RenderFormer: Transformer-based Neural Rendering of Triangle Meshes with Global Illumination

CHONG ZENG, State Key Lab of CAD & CG, Zhejiang University and Microsoft Research Asia, China
YUE DONG, Microsoft Research Asia, China
PIETER PEERS, College of William & Mary, USA
HONGZHI WU, State Key Lab of CAD & CG, Zhejiang University, China
XIN TONG, Microsoft Research Asia, China

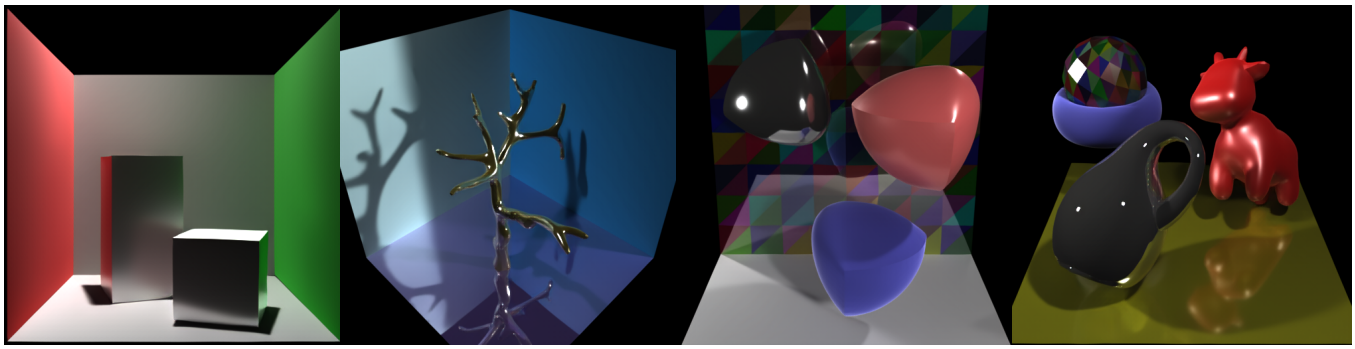

Fig. 1. Examples of triangle-mesh based scenes rendered with RenderFormer without per-scene training or fine-tuning that include (multiple) specular reflections, complex shadows with details finer than a triangle, diffuse indirect lighting, glossy reflections, soft and hard shadows, and multiple light sources.

We present RenderFormer, a neural rendering pipeline that directly renders an image from a triangle-based representation of a scene with full global illumination effects and that does not require per-scene training or fine-tuning. Instead of taking a physics-centric approach to rendering, we formulate rendering as a sequence-to-sequence transformation where a sequence of tokens representing triangles with reflectance properties is converted to a sequence of output tokens representing small patches of pixels. RenderFormer follows a two stage pipeline: a view-independent stage that models triangle-to-triangle light transport, and a view-dependent stage that transforms a token representing a bundle of rays to the corresponding pixel values guided by the triangle-sequence from the the view-independent stage. Both stages are based on the transformer architecture and are learned with minimal prior constraints. We demonstrate and evaluate RenderFormer on scenes with varying complexity in shape and light transport.

CCS Concepts: • **Computing methodologies** → **Rendering**.

Additional Key Words and Phrases: Rendering, Global Illumination, Sequence-to-Sequence, Transformer

Authors' Contact Information: Chong Zeng, State Key Lab of CAD & CG, Zhejiang University and Microsoft Research Asia, Hangzhou, China, chongzeng2000@gmail.com; Yue Dong, Microsoft Research Asia, Beijing, China, yuedong@microsoft.com; Pieter Peers, College of William & Mary, Williamsburg, USA, ppeers@siggraph.org; Hongzhi Wu, State Key Lab of CAD & CG, Zhejiang University, Hangzhou, China, hwu@acm.org; Xin Tong, Microsoft Research Asia, Beijing, China, xtong@microsoft.com.

SIGGRAPH Conference Papers '25, August 10–14, 2025, Vancouver, BC, Canada

ACM ISBN 979-8-4007-1540-2/2025/08
https://doi.org/10.1145/3721238.3730595

ACM Reference Format:
Chong Zeng, Yue Dong, Pieter Peers, Hongzhi Wu, and Xin Tong. 2025. RenderFormer: Transformer-based Neural Rendering of Triangle Meshes with Global Illumination. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers (SIGGRAPH Conference Papers '25), August 10–14, 2025, Vancouver, BC, Canada.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3721238.3730595

1 Introduction

Traditional graphics pipelines render virtual scenes by simulating the physical process of light transport through the scene. Recently, neural rendering has endeavored to bypass the simulation process, and instead *learn* to predict the effects of light transport. However, most neural rendering methods often accomplishes this by overfitting the model to a fixed scene. This raises the intriguing question whether it is possible to learn a rendering *pipeline* rather than a rendering *model*.

In this paper, we take a first step towards a *fully* neural rendering pipeline, named RenderFormer, that (a) does not require per-scene training, (b) takes a classic triangle-mesh based scene description as input, and (c) that renders the scene with full global illumination. To achieve these goals, we offer a new perspective on resolving light transport in a virtual scene, and formulate rendering as a sequence-to-sequence transformation, where each token in the sequence represents a triangle with reflectance properties that is subsequently transformed to a triangle with the converged radiance distribution of the light transport equilibrium. Rather than explicitly describing the resulting radiance distribution and following the flow of light through the virtual scene as dictated by the Rendering Equation [Kajiya 1986], RenderFormer learns a neural rendering

pipeline directly from data with minimal prior constraints. Compared to conventional rendering paradigms: RenderFormer directly 'solves' the rendering equation without Monte-Carlo integration noise and without requiring complex algorithmic modification as in rasterization. In contrast to existing neural rendering methods for synthetic scenes, RenderFormer does not require per-scene/object training (e.g., the objects in Figure 1 are not part of the training set).

RenderFormer follows a two-stage architecture: a view-independent stage that models triangle-to-triangle light transport, and a view-dependent stage that evaluates the transformed sequence of triangle tokens into an image. Both stages are based on the powerful transformer architecture [Vaswani et al. 2017] known for its capability to model long-range relations (e.g., light transport from one triangle to all other triangles). However, in contrast to typical transformer architectures, RenderFormer utilizes a (relative) positional encoding based on the 3D spatial position of the triangles rather than the 1D index position in the sequence. Similar to most neural rendering techniques, RenderFormer does not require recursive computations and directly solves global illumination transport in a single pass. Moreover, RenderFormer is fully based on learnable neural components, and thus naturally fully differentiable, without relying on existing fixed (i.e., non-learnable) rendering algorithms such as rasterization, ray tracing, or ray marching.

In this paper, we present an initial step towards a full neural rendering pipeline on a constrained set of scene types. First, due to the computational costs of transformers, RenderFormer is currently limited to triangle meshes of at most 4,096 triangles. Second, RenderFormer is also constrained by the variations seen during training: currently our training data only includes a single reflectance model [Walter et al. 2007], with its parameters assigned on a per-triangle basis (i.e., no textures). The training scenes include at most 8 diffuse light sources, and the camera (with fixed 512×512 resolution) is placed outside the scene's bounding box. We believe that RenderFormer, with further development and optimization, can potentially offer an alternative rendering paradigm for both forward and inverse rendering applications while leveraging current (and future) advances in transformer-optimized tensor cores.

2 Related Work

Rendering Equation. The Rendering Equation [Kajiya 1986] formally describes light transport in virtual scenes defined by its geometry (often modeled by a triangle mesh), the associated material properties in the form of Bidirectional Reflectance Distribution Functions (BRDFs) [Nicodemus et al. 1992], light sources, and a virtual camera. Over the past four decades a rich variety of methods have been proposed for accurately and efficiently solving the Rendering Equation. Monte Carlo path tracing and variants [Dutré et al. 2018; Pharr et al. 2023] are among the most popular and effective methods for solving the rendering equation. Recently, to offset the significant computational cost, path tracing algorithms have been augmented by machine learning techniques [Wang et al. 2023] (e.g., filtering [Bako et al. 2017] or caching [Coomans et al. 2024; Müller et al. 2021]). All of the above methods explicitly encode the rendering equation as part of the solution method, and hence due to the recursive nature of the Rendering Equation, these methods are also recursive. RenderFormer does not rely explicitly on the Rendering Equation, and directly computes light transport without recursion.

An alternative class of mathematical techniques for solving the Rendering Equation are finite element methods, and the resulting class of rendering algorithms are called *radiosity* methods [Cohen et al. 1988; Goral et al. 1984]. However, classic radiosity methods are mostly limited to isotropic scattering from diffuse surfaces. In the spirit of radiosity, Hadadan et al. [2021] introduce a learning-inspired *neural radiosity* variant that decouples solving the Rendering Equation (i.e., training) and rendering (i.e., inference) that can effectively synthesize arbitrary views of a scene without material restrictions. Neural radiosity (and its extension to dynamic scenes [Su et al. 2024b]) shares similarities to Precomputed Radiance Transfer (PRT) [Sloan et al. 2023] that formulates light transport as an inner-product between the light transport matrix of a scene and the lighting expressed in a suitable basis (e.g., Spherical Harmonics). Inspired by PRT, Rainer et al. [2022] and Gao et al. [2022] leverage neural networks to learn a suitable embedding of the incident lighting instead of a predefined lighting basis. Neural radiosity and PRT methods incur a large precomputation overhead for *each* scene. While training RenderFormer is expensive, training only happens once, after which scenes can be rendered without further training.

Neural Rendering. Neural rendering methods [Tewari et al. 2022] replace the simulation of light transport by a learned neural process, and hence the Rendering Equation is never explicitly imposed and instead an implicit representation of light transport is learned, Neural Radiance Fields (NeRFs) [Mildenhall et al. 2021] is a well known example of such an implicit representation of light transport. Neural rendering in general employs a neural scene representation which requires specialized methods [Granskog et al. 2020, 2021; Haque et al. 2023; Yuan et al. 2022; Zheng et al. 2024] for making scene modifications. In contrast, RenderFormer takes a regular triangle-mesh based scene description as input, and thus is compatible with existing tool-chains for authoring virtual scenes.

Sanzenbacher et al. [2020] perform screen-space neural shading augmented with global neural light transport computed on a point-cloud representation of the scene. While the point-cloud helps to generalize the light transport computations, the two stage network is trained per scene (either static or dynamic). In contrast RenderFormer does not require per scene training. RenderNet [Nguyen-Phuoc et al. 2018] and Neural Voxel Rendering [Rematas and Ferrari 2020] learn a convolutional neural rendering pipeline. However, instead of triangle meshes, both methods take a 3D voxel grid as input, and only learn *local* shading under a single point light. In contrast, RenderFormer renders the scene with full global illumination.

Transformers for Rendering. The key building block in RenderFormer is the transformer architecture [Vaswani et al. 2017] which is built around multi-head attention blocks and that maps a sequence of tokens to another sequence of tokens while handling long-range dependencies. Transformers have proven to be effective architectures for vision tasks [Dosovitskiy et al. 2020] and for driving large language models [Kenton and Toutanova 2019]. In rendering, Ren et al. [2024] leverage the cross-attention mechanism in transformers for accelerating the gather step in neural reflective shadow maps. In the context of NeRFs, NerFormer [Reizenstein

et al. 2021] leverages epipolar constraints and attention to construct feature volumes. IBRNet [Wang et al. 2021a] estimates density along rays with transformers. Recent view-interpolation methods [Liang et al. 2024; Sajjadi et al. 2022; Suhail et al. 2022; Varma et al. 2022], not only employ transformers to compute features along rays, but also use a transformer to aggregate features along rays. LVMS [Jin et al. 2024] takes a different approach, and directly transforms pixel patches from the input images to view-interpolated images. Jin et al. encode the poses of the input and output cameras by tokenizing each pixels' view ray; RenderFormer uses a similar strategy for tokenizing the pose of the virtual camera.

3 RenderFormer

RenderFormer is composed of two stages: a *view-independent* stage and a *view-dependent* stage. Both stages utilize the transformer architecture. The view-independent stage (subsection 3.1) takes a sequence of triangles with corresponding properties as input, and transforms it to a sequence of per-triangle features that store a neural encoding of the triangle's overall outgoing radiance. The view-dependent stage (subsection 3.2) takes the transformed triangle sequence as input as well as tokens that represent bundles of rays corresponding to 8×8 pixel patches in the target image and transforms the latter to outgoing radiance values corresponding to each ray in the bundle. We train RenderFormer end-to-end (subsection 3.3) once, after which a triangle-based scene can be fed into RenderFormer without any fine-tuning or training.

3.1 View-independent Stage

Transformer Architecture. The view-independent stage closely follows the original transformer architecture [Vaswani et al. 2017] with full bidirectional self-attention. The transformer takes as input a sequence of triangle embeddings (i.e., tokens). Each triangle token encodes all relevant information for rendering such as surface normal and reflectance. In addition we add 16 register tokens to the input sequence that can be used by the transformer to store global information and potentially remove high-frequency noise in the embedding [Darcet et al. 2024]. Each triangle and register token is a 768-dimensional vector. The view-independent stage is composed of 12 transformer layers, where each layer has 6 heads and 768 hidden units, followed by a 768×4 feed-forward fully connected network. We follow LLaMA [Touvron et al. 2023] and apply pre-normalization using RMS-Normalization [Zhang and Sennrich 2019] and use SwiGLU as activation function [Shazeer 2020]. Furthermore we leverage QK-Normalization [Henry et al. 2020] to stabilize training. Figure 2 (top) summarizes the architecture of the view-independent stage.

Relative Spatial Positional Embedding. A key difference between RenderFormer and typical uses of transformers (e.g., large language models) is that the index position of the token (i.e., triangle) in the sequence is irrelevant; swapping two triangles in the sequence should produce the same result. However, the position of the triangle in the virtual world matters. The contribution to the global light transport differs for two triangles with exactly the same reflectance properties and shape (and thus with identical token embedding) but at different positions in the scene. Furthermore, translating the whole scene (including light sources and virtual camera) does not alter the light transport. Hence, RenderFormer requires a relative positional encoding based on the 3D spatial location for each triangle with respect to other triangles. We, therefore, do not embed the *absolute* position of the triangle by adding the positional encoding directly to the triangle token, but instead adapt Rotational Positional Encoding (RoPE) [Su et al. 2024a] to modify the triangle token to embed the triangle's *relative* 3D spatial location. RoPE expresses the positional embedding as a rotation and relies on the fact that the composition of two rotations is equivalent to a relative rotation between both. However, unlike a simple index in a sequence, the position of the triangle is determined by three 3D vertices of floating point values. We therefore first concatenate all three vertex positions into a 9D vector and multiply each element, duplicated 6 times, with each of 6 frequencies (with scales exponentially distributed between 1 and 5: $[1.0, 1.3797, 1.9037, 2.6265, 3.6239, 5.0]$), yielding a vector of 54 scaled frequencies. Following RoPE, we encode each coefficient as the sine and cosine of the angle proportional to the scaled frequencies, and create a block-diagonal rotation matrix where each sine/cosine pair determines the rotation for each 2×2 block. Each of the 6 attention heads operates on 128 coefficients of the triangle token embedding ($6 \text{ heads} \times 128 = 768$). Consequently, we apply the block-rotation only to the first 108 coefficients for each head and leave the remaining 20 coefficients unchanged. Similar to RoPE, we apply the relative spatial positional embedding to the tokens at each attention layer. Ideally we would like the relative positional encoding to also be invariant to scene rotations. However, because SO(3) is not commutative, this is difficult to achieve with RoPE.

To ensure that the register tokens are also invariant to scene translations, we also apply relative spatial positional encoding on the register tokens using the average position of all scene vertices.

Triangle Embedding. For each triangle we want to embed all relevant information needed for rendering, such as shading normals, reflectance properties, and emission (in case of a light source). As noted above, the position and shape of the triangle will be encoded via relative spatial positional embedding.

We store a normal per vertex that is interpolated (and normalized) over the triangle using an absolute positional encoding of the per-vertex normals. Practically, we encode all three normals with (NeRF) positional encoding [Mildenhall et al. 2021] with 6 frequencies (using the same frequencies as for relative spatial positional embedding), which are subsequently expanded to a 768-dimensional vector through a single linear layer followed by RMS-Normalization.

We model surface reflectance with a microfacet BRDF model using a GGX normal-facet distribution [Walter et al. 2007] parameterized by diffuse albedo, specular albedo, and roughness. We stack the reflectance parameters as well as emission into a 10 dimensional vector (3D for all parameters except for roughness (1D)). This 10-dimensional vector is expanded to a 768-dimensional vector by a single linear layer followed by RMS-Normalization. The resulting 768-dimensional vector is added to the above normal embedding.

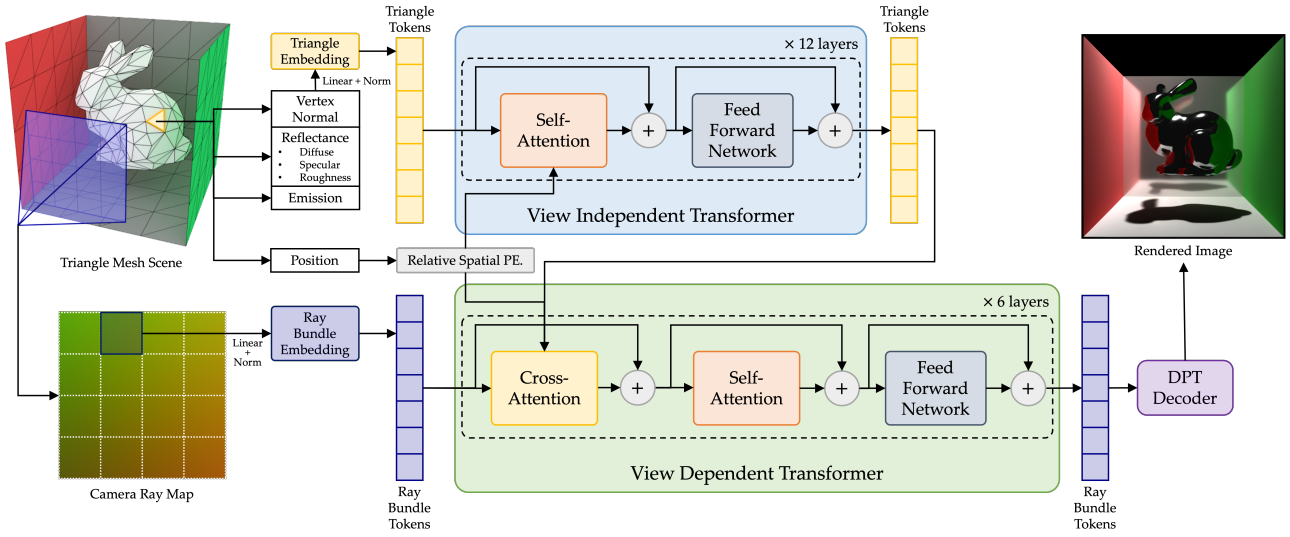

Fig. 2. RenderFormer Architecture Overview. Top: the view-independent stage resolves triangle-to-triangle light transport from a sequence of triangle tokens that encode the reflectance properties of each triangle. The relative position of each triangle is separately encoded, and applied to each token at each self-attention layer. Bottom: the view-dependent stage takes as input the virtual camera position encoded as a sequence of ray-bundles. Guided by the resulting triangle tokens from the view-independent stage via a cross-attention layer, the ray-bundle tokens are transformed to tokens encoding the outgoing radiance per view ray. Finally, the ray-bundle tokens are transformed to log-encoded HDR radiance value through an additional dense vision transformer.

3.2 View-dependent Stage

The goal of the view-dependent stage is to transform the triangle tokens transformed by the previous view-independent stage to radiance pixel values corresponding to a given virtual camera. For performance reasons, we encode an 8×8 radiance pixel patch in an output token. Inspired by Gao et al. [2024] and Jin et al. [2024], we specify the virtual camera to the view-dependent transformer by encoding a bundle of 8×8 rays that pass through the centers of the pixels in the corresponding output patch.

Transformer Architecture. We follow a similar architecture as for the view-independent transformer, except that it transforms a sequence of ray-bundle tokens (instead of triangle tokens), and we only repeat the attention layers 6 times instead of 12. Furthermore we precede each self-attention layer with a cross-attention layer that connects the ray-bundle tokens with the triangle tokens (including register tokens) from the view-independent stage. The role of the cross-attention layer is to find the triangles related to the rays in the ray-bundle. As before, each transformer layer has 6 heads, 768 hidden units, and a 768×4 feed-forward network. We again employ SwiGLU activations, QK-normalization, and RMS-Normalization. Furthermore, we found that the view-dependent stage requires higher precision (tf32) than the view-independent stage (which uses bf16) to convergence during training. In addition, to decode the pixel-patch tokens into $\log(x+1)$-encoded HDR RGB radiance values, we found that even though the radiance observed for each view ray is independent of other view rays, sharing information between view rays through self-attention between ray-bundle tokens improves rendering accuracy (Table 1, 3rd vs. 4th row). Furthermore, we employ a dense vision transformer [Ranftl et al. 2021]

Table 1. Ablation study of different model variants and architectures. Due to computational constraints, all ablation studies are performed at 256×256 resolution. Layer configurations are denoted as: `#view-independent + #view-dependent` layers.

Variant	PSNR ↑	SSIM ↑	LPIPS ↓	FLIP ↓
full view-dependent stage	**29.77**	**0.9526**	**0.05514**	**0.1751**
w/o dpt	29.75	0.9476	0.05519	0.1806
w/o self-attention	29.70	0.9503	0.05703	0.1766
w/o dpt & w/o self-attention	29.15	0.9396	0.06407	0.1836
camera space view-dep. stage	**29.77**	**0.9526**	**0.05514**	**0.1751**
world space view-dep. stage	28.98	0.9420	0.06309	0.1904
205M / 768d tokens / 12 + 6 layers	**29.77**	**0.9526**	**0.05514**	**0.1751**
143M / 768d tokens / 8 + 4 layers	28.99	0.9444	0.06408	0.1873
71M / 512d tokens / 8 + 4 layers	28.27	0.9356	0.07238	0.2032
45M / 384d tokens / 8 + 4 layers	27.87	0.9295	0.07921	0.2075
12 + 6 layers	29.77	0.9526	0.05514	0.1751
9 + 9 layers	30.11	0.9554	0.05121	0.1735
6 + 12 layers	**30.38**	**0.9560**	**0.05043**	**0.1685**
0 + 18 layers	28.28	0.9355	0.07152	0.1994

on the features from the last 4 layers of the view-dependent transformer, to further improve accuracy (Table 1, 1st vs. 2nd row) *and* reduce (but not fully eliminate) resolution dependence. Figure 2 (bottom) summarizes the view-dependent architecture.

To reduce the degrees of freedom in the training data, we perform the view-dependent stage in camera coordinates, rather than in world coordinates as in the view-dependent stage. This is trivially achieved by applying the relative positional spatial embedding using transformed vertex coordinates at each attention layer. We do not apply any other transformation (e.g., normals) because after the view-independent transformation, the interpretation of the triangle tokens does not align anymore with the original embedding. By expressing the triangles' (and registers') positional embedding in

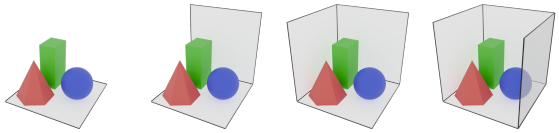

Fig. 3. The four template scenes used for generating training data.

camera coordinates we avoid having to learn the world-to-camera transformation, which also helps to improve accuracy (Table 1, 5th-6th row).

Ray Bundle Embedding. Each ray bundle is a collection of 8×8 rays that go through the center of the pixels of the corresponding pixel patch. Because the scene is expressed in camera coordinates in the view-dependent stage, the origin of all rays is $(0, 0, 0)$. We therefore, only need to encode the normalized directions of each ray. We stack the 64 direction vectors in a 192-dimensional vector which is subsequently encoded by a single linear layer followed by RMS-Normalization into a 768-dimensional token.

3.3 Training

We train RenderFormer end-to-end using the AdamW [Loshchilov and Hutter 2019] optimizer with a batch size of 128, a linear warm-up learning step size of 1.0×10^{-4} for 8,000 steps followed by a cosine decay schedule on 8 NVIDIA A100 GPUs with 40GB of memory using Flash-Attention 2 [Dao 2024] and Liger Kernel [Hsu et al. 2024] for speed-up. We, first train RenderFormer at 256×256 resolution with a maximum mesh size of 1,536 triangles for $500k$ iterations which took approximately 5 days, followed by $100k$ additional fine-tuning iterations at 512×512 resolution with a maximum mesh size of 4,096 triangles, which took an additional 3 days. While RenderFormer is invariant to scene translations due to the relative spatial positional embedding, it is not invariant to rotations. We therefore improve stability to scene rotations by applying a random rotation to the scene (including the camera); this does not require rerendering and thus rotation is performed on-the-fly during training using RoMa [Brégier 2021].

Loss Function. We train RenderFormer in a supervised manner by computing the L1 loss between a rendered reference HDR image of a synthetic scene and the RenderFormer HDR prediction. To avoid that small errors on bright highlights dominate the loss, we first apply a log transform to the images before computing the L1 loss. In addition, to minimize perceptual differences, we also include an LPIPS loss [Zhang et al. 2018] on a tone-mapped version ($clamp(\log I / \log 2, 0, 1)$) of both images. The final loss is then: $loss_{L1} + 0.05 loss_{LPIPS}$.

Training Data. Our training set consists of synthetic scenes composed of 1 to 3 randomly selected objects from the Objaverse dataset [Deitke et al. 2023] randomly placed in one of four template scenes (shown in Figure 3) that consist of a combination of (randomly translated, rotated, and scaled) ground, back, and side walls. The camera is placed outside the scene such that its view is not blocked by any of the template walls and with a field of view (FOV) uniformly sampled between 30° and 60°. The camera is aimed towards the center (with some perturbations to avoid always aiming at the exact center), at a distance uniformly sampled between 1.5 and 2.0 units, where one unit corresponds to the size of the scene's bounding box. Between 1 to 8 light sources (i.e., triangles with a diffuse emission), with an intensity uniformly sampled between 2,500 and 5,000 $W/units^2$, are placed following a similar procedure as the camera, but with a distance uniformly sampled between 2.1 and 2.7 units. We randomly assign material parameters either per-object or per-triangle with a 1:1 ratio. We randomly assign an RGB color to the diffuse albedo with maximum intensity per color channel set such that the sum with the monochromatic specular albedo lies between 0.9 and 1.0 (uniformly sampled). Roughness is log-sampled in $[0.01, 1.0]$. Furthermore, we randomly select, with equal probability, whether the object is shaded with per-vertex normals or flat-shaded.

The runtime-complexity of attention layers scales quadratically with the number of tokens, and thus triangles in our case. As a result, we limit the total number of triangles in our scenes to 4,096; increasing this limit is an interesting avenue for future research. Since the objects in the Objaverse dataset easily exceed our triangle budget, we remesh the objects by first removing interior or malformed triangles (by converting to a signed distance field followed by a marching cubes step to convert it back to a clean triangle mesh), followed by Qslim to lower the number of faces between 256 to 3,072.

We render 8M HDR training images for 2M synthetic scenes from 4 different viewpoints at 256×256 resolution (with a maximum triangle count of 1,536), and an additional 8M HDR training images at 512×512 resolution with a maximum triangle count of 4,096 using Blender Cycles with 4,096 samples per pixel (using adaptive sampling and denoising).

Table 2. Timing comparison between RenderFormer and Blender Cycles with 4,096 samples per pixels (matching the settings for training data generation). We include both timings with and without adaptive sampling and denoising. In addition, we provide a breakdown of time spent in the view-independent and view-dependent stage. Timings are measured in seconds with pre-cached kernels and excluding the cost of scene loading.

	Figure 1			
	First	Second	Third	Fourth
#Triangles	5366	4400	4527	7321
Cycles 4,096 adaptive spp + denoise	3.97	4.73	3.77	2.71
Cycles 4,096 spp	12.05	11.21	9.95	7.83
RenderFormer	0.0760	0.0613	0.0625	0.0978
View-independent stage	0.0282	0.0186	0.0192	0.0429
View-dependent stage	0.0478	0.0427	0.0433	0.0549

4 Results

We demonstrate RenderFormer on a variety of scenes (Figure 1 and Figure 4) showcasing different aspects of global light transport. For each example, we show a reference render computed with Blender Cycles with 4,096 samples per pixel and a difference image scaled 5× as well as the PSNR, SSIM, LPIPS [Zhang et al. 2018] and HDR-FLIP [Andersson et al. 2020] errors. Qualitatively, the RenderFormer results look visually similar albeit not exactly the same. Nevertheless, RenderFormer manages to include many important light transport effects such as shadows, diffuse and specular interreflections, glossy reflections, and multiple specular interreflections. While not explicitly enforced, RenderFormer is stable to changes in scene parameters

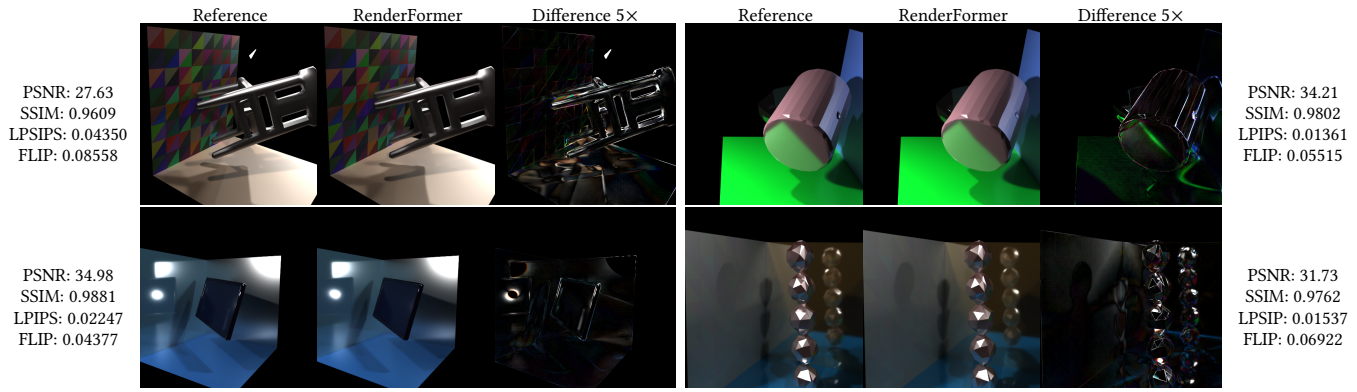

Fig. 4. A variety of scenes rendered with RenderFormer and compared to path-traced reference images. We also list the PSNR, SSIM, LPIPS, and FLIP errors.

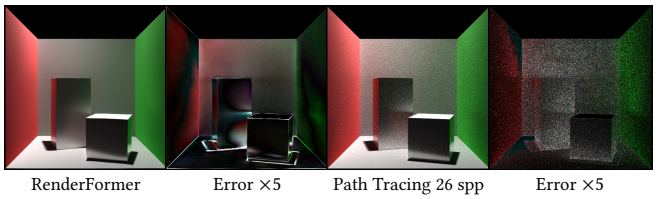

Fig. 5. Equal-time comparison between RenderFormer and Blender Cycles with (non-adaptive) 26 sampler-per-pixel and without denoising.

as shown in the supplemental video where we move the camera, move the lighting, and change reflectance properties.

Table 2 compares the timings on the four scenes in Figure 1 of our unoptimized RenderFormer (pure PyTorch implementation without DNN compilation, but with pre-caching of kernels) and Blender Cycles with 4,096 samples per pixel (matching RenderFormer's training data) at 512×512 resolution on a single NVIDIA A100 GPU. To provide further insight, we also provide a qualitative equal-time comparison (Figure 5) of the first scene from Figure 1; because the fixed cost of denoising exceeds the RenderFormer times and the scene-dependent non-linear cost of adaptive sampling, we disable both optimizations for the equal-time comparison. Besides optimizing the RenderFormer implementation, we can further speed up rendering for static scenes by reusing the view-independent transformed sequence or for animations by rendering 48 frames in parallel by batching.

4.1 Analysis & Ablation Study

RenderFormer's architecture differs from prior neural rendering methods, and it follows a significantly different way of solving the rendering equation compared to classic global illumination methods. To gain more insight on the inner-workings of RenderFormer we perform a series of ablation studies. Due to computational constraints, we perform all ablation experiments at 256×256 resolution.

Relative Spatial Positional Embedding. One of the main differences between RenderFormer and traditional transformers is the positional encoding based on the position of the triangles in world space instead of the sequence index position, using a novel relative spatial positional encoding based on RoPE. However, we also used a NeRF-like positional encoding for embedding the vertex normals. This raises the questions whether it would be possible to also embed the triangle positions together with the normals using a NeRF positional encoding instead. However, we found that training with such positional encoding for the triangles' positions is not stable, and it is prone to converge to a suboptimal local minimum.

Model Size. A key design parameter in transformer models is the token feature length; larger features result in larger models, and thus longer training time. Table 1 (7th to 11th row) lists average PSNR, SSIM, LPIPS [Zhang et al. 2018] and HDR-FLIP [Andersson et al. 2020] errors over 400 test scenes rendered from 4 viewpoints (i.e., 1,600 total) for models trained with feature lengths ranging from 768 to 192. We also adjust the number of attention layers to further reduce the number of model parameters from 205M to 143M, 71M, and 45M parameters respectively. In general, more parameters yield more accurate results.

Number of Layers. In the previous experiment, we purposefully kept the ratio of attention layers between the view-independent and view-dependent stage constant. We perform an additional ablation experiment to better understand the impact of the ratio of attention layers between both stages. Table 1 (rows 11-14) compares RenderFormer models with a different subdivisions of a total of 18 attention layers over the two stages. Figure 6 qualitative shows the impact of varying the attention layers per stage. We observe that RenderFormer benefits from including more attention-layers in the view-dependent stage than in the view-independent stage. Fully eliminating the view-independent stage (Table 1, 14th row and Figure 6, 2nd column) fails to produce good results, indicating that the view-independent stage is necessary for obtaining good results. However, rendering often requires a careful balance between accuracy and speed. The runtime of each stage depends on different factors. The view-independent stage scales roughly by $O(\#tris^2)$, whereas the view-dependent layers scales by $O(\#bundles^2 + \#bundles \times \#tris)$. Furthermore, the difference in precision (bf16 versus tf32) imposes an additional hardware-dependent performance scale between both stages. The ideal number of attention layers per stage is complex and depends on mesh size, resolution, and hardware. We therefore

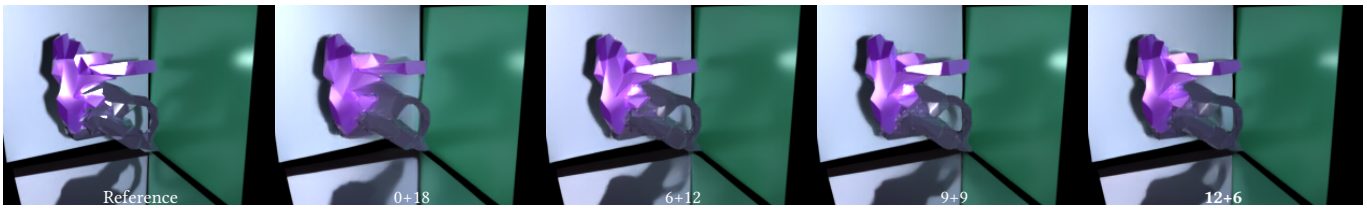

Fig. 6. Qualitative comparison of varying `#view-independent + #view-dependent` attention layers per stage. RenderFormer is shown in the last column with a ratio of 12 view-independent versus 6 view-dependent layers.

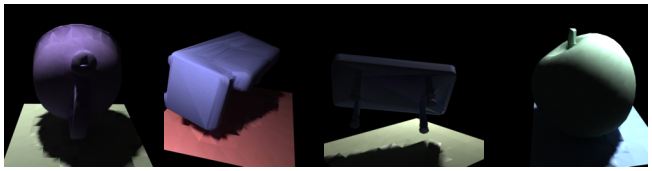

Fig. 7. Visualization of the transformed tokens from the view-independent stage that (after transformation) encode smooth diffuse shading and inter-reflections, as well as shadows at sub-triangle granularity.

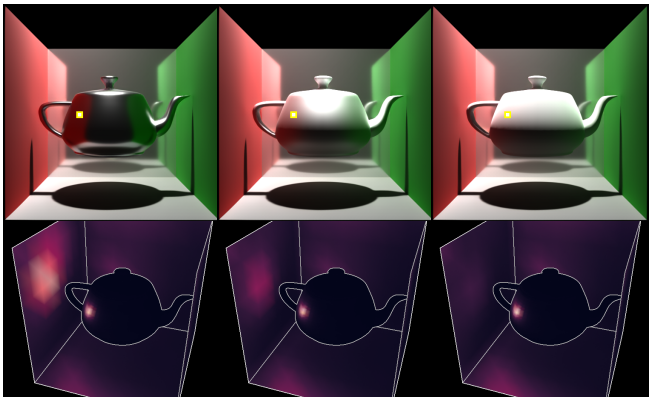

Fig. 8. Visualization of the average attention per triangle for a given ray-bundle in the view-dependent stage. The average attention gives an indication on which triangles RenderFormer uses for computing the outgoing radiance for the rays in the bundle. As expected directly visible triangles and triangles around the reflected direction receive the most attention.

opt for a 12+6 split between view-independent and view-dependent attention layers, balancing accuracy and training/render speed (i.e., ~25% faster for ~5% loss in accuracy).

Role of the Different Stages. The previous ablation study clearly shows that both stages are necessary and each serve a role in the rendering pipeline. However, the previous ablation experiments do not give insights on what exactly each stage does.

The interpretation of the embedding of the triangle tokens does not follow the initial embedding after passing through the view-independent stage, precluding direct visualization of the triangle tokens. We therefore train an small auxiliary MLP that casts a transformed triangle token into a 32×32 RGB texture for each triangle.

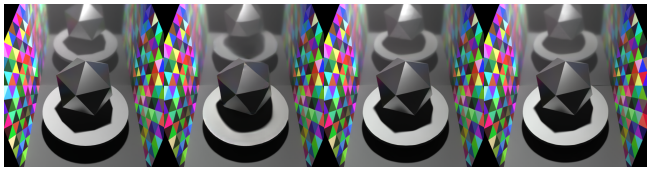

Fig. 9. Using larger than normal triangles for the pedestal and icosphere results in degraded shadows and shading (2nd column). Interestingly, this degradation is also visible in the reflections in the back wall.

Because the register tokens might include important information, we also include a cross-attention layer between each triangle token and the 16 register tokens. We train the MLP and the cross-attention layers, while keeping the view-independent stage frozen, on a small batch (~ 500) of simple solid colored diffuse scenes. The MLP is intentionally kept shallow to limit it to simple operations that directly visualize the information embedded in the tokens. Figure 7 shows that the view-independent stage resolves a significant portion of diffuse light transport between triangles as well as shadows.

The view-dependent stage gathers information from the triangle tokens to compute the observed radiance for each pixel. In Figure 8, we visualize the (sum of the) attention weights projected on their respective triangles for selected ray-bundles and visualized from an appropriate view. This visualization shows how much each triangle contributes to the final radiance observed for the given ray-bundle. From Figure 8 we can see that the main weight lies on the directly visible triangle, as well as triangles around the reflected direction. We can also see that the weight distribution changes when we increase the roughness of the material.

4.2 Generalization of Scene Parameters

While the previous ablation study and analysis provides more insight on the inner-workings of RenderFormer, it does not give an indication on how the model performs in practical situations and what its limits are. Therefore, we perform several experiments to probe RenderFormer's generalization capabilities with respect to the triangle mesh, light sources, and camera.

Triangle Mesh. Currently the training data is generated such that the triangles are all roughly the same size. To better understand if the triangle size affects the accuracy of the solution, we perform an experiment where we render a scene twice (Figure 9), once where the pedestal and icosphere are represented by 1,318 triangles of

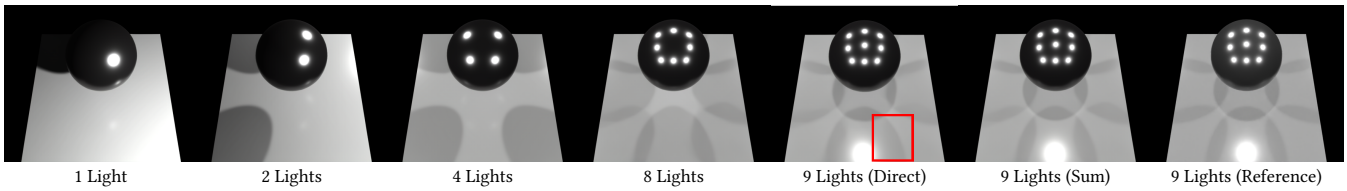

Fig. 10. RenderFormer can handle multiple light sources with correct reflections and shadows (1st to 4th column) as long as the number of lights does not exceed 8 (as seen in training). For more lights, highlights or shadow might be missing (e.g., the missing double shadow at the bottom shadow in the 5th column). In such a case, we can still compute a correct result by compositing multiple single-light images (6th column).

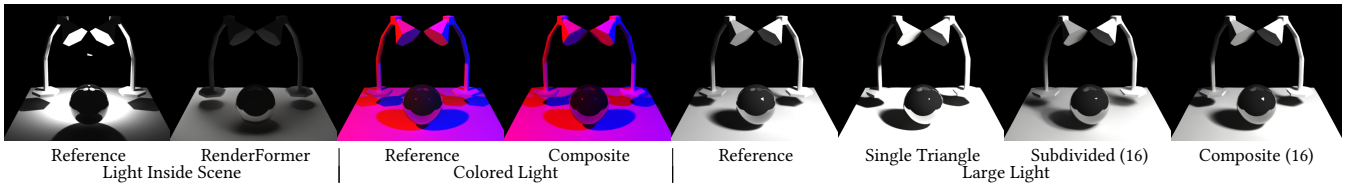

Fig. 11. Left: RenderFormer was never trained with lights inside the scene, and thus fails to correctly render such scenes. Middle: RenderFormer can simulate colored lights by leveraging linearity of light transport and blending three images (one for each color channel). Right: RenderFormer fails to correctly render scenes with light sources larger than those encountered during training. Subdividing the triangle can correct the error if the number of light sources does not exceed the maximum seen during training (8), in which case we can still leverage linearity of light transport by rendering each subdivided light separately.

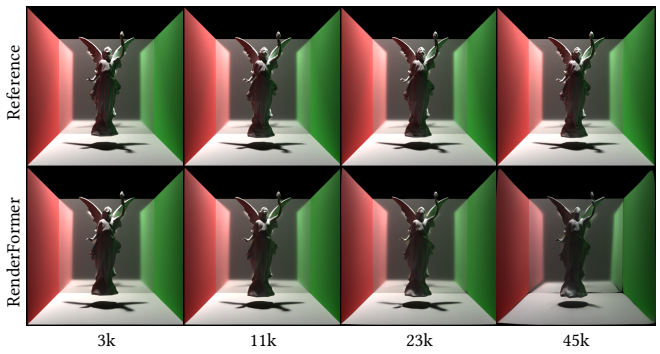

Fig. 12. RenderFormer can handle scenes with more triangles than for which it was trained, albeit with loss of detail and thin features.

average size, and once with 332 larger triangles. As can be seen in Figure 9 (2nd column), the quality of the shading and shadows over larger triangles degrades due to the fact that the triangle-tokens now need to store more complex information per triangle.

The attention layers that form the core of transformers are costly in terms of compute resources. Therefore, the training set for RenderFormer is limited to scenes with at most 4,096 triangles. Figure 12 shows that RenderFormer can handle larger triangle-meshes at inference time, albeit with some loss of details. However, we observe that overall RenderFormer fails gracefully with most of the light transport correctly modeled. We exploit this property during training by first pretraining on smaller scenes (1,536 triangles) and then refine on larger scenes (4,096 triangles). However, due to the $O(N^2)$ complexity of the attention layers, there is a limit to how far the model can be pushed before running out of resources.

Many attention-optimization techniques used in LLMs and Vision Transformers [Dong et al. 2023; Liu et al. 2021; Wang et al. 2021b] leverage properties inherent to 1D sequences or 2D images whereas RenderFormer operates in 3D, making direct application difficult. Adapting state-of-the-art techniques from LLMs and Vision Transformers (such as linear attention mechanisms, native sparse attention, and sequence parallelism) is a promising avenue for future research, especially when combined with established computer graphics methodologies such as LoD and BVH.

Lighting. RenderFormer is currently trained for scenes with 1 to 8 light sources. Figure 10 (columns 1 to 4) shows a sequence of images of a scene with an increasing number of light sources. We observe that RenderFormer does not reliably handle cases where the number of light sources exceeds the maximum seen during training causing incomplete shadows or missing highlights (Figure 10, 5th column). The maximum number of light sources can either be increased by training with more lights, or by exploiting linearity of light transport by rendering each light source separately and adding the resulting images (Figure 10, 6th column).

Currently, RenderFormer is also trained with light sources positioned outside the scene, and placing a light source in the scene yields an incorrect result (Figure 11, 2nd column). Moreover, RenderFormer is also trained for white light sources only; RenderFormer ignores the color when it encounters a colored light. This problem can be solved by either expanding the training set or by leveraging linearity of light transport (Figure 11, 4th column).

In addition, RenderFormer is trained for a limited range in light source size. As expected, exceeding the trained light source size (Figure 11, 6th column) does not produce the correct shadows. We can either expand the training set, or construct larger light sources by subdividing the light source in more (smaller) triangles (Figure 11, 7th (direct render) and 8th column (composite render)).

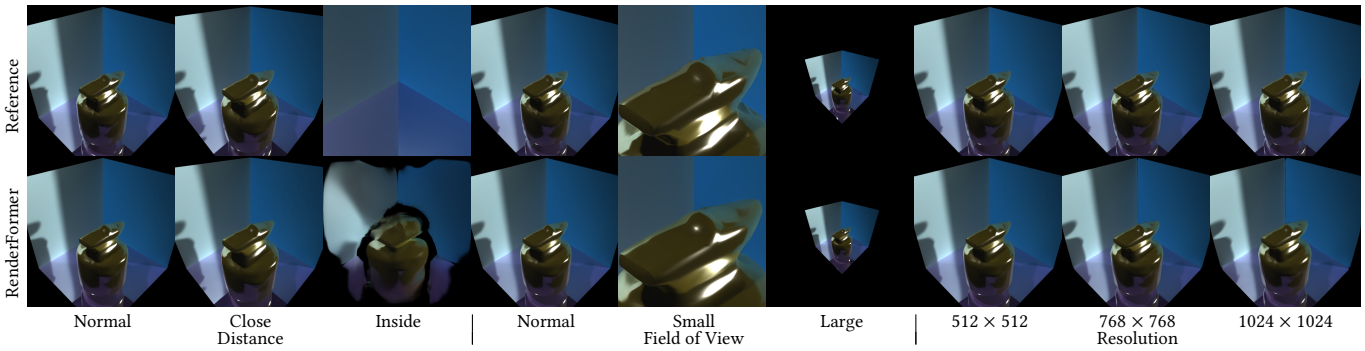

Fig. 13. RenderFormer is robust to moving the camera closer than seen during training (2nd column), as long as the camera remains outside the scene (3rd column). RenderFormer is also robust to exceeding the field of view seen during training (4th-6th column). We also found that RenderFormer fails gracefully when rendering at higher resolutions (7th-9th column), with differences around depth discontinuities (e.g., between the gray and blue walls).

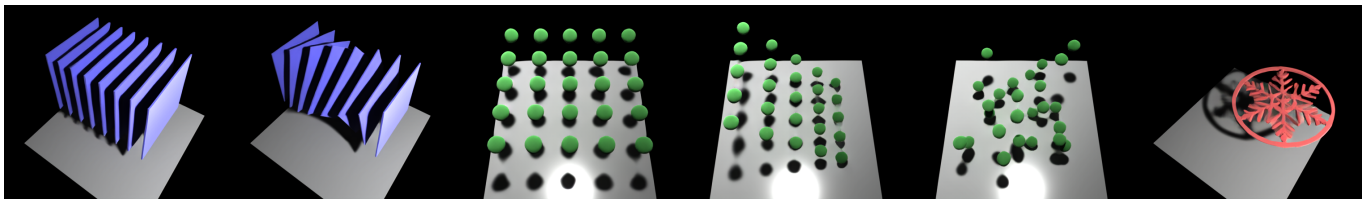

Fig. 14. RenderFormer can correctly reproduce occlusions for scenes with many objects. However, the shadows cast by occluders with very complex shapes, can result in a loss of detail in the cast shadow.

Camera Parameters. Similar to light sources, RenderFormer is also only trained for a camera located outside the scene. As a result RenderFormer has never learned that triangles can be placed behind the camera, and fails to correctly render such scenes (Figure 13, 3nd column). Extending the training set to include such cases could allow RenderFormer to learn how to handle such cases.

RenderFormer is also trained for a limited range of FOV. However, from Figure 13 (4th to 6th column) we can see that RenderFormer appears to be robust to going outside this range.

RenderFormer is also only trained for a fixed 512×512 resolution. While in theory the ray-bundles can model higher resolution, we found that RenderFormer exhibits a minor resolution dependence. As shown in Figure 13 (8th and 9th column), when applied to higher resolutions, RenderFormer fails gracefully, with most of the errors focused around depth discontinuities. We also exploit this property by first training RenderFormer at lower resolutions (256×256), and then fine-tuning it at 512×512 resolution.

Scene Complexity. Finally, we explore the accuracy of RenderFormer with respect to scene complexity. In particular, we investigate if RenderFormer can handle complex occluders and multiple specular reflections. Figure 14 shows scenes with increasing occluder complexity. The first two columns show a series of closely packed planes that cast shadows between the planes and the floor. Figure 14 (3rd to 5th column) shows a series of small occluders placed at varying distances between the ground plane and light source. We observe that the shadows of occluders with complex shapes sometimes miss fine details (Figure 14, columns 3 to 5).

Figure 15 shows how well RenderFormer handles multiple bounces of reflections. While RenderFormer does not reason in terms of physical bounces of light transport, we find that RenderFormer correctly models on average 3 bounces of specular reflections, but higher-order bounces are dropped (e.g., the 2nd reflection of red ball on the back wall in the last example in Figure 15). We found that the reflection depth is independent of the number of view-dependent layers, and we posit that this limitation is mainly due to the scarcity of training examples with higher-order specular bounces, and careful augmentation of the training dataset could improve performance for multi-bounce reflections.

Textures. RenderFormer assumes constant reflectance properties over a triangle. We perform an exploratory experiment to extend RenderFormer to include spatially-varying surface reflectance by modifying the reflectance token embedding. Instead of expanding the stacked reflectance parameters to a 768-dimensional vector, we directly encode spatially varying information at the triangle level. To embed the spatially-varying parameters (i.e., 13 channels containing diffuse albedo, specular albedo, roughness, and surface normal), we first rasterize the parameters to an isosceles right triangle at 32×32 resolution. Next, we concatenate all texels in a 13,312-dimensional vector (i.e., 32×32 texels and 13 channels per texel) that is encoded by a single linear layer followed by RMS-Normalization into a 768-dimensional token. Our initial results (Figure 16) show that RenderFormer is able to model spatially varying reflectance, albeit blurred. Expanding the token length might improve texture quality; we leave this for future research.

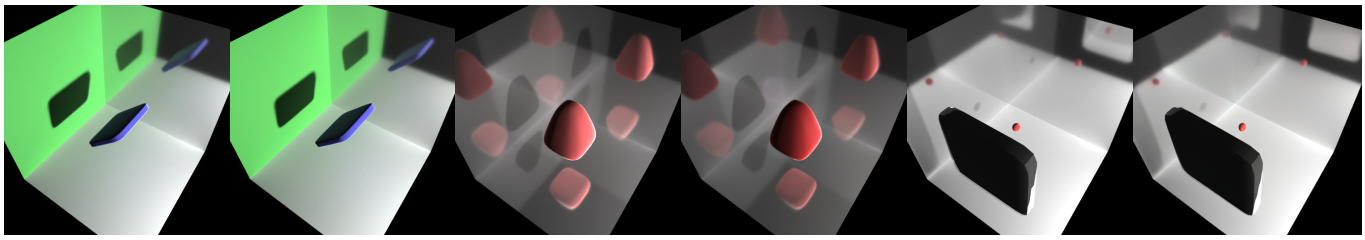

Fig. 15. RenderFormer correctly handles 1 and 2 recursive specular interreflections. However, due to the scarcity of training exemplars with more specular interreflections, it does not always correctly resolve higher order reflections (e.g., the reflection of the red ball in the reflection of the mirror on the top wall).

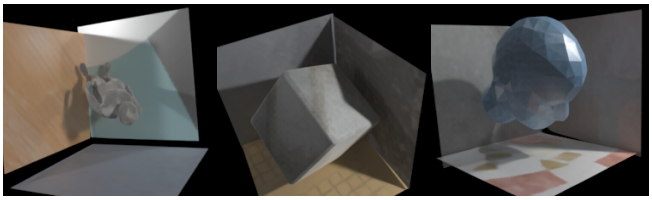

Fig. 16. Preliminary results of extending RenderFormer to support spatially-varying material properties.

5 Conclusion

In this paper we introduce RenderFormer, a transformer-based neural rendering pipeline that takes as input a regular triangle mesh, and outputs an image of the scene accounting for global illumination. While RenderFormer is limited in the scenes it can render (i.e., limited triangle count, number of light sources, camera positions, etc...), it generalizes better than prior neural rendering systems. RenderFormer approaches solving light transport in a virtual scene as a two-stage sequence-to-sequence transformation. The first stage transforms a triangle-sequence to model view-independent triangle-to-triangle transport. The second stage transforms a sequence of ray-bundles to a sequence of corresponding observed radiance values guided by the triangle-sequence from the first stage.

There are ample avenues to further improve RenderFormer. First, we can expand the training set to support a wider variety of camera and light positions. Furthermore, while our current implementation utilizes training data rendered using the GGX BRDF model, we impose no inherent architectural restrictions related with the reflectance model. Hence, RenderFormer could be trained on alternative datasets using other reflectance models including those that model transparency or subsurface scattering. Currently, RenderFormer only supports simple light sources, and extensions to environment lighting and non-diffuse light sources would further generalize RenderFormer. Since RenderFormer is fully transformer based, it is inherently differentiable, allowing us to train RenderFormer directly from data. An interesting and promising direction for future work that leverages the inherent differentiability, would be to apply RenderFormer to inverse rendering applications. Finally, would like to investigate hierarchical attention methods based on existing grouping based acceleration structures for classic rendering methods (e.g., BVH) to support more complex scenes with larger triangle-meshes.

Acknowledgments

We would like to thank Kexun Zhang and Kaiqi Chen for discussions on transformer model design and performance optimizations, and Sam Sartor for Blender Cycle tips and pre-reviewing this work. Pieter Peers was supported in part by NSF grant IIS-1909028. Chong Zeng and Hongzhi Wu were partially supported by NSF China (62332015, 62227806 & 62421003), the XPLORER PRIZE, and Information Technology Center and State Key Lab of CAD&CG, Zhejiang University.

References

Pontus Andersson, Jim Nilsson, Tomas Akenine-Möller, Magnus Oskarsson, Kalle Åström, and Mark D Fairchild. 2020. FLIP: A Difference Evaluator for Alternating Images. *Proc. ACM Comput. Graph. Interact. Tech.* 3, 2 (2020).

Steve Bako, Thijs Vogels, Brian Mcwilliams, Mark Meyer, Jan NováK, Alex Harvill, Pradeep Sen, Tony Derose, and Fabrice Rousselle. 2017. Kernel-predicting convolutional networks for denoising Monte Carlo renderings. *ACM Trans. Graph.* 36, 4 (2017).

Romain Brégier. 2021. Deep Regression on Manifolds: a 3D Rotation Case Study. *International Conference on 3D Vision.*

Michael F Cohen, Shenchang Eric Chen, John R Wallace, and Donald P Greenberg. 1988. A progressive refinement approach to fast radiosity image generation. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*. 75–84.

Arno Coomans, Edoardo Alberto Dominici, Christian Döring, Joerg H. Mueller, Jozef Hladky, and Markus Steinberger. 2024. Real-time Neural Rendering of Dynamic Light Fields . *Comp. Graph. Forum* (2024).

Tri Dao. 2024. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. In *ICLR.*

Timothée Darcet, Maxime Oquab, Julien Mairal, and Piotr Bojanowski. 2024. Vision Transformers Need Registers. In *ICLR.*

Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. 2023. Objaverse: A universe of annotated 3d objects. In *CVPR.* 13142–13153.

Zican Dong, Tianyi Tang, Lunyi Li, and Wayne Xin Zhao. 2023. A survey on long text modeling with transformers. *arXiv preprint arXiv:2302.14502* (2023).

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *ICLR.*

Philip Dutré, Philippe Bekaert, and Kavita Bala. 2018. *Advanced global illumination.* AK Peters/CRC Press.

Duan Gao, Haoyuan Mu, and Kun Xu. 2022. Neural global illumination: Interactive indirect illumination prediction under dynamic area lights. *Trans. Vis. and Comp. Graph.* 29, 12 (2022), 5325–5341.

Ruiqi Gao, Aleksander Holynski, Philipp Henzler, Arthur Brussee, Ricardo Martin-Brualla, Pratul P. Srinivasan, Jonathan T. Barron, and Ben Poole*. 2024. CAT3D: Create Anything in 3D with Multi-View Diffusion Models. *NeurIPS* (2024).

Cindy M Goral, Kenneth E Torrance, Donald P Greenberg, and Bennett Battaile. 1984. Modeling the interaction of light between diffuse surfaces. *ACM SIGGRAPH computer graphics* 18, 3 (1984), 213–222.

Jonathan Granskog, Fabrice Rousselle, Marios Papas, and Jan Novák. 2020. Compositional neural scene representations for shading inference. *ACM Trans. Graph.* 39, 4 (2020).

Jonathan Granskog, Till N Schnabel, Fabrice Rousselle, and Jan Novák. 2021. Neural scene graph rendering. *ACM Trans. Graph.* 40, 4 (2021).

Saeed Hadadan, Shuhong Chen, and Matthias Zwicker. 2021. Neural radiosity. *ACM Trans. Graph.* 40, 6 (2021).

Ayaan Haque, Matthew Tancik, Alexei A Efros, Aleksander Holynski, and Angjoo Kanazawa. 2023. Instruct-nerf2nerf: Editing 3d scenes with instructions. In *CVPR*. 19740–19750.

Alex Henry, Prudhvi Raj Dachapally, Shubham Shantaram Pawar, and Yuxuan Chen. 2020. Query-Key Normalization for Transformers. In *EMNLP*. 4246–4253.

Pin-Lun Hsu, Yun Dai, Vignesh Kothapalli, Qingquan Song, Shao Tang, Siyu Zhu, Steven Shimizu, Shivam Sahni, Haowen Ning, and Yanning Chen. 2024. Liger Kernel: Efficient Triton Kernels for LLM Training. *arXiv preprint arXiv:2410.10989* (2024). arXiv:2410.10989 [cs.LG] https://arxiv.org/abs/2410.10989

Haian Jin, Hanwen Jiang, Hao Tan, Kai Zhang, Sai Bi, Tianyuan Zhang, Fujun Luan, Noah Snavely, and Zexiang Xu. 2024. Lvsm: A large view synthesis model with minimal 3d inductive bias. *arXiv preprint arXiv:2410.17242* (2024).

James T Kajiya. 1986. The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*. 143–150.

Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of naacL-HLT*, Vol. 1. 2.

Yixun Liang, Hao He, and Yingcong Chen. 2024. Retr: Modeling rendering via transformer for generalizable neural surface reconstruction. *NeurIPS* 36 (2024).

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. 2021. Swin transformer: Hierarchical vision transformer using shifted windows. In *CVPR*. 10012–10022.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. In *ICLR*.

Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. 2021. Nerf: Representing scenes as neural radiance fields for view synthesis. *Commun. ACM* 65, 1 (2021), 99–106.

Thomas Müller, Fabrice Rousselle, Jan Novák, and Alexander Keller. 2021. Real-time neural radiance caching for path tracing. *ACM Trans. Graph.* 40, 4 (2021).

Thu Nguyen-Phuoc, Chuan Li, Stephen Balaban, and Yong-Liang Yang. 2018. RenderNet: A deep convolutional network for differentiable rendering from 3D shapes. In *NeurIPS*.

FE Nicodemus, JC Richmond, JJ Hsia, IW Ginsberg, and T Limperis. 1992. Geometrical considerations and nomenclature for reflectance. In *Radiometry*. 94–145.

Matt Pharr, Wenzel Jakob, and Greg Humphreys. 2023. *Physically based rendering: From theory to implementation*. MIT Press.

Gilles Rainer, Adrien Bousseau, Tobias Ritschel, and George Drettakis. 2022. Neural Precomputed Radiance Transfer. *Comp. Graph. Forum* 41, 2 (April 2022).

René Ranftl, Alexey Bochkovskiy, and Vladlen Koltun. 2021. Vision transformers for dense prediction. In *CVPR*. 12179–12188.

Jeremy Reizenstein, Roman Shapovalov, Philipp Henzler, Luca Sbordone, Patrick Labatut, and David Novotny. 2021. Common objects in 3d: Large-scale learning and evaluation of real-life 3d category reconstruction. In *ICCV*. 10901–10911.

Konstantinos Rematas and Vittorio Ferrari. 2020. Neural Voxel Renderer: Learning an Accurate and Controllable Rendering Tool. In *CVPR*. 5416–5426.

Haocheng Ren, Yuchi Huo, Yifan Peng, Hongtao Sheng, Weidong Xue, Hongxiang Huang, Jingzhen Lan, Rui Wang, and Hujun Bao. 2024. LightFormer: Light-Oriented Global Neural Rendering in Dynamic Scene. *ACM Trans. Graph.* 43, 4 (July 2024).

Mehdi SM Sajjadi, Henning Meyer, Etienne Pot, Urs Bergmann, Klaus Greff, Noha Radwan, Suhani Vora, Mario Lučić, Daniel Duckworth, Alexey Dosovitskiy, et al. 2022. Scene representation transformer: Geometry-free novel view synthesis through set-latent scene representations. In *CVPR*. 6229–6238.

Paul Sanzenbacher, Lars Mescheder, and Andreas Geiger. 2020. Learning neural light transport. *arXiv preprint arXiv:2006.03427* (2020).

Noam Shazeer. 2020. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202* (2020).

Peter-Pike Sloan, Jan Kautz, and John Snyder. 2023. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*. 339–348.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024a. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing* 568 (2024), 127063.

Rui Su, Honghao Dong, Jierui Ren, Haojie Jin, Yisong Chen, Guoping Wang, and Sheng Li. 2024b. Dynamic Neural Radiosity with Multi-grid Decomposition. In *SIGGRAPH Asia 2024 Conference Papers*. 1–12.

Mohammed Suhail, Carlos Esteves, Leonid Sigal, and Ameesh Makadia. 2022. Light field neural rendering. In *CVPR*. 8269–8279.

A. Tewari, J. Thies, B. Mildenhall, P. Srinivasan, E. Tretschk, W. Yifan, C. Lassner, V. Sitzmann, R. Martin-Brualla, S. Lombardi, T. Simon, C. Theobalt, M. Niessner, J. T. Barron, G. Wetzstein, M. Zollhöfer, and V. Golyanik. 2022. Advances in Neural Rendering. *Comp. Graph. Forum* 41, 2 (2022), 703–735.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

Mukund Varma, Peihao Wang, Xuxi Chen, Tianlong Chen, Subhashini Venugopalan, and Zhangyang Wang. 2022. Is attention all that NeRF needs?. In *ICLR*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NeurIPS*. 6000–6010.

Bruce Walter, Stephen R. Marschner, Hongsong Li, and Kenneth E. Torrance. 2007. Microfacet models for refraction through rough surfaces. In *EGSR*. 195–206.

Qianqian Wang, Zhicheng Wang, Kyle Genova, Pratul P Srinivasan, Howard Zhou, Jonathan T Barron, Ricardo Martin-Brualla, Noah Snavely, and Thomas Funkhouser. 2021a. Ibrnet: Learning multi-view image-based rendering. In *CVPR*. 4690–4699.

Qi Wang, Zhihua Zhong, Yuchi Huo, Hujun Bao, and Rui Wang. 2023. State of the Art on Deep Learning-enhanced Rendering Methods. *Machine Intelligence Research* 20, 6 (2023), 799–821.

Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. 2021b. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. In *ICCV*. 568–578.

Yu-Jie Yuan, Yang-Tian Sun, Yu-Kun Lai, Yuewen Ma, Rongfei Jia, and Lin Gao. 2022. Nerf-editing: geometry editing of neural radiance fields. In *CVPR*. 18353–18364.

Biao Zhang and Rico Sennrich. 2019. Root mean square layer normalization. *NeurIPS* 32 (2019).

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. 2018. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*. 586–595.

Chuankun Zheng, Yuchi Huo, Hongxiang Huang, Hongtao Sheng, Junrong Huang, Rui Tang, Hao Zhu, Rui Wang, and Hujun Bao. 2024. Neural Global Illumination via Superposed Deformable Feature Fields. In *SIGGRAPH Asia 2024 Conference Papers*. 1–11.