RenderFormer-V2: Neural Rendering with Heterogeneous Scene Primitives

Chong Zeng[1], Yue Dong[2], Pieter Peers[3], Lvmin Zhang[1], and Maneesh Agrawala[1]

[1] Stanford University, USA
[2] Microsoft Research, China
[3] College of William & Mary, USA

Abstract. We present 'RenderFormer-V2', a unified learned transformer-based neural rendering model, complementary to modern physics-based rendering systems, that can handle diverse light-transport effects such as caustics, volumetric scattering, environment lighting, textured and displaced surfaces and out-of-distribution materials without per-scene training or specialized code. RenderFormer-V2 models global light transport as a sequence-to-sequence transformation. Following its predecessor, RenderFormer-V2 also employs a two stage process: a view-independent stage that resolves intra-scene primitive to primitive transport, and a view-dependent stage that transforms the internal neural scene representation into image pixels. Different from RenderFormer, our model employs a novel combined windowed-attention and rendering-informed attention sink in the view-independent stage to improve scalability while maintaining render accuracy. To further improve versatility, RenderFormer-V2 supports heterogeneous scene primitives, including environment maps and participating media, and it employs a material encoding independent of the underlying surface reflectance model that encodes material appearance via a novel neural embedding. We demonstrate the versatility of RenderFormer-V2 on a variety of scenes and perform an extensive ablation of the improved attention mechanism.

Keywords: Neural Rendering, Transformer, Neural Material Embedding, Windowed Attention, Attention Sink

1 Introduction

Neural rendering aims to visualize virtual scenes without relying on manually encoded rules of light transport, but instead based on relations between geometry, materials, and light learned from data. Many neural rendering solutions offer limited generalizability beyond the training data [10, 11] or rely on per-scene training strategies [28]. Recently, RenderFormer [41] formulated light transport simulation as a regressive sequence-to-sequence translation problem, where an input sequence of triangle tokens is transformed into pixel-patch tokens through a transformer-based two stage pipeline: a view-independent stage that resolves

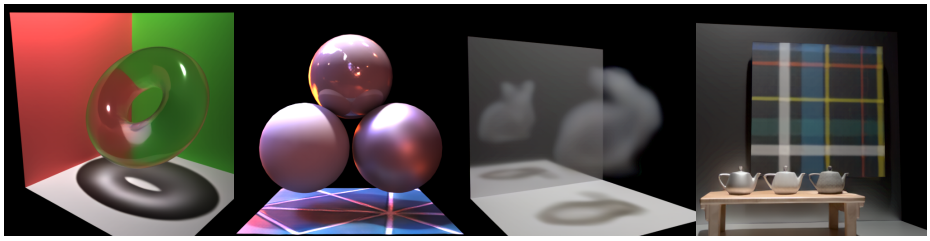


Fig. 1: RenderFormer-V2 can simulate global light transport in scenes that include transparent materials with caustics, environment lighting, volumetric scattering, textures, and scenes with over 100k primitives, without the need for per-scene training.

light transport between triangles, and a view-dependent stage that resolves transport from triangles to the camera. Once trained, RenderFormer can render a wide variety of virtual scenes without fine-tuning or further training. Although RenderFormer is more general than prior solutions, it is still far from practical: it is limited to scenes of less than 4k triangles, it only supports a hard-coded GGX BRDF model [31], and it is limited to scenes with (max. 8) triangular diffuse light sources.

In this paper we introduce 'RenderFormer-V2', a versatile transformer-based neural rendering system that addresses RenderFormer's limitations via a number of carefully designed architectural innovations (Figure 1). Similar to RenderFormer, RenderFormer-V2 formulates light transport simulation as a regressive sequence-to-sequence translation. RenderFormer's main bottleneck in supporting larger triangle meshes is the brute-force self-attention between triangle tokens in the view-independent stage. Not only does this have a quadratic complexity with respect to the number of triangles, it also leads RenderFormer to loose focus for very large triangle meshes. Inspired by recent advances in supporting larger context windows for large-language models, RenderFormer-V2 employs a novel sparse-attention variant, consisting of a combination of windowed attention [19] and render-aware attention-sinks [35], tuned for resolving view-independent light transport. Furthermore, to improve generalizability, we allow for other primitives than triangles (e.g., voxels) and employ a simpler positional encoding based on the centroid of the primitive.

RenderFormer-V2 further decouples the material specification from a hard-coded BRDF model, and instead employs a latent material embedding for encoding different BRDF models including transparent and measured materials. A key observation is that the latent material encoding does not need to be invertible to the input (BRDF) parameters, but it only needs to encode the appearance of the material; we rely on RenderFormer-V2 to learn how to map the material appearance into pixel values. Moreover, to model spatially varying materials, we reuse a pretrained VAE encoder [33] to encode 32×32 texture patches (of latent material properties) per primitive. Similar to the latent material appearance space, we only require the VAE encoder, and let RenderFormer-V2 learn how to interpret the encoded textures during rendering.

Whereas RenderFormer has a dedicated emittance parameter associated with each triangle to model light sources, we leverage RenderFormer-V2's ability to mix different primitives to embed different lighting types, ranging from triangular light sources to environment maps, into specialized tokens.

We demonstrate the versatility of RenderFormer-V2 by rendering more complex and larger scenes than RenderFormer with a greater variety in lighting and materials. We perform an in-depth ablation study to validate our design decisions. The trained RenderFormer-V2 model and code can be found at: `https://renderformer.github.io/v2`.

2 Related Work

Neural Rendering [28] aims to predict the effects of light transport through a virtual scene. Early work in neural rendering employs specially learned neural representations of the scene [10, 11, 13, 40, 46] and thus are overfitted to a single or limited number of scenes. To circumvent the need to learn neural scene representations, image-space neural rendering systems [18, 22, 42] take as input G-buffers of intrinsic components of the scene, and output a shaded image seen from the same viewpoint. Because the G-buffers only capture a portion of the scene, image-space neural rendering methods must necessarily hallucinate (or ignore) transport between visible and non-visible parts of the scene.

Recently, a new class of neural rendering systems leverage attention layers [29] to model light transport between 3D primitives. Xu et al. [36] model diffuse light transport in a point cloud representation of the scene. Closest to our method is RenderFormer [41] which employs a two-stage transformer architecture that models the transport: (1) between triangles and (2) from the triangles to the camera. However, RenderFormer employs a brute-force attention mechanism which does not scale well to large triangle meshes. Moreover, RenderFormer only supports triangles as geometric primitives, diffuse (triangle-shaped) light sources, and a per-triangle hard-coded GGX microfacet BRDF model [31]. In contrast, RenderFormer-V2 employs an efficient sparse attention mechanism to support a large number (> 100k) of primitives, and flexible geometry, lighting, and materials representations and textures.

Long Context Modeling with Transformers Classic transformers compute attention between all pair-wise token combinations, resulting a quadratic complexity with respect to the number of tokens in the sequence. Moreover, when the sequence grows, attention per-token tends to decrease and be spread over many tokens, and as a consequence the transformer loses focus, resulting in a decreased performance. Addressing both issues is critical for scaling a transformer-based rendering architecture beyond a few thousand tokens. Here, we focus on the most relevant classes of transformer scaling methods, and refer to Tay et al. [27] for a detailed overview.

Windowed attention mechanisms [3, 19, 34, 38] focus on addressing the compute complexity, and built on the observation that in many cases proximity

is a good indicator of importance, and hence these mechanism hard-constrain the attention computation to a small window around the target token. Consequently, windowed attention mechanisms ignore long-range interactions which can be important for light transport modeling. More generally, windowed attention mechanisms belong to a class of *sparse* attention methods that employ static attention patterns [14, 17, 23] and their effectiveness is highly dependent on whether the attention sparsity matches the attention pattern. While light transport through a scene can be sparse, it does not follow a pre-determined sparsity pattern.

Native-Sparse Attention (NSA) [39] dynamically determines the sparseness by employing three different attention streams: (i) a sliding window to capture local attention, (ii) compressed attention that determines the importance of groups of input tokens, and (iii) a fine-grained attention on the groups of tokens identified as important. However, the computational cost of NSA is significantly higher than static sparse attention patterns due to the secondary retrieval stage. Moreover, NSA requires Grouped-Query Attention [1] which lowers the model's capacity, and thus adversely affects performance.

Hierarchical attention mechanisms (e.g., [37, 43, 48]) leverage the observation that attention tends to be focused near the query, and that the attention variation at distant tokens decreases. Hence, by creating a multi-resolution hierarchy of token and computing attention with the token selected from the hierarchy based on distance, attention can be better focused and more efficiently computed. However, multi-resolution hierarchies implicitly assume that positional distance is proportional to distance in the sequence or image, and thus implicitly assume a (regular) uniform spatial distribution of tokens. This is not the case for 3D scenes, where primitives are clustered at various points in space (i.e., objects). Point Transformer v3 [34] addresses this limitation by (i) serializing the point cloud along space-filling curves, and (ii) grouping and padding to ensure the point cloud is divisible by the target patch size. While, Point Transformer v3 improves speed and memory overhead, its implementation is more complex and is computationally more expensive than sparse attention methods. We employ a less complex and resource intensive strategy for extending the context window using a similar serialization strategy as Point Transformer v3.

Xiao et al. [35] observed that the soft-max operation in the attention computation tends to *'dump'* excess attention in a single token (i.e., attention sink). A similar behavior was also observed in vision transformers [16]. To avoid attention being dumped in a random token, Xiao et al. [35] propose to keep a few dedicated attention sink tokens to model global relations and a local sliding windowed attention to model local relations. This local-global dichotomy has been further refined in follow up work [21, 45]. We also build on this idea, and introduce rendering-relevant semantics for the sinks. First, we place all light sources in the sinks as these are likely to interact with all surfaces. Second, inspired by the compressed tokens in NSA [39], we add to the sink summarization tokens for groups of primitives based on Hilbert space-filling curves.

3 Background - RenderFormer

RenderFormer-V2 builds and improves on RenderFormer [41]. We therefore first review RenderFormer's architecture before detailing RenderFormer-V2.

RenderFormer is an end-to-end trained transformer-based neural renderer that takes as input a sequence of triangles with GGX BRDF parameters [31] and emittance strength, as well as camera parameters, and it outputs a rendered image of the scene with full global illumination. RenderFormer consist of two stages with a slightly different architecture. The first (i.e., view-independent) stage, consisting of 12 self-attention layers [29], transforms the input sequence of embedded triangle tokens (expressed in *world* coordinates) to a sequence of per-triangle tokens which encode triangle-to-triangle light transport. The second stage (i.e., view-dependent stage), consisting of 6 repetitions of a cross-attention layer [29] followed by a self-attention layer, operates on view-bundle tokens. A view-bundle token is an embedding of a 8×8 grid of camera rays expressed in the *camera* coordinate system. The cross-attention layer computes the attention between the view-bundle tokens and the transformed triangle tokens from the first stage. The view-dependent stage is followed by a dense vision transformer to convert the transformed ray-bundle tokens into pixel values for each ray. The triangles are embedded as the sum of: (a) the per-vertex normal embedding (using NeRF positional encoding with 6 frequencies that is subsequently expanded to the 768 token-length vector through a linear layer), (b) the GGX BRDF parameters (expanded by a linear layer to the 768 token-length vector), and (c) the monochrome emittance (expanded by a linear layer). RenderFormer adapts RoPE [25] to apply a relative positional encoding on the 9D vector obtained by stacking the 3D coordinates of the triangle's vertices. RoPE is applied at each layer in RenderFormer with the vertices expressed in world coordinates in the view-independent stage and in camera coordinates in the view-dependent stage. Hence, only the ray direction of the 8×8 camera rays is embedded by stacking the 64 view rays and subsequently expanded them to a 768 length ray-bundle embedding via a linear layer. RenderFormer is trained end-to-end, first at a 256×256 resolution and with scenes containing at most 1.5k triangles followed by a second training stage where the output resolution is increased to 512×512 and the triangle count is increased to 4k. RenderFormer is trained with a weighted L_1 and LPIPS [44] loss on log-transformed reference renders.

4 Overview

Similar to RenderFormer, RenderFormer-V2 features a two-stage transformer-based neural rendering pipeline where the first stage resolves view-independent intra-primitive transport and the second stage transforms view-dependent ray-bundles to output tokens based on the transformed scene primitives from the first stage. However, RenderFormer-V2 deviates from RenderFormer is a number of critical steps: (i) RenderFormer-V2 is not limited to only triangle tokens and it supports a mixture of different scene primitives, including different types of

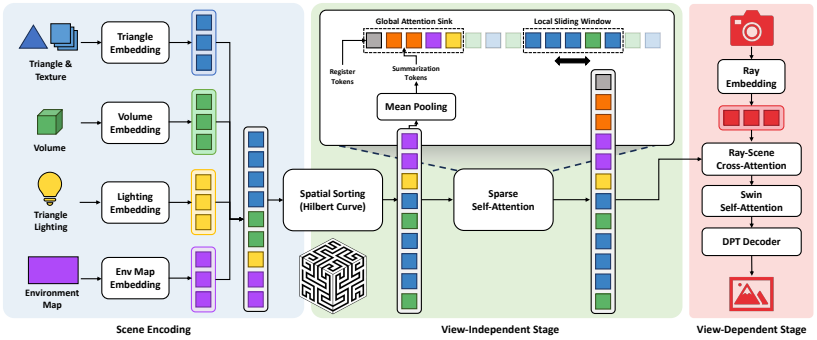


Fig. 2: RenderFormer-V2 Neural Rendering Pipeline.

light sources (Section 5), and (ii) RenderFormer-V2 scales better in terms of efficiency and accuracy to a larger number of scene primitives (Section 6). Figure 2 summarizes the RenderFormer-V2 pipeline.

5 Scene Embedding

We represent a virtual scene as a sequence of heterogeneous tokens that encode geometry, material, lighting, and camera information. In contrast to RenderFormer where the positional encoding is tailored to triangles as scene primitives, and which relies on a hard-coded camera-transformation to encode the camera position, we employ a uniform relative positional encoding strategy for all tokens (including ray-bundles):

1. For **tokens representing a concept with a 3D spatial location** (e.g., geometric primitive or camera) we use RoPE to encode the centroid of the concept with a RoPE dimension of 40 (i.e., 20 frequencies). RoPE ensures that the scene embedding is invariant to scene translations.
2. For **tokens representing positionless concepts** (e.g., environment map) we employ RoPE using the centroid of the whole scene to ensure that translating the scene does not affect the relative attention computation between tokens from both categories.

As the different concepts are defined by different parameters, we employ a separate embedding for each token type, and rely on the training process to enable RenderFormer-V2 to differentiate between the different primitive embeddings.

5.1 Triangle & Material Embedding

To embed a triangle, we first embed the different components (vertices, normals, and materials) and combine them via addition into the final token.

Vertex Embedding We stack the 3 positions of vertices (minus the centroid of the triangle) in a 9D vector, and apply (NeRF) positional encoding [20] with

Fig. 3: Latent material space visualization (tSNE).

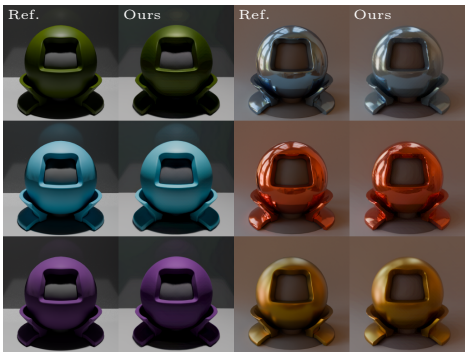


Fig. 4: The learned material appearance space is sufficiently expressive to encode measured BRDF outside the training set.

12 frequencies exponentially spaced between 2^0 and 2^{11}. Finally, we apply a (trainable) linear layer to expand to a token-length (i.e., 768) vector followed by RMS-normalization.

Normal Embedding We apply the same process as for vertex embedding to encode the per-vertex normals. Note, the vertex and normal embedding use separately trainable linear layers for expansion.

Material Embedding We desire an embedding of material appearance that is not tied to a particular BRDF model. Inspired by prior work on learning a latent embedding for BRDFs [9, 12, 15, 24, 26, 47], we also learn a material appearance embedding. A key advantage of RenderFormer-V2's transformer architecture is that it is powerful enough to directly learn how to evaluate the embedded material appearance (given the view and lighting) without the need to rely on a pretrained reverse mapping from latent code to material appearance. As we are interested in encoding the appearance rather than the exact BRDF, we follow an encoding inspired by Serrano et al.'s [24] perceptual material similarity metric and embed rendered images of a sphere under the Uffizi Gallery light probe. We opt for a sphere for it simplicity and the Uffizi Gallery light probe because it is color neural and it contains a good mix of low and high frequency lighting features [4]. Practically, we employ a CNN-based auto-encoder with a 9D latent feature vector at the bottleneck. We pretrain this encoder with an L1 loss on images rendered with Blender Cycles of randomly generated materials with the Principled BRDF model [5]. Furthermore, to encourage a coherent manifold, we apply a smoothness regularization term [8], and a tanh activation to constrain the values in the embedding to $[-1, +1]$. Figure 3 visualizes the learned latent material appearance space. While we currently use the Principled BRDF model for generating training data, this can easily be extended to include other analytical BRDF models or measured BRDFs. For efficiency, we also train an additional MLP for each analytical BRDF model (after the latent space is

trained) to map its parameters directly into the latent space to bypass the need to render a sphere; for measured BRDFs we render the material and use the pretrained encoder.

Texture Embedding To support spatially varying materials, we embed all materials in a texture. For each triangle, we first project the BRDF parameters into the learned latent space and subsequently rasterize the per-triangle texture (9 channels), local normal map (3 channels), and a displacement map (as a 1 channel height offset) in 32×32 image patches, which we subsequently encode with a pretrained VAE [33] into an 4×4 80-channel latent feature map. Finally, we compress the feature map via a single learnable linear layer to a 768-length vector and add it to the token embedding.

5.2 Voxel Embedding

To demonstrate RenderFormer-V2's ability to handle heterogenous geometric primitives, we also encode voxels filled with a scattering medium into a separate token-type.

Rotation and Scale Embedding For each voxel we encode the rotation matrix and scale vector that describes the voxel's relative rotation and per-axis scale in world coordinates. Similar to the normal encoding for triangles, we employ (NeRF) positional encoding with 12 frequencies, which is subsequently expanded via a linear layer to a token-length vector.

Scattering and Absorption As (RGB) scattering and (RGB) absorption coefficients are optical parameters of scattering media, and thus model independent, we opt to directly encode them. In addition, we also encode the anisotropy coefficient of the scattering function, yielding a 7D vector. To support spatially-varying scattering, we encode a $4 \times 4 \times 4$ volumetric texture of the 7D scattering feature vector, and linearly project the feature vector into a token-length vector that is added to the rotation and scale embedding.

5.3 Triangular Light Source Embedding

Similar to RenderFormer, we embed triangular light sources with homogeneous diffuse emittance. In contrast to RenderFormer, we store colored RGB emittance as an explicit light source token (instead of combining it with geometry tokens) which is expanded via a separate linear layer to the token-length. Similar to the triangle primitives, we add the vertex positions and per-vertex normals embedding to the token.

5.4 Environment Lighting Embedding

We follow an environment encoding similar to DiffusionRenderer [18].

Texture Embedding We store both an LDR (clamped to $[0, 1]$) and (log-encoded) HDR version (normalized by the log maximum value) of the environment map at

512×256 resolution, and encode each map using a pretrained VAE [33] yielding a $64 \times 32 \times 16$ latent feature map for each. This feature map is too large to store in a single token. Hence, we opt to split the latent feature map in 8×4 patches (of size $8 \times 8 \times 16$) that are compressed by a linear layer into two token-length vectors; hence yielding separate LDR and HDR environment map tokens.

Direction Embedding While a environment map token does not have a position, each pixel in the 64×64 pixel patch does correspond to a lighting direction. To make RenderFormer-V2 aware of the exact bundle of rays that correspond to the pixels in the patch per token, we create a direction map that, for each pixel, stores the corresponding (normalized) 3D direction vector in world coordinates. This direction map is projected via a linear layer to a token-length vector and added to the environment lighting embedding.

Lighting Strength Embedding During texture embedding we normalized the log-encoded HDR by the log maximum value. To retain this information, we expand this value to a token-length vector and add it to the final embedding.

5.5 Camera Embedding

Similar to RenderFormer, we embed the (normalized) ray directions in an 8×8 map, and project it to a token-length vector using a single linear layer. Whereas in RenderFormer the ray directions are expressed in camera coordinates, we encode them directly in world coordinates; the origin of the ray bundle is already taken care of via the uniform relative positional encoding strategy.

6 RenderFormer-V2 Architecture

While RenderFormer-V2 follows RenderFormer's two stage design, each stage differs in how attention is computed. We first discuss the modification to the second stage (i.e., view-dependent), follow by the more significant modifications to the first stage (i.e., view-independent).

6.1 View-dependent Stage

Unlike RenderFormer's view-dependent stage, RenderFormer-V2's view-dependent stage operates in world coordinates (rather than camera coordinates). Moreover, we replace the full self-attention layers by SWIN windowed attention layers [19] with a window size of 8 and a shift of 4. While the computation cost changes modestly from $\mathcal{O}(T \times R + R^2)$ to $\mathcal{O}(T \times R + W^2)$ (T is the number of scene tokens (the dominating factor), R the number of ray-bundles, and W the window size), its main advantage is the ability to scale to larger resolutions more easily because the context window size is now resolution independent.

6.2 View-independent Stage

The view-independent stage is the main bottle neck when increasing the number of tokens as computation cost scales quadratically with the number of primitives. Moreover, when the sequence grows large, naive attention tends to loose focus, resulting in a loss of render fidelity. While often the dominating factor, primitive-to-primitive light transport is not solely a local phenomena; a subset of global factors such as illumination and large-scale occlusion can significantly affect light transport through the scene. Hence, we combine attention from local primitives with attention from global tokens, while mitigating focus-loss on large sequences.

Sorting & Serialization Unlike ray-bundles, scene primitives are not regularly spaced, making it more challenging to exploit locality while retaining efficient GPU-computation. Inspired by Point Transformer v3 [34], we sort and linearize geometry tokens using Hilbert curves such that nearby (in the 1D sequence) tokens are likely close in 3D space too. This allows us to model local attention with sliding window attention [3]; we take 256 tokens before and after the target token, yielding a computational complexity independent of the number of primitives.

Attention Sinks for Rendering To model global light transport, we adapt attention sinks [35]. The tokens in the attention sink are always included in the attention calculations. We strategically assign three different token types with rendering relevant semantics to the sink:

1. *Global Register Tokens*: we add 16 register tokens [6] to the input sequence for storing storing global information.
2. *Light Source Tokens*: it is likely that the light transport on most geometric primitives is directly affected by the light sources in the scene. Hence, by placing the light sources in the sink we ensure that each light source is taken in account for each primitive regardless of distance. Conceptually, the attention computation with respect to the light sources is analogous to importance sampling the light sources in path tracing.
3. *Summarization Tokens*: while long range light transport is important, we argue that the precise details of distant geometry matter less. Therefore, we model long-range interactions with a coarser geometry representation. Specifically, we perform mean pooling over every 64 consecutive geometry tokens to form a summarization token that we add to the attention sink.

7 Training

Training Data Similarly to RenderFormer, we generate scenes by placing 1 to 3 randomly selected objects from the ObjaVerse dataset in one of four randomly selected template scenes (a ground plane with one, two or three walls). Different from RenderFormer's scene generation process, we assign randomly generated materials (using the Principled BRDF model [5] mapped into our material appearance latent) from the following five categories: (a) homogeneous opaque

diffuse+specular material (2/9 chance; with the sum of the diffuse and specular albedo restricted to $[0.9, 1]$, and roughness log-sampled in $[0.01, 1]$), (b) a homogeneous opaque material with metallic+roughness parameters (2/9), (c) a diffuse+specular SVBRDF randomly sampled from the MatSynth SVBRDF dataset [30] (1/9), (d) a metallic+roughness SVBRDF from MatSynth (1/9), or (e) a homogeneous transparent metallic+roughness material (3/9; with metallic sampled in $[0, 1]$, roughness log-sampled in $[0.01, 1]$, and each RGB channel of the base color sampled in $[0, 1]$). In 2/3 of the scenes, we add one (1/2) or two (1/2) volumes with randomly chosen scattering and absorption. The camera is randomly placed and aimed at the scene with a FOV chosen between 30° and 60°. Unlike RenderFormer, we allow the camera to be placed inside the scene. Up to 8 triangular light sources are randomly placed outside the scene with a random (RGB) intensity between 2,500 and 5,000 W/m^2. Furthermore, for 5/6 of the scenes, we add environment lighting randomly selected from PolyHaven. To avoid color bias, we randomly swap the color channels in the environment map. Each scene is rendered offline using Blender Cycles with 4,096 samples per pixel. For efficiency, we pre-generate a training dataset of $\sim$10M randomly sampled scenes spanning resolutions 256^2–2048^2 with 1k–64k primitives, totaling $\sim$70TB.

Loss Function We employ the same loss as RenderFormer:

$$L_1(\log I) + 0.05\ L_{LPIPS}(clamp(\log I / \log 2, 0, 1)), \tag{1}$$

where the log encoding of the image I serves to avoid specular reflections dominating the L1 error, and the LPIPS loss [44] minimizes perceptual differences.

Training Process & Refinement We employ a five-stage training regime to first focus on learning the principles of light transport, before adding scene-complexity:

- In stage 1, we decimate the generated scenes to 1k primitives and render the scene at 256×256 resolution. To prioritize learning accurate coarse-scale transport, we employ full-attention at this stage. Furthermore, we gradually increase the complexity of the scene by first training exclusively on homogeneous materials (1 day on 32× A100 GPUs). Next we include SVBRDFs (1 additional day), followed by the inclusion of environment lighting (1 day; to focus training on the lighting effects, we mask out pixels that directly see the environment map), and finally adding volumetric objects (1 day).
- In stage 2 we increase the primitive budget to 4k as well as the resolution to 512×512, and continue to train for 2 days on the same setup.
- In stage 3, we keep the scene parameters the same, but switch from full-attention to our combined attention sink and windowed attention (3 days).
- In stage 4, we increase the primitive budget to 16k (1 week).
- Finally, in stage 5 we increase the primitive budget to 64k as well as the resolution to 2048×2048 for another 3 days of training.

Yielding a total training time of 19 days on 32× A100 GPUs. While the training cost is significant, we emphasize that once pretrained, no fine-tuning or training is needed for rendering a new scene.

Fig. 5: Additional results demonstrating RenderFormer-V2's ability to handle displacement mapping and volumetric scattering without the need for specialized code, and two complex scenes with a large number of primitives.

8 Results

Capabilities Figures 1 and 5 demonstrate the capabilities of RenderFormer-V2 on a wide variety of scenes. Figure 1 demonstrates that RenderFormer-V2 can render, with full global light transport, scenes containing refractive surfaces (1st) including caustics, environment lighting (2nd), volumetric scattering (3rd), and textures (4th). Figure 5 demonstrates displacement mapping (1st), volumetric scattering (2nd), and complex scenes with a large number of primitives (3rd and 4th). Prior neural rendering methods either require per-scene fine-tuning and/or cannot support all of these effects. Compared to path tracing, RenderFormer-V2 does not require specialized code to handle displacement mapping or volumetric scattering, and it can learn such effects solely by example.

RenderFormer-V2 employs a flexible material appearance latent space for assigning material properties to surface in the scene. Figure 4 demonstrates that, even though our latent space is trained on materials modeled with the Disney principled BRDF model [5], it can also model materials not part of the training set (e.g., in this case selected measured materials from the RGL BRDF dataset [7]). Moreover, as our latent space is based on encoding rendered images, it can easily be retrained to encompass material appearances currently not covered (e.g., anisotropic and color changing materials).

Comparison to RenderFormer RenderFormer-V2 is closely related to RenderFormer [41]; both use a similar two-stage transformer-based pipeline. Compared to RenderFormer, RenderFormer-V2 achieves better accuracy when rendering scenes with a large number of triangles thanks to its sparse attention mechanism (Figure 6). Figure 9 qualitatively compares render quality for a scene with varying number of triangles; the darkening at 83.2k triangles when rendered with RenderFormer is a direct consequence of loss of attention. We used the publicly available version of RenderFormer which is trained on $4k$ triangles; we found that training RenderFormer for larger triangle meshes is unstable. Moreover, RenderFormer-V2 is also considerably more efficient as shown in Figure 7 and Figure 8 thanks to the render-informed sparse attention in the view-independent stage and the SWIN-attention in the view-dependent stage re-

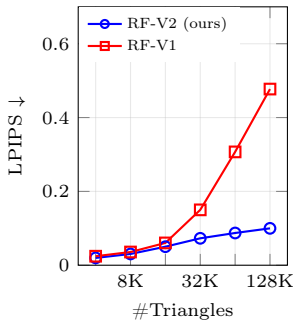


Fig. 6: Render quality degradation for increasing primitive budget.

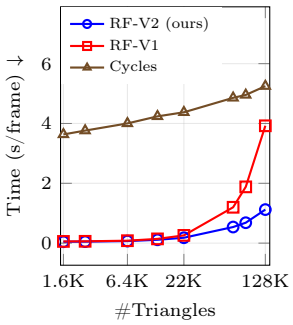


Fig. 7: Runtime scaling versus mesh complexity.

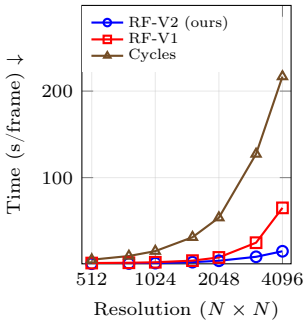


Fig. 8: Resolution scaling on a 64k-triangle scene.

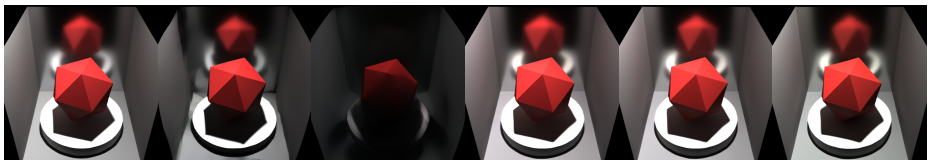


Fig. 9: Qualitative comparison of render quality of RenderFormer vs. RenderFormer-V2 for an increasing number of primitives. At 83.2K primitives, RenderFormer loses attentional focus, resulting in a darkened image.

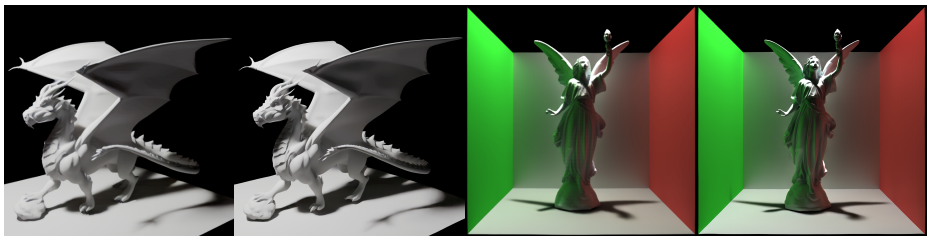


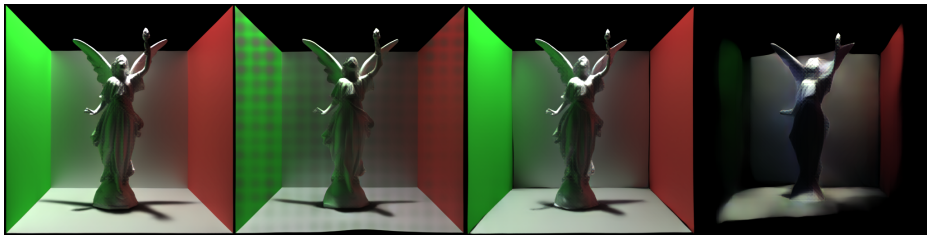
Fig. 10: Rendering at higher resolutions also improves geometric details even for features larger than a pixel. *E.g.*, the claws on the dragon and Lucy's face and hands are better resolved at 2048 (right) than at 512 (left) resolution.

spectively (all timings are measured on a single NVIDIA A100). Empirically, we found that rendering time is approximately equally distributed over both stages (i.e., view-dependent vs. view-independent). For reference, we also include timings of Blender Cycles on the same scenes using 4096 adaptive samples per pixel (i.e., the same setting as used for the training images).

RenderFormer-V2 not only supports larger triangle meshes, it can also be more easily fine-tuned for higher image resolution. Figure 10 compares two scenes rendered at 512×512 and 2048×2048. An interesting observation is that despite

Table 1: Sparse attention ablation with or without sliding windowed attention (SW) and attention sink (AS) with inclusion of light source tokens (L) and summary tokens (S), as well as varying number of summarization ratios. For each metric, the best, second best, and third best results are highlighted.

Model Variant	PSNR ↑	SSIM ↑	LPIPS ↓	HDR-FLIP ↓
RenderFormer-V2	28.25	0.8982	0.0997	0.4200
w/o AS	27.36	0.8841	0.1247	0.4359
w/o SW	26.92	0.8715	0.1289	0.4517
AS w/o L	27.40	0.8813	0.1081	0.4250
AS w/o S	27.09	0.8754	0.1210	0.4485
AS w/o L, S	26.13	0.8474	0.1514	0.4959
More S (#seq / 32)	27.78	0.8923	0.1072	0.4231
Less S (#seq / 128)	26.48	0.8690	0.1346	0.4655
Less S (#seq / 256)	26.93	0.8663	0.1316	0.4580
Full Attention	27.91	0.8953	0.1027	0.4287



All components w/o AS w/o SW AS w/o L, S

Fig. 11: A qualitative comparison of the ablation variants indicates that all components are essential to achieve the highest render quality.

both scenes containing the same number of triangles, that at higher resolution RenderFormer-V2 is able to more faithfully render fine detailed geometry (e.g., the dragon's claws and Lucy's face and hand).

Ablation We ablate the different components of RenderFormer-V2's sparse attention mechanism. Table 1 shows average PSNR, SSIM [32], LPIPS [44], and HDR-FLIP [2] over randomly generated scenes (using the same distribution as the training data, but with a different set of environment maps, SVBRDFs, and shapes than used for training) for different ablation variants: without sliding windowed attention, without attention sink, and leaving out lighting and/or summarization token from the attention sink – each ablation variant is trained up to stage 4 (16k primitives). Only when all components are included, we achieve the highest accuracy in rendering. Surprisingly, our sparse attention outperforms full-attention, which we attribute to the full-attention model having to distribute its attention over too many tokens (i.e., loss of focus). Figure 11 further qualitatively demonstrates the difference in quality between the different ablation

variants. We refer to the supplemental material for additional results and comparisons.

Limitations While RenderFormer-V2 addresses many shortcomings of RenderFormer, it is not without limitations. First, RenderFormer-V2 is still limited to maximum 8 light sources per scene, a constraint inherited from its training data. However, we argue that more complex lighting conditions are more efficiently modeled with the addition of an environment map. Furthermore, similar to RenderFormer, our model is trained on single frames, and it does not explicitely enforce temporal coherence. While RenderFormer-V2 supports textures/SVBRDFs, the resolution is fixed per triangle primitive (32×32). Consequently, significant texture-quality degradation can occur for large triangles. However, this can easily be resolved by subdividing triangles to impose a maximum triangle size. By supporting heterogeneous primitives and using a material appearance parameterization independent of a hard-coded BRDF model, RenderFormer-V2 can easily be extended to support new primitives. However, training is most effective when new primitives are added in the first training stage. Consequently, adding a new primitive typically requires significant retraining. Improving extensibility without requiring a full retraining is an interesting avenue for future research.

9 Conclusion

In this paper we presented RenderFormer-V2, a transformer-based neural rendering model that takes as input a sequence of primitives (i.e., textured triangles, volumetric elements, light sources, environment maps, and camera) and outputs a rendered image of the scene with full global illumination. RenderFormer-V2 employs a rendering-aware sparse-attention to resolve intra-primitive light transport which enables RenderFormer-V2 to scale better to larger scenes, both for training as well as inference. Furthermore, we employ a flexible latent material appearance space to specify surface reflectance. RenderFormer-V2 is trained end-to-end, thereby avoiding the need for specialized code to handle displacement mapping or volumetric scattering.

Acknowledgment

Chong Zeng was supported by the Stanford Graduate Fellowship. This work was partially supported by the Brown Institute for Media Innovation at Stanford University.

References

1. Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebron, F., Sanghai, S.: GQA: Training generalized multi-query transformer models from multi-head checkpoints. In: EMNPL. pp. 4895–4901 (Dec 2023). https://doi.org/10.18653/v1/2023.emnlp-main.298, https://aclanthology.org/2023.emnlp-main.298/

2. Andersson, P., Nilsson, J., Akenine-Möller, T., Oskarsson, M., Åström, K., Fairchild, M.D.: FLIP: A Difference Evaluator for Alternating Images. Proceedings of the ACM on Computer Graphics and Interactive Techniques **3**(2), 15:1–15:23 (2020)
3. Beltagy, I., Peters, M.E., Cohan, A.: Longformer: The long-document transformer (2020), https://arxiv.org/abs/2004.05150
4. Bieron, J., Peers, P.: An adaptive brdf fitting metric. Computer Graphics Forum **39**(4) (July 2020). https://doi.org/http://doi.org/10.1111/cgf.14054
5. Burley, B.: Physically Based Shading at Disney. In: SIGGRAPH 2012 Course Notes. ACM (2012)
6. Darcet, T., Oquab, M., Mairal, J., Bojanowski, P.: Vision transformers need registers. In: International Conference on Learning Representations (ICLR) (2024)
7. Dupuy, J., Jakob, W.: An adaptive parameterization for efficient material acquisition and rendering. Transactions on Graphics (Proceedings of SIGGRAPH Asia) **37**(6), 274:1–274:18 (Nov 2018). https://doi.org/10.1145/3272127.3275059
8. Gao, D., Li, X., Dong, Y., Peers, P., Xu, K., Tong, X.: Deep inverse rendering for high-resolution svbrdf estimation from an arbitrary number of images. ACM Trans. Graph. **38**(4) (Jul 2019). https://doi.org/10.1145/3306346.3323042, https://doi.org/10.1145/3306346.3323042
9. Gokbudak, F., Sztrajman, A., Zhou, C., Zhong, F., Mantiuk, R., Oztireli, C.: Hypernetworks for generalizable brdf representation. In: ECCV. p. 73–89 (2024). https://doi.org/10.1007/978-3-031-73116-7_5, https://doi.org/10.1007/978-3-031-73116-7_5
10. Granskog, J., Rousselle, F., Papas, M., Novák, J.: Compositional neural scene representations for shading inference. ACM Trans. Graph. **39**(4) (2020)
11. Granskog, J., Schnabel, T.N., Rousselle, F., Novák, J.: Neural scene graph rendering. ACM Trans. Graph. **40**(4) (2021)
12. Guo, J., Li, Z., He, X., Wang, B., Li, W., Guo, Y., Yan, L.Q.: Metalayer: A meta-learned bsdf model for layered materials. ACM Trans. Graph. **42**(6) (Dec 2023). https://doi.org/10.1145/3618365, https://doi.org/10.1145/3618365
13. Haque, A., Tancik, M., Efros, A.A., Holynski, A., Kanazawa, A.: Instruct-nerf2nerf: Editing 3d scenes with instructions. In: CVPR. pp. 19740–19750 (2023)
14. Ho, J., Kalchbrenner, N., Weissenborn, D., Salimans, T.: Axial attention in multidimensional transformers. arXiv preprint arXiv:1912.12180 (2019)
15. Hu, B., Guo, J., Chen, Y., Li, M., Guo, Y.: Deepbrdf: A deep representation for manipulating measured brdf. Computer Graphics Forum **39**(2), 157–166 (2020). https://doi.org/https://doi.org/10.1111/cgf.13920, https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13920
16. Kang, S., Kim, J., Kim, J., Hwang, S.J.: See what you are told: Visual attention sink in large multimodal models. In: ICLR (2025)
17. Li, X., Li, M., Cai, T., Xi, H., Yang, S., Lin, Y., Zhang, L., Yang, S., Hu, J., Peng, K., Agrawala, M., Stoica, I., Keutzer, K., Han, S.: Radial attention: $\mathcal{O}(n \log n)$ sparse attention with energy decay for long video generation. arXiv preprint arXiv:2506.19852 (2025)
18. Liang, R., Gojcic, Z., Ling, H., Munkberg, J., Hasselgren, J., Lin, C.H., Gao, J., Keller, A., Vijaykumar, N., Fidler, S., et al.: Diffusion renderer: Neural inverse and forward rendering with video diffusion models. In: CVPR. pp. 26069–26080 (2025)
19. Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., Guo, B.: Swin transformer: Hierarchical vision transformer using shifted windows. In: ICCV. pp. 9992–10002 (2021). https://doi.org/10.1109/ICCV48922.2021.00986

20. Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R., Ng, R.: Nerf: representing scenes as neural radiance fields for view synthesis. Commun. ACM **65**(1), 99–106 (Dec 2021). https://doi.org/10.1145/3503250
21. Munkhdalai, T., Faruqui, M., Gopal, S.: Leave no context behind: Efficient infinite context transformers with infini-attention (2024), https://arxiv.org/abs/2404.07143
22. Nalbach, O., Arabadzhiyska, E., Mehta, D., Seidel, H.P., Ritschel, T.: Deep shading: Convolutional neural networks for screen space shading. Comput. Graph. Forum **36**(4), 65–78 (Jul 2017). https://doi.org/10.1111/cgf.13225, https://doi.org/10.1111/cgf.13225
23. Parmar, N., Vaswani, A., Uszkoreit, J., Kaiser, L., Shazeer, N., Ku, A., Tran, D.: Image Transformer. In: ICML. pp. 4052–4061 (2018)
24. Serrano, A., Chen, B., Wang, C., Piovarči, M., Seidel, H.P., Didyk, P., Myszkowski, K.: The effect of shape and illumination on material perception: model and applications. ACM Transactions on Graphics (TOG) **40**(4), 1–16 (2021)
25. Su, J., Lu, Y., Pan, S., Murtadha, A., Wen, B., Liu, Y.: Roformer: Enhanced transformer with rotary position embedding. Neurocomputing **568**, 127063 (2024)
26. Sztrajman, A., Rainer, G., Ritschel, T., Weyrich, T.: Neural brdf representation and importance sampling. In: Computer Graphics Forum. vol. 40, pp. 332–346. Wiley Online Library (2021)
27. Tay, Y., Dehghani, M., Bahri, D., Metzler, D.: Efficient transformers: A survey. ACM Comput. Surv. **55**(6) (Dec 2022). https://doi.org/10.1145/3530811, https://doi.org/10.1145/3530811
28. Tewari, A., Thies, J., Mildenhall, B., Srinivasan, P., Tretschk, E., Yifan, W., Lassner, C., Sitzmann, V., Martin-Brualla, R., Lombardi, S., Simon, T., Theobalt, C., Niessner, M., Barron, J.T., Wetzstein, G., Zollhöfer, M., Golyanik, V.: Advances in neural rendering. Comp. Graph. Forum **41**(2), 703–735 (2022)
29. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. In: NeurIPS. p. 6000–6010 (2017)
30. Vecchio, G., Deschaintre, V.: Matsynth: A modern pbr materials dataset. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 22109–22118 (June 2024)
31. Walter, B., Marschner, S.R., Li, H., Torrance, K.E.: Microfacet models for refraction through rough surfaces. In: Proceedings of the 18th Eurographics Conference on Rendering Techniques. p. 195–206. EGSR'07 (2007)
32. Wang, Z., Bovik, A.C., Sheikh, H.R., Simoncelli, E.P.: Image quality assessment: from error visibility to structural similarity. IEEE transactions on image processing **13**(4), 600–612 (2004)
33. Wu, C., Li, J., Zhou, J., Lin, J., Gao, K., Yan, K., ming Yin, S., Bai, S., Xu, X., Chen, Y., Chen, Y., Tang, Z., Zhang, Z., Wang, Z., Yang, A., Yu, B., Cheng, C., Liu, D., Li, D., Zhang, H., Meng, H., Wei, H., Ni, J., Chen, K., Cao, K., Peng, L., Qu, L., Wu, M., Wang, P., Yu, S., Wen, T., Feng, W., Xu, X., Wang, Y., Zhang, Y., Zhu, Y., Wu, Y., Cai, Y., Liu, Z.: Qwen-image technical report (2025), https://arxiv.org/abs/2508.02324
34. Wu, X., Jiang, L., Wang, P.S., Liu, Z., Liu, X., Qiao, Y., Ouyang, W., He, T., Zhao, H.: Point transformer v3: Simpler, faster, stronger. In: CVPR (2024)
35. Xiao, G., Tian, Y., Chen, B., Han, S., Lewis, M.: Efficient streaming language models with attention sinks (2024), https://arxiv.org/abs/2309.17453
36. Xu, B., Wang, C., Li, T., Wu, L., Wronski, B., Ramamoorthi, R., Salvi, M., et al.: A generalizable light transport 3d embedding for global illumination. arXiv preprint arXiv:2510.18189 (2025)

37. Yang, J., Li, C., Zhang, P., Dai, X., Xiao, B., Yuan, L., Gao, J.: Focal attention for long-range interactions in vision transformers. In: Neurips (2021)
38. Yang, Y.Q., Guo, Y.X., Xiong, J.Y., Liu, Y., Pan, H., Wang, P.S., Tong, X., Guo, B.: Swin3d: A pretrained transformer backbone for 3d indoor scene understanding (2023)
39. Yuan, J., Gao, H., Dai, D., Luo, J., Zhao, L., Zhang, Z., Xie, Z., Wei, Y., Wang, L., Xiao, Z., Wang, Y., Ruan, C., Zhang, M., Liang, W., Zeng, W.: Native sparse attention: Hardware-aligned and natively trainable sparse attention. In: ACL (Jul 2025). https://doi.org/10.18653/v1/2025.acl-long.1126
40. Yuan, Y.J., Sun, Y.T., Lai, Y.K., Ma, Y., Jia, R., Gao, L.: Nerf-editing: geometry editing of neural radiance fields. In: CVPR. pp. 18353–18364 (2022)
41. Zeng, C., Dong, Y., Peers, P., Wu, H., Tong, X.: Renderformer: Transformer-based neural rendering of triangle meshes with global illumination. In: Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers. SIGGRAPH Conference Papers '25 (2025). https://doi.org/10.1145/3721238.3730595, https://doi.org/10.1145/3721238.3730595
42. Zeng, Z., Deschaintre, V., Georgiev, I., Hold-Geoffroy, Y., Hu, Y., Luan, F., Yan, L.Q., Hašan, M.: Rgb↔x: Image decomposition and synthesis using material- and lighting-aware diffusion models. In: ACM SIGGRAPH 2024 Conference Papers. SIGGRAPH '24 (2024). https://doi.org/10.1145/3641519.3657445, https://doi.org/10.1145/3641519.3657445
43. Zhang, P., Dai, X., Yang, J., Xiao, B., Yuan, L., Zhang, L., Gao, J.: Multi-scale vision longformer: A new vision transformer for high-resolution image encoding. In: ICCV. pp. 2978–2988 (2021). https://doi.org/10.1109/ICCV48922.2021.00299
44. Zhang, R., Isola, P., Efros, A.A., Shechtman, E., Wang, O.: The unreasonable effectiveness of deep features as a perceptual metric. In: CVPR (2018)
45. Zhang, Z., Sheng, Y., Zhou, T., Chen, T., Zheng, L., Cai, R., Song, Z., Tian, Y., Re, C., Barrett, C., Wang, Z., Chen, B.: H2o: Heavy-hitter oracle for efficient generative inference of large language models. In: Neurips (2023), https://openreview.net/forum?id=RkRrPp7GKO
46. Zheng, C., Huo, Y., Huang, H., Sheng, H., Huang, J., Tang, R., Zhu, H., Wang, R., Bao, H.: Neural global illumination via superposed deformable feature fields. In: SIGGRAPH Asia 2024 Conference Papers. pp. 1–11 (2024)
47. Zheng, C., Zheng, R., Wang, R., Zhao, S., Bao, H.: A compact representation of measured brdfs using neural processes. ACM Transactions on Graphics (TOG) **41**(2), 1–15 (2021)
48. Zhu, Z., Soricut, R.: H-transformer-1d: Fast one-dimensional hierarchical attention for sequences (2021)