Neural Scene Graph Rendering

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

We present a neural scene graph—a modular and controllable representation of scenes with elements that are learned from data. We focus on the forward rendering problem, where the scene graph is provided by the user and references learned elements. The elements correspond to geometry and material definitions of scene objects and constitute the leaves of the graph; we store them as high-dimensional vectors. The position and appearance of scene objects can be adjusted in an artist-friendly manner via familiar transformations, e.g. translation, bending, or color hue shift, which are stored in the inner nodes of the graph. In order to apply a (non-linear) transformation to a learned vector, we adopt the concept of linearizing a problem by lifting it into higher dimensions: we first encode the transformation into a high-dimensional matrix and then apply it by standard matrix-vector multiplication. The transformations are encoded using neural networks. We render the scene graph using a streaming neural renderer, which can handle graphs with a varying number of objects, and thereby facilitates scalability. Our results demonstrate a precise control over the learned object representations in a number of animated 2D and 3D scenes. Despite the limited visual complexity, our work presents a step towards marrying traditional editing mechanisms with learned representations, and towards high-quality, controllable neural rendering.

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 N. Schnabel, Fabrice Rousselle, and Jan Novák. 2021. Neural Scene Graph Rendering. *ACM Trans. Graph.* 40, 4, Article 164 (August 2021), 11 pages. https://doi.org/10.1145/3450626.3459848

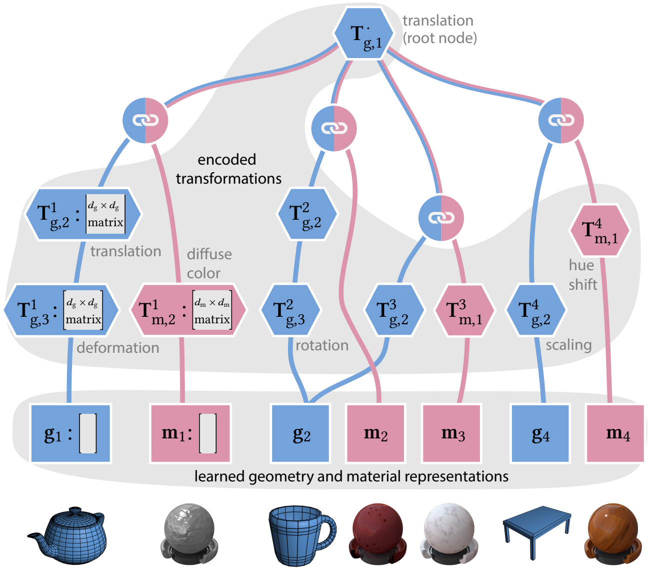

Fig. 1. Our scene graphs consist of leaf nodes (drawn as rectangles) that contain learned vectors storing geometry (blue) and materials (pink), and interior nodes that hold transformations (hexagons), and combine them into object instances (blue-pink circles). Symbols and indices are explained in Section 4 and 5. The geometry and material vectors, as well as encoding of user-defined transformations and deformations, are learned from data.

1 INTRODUCTION

In recent years, computer-vision algorithms have demonstrated a great potential for extracting scenes from images and videos in a (semi-)automated manner [Eslami et al. 2018]. The main limitation, common to most of these techniques, is that the extracted scene representation is monolithic with individual scene objects mingled together. While this may be acceptable on micro and meso scales, it is undesired at the level of semantic components that an artist may need to animate, relight, or otherwise alter.

Compositionality and modularity—patterns that arise naturally in the graphics pipeline—are key to enable fine control over the placement and appearance of individual objects. Classical 3D models and their laborious authoring, however, are ripe for revisiting as deep learning can circumvent (parts of) the tedious creation process.

We envision future renderers that support graphics and neural primitives. Some objects will still be handled using classical models (e.g. triangles, microfacet BRDFs), but whenever these struggle with realism (e.g. parts of human face), fail to appropriately filter details (mesoscale structures), or become inefficient (fuzzy appearance), they will be replaced by neural counterparts that demonstrated great potential. To enable such hybrid workflows, compositional and controllable neural representations need to be developed first.

We present one such attempt: a modular scene representation that leverages learned object-level representations and allows transforming them using familiar editing mechanisms. Our goal is to fulfill the following constraints: i) geometry and materials shall be treated as orthogonal and represented such that, ii) the artist can transform the representations via linear and non-linear editing operations, and iii) the number of objects and the overall composition can be directly controlled. We propose to accommodate these constraints using two simple, yet powerful concepts.

Authors' addresses: Jonathan Granskog, NVIDIA, jgranskog@nvidia.com; Till N. Schnabel, NVIDIA, till@familie-schnabel.de; Fabrice Rousselle, NVIDIA, frousselle@nvidia.com; Jan Novák, NVIDIA, jnovak@nvidia.com.

First, we lift the representations of the scene elements and transformations to a high-dimensional space; we borrow this concept from support vector machines that linearize classification problems. We represent geometry and materials using vectors, and transformations using matrices. Non-linear transformations, once lifted to this high-dimensional space, can be performed using standard linear algebra, i.e. vector-matrix multiplication. We obtain the high-dimensional transformation matrices using neural networks: we input the parameters of an editing operation into the network, which outputs the matrix weights. The network is optimized such that multiplying the matrix with any of the learned geometry (or material) vectors yields the desired effect (e.g. shape translation, surface deformation, or texture recoloring). To create a specific scene, we allow the artist to organize the geometry and material vectors, and the transformation matrices, into a *neural scene graph*: the vectors are stored in leaf nodes, and the transformation matrices reside in inner nodes altering the vectors inside the subgraph.

Second, we propose to render the scene graph using a streaming neural renderer. The renderer processes objects sequentially, without imposing a hard limit on their number, while correctly resolving visibility and shading. We demonstrate scalability and artistic control on animated 2D and 3D scenes; animations are provided in the supplementary video. Our approach preserves the fundamental concepts of scene authoring, while sidestepping the minutiae of evolving complex models from basic primitives; this task is replaced by learning from data.

In this article, we focus on the forward rendering problem and assume the scene graph is provided by the user. We *do not* present a technique for handling complex, real-world environments; we are currently far from that, dealing with scenes reminiscent of the early years of computer graphics. Our present goal is to provide a scalable solution that handles environments with *arbitrary numbers* of (simple) elements, each represented by a neural representation that can be further transformed and manipulated. Our focus is therefore on the modularity and controllability of the neural scene representation. While our work does not unlock any immediate applications, we believe such representations will play a central role in bringing neural rendering to the practical realm.

2 RELATED WORK

Developing efficient *neural* methods for representing and manipulating scenes is an active area of research. We now discuss the most related work on neural scene representations and compositionality of learned representations; see the report by [Tewari et al. 2020] for a thorough review.

Neural scene representations. A popular representation that aligns well with convolutional architectures are voxel grids [Lombardi et al. 2019; Nguyen-Phuoc et al. 2019, 2020, 2018; Olszewski et al. 2019; Rematas and Ferrari 2020]. For example, Nguyen-Phuoc et al. [2018] input a single grid representing an object to a network to render novel views. Similarly, Rematas and Ferrari [2020] improve visual fidelity by learning to render global illumination for scenes consisting of an area light, a ground plane, and a single 3D object. Lombardi et al. [2019] render more complex objects, such as human faces, with learned warped volumetric grids.

Tatarchenko et al. [2016] encode a rendered image of an object into a latent representation such that the object can be rendered from other views. Similarly, Eslami et al. [2018] utilize multiple images to produce a view-invariant representation for novel view synthesis of static scenes composed of multiple objects. Granskog et al. [2020] further partition the representation into material, lighting, and geometry components and, similarly to neural textures [Thies et al. 2019], rely on a classical rasterizer to resolve primary visibility and achieve sharp predictions.

Implicit representations, such as signed distance fields, are used in many recent works [Mescheder et al. 2019; Niemeyer et al. 2020; Oechsle et al. 2019, 2020; Park et al. 2019; Sitzmann et al. 2019]. Neural radiance fields (NeRF) [Mildenhall et al. 2020] rendered via ray marching represent a promising direction for both static and dynamic scenes [Guo et al. 2020; Li et al. 2020; Liu et al. 2020; Ost et al. 2020; Park et al. 2020; Pumarola et al. 2020; Xian et al. 2020].

Point cloud representations have also been used, e.g. for learning light transport [Hermosilla et al. 2019; Sanzenbacher et al. 2020], and novel view synthesis [Aliev et al. 2020].

In contrast to the aforementioned approaches utilizing traditional graphics representations and rendering algorithms, we rely solely on neural rendering and abstract vector representations, much like the approach of Eslami et al. [2018] and its derivatives. This design choice comes at the cost of reduced rendering quality, but it facilitates developing a general approach, which can be combined with stronger graphics priors in the future.

Compositionality and controllability. Generalization, scalability, and artistic control belong to actively sought features of neural representations [Tewari et al. 2020] as many of the aforementioned methods offer these in limited degree; we list a few examples here. Both Nguyen-Phuoc et al. [2018] and Rematas and Ferrari [2020] allow camera and lighting changes in single-object scenes. The generative method of Chen et al. [2021] also includes control over shape and appearance. Eslami et al. [2018] use scene "algebra" to perform basic scene manipulations, albeit with a rather coarse control over the edits. The extension by Granskog et al. [2020] allows changing also the appearance and lighting, but only on scene level; altering individual objects is not possible. Thies et al. [2019] can apply learned neural textures to new scene objects. Nguyen-Phuoc et al. [2019] and Olszewski et al. [2019] allow scaling and rotation of neural grids. Liu et al. [2019] propose scene programs, a syntax for building scenes, to enable controllable neural image generation.

Latent representations in image generation networks, such as in generative adversarial networks [Goodfellow et al. 2014], lack explicit control and interpretability. As such, many works [Chen et al. 2016; Karras et al. 2019, 2020; Kulkarni et al. 2015; Nie et al. 2020] learn to disentangle latent representations to improve interpretation and stylization, but rarely feature precise control over placement and the disentanglement may be only partial. Härkönen et al. [2020] extract principal directions of latent representations in pre-trained networks to discover useful axes of control.

The limited degree of control and scalability in most prior works is because these features are often not the primary goal; they appear as a byproduct of pursuing a different target.

Scene graphs in neural rendering. Prior works have applied scene graphs, containing object classes and relational descriptors, to realistic 2D image generation [Ashual and Wolf 2019; Herzig et al. 2020; Ivgi et al. 2020; Johnson et al. 2018]. BlockGAN [Nguyen-Phuoc et al. 2020] synthesizes images of novel views from a neural voxel grid scene representation, which is computed by aggregating object-specific grid representations. Transformations can be applied to the bounding boxes of object representations before aggregation to adjust their location. Ehrhardt et al. [2020] further model inter-object relationships with an additional interaction module.

Concurrent to our work, Ost et al. [2020], Guo et al. [2020] and Niemeyer and Geiger [2021] propose to decompose a scene into multiple neural radiance fields [Mildenhall et al. 2020], and manipulate them by transforming their bounding boxes. Similar to BlockGAN, this approach is powerful as it allows leveraging traditional 3D transformations on top of a learned object representations.

We pursue the same goal of direct control over the placement of scene elements, but in a different setup: we do not prescribe a specific interpretation (such as voxel grids or signed-distance fields) over the geometry representation. We also strive for a wider gamut of linear and non-linear transformations and propose a unified approach based on linearly transforming high-dimensional embeddings.

3 CLASSICAL SCENE GRAPHS

Scene hierarchies—scene graphs—are often used to compose complex scenes from basic geometric primitives, transformations, or material definitions. In order to relate our neural model to existing scene graphs, we first review a specific type of scene graphs here and describe how our approach maps to it in the next section.

We assume a directed acyclic scene graph where geometry and appearance data is stored in outer (terminal) nodes; see Figure 1. All outer nodes are reachable by traversing the graph from a single *root* node representing the entire scene. An inner node represents either a *geometry transformation*, *material transformation*, or linking of a geometry to a material. Branching occurring at a transformation node (e.g. root node) corresponds to geometries or materials sharing the same transformation. Merging of edges at a child node (e.g. node $\mathbf{g}_2$) indicates instancing: the same subgraph is duplicated multiple times in the scene using different transformations.

Preparation for rendering. In order to render the scene graph, we visit all outer nodes performing the following steps for each of them:

(1) We identify each unique path from the outer node to the root node of the graph.
(2) For each path, we collect transformation nodes in order of their appearance on the path, compute the total transformation, and apply it to the outer node.
(3) Geometry and material paths that are merged using a link node (illustrated by blue-pink circles and edges in Figure 1) are input into the renderer together.

This procedure yields a list of transformed geometry-material pairs that will be passed to the renderer to produce the final image. For the illustration in Figure 1, the list would contain a teapot, two instances of a mug, and a table, each with a different material.

The main contribution of this article is the application of the scene graph concept to neural scene representations.

4 NEURAL SCENE GRAPH

In this section, we detail the representations used in the outer and inner nodes of the neural scene graph. We do not prescribe any specific interpretation of the geometry and material representations, e.g. meshes, voxel grids, or textures. Instead, we consider shapes and materials as points in high-dimensional spaces (Section 4.1). We rely on linear algebra in the high-dimensional space to apply classical shape and material transformations (Section 4.2), which may themselves be non-linear in 2D and 3D. Specifically, in order to transform a d-dimensional representation, we multiply it by an encoded $d \times d$ transformation matrix. The optimization of the geometry and material representations and neural networks, which encode the transformations, is described in Section 4.3.

4.1 Outer nodes

We use a common abstract representation for both geometry and material elements of the scene: points in high-dimensional spaces, stored as vectors. The representation is dissociated from any specific geometric interpretation, like triangle meshes; we let the model form its own interpretation. Using such abstract representations provides a unified handling of geometry and materials, and a convenient way of assigning a material to a geometric shape by merely concatenating their vectors; we use 32-dimensional vectors in all our experiments.

Geometry. An outer geometry node stores a learned representation of a *canonical* geometric shape. The representations of individual canonical shapes are extracted from the training dataset. In our experiments, we use relatively simple geometries (sphere, torus, teapot, etc.), but the canonical elements can theoretically be as complex as a human face in the neutral pose, which is then deformed or animated using transformation nodes of the graph. Each canonical representation is encoded using a vector $\mathbf{g} \in \mathbb{R}^{d_\mathrm{g}}$.

Materials. Visual appearance of an object is typically described using one of the many surface and volumetric models. These can range from low-parametric, physically based scattering functions that are driven by textures like the Burley BSDF [Burley 2012] to high-dimensional texture functions (BTFs). We encode each unique material *class* (e.g. plastic material, volume, textured diffuse) using a vector, $\mathbf{m} \in \mathbb{R}^{d_\mathrm{m}}$, where d_m specifies the dimensionality.

Figure 2 demonstrates the orthogonality of geometry and materials. The first two rows show two canonical geometry vectors $\mathbf{g}$ that are assigned three different canonical materials: diffuse, glossy, and volumetric (columns). The appearance is further adjusted by applying a material transformation that we discuss next. The third row shows multiple objects in one scene.

4.2 Inner nodes

Our scene graphs can contain two types of inner nodes: n-ary nodes that transform (possibly multiple) geometry or material vectors (hexagons in Figure 1), and binary nodes that link geometry to material (blue-pink circles). We will now detail the transformation nodes and the encoding of the transformations.

Transformations. Our goal is to let the user alter the geometry and material representations using familiar transformations, such as translation and rotation of geometry, and re-coloring of diffuse

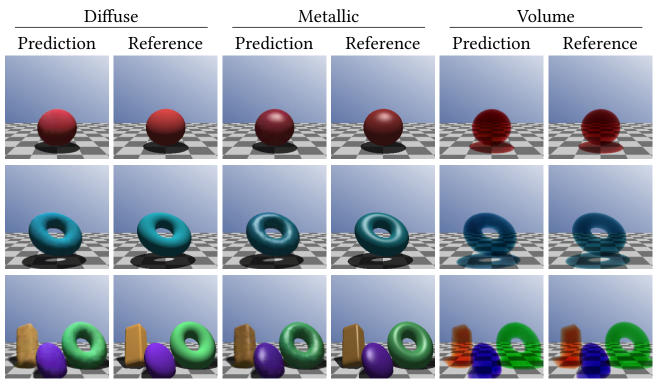

Fig. 2. Combinations of (learned) geometry $\mathbf{g}$ and material $\mathbf{m}$ representations rendered with our neural renderer (odd columns) and a reference renderer (even columns). Each material representation is left-multiplied by a $d_{\mathrm{m}} \times d_{\mathrm{m}}$ color-transformation matrix to alter the base color of the canonical material. The bottom row shows scenes with multiple geometries that are assigned the same canonical material but different color transformations.

textures. To unify handling of arbitrary transformations, we choose to represent each transformation using a high-dimensional matrix, and we apply the transformation by *left-multiplying* the material/geometry column vector by the matrix. More formally, transforming a representation vector $\mathbf{x}$ using a parametric transformation f with parameters γ is defined as:

$$f(\mathbf{x};\gamma) := \mathbf{T} \cdot \mathbf{x}, \quad \text{where } \mathbf{T} = h(\gamma;\theta).$$

The transformation matrix $\mathbf{T} \in \mathbb{R}^{d \times d}$ is obtained using an encoding function h.

Encoding. The parameters θ of the encoding function $h(\cdot;\theta)$ are optimized such that the matrix-vector product $\mathbf{T} \cdot \mathbf{x}$ corresponds to applying the transformation to the learned representation. We use an ensemble of neural networks to represent h; one for each distinct set of transformations.

For instance, all classical 3D projective transformations that can be represented using 4×4 matrices (we use translation, rotation, scaling, and skewing) are encoded using the same neural network. This network takes the flattened 4×4 matrix $\mathbf{Q}$ as input and produces the corresponding high-dimensional matrix: $\mathbf{T}_{\mathrm{proj3D}} = h_{\mathrm{proj3D}}(\mathbf{Q};\theta)$. Deformations, such as twirl and bending, that have either a very unique visual impact and/or a different set of input parameters are encoded using their own dedicated networks. Material transformations are handled analogously. A complete list of the transformations is provided in the supplementary material.

All encoding networks are fully-connected perceptrons with three layers that have d, d^2, and d^2 neurons, respectively. The input layer has dimensionality matched to the size of the parameters set $|\gamma|$. The output layer contains d^2 values of the linearized matrix $\mathbf{T}$. For geometric transforms $d := d_{\mathrm{g}}$; for material transforms $d := d_{\mathrm{m}}$. Since we use 32-dimensional material and geometry representation vectors in all our experiments, i.e. $d_{\mathrm{m}} = d_{\mathrm{g}} = 32$, the encoding function h always outputs a 32×32 transformation matrix $\mathbf{T}$. See the supplementary material for comparisons of different vector sizes.

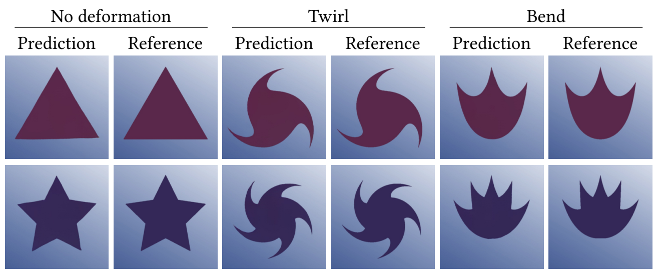

Fig. 3. Examples of user-defined edits—twirl and bend—whose parameters are encoded using a neural network into a transformation matrix that left-multiplies the triangle and star geometry representations.

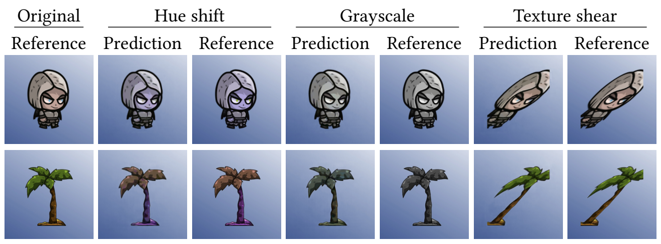

Fig. 4. Material transformations (hue shift, grayscale, shearing) applied to two material vectors (rows), each representing a different texture.

Figure 3 shows two *geometric* deformations applied to two 2D shapes. Figure 4 shows three *material* transformations, that change colors and mapping of the textures; each texture is represented by a dedicated material vector.

4.3 End-to-end training

As we aim to eventually sidestep the tedious authoring of scene assets, e.g. by extracting canonical representations from photographs, we require an end-to-end training procedure. This in turn requires a differentiable renderer; we propose one in Section 5.

To extract the canonical representations and to optimize the encoders and the renderer, we iterate the following procedure. We start with a target image and a corresponding scene graph. We encode all transformations in the graph and apply them to the representations stored in the outer geometry and material nodes. The transformed representations are input into the renderer, and the rendered image is compared to the target. The resulting error gradients are then propagated back to adjust the parameters of the renderer, transformation encoders, and the values of canonical representations present in the scene. We detail the optimization in Section 6.

4.4 Discussion

Our choice of representing 2D and 3D (non-linear) transformations using a linear operation—matrix multiplication—exploits the fact that non-linear, low-dimensional transformations can be approximated well by linear transformations in higher-dimensional spaces; our reasoning is analogous to the kernel trick used in support vector machines [Cortes and Vapnik 1995]. This observation enables our

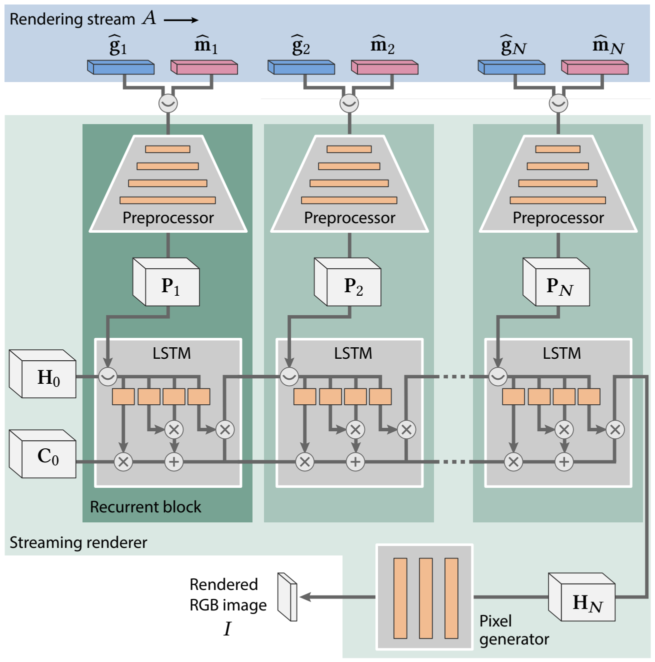

Fig. 5. We render the scene graph by collapsing it into a stream of transformed geometry and material representations. These are concatenated and sequentially input into a recurrent unit (preprocessor and an LSTM). After the stream is processed, the hidden state of the LSTM is converted into an RGB image using a pixel generator. 3D boxes and orange rectangles depict data tensors and trainable layers, respectively.

unified treatment across simple and complex transformations for both geometry and materials, and minimally constrains the model.

It is worth noting that the trainable model still has the option of relying on low-dimensional (e.g. 3D) transformations and using the remainder of dimensions for storing transformation-invariant information about the shape, such as topology information. In such cases, the encoded matrices **T** would feature rows and columns filled with zeros, except for entries on the main diagonal.

5 STREAMING NEURAL RENDERER

Renderers need to handle scenes containing an arbitrary number of elements. Many prior works assume a fixed-size, neural scene representation [Eslami et al. 2018; Granskog et al. 2020] and therefore cannot adapt to increasing scene complexity. We propose a *streaming* architecture to render our scene graphs. The streaming design is key to support variable scene content and to permit scaling up the scene complexity beyond what the model was trained on.

Rendering stream. We first collapse the scene graph into a stream of objects using the procedure described in Section 3. The stream A consists of N *transformed* and concatenated geometry and material vectors ($\smile$ denotes concatenation):

$$A = \left\{ \widehat{\mathbf{g}}_i \smile \widehat{\mathbf{m}}_i \,:\, \widehat{\mathbf{g}}_i = \left(\prod_{k=1}^{K(i)} \mathbf{T}^i_{\mathrm{g},k} \right) \cdot \mathbf{g}_i \,,\ \widehat{\mathbf{m}}_i = \left(\prod_{l=1}^{L(i)} \mathbf{T}^i_{\mathrm{m},l} \right) \cdot \mathbf{m}_i \right\}_{i=1..N} ,$$

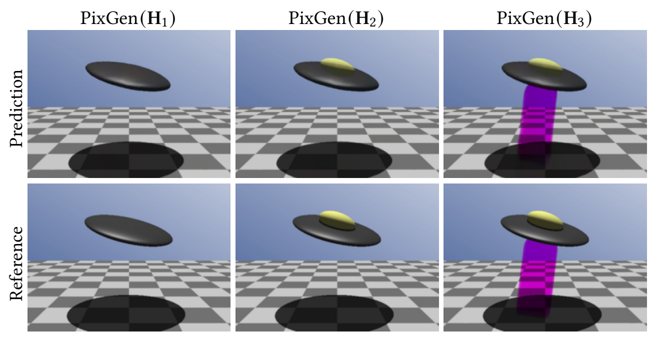

Fig. 6. The hidden state of the LSTM captures the content of the scene at any point of processing the rendering stream. The rendered image is obtained by running the final hidden state through a pixel generator. Here we visualize the hidden state after processing each object in the stream.

where, for the i-th object in the stream, the matrix $\mathbf{T}^i_{\mathrm{g},k}$ represents the k-th geometric transformation matrix on the path from the root to the leaf node (containing $\mathbf{g}_i$); $K(i)$ is the total number of geometric transformations along the path. Material transformations $\mathbf{T}^i_{\mathrm{m},l}$ and their number $L(i)$ are defined analogously.

5.1 Architecture

The renderer r is tasked with generating image I from the stream A, i.e. $I = r(A;\phi)$, where ϕ are the trainable parameters of the renderer. We leverage a recurrent architecture consisting of three trainable components: a preprocessor, an LSTM cell [Hochreiter and Schmidhuber 1997], and a pixel generator [Sitzmann et al. 2019]; see Figure 5. The preprocessor and the LSTM cell form a recurrent block. The recurrent block sequentially processes individual objects in the rendering stream and maintains an internal representation of the scene, which is later converted into the image I.

The task of the *preprocessor* is to map the i-th high-dimensional point in the stream—the concatenation $\widehat{\mathbf{g}}_i \smile \widehat{\mathbf{m}}_i$ which has 64 dimensions in our experiments—to a feature map $\mathbf{P}_i$. We use an upsampling, convolutional network that outputs $\mathbf{P}_i$ as a $c \times h \times w$ tensor, where c is the number of 2D features in the map, and h and w are set to the height and width of the rendered image, respectively.

The map $\mathbf{P}_i$ is input into the *LSTM cell*. The cell has the ability to aggregate and retain information across long streams of inputs by maintaining a cell state **C** and a hidden state **H**; see Hochreiter and Schmidhuber [1997] for details. The states have the same dimensionality as the feature map **P** and are initialized to zeros before rendering. Each of the four trainable components of the LSTM is implemented using a convolutional layer with 5×5 filters.

The preprocessor and the LSTM cell are executed iteratively—once for each object in the stream—with **C** and **H** being input into the LSTM in the next iteration. When all objects are processed, the hidden state of the LSTM is fed into a pixel generator that outputs the final image. The *pixel generator* is an MLP with three fully-connected layers that maintain the dimensions of **P**, except for the third layer that outputs only three channels—the RGB components of the rendered image I. Figure 6 shows a sequence of images obtained by running the pixel generator after every object in the stream.

Additionally, we observed a slight increase in model stability when concatenating 2D image-space coordinates encoded with the method of Tancik et al. [2020] to each preprocessor output $\mathbf{P}_i$ and to each layer input in the pixel generator.

While the scene graph representation is general, the rendering architecture is likely worth revisiting in the future. We present another promising architecture in the supplementary material.

6 OPTIMIZATION

In this section, we describe an end-to-end approach for extracting the vectors of canonical geometries and materials (Section 4.1) and optimizing the transformation encoders (Section 4.2) and the streaming neural renderer (Section 5). We train all components jointly using a synthetic dataset, where each dataset entry comprises:

- a list of N random geometry and material identifiers,
- a set of $K(i)$ and $L(i)$ random geometric and material transformations for each i-th geometry and material identifier, respectively ($K(i)$ and $L(i)$ is randomized between 0 and 8),
- a ground-truth rendering $\hat{I}$ of the transformed, shaded objects with a static floor and static (image) background.

For each training entry k, we assemble the rendering stream A_k; the geometry and material identifiers are used to select the corresponding canonical vectors. We then render the image $I_k = r(A_k; \phi)$ and evaluate its loss to the reference $\hat{I}_k$.

The loss consists of a VGG term [Simonyan and Zisserman 2015; Zhang et al. 2018] for improved sharpness and a negative log-likelihood term utilizing a normal distribution with annealed standard deviation [Eslami et al. 2018]; we decrease the deviation from 2.0 to 0.7 over the first 10k iterations. The optimal balance between the two loss terms was found by a parameter sweep for each dataset.

We use the Adam optimizer [Kingma and Ba 2015] with a learning rate of 10^{-4}, which is divided by $\sqrt{10}$ when approaching convergence to facilitate it. We initialize the weights of all networks as proposed by He et al. [2015] and the representation vectors using the standard normal distribution.

We train our networks for approximately one million iterations (roughly 7 days on a single NVIDIA V100 GPU) with a batch size of 16 entries. Training records are generated online and each contains up to four random objects. The ground-truth image $\hat{I}$ is rendered online using a rasterizer or a signed distance field renderer.

7 RESULTS

We implemented our proposed streaming neural renderer architecture (Section 5) using Pytorch [Paszke et al. 2019], and applied it to neural scene graphs (Section 4). As the focus of our work is on controllability and compositionality, we designed a set of simple experiments to highlight these two characteristics.

While our results do not feature high complexity, they suggest good generalization. In particular, we stress that our network was only trained on images featuring a handful of objects (up to four typically, see Figure 13) with random geometry and material transformations. Also, we present temporally stable animations, as well as results with a scene containing an order of magnitude more objects than seen during training. This generalization comes as a by-product of our architecture, not through specialized solutions such as using a temporal coherence loss; our network actually never "saw" an animation during training. To highlight the direct artistic control over the scene graph, most of the results are animated using traditional curve-based animation tools. Please see the supplementary video.

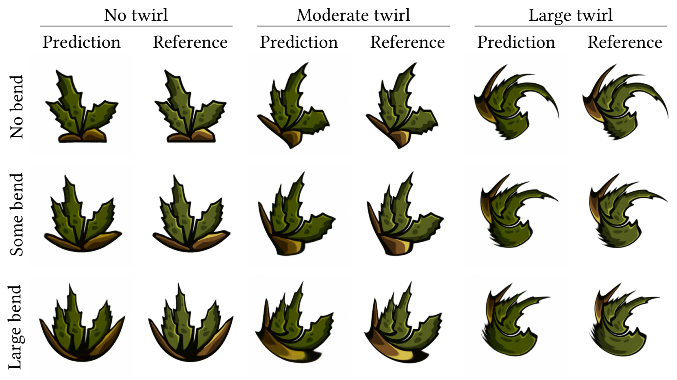

Fig. 7. A material (here a 2D texture) can be deformed by successively applying multiple deformations, i.e. chaining corresponding transformation nodes in the scene graph. Here we show a combination of bending (rows) and twirling (columns) with different intensities.

7.1 Orthogonality

The orthogonality of geometry, material and transformation is fundamental in a scene graph, and arguably the key ingredient of compositionality. As demonstrated in Figure 2, our approach preserves this orthogonality. The figure features three types of geometry (sphere, cube, torus), three types of materials (diffuse, metallic, volume), and uses affine transformations to position the scene elements. While our results lack sharpness, they do reproduce the distinctive appearance of each material, irrespective of the underlying geometry and transformation. In particular, we observe view-dependent highlights for metallic materials, and colored shadows as well as transparency for volumetric materials.

An important consideration with transformations is the ability of chaining them. This is trivial when considering only transformations that can readily be encoded in 4D matrices (rotation, translation, scaling), as we can first multiply the 4D matrices and encode directly the product. In Figure 7, we illustrate a more challenging scenario where we combine *order-dependent*, non-linear transformations (twirl and bend) while varying the magnitude of each transformation. As with our previous experiment, the network is capable of faithfully reproducing the reference behavior. We note, however, that achieving analogous results in 3D was considerably more challenging; results with reasonable quality required significantly longer training times.

7.2 2D animations

A classical use case where scene graphs are employed is animation, where the scene state is prescribed at key frames and smoothly interpolated for all in-between frames. In order to generate such an animation, we simply render every frame independently, as would be done with a standard renderer. As can be seen in the supplemental video, this results in temporally stable animations.

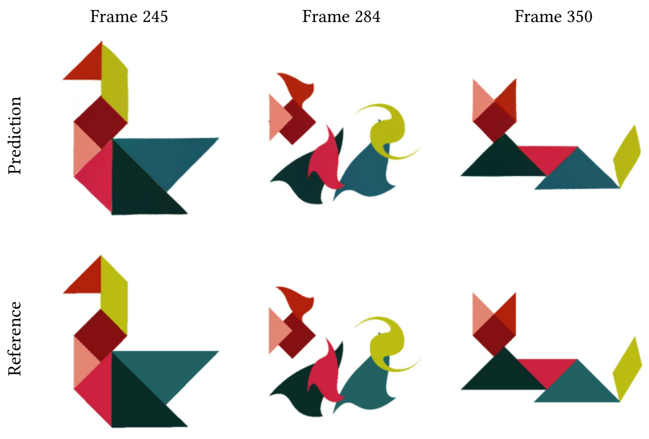

Fig. 8. Three frames from an animation with tangram shapes that gradually morph from one assembly into another. The twirl deformation is applied to individual pieces during the transition. Please see the supplementary video for the full animation.

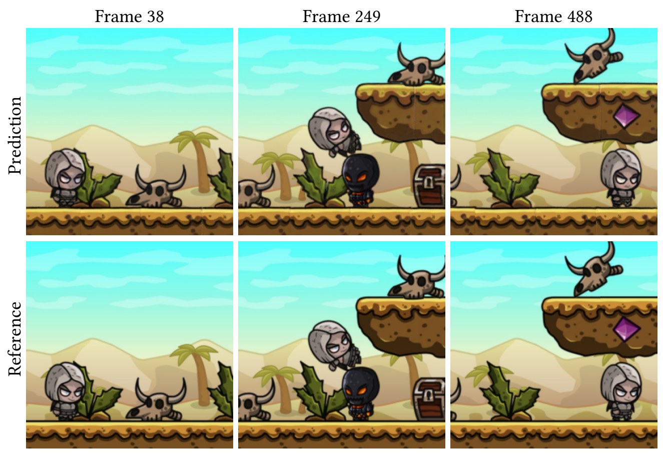

Fig. 9. Frames from a 2D sprite animation featuring 16 alpha-masked textures that are instantiated over a static background image. The prediction attains most of the texture detail. Artifacts appear primarily where two "ground" tiles meet due to slightly softer reproduction of texture edges.

Tangram animation. Figure 8 shows a few frames of an animation with basic colored shapes. The complete model (shape and material representations, transformation encoders, and the renderer) was trained on rendered images containing squares, triangles, and parallelograms; we used up to four shapes in each image. Each training image corresponds to a scene graph that contains transformation nodes for translating, rotating, scaling, and twirling the shapes, and transformation nodes for coloring materials. The model therefore utilized three geometry vectors and one material vector representing a constant neutral color. We employed three encoders: one for the 2D affine transforms, one for the twirling transform, and one for the coloring transform.

The animation (rendered at 256×256 resolution) features seven shapes that were scaled and colored to resemble tangram puzzle pieces. The pieces are animated in order to gradually morph between different puzzle assemblies. The supplementary video demonstrates good temporal stability and a close match to reference renders; the most notable artifact is slightly inaccurate alignment of objects.

Sprite animation. Figure 9 shows images from a 2D sprite animation. The model uses a single geometry vector—all objects are rendered as a transformed square—and 16 material vectors, each representing one of the alpha-masked textures used in the animation as sprites. During training, the shape instances are randomly translated, rotated, flipped, and skewed and each instance is assigned one of the 16 textures.

The test images (rendered at 256×256 resolution) demonstrate good agreement in placement of individual sprites; the main deficiency is the lack of detail in some textures. Across all our experiments with this setup, the texture detail matched the references better in low-resolution settings than in high-resolution ones. This is likely due to insufficient training time, or insufficient size of material representations to hold the fine details needed at higher resolutions.

7.3 3D experiments

In a 3D setting, the renderer has to handle perspective projection, shadowing, occlusions, and shading. The setup is therefore more complex than in the 2D case, resulting in longer training times and reduced accuracy. The problem statement, however, remains the same: given a stream of transformed geometry-material pairs, the neural renderer must reproduce the target image.

We trained the model using simple geometric primitives, e.g. sphere, cube, torus, and two meshes (the Utah teapot and Suzanne the monkey). We used 3D projective transforms for controlling the placement of the geometry, and a color transformation to adjust the base color of one of the three materials: diffuse, Phong, and volumetric. We also experimented with texturing the objects using surface and volumetric textures, but the model struggled with preserving the details except for very simple checkerboard patterns, hence we excluded texturing from the results. The only textured objects are the floor and the background, which are both held static during training; the renderer can thus easily memorize them. The scenes are lit by a single point emitter.

Bouncing tori. In Figure 10, we show an animation of two bouncing tori playing volleyball with a volumetric sphere. The model was trained using spheres and tori with diffuse or volumetric materials, and translation, rotation, scaling, and coloring transformations. Most effects are correctly reproduced in our results, including ground shadows, material appearances, and geometric transformations. The key missing component in our results is the shadows cast onto dynamic scene elements. For instance, the red shadow cast by the volumetric sphere onto the orange torus (visible in the leftmost column reference image) is completely missing in the network prediction. We speculate that the issue is due to the difficulty of maintaining a sufficiently rich scene representation in the LSTM state while rendering, compounded with the fact that such shadows are only sporadically present in training images.

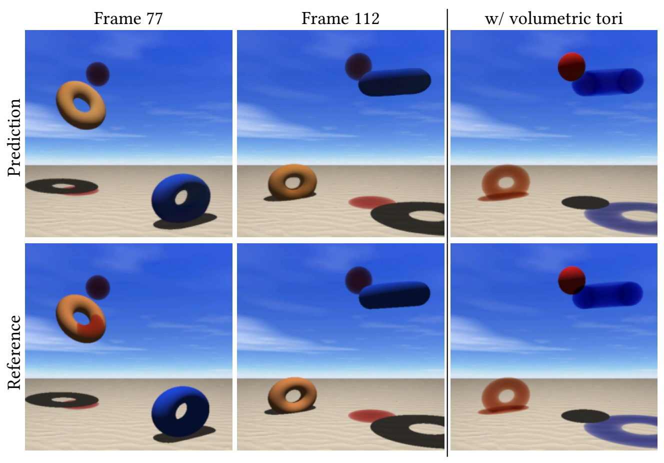

Fig. 10. Two diffuse tori playing beach volleyball with a volumetric ball. In the right-most column, the materials of the ball and tori are swapped.

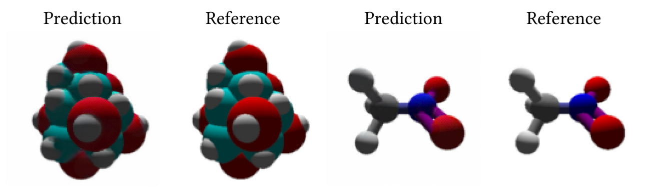

Fig. 11. Handling of occlusions and overlaps between many objects organized in molecular structures of glucose (left) and nitromethane (right). The model was trained with up to eight objects per scene.

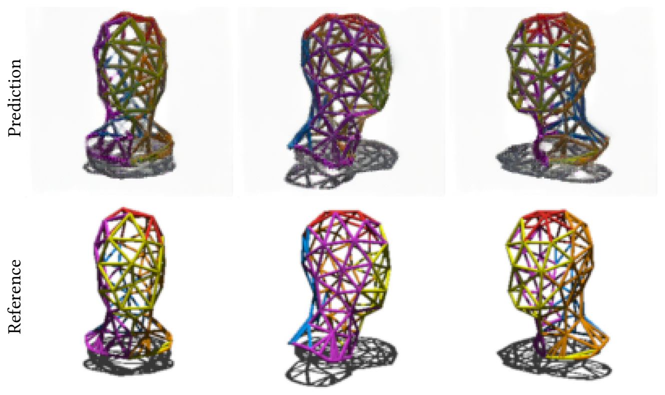

Fig. 12. Scalability stress test. The sculpture is made of 250 instances of cylindrical geometry and rendered with a model trained on scenes with up to 4 objects. The images display the sculpture from different viewing angles; this was achieved by rotating all instances around a common axis.

Occlusions. Complex occlusion scenarios can rapidly arise when authoring scenes, however, it is not clear whether training with random scene configurations leads to sufficient robustness in that regard. In Figure 11, we visualize two molecules to test overlaps and occlusions. The left molecule is represented using 24 intersecting spheres, the right one consists of 14 spheres and cylinders. The model for rendering these results was trained using images containing up to eight spheres and cylinders. The intricate object intersections are adequately captured, suggesting the neural renderer should scale well to more complex assemblies.

Scalability. In Figure 12, we perform a scalability stress test by rendering a coarse head sculpture using 250 cylinders. The model was trained with up to 4 objects using various types of geometry (sphere, cube, torus, cylinder, capsule) and transformations (translation, rotation, scaling, coloring). This result demonstrates a surprisingly good scalability, albeit with darker and less saturated colors. Nevertheless, the overall appearance remains visually consistent suggesting a graceful degradation of the network performance as its modeling capacity is progressively overloaded.

8 DISCUSSION AND FUTURE WORK

This section first highlights some specificities of our approach, and then discusses key limitations and outline research directions.

8.1 Discussion

Transformations. We chose to represent arbitrary (non-)linear transformations using high-dimensional matrices constructed using learned encoders, and applied using matrix-vector multiplication. We also considered transforming the learned tensors using a neural network directly. This might offer some advantages, especially for highly non-linear deformations and pattern generators, but it would not feature the associativity of matrix multiplication. The associativity, while not required, allows combining chains of transformations into a single matrix; a feature that applications may benefit from.

Representation-renderer coupling. End-to-end optimization leads to learned representations that are tightly coupled to the renderer. While this prevents adding new geometries and materials, similar couplings exist in classical renderers, e.g. a rasterizer cannot directly render a signed distance field. Still, the coupling in data-driven approaches is more prominent and our approach is no exception.

Generalization. Although our model cannot generalize to geometry and materials that it was not trained on, it can synthesize images with geometry-material *combinations* not seen during training. In this experiment, we leverage two renderers: a rasterizer and a signed-distance-field (SDF) renderer. The rasterizer does not support volumes, but it can handle triangle meshes, such as the Utah teapot. The SDF renderer cannot render such meshes, but features the complementary volumetric material; examples of training images are shown in Figure 13. In Figure 14, we show that the model can render the teapot geometry with a volumetric material without ever observing this combination during training.

The result suggests that we could eventually combine real-world and synthetic elements in a data-driven manner, e.g. by learning the appearance of a cat from photographs and applying it to a learned representation of a virtual geometry. The main requirement to enable such generalization is to provide a sufficient overlap in the content of the natural and virtual image sets (i.e. objects that exist in both to reduce the chances of set-specific performance).

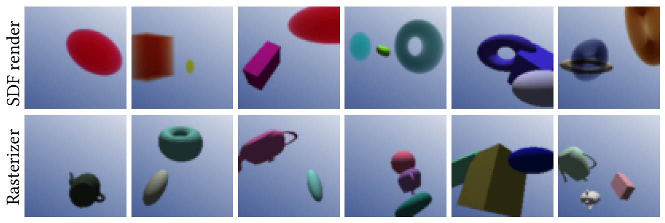

Fig. 13. Example target images of scenes used for training the model in Figure 14. We used two renderers, an SDF renderer and a rasterizer. Certain geometries, such as the teapot, can only be rendered using the rasterizer. The rasterizer, however, supports only a subset of the materials. Specifically, the volumetric appearance is only achievable with the SDF renderer.

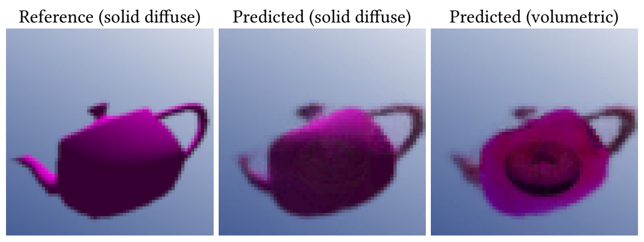

Fig. 14. The trained model never experienced a volumetric teapot during training, yet the neural renderer is capable of generalizing the volumetric appearance that it observed on other geometries. It is capable of rendering the teapot as a semi transparent volume, and thereby revealing a torus placed behind the teapot.

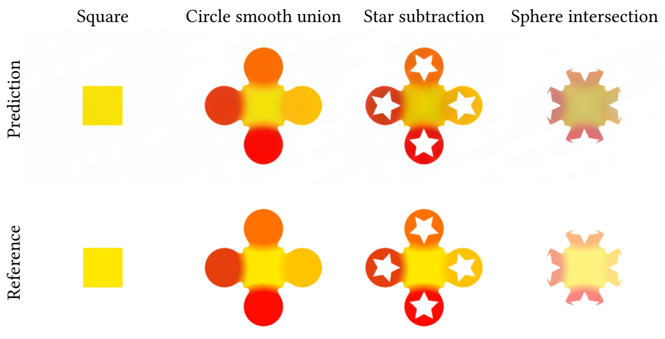

Fig. 15. Our prototype of neural computer-solid-geometry graphs allows combining 2D shapes using boolean operations: we combine a square (first column) with four circles (second column) using a "smooth" union operation. Four stars are the subtracted (third column) and the assembly intersected with a white sphere that desaturates the colors (last column).

Compositionality and scalability. We are not the first to impose a structure over neural representations. However, prior works either operate at the scene level [Granskog et al. 2020; Härkönen et al. 2020; Kulkarni et al. 2015; Nie et al. 2020], or rely on learned representations of standard graphics primitives, e.g. voxel grids [Nguyen-Phuoc et al. 2019, 2020, 2018; Olszewski et al. 2019; Rematas and Ferrari 2020], textures [Thies et al. 2019], and implicit volumetric representations [Mildenhall et al. 2020; Ost et al. 2020]. Our model provides a wider gamut of geometric and material transformations, enabling precise artistic control; see the accompanying video.

Furthermore, the streaming renderer allows increasing the object count by an order of magnitude compared to the training scenes; fixed-size representations are more constrained in that respect.

8.2 Future Work

Geometry and material tensors. The size of our neural scene representation is adaptive, i.e. it grows and shrinks with the number of scene objects. The individual objects, however, are represented using vectors of *fixed* height. Scaling up the complexity of individual scene elements requires increasing the height and results in quadratic growth of the transformation matrices.

We believe this can be rectified by simply replacing the "tall" column vector with a concatenation of many short ones, with their number adapted to the complexity of the object. Using such rank-2 representation tensors would force the model to break the learned representation into elements (tensor columns) that undergo the same treatment. This is analogous to how complex meshes are handled in the graphics pipeline: the model is broken into smaller elements, e.g. vertices, that undergo the same set of transformations.

Our ongoing experiments with (geometry) tensors show promising results. The main culprit is the mapping of material to geometry; simple concatenation is not possible as the width of the two tensors may differ. We believe the mapping can be handled by a trainable "texturing" unit; we leave this development for future work.

Content-dependent training performance. We observed poor learning with ambiguous scene configurations. For instance, translating the geometry and the texture in opposite directions produces an image largely identical to one without any translation. The system might eventually learn to recognize both translations correctly as some pixels still differ between the two configurations. Nevertheless the visual quality achieved (at equal training time) in such experiments was significantly lower than in simpler setups, e.g. when translating only the geometry and rotating only the textures.

Our current framework does not allow direct control of the camera. Instead, we control the placement of objects, which is functionally equivalent; see Figure 9 for an illustration of a horizontal camera pan. Based on our experiments with texture transformations, we expect combining camera and geometry transformations to hinder training. Additional work is needed to address this issue, e.g. with a curriculum-based learning approach [Bengio et al. 2009].

Lighting. In addition to static backgrounds, all our experiments used static lighting. Allowing arbitrary of a single light source should be well within the modeling capacity of our framework—the parameters of the light could be input alongside each object. Thanks to the linearity of light transport, one could handle multiple light sources by linearly combining multiple rendered images—one for each light source. However, as long as rendering artifacts prevail, combining multiple renders also means combining multiple artifacts. An alternative would be to add a recurrent block to iterate through all light sources for each object. We leave these investigations for future work.

Constructive solid geometry. Extending our framework to handle boolean operators such as union, difference, and intersection would enable constructive solid geometry (CSG) techniques common in computer aided design. We implemented a proof of concept by leveraging a postfix streaming architecture (where we first provide the geometry representations, and then the operator to be applied), and show the results in Figure 15. We note that this result required a different scene graph parser (involving two passes with distinct implementations), suggesting some refactoring of our framework is needed to properly handle CSG techniques—we provide details in the supplementary material. Nevertheless, our result is encouraging and suggests this is an avenue worth investigating.

Graphics priors. Our model embeds a number of graphics priors: i) orthogonality of geometry and materials, ii) explicit placement of geometry, and iii) universality of transformations—each impacting every object in the same way. The proposed renderer, however, remains fairly generic aside for the streamed processing that aids scalability. Embedding additional graphics priors is likely to further improve visual results. For instance, ray-marching through a neural light field [Mildenhall et al. 2020] has shown great potential and permits compositionality in a straightforward manner: each object in the scene can be represented using a distinct neural radiance field, which can be positioned by transforming its bounding box. The concurrent work by Ost et al. [2020] represents a commendable effort in this direction. One limitation of radiance fields is that materials are effectively baked in the representation. Our approach is more general, but the burden of learning visibility and masking from scratch—as opposed to querying the model at ray-marched 3D points—is likely to lower the prediction quality. We believe both approaches are worth developing further.

9 CONCLUSION

We have described a neural scene representation that has many desirable properties, such as controllability, modularity, and unified handling of diverse transformations. The core idea of our approach is to leverage high-dimensional embeddings for geometry and materials—these are learned directly from data—and to transform them using matrices that are encoded from artist-friendly parameters by a set of neural encoders. Despite the content of the vectors not having any specific interpretation (e.g. voxels or distance fields), the transformations are still general and produce the same effect irrespective of the object they modify.

We demonstrate that learned representations can be organized into scene graphs that facilitate direct control over the placement of individual objects and the overall scene composition. The streaming renderer can handle an order of magnitude more scene elements than it was trained on, and produces temporally stable animations. We hope our work will stimulate further developments to turn neural rendering into a universally applicable tool.

ACKNOWLEDGMENTS

We thank Jacob Munkberg, Jon Hasselgren and Marco Salvi for insightful discussions and feedback. We used the following models: the Utah teapot by Martin Newell, the Suzanne model (Blender monkey) by Willem-Paul van Overbruggen, and the Stanford bunny.

REFERENCES

Kara-Ali Aliev, Artem Sevastopolsky, Maria Kolos, Dmitry Ulyanov, and Victor Lempitsky. 2020. Neural Point-Based Graphics. In *Computer Vision – ECCV 2020*, Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (Eds.). Springer International Publishing, Cham, 696–712.

Oron Ashual and Lior Wolf. 2019. Specifying Object Attributes and Relations in Interactive Scene Generation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum Learning. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML '09)*. Association for Computing Machinery, New York, NY, USA, 41–48. https://doi.org/10.1145/1553374.1553380

Brent Burley. 2012. Physically based shading at Disney. In *ACM SIGGRAPH Courses: Practical Physically-Based Shading in Film and Game Production*. ACM, New York, NY, USA, 18:35–18:48. https://doi.org/10.1145/2343483.2343493

Xuelin Chen, Daniel Cohen-Or, Baoquan Chen, and Niloy J. Mitra. 2021. Towards a Neural Graphics Pipeline for Controllable Image Generation. *Computer Graphics Forum* 40, 2 (2021).

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*. Curran Associates, Inc., 2172–2180.

Corinna Cortes and Vladimir Vapnik. 1995. Support-Vector Networks. *Mach. Learn.* 20, 3 (Sept. 1995), 273–297. https://doi.org/10.1023/A:1022627411411

Sebastien Ehrhardt, Oliver Groth, Aron Monszpart, Martin Engelcke, Ingmar Posner, Niloy Mitra, and Andrea Vedaldi. 2020. RELATE: Physically Plausible Multi-Object Scene Synthesis Using Structured Latent Spaces. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 11202–11213. https://proceedings.neurips.cc/paper/2020/file/806beafe154032a5b818e97b4420ad98-Paper.pdf

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

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger (Eds.), Vol. 27. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf

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).

Michelle Guo, Alireza Fathi, Jiajun Wu, and Thomas Funkhouser. 2020. Object-Centric Neural Scene Rendering. *arXiv preprint arXiv:2012.08503* (2020).

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV) (ICCV '15)*. IEEE Computer Society, USA, 1026–1034. https://doi.org/10.1109/ICCV.2015.123

Pedro Hermosilla, Sebastian Maisch, Tobias Ritschel, and Timo Ropinski. 2019. Deep-learning the Latent Space of Light Transport. *Computer Graphics Forum* 38, 4 (2019), 207–217. https://doi.org/10.1111/cgf.13783

Roei Herzig, Amir Bar, Huijuan Xu, Gal Chechik, Trevor Darrell, and Amir Globerson. 2020. Learning Canonical Representations for Scene Graph to Image Generation. In *European Conference on Computer Vision*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735 arXiv:https://doi.org/10.1162/neco.1997.9.8.1735

Erik Härkönen, Aaron Hertzmann, Jaakko Lehtinen, and Sylvain Paris. 2020. GANSpace: Discovering Interpretable GAN Controls. In *Proc. NeurIPS*.

Maor Ivgi, Yaniv Benny, Avichai Ben-David, Jonathan Berant, and Lior Wolf. 2020. Scene Graph to Image Generation with Contextualized Object Layout Refinement. arXiv:cs.CV/2009.10939

Justin Johnson, Agrim Gupta, and Li Fei-Fei. 2018. Image Generation from Scene Graphs. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1219–1228. https://doi.org/10.1109/CVPR.2018.00133

Tero Karras, Samuli Laine, and Timo Aila. 2019. A Style-Based Generator Architecture for Generative Adversarial Networks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 4396–4405. https://doi.org/10.1109/CVPR.2019.00453

Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. 2020. Analyzing and Improving the Image Quality of StyleGAN. In *Proc. CVPR*.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1412.6980

Tejas D Kulkarni, William F. Whitney, Pushmeet Kohli, and Josh Tenenbaum. 2015. Deep Convolutional Inverse Graphics Network. In *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc., 2539–2547.

Zhengqi Li, Simon Niklaus, Noah Snavely, and Oliver Wang. 2020. Neural Scene Flow Fields for Space-Time View Synthesis of Dynamic Scenes. *ArXiv* abs/2011.13084 (2020).

Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. 2020. Neural Sparse Voxel Fields. *NeurIPS* (2020).

Yunchao Liu, Jiajun Wu, Zheng Wu, Daniel Ritchie, William T. Freeman, and Joshua B. Tenenbaum. 2019. Learning to Describe Scenes with Programs. In *International Conference on Learning Representations*. https://openreview.net/forum?id=SyNPk2R9K7

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

Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. 2019. Occupancy Networks: Learning 3D Reconstruction in Function Space. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.

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*.

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 Nguyen-Phuoc, Christian Richardt, Long Mai, Yong-Liang Yang, and Niloy Mitra. 2020. BlockGAN: Learning 3D Object-aware Scene Representations from Unlabelled Images. In *Advances in Neural Information Processing Systems 33*.

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*. Curran Associates, Inc., 7891–7901.

Weili Nie, Tero Karras, Animesh Garg, Shoubhik Debhath, Anjul Patney, Ankit B. Patel, and Anima Anandkumar. 2020. Semi-Supervised StyleGAN for Disentanglement Learning. arXiv:cs.CV/2003.03461

Michael Niemeyer and Andreas Geiger. 2021. GIRAFFE: Representing Scenes as Compositional Generative Neural Feature Fields. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. 2020. Differentiable Volumetric Rendering: Learning Implicit 3D Representations without 3D Supervision. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.

Michael Oechsle, Lars Mescheder, Michael Niemeyer, Thilo Strauss, and Andreas Geiger. 2019. Texture Fields: Learning Texture Representations in Function Space. In *International Conference on Computer Vision*.

Michael Oechsle, Michael Niemeyer, Christian Reiser, Lars Mescheder, Thilo Strauss, and Andreas Geiger. 2020. Learning Implicit Surface Light Fields. In *International Conference on 3D Vision (3DV)*.

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).

Julian Ost, Fahim Mannan, Nils Thuerey, Julian Knodt, and Felix Heide. 2020. Neural Scene Graphs for Dynamic Scenes.

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. 2019. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Keunhong Park, Utkarsh Sinha, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Steven M. Seitz, and Ricardo Martin-Brualla. 2020. Deformable Neural Radiance Fields. *arXiv preprint arXiv:2011.12948* (2020).

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. 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., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. 2020. D-NeRF: Neural Radiance Fields for Dynamic Scenes. *arXiv preprint arXiv:2011.13961* (2020).

Konstantinos Rematas and Vittorio Ferrari. 2020. Neural Voxel Renderer: Learning an Accurate and Controllable Rendering Tool. In *CVPR*.

Paul Sanzenbacher, Lars Mescheder, and Andreas Geiger. 2020. Learning Neural Light Transport. arXiv:cs.CV/2006.03427

Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1409.1556

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.

Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. 2020. Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains. *NeurIPS* (2020).

Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. 2016. Multi-view 3D Models from Single Images with a Convolutional Network. In *Computer Vision – ECCV 2016*. Springer International Publishing, Cham, 322–337.

Ayush Tewari, Ohad Fried, Justus Thies, Vincent Sitzmann, Stephen Lombardi, Kalyan Sunkavalli, Ricardo Martin-Brualla, Tomas Simon, Jason Saragih, Matthias Nießner, Rohit Pandey, Sean Fanello, Gordon Wetzstein, Jun-Yan Zhu, Christian Theobalt, Maneesh Agrawala, Eli Shechtman, Dan B Goldman, and Michael Zollhöfer. 2020. State of the Art on Neural Rendering. *Computer Graphics Forum* 39, 2 (2020), 701–727. https://doi.org/10.1111/cgf.14022

Justus Thies, Michael Zollhöfer, and Matthias Nießner. 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

Wenqi Xian, Jia-Bin Huang, Johannes Kopf, and Changil Kim. 2020. Space-time Neural Irradiance Fields for Free-Viewpoint Video. *arXiv preprint arXiv:2011.12950* (2020).

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)*.