Neural Scene Graph Rendering: supplementary material

JONATHAN GRANSKOG, NVIDIA
TILL SCHNABEL, NVIDIA
FABRICE ROUSSELLE, NVIDIA
JAN NOVÁK, NVIDIA

This document provides additional information about the transformations, the datasets, and the renderer architecture used in the paper. It also analyzes the impact of geometry and material representation sizes, and discusses one alternative rendering architecture.

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

Additional Key Words and Phrases: rendering, neural networks, neural scene representations, modularity, generalization

ACM Reference Format:
Jonathan Granskog, Till Schnabel, Fabrice Rousselle, and Jan Novák. 2021. Neural Scene Graph Rendering: supplementary material. *ACM Trans. Graph.* 40, 4, Article 164 (August 2021), 5 pages. https://doi.org/10.1145/3450626.3459848

1 TRANSFORMATIONS

A number of geometry and material transformations are supported in our pipeline. In practice, we limit the magnitude of these transformations, as well as the number of combinations in a single training job. Networks struggle to learn transformations simultaneously if they allow for ambiguous interpretations, e.g. texture translation and geometric translation, even though each transformation could be learned robustly in isolation.

1.1 Geometry transformations

We support the following geometry transformations: translation, rotation, scaling, shearing, twirl, and bend. The number and ordering of transformations is not constrained.

Affine transformations (translation, rotation, scaling and shearing) are encoded from 4×4 matrices $\mathbf{Q}^i_{\mathrm{g},k}$, i.e. 16 parameters, into $d_\mathrm{g} \times d_\mathrm{g}$ matrices $\mathbf{T}^i_{\mathrm{g},k}$ using a single encoding network.

Twirl and bend transformations are encoded into the corresponding $d_\mathrm{g} \times d_\mathrm{g}$ matrix $\mathbf{T}^i_{\mathrm{g},k}$ from its input parameters with a unique encoder network each. Both transformations have a single parameter to set the magnitude; see Listing 1. We also experimented with displacement mapping and 3D twisting. The model did handle them similarly to the other deformations, but as they were not included in any of the results, we do not detail them.

Authors' addresses: Jonathan Granskog, NVIDIA, jgranskog@nvidia.com; Till Schnabel, NVIDIA, tschnabel@nvidia.com; Fabrice Rousselle, NVIDIA, frousselle@nvidia.com; Jan Novák, NVIDIA, jnovak@nvidia.com.

© 2021 Association for Computing Machinery.
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*, https://doi.org/10.1145/3450626.3459848.

Listing 1. Bend and Twirl transformations in GLSL

```
vec3 rotateZ(in vec3 p, float t) {
    float co = cos(t);
    float si = sin(t);
    p.xy = mat2(co, -si, si, co) * p.xy;
    return p;
}
vec3 bend(in vec3 p, float k) {
    return rotateZ(p, k*p.x);
}
vec3 twirl(in vec3 p, float k) {
    return rotateZ(p, length(p) * k);
}
```

1.2 Texture transformations

We experimented with translation, rotation, scaling, shearing, coloring, grayscale, and hue shift transformations. Translation, rotation, scaling, and shearing are encoded and applied exactly like the affine geometry transformations except that the encoder network uses different weights and the output matrix $\mathbf{T}^i_{\mathrm{m},l}$ is of size $d_\mathrm{m} \times d_\mathrm{m}$.

Every remaining texture transformation is processed using a distinct encoder network. Coloring multiplies the current diffuse or volumetric color by the three-dimensional RGB input color. We utilize this transformation whenever the color appearance of a material is adjusted. The grayscale transformation blends between grayscale and colored versions of the texture, and the hue shift transformation operates in the HSV color space; both are controlled by a single parameter.

2 OPTIMAL SIZE OF LEARNED REPRESENTATIONS

In the article, we used 32-dimensional geometry and material vectors. Here we study the impact of the size of the vectors on the visual quality.

Geometry representation. For the first experiment, we generated 128×128 images with up to four objects. Each object is one of three geometries: bunny, teapot, and monkey head. The objects are randomly translated, rotated, and scaled (the order is random), and always assigned the diffuse material with a randomized RGB color transform. We run seven different experiments, varying the size d_g of the geometry vector between 1 and 64 (in powers of 2). The size of the geometry transformation matrices is adjusted accordingly. The size of the material vector is set to $d_\mathrm{m} = 32$ in all experiments. Figure 1 shows the visual accuracy for the different experiments. We observe that, as the representation size diminishes, all shapes regress to similar blobs.

Material representation. We similarly tested the impact of the material representation size. For these experiments, we generated 256×256 images with up to four objects. Each object is a square with a randomly assigned texture; we use 16 textures in total. The

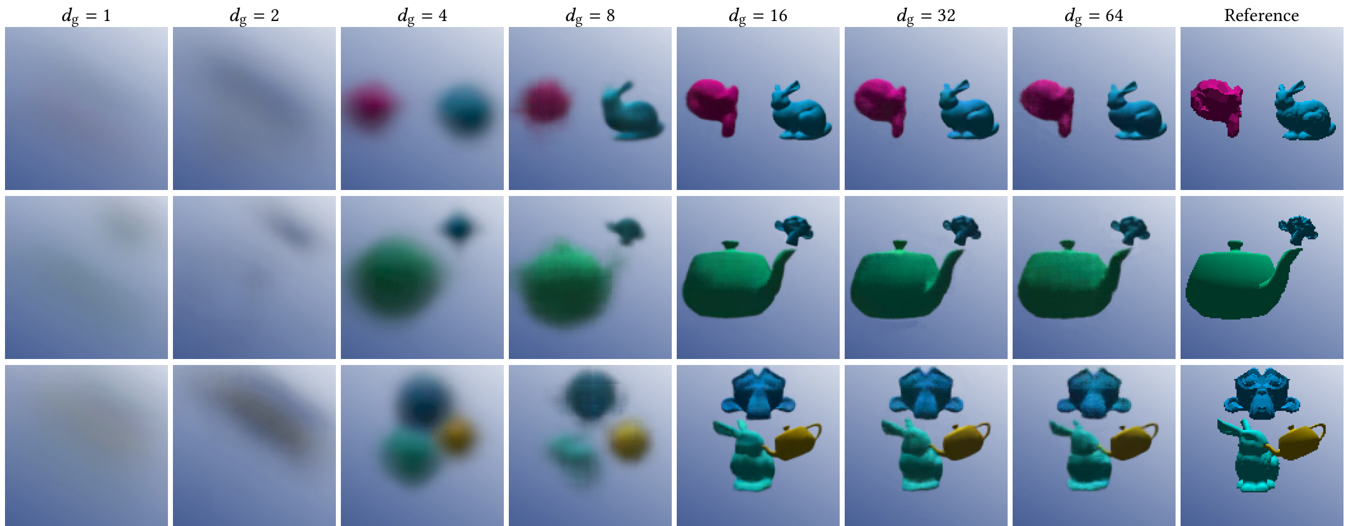

Fig. 1. Impact of the size d_g of the geometry representation on the quality of rendered results. The results with $d_g = 32$ feature highest quality; we used this configuration to generate all results in the main article and in the other figures in this supplementary document.

Fig. 2. Impact of the size d_m of the material representation on the quality of rendered results. The results with $d_m = 32$ feature highest quality; we used this configuration to generate all results in the main article and in the other figures in this supplementary document.

square is randomly translated and scaled. The textures are altered by the following randomized material transformations: x-axis flip, rotation, and hue shift. Figure 2 shows the impact of the size d_m of the material representation; the size of the geometry representation was held fixed $d_g = 32$. The most notable impact of reducing the material representation size is the gradual loss of color and rotational information.

Summary. In Table 1, we provide a quantitative summary for the experiments in Figure 1 and Figure 2. The reported metrics are averaged over 1000 rendered images of random scenes.

3 QUANTITATIVE EVALUATION

Table 2 provides error metrics for two of the models used to generate results in the paper: the one from Figure 3 (shape deformation), and the one from Figure 10 (volleyball animation).

For the first model, we used a signed distance field (SDF) renderer and training images had up to four random shapes (square, triangle, star or pentagon) at resolution 256 × 256 with 4 samples per pixel. The shapes were transformed using twirl, bend, scaling, rotation, translation (always in this order). We use a single canonical material of constant color, which is modified using a random color transformation; see Figure 3. Our loss combines the negative log likelihood

Table 1. Comparison between different vector/matrix sizes for geometry and materials after 1M iterations of training averaged over 1000 random scenes.

		PSNR ↑	SSIM ↑	LPIPS ↓	ꟻLIP ↓
Material size d_{m}	1	20.55	0.867	0.185	0.141
	2	21.59	0.882	0.154	0.127
	4	26.15	0.933	0.080	0.083
	8	29.56	0.968	0.038	0.060
	16	29.43	0.966	0.041	0.062
	32	**29.89**	**0.968**	**0.037**	0.060
	64	28.86	0.960	0.048	**0.057**
Geometry size d_{g}	1	18.25	0.820	0.523	0.285
	2	18.40	0.820	0.521	0.274
	4	19.65	0.823	0.471	0.211
	8	21.49	0.835	0.375	0.160
	16	**28.92**	**0.942**	0.076	0.054
	32	28.77	0.939	**0.073**	**0.053**
	64	28.43	0.934	0.082	0.056

Table 2. Quantitative evaluation of models used in Figure 3 and Figure 10 in the main article. Metrics are averaged over 1000 random scenes.

	PSNR ↑	SSIM ↑	LPIPS ↓	ꟻLIP ↓
Figure 3	35.91	0.984	0.007	0.050
Figure 10	36.41	0.982	0.007	0.049

(NLL) with the VGG loss: NLL + 0.5× VGG. The learning rate was lowered from 10^{-4} to $\sqrt{10}\times 10^{-5}$ after 400k iterations. We used 940k iterations in total.

For the second model, training images had up to three random objects (sphere or torus) at 256×256 resolution with 4 samples per pixel. The shapes were transformed by scaling, rotation, and translation. The material is either diffuse or volumetric with a random color transformation. The scene is lit by a point light, the ground and the sky are static images. As loss we used: NLL + 0.3× VGG. The learning rate was lowered from 10^{-4} to $\sqrt{10} \times 10^{-5}$ after 800k iterations. We used 1.58M iterations in total.

4 RENDERER ARCHITECTURE

The foundations of our scene representations are the geometry and material representations. Except for Figure 1 and Figure 2 in this document, we used 32-dimensional vectors to represent each shape and material. Thus, every geometry and texture transformation matrix is 32×32. The transformed geometry and material representations are concatenated into a 64-dimensional vector which describes the geometry and appearance of a single object in the scene.

These 64-dimensional vectors are passed to the preprocessor network, which is a convolutional upsampling network. We use 4×4 deconvolutions with stride 2 and padding 1, and ReLU activation functions after each deconvolution. Table 3 lists the number of filters at each level of the preprocessor.

The filters used by the LSTM cells, which use 5×5 convolutions with stride 1 and padding 2, always match the number of filters used by the last layer of the preprocessor.

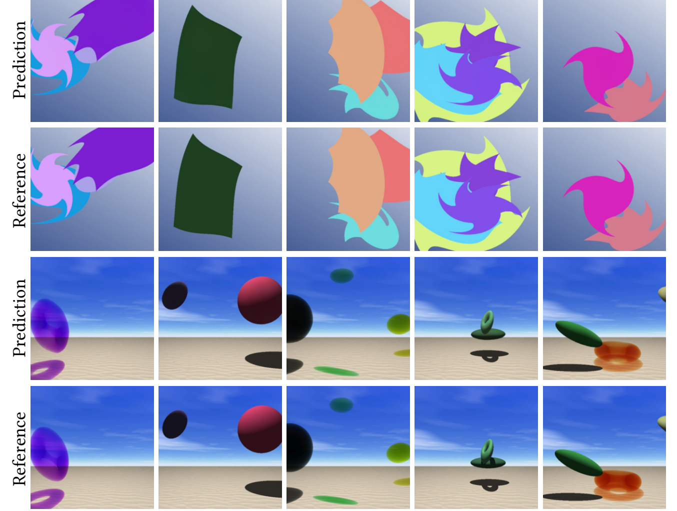

Fig. 3. Random scenes from training datasets of models used in Figure 3 and Figure 10 of the main article.

The last network—the pixel generator—converts the LSTM state into an image by processing the state using three 1×1 convolutional layers. Here, we also use ReLU activation functions, and the number of filters matches the LSTM cells except in the last layer where it flattens the feature map into an RGB image.

4.1 Progressive growing

The aforementioned architecture works well at low resolutions (up to 256×256 pixels), but at higher resolutions, the training iterations become prohibitively expensive and the memory footprint excessive. We thus also tested the idea of progressively growing the model during training, originally proposed for GAN architectures [Karras et al. 2018]. While the progressive growing approach worked better in some of our tests, it occasionally performed worse than training directly at the target resolution. We thus opted for simplicity and did not utilize it in any of the results. We still discuss it here as it should facilitate higher resolutions in the future.

We start at $h \times w = 4 \times 4$ resolution with $c = 256$ filters per layer, and progressively double h and w while reducing c to roughly maintain a constant evaluation time. The filter numbers and the sizes of training batches for each resolution are summarized in Table 3. We increase the resolution after every 800 000 entries. At each growing step, we add another deconvolutional layer and ReLU-activation to the preprocessor, initialize a new convolutional LSTM cell, and create a new output pixel generator. These new components provide a new path for generating the image. Following the recommendation of Karras et al. [2018] to prevent sudden shocks to the model, we blend the result from the new and old path over the next 800 000 entries. The old path is then dropped and the growing step repeated to advance to the next resolution level.

Table 3. The number of filters used for different resolutions. The right-most column reports the batch size used when progressively growing the model (Section 4.1).

Resolution $h \times w$	Number of filters c	Batch size
2×2	256	32
4×4	256	32
8×8	256	32
16×16	256	32
32×32	256	32
64×64	128	32
128×128	64	32
256×256	32	16
512×512	32	4

Table 4. Comparison between the streaming renderer and the per-pixel reading model on 1000 random scenes.

	PSNR ↑	SSIM ↑	LPIPS ↓	ꟻLIP ↓
Streaming model	30.52	0.967	**0.010**	**0.058**
Reading model	**30.65**	**0.968**	0.011	0.059

5 PER-PIXEL READ ARCHITECTURE

Our method uses a recurrent convolutional architecture to render objects from the neural scene representation. Many recent articles present image generation networks that process pixels independently [Anokhin et al. 2020; Granskog et al. 2020; Mildenhall et al. 2020; Sitzmann et al. 2019].

We experimented with a per-pixel architecture that *reads* from the scene representation independently for every pixel. First, for each pixel, we process the scene objects separately using a gated recurrent unit (GRU): we concatenate the pixel position encoded using 2D Fourier features [Mildenhall et al. 2020] to the current state vector and the object representation. Once all objects are processed, we aggregate the outputs of the GRU using max pooling and pass it to another GRU to produce a new state; see Figure 4. The renderer performs this operation eight times before producing the final state. Finally, a fully-connected network maps the final state to the pixel color.

In most tests, the reading and streaming architectures produced results of similar quality (see Table 4), but the reading architecture required longer training times. Still, the per-pixel architecture tends to better capture interactions between objects (see Figure 5 for an illustration). We hypothesize that the streaming approach makes it challenging for the model to learn to modify the color of objects after they have been drawn due to the sequential processing. The reading operation, on the other hand, can read from multiple object representations simultaneously via the max-pooling operation. Additionally, it can perform reading operations with different weights multiple times in sequence, which increases its flexibility even further. If its cost could be reduced, the per-pixel architecture would offer a compelling alternative.

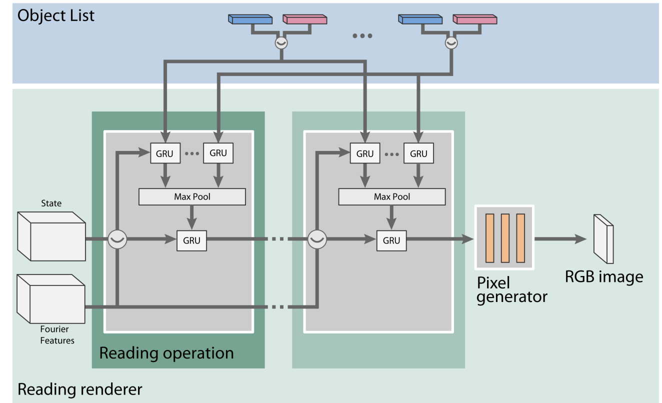

Fig. 4. The per-pixel-read architecture performs multiple reading operations, which aggregate object representations independently for each pixel, in sequence to produce a final state feature map. The final features are then compressed into the final output image with a pixel generator similar to the streaming renderer.

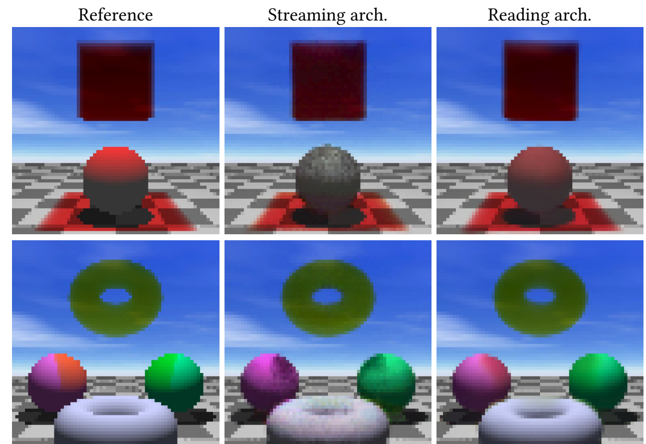

Fig. 5. The streaming neural renderer fails to cast volumetric shadows onto other objects. The per-pixel reading architecture is more expensive to evaluate but can render the shadows correctly.

6 BOOLEAN OPERATIONS

For the extension to arbitrary boolean operations between geometries, we use a simpler form of a graph, where each inner node can have *at most* one child be another inner node. This allows us to feed each boolean operation into the renderer with one of the scene objects. We support four types of boolean operations: union, smooth union, subtraction, and intersection.

In the renderer, we concatenate the output of the preprocessor with the corresponding boolean operation. It is then the responsibility of the recurrent block to interpret the boolean operation correctly, i.e. to combine the previous LSTM state with the current input. For the union operation in 2D, the LSTM has to function like a normal z-buffer by drawing more recent shapes over the previous

ones. For other operations, the LSTM has to learn to alter the previous objects, e.g. by creating smooth transitions, or removing parts. Our results show that the LSTM can learn to synthesize scenes made with these boolean operations.

REFERENCES

Ivan Anokhin, Kirill Demochkin, Taras Khakhulin, Gleb Sterkin, Victor Lempitsky, and Denis Korzhenkov. 2020. Image Generators with Conditionally-Independent Pixel Synthesis. *arXiv e-prints*, Article arXiv:2011.13775 (Nov. 2020), arXiv:2011.13775 pages. arXiv:cs.CV/2011.13775

Jonathan Granskog, Fabrice Rousselle, Marios Papas, and Jan Novák. 2020. Compositional Neural Scene Representations for Shading Inference. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 39, 4 (July 2020).

Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. 2018. Progressive Growing of GANs for Improved Quality, Stability, and Variation. In *International Conference on Learning Representations*.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *ECCV*.

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