



CHALMERS
UNIVERSITY OF TECHNOLOGY



TransFURmer: Hair and Fur rendering using SwinIR

Transformer-based Neural Rendering of Hair and Fur using SwinIR

Master's thesis in Computer Science - Algorithms, Languages and Logic

Ryan Jin Kevin Jönsson

MASTER'S THESIS 2026

TransFURmer: Hair and Fur rendering using SwinIR

Transformer-based Neural Rendering of Hair and Fur using SwinIR

Ryan Jin Kevin Jönsson



CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026

TransFURmer: Hair and Fur rendering using SwinIR
Transformer-based Neural Rendering of Hair and Fur using SwinIR
Ryan Jin, Kevin Jönsson

© Ryan Jin, Kevin Jönsson, 2026.

Supervisor: Ulf Assarsson, Department of Computer Science and Engineering
Examiner: Erik Sintorn, Department of Computer Science and Engineering

Master's Thesis 2026
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Render of a bunny using the final model described in Section 4.8.

Gothenburg, Sweden 2026

TransFURmer: Hair and Fur rendering using SwinIR
Transformer-based Neural Rendering of Hair and Fur using SwinIR
Ryan Jin, Kevin Jönsson
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

While ray tracing algorithms offer photorealistic rendering of hair and fur, their high computational cost makes them unfeasible for real-time applications. Conversely, fast rendering techniques such as rasterization fail to capture accurate lighting effects and high-frequency details. This thesis investigates the feasibility of a Transformer-based pipeline tailored for rendering hair and fur. A SwinIR-based architecture is adapted within a Conditional Generative Adversarial Network (CGAN) framework to translate input buffers into high-quality renders. Trained on the SyntheticFur dataset, the model utilizes a combination of losses, notably Frequency Domain (FFT) loss, Learned Perceptual Image Patch Similarity (LPIPS) loss, and Wasserstein GAN with gradient penalty (WGAN-GP) loss, to accurately reconstruct the fine details and improve perceptual realism. The final model successfully generalizes to unseen geometry and the implemented FFT and LPIPS loss proved highly effective at preserving the texture of individual hair strands. However, the current model yields an average throughput of 0.13 fps, failing to satisfy real-time latency constraints. Additionally, hardware limits restricted the training crop size, and the training dataset was limited to simple primitives lacking complex self-shadowing scenarios. This likely contributed to shading inconsistencies and visual artifacts observed in the final renders. Despite these limitations, the final model is a faster alternative to path tracing for offline rendering tasks, and demonstrates the potential viability of Transformer-based neural rendering for complex hair and fur.

Keywords: computer, science, computer science, swin, transformer, swinir, artificial intelligence, ai, machine learning, graphics.

Acknowledgements

We would like to express our sincere gratitude to our supervisor, Ulf Assarsson, and our examiner, Erik Sintorn, for believing in the project from its start and for taking on the responsibility of supervising and examining this thesis. Their willingness to support the project, together with the valuable guidance, feedback, and encouragement they provided throughout the project, have been instrumental in shaping the quality of this work.

We also thank the Department of Computer Science and Engineering at Chalmers University of Technology for providing the resources necessary to carry out this research.

Ryan Jin and Kevin Jönsson, Gothenburg, 2026-07-28

Contents

List of Figures	xi
List of Tables	xiii
1 Introduction	1
1.1 Background	1
1.1.1 Neural Radiance Field	1
1.1.2 RenderFormer	2
1.1.3 Hair and Fur Rendering	2
1.2 Problem Formulation	2
1.3 Aim	3
1.4 Limitations	3
1.5 Ethical Aspects	3
1.5.1 Misuse of realism in synthetic images	4
1.5.2 Environmental impact	4
1.5.3 Dataset bias	4
1.5.4 Reproducibility	5
2 Theory	7
2.1 Rendering	7
2.1.1 Scene description	7
2.1.2 Rasterization	8
2.1.3 Ray Tracing	9
2.2 Neural Networks	10
2.3 Training and Loss	11
2.4 Convolutional Neural Networks	12
2.5 Transformer	13
2.6 Swin-Transformer	14
2.7 Generative Adversarial Networks	15
3 Methods	17
3.1 Tools	17
3.1.1 PyTorch	17
3.1.2 Minerva Compute Cluster	17
3.1.3 Apptainer	18
3.2 Dataset	18

3.3	Data Preprocessing	20
3.4	Data augmentation	20
3.5	Model Architecture	21
3.5.1	SwinIR	21
3.5.2	GAN	23
3.6	Loss Functions	23
3.6.1	L1 Loss	24
3.6.2	Frequency Domain Loss	24
3.6.3	Gradient Difference Loss	24
3.6.4	VGG Perceptual Loss	25
3.6.5	SSIM	25
3.6.6	GAN Loss	26
3.6.7	WGAN-GP Loss	27
3.6.8	LPIPS Loss	27
3.7	Training Procedure	28
3.8	Evaluation	29
3.8.1	Qualitative	29
3.8.2	Quantitative	30
4	Results	31
4.1	Rasterized Baseline	31
4.2	L1 Loss	32
4.3	Frequency Domain Loss	33
4.4	GDL Loss	33
4.5	VGG Perceptual Loss	35
4.6	GAN Loss	35
4.7	Effects of Rasterized Buffer	37
4.8	Final Model	38
4.9	Runtime Performance	38
5	Discussion	41
5.1	Positive Findings	41
5.2	Limitations	42
5.3	Application	43
5.4	Future Work	43
6	Conclusion	45
	Bibliography	47
A	Additional Rendered Images	I
A.1	Effects of rasterized input buffer	I

List of Figures

2.1	Example of a triangle mesh representing a dolphin. [12]. Public Domain.	7
2.2	First rasterization step is projecting triangles 3D vertices onto a 2D screen. Second step is to fill the pixels that lie within the resulting triangle with the color of the triangle. [14], CC BY-NC-ND 4.0	8
2.3	When two triangles overlap the same pixel on the screen, the color of the pixel is set to the triangle closest to the camera. [15], CC BY-NC-ND 4.0	9
2.4	Illustration of the ray tracing algorithm for rendering an image. [18], CC BY-SA 4.0	10
2.5	A visual demonstration of a 3×3 kernel convolving over a zero-padded 4×4 input matrix with a stride of 1 to generate a 4×4 output activation map.	12
2.6	The Swin Transformer Layer (STL) consists of a variant of an MSA module (W-MSA or SW-MSA) followed by an MLP module. Each module is preceded by Layer normalization and residual connections are employed to bypass each module.	15
2.7	Architecture of a Conditional CGAN. The additional data is provided as an additional input to both the generator and the discriminator. .	16
3.1	Ground truth renders from the Syntheticfur dataset showcasing the different primitives that were used during training of the model. [33], CC BY-NC-SA 4.0 license.	19
3.2	Scene sample from the SyntheticFur dataset. Figure reproduced from Le et al. [33], CC BY-NC-SA 4.0 license.	19
3.3	An overview of a Residual Swin Transformer Block (RSTB) used in the SwinIR model [34].	22
3.4	An overview of the final model with the added convolutional layer at the end of the base SwinIR model.	22
3.5	Architectural overview of the implemented CGAN framework. The architecture consists of a CGAN setup without a noise prior where the SwinIR generator uses the input buffers to produce generated images. The PatchGAN Discriminator then evaluates generated images and ground truth images, using the original input buffers as context. . . .	23

4.1	Qualitative comparison for the rasterized baseline and ground-truth ray-traced rendering.	32
4.2	Graph of L1 loss over Epoch during training	32
4.3	Qualitative comparison between the model output using only L1 loss and the ground-truth ray-traced rendering (Exp01 in Table 4.1). . . .	33
4.4	Qualitative comparison between the generated output using FFT loss and the ground-truth ray-traced rendering (Exp03 in Table 4.1). . . .	34
4.5	Qualitative comparison between the generated output using GDL loss and the ground-truth ray-traced rendering (Exp05 in Table 4.1). . . .	34
4.6	Qualitative comparison between the generated output using FFT + GDL loss and the ground-truth ray-traced rendering (Exp07 in Table 4.1).	35
4.7	Qualitative comparison between the generated output using VGG loss and the ground-truth ray-traced rendering (Exp09 in Table 4.1). . . .	36
4.8	Results from standard GAN loss and WGAN loss. From left to right, Exp10, Exp11, and ground-truth.	36
4.9	Effects of the rasterized buffer for model with simpler loss function. From left to right, Exp01, Exp02, and the ground-truth, see Table 4.1.	37
4.10	Effects of the rasterized buffer for model with more complex loss function. From left to right, Exp11, Exp12 and the ground-truth, see Table 4.1.	37
4.11	Graph of train and validation loss over Epoch during training	39
4.12	Graph of L1, FFT, and LPIPS loss over Epoch during training	39
4.13	Graph of Generator and Critic loss over Epoch during training	40
4.14	Qualitative comparison between the generated output using the final model and the ground-truth ray-traced rendering (Exp13 in Table 4.1).	40
A.1	Effects of the rasterized buffer for model with only FFT loss. From left to right, Exp03, Exp04 and the ground-truth, see Table 4.1. . . .	I
A.2	Effects of the rasterized buffer for model with only GDL loss. From left to right, Exp05, Exp06 and the ground-truth, see Table 4.1. . . .	I
A.3	Effects of the rasterized buffer for model with only GDL loss. From left to right, Exp05, Exp06 and the ground-truth, see Table 4.1. . . .	II
A.4	Twelve frames sampled from the fully rendered <code>bunny_3Lights_rotateUp</code> scene using Exp13 from Table 4.1, with one frame shown every 30 frames. The second frame exhibits noticeable visual artifacts in the form of black spots.	III
A.5	Twelve frames sampled from the fully rendered <code>sphere_rimLight_static</code> scene using Exp13 from Table 4.1, with one frame shown every 30 frames	IV
A.6	Twelve frames sampled from the fully rendered <code>torus_3Lights_rotateRightForward</code> scene using Exp13 from Table 4.1, with one frame shown every 30 frames	V

List of Tables

3.1	Overview of the conducted experiments and weighting coefficients. All experiments used L1 loss in combination with the additional losses listed in the table. FFT denotes Frequency Domain Loss and GDL denotes Gradient Difference Loss.	29
4.1	Overview of the conducted experiments and their highest achieved SSIM values. All experiments used L1 loss in combination with the additional losses listed in the table. FFT denotes Frequency Domain Loss and GDL denotes Gradient Difference Loss.	31

1

Introduction

Accurately rendered hair strands in human hair and fur are commonly utilized in both video games and the film industry, where they play a crucial role in achieving believable characters and natural environments. At its core, rendering is the process of creating an image from a 3D scene description. The scene description contains information such as geometric data, camera viewpoints, material properties, and light sources.

Among the various rendering methods, rasterization is widely used. Due to its fast rendering speed it is typically used in real-time applications, but at the cost of less accurate lighting effects from approximations.

To achieve photorealistic renders, ray tracing is typically used. This rendering method aims to simulate how light travels and interacts with the scene, resulting in more realistic renders, but at the cost of being very computationally intensive [1]. This computational cost increases drastically when processing the massive amount of geometry and complex light interactions required for hair and fur.

This master thesis aims to address these limitations by exploring a deep learning alternative to traditional rendering methods, specifically focused on the complex geometry and light interactions of hair and fur [2], [3].

1.1 Background

One approach to rendering is to approximate components of the rendering pipeline or the entire pipeline using neural networks, a direction broadly referred to as neural rendering [4]. This is also the direction NVIDIA, one of the largest companies in the graphics industry, is moving towards [5].

1.1.1 Neural Radiance Field

Neural Radiance Field (NeRF) is a method for learning a continuous representation of a 3D scene allowing it to render from any angle. The limitations of this approach are that it does not generalize well since the model has to be trained for each new scene [6].

1.1.2 RenderFormer

Another neural rendering method that shows promising results and generalizes well is RenderFormer, which is a Transformer-based neural renderer that learns a full rendering pipeline without requiring per-scene training [7]. RenderFormer tokenizes triangle-mesh-based scene descriptions and uses this as input for the network. The computational cost of Transformers is limited by the quadratic complexity of the attention mechanisms. Consequently, this limits the triangle count in triangle meshes.

1.1.3 Hair and Fur Rendering

Hair and fur rendering is a challenging problem in computer graphics due to the geometric and lighting complexity of individual hair strands. Traditional strand-based rendering methods often require simulating and rendering millions of hair fibers, which is computationally expensive. To improve efficiency, recent research has explored alternative geometric representations and physically based approximations.

Bhokare et al. [8] proposed Hair Meshes, a representation that enables efficient real-time rendering of highly detailed hair models. Similarly, Hsu et al. [9] introduced a physically guided interpolation framework that generates dense strand representations from a limited number of simulated guide hairs, while preserving physically plausible behavior.

Research has also focused on improving the realism of light transport in hair and fur. Earlier work by Sintorn and Assarsson [3] addressed challenges of self-shadowing and transparency ordering in hair rendering through occupancy maps, which enabled efficient approximation of light interactions in dense hair volumes. More recently, Curruis, Assarsson, and Sintorn [2] demonstrated that convolutional neural networks can be used for real-time hair filtering, producing visually convincing results while reducing computational cost.

1.2 Problem Formulation

While global illumination algorithms, like path tracing, offer accurate photorealistic hair rendering, their computational cost makes them unfeasible for real-time applications. Conversely, emerging neural rendering methods have potential for speed and generalization but currently face limitations when applied to hair.

The primary challenge lies in the geometric complexity of hair. Realistic hair requires highly dense geometric representations to capture individual strands and their light interactions with accuracy. However, Transformer-based methods like RenderFormer are computationally limited by the triangle count. Therefore, they cannot process the dense meshes required for high-fidelity hair.

The main problem addressed in this thesis is the poor scene generalization capabilities, the lack of support for high-density geometry, and expensive computational cost in neural rendering methods.

1.3 Aim

The primary objective of this thesis is to investigate the feasibility of a rendering pipeline with Transformer-based post-processing specifically tailored for hair and fur. To address the limitations identified in the problem section, the thesis focuses on three core objectives:

- Training a model capable of handling different scene configurations, camera angles, and lighting conditions without requiring per-scene training, overcoming the generalization constraints of methods like NeRF.
- To use the model in conjunction with a rasterizer for practical use in real time.
- To achieve high-quality visual results that closely approximate the ground-truth ray-traced images provided by the dataset.

1.4 Limitations

This thesis is subject to multiple limitations.

First, the model is trained on a fairly limited dataset. Although the dataset provides a variety in geometry, lighting conditions, and viewing angles, it does not capture the true diversity and complexity of real-world hair and fur. As a result, the proposed method should primarily be regarded as an exploratory proof-of-concept study, and its performance on unseen real world data remains uncertain.

Second, the proposed approach focuses specifically on hair and fur rendering and will not adequately address other materials and surface types. While this specialization allows for targeted improvements in a challenging domain, it limits the applicability of the method to more general rendering tasks involving different types of geometry and materials.

Third, the evaluation of the model primarily relies on comparisons between the generated results and ray-traced reference images. Hence, the assessment of visual quality is subjective and dependent on human perception. However, no user studies are conducted within the scope of this thesis to systematically assess how the results are perceived by observers.

Lastly, the hardware used is limited to the computing cluster available at Chalmers. While the experiments conducted on the computing cluster provide some indication of the model's practical performance, these results may not generalize across different hardware configurations. Variations in GPU and memory capacity may impact both training and inference.

1.5 Ethical Aspects

Advancements in rendering techniques have the potential to significantly impact both the computer graphics industry and society at large. Although the primary

focus of this thesis is technical, it is important to consider the ethical implications associated with such technology. These include the potential misuse of increasingly realistic synthetic imagery, the environmental impact of large-scale data generation and model training, representational limitations and bias in synthetic datasets, and challenges related to reproducibility and accessibility of research.

1.5.1 Misuse of realism in synthetic images

The rise of generative AI has led to a surge in generative media, including Deepfakes, which are increasingly utilized to spread disinformation. While the primary objective of this thesis is to optimize rendering of realistic hair and fur for applications in the entertainment industries, it is necessary to acknowledge the potential for dual-use. The improvements in realism achieved could be used by malicious actors to enhance Deepfakes, making them harder to distinguish from reality.

1.5.2 Environmental impact

The primary objective of this research is to accelerate the rendering process using a transformer-based model. Rendering path-traced frames for complex scenes is computationally intensive, often requiring minutes or hours per frame. This latency results in substantial electricity consumption. By reducing the inference time required to render a frame, this project aims to lower the aggregate energy consumption of the rendering pipeline.

However, it is necessary to address the environmental cost of developing such solutions. Deep learning models, particularly Transformers, require significant computational resources during the training phase. Strubell et al. [10] highlight the carbon footprint of AI, noting that the *neural architecture search* required to optimize large models can emit nearly five times the lifetime carbon of an average car. While the training of the specific model used in this thesis is less intensive than a full architecture search, the energy cost remains non-negligible.

However, unlike the training phase, which is a one-time expenditure, the energy savings from accelerated rendering accumulate with every frame generated. Therefore, if the model is widely adopted or used for high-volume rendering tasks, the long-term reduction in power usage could outweigh the initial training cost, resulting in a net positive environmental impact.

1.5.3 Dataset bias

Synthetic datasets are limited by assumptions made by the creators. Consequently, a model trained on such data could suffer from poor generalization to content in the real world. The lack of diversity in the training data can lead to poor rendering quality for underrepresented demographics. In this thesis there is therefore a risk of high-quality rendering only being accessible for a narrow set of hair features.

1.5.4 Reproducibility

The computational requirements for training Transformer-based models may limit reproducibility for researchers without access to high-end GPU resources. This raises concerns regarding accessibility, as reproducing or extending the proposed method may not be feasible in resource-constrained environments. Although this limitation cannot be fully addressed within the scope of this thesis, transparency in reporting model architecture, training procedures, and evaluation metrics is emphasized to enable comparison and partial reproducibility.

2

Theory

This chapter presents the theoretical background necessary to understand the methodologies applied in this thesis. The first section introduces the basic components of rendering, rasterization for generating the input buffers, and ray tracing for producing ground-truth images. The latter part of the chapter transitions into an overview of foundational deep learning principles, Transformer architectures, and lastly Generative Adversarial Networks.

2.1 Rendering

The main goal of rendering is to create a 2D image from a 3D scene description. In this section, the basic components of rendering and two rendering techniques will be introduced.

2.1.1 Scene description

Within computer graphics, a 3D scene defines an environment mapped by a Cartesian coordinate system, where objects are positioned using coordinates along the x, y, and z-axes. The basic components of a scene description consist of geometry, lights, and a camera.

The geometry describes one or multiple 3D objects in the scene and is commonly converted to a triangle mesh. Describing all geometry as triangles standardizes rendering to a single coplanar primitive [11]. This uniformity means that advanced rendering features only need to be programmed once and allows rendering operations to be built directly into GPU hardware for maximum performance and efficiency.

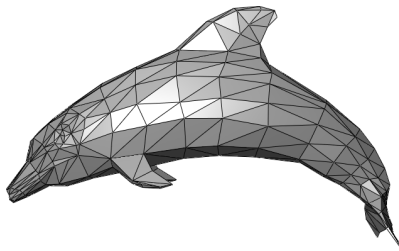


Figure 2.1: Example of a triangle mesh representing a dolphin. [12]. Public Domain.

In order for a renderer to calculate shading, shadows, and light interactions with the geometry surface, light sources are needed in the scene. Without the inclusion of lights, no illumination can be calculated, and the resulting image would render entirely black.

Finally, the third essential component of a scene description is the camera. The camera's position and orientation within the Cartesian coordinate system dictate which parts of the geometry are visible on the screen.

2.1.2 Rasterization

Rasterization is a rendering technique that focuses on determining which parts of 3D objects are visible to the camera. The basic process of rasterization can be decomposed into two steps [13]. The first step is to project 3D vertices of the triangles that make up the 3D objects in the scene onto a 2D screen using perspective projection. Step two fills all the pixels that are within the projected triangle. This is done for all triangles within the cameras view.

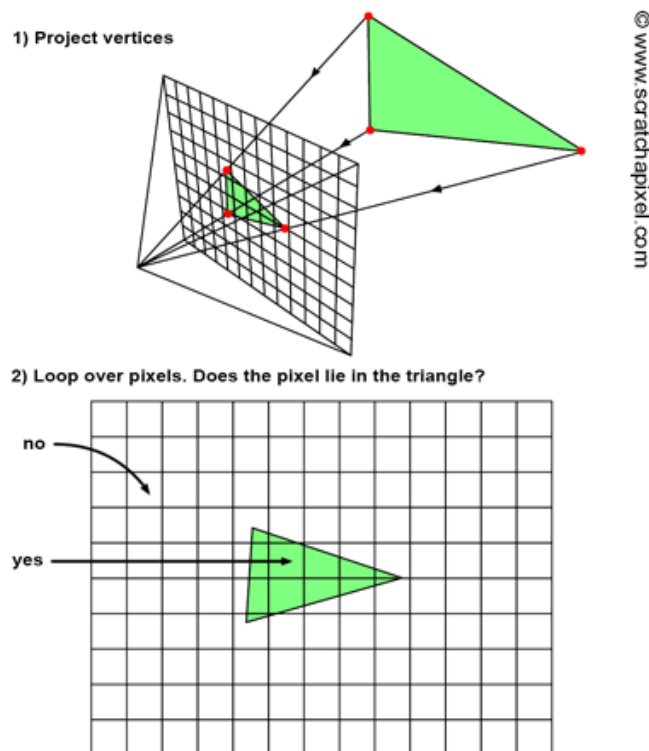


Figure 2.2: First rasterization step is projecting triangles 3D vertices onto a 2D screen. Second step is to fill the pixels that lie within the resulting triangle with the color of the triangle. [14], CC BY-NC-ND 4.0

To solve the problem where objects overlap each other, see Figure 2.3, a depth buffer is introduced [13]. The depth buffer is a 2D-matrix with the same dimension as the rendered image and records the smallest distance between the camera and the 3d objects for each pixel. The elements of the depth buffer are initialized with a large number. Then, during rasterization, the distance between the camera and the part

of the 3D object that overlaps a pixel is compared against the corresponding pixel in the depth buffer. If the newly calculated distance is smaller, it indicates that this 3D object is closer to the camera and should be visible. The value in the depth buffer is then overwritten with the new distance, and the projected pixel's color is drawn onto the screen.

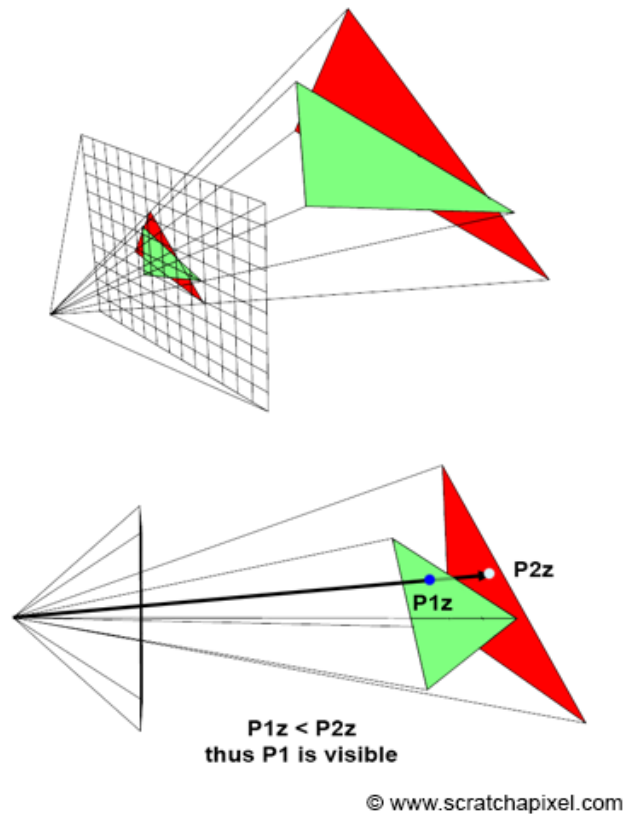


Figure 2.3: When two triangles overlap the same pixel on the screen, the color of the pixel is set to the triangle closest to the camera. [15], CC BY-NC-ND 4.0

It is critical to emphasize that the rasterization algorithm is strictly limited to solving the visibility problem and does not inherently compute any lighting, shadows, or complex optical phenomena [16].

2.1.3 Ray Tracing

Ray tracing is a rendering technique designed to realistically simulate the physical behavior of light within a three-dimensional environment. The algorithm fundamentally relies on ray casting, a process that traces light paths backward from the camera, through the image plane's pixels, and into the scene until they intersect with light sources [1], as depicted in Figure 2.4. Upon intersecting an object, a ray may be reflected, absorbed, or refracted. To govern how these rays scatter at surface boundaries, mathematical models known as Bidirectional Scattering Distribution Functions (BSDFs) are employed [17].

A BSDF evaluates the probability distribution of a ray scattering in a given direction

after striking a material. The final color of a given pixel is thus calculated by accumulating the shading data from the ray's primary intersection, along with the subsequent contributions gathered by the scattered secondary rays.

This process produces realistic lighting, even for complex objects such as hair. However, due to requiring the evaluation of a large number of paths for each pixel to get realistic results, this method is computationally very expensive and can require seconds to hours per frame depending on the complexity of the scene.

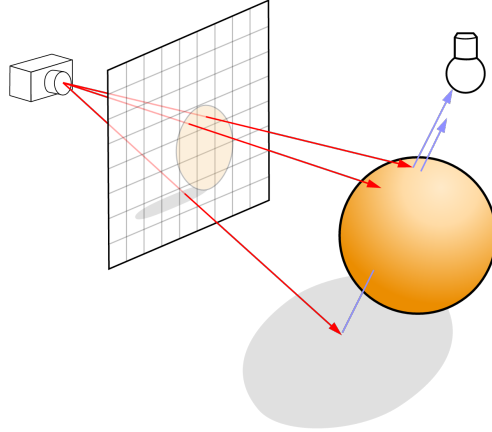


Figure 2.4: Illustration of the ray tracing algorithm for rendering an image. [18], CC BY-SA 4.0

2.2 Neural Networks

A neural network is a class of machine learning models inspired by the structure of the human brain. It can be trained to approximate complex functions by mapping input data to output predictions through a series of nonlinear transformations. Neural networks consist of interconnected computational units, referred to as neurons, which are organized into layers that pass information between each other.

The core concept of a neuron is based on the perceptron algorithm, originally proposed by *Frank Rosenblatt* in 1958 [19]. A neuron computes a weighted sum of its inputs, adds a bias term, and applies a nonlinear activation function. The weights determine the relative importance of each input feature, while the bias allows the neuron to shift the activation function. Formally, a neuron computes

$$y = \sigma \left(\sum_i w_i x_i + b \right), \quad (2.1)$$

where x_i are the inputs, w_i the weights, b the bias, σ the activation function and y the output.

A single neuron can represent linearly separable functions, and combining multiple neurons into layered architectures enables modeling of more complex nonlinear relationships [20].

In its simplest form, a neural network consists of an input layer and an output layer. Neural networks that include additional intermediate layers between input and output are called deep neural networks, where these intermediate layers are referred to as hidden layers. The presence of multiple hidden layers enables the model to learn hierarchical representations of data, where lower levels typically capture simple features, and deeper layers capture more abstract and high-level patterns [21].

Training a neural network involves optimizing its parameters, namely the weights and biases of the neurons, to minimize a predefined loss function.

2.3 Training and Loss

A loss function, also referred to as a cost function, quantifies the discrepancy between a model's prediction and the corresponding ground-truth [20]. There exists a wide variety of loss functions, and in many practical applications multiple loss terms are combined into a single objective by summation. This allows different aspects of the prediction quality to be optimized simultaneously.

A commonly used loss function is the L1 loss, also known as mean absolute error (MAE), which measures the average absolute difference between the prediction and the ground truth expressed as

$$\mathcal{L}_{L1}(x, y) = \frac{1}{N} \sum_{n=1}^N |x_n - y_n|, \quad (2.2)$$

where x denotes the prediction, y the ground truth, and N the total number of elements in the tensors.

Minimizing the loss function corresponds to improving the model's predictions. This is typically achieved using gradient-based optimization methods, most commonly Stochastic Gradient Descent (SGD)[21]. The key idea is to iteratively update the model parameters in the direction that reduces the loss.

A simplified description of the SGD algorithm explained by *Goodfellow et al.* [21] is as follows:

- Sample a mini-batch of size m from the dataset M .
- For each sample, compute the model prediction x and evaluate the loss function $L(x, y, \theta)$, where y is the ground truth and θ the model parameters.
- Compute an estimated gradient g of the loss with respect to the parameters, as shown in equation 2.3.
- Update the parameters using the gradient:
 $\theta = \theta - \epsilon g$ where ϵ is the learning rate.
- Repeat the process until convergence or satisfactory performance is achieved.

$$g = \frac{1}{m} \nabla_{\theta} \sum_{i=1}^m L(x^{(i)}, y^{(i)}, \theta), \quad (2.3)$$

In practice, the gradient $\nabla_{\theta} L$ is computed efficiently using the backpropagation algorithm, which applies the chain rule to propagate errors through the network. This enables the model to update all weights and biases based on their contribution to the final loss.

2.4 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) process image data using local connectivity, where each neuron connects only to a small, specific area of the input called its receptive field [22]. The core computational unit is the convolutional layer, which applies learnable filters, or kernels, across the spatial dimensions of the input. A CNN learns these specific kernels that produce high activation when they recognize specific visual features. As a kernel glides across the input, the network computes the scalar product between the kernel weights and the input region to produce a two-dimensional activation map. Figure 2.5 shows a demonstration of 3×3 convolution over a 4×4 input matrix.

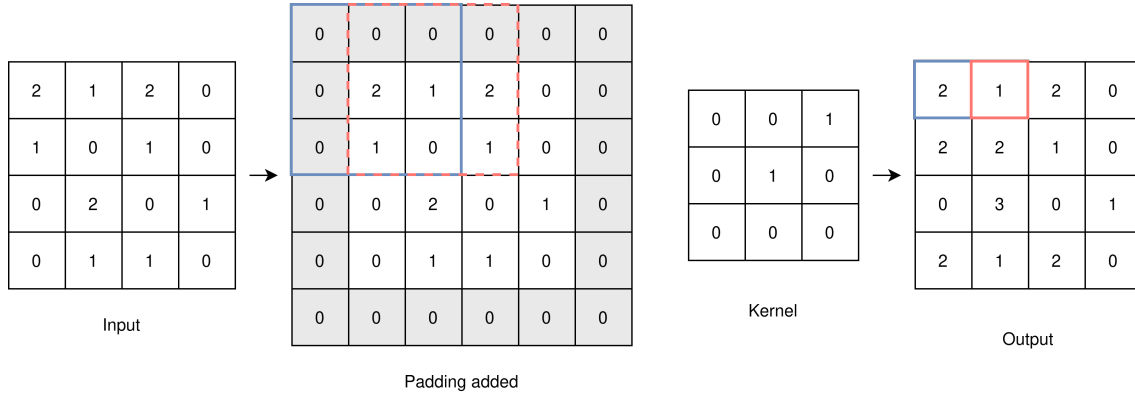


Figure 2.5: A visual demonstration of a 3×3 kernel convolving over a zero-padded 4×4 input matrix with a stride of 1 to generate a 4×4 output activation map.

The activation maps from the applied kernels are then stacked to form the final output of the layer.

Convolutional layers can control its final output through three key hyperparameters:

- *Depth*: Represents the number of activation maps that get stacked together or the number of learnable kernels that are applied.
- *Stride*: Determines the step size for when the kernel glides across the input. Setting a larger stride reduces the overlap and produces an output of lower spatial dimensionality.

- *Zero padding*: Adds a padding of zeros to the border of the input. As demonstrated in Figure 1, this allows the kernel to sweep over edge pixels, preventing the spatial size of the input from shrinking after the convolution operation.

Beyond these hyperparameters, CNNs further optimize computational complexity through parameter sharing by applying identical kernels across the entire image. This process allows CNNs to efficiently extract visual features while reducing the number of required parameters compared to traditional networks.

Despite their effectiveness, CNNs primarily model local spatial relationships and often require many layers to capture long-range dependencies. This limitation motivated the development of attention-based architectures, such as the Transformer, which can model global dependencies.

2.5 Transformer

The Transformer architecture, originally introduced for Natural Language Processing by Vaswani et al. [23], is based on the self-attention mechanism, which enables the model to capture global dependencies in data. In its original formulation, the input consists of a sequence of tokens, such as words in a sentence. In contrast to traditional convolutional or recurrent architectures, Transformers process input sequences in parallel and explicitly model interactions between all tokens, regardless of their relative position in the input.

In the context of rendering, such global dependencies are important for approximating light transport phenomena, such as shadows, reflections, and indirect illumination between distant hair strands. Capturing these interactions is challenging for local methods, but well-suited for attention-based architectures.

To apply the Transformer architecture to images, Dosovitskiy et al. [24] proposed the Vision Transformer (ViT). In this approach, an input image is divided into fixed-size patches (e.g. 16×16 pixels), which are flattened, linearly embedded and then treated as a sequence of tokens, similarly to words in a sentence.

The core computation of the Transformer is the self-attention mechanism, which models relationships between all pairs of tokens. For each token, three vectors are derived through linear projections: a query (Q), a key (K), and a value (V). The self-attention mechanism determines how much each token attends to every other token by comparing queries and keys, as shown in Equation 2.4

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V, \quad (2.4)$$

where d_k is the dimensionality of the key vectors. The scaling factor $\sqrt{d_k}$ is introduced to stabilize training by preventing excessively large values in the dot-product.

In practice, the Transformer architecture does not rely on a single self-attention function. Instead, the attention mechanism is computed h times in parallel, a process

known as Multi-Head Self-Attention (MSA). For each parallel computation, referred to as an attention head, the network initializes a different set of learnable projection matrices to produce its own set of queries, keys, and values. The individual outputs from all h parallel heads are then concatenated together and linearly projected to the same dimension as the passed in token.

A limitation of the standard self-attention mechanism is its quadratic computational complexity $O(N^2)$ with respect to the number of tokens. For image data, this implies that increasing the number of patches significantly increases computational cost, which poses challenges for high-resolution rendering tasks and dense geometric representations, such as hair, where a large number of tokens is required. To address this limitation, more efficient attention mechanisms, such as window-based approaches, have been developed.

2.6 Swin-Transformer

To reduce the computational complexity of self-attention, Liu et al.[25] introduced the Swin Transformer. Instead of computing attention globally across all tokens, the Swin Transformer restricts self-attention to non-overlapping local windows, reducing the complexity to linear.

However, limiting attention restricts the model’s ability to capture global context. To mitigate this, the Swin Transformer employs a shifted window mechanism between consecutive layers. In this approach, the window partitioning is shifted such that tokens that were previously located in different windows are grouped together in subsequent layers. This enables information to propagate across window boundaries, allowing the model to capture both local and global dependencies efficiently.

Internally, the self-attention computations are processed within the Swin Transformer Layer (STL). Structurally, an STL consists of a variant of the MSA module followed by a Multi-Layer Perceptron (MLP). The MLP consists of two fully-connected layers with the Gaussian Error Linear Unit (GELU) activation function [26] where the first layer expands the tokens feature space and the second layer compresses feature space back to the tokens original dimension. Expanding the feature space allows the network to more easily learn non-linear visual patterns. To stabilize the learning process, Layer Normalization is applied immediately before both the MSA and MLP modules. Layer normalization subtracts the mean of the channel dimension for each feature value and then divides that result by the standard deviation of the channel dimension. Additionally, residual connections are employed to bypass each module, see Figure 2.6.

The first variant of MSA utilized in an STL is Window Multi-head Self-attention (W-MSA). Instead of computing relationships across the entire image simultaneously, W-MSA partitions the input into non-overlapping local windows of a fixed size. Self-attention is computed separately within the bounds of each individual window and the computational complexity therefore scales linearly with respect to the image size.

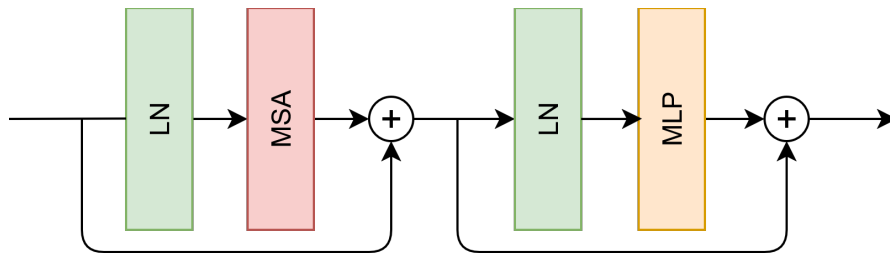


Figure 2.6: The Swin Transformer Layer (STL) consists of a variant of an MSA module (W-MSA or SW-MSA) followed by an MLP module. Each module is preceded by Layer normalization and residual connections are employed to bypass each module.

However, because the windows are isolated from each other, W-MSA lacks the ability to connect features across the entire image. To resolve this limitation, the second variant of MSA, Shifted Window Multi-head Self-Attention (SW-MSA) was introduced. The SW-MSA configuration shifts the window partitioning by half a window size relative to the regular W-MSA grid. This shift creates a new set of windows that physically cross the previous boundaries, establishing cross-window communication while preserving the linear computational complexity.

2.7 Generative Adversarial Networks

Generative Adversarial Networks (GANs) are a machine learning framework introduced by Goodfellow et al. [27], where two neural networks, a generator G and a discriminator D , are trained simultaneously. The generator's role is to create realistic samples by passing random noise through the network to map it to the target data space, while the discriminator's goal is to distinguish whether a given sample originates from the true data space or the synthetic outputs of the generator. As the authors mention, G and D compete against each other similar to how counterfeiters try to produce fake currency that can successfully fool the police.

When D is trained to its optimum, the objective of the generator is equivalent to minimizing the Jensen-Shannon (JS) divergence between the real and the generated data distributions [27]. However, optimizing the JS divergence can cause vanishing gradients if the underlying distribution of the real and generated data does not overlap which halts the generator's learning [28].

To address the instability of JS divergence, Arjovsky et al. [28] proposed the Wasserstein GAN (WGAN) that changes the objective to minimizing the Wasserstein distance between the real and generated distributions. Unlike JS divergence, the authors found that the Wasserstein distance provides meaningful and non-vanishing gradients even when the distributions are entirely disjoint. Computing this distance benefits from a discriminator, now called a critic, that approximately satisfies a 1-Lipschitz constraint [28]. A differentiable function is considered 1-Lipschitz if its gradient norm is at most one everywhere. One way to encourage this constraint, introduced by Gulrajani et al. [29], is to penalize the norm of the critic's gradient

with respect to its input, encouraging it to stay close to one. This is referred to as WGAN with gradient penalty (WGAN-GP) and significantly improves the stability of WGAN training in practice.

While standard GANs learn to generate realistic samples, they offer no control on modes of the data being generated. To address this, Conditional Generative Adversarial Networks (CGANs) extends standard GANs by conditioning the generator and discriminator on additional information such as class labels or data from other modalities, allowing the model to direct the data generation process [30]. Figure 2.7 illustrates the structure of a CGAN, demonstrating how the additional information is integrated into both the generator and the discriminator as an additional input.

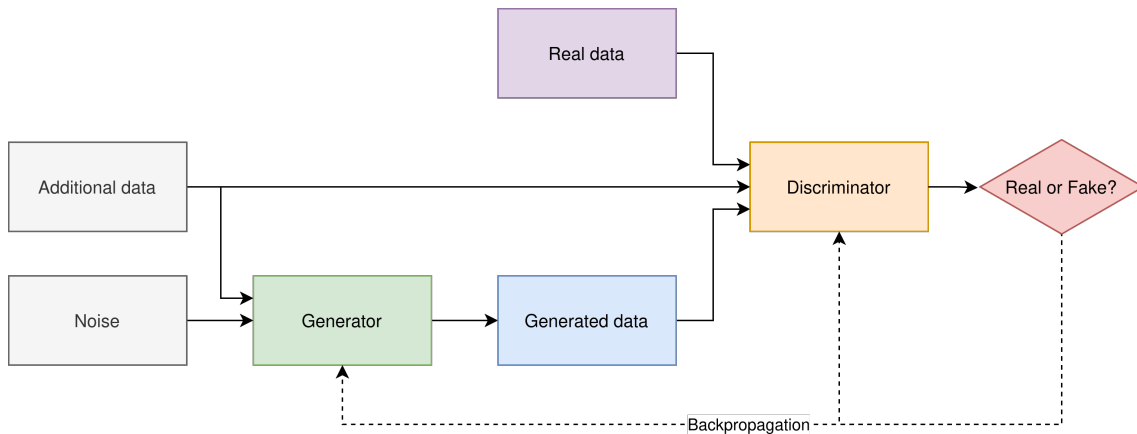


Figure 2.7: Architecture of a Conditional CGAN. The additional data is provided as an additional input to both the generator and the discriminator.

When employing CGANs for image-to-image translation related tasks, restricting the discriminator to the structure in local image patches is sufficient to model high-frequency details [31]. This architecture, known as PatchGAN, classifies each individual $N \times N$ patch in an image as real or fake rather than evaluating the entire image as a whole. The discriminator is run convolutionally across the image, and all responses are averaged to provide the final output. The patch size N can be much smaller than the full image size while still producing good results, which is advantageous because it requires fewer parameters, runs faster, and can be applied to arbitrarily large images.

3

Methods

3.1 Tools

This section outlines the software library, hardware, and environment management tools used to implement, train, and replicate the models. It introduces the PyTorch library used for developing deep learning models, the Minerva high-performance compute cluster that handles the heavy computational workloads, and the App-tainer containerization system used to ensure strict reproducibility across different computing environments.

3.1.1 PyTorch

PyTorch is an open-source Python machine learning library developed by Paszke et al. [32] used to implement, train, and evaluate neural network models. PyTorch provides a flexible programming interface for model development and debugging while maintaining compatibility with other popular scientific computing libraries. It also provides hardware acceleration, such as GPU acceleration through CUDA.

In this thesis, the PyTorch library was used for the model architectures SwinIR and PatchGAN, which are explained more thoroughly in Section 3.5. It was also used to define loss functions, handle data loading, data augmentation, and model optimization. Additionally, the library provides support for mixed-precision training and learning-rate scheduling, which were utilized to improve computational efficiency and training stability.

3.1.2 Minerva Compute Cluster

Training transformer models requires significant hardware acceleration. Therefore all computational workloads were executed on Minerva, a high-performance compute cluster hosted by the Computer Science and Engineering (CSE) department at Chalmers University of Technology and the University of Gothenburg. In this project the compute nodes used where *io*, *europa*, *ganymede*, and *callisto*. These three nodes share the hardware specifications below

- *Processors*: Dual INTEL XEON GOLD 6548N processors (2×64 cores).
- *Memory*: 512 GiB of RAM.

- *GPUs*: $8 \times$ NVIDIA L4 GPUs, with 24 GiB VRAM and approximately 7,500 CUDA cores for each GPU.
- *Storage*: $2 \times$ 1.92 TB SSDs.

Since the PyTorch library cannot automatically distribute operations across multiple GPUs without specific code modifications. The training workloads were thereby configured to utilize only one GPU per job.

To schedule jobs on the cluster Minerva uses the SLURM workload manager. Since training a transformer model is a highly time-intensive process, all batch jobs were submitted to the cluster’s *long* queue partition which extends the execution time limit to one full day.

3.1.3 Apptainer

To ensure strict reproducibility of the software environment and to keep track of all the dependencies required, the environment was managed using Apptainer containers. The Minerva cluster provides native support for Apptainer, which enables execution of containerized workflows directly on the compute nodes. By packaging the dependencies into an isolated container, a consistent environment could be used on the compute nodes to run the training and inference.

3.2 Dataset

Neural rendering methods require substantial amounts of high-quality data to perform well. To address this, Le et al. created the SyntheticFur dataset [33], a procedurally generated dataset specifically built for machine learning. It has the advantage of being cleaner to process, with less feature engineering involved compared to working with a generic photographic dataset.

The full dataset comprises of roughly 140,000 images generated across 15 simulations using the Houdini software [33]. Each image is rendered at a resolution of 1024×1024 pixels and saved in PNG format. The dataset is organized into different scenes, where each scene represents a sequence of frames capturing the continuous motion of fur. For this project, the data used for training and validation consists of all scenes containing ground truth images. The base primitives used were *bunny*, *sphere*, *torus*, and *tube*, shown in Figure 3.1. These scenes incorporate various primitive motions under predefined lighting conditions, such as moving the primitive from left to right.

For each frame, the dataset provides a high-quality render that is used as the ground truth image and several corresponding input image buffers. These images are shown in Figure 3.2, and are categorized as follows:

- (a) *Ground truth*: Target images rendered using ray tracing with the Mantra rendering engine.
- (b) *Colored guides*: A buffer representing the underlying procedural guide curves that dictate the flow and structural layout of the fur strands.

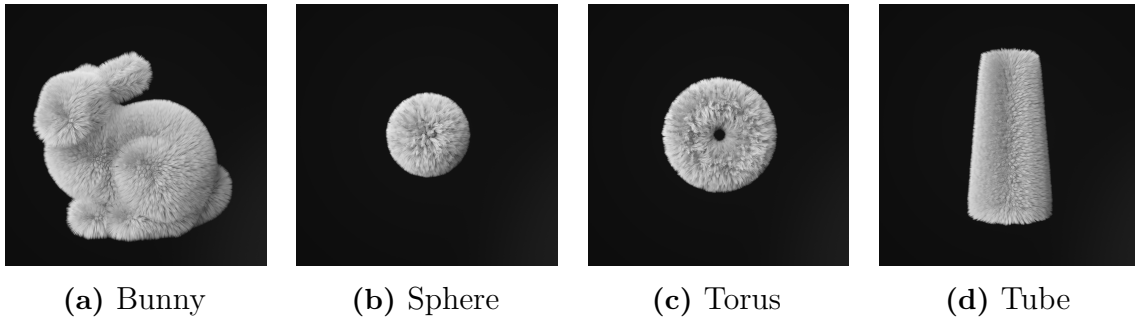


Figure 3.1: Ground truth renders from the SyntheticFur dataset showcasing the different primitives that were used during training of the model. [33], CC BY-NC-SA 4.0 license.

- (c) *Lit primitive*: Displays the diffuse shading of the underlying primitive with lighting, but without the fur.
- (d) *Scene depth*: Indicates normalized scene depth, mapping the closest and furthest points of the geometry from the camera into a $[0.0, 1.0]$ grayscale range where darker areas are closer to the camera and lighter areas are further away.
- (e) *World normal*: Displays the orientation of the surface normal for the primitive without fur at every pixel.
- (f) *Mask*: Displays a silhouette of the primitive without fur in the scene.
- (g) *Rasterized*: A render of the primitive with fur using rasterization with constant brightness.

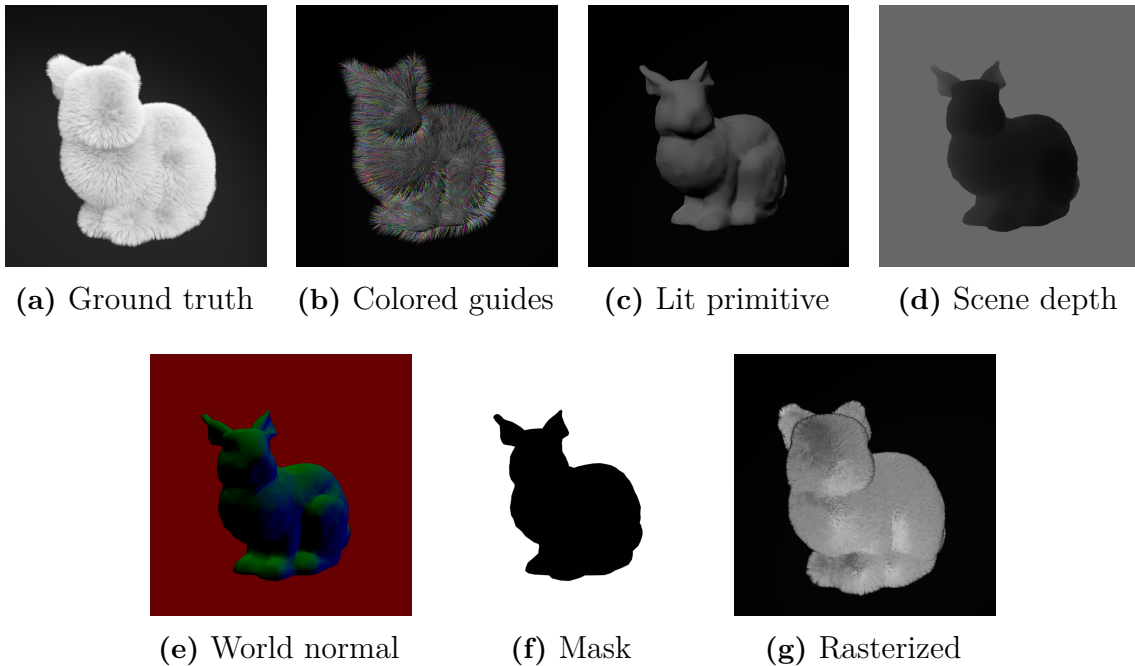


Figure 3.2: Scene sample from the SyntheticFur dataset. Figure reproduced from Le et al. [33], CC BY-NC-SA 4.0 license.

3.3 Data Preprocessing

Before training, several preprocessing steps were applied to the SyntheticFur [33] dataset to ensure data consistency and compatibility with the proposed pipeline.

During the initial data exploration, a mismatch in rotation angles was discovered between the ground-truth images and the corresponding input buffers for each frame in the `bunny_3Lights_rotateUp` scene. This mismatch occurred because the ground-truth images had been rendered at double the frame rate compared to the input buffers.

To resolve this issue, the data for this specific scene was manually corrected by removing every second image from the ground-truth buffer and discarding the second half of all the frames for all input buffers. The resulting scene was thereby reduced to 360 frames.

Several scenes in the SyntheticFur dataset did not contain rasterized input buffers, including `sphere_3Lights_sbPlateRL_static`, `sphere_hdriCapeHill_static`, and `bunny_curly_clumpLarge_brown_3Lights_static`. Consequently, these scenes were excluded from the experiments that required the rasterized buffer.

To evaluate the model’s ability to generalize across different geometric shapes, the dataset was split into separate training, validation, and test sets based on the underlying primitive geometry. All scenes containing the *sphere*, *torus* and one of the *tube* shape scenes, `tube_3Lights_rotateRight`, were used in the training split. The other *tube* scene, `tube_3Lights_static`, was used in the validation split. Finally, all *bunny* scenes were used exclusively for testing. This split was chosen to evaluate the model’s ability to generalize to unseen geometry during inference. Since the *bunny* geometry differs significantly from the other primitive shapes, it provides a suitable benchmark for evaluating geometric generalization.

For each sample, the selected input buffers were loaded and converted into tensors before being concatenated along the channel dimension. The input buffers consisted of colored guides, lit primitive, scene depth, world normal, mask, and rasterized. The corresponding ground-truth ray-traced image was used as the target output during training.

3.4 Data augmentation

Data augmentation is a commonly used technique to generate additional training samples from existing data by altering them without changing the fundamental properties of the data. The purpose of data augmentation is to improve the model’s ability to generalize by increasing the diversity of the training data and reducing overfitting. This is particularly important in computer vision tasks, as collecting large and diverse datasets can be difficult and computationally expensive.

To improve generalization, augmentations were applied to the SyntheticFur [33] dataset during training to artificially increase data diversity while preserving the

underlying scene structure. The augmentations applied consisted of random horizontal and vertical flips. These transformations preserve the physical appearance and structural properties of the fur while exposing the model to a larger variety of spatial configurations.

In addition, instead of processing the full resolution images of size 1024×1024 provided by the SyntheticFur dataset, random crops of size 128×128 were sampled during training. This reduced the memory requirements and computational cost of training. Due to the available hardware constraints, a crop size of 128×128 was the largest patch size that could be processed reliably during training within the available GPU memory limits.

More aggressive augmentations, such as modifications of pixel intensities, color distributions, or geometric transformations, were intentionally avoided. Since the objective of the model is to approximate physically based rendering outputs, altering the underlying image statistics could introduce unrealistic training samples and weaken the relationship between the input buffers and the ground-truth renderings.

3.5 Model Architecture

The proposed neural rendering pipeline is based on an image-to-image translation framework that is designed to map different rendering buffers to high-quality fur renders. The architecture consists of two main components: a generator responsible for image synthesis and a discriminator or critic used to enforce realism through adversarial training.

The generator is built on the SwinIR architecture, originally proposed by Liang et al. [34], and adapted to handle multi-channel input buffers. The discriminator is implemented as a PatchGAN tailored for conditional image synthesis.

3.5.1 SwinIR

The core model used to generate the final render is based on the SwinIR architecture, originally proposed by Liang et al. [34] and implemented using the framework from the KAIR repository [35]. Structurally the original SwinIR network is divided into three stages: shallow feature extraction, deep feature extraction, and high-quality image reconstruction.

The shallow feature extraction module utilizes a convolutional layer using a 3×3 kernel, with a stride of one and zero padding to map the input into a higher-dimensional feature space. The resulting features are passed to the deep feature extraction module, which is composed of a series of Residual Swin Transformer Blocks (RSTB). An RSTB consists of several STLs that alternate between W-MSA and SW-MSA to introduce connections between neighboring non-overlapping windows from the previous layer. Each RSTB concludes with an additional convolutional layer and a residual connection that aggregates the output of the convolutional layer with the input for the RSTB, as shown in Figure 3.3.

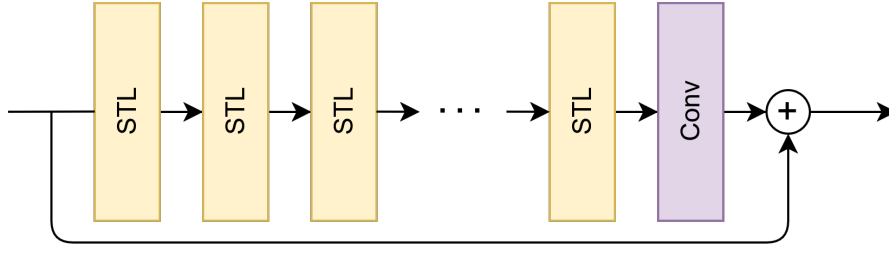


Figure 3.3: An overview of a Residual Swin Transformer Block (RSTB) used in the SwinIR model [34].

Finally, a long skip connection that aggregates the deep feature extraction output with the shallow feature extraction output is introduced. The resulting output from the residual connection is then passed to the high-quality image reconstruction module which consists of a single convolution layer for tasks that do not need upsampling. The last step in the image reconstruction module consists of a global residual connection that aggregates the output from the convolutional layer with the original input to produce the final render, resulting in an image with the same number of channels as the input.

In this project, the input consists of concatenated image buffers which increase the number of channels beyond the standard three channels of the RGB format, and the model would therefore not output an RGB image. To ensure the network predicts a standard RGB output, a custom convolutional layer is appended to the end of the base SwinIR model. Utilizing a kernel size of three, an output channel depth of three, and a zero-padding of one, this layer compresses the aggregated high-dimensional features from the base SwinIR model down to the required three-channel RGB space, producing the final high-quality fur render. The final SwinIR model is shown in Figure 3.4.

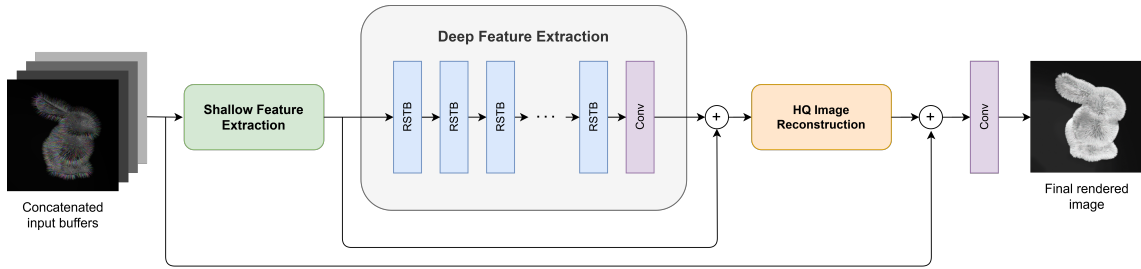


Figure 3.4: An overview of the final model with the added convolutional layer at the end of the base SwinIR model.

The instantiation of the custom SwinIR model can be configured with hyper-parameters. To handle the data format of this project, the input channel depth is set to the total number of concatenated buffer channels, allowing the base network to process the high-dimensional buffers instead of standard images. Since the objective of this thesis is to translate input buffers into realistic fur renders without altering the image dimensions, the upsampling factor is set to one, which instructs the model to bypass any spatial upsampling modules. The embedding dimension establishes the size of

the embedding for each token which was set to 90. The number of RSTB blocks used in deep feature extraction was set to four where each RSTB block utilized six STLs, and the number of parallel attention heads is set to six. The window size utilized by the W-MSA and SW-MSA was set to 8×8 pixels. Finally, the MLP expansion ratio is set to four. This ratio dictates the expansion factor for the token feature space for the MLPs in the STLs.

3.5.2 GAN

The model implements a CGAN framework where the input buffers serve as the conditional data that guides the generation process. The generator, implemented as a SwinIR model, maps these buffers directly to the ground truth image space. Following the observation that generators in image-to-image tasks often learn to ignore random noise [31], this setup omits a noise prior to create a deterministic mapping.

For the discriminator network, a PatchGAN discriminator adapted from the KAIR repository [35] was utilized to effectively model high-frequency details. Specifically three convolutional layers and a base channel of 64 was used. To incorporate the additional input buffers as conditional data, the number of input channels were adjusted accordingly. The internal layers consist of 4×4 kernels with a stride of two. This setup yields a patch size of 70×70 pixels.

During training, the discriminator received concatenated pairs consisting of the input buffers and either the synthetic output from the SwinIR generator or the ground truth image. An overview of the implemented CGAN framework can be seen in Figure 3.5.

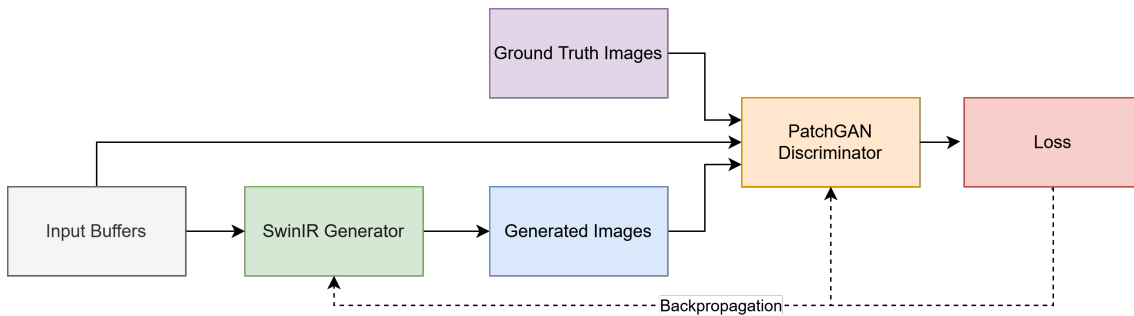


Figure 3.5: Architectural overview of the implemented CGAN framework. The architecture consists of a CGAN setup without a noise prior where the SwinIR generator uses the input buffers to produce generated images. The PatchGAN Discriminator then evaluates generated images and ground truth images, using the original input buffers as context.

3.6 Loss Functions

For complex image generation that has high-frequency textures, such as fur, relying on a single loss function is often insufficient. Therefore, this work employs multiple

different loss functions used during training of the network.

3.6.1 L1 Loss

To train the network, the L1 loss function was used as the primary loss function. In this project setting, the L1 loss measures the mean absolute difference, see Equation 2.2, between the pixel values of the predicted image and the ground truth rendered image. This encourages the model to generate outputs that match the ground truth image at the pixel level.

However, since L1 loss operates on differences between individual pixels, it does not explicitly preserve perceptual sharpness. Small spatial inaccuracies in thin fur strands can produce large pixel level errors, encouraging the network to generate smoother transitions in order to minimize the overall loss that could lead to blurring or smoothing of fur details. Despite these limitations, L1 loss serves as an effective base reconstruction objective for image synthesis.

3.6.2 Frequency Domain Loss

To address the blurring that can occur with L1 loss a frequency domain loss was introduced, also known as FFT loss, adapted from the foundational approach proposed by Yadav et al. [36]. The frequency domain loss transforms the predicted image I_p and the ground truth image I_{gt} into the frequency domain using the two-dimensional Fast Fourier Transform, yielding $F_p(u, v) = \mathcal{F}\{I_p\}$ and $F_{gt}(u, v) = \mathcal{F}\{I_{gt}\}$. The loss is then computed on the magnitude spectra as

$$\mathcal{L}_{\text{FFT}}(I_p, I_{gt}) = \frac{1}{N} \sum_{u,v} w(u, v) ||F_p(u, v)| - |F_{gt}(u, v)||, \quad (3.1)$$

where u and v denote frequencies in the vertical and horizontal directions, respectively, and N denotes the total number of frequency coefficients. The weighting function $w(u, v)$ is defined as

$$w(u, v) = 1 + (u^2 + v^2), \quad (3.2)$$

which assigns higher penalties to differences in higher frequencies. This encourages the network to better match high frequency details such as thin fur strands.

3.6.3 Gradient Difference Loss

Another way to preserve sharpness and prevent blurring is to use the Gradient Difference Loss (GDL). Introduced by Mathieu et al. [37], GDL focuses on comparing image gradients rather than the absolute pixel value. By penalizing the difference in image gradients between the predicted image P and the ground truth image T , the model is incentivized to maintain the sharpness of edges.

The GDL in this project is defined as:

$$\begin{aligned}\mathcal{L}_{\text{GDL}}(P, T) = & \frac{1}{N_{dx}} \sum_{i,j} ||P_{i,j} - P_{i,j-1}| - |T_{i,j} - T_{i,j-1}|| \\ & + \frac{1}{N_{dy}} \sum_{i,j} ||P_{i,j} - P_{i-1,j}| - |T_{i,j} - T_{i-1,j}||.\end{aligned}\tag{3.3}$$

Here, N_{dx} and N_{dy} represent the total number of horizontal and vertical neighbor pairs, respectively. The first term evaluates the horizontal gradients by calculating the absolute difference between each pixel and its immediate horizontal neighbor. Similarly, the second term captures the vertical gradients by computing the difference between adjacent pixels along the vertical axis.

Calculating the mean instead of the sum ensures the loss magnitude remains invariant to image resolution, and it prevents excessively large scalar values. If left unscaled, these large values could disproportionately dominate other loss components when combined.

3.6.4 VGG Perceptual Loss

Similarly to Frequency Domain Loss and Gradient Difference Loss, VGG Perceptual Loss, more specifically Feature Reconstruction Loss, was introduced by Johnson et al. [38] to address the overly smooth and blurred results that are often achieved when optimizing only pixel-wise losses, such as L1 loss.

The VGG Perceptual Loss functions are based on high-level feature representations extracted from a pretrained VGG network introduced by Simonyan and Zisserman [39], commonly referred to as the *loss network* ϕ . Instead of comparing images in pixel space, perceptual loss compares feature activations extracted from intermediate layers ϕ_j of the loss network, where the feature representation has dimensions $C_j \times H_j \times W_j$.

The Feature Reconstruction Loss is the squared normalized Euclidean distance between the feature representations of the output image $\hat{y}$ and the target image y :

$$\mathcal{L}_{feat}^{\phi,j}(\hat{y}, y) = \frac{1}{C_j H_j W_j} ||\phi_j(\hat{y}) - \phi_j(y)||_2^2.\tag{3.4}$$

By comparing high-level feature representations rather than pixel values, the perceptual loss encourages perceptually similar outputs by preserving textures, edges, and visually important structures. This makes the loss particularly useful for neural rendering tasks where perceptual realism is important.

3.6.5 SSIM

The Structural Similarity Index Measure (SSIM), introduced by Wang et al. [40], is a metric used for perceptual image quality designed to measure the similarity between two images based on structural information. SSIM aims to reflect human visual perception by comparing luminance, contrast, and structural properties between

images, rather than their absolute pixel values. Hence, it is commonly used as an evaluation metric for image synthesis and neural rendering tasks.

Given two image patches x and y , the SSIM index is defined as

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}, \quad (3.5)$$

where μ_x and μ_y denote the mean intensities of the image patches, σ_x^2 and σ_y^2 their variances, and σ_{xy} the covariance between the two patches. The constants C_1 and C_2 are stabilization terms introduced to avoid numerical instability when the denominator approaches zero.

SSIM values range from -1 to 1 , where a value of 1 indicates perfect structural similarity between the images.

In this thesis, SSIM was used as a quantitative evaluation metric to assess the structural similarity between generated images and ground-truth renderings.

3.6.6 GAN Loss

The standard GAN loss was implemented using the binary cross-entropy (BCE) loss function. Since the final layer of the PatchGAN discriminator produces raw, unactivated logits, a Sigmoid activation function σ is first applied to map the logits into probabilities. For N patch prediction logits x_n and target labels y_n , the BCE loss is defined as

$$\mathcal{L}_{\text{BCE}}(x, y) = -\frac{1}{N} \sum_{n=1}^N [y_n \cdot \log(\sigma(x_n)) + (1 - y_n) \cdot \log(1 - \sigma(x_n))]. \quad (3.6)$$

To update the generator and discriminator, the loss is calculated separately. The discriminator's loss $\mathcal{L}_D$ is the mean of the BCE loss on real image patches evaluated on a target of ones, and generated patches evaluated against a target of zeros:

$$\mathcal{L}_D(x_{\text{real}}, x_{\text{fake}}) = \frac{1}{2}(\mathcal{L}_{\text{BCE}}(x_{\text{real}}, 1) + \mathcal{L}_{\text{BCE}}(x_{\text{fake}}, 0)). \quad (3.7)$$

This penalizes the discriminator when it incorrectly identifies real image patches as fake ones and vice versa.

The generator's loss $\mathcal{L}_G$ is calculated by evaluating the BCE loss of the generated image patches evaluated on a target of ones:

$$\mathcal{L}_G(x_{\text{fake}}) = \mathcal{L}_{\text{BCE}}(x_{\text{fake}}, 1). \quad (3.8)$$

This penalizes the generator anytime the discriminator successfully identifies the generated patch as fake.

3.6.7 WGAN-GP Loss

To address the training instability and vanishing gradients with standard GAN loss mentioned in Section 2.7, WGAN-GP was implemented. From Gulrajani et al. [29], the calculation of the loss begins by constructing interpolated images $\hat{\mathbf{x}}$. For each pair of ground truth images, $\mathbf{x}$, and generated images $\tilde{\mathbf{x}}$, a random interpolation factor ϵ is sampled from a uniform distribution $U(0, 1)$. The interpolated image is then defined as

$$\hat{\mathbf{x}} = \epsilon \mathbf{x} + (1 - \epsilon) \tilde{\mathbf{x}}. \quad (3.9)$$

The interpolated images are then concatenated with the corresponding input buffers, $\mathbf{b}$. To calculate the gradient penalty, the gradient norm of the critic D with respect to the interpolated images $\|\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}}, \mathbf{b})\|_2$ is calculated per image pair. The final loss function for the critic is then defined as

$$\mathcal{L}_D = \mathbb{E}_{\tilde{\mathbf{x}} \sim \mathbb{P}_g} [D(\tilde{\mathbf{x}}, \mathbf{b})] - \mathbb{E}_{\mathbf{x} \sim \mathbb{P}_r} [D(\mathbf{x}, \mathbf{b})] + \lambda \mathbb{E}_{\hat{\mathbf{x}} \sim \mathbb{P}_{\hat{\mathbf{x}}}} [(\|\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}}, \mathbf{b})\|_2 - 1)^2]. \quad (3.10)$$

The first two terms of the loss compute the Wasserstein distance by taking the expected critic score between the generated and real images [28]. This trains the critic to output lower scores for generated images and higher scores for real images. Furthermore, λ represents the gradient penalty regularization factor which is set to 10 as recommended by the authors. The critic in this loss gets penalized whenever the gradient norm deviates from the target of one, which encourages it to be within the 1-Lipschitz constraint.

Finally, the goal of the generator is to maximize the discriminator’s score for the generated images, which would increase the critic’s loss. The generator’s loss is therefore calculated by minimizing the negative expected value of the critic’s output on generated images:

$$\mathcal{L}_G = - \mathbb{E}_{\tilde{\mathbf{x}} \sim \mathbb{P}_g} [D(\tilde{\mathbf{x}}, \mathbf{b})]. \quad (3.11)$$

3.6.8 LPIPS Loss

Learned Perceptual Image Patch Similarity (LPIPS), introduced by Zhang et al. [41], is a perceptual similarity metric designed to better align with human visual perception than traditional pixel-wise losses. Similarly to VGG Perceptual Loss, LPIPS compares high-level feature representations extracted from pretrained convolutional neural networks.

However, unlike VGG loss, which relies on fixed feature distances, LPIPS uses learned weighting parameters calibrated against human perceptual judgments. This allows the metric to better correlate with perceived visual similarity [41].

The LPIPS loss between a generated image $\hat{x}$ and a target image x is computed by passing both images through a pretrained feature extraction network ϕ . For each layer l , feature activations are extracted, normalized, and stacked, resulting in normalized feature representations $\hat{y}^l, y^l \in \mathbb{R}^{H_l \times W_l \times C_l}$. The channel-wise Euclidean distance between the feature representations is then calculated after scaling the activations using learned weighting vectors $w^l \in \mathbb{R}^{C_l}$. The resulting distances are spatially averaged and summed across all selected layers, as shown in Equation 3.12, where H_l and W_l denote the spatial dimensions of the feature map at layer l .

$$\mathcal{L}_{\text{LPIPS}} = \sum_l \frac{1}{H_l W_l} \sum_{h,w} \|w_l \odot (\hat{y}^l - y^l)\|_2^2 \quad (3.12)$$

3.7 Training Procedure

Multiple experiments were conducted in which different models were trained using varying configurations of hyper parameters, loss functions, and learning-rate schedulers. Consistent across all experiments was the use of the AdamW optimizer, introduced by Loshchilov and Hutter [42], together with mini-batch stochastic gradient descent. AdamW was selected due to its widespread use and stability when training Transformer-based architectures.

Training was performed using mixed-precision arithmetic through automatic mixed precision (AMP) provided by the PyTorch library. This reduces memory usage and improves computational efficiency during training.

To improve optimization stability, gradient clipping was applied during each optimization step. In addition, learning-rate scheduling strategies consisting of warm up and cosine annealing were explored to stabilize the early stages of training and improve convergence during later epochs.

During training, several combinations of the loss functions described in Section 3.6 were evaluated. For each training sample, the individual loss terms were multiplied by predefined weighting coefficients, denoted as λ , which controlled the contribution of each loss function to the total loss. The weighting coefficients for each experiment can be viewed in Table 3.1. The weighted loss terms were then summed to form the final loss used for optimization. Different training schedulers were also explored, in which loss terms were introduced progressively throughout the training. In this approach, low-level reconstruction was prioritized before focusing on perceptual quality and high detail.

During training, validation was performed periodically using the separate validation dataset described in Section 3.3.

Model checkpoints were saved continuously throughout training to enable recovery and comparison between different training stages. This was particularly important since the models were trained on the Minerva Compute Cluster described in Section 3.1.2, where the execution time of a single job is limited to one day while complete model training required multiple days.

Table 3.1: Overview of the conducted experiments and weighting coefficients. All experiments used L1 loss in combination with the additional losses listed in the table. FFT denotes Frequency Domain Loss and GDL denotes Gradient Difference Loss.

Experiment	Rasterized	Weighting Coefficients λ	Epochs
Exp01	Yes	None	20
Exp02	No	None	20
Exp03	Yes	$\lambda_{\text{FFT}} = 0.05$	60
Exp04	No	$\lambda_{\text{FFT}} = 0.3$	60
Exp05	Yes	$\lambda_{\text{GDL}} = 0.05$	60
Exp06	No	$\lambda_{\text{GDL}} = 0.3$	90
Exp07	Yes	$\lambda_{\text{FFT}} = 0.05, \lambda_{\text{GDL}} = 0.05$	60
Exp08	Yes	$\lambda_{\text{FFT}} = 0.05, \lambda_{\text{GDL}} = 0.05, \lambda_{\text{VGG}} = 0.08$	75
Exp09	Yes	$\lambda_{\text{FFT}} = 0.05, \lambda_{\text{VGG}} = 0.08$	100
Exp10	Yes	$\lambda_{\text{VGG}} = 0.3, \lambda_{\text{GAN}} = 0.2$	300
Exp11	Yes	$\lambda_{\text{FFT}} = 0.05, \lambda_{\text{VGG}} = 0.08, \lambda_{\text{WGAN}} = 0.1$	300
Exp12	No	$\lambda_{\text{FFT}} = 0.4, \lambda_{\text{VGG}} = 0.3, \lambda_{\text{WGAN}} = 5.0$	300
Exp13	Yes	$\lambda_{\text{FFT}} = 0.03, \lambda_{\text{LPIPS}} = 0.08, \lambda_{\text{WGAN}} = 0.08$	200

3.8 Evaluation

To evaluate the performance of the proposed neural rendering pipeline, both qualitative and quantitative evaluations were performed. The rasterized images and the ray-traced renderings provided by the SyntheticFur[33] dataset were used as baselines throughout the evaluation.

The rasterized images served as the primary baseline, as they represent a computationally inexpensive real-time rendering approach. The ray-traced renderings served as a reference target for the desired visual output. The purpose of these baselines was to evaluate whether the proposed method could improve the visual quality of rasterized fur rendering while remaining feasible for real-time rendering applications.

3.8.1 Qualitative

A qualitative evaluation was performed through visual comparison between the rasterized baseline and the generated output, as well as the ray-traced renderings. The purpose of the qualitative evaluation was to assess the perceptual image quality of the generated images, which is difficult to do using quantitative metrics alone.

The evaluation focused particularly on correct shading and shadows, preservation of thin strands, and overall perceptual realism. Since hair and fur contain complex structures and fine textures, qualitative inspection was essential to evaluate whether the generated output preserves high detail without introducing smoothing or visual artifacts.

To highlight differences between methods, visual comparisons were presented as full-frame renders. In addition, failure cases and visually noticeable artifacts were analyzed to identify limitations of the method.

3.8.2 Quantitative

To properly evaluate the proposed neural rendering pipeline, a quantitative evaluation was performed using numerical metrics to assess both the visual similarity of the generated images and the practical feasibility of the proposed rendering pipeline for real-time applications.

To evaluate the computational performance of the final model, the test scene `bunny_3Lights_rotateUp` from the SyntheticFur dataset [33] was rendered in its entirety, consisting of 360 frames. The total execution time was measured and used to compute the average rendering performance in frames per second (fps). Since frame rates of 30 fps are generally considered sufficient for real-time rendering applications, the proposed pipeline would need to process the complete scene in less than 12 seconds to satisfy this requirement.

To evaluate image quality, the Structural Similarity Index Measure (SSIM), described in Section 3.6.5, was used as the primary image-quality metric to measure the similarity between the generated images and the ground-truth ray-traced renderings. SSIM was calculated using Equation 3.5 for both generated outputs and rasterized baseline images to allow direct comparison between methods.

4

Results

In this chapter, the results of the various experiments conducted throughout the thesis are presented. The performance of the models are evaluated both qualitatively, through visual comparisons, and quantitatively, utilizing SSIM for evaluating image similarity and total execution time for computational speed.

Table 4.1: Overview of the conducted experiments and their highest achieved SSIM values. All experiments used L1 loss in combination with the additional losses listed in the table. FFT denotes Frequency Domain Loss and GDL denotes Gradient Difference Loss.

Experiment	Rasterized	Additional Losses	Epochs	SSIM
Exp01	Yes	None	20	0.797
Exp02	No	None	20	0.763
Exp03	Yes	FFT	60	0.766
Exp04	No	FFT	60	0.767
Exp05	Yes	GDL	60	0.802
Exp06	No	GDL	90	0.782
Exp07	Yes	FFT + GDL	60	0.766
Exp08	Yes	FFT + GDL + VGG	75	0.792
Exp09	Yes	FFT + VGG	100	0.774
Exp10	Yes	VGG + GAN	300	0.739
Exp11	Yes	FFT + VGG + WGAN	300	0.787
Exp12	No	FFT + VGG + WGAN	300	0.748
Exp13	Yes	FFT + LPIPS + WGAN	200	0.795

4.1 Rasterized Baseline

A visual comparison between the rasterized baseline and the ground-truth ray-traced image can be seen in Figure 4.1. The baseline clearly lacks detail and areas with high frequency fur looks blurry and smooth. The SSIM score between these two images is **0.265**.

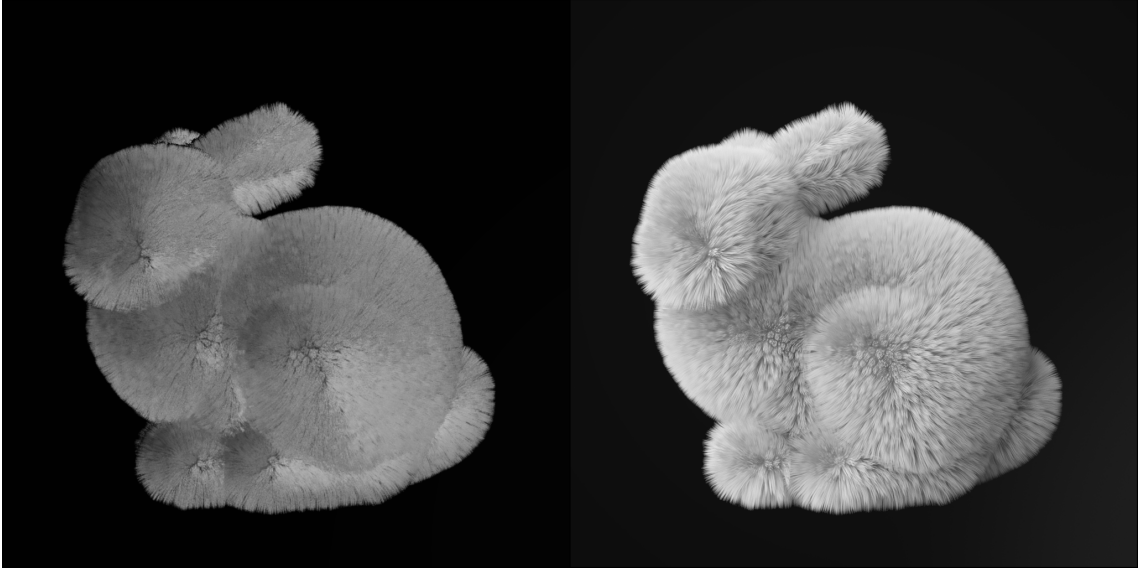


Figure 4.1: Qualitative comparison for the rasterized baseline and ground-truth ray-traced rendering.

4.2 L1 Loss

This experiment evaluates the performance of a purely pixel-wise reconstruction objective using only L1 loss. The model was trained until convergence for approximately 20 epochs. The training loss throughout optimization is shown in Figure 4.2.

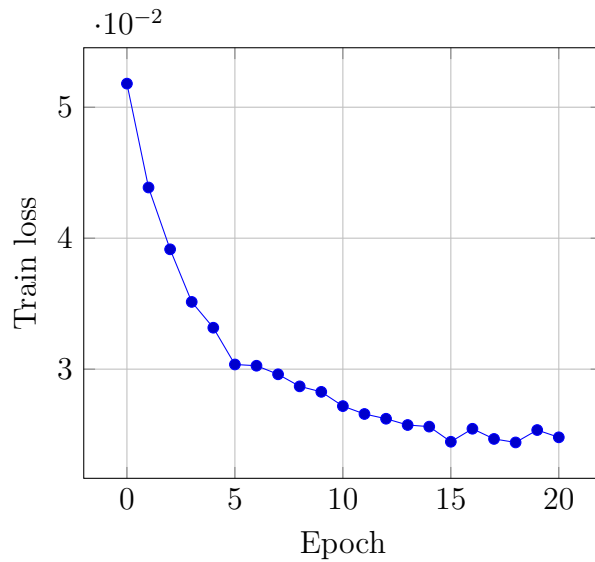


Figure 4.2: Graph of L1 loss over Epoch during training

The model achieved an SSIM score of **0.797** on the test set when evaluated against the ground-truth ray-traced renderings.

Figure 4.3 presents a qualitative comparison between the generated output and the ground-truth rendering. The model successfully reconstructs the overall silhouette

and captures portions of the fur directionality. However, the generated image lacks high-frequency detail and exhibits smoother transitions in dense fur regions, producing a visibly blurred appearance compared to the target render. In addition, inconsistencies in shading are visible, particularly in darker regions of the fur where localized artifacts appear.

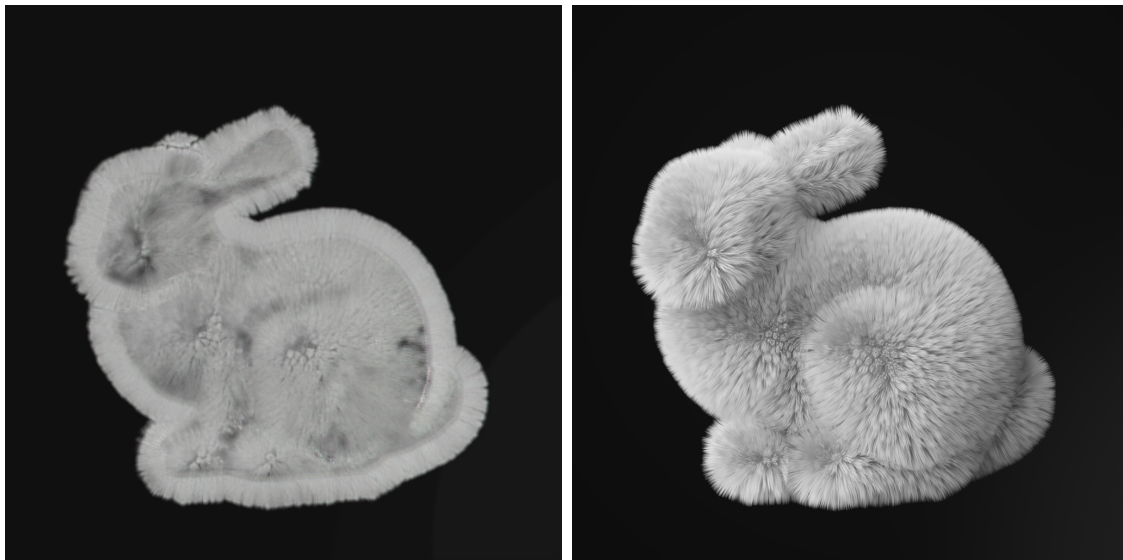


Figure 4.3: Qualitative comparison between the model output using only L1 loss and the ground-truth ray-traced rendering (Exp01 in Table 4.1).

4.3 Frequency Domain Loss

To improve the preservation of high-frequency details, Frequency Domain (FFT) Loss was introduced together with the L1 reconstruction Loss. The model was first trained for 20 epochs using only L1 and then further optimized for an additional 40 epochs combining the losses, for a total of 60 epochs trained.

The model achieved an SSIM score of **0.766** when evaluated against the ground-truth ray-traced renderings.

Figure 4.4 presents a qualitative comparison between the model output and the ground-truth rendering. Compared to the model trained using only L1 loss, the frequency domain-based approach preserves substantially more high-frequency detail. Individual fur strands are more distinguishable, and the generated image appears sharper, with less visible blurring in dense fur regions. However, the shading remains inconsistent with the target rendering, particularly in darker regions of the fur.

4.4 GDL Loss

The GDL loss used a similar setup as the frequency domain loss and achieved an SSIM score of **0.802**.

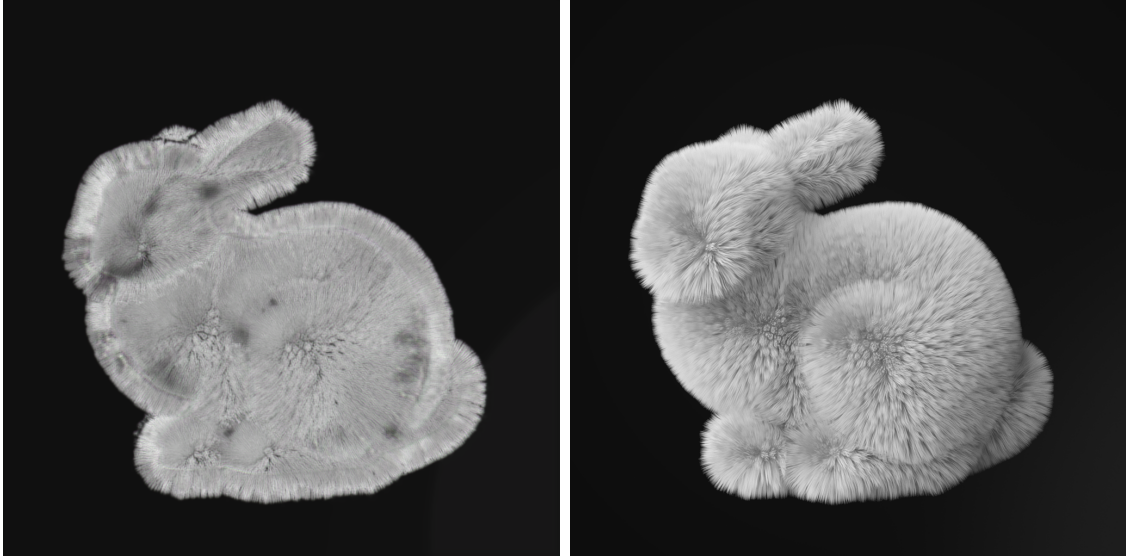


Figure 4.4: Qualitative comparison between the generated output using FFT loss and the ground-truth ray-traced rendering (Exp03 in Table 4.1).

As shown in Figure 4.5, utilizing GDL loss results in fur strands that appear softer and more blurry compared to those generated with FFT loss. Despite reduction in sharpness, GDL still preserves some of the distinct clusters of fur strands and does not blur them out completely. However, the shading artifacts initially introduced by the L1 loss still remain visible and the shading in darker regions are inconsistent with the ground truth.

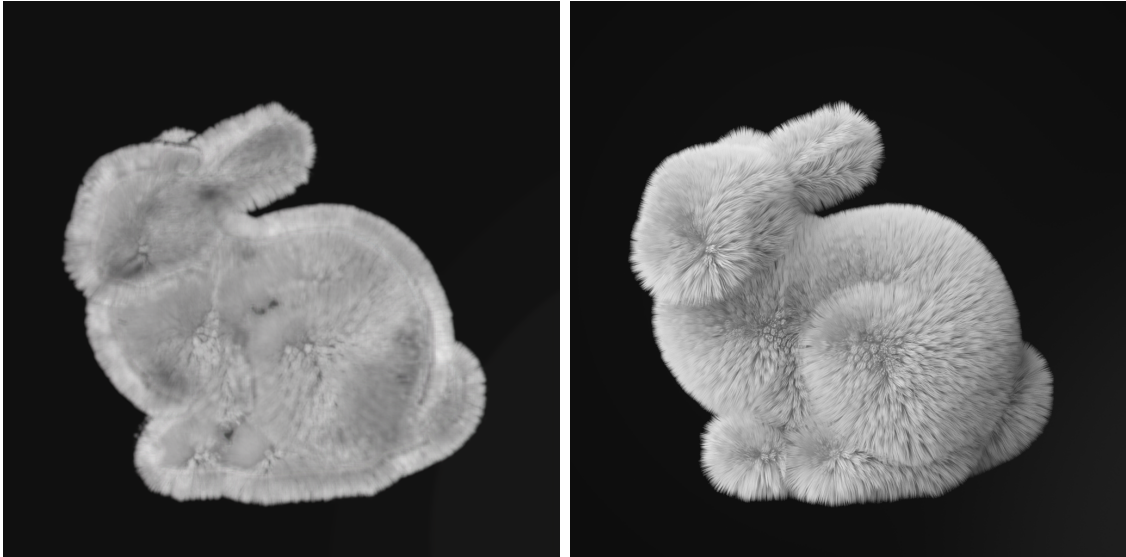


Figure 4.5: Qualitative comparison between the generated output using GDL loss and the ground-truth ray-traced rendering (Exp05 in Table 4.1).

Using GDL loss combined with the FFT loss an SSIM score of **0.766** was achieved. The render from this model appears as a blurrier version of Exp03 and also here the artifacts from the L1 loss remain, see Figure 4.6.

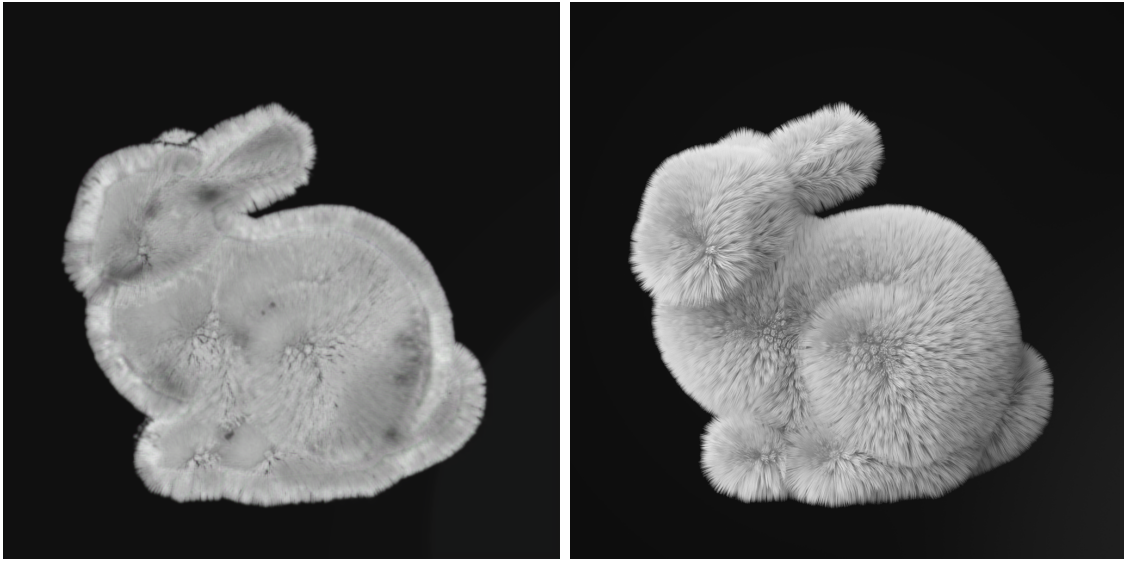


Figure 4.6: Qualitative comparison between the generated output using FFT + GDL loss and the ground-truth ray-traced rendering (Exp07 in Table 4.1).

4.5 VGG Perceptual Loss

VGG Perceptual Loss was introduced to increase the perceptual realism of the generated image. The loss was evaluated together with the FFT-based models from Exp03 and Exp07 in Table 4.1. However, the configuration combining FFT, GDL, and VGG was terminated early due to persistent blurring in dense fur regions.

The final configuration was based on the pretrained model from Exp03 in Table 4.1, which had been trained for 60 epochs using L1 and FFT loss. The model was then further optimized for an additional 40 epochs using the added VGG Perceptual loss.

The model achieved a SSIM score of **0.774** when evaluated against the ground-truth ray-traced renderings.

Figure 4.7 presents a qualitative comparison between the generated output and the ground-truth rendering. Compared to the FFT model, the addition of VGG perceptual loss improved the overall perceptual realism of the generated image. In particular, the overly dark artifacts visible in dense fur regions were reduced while the sharper details introduced by FFT loss were preserved. However, the shading remains inconsistent with the ground-truth image, especially in regions where fur strands extend away from the camera. Although, the overall shading quality appears improved compared to previous experiments.

4.6 GAN Loss

Both standard GAN and WGAN loss were tested during training. For Exp10, the model trained using VGG loss combined with standard GAN loss, an SSIM score of **0.739** was achieved. The resulting render from Exp10, seen in the left image of

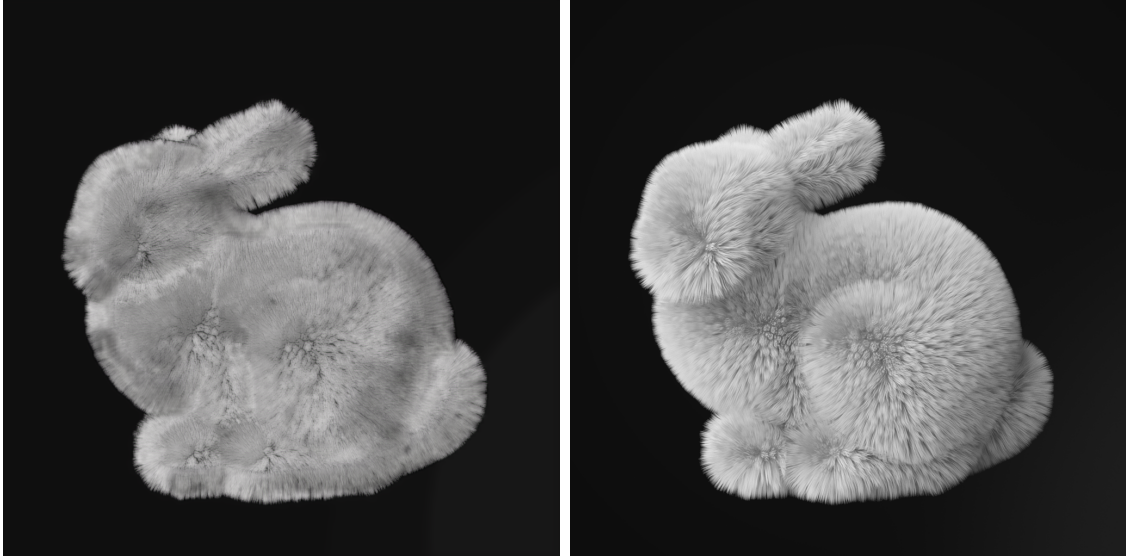


Figure 4.7: Qualitative comparison between the generated output using VGG loss and the ground-truth ray-traced rendering (Exp09 in Table 4.1).

Figure 4.8, shows that the model could preserve some clusters of fur strands and on a macro-level the texture perceptually resembles fur but individual fur strands are not as detailed. The artifacts appearing as dark spots present in previous experiments have mostly disappeared. However, this model still struggles with inconsistent shading such as on the bunny’s tail. Zooming in at the render reveals that some color fringing is present.

For Exp11, which represents the model trained using FFT, VGG, and WGAN loss, an SSIM score of **0.787** was achieved. This model manages to preserve the fur textures and also eliminates the artifacts. The fur strands are, however, not as defined as Exp10 and appear to be slightly softer, but no color fringing appears and the the individual fur strands are more detailed. The shading, similar to the previous experiment, is not very consistent when matching against the ground truth image.



Figure 4.8: Results from standard GAN loss and WGAN loss. From left to right, Exp10, Exp11, and ground-truth.

4.7 Effects of Rasterized Buffer

Training the model with and without the rasterized buffer was also tested. A lower SSIM score was noticed when dropping the rasterized buffer for all comparisons except for Exp03 and Exp04, where the SSIM difference was negligible (**0.766** and **0.767** respectively).

Qualitative visual inspection aligns with the quantitative drops. Figure 4.9 illustrates a comparison between models using L1 loss with and without rasterized buffer. Dropping the rasterized buffer resulted in a highly blurry render that failed to capture high-frequency details. Conversely using the rasterized buffer allowed the model to preserve distinct fur clustering.

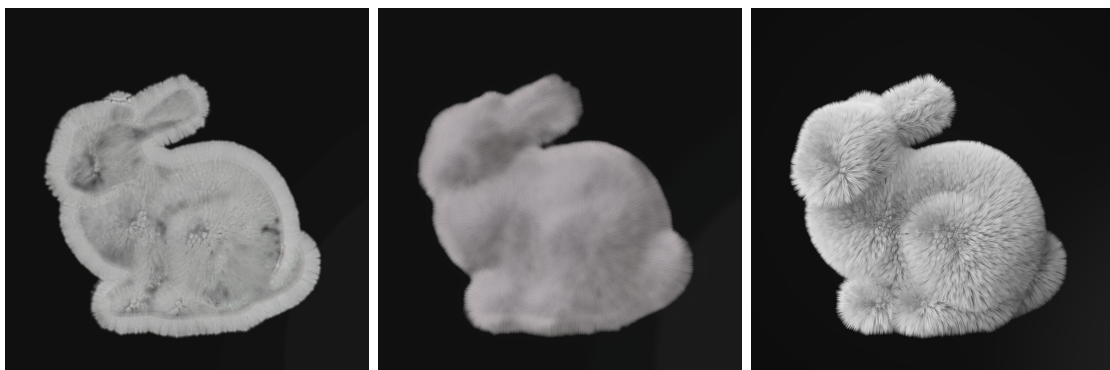


Figure 4.9: Effects of the rasterized buffer for model with simpler loss function. From left to right, Exp01, Exp02, and the ground-truth, see Table 4.1.

When looking at a comparison of the model with a more complex loss function that consists of FFT, VGG, and WGAN loss, as shown in Figure 4.10, it can be seen that the model from Exp11 successfully captures macro-level fur patches that aligns more closely with the ground truth image compared to results from Exp12.

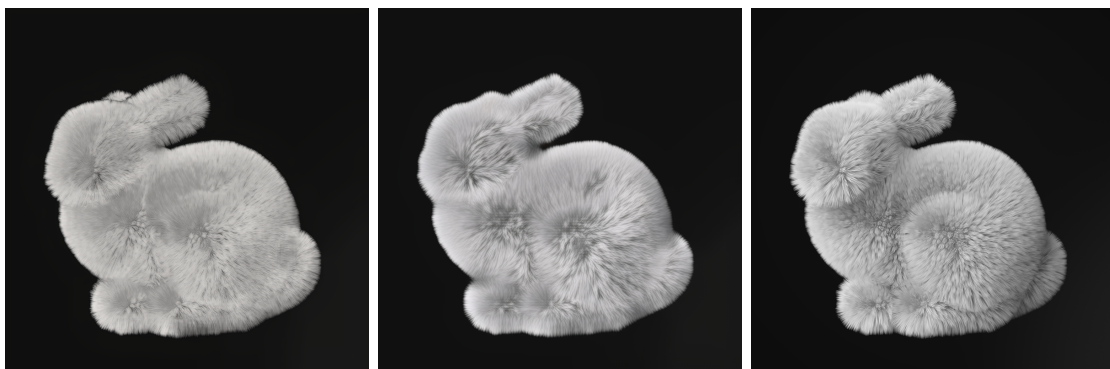


Figure 4.10: Effects of the rasterized buffer for model with more complex loss function. From left to right, Exp11, Exp12 and the ground-truth, see Table 4.1.

For more comparisons of models trained with and without the rasterized buffer, see Appendix A.1.

4.8 Final Model

The final model, Exp13 in the Table 4.1, achieved the best visual results and used a combination of L1, FFT, LPIPS, and WGAN losses. This model also served as an evaluation for the LPIPS loss function. The model was trained from scratch using a combination of L1, FFT, and LPIPS from epoch 0, while WGAN loss was introduced after 20 epochs. The weighting coefficient λ_{WGAN} was slowly increased during training to encourage stable adversarial learning and to prevent the model from deviating excessively from the reconstruction objectives learned during the initial training stage.

The model achieved an SSIM score of **0.795** when evaluated against the ground-truth ray-traced renderings. Figure 4.11 shows the training and validation losses throughout the training. Both losses decrease rapidly during the initial stages of training and appear to converge after approximately 140 epochs. A more detailed view of each loss component can be seen in Figure 4.12.

Figure 4.13 illustrates the WGAN loss for the final model. At 20 epochs the WGAN loss starts and the critic loss exhibits an expected spike due to the untrained PatchGAN network with newly initialized weights. The network then quickly corrects itself, and the critic loss steadily drops below zero. This confirms that the network is successfully estimating the Wasserstein distance. Conversely the generator loss exhibits a smooth, stable climb before converging. Together these curves demonstrates that the WGAN-GP training was stable and that the gradient penalty worked as expected.

Figure 4.14 shows a qualitative comparison between the output of the model and the ground-truth image. The generated image preserves a high level of detail, with individual distinguishable hair strands throughout most regions of the fur. The shading appears closer to the ground-truth rendering, particularly in shadowed regions between fur strands. However, some shading inconsistencies remain, especially in areas partially occluded by protruding parts of the underlying geometry. Additionally, some dark artifacts appear in other angles, which can be seen from the sampled frames of the full scene render `bunny_3Lights_rotateUp` in Appendix A.4. Other full scene renders can be viewed in the Appendix, such as A.5 and A.6.

4.9 Runtime Performance

Rendering the test scene `bunny_3Lights_rotateUp` from the SyntheticFur dataset, consisting of 360 frames, took a total of 2819 seconds. This corresponds to an average rendering time of approximately 7832 ms per frame. The resulting throughput is approximately 0.13 fps.

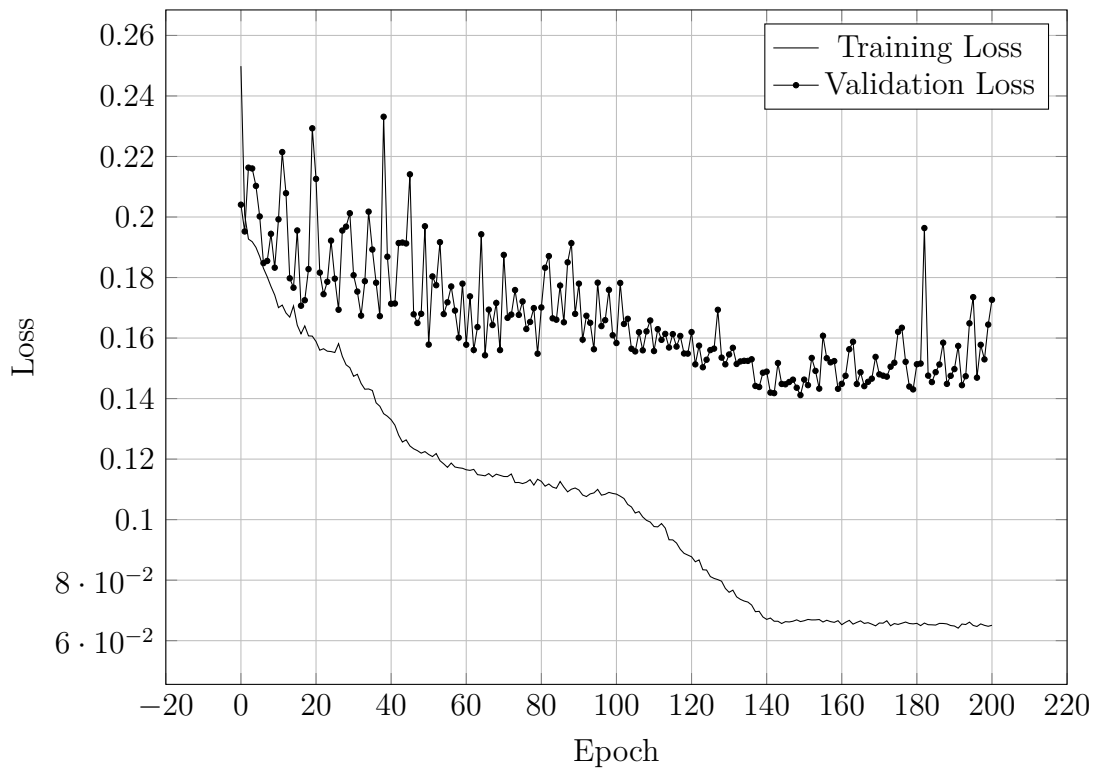


Figure 4.11: Graph of train and validation loss over Epoch during training

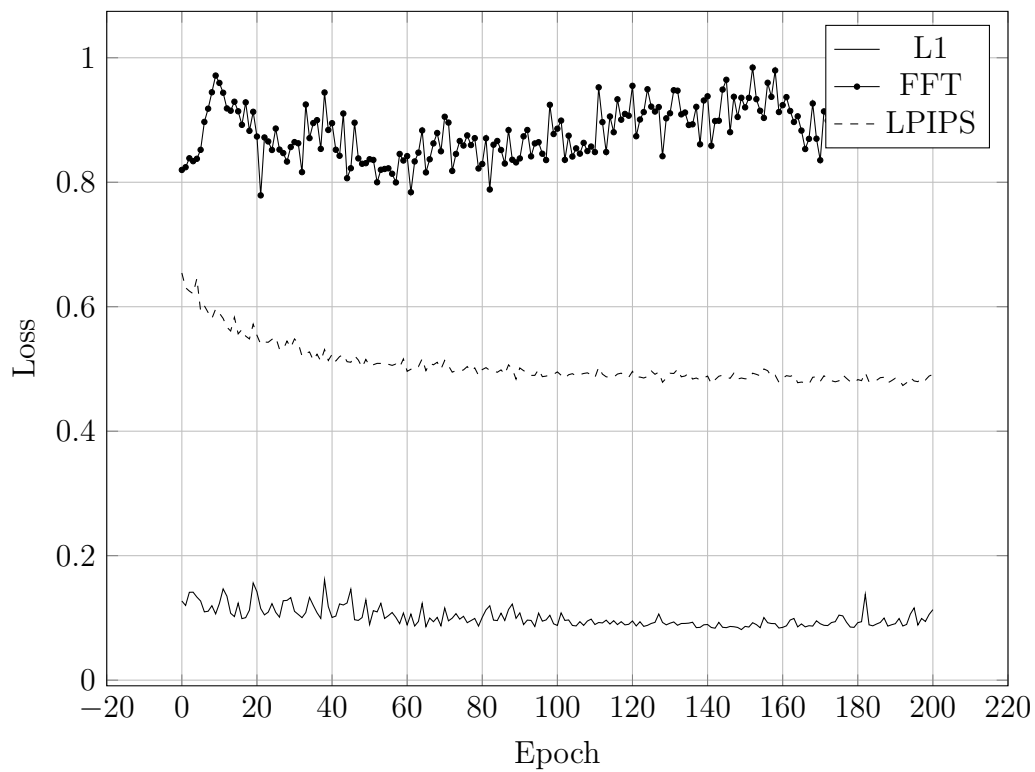


Figure 4.12: Graph of L1, FFT, and LPIPS loss over Epoch during training

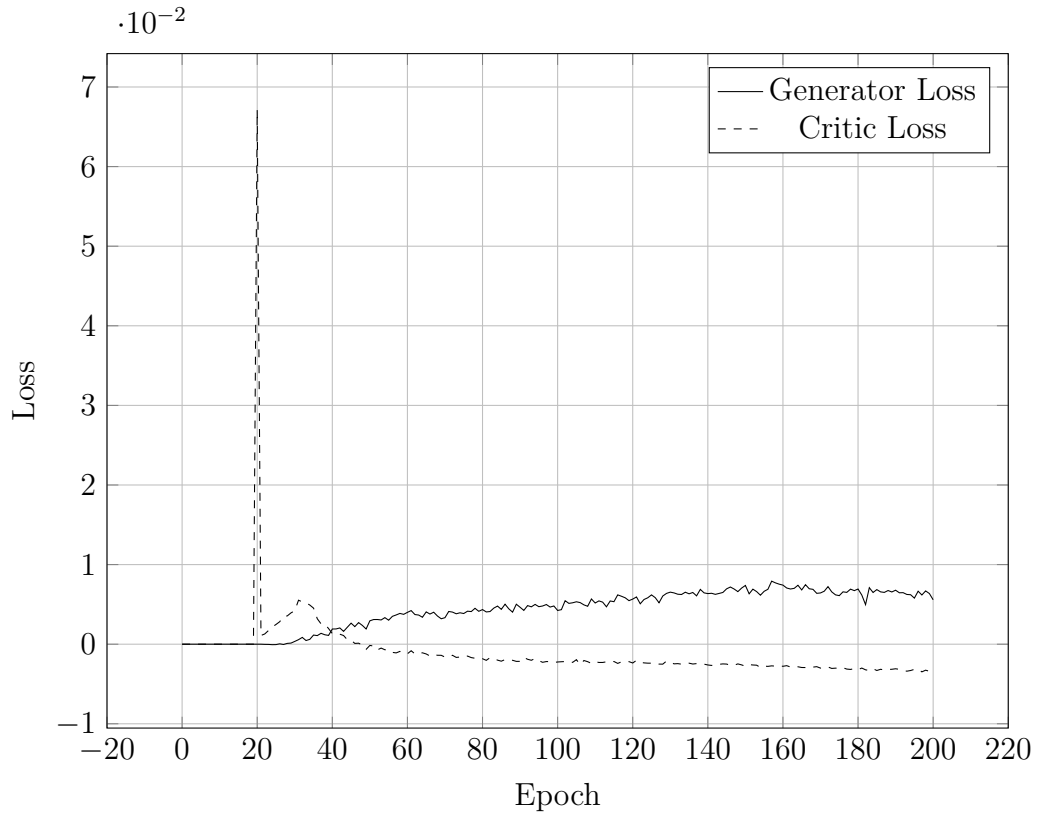


Figure 4.13: Graph of Generator and Critic loss over Epoch during training

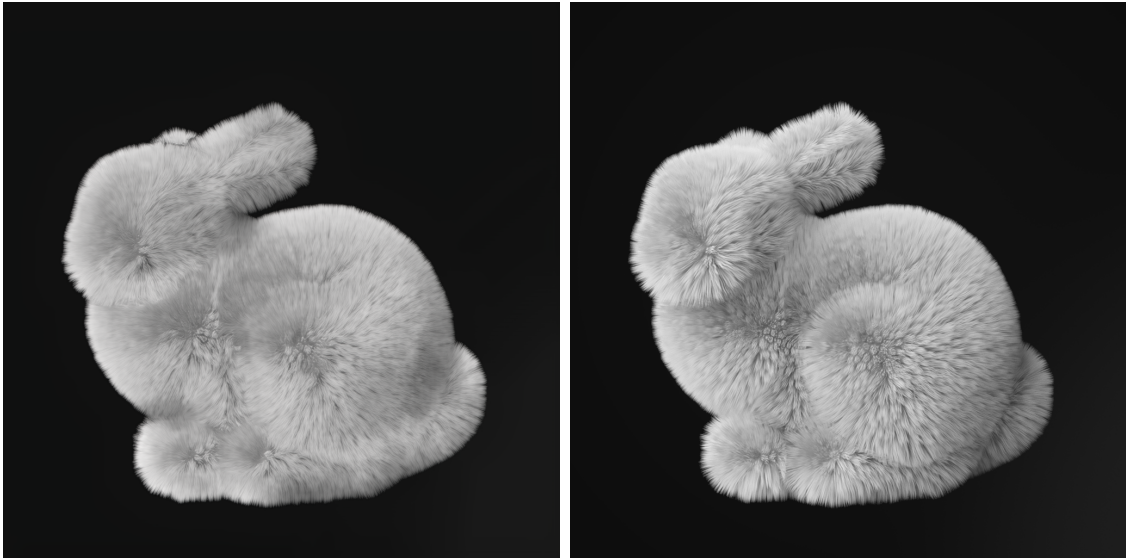


Figure 4.14: Qualitative comparison between the generated output using the final model and the ground-truth ray-traced rendering (Exp13 in Table 4.1).

5

Discussion

The results from Chapter 4 provide several insights into the feasibility of utilizing SwinIR-based architecture for hair and fur rendering. While the qualitative and quantitative results show promising results they also highlight limitations of the model.

5.1 Positive Findings

The experiments demonstrated that FFT loss was one of the most influential loss functions for preserving high-frequency fur. Although the introduction of FFT loss in Exp03 decreased the SSIM score, compared to the baseline L1 model, qualitative inspection revealed a substantial increase in hair strand detail and a reduction in blurring.

The feature-based loss functions played a large role in improving the overall visual appearance and reducing some of the dark artifacts observed in earlier experiments. While both VGG perceptual loss and LPIPS loss achieved this, LPIPS was by far superior in preserving hair strand texture and overall fur appearance.

Adversarial training acted primarily as a refinement stage rather than introducing entirely new visual features for the generator. As it seems that most structural fur detail was learned through FFT and perceptual losses, the adversarial objective might have encouraged the generator to produce outputs that appeared more natural and realistic.

One of the core objectives of this thesis was to train a model capable of generalizing over unobserved scenes. The final proposed model, Exp13 in Table 4.1, was capable of rendering the scene `bunny_3Lights_rotateUp` with high detail and results that closely resemble their respective ground truth images. This is especially evident in the hair strand detail, even though the model was not trained on any scenes containing this primitive object. Hence, achieving this objective, as well as the third objective to achieve high-quality visual results that closely approximate the ground-truth images.

As demonstrated in Section 4.7, the inclusion of the rasterized input buffer yielded a significant improvement in the level of detail of the fur strands. Models trained with this buffer also preserved distinct fur clustering and macro-level structures better. This improvement is likely because the rasterized buffer provides the model with a

spatial map of the individual hair strands at a constant brightness. Consequently, the generator model does not need to infer the position of the fur strands and can instead focus on learning shading details.

However, imperfections in the model remain, specifically shading inconsistencies, artifacts, and challenges related to real-time rendering.

5.2 Limitations

The proposed method faces multiple limitations. Most notably, the inference time exceeds the latency requirements for practical real-time rendering, which was a primary objective outlined in Section 1.3. As demonstrated in Section 4.9, the model currently achieves an average throughput of roughly 0.13 fps. This computational bottleneck can likely be attributed to the STL of the SwinIR generator. Although an STL scales linearly with image size to successfully avoid the quadratic computational bottlenecks of standard global MSA, its practical execution remains constrained by the sheer volume of mathematical operations inherent to the MLP and modified MSA modules.

Beyond inference speed, the model’s performance may also have been impacted by the hardware constraints encountered during training. As noted in Section 3.4, the crop size during the training process was restricted to 128×128 pixels due to the VRAM limit on the Minerva compute cluster. Consequently, the network was primarily exposed to localized contexts rather than the context found in the full-resolution image. While this allowed the SW-MSA to successfully learn high-frequency details and macro-level fur clusters, it is likely that this constrained global receptive field hindered the model’s ability to accurately capture global lighting consistency across the entire geometry. This spatial limitation could be a contributing factor to the visual artifacts observed in the final renders, most notably the persistent shading inconsistencies.

Furthermore, these issues might also stem from the characteristics of the dataset used during training. Since the training data relies on simple geometric primitives and predefined lighting conditions, described in Section 3.3, the network may not have been sufficiently exposed to more complex shading scenarios. As a result, when the model attempts to generate fur in occluded areas, it struggles to accurately approximate the light transport, ultimately leading to darkened artifacts in the qualitative results.

Using SSIM as a quantifiable measure for how well the output resembles the ground-truth also proved to be a limitation in this study. Although SSIM is a widely used measure in reconstruction tasks, the experiment results demonstrated that a higher SSIM score did not necessarily correspond to a better visual result. For example, the baseline model, trained only on L1 loss, had the second highest SSIM score while producing a blurred results with non distinguishable hair strands, shading that is highly inconsistent with the ground truth, and plenty of visual artifacts in forms of black spots in the fur. This suggests that SSIM is insufficient for evaluating

neural fur rendering. Consequently, qualitative visual inspection was required to complement the quantitative evaluation.

5.3 Application

Despite these limitations, since the proposed method is faster than path tracing, practical applications in speeding up offline rendering tasks remain. One such application is for the model to give an approximated preview of the final result of an rendering engine before committing to rendering an entire scene.

5.4 Future Work

Future research should focus on addressing the two main limitations identified in this study, the visual shading artifacts and the inference speed.

To address the shading inconsistencies and darkened artifacts, curating a more complex training dataset should be a primary focus. Future datasets should incorporate complex geometric structures and self-shadowing scenarios to provide the network with the ground-truth examples necessary to learn accurate shading. Furthermore, training the network on image crops exceeding the current 128×128 pixel limit using hardware with greater VRAM capacity should also be explored to improve its ability to maintain lighting consistency across large spatial areas.

The second critical avenue for future work involves optimizing the network to achieve real-time rendering capabilities. Future iterations could increase the inference speed by testing a more lightweight architectural configuration by for example dropping the number of RSTBs or decreasing the embedding channel depth, which drastically lowers the parameter count at the cost of potential performance drops. Finally, the inference speed could be accelerated through low-level software optimizations, such as implementing custom CUDA kernels tailored specifically for W-MSA.

6

Conclusion

The primary objective of this thesis was to investigate the feasibility of a rendering pipeline with Transformer-based post-processing tailored specifically for hair and fur. By leveraging a SwinIR-based architecture, the final model successfully generated realistic renders that generalized to unseen geometric primitives without requiring per-scene training. Incorporating Frequency Domain loss was effective at preserving the high-frequency details of individual hair strands, while LPIPS loss significantly enhanced the overall perceptual realism of the fur textures.

However, the current model faces critical limitations such as low inference speeds that prevents its deployment in real-time applications. Additionally, the output exhibits visual artifacts and shading inconsistencies in occluded areas.

Ultimately, these initial results indicate that a Transformer-based pipeline for complex hair and fur might be feasible, but extensive future work in architectural optimization and curating a more complex dataset is required to truly confirm its viability.

Bibliography

- [1] *Ray Tracing*, en-US. Accessed: Apr. 8, 2026. [Online]. Available: <https://developer.nvidia.com/discover/ray-tracing>.
- [2] R. R. Currius, U. Assarsson, and E. Sintorn, “Real-Time Hair Filtering with Convolutional Neural Networks,” *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, vol. 5, no. 1, 15:1–15:15, May 2022. DOI: 10.1145/3522606. Accessed: Jun. 9, 2026. [Online]. Available: <https://dl.acm.org/doi/10.1145/3522606>.
- [3] E. Sintorn and U. Assarsson, “Hair self shadowing and transparency depth ordering using occupancy maps,” in *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, ser. I3D ’09, New York, NY, USA: Association for Computing Machinery, Feb. 2009, pp. 67–74, ISBN: 978-1-60558-429-4. DOI: 10.1145/1507149.1507160. Accessed: Jun. 9, 2026. [Online]. Available: <https://dl.acm.org/doi/10.1145/1507149.1507160>.
- [4] A. Tewari et al., *Advances in Neural Rendering*, arXiv:2111.05849 [cs], Mar. 2022. DOI: 10.48550/arXiv.2111.05849. Accessed: Nov. 19, 2025. [Online]. Available: <http://arxiv.org/abs/2111.05849>.
- [5] I. Nnoli, *NVIDIA RTX Neural Rendering Introduces Next Era of AI-Powered Graphics Innovation*, en-US, Jan. 2025. Accessed: Nov. 21, 2025. [Online]. Available: <https://developer.nvidia.com/blog/nvidia-rtx-neural-rendering-introduces-next-era-of-ai-powered-graphics-innovation/>.
- [6] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, *NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis*, arXiv:2003.08934 [cs], Aug. 2020. DOI: 10.48550/arXiv.2003.08934. Accessed: Nov. 20, 2025. [Online]. Available: <http://arxiv.org/abs/2003.08934>.
- [7] C. Zeng, Y. Dong, P. Peers, H. Wu, and X. Tong, “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*, arXiv:2505.21925 [cs], Aug. 2025, pp. 1–11. DOI: 10.1145/3721238.3730595. Accessed: Nov. 6, 2025. [Online]. Available: <http://arxiv.org/abs/2505.21925>.
- [8] G. Bhokare, E. Montalvo-Ruiz, E. Diaz, and C. Yuksel, “Real-Time Hair Rendering with Hair Meshes,” in *SIGGRAPH 2024 Conference Papers*, ser. SIGGRAPH ’24, New York, NY, USA: ACM Press, Jul. 2024, 61:1–61:10, ISBN: 979-8-4007-0525-0. DOI: 10.1145/3641519.36575214. [Online]. Available: <https://doi.org/10.1145/3641519.3657521>.

- [9] J. Hsu, T. Wang, Z. Pan, X. Gao, C. Yuksel, and K. Wu, “Real-Time Physically Guided Hair Interpolation,” *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2024)*, vol. 43, no. 4, 95:1–95:11, Jul. 2024, ISSN: 0730-0301. DOI: 10.1145/3658176. [Online]. Available: <https://doi.org/10.1145/3658176>.
- [10] E. Strubell, A. Ganesh, and A. McCallum, *Energy and Policy Considerations for Deep Learning in NLP*, arXiv:1906.02243 [cs], Jun. 2019. DOI: 10.48550/arXiv.1906.02243. Accessed: Jan. 21, 2026. [Online]. Available: <http://arxiv.org/abs/1906.02243>.
- [11] J.-C. Prunier, *Rendering an Image of a 3D Scene*. Accessed: Apr. 16, 2026. [Online]. Available: <https://www.scratchapixel.com/lessons/3d-basic-rendering/rendering-3d-scene-overview/rendering-3d-scene.html>.
- [12] Chrschn, *English: An example of a polygon mesh. Illustration of a dolphin, represented with triangles*. Public domain, Mar. 2007. Accessed: Apr. 20, 2026. [Online]. Available: https://commons.wikimedia.org/wiki/File:Dolphin_triangle_mesh.png.
- [13] J.-C. Prunier, *An Overview of the Rasterization Algorithm*. Accessed: Apr. 21, 2026. [Online]. Available: <https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation/overview-rasterization-algorithm.html>.
- [14] Scratchapixel, *Figure 2: Rasterization decomposition*, Licensed under CC BY-NC-ND 4.0. Accessed: Apr. 20, 2026. [Online]. Available: <https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation/overview-rasterization-algorithm.html>.
- [15] Scratchapixel, *Figure 5: Depth buffer*, Licensed under CC BY-NC-ND 4.0. Accessed: Apr. 20, 2026. [Online]. Available: <https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation/overview-rasterization-algorithm.html>.
- [16] J.-C. Prunier, *Rasterization: A Practical Implementation*. Accessed: Apr. 21, 2026. [Online]. Available: <https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation/rasterization-practical-implementation.html>.
- [17] *Understanding Mantra raytraced shading*. Accessed: Apr. 28, 2026. [Online]. Available: <https://www.sidefx.com/docs/houdini/render/bsdf.html>.
- [18] Henrik, *English: This diagram illustrates the ray tracing algorithm for rendering an image*, Licensed under CC BY-SA 4.0, Apr. 2008. Accessed: Apr. 21, 2026. [Online]. Available: https://commons.wikimedia.org/wiki/File:Ray_trace_diagram.svg.
- [19] F. Rosenblatt, *The Perceptron, a Percieving and Recognizing Automation*. Cornell Aeronautical Laboratory, vol. 1957, <https://bpb-us-e2.wpmucdn.com/websites.umass.edu/cdn11111/files/2015/07/1957.pdf>.
- [20] K. P. Murphy, *Machine Learning*. MIT Press, vol. 2012, ISBN: 978-0-262-01802-9.
- [21] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, vol. 2016, \url{http://www.deeplearningbook.org}.
- [22] K. O’Shea and R. Nash, *An Introduction to Convolutional Neural Networks*, arXiv:1511.08458 [cs.NE], Dec. 2015. DOI: 10.48550/arXiv.1511.08458.

- Accessed: May 24, 2026. [Online]. Available: <http://arxiv.org/abs/1511.08458>.
- [23] A. Vaswani et al., *Attention Is All You Need*, arXiv:1706.03762 [cs], Aug. 2023. DOI: 10.48550/arXiv.1706.03762. Accessed: Dec. 10, 2025. [Online]. Available: <http://arxiv.org/abs/1706.03762>.
 - [24] A. Dosovitskiy et al., *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*, arXiv:2010.11929 [cs], Jun. 2021. DOI: 10.48550/arXiv.2010.11929. Accessed: Dec. 10, 2025. [Online]. Available: <http://arxiv.org/abs/2010.11929>.
 - [25] Z. Liu et al., *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows*, arXiv:2103.14030 [cs], Aug. 2021. DOI: 10.48550/arXiv.2103.14030. Accessed: Nov. 21, 2025. [Online]. Available: <http://arxiv.org/abs/2103.14030>.
 - [26] D. Hendrycks and K. Gimpel, *Gaussian Error Linear Units (GELUs)*, arXiv:1606.08415 [cs.LG], Jun. 2023. DOI: 10.48550/arXiv.1606.08415. Accessed: May 26, 2026. [Online]. Available: <http://arxiv.org/abs/1606.08415>.
 - [27] I. J. Goodfellow et al., *Generative Adversarial Networks*, arXiv:1406.2661 [stat], Jun. 2014. DOI: 10.48550/arXiv.1406.2661. Accessed: May 6, 2026. [Online]. Available: <http://arxiv.org/abs/1406.2661>.
 - [28] M. Arjovsky, S. Chintala, and L. Bottou, *Wasserstein GAN*, arXiv:1701.07875 [stat.ML], Dec. 2017. DOI: 10.48550/arXiv.1701.07875. Accessed: May 11, 2026. [Online]. Available: <http://arxiv.org/abs/1701.07875>.
 - [29] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, *Improved Training of Wasserstein GANs*, arXiv:1704.00028 [cs.LG], Dec. 2017. DOI: 10.48550/arXiv.1704.00028. Accessed: May 12, 2026. [Online]. Available: <http://arxiv.org/abs/1704.00028>.
 - [30] M. Mirza and S. Osindero, *Conditional Generative Adversarial Nets*, arXiv:1411.1784 [cs], Nov. 2014. DOI: 10.48550/arXiv.1411.1784. Accessed: May 6, 2026. [Online]. Available: <http://arxiv.org/abs/1411.1784>.
 - [31] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, *Image-to-Image Translation with Conditional Adversarial Networks*, arXiv:1611.07004 [cs.CV], Nov. 2018. DOI: 10.48550/arXiv.1611.07004. Accessed: May 19, 2026. [Online]. Available: <http://arxiv.org/abs/1611.07004>.
 - [32] A. Paszke et al., *PyTorch: An Imperative Style, High-Performance Deep Learning Library*, arXiv:1912.01703 [cs.LG], Dec. 2019. DOI: 10.48550/arXiv.1912.01703. Accessed: May 19, 2026. [Online]. Available: <http://arxiv.org/abs/1912.01703>.
 - [33] T. Le, R. Poplin, F. Bertsch, A. S. Toor, and M. L. Oh, *SyntheticFur dataset for neural rendering*, arXiv:2105.06409 [cs], May 2021. DOI: 10.48550/arXiv.2105.06409. Accessed: Nov. 6, 2025. [Online]. Available: <http://arxiv.org/abs/2105.06409>.
 - [34] J. Liang, J. Cao, G. Sun, K. Zhang, L. V. Gool, and R. Timofte, *SwinIR: Image Restoration Using Swin Transformer*, arXiv:2108.10257 [eess.IV], Aug. 2021. DOI: 10.48550/arXiv.2108.10257. Accessed: May 19, 2026. [Online]. Available: <http://arxiv.org/abs/2108.10257>.

- [35] K. Zhang, *Csxn/KAIR*, original-date: 2019-12-15T19:42:25Z, May 2026. Accessed: May 19, 2026. [Online]. Available: <https://github.com/csxn/KAIR>.
- [36] O. Yadav, K. Ghosal, S. Lutz, and A. Smolic, *Frequency Domain Loss Function for Deep Exposure Correction of Dark Images*, en, Apr. 2021. DOI: 10.1007/s11760-021-01915-4. Accessed: May 26, 2026. [Online]. Available: <https://arxiv.org/abs/2104.10856v2>.
- [37] M. Mathieu, C. Couprie, and Y. LeCun, *Deep multi-scale video prediction beyond mean square error*, arXiv:1511.05440 [cs], Feb. 2016. DOI: 10.48550/arXiv.1511.05440. Accessed: May 9, 2026. [Online]. Available: <http://arxiv.org/abs/1511.05440>.
- [38] J. Johnson, A. Alahi, and L. Fei-Fei, *Perceptual Losses for Real-Time Style Transfer and Super-Resolution*, arXiv:1603.08155 [cs], Mar. 2016. DOI: 10.48550/arXiv.1603.08155. Accessed: Mar. 4, 2026. [Online]. Available: <http://arxiv.org/abs/1603.08155>.
- [39] K. Simonyan and A. Zisserman, *Very Deep Convolutional Networks for Large-Scale Image Recognition*, arXiv:1409.1556 [cs], Apr. 2015. DOI: 10.48550/arXiv.1409.1556. Accessed: May 10, 2026. [Online]. Available: <http://arxiv.org/abs/1409.1556>.
- [40] Zhou Wang, A. Bovik, H. Sheikh, and E. Simoncelli, “Image quality assessment: From error visibility to structural similarity,” *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, Apr. 2004, ISSN: 1057-7149, 1941-0042. DOI: 10.1109/TIP.2003.819861. Accessed: Jun. 3, 2026. [Online]. Available: <https://ieeexplore.ieee.org/document/1284395/>.
- [41] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, *The Unreasonable Effectiveness of Deep Features as a Perceptual Metric*, en, Jan. 2018. Accessed: May 29, 2026. [Online]. Available: <https://arxiv.org/abs/1801.03924v2>.
- [42] I. Loshchilov and F. Hutter, *Decoupled Weight Decay Regularization*, arXiv:1711.05101 [cs], Jan. 2019. DOI: 10.48550/arXiv.1711.05101. Accessed: May 10, 2026. [Online]. Available: <http://arxiv.org/abs/1711.05101>.

A

Additional Rendered Images

This appendix provides additional rendered images supplementing Chapter 4.

A.1 Effects of rasterized input buffer

Additional comparisons between models trained with and without the rasterized input buffer are shown here. Figure A.1 compares models with only FFT loss, Figure A.3 compares models with only GDL loss, and Figure ??.

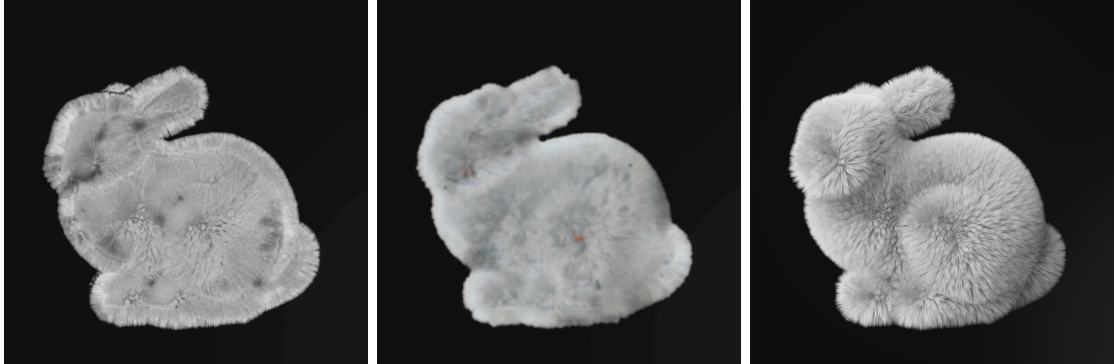


Figure A.1: Effects of the rasterized buffer for model with only FFT loss. From left to right, Exp03, Exp04 and the ground-truth, see Table 4.1.

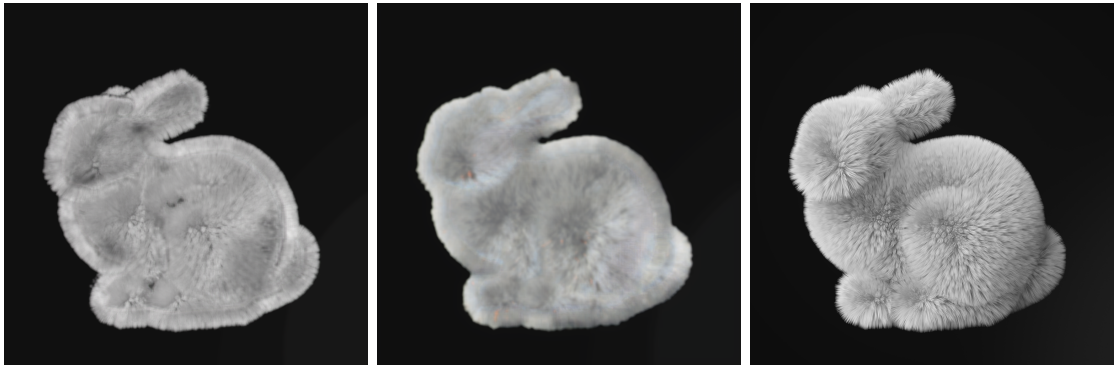


Figure A.2: Effects of the rasterized buffer for model with only GDL loss. From left to right, Exp05, Exp06 and the ground-truth, see Table 4.1.

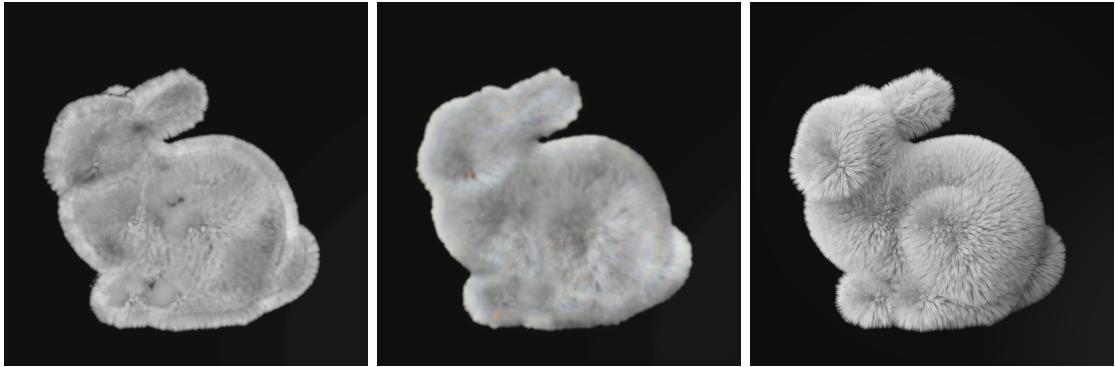


Figure A.3: Effects of the rasterized buffer for model with only GDL loss. From left to right, Exp05, Exp06 and the ground-truth, see Table 4.1.

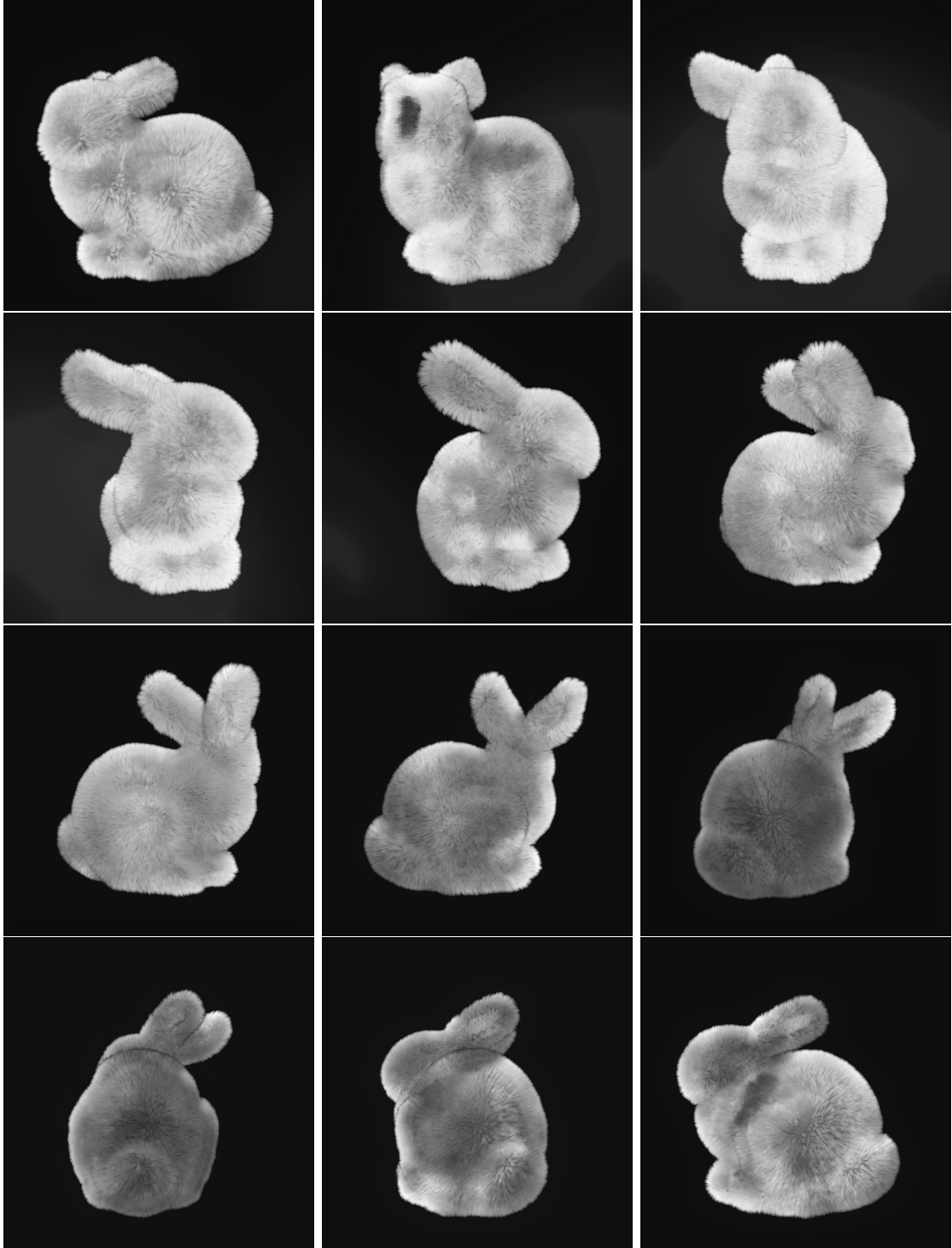


Figure A.4: Twelve frames sampled from the fully rendered `bunny_3Lights_rotateUp` scene using Exp13 from Table 4.1, with one frame shown every 30 frames. The second frame exhibits noticeable visual artifacts in the form of black spots.

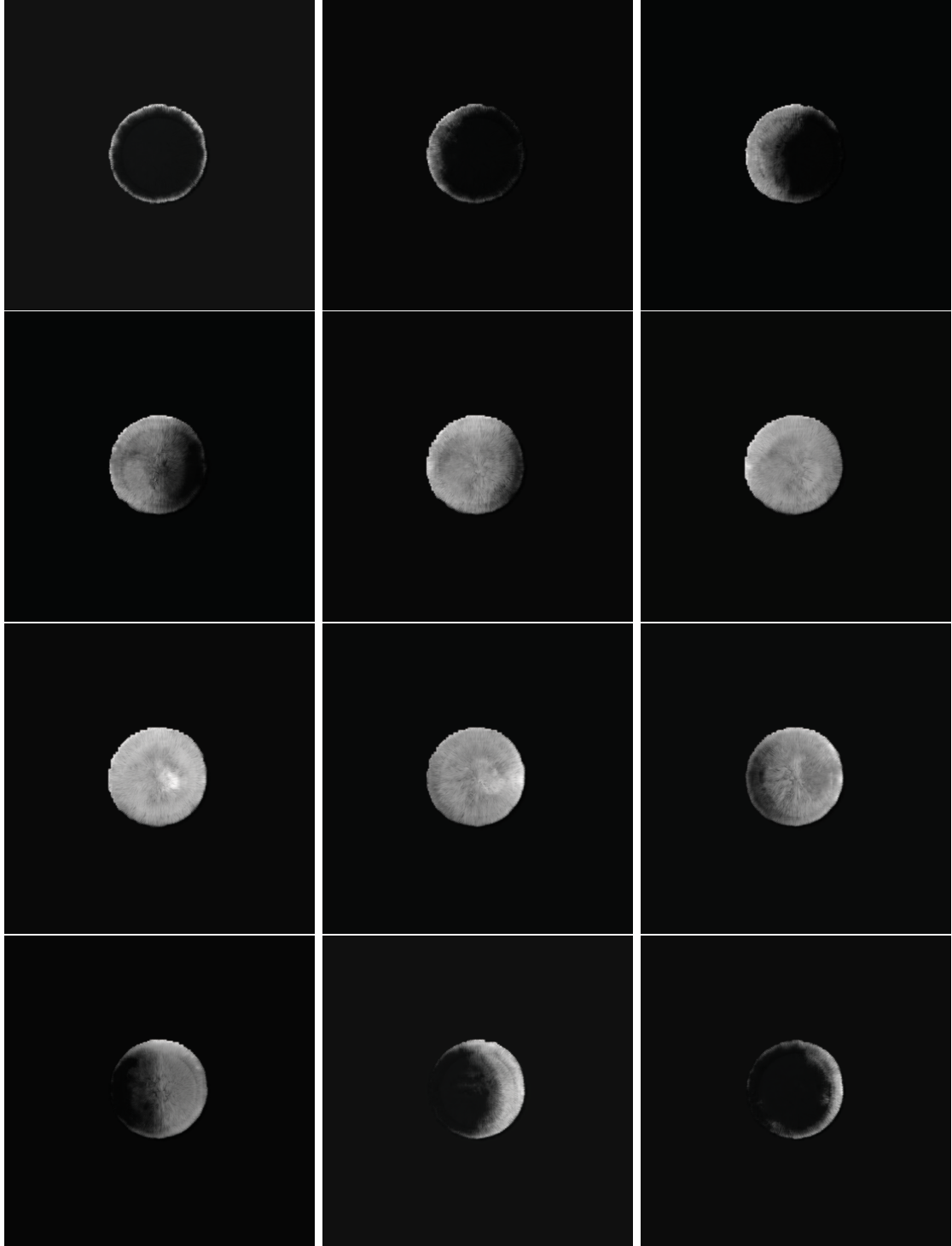


Figure A.5: Twelve frames sampled from the fully rendered `sphere_rimLight_static` scene using Exp13 from Table 4.1, with one frame shown every 30 frames

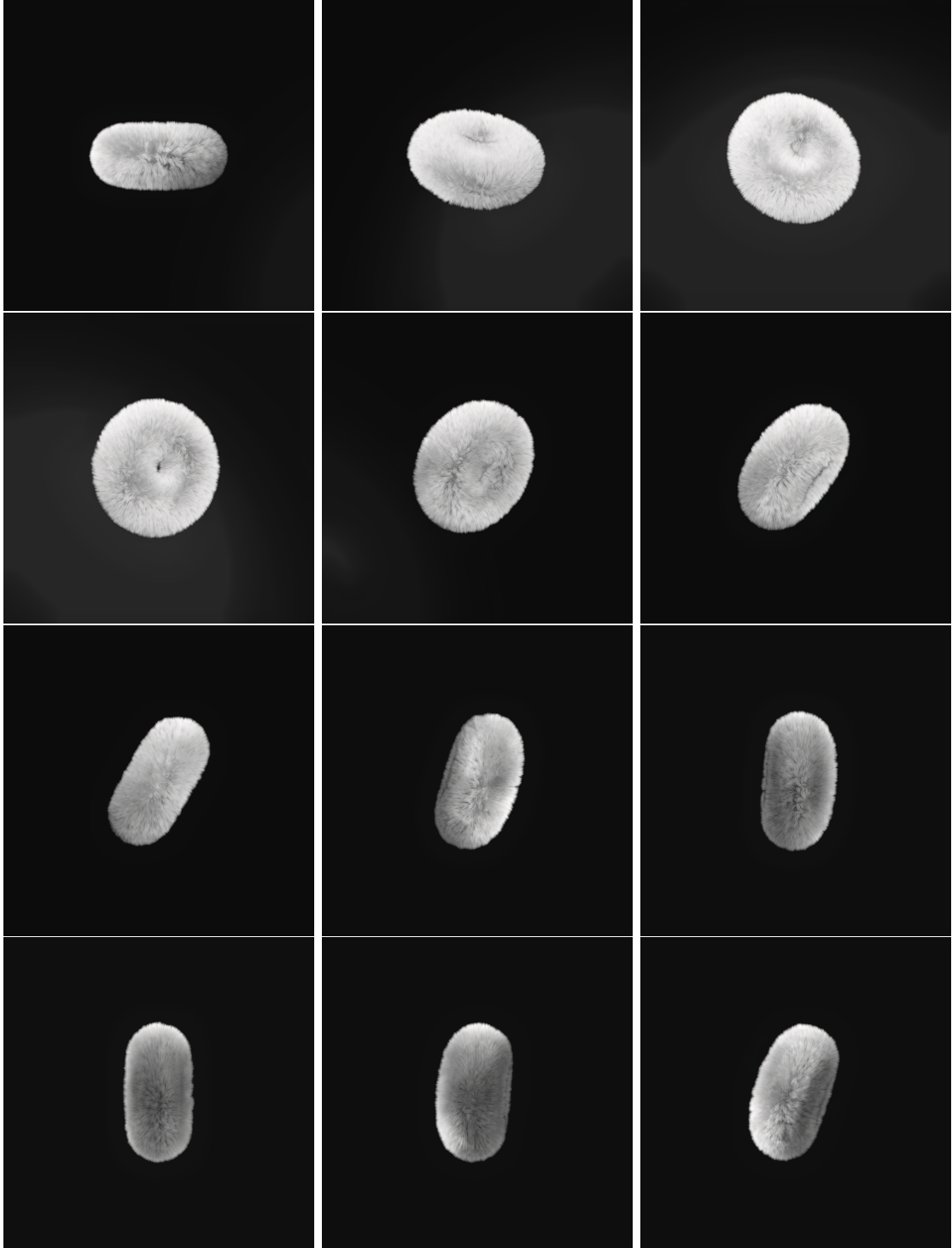


Figure A.6: Twelve frames sampled from the fully rendered `torus_3Lights_rotateRightForward` scene using Exp13 from Table 4.1, with one frame shown every 30 frames