generator. Employing neural scene representations in such scenarios enjoys a different set of constraints and conveniences than other (e.g. computer vision) problems. We exploit the main convenience—the existence of a 3D scene model—and provide the neural generator with geometry information (G-buffer) of novel views. We draw inspiration and architectures from works where a 3D model is not available; the focus is on extracting it [Esami et al. 2018; Sitzmann et al. 2019], but we leverage these architectures to complement the strengths of classical renderers, rather than replacing them. Finally, we demonstrate the benefits of compositional neural scene representations on the application of relighting: we create novel scene configurations by exchanging lighting partitions of different scenes.

Scope. Our method partitions the neural representation via explicit constraints enforced during training. The underlying neural model is non-probabilistic and relies on inputs from classical renderers. Nevertheless, the proposed mechanism for encoding adaptive, compressible neural representations is universal and applicable to other neural architectures and approaches.

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. We refer the reader to a recent survey by Tewari et al. [2020] for a more complete overview.

Neural scene representations. A recurring approach to render a scene using neural networks is to start 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. [2016] 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) [Esami 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 [Esami et al. 2018]. This significantly accelerates training (days instead of weeks) and improves accuracy.

Disentanglement and user control. Disentanglement has been presented as a key attribute of robust representations [Bengio et al. 2013] and interpretability [Chen et al. 2016], but also as crucial to controllable generation [Nie et al. 2020]. Multiple approaches have been explored to introduce user control, such as latent-space transformations [Nguyen-Phuoc et al. 2019; Olaszewski et al. 2019], or disentanglement of the latent variables using unsupervised [Chen et al. 2016; Higgins et al. 2017] or (semi-)supervised methods [Kulkarni et al. 2015; Nie et al. 2020]. Our work builds upon the 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 for 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 perturbation-based [Wagner et al. 2019; Zeiler and Fergus 2014] and gradient-based [Ancona et al. 2018; Shrikumar et al. 2017] methods; we use the latter 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]. Hybrid renderers have also gained traction in computer vision applications, such as video retargeting [Kim et al. 2018], due to their increased realism. Similarly to these approaches, we propose to 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 rendering techniques. We show examples where the image generator translates a G-buffer into a shaded image, or a G-buffer with direct illumination into an indirectly lit image only. Our model bears similarities with Deep Shading [Nalbach et al. 2017]; the main difference is the utilization of the neural scene representation (and the encoder) that provides complementary information to the G-buffer.

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, described in Section 4 and Section 5, pertain to the combination of these components, as well as the optimization and analysis of the resulting model.

The rest of this section discusses individual components of the model, the inputs, datasets, and loss functions used in our tests.

3.1 Scene encoder

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

The main component of the encoder is a convolutional neural network \( R \), which extracts a partial, high-dimensional scene representation \( r_i \) for each observation \( i \): \( r_i = R(c_i, i_i, g_i) \). We use the pool architecture [Eslami et al. 2018] for the encoding network \( R \). The network takes a tensor of concatenated image buffers \( (i_i, g_i) \) and processes them with strided convolutions. The viewpoint parameters \( c_i \) 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 \( r \), where \( k \) is the number of desired values in representations; see [Eslami et al. 2018] for details.

All partial representations are combined to obtain a (more) complete representation of the scene using an order-independent aggregator; we use componentwise averaging: \( r = \text{avg}(\{r_i\}_{i=1..n}) \). 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 \( c_v \), (ii) a G-buffer \( g_v \) rendered using a traditional renderer from the novel view \( v \), and (iii) a view-independent scene representation \( r \) extracted by the encoder.

Numerous architectures have been proposed for neural image synthesis. In what follows, we utilize two image-to-image translation models: the convolutional U-net architecture [Ronneberger et al. 2015] and the pixel generator [Sitzmann et al. 2019].

The main strength of the convolutional U-net is the large receptive field, i.e. the ability to source information from distant pixels in the G-buffer. In contrast, the pixel generator—being a simple multilayer perceptron—processes each pixel independently and cannot rely on pixel neighborhoods; this has been leveraged as a strength to ensure multi-view consistency [Sitzmann et al. 2019].

Please see the supplementary material for illustrations of the architectures, implementation details, and an expanded analysis of the strengths and weaknesses including also the GQN generator proposed by Eslami et al. [2018].
We use two datasets in our experiments. Each dataset consists of where the G-buffer contains also material and lighting information, well. In this paper, the decision of what surface features to include in the G-buffer is driven by the ease of analysis and interpretability of the 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 offscreen or occluded geometry is delivered to the generator only through the neural scene representation. Figure 3 demonstrates the impact of the scene representation on generated images.

The distinct separation of scene properties allows analyzing the flow of information inside the model. A more practical scenario, where the G-buffer contains also material and lighting information, is discussed in Section 6.1.

3.3 G-buffer Content

The encoder and the generator utilize G-buffers rendered from the observed locations and from the novel view, respectively. 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 driven by the ease of analysis and interpretability of the 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 offscreen or occluded geometry is delivered to the generator only through the neural scene representation. Figure 3 demonstrates the impact of the scene representation on generated images.

The distinct separation of scene properties allows analyzing the flow of information inside the model. A more practical scenario, where the G-buffer contains also material and lighting information, is discussed in Section 6.1.

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 color images (see Figure 4 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 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 are biased towards an apartment design. The variations are created by randomizing the luminaire and furniture placement, and lighting are biased towards an apartment design. The variations are created by randomizing the luminaire and furniture placement, and lighting are biased towards an apartment design.

Fig. 3. 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. 4. Random scenes from the PrimitiveRoom and the ArchViz datasets.

Fig. 3. 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. 4. Random scenes from the PrimitiveRoom and the ArchViz datasets.

4 COMPOSITIONAL SCENE REPRESENTATION

Our goal in this article is to impose a structure over the neural scene representation such that it respects the orthogonalities between individual scene properties. In Section 4.1, we apply the static partitioning approach of Kulkarni et al. [2015], and extend it to enable adaptive, compressive partitioning of the neural scene representation in Section 4.2. Our method adaptively disentangles the lighting, geometry, and material properties of scenes in one dataset, and stores them in disjoint partitions $R_M$, $R_L$, and $R_G$, respectively.

The adaptive partitioning has a number of advantages: (i) it respects the degree of variation of each component, (ii) it improves interpretability of the scene representation and provides insights into the inner mechanisms of different neural architectures—Figure 14, and (iii) it permits basic compositing operations, such as swapping lighting information between different scenes—Section 6.2.
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, $||\mathcal{R}_M|| = ||\mathcal{R}_L|| = ||\mathcal{R}_G|| = ||\mathcal{R}||/3$, 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 method of Kulkarni et al. [2015] to disentangle scene components by carefully averaging scene representations and adjusting 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. We may decide to randomize, for instance, the 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 to update the weights of the scene encoder.

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 5 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, e.g. the lighting partition occupies the first third of the representation. This constraint, which is not present in monolithic representations, is undesired. We now discuss a mechanism for optimizing the partition sizes during training.

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: $b^-(i) = ||\mathcal{R}|| \cdot \sum_{j=i}^{k-1} \sigma(s_j)$, and adding the size of the $i$-th partition: $b^+(i) = b^-(i) + ||\mathcal{R}|| \cdot \sigma(s_i)$, respectively. The softmax function $\sigma$ ensures that the learned sizes form a partition of 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:

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

where $S$ is the sigmoid function used to model the fuzzy boundaries; see Figure 6. The parameter $\alpha$ controls the spread of the sigmoid; we progressively increase $\alpha$ during training to approach a step function, 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 7 and 8. 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 geometry partition.
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 the pixel generator cannot rely on pixel neighborhoods in the same way as convolutional approaches. In general, the convolutional generators (U-net and GQN analyzed in the supplementary material) utilized the geometry partition less in all our experiments.

In Table 1, we show how the scene representation adapts to datasets with different complexity of lighting and materials. The “many lights” dataset features scenes with up to 10 randomly placed light sources in each scene. The optimization yields a larger lighting partition than in the “many materials” dataset, where scenes contain always a single light source, but the material complexity is doubled compared to the “many lights” dataset.

Table 1. Relative partition sizes of models trained on variations of the PrimitiveRoom dataset with different degrees of material and lighting variation.

<table>
<thead>
<tr>
<th></th>
<th>Many lights</th>
<th>Many materials</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lighting partition</td>
<td>39%</td>
<td>36%</td>
</tr>
<tr>
<td>Geometry partition</td>
<td>20%</td>
<td>23%</td>
</tr>
<tr>
<td>Material partition</td>
<td>37%</td>
<td>43%</td>
</tr>
<tr>
<td>Null partition</td>
<td>4%</td>
<td>4%</td>
</tr>
</tbody>
</table>

Fig. 7. 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.

Fig. 8. Same as in Figure 7 but trained on the AnciViz dataset.

Fig. 9. Average metrics of the pixel generator model with no partitioning, static partitioning, and our adaptive partitioning (lower means better).

Performance assessment. In order to analyze the penalties due to imposing static and adaptive partitioning mechanisms, we compare the quality of synthesized images using three error metrics: mean absolute percentage error (MAPE), PieAPP\(^1\) [Prashnani et al. 2018] and root mean square error (RMSE). Figure 9 shows the average performance for (i) a monolithic scene representation, (ii) statically partitioned representation, and (iii) our adaptively partitioned representation. The height of each bar corresponds to the mean value of the metric across 4000 test scenes that were generated using the same recipes as training scenes, but were excluded from training.

Partitioning the representation comes at a penalty in prediction accuracy; similar tradeoffs have been reported in prior work [Nie et al. 2020]. Static partitioning (dark blue) tends to yield the worst results as the static sizes of some partitions may be inappropriately small given the entropy of corresponding scene components. Our adaptive partitioning (green) mitigates this issue by optimizing the bit allocation in the representation, however, it still inherits the tradeoff in prediction accuracy due to enforcing disentanglement.

\(^1\)We first gamma-correct the high dynamic-range images (\(\gamma = 2.2\)), clip the values to range [0, 1], and quantize them using 8 bits per channel.
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 $R_0$, 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 $\beta (\|R\| - \|R_0\|)$ to the loss function, which is proportional to the number of dimensions in all other (non-null) partitions.

Analysis and Discussion. Figures 7 and 8 show the impact of the null partition on 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 null partition removes the burden of guessing the optimal size of the neural scene representation. One can over-allocate the space conservatively and rely on the optimization to grow the null partition appropriately. We confirm this in Figure 10 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. The tradeoff between the compression ratio and the loss of information can be controlled by adjusting $\beta$, as shown in Figure 11.

5 Attribution, Analysis, and Improvements

The partitioning of the scene representation allows us to attribute visual artifacts and poor image quality to specific information that is missing or incomplete in the representation. In this section, we utilize the gradient $\times$ input attribution method [Shrikumar et al. 2017] 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 $\times$ input products for individual elements and sum their absolute values to obtain a single attribution scalar.

5.1 Attribution of Partition Activations

In Figure 12, we attribute activations in partitions of the representation to the color, position, and normal channels of two encoded observations. The yellow, blue, and red color shades in the bottom row correspond to attributions of the lighting, geometry, and material partition, respectively. For each pixel, we first compute three attribution scalars—one per partition. Each scalar is then mapped to color shades of the corresponding partition, and the colors are added together. Brighter values indicate higher magnitude of attribution, white signifies high influence on all partitions.

Each partition is attributed to fairly large spatial regions, although certain regions are attributed more than others. For example, high intensities can be observed on the floor, while walls are typically darker. We interpret this as a strong signal that the model specializes to the biases of the dataset, in which most variations occur on the floor (e.g., due to random placement of the furniture and its shadows) and little information needs to be sourced from the walls (only textures and the position of the mirror frame).

The lighting partition (yellow shades) is mainly attributed to regions with visible emitters and to the floor. The emitter intensity needs to be inferred from the color channel, and its contribution can be inferred with the help of additional information in the position and normal channels. We hypothesize that shadow patterns on the floor (color channel), in conjunction with geometry information about the shadow casters (position and normal channels) contribute to the localization and intensity of emitters. In fact, these are the most prominent cues if the emitter is not directly visible in any of the observations. The relatively weak attribution to the wall emitter, specifically to the position and normal channels of observation 2.
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, and demonstrate that this balance may be specific to the architecture of the generator.

In the top row of Figure 14, we analyze the quality of synthesized shadows in a simple scene that contains a cylindrical shadow caster. In the left configuration, the cylinder appears in the camera frustum and is therefore captured by the G-buffer. The generators can rely on the G-buffer to identify its shape and position. In the right configuration, the cylinder is outside of the frustum—we observe only its shadow. The position and shape of the shadow caster can be inferred only from the neural scene representation.

The table below the images attributes the predicted colors in the red patches to (i) the partitions of the neural scene representation (top three rows) and (ii) the channels of the G-buffer—we used positions and normals in this experiment (bottom two rows). Each cell in the table was computed by summing the gradient \( \times \) input values across all elements of the respective partition (or the G-buffer channel) and averaging the sums across the pixels in the red patch. We normalized all values within each column and listed them as percentages to drive the focus on relative comparisons.

The table indicates that the pixel generator relies primarily on the neural scene representation (for pixels in the red patch) with the geometry partition being attributed the most (48%). In contrast, the U-net generator barely utilizes the geometry partition (4%) and sources information about geometry almost exclusively from the G-buffer (45% of all attribution). 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.
The results of this comparison are not entirely unexpected. Pixel generators, which access one pixel at a time, force the encoder to put more information about the geometry into the scene representation (as already suggested by Figures 7 and 8) and their performance is thus less impacted by the content of the G-buffer. The U-net generator—being convolutional—has a strong ability to source information from pixel neighborhoods in the G-buffer, but fails when the G-buffer lacks key information. This occurs whenever objects occlude each other or reflect surfaces not present in the view. The resulting artifacts are distracting, but do not appear sufficiently often during training to "teach" the U-net generator to utilize the geometry partition of the neural representation. We address this issue in the next section.

### 5.4 Auxiliary Generators

Our goal here is to force the scene encoder to extract information that may be underrepresented, but vital for good performance in certain situations. We employ an auxiliary generator [Philip et al. 2019] that is trained concurrently with the main generator, but focuses on the failure cases.

Specifically, to rectify the poor performance of the U-net generator on shadows due to invisible casters, we add a pixel generator that is optimized to produce shadows—grayscale images of (partial) visibility of light sources. The output of the auxiliary "shadow" generator is fed into the U-net generator. The loss function of the auxiliary 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 primarily the scene encoder of the original model; it keeps the loss of the original generator intact. Since both generators use the same scene representation during training, the auxiliary generator steers the encoder to extract the missing information. The second, and likely the key benefit in this specific case, is the shadow image that the auxiliary generator provides to the U-net. The right column in Figure 14 illustrates the improvement of the U-net generator after adding the auxiliary shadow (pixel) generator—the model draws more information from the geometry partition (the attribution increases from 10% to 17%) and succeeds in synthesizing the shadow.

Fig. 13. We attribute the colors of four patches in a generated image (left) to dimensions in the neural scene representation (middle), and to the color, position, and normal 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 shade, for instance, indicates that an observation pixel contributed more information about the geometry into the scene representation (through the geometry partition) to the generated patch; the brightness corresponds to the aggregate attribution.

Table: Attribution to neural representation

<table>
<thead>
<tr>
<th>In-view shadow caster</th>
<th>Out-of-view shadow caster</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Attrition to G-buffer</strong></td>
<td><strong>Attrition to representation</strong></td>
</tr>
<tr>
<td>light: 22%</td>
<td></td>
</tr>
<tr>
<td>geo.: 48%</td>
<td>43%</td>
</tr>
<tr>
<td>mat.: 13%</td>
<td>55%</td>
</tr>
<tr>
<td>pos.: 9%</td>
<td>8%</td>
</tr>
<tr>
<td>norm.: 8%</td>
<td>19%</td>
</tr>
</tbody>
</table>

Fig. 14. Shadow synthesis in a scene with a directly visible shadow caster (left pair) and with the same shadow caster being out of the view (right triple). The pixel generator and the U-net generator are both able to synthesize shadows of visible shadow casters. However, when the caster is out of the view, the U-net fails to produce the shadow since it relies primarily on the G-buffer. This is evidenced in the table below that lists relative attributions (gradient \times input) averaged over the marked pixels. Combining the U-net generator with an auxiliary pixel generator increases the attribution to the geometry partition, from 10% to 17%, and helps the U-net to correctly synthesize the shadow.
We also tested whether the auxiliary shadow generator can improve the performance of pixel generators. The increase in visual quality is subtle but still noticeable; Figure 15 shows three examples. The impact on the partitioning is more pronounced; Figure 16 and Table 2 confirm the growth of the geometry partition 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 (see the middle row of Figure 15). Adding an auxiliary generator thus does not necessarily lead to better overall results—quantitative metrics remained unaffected in this case—but rebalances the focus of the encoder.

6 APPLICATIONS
In this section, we explore two possible applications that (compositional) neural scene representations could improve in the future.

6.1 Rendering Indirect Illumination
One shortcoming of the neural model employed in this article (up to this point) is the rather poor visual quality on high-frequency visual features. However, the fine details and structures due to local illumination can be synthesized inexpensively using classical methods (if an accurate 3D model is available). The output from the classical renderer can be provided to the neural renderer as an additional input, or simply combined with the generated image. The two renderers can complement each other with the neural one focusing on the costly effects only.

We investigate one such scenario in Figure 17: we compute direct illumination via ray tracing and optimize the neural model to produce only indirect illumination. We augment the G-buffer to include material information (diffuse albedo and surface roughness); this improves material encoding and synthesis quality. Lastly, the ray-traced image is also input into the neural generator, allowing it to learn direct-to-indirect radiance transfer; this works remarkably well with convolutional architectures [Nalbach et al. 2017].

In our experiments, however, we used the pixel generator that upscaling the scene information (learned at low resolutions) to high resolutions more gracefully than the U-net generator (see the comparisons of different generators in the supplementary material). The test scenes feature notable interreflection between objects, e.g. between teapots and tables or between sofas and pillows, that are synthesized fairly well. Indirect illumination over long distances, such as the yellow tint on the ceiling (top row) due to light bouncing off the carpet and sofas, is reproduced by the neural model as well.

Our unoptimized implementation utilizing PyTorch for inference and accelerating tracing of rays using Optix required: 7 minutes to trace the direct-illumination buffer (8k spp), and 400 ms to predict the indirect-illumination; both at 1k×1k resolution using an NVIDIA RTX 6000 GPU. Path-tracing the indirect-illumination references (8k spp) required 25 minutes; we list this merely for completeness and leave comparisons to state-of-the-art GI renderers for future work.

6.2 Editing in Latent Space
We can exploit the compositionality of our neural scene representation to perform scene edits directly on the (latent) scene representation. The task of transferring information between latent codes of different scenes is challenging for monolithic representations that do not feature a clear separation of individual components. As long as the scene representation is appropriately partitioned, such edits become straightforward.

Table 2. Relative partition sizes for the models from Figure 15 and Figure 16 after one million of training iterations.

<table>
<thead>
<tr>
<th>Pixel generator + aux. shadow generator</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Lighting partition</td>
<td>25%</td>
<td>23%</td>
</tr>
<tr>
<td>Geometry partition</td>
<td>17%</td>
<td>28%</td>
</tr>
<tr>
<td>Material partition</td>
<td>53%</td>
<td>43%</td>
</tr>
<tr>
<td>Null partition</td>
<td>5%</td>
<td>6%</td>
</tr>
</tbody>
</table>
Compositional Neural Scene Representations for Shading Inference

Fig. 17. Neural synthesis of indirect illumination. The neural model is optimized to predict indirect illumination only from the neural scene representation, G-buffer w/ geometry and material information, and the direct illumination buffer. Results show that both local and distant interreflections are synthesized fairly accurately. Predictions of mirror reflections (teapot in top row and mirror in bottom row) suffer from overblurring. The supplementary material describes a possible remedy by including information about reflection rays in the G-buffer.

Fig. 18. Examples of relighting application, where the lighting partition of scene A replaces the lighting partition of scene B. The relighting result is shown in the right column.

In Figure 18, we replace the lighting partition in the representation of Scene B with the lighting partition of Scene A. We then feed the composited representation and the G-buffer of Scene B into the pixel generator. The generated image is shown in the third column; reference renderings are included below. The shadows and highlights are repositioned correctly in both studied cases.

One could also exercise the idea of transferring materials. Our model supports this as long as the two scenes have identical geometry, i.e. the mapping of materials to objects is identical in both scenes and transferring the material partition merely changes the material definition. When the scenes differ in geometry, material transfer requires deeper understanding of the content (e.g. semantic roles of individual surfaces) to map material sources to targets.

In Figure 19, 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; see the supplementary video for additional animations.

7 DISCUSSION AND FUTURE WORK

Implicit vs. explicit disentanglement. Neural representations tend to be naturally disentangled to some degree, as demonstrated with GQNs using scene algebra experiments [Eslami et al. 2018] or with VAEs by manipulating latent variables [Higgins et al. 2017]. These
work in the domain suffer from poor generalization as well. We
expect refined compositionality of the scene representation to play
a central role also in addressing generalization issues.

In this article, the content of individual partitions is orthogonal.
Having a clear separation between the scene components has two
distinct benefits. First, the explicit partitioning facilitates a straight-
forward attribution analysis that offers valuable insights into the
inner workings of different neural models (see Figure 14). Second,
it enables basic scene compositing operations, such as relighting
through the swapping of lighting partitions (Figure 18). It is worth
noting that explicit partitioning does not preclude discovering addi-
tional factors of variation using different methods. Striking a good
balance between the two is an interesting topic for future research.

Fine-grained compositionality. We partition the scene information
at a very coarse level: materials, lighting, and geometry. We view
this as the first step towards truly compositional, modular represen-
tations that will enable building complex scenes from smaller
modules. Such (hierarchical) scene representations will allow fine-
grained edits, for instance replacing the material of one specific
object with the material of another object, possibly from a different
scene. Enforcing compositionality at a finer level is an important
goal that we hope future work will achieve.

Generalization and challenging cases. Our model is capable of
extracting subtle clues from indirect illumination, e.g. to correctly
predict the material of an unobserved wall (see Figure 20). However,
it sometimes struggles with seemingly simple configurations (see
Figure 21), if they were not encountered during training. While
concerning, this limitation is not specific to our approach and other
works in the domain suffer from poor generalization as well. We
expect refined compositionality of the scene representation to play
a central role also in addressing generalization issues.

Validity of design decisions. We deliberately restricted the use of
classical rendering algorithms to a specific subset of image buffers.
This was done with the intent of facilitating the analysis of the
inner workings of the model, for instance by providing no material
and lighting information in the G-buffer when studying attribution
in Figure 12.

In Figure 17, we departed from the clear separation and added
material information to the G-buffer. The final image was obtained
by combining outputs from a classical and a neural renderer; we
expect such splits of the rendering task to be most rewarding in
the future. We also note that specular and glossy reflections can be
further improved by including parameters of perfect-reflection rays;
see the supplementary material for additional results.

Another decision worth revisiting is the extraction of the scene
representation from a set of image buffers. The observations have
the benefit of including all effects of global light transport that we
strive to synthesize, but other forms of input, e.g. omnimaps, voxel
representations, unstructured sets of radiance estimates, light fields,
or textual descriptions could suit specific applications better.

8 CONCLUSION

We extended the disentanglement technique of Kulkarni et al. [2015]
to respect the entropy of individual components and applied it to
neural scene representations to adaptively partition lighting,
geometry, and material information. By adding the null partition,
which serves as a reservoir of available bits during training, we
extract compressed neural representations of the scene.

Our experiments highlight the desirable properties offered by
adaptive partitioning, namely the enhanced interpretability and
control. Specifically, we were able to refine existing attribution
methods to reveal whether pixels in observations influenced the
final synthesized pixel values through the lighting, geometry, and/or
material information encoded in the scene representation. We also
studied the artifacts of different image generators, and propose to
fix poor performance by adding an auxiliary generator that steers
the scene encoding towards information that is otherwise missing
from the neural scene representation.

Yet, the results generated by most models in this paper suffer
from visual artifacts. We thus investigated a scenario, where the
neural renderer generates indirect illumination only to complement
a classical ray tracer; we believe complementing traditional graphics
algorithms, rather than replacing them, is a fruitful direction for
future research if production rendering is the ultimate goal.

Addressing the quality and scaling remains a priority for future
work as artifacts may not go away by merely increasing the size
of networks and datasets. We believe that imposing additional con-
straints and principles of light transport over neural models will be
necessary to yield realistically looking images. Understanding the
inner workings of a model will be paramount for injecting these
constraints; our work presents an important step in that direction.

Fig. 20. The model is capable of extracting the color of the orange wall from
indirect lighting in the observations when the wall is not observed directly.

Fig. 21. 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.
ACKNOWLEDGMENTS
We thank Jacob Munkberg, Jon Hasselgren, Marco Salvi, Pontus Andersson, Till Schnabel, and the anonymous reviewers for constructive feedback. We also thank Jaakko Lehtinen, S. M. Ali Eslami, Cengiz Öztireli, and Marco Ancona for fruitful discussions, and Jay-Artist, Wig42, dpamlin, Wolgraph, MZiemyx, SylkDrako, Tskuionyo, and CtrlArt MDL for models and textures.

REFERENCES


REFERENCES


ACM Trans. Graph., Vol. 39, No. 4, Article 135. Publication date: July 2020.