RenderGAN: Enhancing Real-time Rendering Efficiency with Deep Learning

MARCO MAMELI, Dipartimento di Ingegneria dell' Informazione (DII), Università Politecnica delle Marche, Ancona, Italy
MARINA PAOLANTI, University of Macerata, Macerata, Italy
ADRIANO MANCINI and PRIMO ZINGARETTI, Università Politecnica delle Marche, Ancona, Italy
ROBERTO PIERDICCA, Università Politecnica delle Marche, Dipartimento di Ingegneria Civile, Edile e dell' Architettura (DICEA), Ancona, Italy

In the domain of computer graphics, achieving high visual quality in real-time rendering remains a formidable challenge due to the inherent time-quality tradeoff. Conventional real-time rendering engines sacrifice visual fidelity for interactive performance, while image generation using path-tracing techniques can be exceedingly time-consuming. In this article, we introduce RenderGAN, a deep learning-based solution designed to address this critical challenge in real-time rendering. RenderGAN uses G-Buffers and information from a real-time rendering engine as inputs to produce output images with exceptional visual fidelity. Its encoder–decoder architecture, trained using the Generative Adversarial Network (GAN) framework with perceptual loss, enhances image realism. To evaluate RenderGAN's effectiveness, we quantitatively compare the generated images with those of a path-tracing engine, obtaining a remarkable Universal Image Quality Index (UIQI) value of 0.898. RenderGAN's open source nature fosters collaboration, driving advancements in real-time computer graphics and rendering techniques. By bridging the gap between real-time and path-tracing rendering, RenderGAN opens new horizons for accelerated image generation, inspiring innovation and unlocking the full potential of real-time visual experiences. Project page: https://github.com/marcomameli1992/RenderNet

CCS Concepts: • **Networks** → **Layering**;

Additional Key Words and Phrases: Computer Graphics, Deep Learning, Generative Adversarial Networks, RenderGAN

ACM Reference format:
Marco Mameli, Marina Paolanti, Adriano Mancini, Primo Zingaretti, and Roberto Pierdicca. 2025. RenderGAN: Enhancing Real-time Rendering Efficiency with Deep Learning. *ACM Trans. Multimedia Comput. Commun. Appl.* 21, 3, Article 99 (March 2025), 22 pages.
https://doi.org/10.1145/3712263

Authors' Contact Information: Marco Mameli (corresponding author), Dipartimento di Ingegneria dell' Informazione (DII), Università Politecnica delle Marche, Ancona, Italy; e-mail: m.mameli@pm.univpm.it; Marina Paolanti, University of Macerata, Macerata, Italy; e-mail: marina.paolanti@unimc.it; Adriano Mancini, Università Politecnica delle Marche, Ancona, Italy; e-mail: a.mancini@univpm.it; Primo Zingaretti, Università Politecnica delle Marche, Ancona, Italy; e-mail: p.zingaretti@staff.univpm.it; Roberto Pierdicca, Università Politecnica delle Marche, Dipartimento di Ingegneria Civile, Edile e dell' Architettura (DICEA), Ancona, Italy; e-mail: r.pierdicca@staff.univpm.it.

ACM 1551-6865/2025/3-ART99
https://doi.org/10.1145/3712263

1 Introduction

Computer Graphics (CG) has revolutionized many fields, enabling the creation of visually impressive digital content [1]. In this dynamic and constantly evolving field, achieving real-time rendering with uncompromising visual quality remains an elusive challenge. The demand for high-fidelity images in real-time applications, such as video games, simulations, and virtual reality experiences, has led to a pressing need for innovative solutions that address the time quality tradeoff. Achieving high visual quality in CG remains a complex task, particularly when time is of the essence [2]. One critical aspect of CG lies in rendering a 3D scene into an image, a process that demands substantial computing resources and time [3]. The rendering operation plays a pivotal role within the simulation context, and it encompasses two main techniques—real-time rendering and path-tracing rendering. Real-time rendering offers rapid image generation but compromises on the intricate details that contribute to visual realism. In contrast, path-tracing rendering produces images of exceptional quality, considering shadows, lighting, and material intricacies, but at the cost of substantial time [4]. Bridging the gap between real-time and path-tracing rendering is a longstanding challenge that has profound implications across various domains [5, 6].

In response to these challenges, **Deep Learning (DL)** has emerged as a transformative force in the domain of real-time rendering, revolutionizing the way we approach the age-old challenge of achieving high visual quality with limited computational resources [7, 8]. Traditionally, real-time rendering has grappled with the tradeoff between speed and visual fidelity, often sacrificing the latter for interactive performance. However, the advent of DL techniques, such as **Generative Adversarial Networks (GANs)** [9] and **Convolutional Neural Networks (CNNs)**, has opened up new possibilities for accelerating image generation without compromising on realism [10]. By training encoder–decoder architectures on huge datasets of rendered images, DL models can now approximate the intricate details of scenes and produce visually compelling results in real time [11]. From video games to virtual reality experiences, the integration of DL in real-time rendering has not only elevated visual quality but also unlocked innovative approaches to bridging the gap between real-time and offline rendering techniques. By harnessing the power of convolutional networks and leveraging images generated by real-time engines, it becomes possible to generate output images that closely resemble those produced by path-tracing engines. This convergence of DL and real-time rendering has the potential to revolutionize the field, opening up new possibilities for accelerated image generation in CG [11].

Considering this context, this article introduces RenderGAN, a groundbreaking DL-based solution tailored to tackle the time–quality tradeoff in real-time CG. By leveraging G-Buffers and information from a real-time rendering engine, RenderGAN generates output images with extraordinary visual fidelity. Its encoder–decoder architecture, trained using the GAN framework with perceptual loss, enhances the realism of the generated images. We prove RenderGAN's effectiveness through a rigorous quantitative comparison with images produced by a path-tracing engine. This evaluation showcases RenderGAN's impressive performance, achieving a remarkable **Universal Image Quality Index (UIQI)** value of 0.898. Additionally, visualizations demonstrate the striking resemblance between RenderGAN's rendered images and those produced by path-tracing engines. What sets RenderGAN apart is its commitment to openness, encouraging collaboration, innovation, and accessibility within the CG community. Unlike proprietary alternatives, RenderGAN fosters collective efforts to advance real-time CG and rendering techniques, benefiting diverse applications and industries. By seamlessly merging DL and real-time rendering, RenderGAN opens up new avenues for accelerated image generation, bridging the gap between real-time and path-tracing rendering. The remarkable reduction in rendering time, without compromising visual quality, promises to revolutionize real-time CG, propelling innovation in this dynamic and ever-evolving

field. RenderGAN's open source nature empowers researchers and practitioners to explore the forefront of accelerated image generation, driving the future of real-time rendering and fostering a collaborative, forward-thinking community.

The main contributions of this article are as follows: (i) the design of RenderGAN, a novel DL-based solution that addresses the time–quality tradeoff in real-time CG. RenderGAN uses G-Buffers and information from a real-time rendering engine as inputs to generate output images with remarkable visual fidelity; (ii) presenting an innovative encoder–decoder architecture for image generation, trained using the GAN framework with perceptual loss, it enhances the realism of the generated images, achieving high-quality outputs; (iii) distinguishing itself as an open source solution, RenderGAN fosters collaboration, innovation, and accessibility within the CG community. Unlike proprietary alternatives, RenderGAN promotes collective efforts to advance real-time CG and rendering techniques.

The article is organized as follows: in Section 2, we embark on a comprehensive exploration of the state-of-the-art techniques meticulously analyzing the advancements in CG and rendering methods. Following this analysis, in Section 3, we introduce our innovative solution, RenderGAN, designed to tackle the time–quality tradeoff in real-time CG. We present the methodology employed and the details of our encoder–decoder architecture for image generation using DL. The remarkable result achieved by RenderGAN is thoroughly discussed and quantitatively evaluated in Section 4, highlighting the compelling UIQI value attained. Additionally, visualizations of the rendered images produced by RenderGAN are provided, visually showcasing its effectiveness. Finally, in Section 5, we conclude the article by summarizing the contributions and impact of our work and outline future research directions to further advance the field of explainability in 3D point cloud analysis.

2 Related Works

In this section, we conduct a comprehensive review of the state-of-the-art techniques in two key areas—real-time rendering and image and video superresolution. We delve into the advancements and innovations within these domains, exploring how they have tackled the challenges of achieving high visual quality and real-time performance in CG and image processing. Through this review, we aim to shed light on the latest methodologies and cutting-edge approaches that have significantly contributed to pushing the boundaries of real-time rendering and elevating the visual fidelity of images and videos through superresolution techniques.

2.1 Real-time Rendering

One of the crucial challenges in real-time rendering is undersampling, where each rendered sample represents a single point in the scene. However, to achieve an antialiased image, it is necessary to compute multiple samples within a sensor's pixel area. This discrepancy becomes particularly problematic for highly detailed surfaces, as undersampling can result in jarring visual effects such as shimmering and flickering when surface features are reduced to subpixel size. To address this issue, various methods have been developed to mitigate the different sources of aliasing in undersampled images. We can categorize these methods into two groups—spatial-only antialiasing techniques, which utilize information from a single rendered image, and **Temporal Antialiasing (TAA)** methods, which leverage temporal history from multiple rendered frames. By employing these approaches, real-time rendering can overcome undersampling challenges and achieve improved visual quality in dynamic scenes.

2.1.1 Spatial Antialiasing. In real-time graphics, **Multisampling Antialiasing (MSAA)** is a classic technique that falls under the category of supersampling. With MSAA, the color of a

polygon covered by a pixel is calculated only once, avoiding the need to compute multiple subpixel samples for the same polygon [12]. Another well-established method is texture filtering, wherein high-frequency details from surface textures are prefiltered using image pyramids [13]. A suitable prefiltered region is selected during runtime based on the pixel's footprint projected onto the textured surface. Specialized methods can also address aliasing from other shading components, such as specular highlights or shadows [14, 15].

Most spatial antialiasing methods rely on image enhancement techniques from image processing. The fundamental idea is to identify image discontinuities and cleverly blur them to smooth out jagged edges. **Morphological Antialiasing (MLAA)** attempts to estimate the pixel coverage of the original geometry by analyzing color discontinuities in the vicinity of pixels in the final image [16]. Fast approximate antialiasing tackles the undersampling problem by attenuating subpixel features, thereby enhancing perceived temporal stability. Subpixel morphological antialiasing combines MLAA with MSAA to further improve the image quality. While these methods can produce satisfactory result for static images, the temporal variation between frames antialiased with these techniques may still exhibit visible flicker and other false pixel motion. As a result, real-time rendering faces ongoing challenges in achieving both spatial antialiasing and TAA with superior visual fidelity [17].

2.1.2 Temporal Antialiasing and Reconstruction. These techniques used temporal history, often stored in a temporally accumulated buffer, coupled with various forms of temporal rejection filtering. The primary distinction lies in the approach of high-quality temporal methods, such as amortized supersampling [18], which involve reprojecting history from one frame to another to compensate for motion, similar to motion compensation in video compression. TAA [19], on the other hand, used an edge detection filter to suppress flickering through heavier temporal accumulation. Recently, TAA has also been employed for temporal upsampling [20].

Deep-Learned Supersampling (DLSS) used temporal history and neural networks to enhance edges and perform upscaling [21]. However, details, quality, and performance information for DLSS are not readily available in the public domain which provides comprehensive details and evaluations, accessible without the need for proprietary hardware or software.

Furthermore, a recent trend in reducing rendering costs involves applying reconstruction methods to sparsely ray-traced [22] and foveated images [23]. In this context, machine learning methods have been explored for real-time low-sample-count reconstruction [24] and foveated reconstruction [25]. These techniques employ temporally stable UNet architectures to achieve stable reconstructed videos from noisy and/or sparse input frames, which relates to our task of interpolation for upsampling.

2.2 Image and Video Superresolution

2.2.1 Single-image Superresolution. SISR tackles the challenging task of restoring a high-resolution image from a degraded, low-resolution version. The groundbreaking SRCNN [26] started the use of a 3-layer CNN for SISR, sparking a surge of deep neural network approaches that now set the standard for quality. Rather than directly mapping the high-resolution target image to the low-resolution input image, VDSR [27] adopts a clever approach of learning the residual between the two. SRResNet [28], inspired by the residual network architecture [29], addresses superresolution by leveraging residual blocks. EDSR [30] elevates performance further by incorporating a greater number of modified residual blocks. For real-time performance, ESPCN [31] introduces a subpixel CNN that operates at low resolution. LapSRN [32], on the other hand, proposes a Laplacian pyramid network to progressively reconstruct sub-band residuals of high-resolution images. RDN [33] effectively combines residual and dense connections [34] for hierarchical feature extraction. Taking

it a step further, RCAN [35] introduces a residual-in-residual structure, forming an impressively deep network with over 400 convolutional layers, and a channel attention mechanism to adaptively rescale channel-wise features, leading to the state-of-the-art quality result. Some approaches [28, 36] rely on GANs to optimize perceptual quality, although this often comes at the expense of limited reconstruction fidelity and temporal consistency.

2.2.2 Video Superresolution. Video superresolution methods often leverage the temporal coherence present in neighboring frames to enhance the reconstruction beyond what can be achieved using SISR methods. Many of the existing methods in this domain rely on motion estimation between frames as a crucial element. VESPCN [37] introduces a multi-resolution spatial transformer module, serving the dual purpose of joint motion compensation and video superresolution. SPMC-VSR [38], on the other hand, incorporates a subpixel motion compensation layer to merge multiple frames, revealing intricate image details. EDVR [39] takes a comprehensive approach, implementing a pyramid, cascading, and deformable alignment module, alongside a temporal and spatial attention module. In contrast, DUF [40] proposes a unique deep neural network that generates dynamic upsampling filters based on the local spatio-temporal neighborhood of each pixel, effectively bypassing the need for explicit motion compensation.

Alternatively, another category of methods harnesses the power of **Recurrent Neural Networks (RNNs)**, which naturally encourage temporally consistent outcomes. FRVSR [41] introduces an RNN that warps the previously estimated frame, facilitating the subsequent frame's generation. RBPN [42], on the other hand, adopts a recurrent encoder–decoder architecture, combining features from single-image and multi-frame modules. While RNN methods promote temporal coherence, their result may lack spatial details due to the averaging nature of simple norm loss functions during training. In an attempt to address this limitation, TecoGAN [43] enhances the training loss and introduces a temporally self-supervised adversarial learning algorithm, aiming to retain both temporal consistency and spatial details in the generated video sequences.

2.3 Novelty Brought by RenderGAN

RenderGAN stands apart from the state-of-the-art approaches in real-time rendering and image superresolution due to its unique combination of DL and rendering techniques. While traditional real-time rendering engines, such as those employing real-time rendering and path-tracing rendering, the tradeoff between speed and visual quality [6], RenderGAN introduces a novel encoder–decoder architecture for image generation that leverages DL through the GAN framework. Unlike some other approaches that solely focus on improving spatial resolution or applying TAA [44], RenderGAN addresses the time–quality tradeoff inherent in real-time rendering by efficiently generating high-quality images in a real-time context. By utilizing G-Buffers and information from a real-time rendering engine as inputs (namely Albedo, Emission, Roughness, Metalness, Depth, Normal) as described in Section 3.1, RenderGAN achieves remarkable visual fidelity, surpassing the limitations of conventional real-time rendering engines. RenderGAN also stands out from traditional SISR methods by targeting the specific challenges of real-time rendering and addressing them through a DL-based solution. While SISR techniques typically enhance the resolution of a single image, RenderGAN goes beyond this scope to generate images in real time while maintaining high visual quality. Furthermore, RenderGAN distinguishes itself by being an open source solution, promoting collaboration, accessibility, and innovation within the CG community. This openness contrasts with some existing proprietary methods, making RenderGAN a platform for collective efforts to advance real-time rendering and CG techniques.

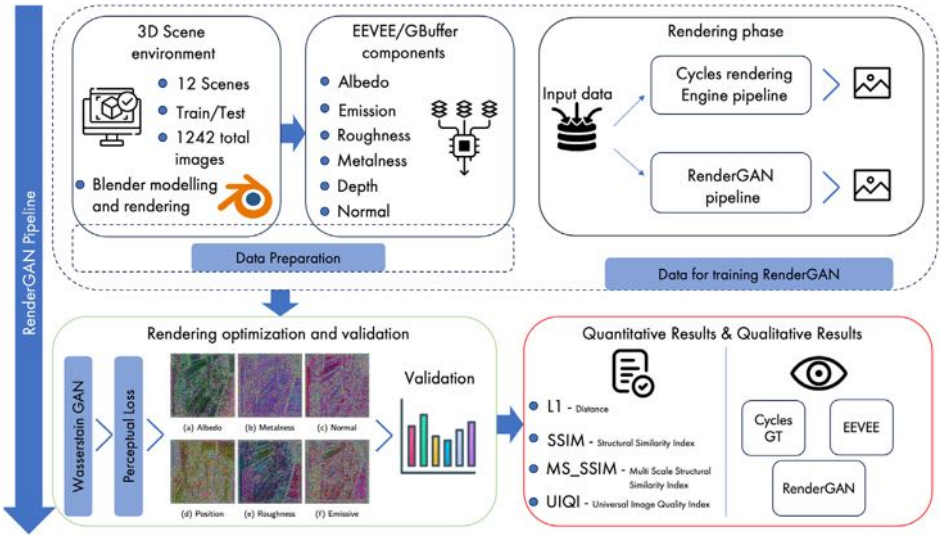


Fig. 1. The general workflow of the methodology proposed.

3 Methods

In this section, we introduce our approach based on RenderGAN, a DL-based solution designed to tackle the time–quality tradeoff in real-time CG. By using G-Buffers and data from a real-time rendering engine, RenderGAN generates output images with remarkable visual fidelity. We present the architecture of our encoder–decoder network, trained using the GAN framework with perceptual loss to enhance visual realism. The construction of the dataset, containing essential components such as G-Buffers, real-time renders, and path-tracing renders, is detailed in the following subsection. Additionally, we explain the process of training the neural network and the quantitative evaluation used to assess RenderGAN's effectiveness. Visualizations of the generated images and their similarity to path-tracing engine results further support our findings. Moreover, we emphasize RenderGAN's distinction as an open source solution, fostering collaboration, innovation, and accessibility within the CG community, setting it apart from proprietary alternatives. Through this comprehensive methodology, depicted in the workflow in Figure 1, RenderGAN paves the way for accelerated image generation and revolutionizes the future of real-time rendering, offering new possibilities for the field.

3.1 Dataset Collection

Developing an *ad hoc* dataset is essential to facilitate the training of our neural network for rendering. Despite the availability of some datasets like Structured3D [45] that offer a variety of indoor scenes, none of them provided the necessary information on Roughness, Metalness, and Emissivity. An ad-hoc dataset was carefully constructed to include all required characteristics. The dataset comprises fundamental data components, including G-Buffers, renders computed using a real-time engine, and renders generated using a path-tracing-based engine. To acquire the necessary data, we have used graphics software, with a preference for open source tools that offer scripting capabilities, enabling the extension of software functionalities and data extraction. For this purpose, Blender[1] is selected, which houses both a real-time render engine called EEVEE and a path-tracing render engine known as Cycles, while also supporting Python for further customization.

[1] https://www.blender.org

Table 1. The Dataset Presented in This Work Comprises 12 Scenes Used to Render the Images Utilized for the Rendering Process

12 Scene in total			Total images
Scene number	Train	Test	
1	70	22	92
2	82	20	102
3	74	21	95
4	54	20	74
5	52	20	72
6	177	19	196
7	87	19	106
8	10	14	24
9	73	14	87
10	121	13	134
11	139	20	159
12	96	24	120
Total	1035	207	1242

Different viewpoints are extracted from each scene through the use of multiple virtual cameras positioned within the scene to capture it from various angles. This allows for the acquisition of distinct perspectives of the same scene, accounting for varying modes of light refraction and reflection. As a result, a different number of images are obtained from each scene; some of them are used for testing, evaluating the performance metrics.

The scenes used in the dataset are sourced from SketchFab, providing diverse viewpoints for a comprehensive representation. For each selected scene, various information is extracted, encompassing:

— *Albedo*: Basic color information of materials present in the scene.
— *Emission*: Color data for light-emitting materials within the scene.
— *Roughness*: Information about the scattering of light and directions for each material.
— *Metalness*: Data indicating how a material interacts with light, reflecting or absorbing it.
— *Depth*: Information measuring the distance of objects from the camera plane within the scene.
— *Normal*: Data representing the surface normals' orientation in 3D space.

Table 1 presents an overview of the dataset, including the various information extracted from the scenes and the corresponding renders.

The 3D scenes were obtained from Sketchfab[2] and imported into Blender, where necessary adjustments were made to ensure their suitability for the intended purpose. Multiple cameras were strategically positioned to capture various views of each scene. Subsequently, the scenes were rendered using both the real-time engine and the path-tracing engine. Additionally, G-Buffer were computed simultaneously, storing all relevant information in image format and organizing it into a structured dataset. This information is depicted in Figure 2, which illustrates the basic components used for training and generating the network's output. Specifically, Figure 2(a)–(g) is used as input for the neural network presented here, while Figure 2(h) serves as a reference for the network to learn from and for assessing the test metrics.

[2]https://sketchfab.com/

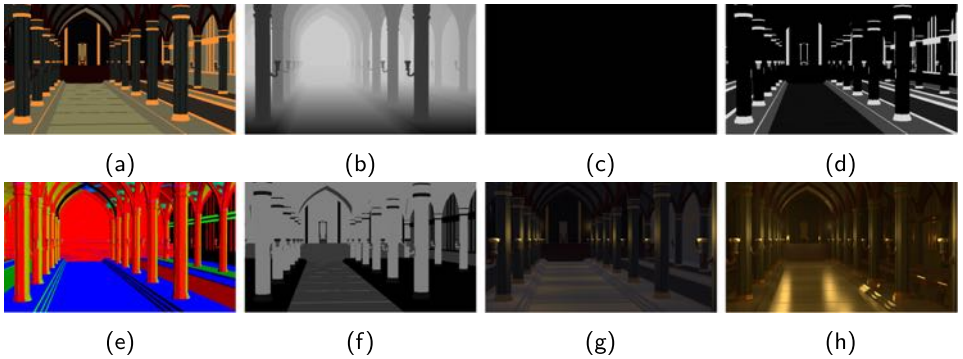


Fig. 2. Examples of the rendering dataset. (a) An example of albedo data; (b) an example of the depth map; (c) an example of the emissive information; (d) an example of the metallicness information; (e) an example of the surface normal information; (f) the roughness information; (g) an example of rendering using the real-time rendering engine EEVEE; and (h) an example of rendering using the path-tracing render engine called Cycles.

Fig. 3. Example of the rendering result. On the left are the EEVEE rendering result, and on the right, there is the Cycles rendering result.

During the image selection process, a careful examination of the rendered images enables the identification of distinct characteristics that distinguish the outputs of the EEVEE and Cycles render engines. Examples highlighting these differences are shown in Figure 3 that clearly exhibit the stark contrast in the rendering result, with Cycles providing more accurate and realistic volume and shadow arrangements, while EEVEE produces flatter and less refined images, resembling sketches rather than renders.

3.2 Network Configuration

The primary objective of our network is to discover a function that can generate rendered images perceptually similar to those produced by the Cycles engine, given input from the EEVEE image and information from the G-Buffer. To achieve this, it is crucial to consider how the algorithm interprets perception.

3.2.1 Perceptual Loss. Perceptual loss is a critical component of the overall loss function, driving the network to produce natural and visually pleasing result. Previous studies [46, 47] have explored various loss functions, including $L1$, $L2$, **Structural Similarity Index Measure (*SSIM*)**, and Multi-Scale Structural Similarity Index (MS_SSIM), to measure image quality in superresolution algorithms. However, classical metrics like $L1$ tend to produce sharpened images, disregarding the

varying importance of different regions in the image. To address this issue, we adopt the perceptual loss function proposed by Johnson et al. [48].

The perceptual loss leverages a *Loss Network*, based on the VGG16 convolutional network architecture [49], which is pre-trained on ImageNet [50]. The Loss Network extracts content and style information from both the reference (Cycles_rendered) and generated images. Specifically:

—Content information is extracted from the "$relu3_3$" layer of the generated image.
—The reference representation is extracted from the "$relu1_2$," "$relu2_2$," "$relu3_3$," and "$relu4_3$" layers of the reference image (rendered using the Cycles engine).

The extracted representations are used to compute two types of loss—Feature Reconstruction Loss and Style Reconstruction Loss [51].

The feature reconstruction loss is calculated using the output image, i.e., the image generated by the generator, and extracting its representation from the "$relu3_3$" layer. The loss function is defined as follows in Equation (1):

$$l_{feat}^{\Phi,J}(\hat{y}, y) = \frac{1}{C_J * H_J * W_J} ||\Phi_J(\hat{y})\Phi_J(y)||_2^2 \tag{1}$$

The style reconstruction loss is calculated using the generated image and the style reference obtained from the "$relu1_2$," "$relu2_2$," "$relu3_3$," and "$relu4_3$" layers of the Loss Network in Equation (2):

$$l_{style}^{\Phi,J} = ||G_J^{\Phi}(\hat{y})G_J^{\Phi}(y)||_F^2 \tag{2}$$

The total loss is typically the weighted sum of Equation (2).

The network architecture used for rendering image generation combines the encoder–decoder approach with the GAN framework. The network consists of seven encoders based on the EfficientNet [52], which extract features from the input images (EEVEE and G-Buffer). These extracted features are then concatenated and used as input for the decoder, which employs transposed convolution for image generation.

3.3 Transposed Convolution Issues

Transposed convolution, also known as Upsampled Convolution, allows upsampling of input feature maps. It overcomes some of the limitations of other upsampling methods, such as Nearest Neighbors, BiLinear Interpolation, Bed of Neils, and Max Pooling.

T convolution stands as a pivotal operation within neural network architectures, particularly when upsampling spatial dimensions is imperative, such as in GAN or certain segmentation tasks. Instead of aggregating spatial information as in standard convolution, transposed convolution aims to distribute spatial information, effectively expanding the spatial dimensions of input feature maps. The procedure entails the application of a convolutional kernel to each input value, producing expanded local outputs which might overlap with outputs generated from neighboring inputs. These overlaps, when summed together, form the upsampled result. The intricacies of this operation, including kernel size, stride, and padding, determine the spatial relationships in the resulting output, making it a versatile tool in model design but also one that requires careful configuration to prevent artifacts.

To address this issue, previous studies [53, 54] proposed various solutions, including using non-divisible kernel sizes for the stride and employing the nearest neighbor approach followed by convolution. In our approach, we combine the strengths of these methods by using transposed convolution layers in conjunction with convolution layers to eliminate patterns. Specifically, we apply normal convolution with a kernel size equal to 1, which leaves the feature size unchanged.

This creates a projection of the image obtained through transposed convolution onto a continuous space, preserving all information while eliminating undesirable patterns. The modified approach also establishes connections between various extracted data, ensuring a cohesive link between different input sources. This solution employs transposed convolution layers with convolution layers to eliminate patterns. Specifically, we use normal convolution with a kernel size equal to 1, leaving the feature size unchanged. This creates a projection of the image obtained through transposed convolution onto a continuous space, preserving all information while eliminating undesirable patterns. The modified approach establishes connections between various extracted data, ensuring a cohesive link between different input sources. By leveraging this hybrid approach, we mitigate the issues associated with transposed convolution and improve the overall quality of the rendered images. The resulting network architecture effectively addresses perceptual loss and transposed convolution problems, allowing us to generate visually appealing and realistic images closely resembling those produced by the Cycles engine.

3.4 Image Quality Evaluation Indices

In order to assess the performance and improvement of RenderGAN, a comprehensive image quality evaluation is essential. For this purpose, we employ two prominent indices—the SSIM [55] and the UIQI [56].

- —The SSIM is employed in this study to evaluate the similarity between two images. SSIM, as defined in [57], incorporates the concept of MSSSIM, which applies SSIM on M scales, with scale 1 representing the input image. The SSIM index measures the structural similarity between the generated images and the ground truth. The SSIM index ranges from −1 to 1, where a value of 1 indicates perfect similarity, a value close to −1 suggests strong dissimilarity, and 0 indicates no correlation between the images.
- —The UIQI is another metric employed in this study, and its equations are provided in [56]. The UIQI mathematically defines image distortion relative to a reference image by considering the following three factors: loss of correlation, luminance distortion, and contrast distortion.

These metrics offer valuable insights into the similarity and distortion between the generated images and the reference images. The SSIM utilizes a multi-scale approach, measuring image similarity across different scales, while the UIQI models image distortion by considering the loss of correlation, luminance distortion, and contrast distortion relative to the reference image.

The trend of these metrics is analyzed to assess the progressive improvement in the quality of the network-generated images throughout the training process.

4 Results and Discussion

In this section, we present the result obtained from our rendering approach, which was evaluated using a 40GB A100 and trained for 500 epochs. The RMSProp optimizer, recommended in the Wasserstein GAN framework, was utilized to optimize the network's performance. To gauge the progress of the training process, we monitored the loss trends of both the discriminator and the generator. As expected, the discriminator loss demonstrated a decreasing trend, reaching a final value of 0.000146, indicating that the discriminator effectively learned to distinguish between real and generated images. On the other hand, the generator's $L1$ distance steadily decreased to a value of 0.009841, signifying the consistent improvement of the generator's output over the training period. Furthermore, the perceptual loss, an important component of our network's overall loss function, exhibited a declining trend, reaching a minimum value of 0.09785. Additionally, the graphs revealed that the generator was trained every three updates of the discriminator, ensuring a

Table 2. Result of the RenderGAN Network with One Additional Input

EEVEE	$L1$	PL	SSIM	MSSIM	UIQI
Albedo	0.303	1.51	0.695	0.586	0.00087
Depth	0.235	1.62	0.699	0.794	0.003
Normal	0.323	1.68	0.600	0.553	0.00084
Emissive	0.398	1.55	0.719	0.689	0.00011
Metalness	0.368	2.03	0.776	0.708	0.0025
Roughness	0.368	1.88	0.723	0.653	0.00050
Position	0.262	1.45	0.745	0.591	0.00056
All	0.009	0.09	0.912	0.982	0.898

balanced and effective training process. These results provide valuable insights into the effectiveness of our rendering approach, demonstrating the network's ability to produce high-quality and visually realistic images.

4.1 Ablation Study

To verify that the result obtained need all input data, an ablation study was applied to the convolutional layers of the encoder to understand which data are beneficial. Data were used individually as well as in combination. Tables 2–5 present, in the first column, the additional data used by the neural network for generating the output. Specifically, for each table, the first row defines the common data, while the subsequent rows detail the additional data used for training and, consequently, for testing the network. All tests were conducted using renderings obtained with the Cycles rendering engine in Blender as reference baseline.

The visualizations in Appendix A provide further insights into the impact of different input combinations and the network's performance with 500 epochs of training.

From the visualizations, it is evident that using fewer input data during training (e.g., Albedo and Depth, Albedo and Emissive) allows the network to learn the structural representation from the input images. However, the reconstruction of colors associated with the structures is compromised. Figure 4 displays several examples of images generated with the Cycles rendering engine, which are used as references for calculating the SSIM, MSSIM, and UIQI metrics in Section 3.4. Additionally, analyzing the usefulness of convolutional components for the intermediate data space (Z for encoder–decoder architectures), we find that combining EEVEE as input with Albedo, Depth, Emissive, Metalness, Position, and Roughness information allows the network to effectively reconstruct the image structure. Figures 4–6 are the visualization obtained with the use of the basic rendered image with EEVEE and the one obtained with all the additional component from the G-Buffer used as input for the neural network rendering generation, respectively.

Further analysis reveals that utilizing additional information such as Albedo and Normal, Albedo and Emissive, and so on, provides better results in terms of image structure recognition. This is supported by numerical results for SSIM and MSSIM metrics, whereas for Perceptual Loss, Albedo and Normal perform well. Similarly, using Roughness information in combination with Depth yields favorable visual results and is confirmed by the Perceptual Loss value. However, when Emissive is used in combination with other information, the network shows excellent numerical results on average but fails to consistently recognize the structure correctly in visualizations.

Furthermore, the use of Normal in combination with other inputs allows the network to recognize the image's structure, but the image reconstruction is not as reliable, except when combined with Position information. On the other hand, Emissive combined with Metalness enables structure

Table 3. RenderGAN Result on Double Input Data

EEVEE	$L1$	PL	SSIM	MSSIM	UIQI
Depth+					
Normal	0.362	1.62	0.651	0.537	0.002
Albedo	0.244	1.61	0.661	0.663	0.005
Emissive	0.432	2.29	0.930	0.826	0.001
Metalness	0.501	2.20	0.773	0.598	0.0005
Roughness	0.408	1.51	0.735	0.598	0.0005
Position	0.392	2.10	0.756	0.671	0.001
Albedo+					
Normal	0.305	1.57	0.633	0.549	0.001
Emissive	0.449	2.53	0.913	0.836	0.0005
Metalness	0.409	2.09	0.687	0.637	0.001
Roughness	0.398	2.11	0.601	0.605	0.005
Position	0.414	2.12	0.664	0.607	0.0001
Normal+					
Emissive	0.406	1.98	0.883	0.688	0.0003
Metalness	0.494	2.39	0.661	0.559	0.0003
Roughness	0.612	2.10	0.624	0.515	0.0004
Position	0.454	2.15	0.622	0.597	0.0009
Emissive+					
Metalness	0.349	2.25	0.895	0.775	0.003
Roughness	0.283	2.09	0.900	0.837	0.002
Position	0.445	1.90	0.922	0.823	0.0008
Metalness+					
Roughness	0.590	2.29	0.275	0.901	0.0001
Position	0.443	2.31	0.716	0.649	0.0009

recognition and reconstruction, but not entirely. The numerical values confirm these observations. Finally, using four pieces of information as input allows the network to recognize the structure effectively, especially when combining Roughness and Position with Normal and Emissive.

In evaluating the numerical metrics, it becomes apparent that the values of SSIM and MSSIM are strongly influenced by the recognition of the image structure rather than its content, including brightness, contrast, and shadow gradations. Consequently, the UIQI emerges as the most reliable metric for verifying the correct reconstruction of the rendering. Comparing the generated images (Figure 6) with the ground truth images (Figure 4), it is evident that the structural reconstruction is accurate, as supported by the Perceptual Loss and average UIQI values. Furthermore, the illumination in the generated images, while differing from the expected illumination, allows for correct visualization. This is especially noticeable when the generated images have more lumen than the ground truth images, as the light distribution is better calculated, as demonstrated in Figure 4 compared to the generated one in Figure 6 where the light is lower than the ground truth. This highlights the significance of the information present in the G-Buffer for improved light distribution in the generated images.

The RMSProp optimizer was used as suggested in the Wasserstein GAN framework [58]. The network was trailed for 500 epochs. Graphs of the loss trend are shown on the x-axis are the steps, while on the y-axis are the values achieved. As expected from the discriminator loss, it is decreasing and at the end of the training has the value of 0.000146. For the generator, the $L1$ distance reaches a

Table 4. RenderGAN Result on Triple Input Data

EEVEE	$L1$	PL	SSIM	MSSIM	UIQI
Depth+Albedo+					
Normal	0.285	1.71	0.606	0.640	0.004
Emissive	0.220	0.98	0.870	0.736	0.002
Metalness	0.227	1.56	0.715	0.708	0.006
Roughness	0.300	1.23	0.667	0.575	0.001
Position	0.305	1.64	0.692	0.643	0.002
Albedo+Normal+					
Metalness	0.229	1.36	0.686	0.679	0.007
Emissive	0.289	1.53	0.852	0.675	0.001
Roughness	0.167	1.27	0.622	0.677	0.011
Position	0.239	1.77	0.528	0.612	0.006
Normal+Emissive+					
Metalness	0.318	1.83	0.856	0.767	0.004
Roughness	0.286	1.46	0.869	0.701	0.002
Position	0.250	1.55	0.813	0.678	0.002
Emissive+Metalness+					
Roughness	0.244	1.07	0.864	0.681	0.0001
Position	0.229	1.32	0.862	0.732	0.003

Table 5. RenderGAN Result on Four Input Data

EEVEE	$L1$	PL	SSIM	MSSIM	UIQI
Depth+Albedo+Normal+					
Emissive	0.224	0.94	0.638	0.508	0.0005
Metalness	0.290	0.94	0.874	0.644	0.001
Roughness	0.241	1.46	0.534	0.549	0.002
Position	0.305	1.52	0.678	0.641	0.003
Albedo+Normal+Emissive+					
Metalness	0.247	1.80	0.783	0.688	0.002
Roughness	0.304	1.87	0.864	0.780	0.004
Position	0.390	1.95	0.838	0.712	0.002
Normal+Emissive+Metalness+					
Roughness	0.249	1.81	0.816	0.772	0.006
Position	0.278	1.41	0.899	0.805	0.004
Position	0.257	1.14	0.826	0.691	0.001

value of 0.009841, with a decreasing monotony. The perceptual loss has a decreasing trend with a minimum value of 0.09785.

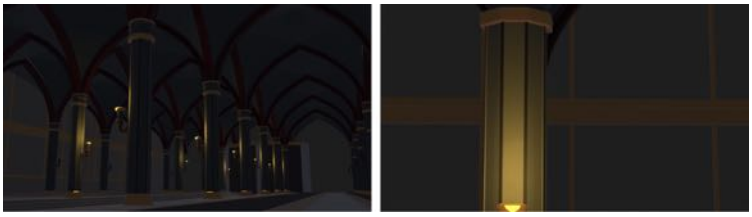
As can be seen from the graphs, the generator is trained every three updates of the discriminator. The metrics measured in the training phase and their trend are shown in Figure 7.

The *SSIM* is used to measure the similarity between two images and is defined in [57] as the *multi-scale SSIM* makes use of the SSIM on M scales with scale 1 as the input image.

The UIQI is defined in, and the equations are given in [56].

(a) (b)

Fig. 4. The ground truth images for the generated render in Figure 6.

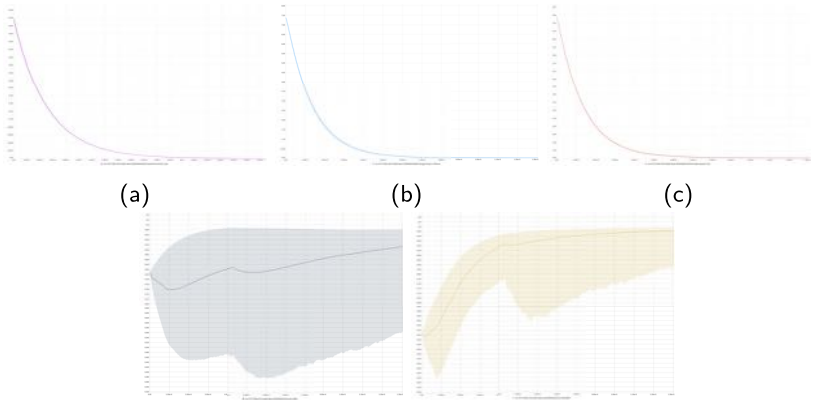


(a) (b)

Fig. 5. The EEVEE input images for the render in Figure 6.

(a) (b)

Fig. 6. The RenderGAN output images for the images shown in other pictures but obtained with EEVEE combined with all the other information from G-Buffer.



(a) (b) (c)

(d) (e)

Fig. 7. (a) Discriminator loss; (b) generator $L1$ distance; (c) perceptual loss; (d) SSIM metric; and (e) MSSIM.

4.2 Time for Render Generation

To evaluate the performance of RenderGAN in terms of rendering time, two different hardware configurations were used for testing:

— *Configuration 1*: This setup includes an Intel i7 10th generation processor, 64GB DDR4 RAM, a 1TB SSD, an NVIDIA RTX 2070 Super with 8GB GDDR6 video memory, and Ubuntu Linux 21.04 as the operating system.
— *Configuration 2*: This configuration comprises an Intel Xeon E5 Silver processor, 128GB DDR4 RAM, a 4TB HDD, an NVIDIA RTX 2080 Ti with 11GB GDDR6 video memory, and Ubuntu Linux 20.04 as the operating system.

During the test phase, only the generator block of the GAN configuration was used for rendering, occupying approximately 9,761MB of the available 11GB video memory on the RTX 2080 Ti.

Two tests were conducted to assess the rendering speed of the network using information from the G-Buffer and pre-calculated EEVEE renders:

— *Test 1*: In this test, a single render was requested from the network. The render path tracing was returned within approximately 1 second.
— *Test 2*: The network was requested to create 25 consecutive renders. On average, each render took approximately 1.05 seconds.

The test result provides valuable insights into the rendering performance of RenderGAN. For a single render request, the network demonstrated swift response times, taking around 1 second in both hardware configurations. Moreover, in a more demanding scenario where multiple renders were requested consecutively, the network continued to deliver efficient performance. On average, each of the 25 consecutive renders took approximately 1.05 seconds, showcasing the network's capability to handle continuous rendering tasks effectively.

The tests were further categorized into the following scenarios:

— *Test 1*: This scenario emulates a spot request, where a single mesh generation is requested from the network. The resulting mesh was obtained within 4 seconds using Configuration 1 and 3 seconds using Configuration 2.
— *Test 2*: In this scenario, multiple mesh generations with successive requests were conducted to simulate a working environment. The network was requested to create several basic meshes consecutively. For this purpose, the request to generate a mesh was repeated 25 consecutive times. The average generation times for this scenario were 4.16 seconds on Configuration 1 and 3.27 seconds on Configuration 2.

The rendering times obtained from these tests demonstrate the efficient and practical applicability of RenderGAN in real-world scenarios of designers and practitioners dealing with real-time rendering, making it a promising tool for various 3D rendering applications.

5 Conclusions and Future Works

In this article, we introduced RenderGAN, a novel DL-based solution that successfully addresses the time–quality tradeoff in real-time CG. By leveraging G-Buffer and information from a real-time rendering engine, RenderGAN achieves remarkable visual fidelity in generating output images. The encoder–decoder architecture, trained using the GAN framework with perceptual loss, enhances image realism, showcasing RenderGAN's ability to produce high-quality result. Our quantitative evaluation revealed RenderGAN's impressive performance, with a UIQI value of 0.898, demonstrating its superiority over traditional rendering methods. Visualizations further confirmed the striking

similarity between RenderGAN's generated images and those produced by path-tracing engines, validating the effectiveness of our approach. It is worth mentioning, as stated in Section 2, that Cycles[3] was selected as the rendering engine because it provides access to G-Buffer information as a preliminary stage of the final computation, being an open source engine freely available on the Internet. In contrast, competing engines, while valid alternatives, do not offer this information easily due to their closed source nature. RenderGAN's openness as an open source solution promotes collaboration, innovation, and accessibility within the CG community. This fosters collective efforts to advance real-time rendering techniques, benefiting a wide range of applications and industries. While RenderGAN marks a significant advancement in real-time CG, there are several avenues for future research and improvements. Further exploring optimization techniques to enhance RenderGAN's real-time performance, ensuring its applicability in interactive and dynamic rendering environments. It is possible to investigate hybrid approaches that combine real-time rendering with path-tracing techniques to achieve an even higher level of visual fidelity and realism as well as explore advanced GAN architectures and incorporate cutting-edge techniques from the DL community to enhance the capabilities of RenderGAN. We also plan to address the challenges posed by complex scenes and large-scale environments to ensure RenderGAN's scalability and applicability to a wide variety of real-world scenarios, focusing on improving temporal consistency in video superresolution tasks, and enabling RenderGAN to deliver smooth and coherent video sequences. Extending RenderGAN's capabilities, we aim to handle diverse types of input data and generate high-quality outputs for a broad range of rendering and image generation tasks, as well as enabling immersive and visually stunning experiences in virtual environments. By pursuing these avenues of research, we aim to further enhance RenderGAN's performance, expand its scope of applications, and continue pushing the boundaries of real-time CG and rendering techniques, ultimately advancing the state-of-the-art in this rapidly evolving field.

Acknowledgment

The authors would like to thank Roberto Broccoletti for helping in the dataset collection and annotation.

References

[1] Mamurova Feruza Islamovna, Nadira Sharifovna Khadjaeva, and Elena Vladimirovna Kadirova. 2023. Role and application of computer graphics. *Innovative Society: Problems, Analysis and Development Prospects (Spain)*, 1–3.

[2] Max K. and Agoston, Max K. Agoston. 2005. *Computer Graphics and Geometric Modeling*. Vol. 1. Springer.

[3] Pavel Peresunko, Denis Mamatin, Oleslav Antamoshkin, Evgeniya Peresunko, and Alexander Nikitin. 2021. Models of experts for shaders estimation of rendering complex 3D scenes in real time. In *2021 3rd International Conference on Control Systems, Mathematical Modeling, Automation and Energy Efficiency (SUMMA)*. IEEE, 895–897.

[4] Xiaotian Ren, Zhengyong Jiang, and Jionglong Su. 2021. The use of features to enhance the capability of deep reinforcement learning for investment portfolio management. In *2021 IEEE 6th International Conference on Big Data Analytics (ICBDA)*. IEEE, 44–50.

[5] Lifan Wu, Guangyan Cai, Shuang Zhao, and Ravi Ramamoorthi. 2020. Analytic spherical harmonic gradients for real-time rendering with many polygonal area lights. *ACM Transactions on Graphics* 39, 4 (2020), 134–131.

[6] Khandoker Ashik Uz Zaman, Ashraful Islam, and Md Abu Sayed. 2024. Render lighting dataset: A collection of rendered images with varied lighting conditions using blender render engines. *Data in Brief* 54 (2024), 110331.

[7] Lei Xiao, Salah Nouri, Matt Chapman, Alexander Fix, Douglas Lanman, and Anton Kaplanyan. 2020. Neural supersampling for real-time rendering. *ACM Transactions on Graphics* 39, 4 (2020), 142–141.

[8] Lucas Figueiredo, Paulo Ivson, and Waldemar Celes. 2023. Unsupervised method for identifying shape instances on 3D CAD models. *Computers & Graphics* 116 (2023), 228–238.

[9] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2020. Generative adversarial networks. *Communications of the ACM* 63, 11 (2020), 139–144.

[3] https://github.com/blender/cycles

[10] Vinicius Luis Trevisan de Souza, Bruno Augusto Dorta Marques, Harlen Costa Batagelo, and João Paulo Gois. 2023. A review on generative adversarial networks for image generation. *Computers & Graphics* 114 (2023), 13–25.
[11] Liang Shi, Beichen Li, Changil Kim, Petr Kellnhofer, and Wojciech Matusik. 2021. Towards real-time photorealistic 3D holography with deep neural networks. *Nature* 591, 7849 (2021), 234–239.
[12] Kurt Akeley. 1993. Reality engine graphics. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, 109–116.
[13] Lance Williams. 1983. Pyramidal parametrics. In *Proceedings of the 10th Annual Conference on Computer Graphics and Interactive Techniques*, 1–11.
[14] Anton S. Kaplanyan, Stephan Hill, Anjul Patney, and Aaron E. Lefohn. 2016. Filtering distributions of normals for shading antialiasing. In *Proceedings of the Conference on High Performance Graphics*, 151–162.
[15] William T. Reeves, David H. Salesin, and Robert L. Cook. 1987. Rendering antialiased shadows with depth maps. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, 283–291.
[16] Alexander Reshetov. 2009. Morphological antialiasing. In *Proceedings of the Conference on High Performance Graphics*, 109–116.
[17] Jorge Jimenez, Jose I. Echevarria, Tiago Sousa, and Diego Gutierrez. 2012. SMAA: Enhanced subpixel morphological antialiasing. In *Computer Graphics Forum*. Vol. 31, Wiley Online Library, 355–364.
[18] Lei Yang, Diego Nehab, Pedro V. Sander, Pitchaya Sitthi-Amorn, Jason Lawrence, and Hugues Hoppe. 2009. Amortized supersampling. *ACM Transactions on Graphics* 28, 5 (2009), 1–12.
[19] Brian Karis. 2014. High quality temporal anti-aliasing. *Advances in Real-Time Rendering for Games, SIGGRAPH Courses*.
[20] Oliver Osman. 2021. Design and development of a 3D survival platformer game ith unreal engine 4.
[21] Andrew Edelsten, Paula Jukarainen, and Anjul Patney. 2019. Truly next-gen: Adding deep learning to games and graphics. In *NVIDIA Sponsored Sessions (Game Developers Conference)*.
[22] Christoph Schied, Anton Kaplanyan, Chris Wyman, Anjul Patney, Chakravarty R. Alla Chaitanya, John Burgess, Shiqiu Liu, Carsten Dachsbacher, Aaron Lefohn, and Marco Salvi. 2017. Spatiotemporal variance-guided filtering: real-time reconstruction for path-traced global illumination. In *Proceedings of the Conference on High Performance Graphics*, 1–12.
[23] Anjul Patney, Marco Salvi, Joohwan Kim, Anton Kaplanyan, Chris Wyman, Nir Benty, David Luebke, and Aaron Lefohn. 2016. Towards foveated rendering for gaze-tracked virtual reality. *ACM Transactions on Graphics* 35, 6 (2016), 1–12.
[24] Chakravarty R. Alla Chaitanya, Anton S. Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila. 2017. Interactive reconstruction of Monte Carlo image sequences using a recurrent denoising autoencoder. *ACM Transactions on Graphics* 36, 4 (2017), 1–12.
[25] Anton S. Kaplanyan, Anton Sochenov, Thomas Leimkühler, Mikhail Okunev, Todd Goodall, and Gizem Rufo. 2019. DeepFovea: Neural reconstruction for foveated rendering and video compression using learned statistics of natural videos. *ACM Transactions on Graphics* 38, 6 (2019), 1–13.
[26] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. 2015. Image super-resolution using deep convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38, 2 (2015), 295–307.
[27] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. 2016. Accurate image super-resolution using very deep convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1646–1654.
[28] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. 2017. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4681–4690.
[29] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 770–778.
[30] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. 2017. Enhanced deep residual networks for single image super-resolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 136–144.
[31] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew, P. Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. 2016. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1874–1883.
[32] Wei-Sheng Lai, Jia-Bin Huang, Narendra Ahuja, and Ming-Hsuan Yang. 2017. Deep Laplacian pyramid networks for fast and accurate super-resolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 624–632.
[33] Yulun Zhang, Yapeng Tian, Yu Kong, Bineng Zhong, and Yun Fu. 2018. Residual dense network for image super-resolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2472–2481.

[34] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4700–4708.
[35] Yulun Zhang, Kunpeng Li, Kai Li, Lichen Wang, Bineng Zhong, and Yun Fu. 2018. Image super-resolution using very deep residual channel attention networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 286–301.
[36] Weifeng Ge, Bingchen Gong, and Yizhou Yu. 2018. Image super-resolution via deterministic-stochastic synthesis and local statistical rectification. *ACM Transactions on Graphics* 37, 6 (2018), 1–14.
[37] Jose Caballero, Christian Ledig, Andrew Aitken, Alejandro Acosta, Johannes Totz, Zehan Wang, and Wenzhe Shi. 2017. Real-time video super-resolution with spatio-temporal networks and motion compensation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4778–4787.
[38] Xin Tao, Hongyun Gao, Renjie Liao, Jue Wang, and Jiaya Jia. 2017. Detail-revealing deep video super-resolution. In *Proceedings of the IEEE International Conference on Computer Vision*, 4472–4480.
[39] Xintao Wang, Kelvin, C. K. Chan, Ke Yu, Chao Dong, and Chen Change Loy. 2019. EDVR: Video restoration with enhanced deformable convolutional networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*.
[40] Younghyun Jo, Seoung Wug Oh, Jaeyeon Kang, and Seon Joo Kim. 2018. Deep video super-resolution network using dynamic upsampling filters without explicit motion compensation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 3224–3232.
[41] Raviteja Vemulapalli, Matthew Brown, and Seyed Mohammad Mehdi Sajjadi. 2020. Frame-recurrent video super-resolution. US Patent No. 10,783,611.
[42] Muhammad Haris, Gregory Shakhnarovich, and Norimichi Ukita. 2019. Recurrent back-projection network for video super-resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 3897–3906.
[43] Mengyu Chu, You Xie, Jonas Mayer, Laura Leal-Taixé, and Nils Thuerey. 2020. Learning temporal coherence via self-supervision for GAN-based video generation. *ACM Transactions on Graphics* 39, 4 (2020), 75–71.
[44] Xueyan Zou, Fanyi Xiao, Zhiding Yu, Yuheng Li, and Yong Jae Lee. 2023. Delving deeper into anti-aliasing in convnets. *International Journal of Computer Vision* 131, 1 (2023), 67–81.
[45] Jia Zheng, Junfei Zhang, Jing Li, Rui Tang, Shenghua Gao, and Zihan Zhou. 2020. Structured3d: A large photo-realistic dataset for structured 3d modeling. In *European Conference on Computer Vision*, Springer, 519–535.
[46] Changsheng Zhou, Jiangshe Zhang, Junmin Liu, Chunxia Zhang, Rongrong Fei, and Shuang Xu. 2020. PercepPan: Towards unsupervised pan-sharpening based on perceptual loss. *Remote Sensing* 12, 14 (2020), 2318.
[47] Keyan Ding, Kede Ma, Shiqi Wang, and Eero P. Simoncelli. 2021. Comparison of full-reference image quality models for optimization of image processing systems. *International Journal of Computer Vision* 129, 4 (2021), 1258–1281.
[48] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. 2016. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, Springer, 694–711.
[49] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. arXiv:1409.1556. Retrieved from https://arxiv.org/abs/1409.1556
[50] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 248–255. DOI: https://doi.org/10.1109/CVPR.2009.5206848
[51] Qiang Cai, Mengxu Ma, Chen Wang, and Haisheng Li. 2023. Image neural style transfer: A review. *Computers and Electrical Engineering* 108 (2023), 108723.
[52] Mingxing Tan and Quoc Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the International Conference on Machine Learning*. PMLR, 6105–6114.
[53] Jon Gauthier. 2014. Conditional generative adversarial nets for convolutional face generation. *Class Project for Stanford CS231N: Convolutional Neural Networks for Visual Recognition, Winter Semester*, 2.
[54] Augustus Odena, Vincent Dumoulin, and Chris Olah. 2016. Deconvolution and checkerboard artifacts. *Distill*. DOI: https://doi.org/10.23915/distill.00003. Retrieved from http://distill.pub/2016/deconv-checkerboard.
[55] Dominique Brunet, Edward R. Vrscay, and Zhou Wang. 2012. On the mathematical properties of the structural similarity index. *IEEE Transactions on Image Processing* 21, 4 (2012), 1488–1499. DOI: https://doi.org/10.1109/TIP.2011.2173206
[56] Zhou Wang and A. C. Bovik. 2002. A universal image quality index. *IEEE Signal Processing Letters* 9, 3 (2002), 81–84. DOI: https://doi.org/10.1109/97.995823
[57] Zhou Wang, Eero P. Simoncelli, and Alan C. Bovik. 2003. Multiscale structural similarity for image quality assessment. In *37th Asilomar Conference on Signals, Systems & Computers*, Vol. 2. IEEE, 1398–1402.
[58] Jonas Adler and Sebastian Lunz. 2018. Banach Wasserstein GAN. In *Proceedings of the 32nd Advances in Neural Information Processing Systems*, 6755–6764.

A Appendix

This appendix is meant to complete the information about our ablation study. In particular, Figures A1–A13 report the different combination of parameters from the G-Buffer.

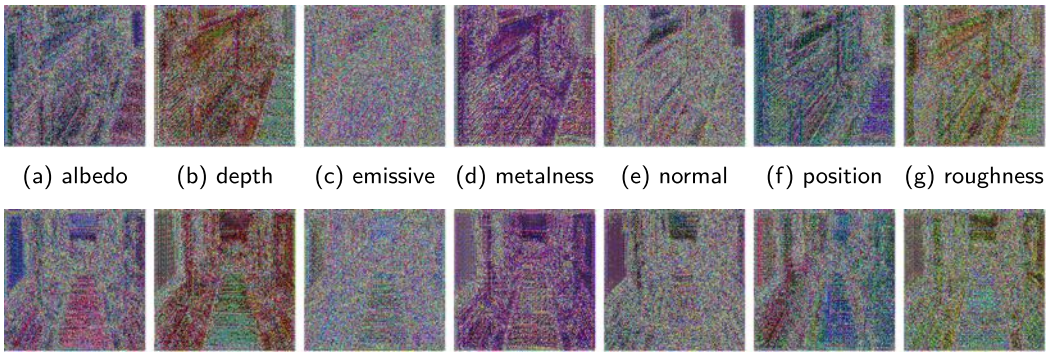


Fig. A1. The output generated from RenderGAN with single additional information used in combination with the EEVEE. Numerical result is reported in Table 2.

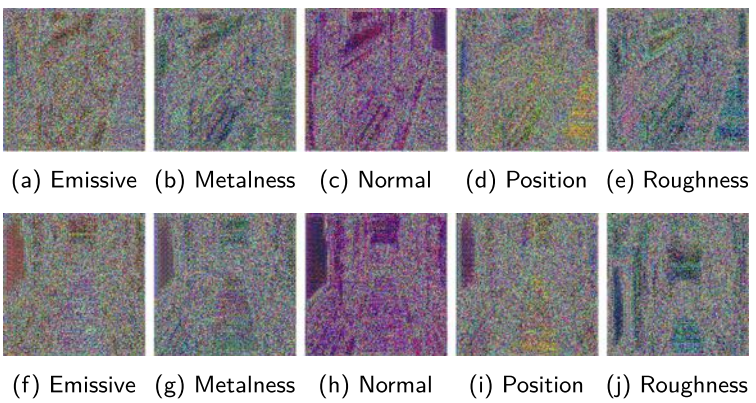
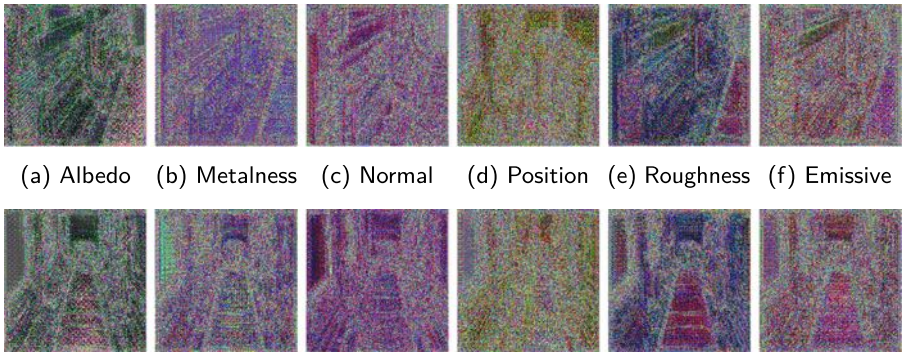


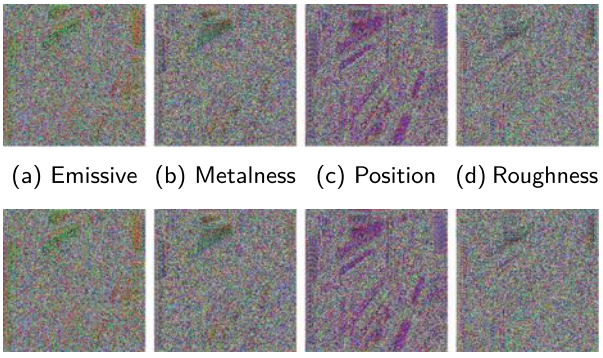
Fig. A2. The output generated from RenderGAN with two additional pieces of information used in combination with the EEVEE. In this picture, the combination first element is fixed on the albedo information and the second one change. Numerical result is reported in Table 3.



(a) Albedo (b) Metalness (c) Normal (d) Position (e) Roughness (f) Emissive

(g) Albedo (h) Metalness (i) Normal (j) Position (k) Roughness (l) Emissive

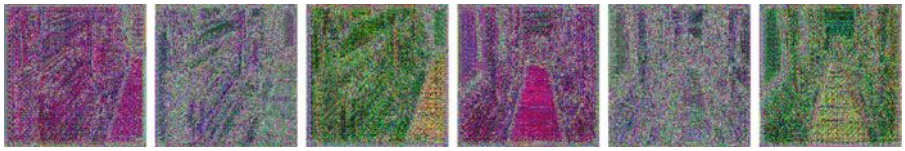
Fig. A3. The output generated from RenderGAN with two additional pieces of information used in combination with the EEVEE. In this picture, the combination first element is fixed on the depth information and the second one change. Numerical result is reported in Table 3.



(a) Emissive (b) Metalness (c) Position (d) Roughness

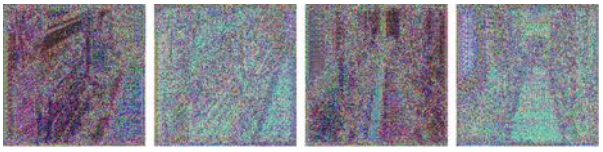
(e) Emissive (f) Metalness (g) Position (h) Roughness

Fig. A4. The output generated from RenderGAN with two additional pieces of information used in combination with the EEVEE. In this picture, the combination first element is fixed on the normal information and the second one change. Numerical result is reported in Table 3.



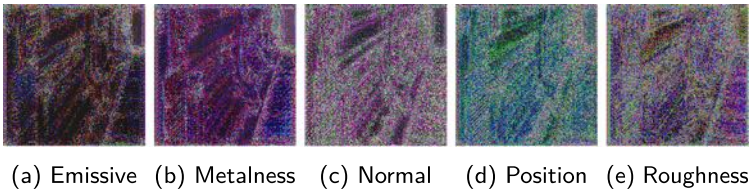
(a) Metalness (b) Position (c) Roughness (d) Metalness (e) Position (f) Roughness

Fig. A5. The output generated from RenderGAN with two additional pieces of information used in combination with the EEVEE. In this picture, the combination first element is fixed on the emissive information and the second one change. Numerical result is reported in Table 3.

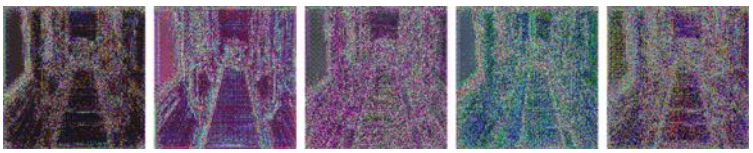


(a) Position (b) Roughness (c) Position (d) Roughness

Fig. A6. The output generated from RenderGAN with two additional pieces of information used in combination with the EEVEE. In this picture, the combination first element is fixed on the metalness information and the second one change. Numerical result is reported in Table 3.

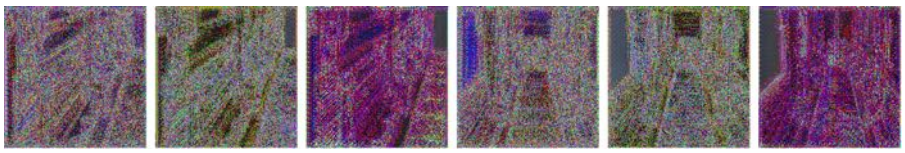


(a) Emissive (b) Metalness (c) Normal (d) Position (e) Roughness



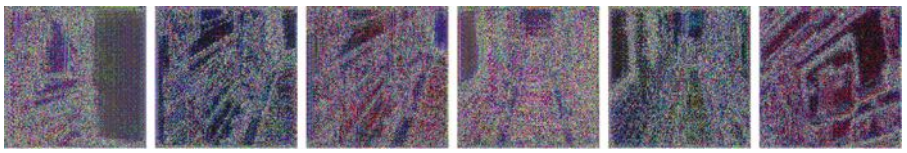
(f) Emissive (g) Metalness (h) Normal (i) Position (j) Roughness

Fig. A7. The output generated from RenderGAN with three additional pieces of information used in combination with the EEVEE. In this picture, the combination first two elements are fixed on the depth and albedo information and the second one change. Numerical result is reported in Table 4.



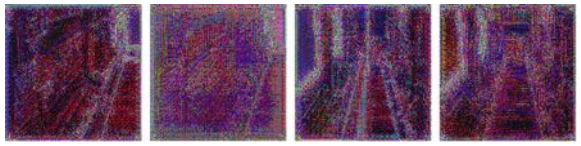
(a) Emissive (b) Metalness (c) Position (d) Emissive (e) Metalness (f) Position

Fig. A8. The output generated from RenderGAN with three additional pieces of information used in combination with the EEVEE. In this picture, the combination of first two elements are fixed on the albedo and normal information and the second one change. Numerical result is reported in Table 4.



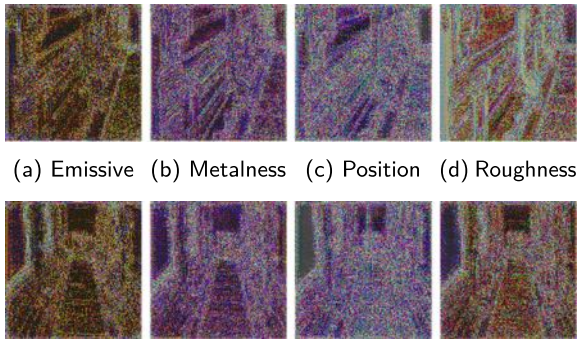
(a) Metalness (b) Position (c) Roughness (d) Metalness (e) Position (f) Roughness

Fig. A9. The output generated from RenderGAN with three additional pieces of information used in combination with the EEVEE. In this picture, the combination first two elements are fixed on the normal and emissive information and the second one change. Numerical result is reported in Table 4.



(a) Position (b) Roughness (c) Position (d) Roughness

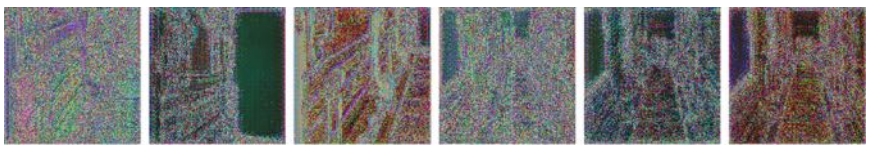
Fig. A10. The output generated from RenderGAN with three additional pieces of information used in combination with the EEVEE. In this picture, the combination first two elements are fixed on the emissive and metalness information and the second one change. Numerical result is reported in Table 4.



(a) Emissive (b) Metalness (c) Position (d) Roughness

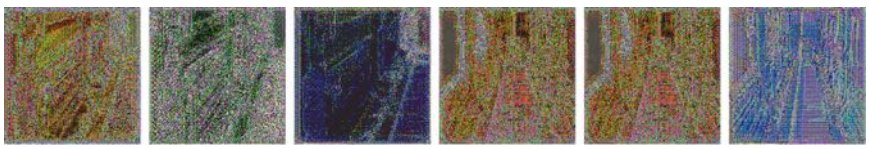
(e) Emissive (f) Metalness (g) Position (h) Roughness

Fig. A11. The output generated from RenderGAN with four additional pieces of information used in combination with the EEVEE. In this picture, the combination first two elements are fixed on the depth and albedo and normal information and the second one change. Numerical result is reported in Table 5.



(a) Position (b) Metalness (c) Roughness (d) Position (e) Metalness (f) Roughness

Fig. A12. The output generated from RenderGAN with four additional information used in combination to the EEVEE. In this picture, the combination first two elements is fixed on the albedo and normal and emissive information and the second one change. Numerical result is reported in Table 5.



(a) Metalness_Position (b) Metalness_Roughness (c) Roughness_Position (d) Metalness_Position (e) Metalness_Roughness (f) Roughness_Position

Fig. A13. The output generated from RenderGAN with four additional pieces of information used in combination with the EEVEE. In this picture, the combination first two elements are fixed on the normal and emissive information and the second one change. Numerical result is reported in Table 5.

Received 9 December 2023; revised 14 September 2024; accepted 5 January 2025