Title

Neural Reconstruction for Real-time Rendering

Permalink

https://escholarship.org/uc/item/5kw81908

Author

Thomas, Manu Mathew

Publication Date

2022

Copyright Information

UNIVERSITY OF CALIFORNIA
SANTA CRUZ

NEURAL RECONSTRUCTION FOR REAL-TIME RENDERING

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTATIONAL MEDIA

by

Manu Mathew Thomas

March 2022

The Dissertation of Manu Mathew Thomas
is approved:

Associate Professor Angus G. Forbes, Chair

Professor James Davis

Assistant Professor Adam M. Smith

Peter Biehl
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

List of Tables

Abstract

Neural Reconstruction for Real-time Rendering

by

Manu Mathew Thomas

Recent advancements in ray tracing hardware have shifted video game graphics towards more realistic effects such as soft shadows, reflections, and global illumination. These effects are achieved by tracing light rays through the scene, accumulating visibility and illumination components. However, due to the real-time constraints inherent in a game, we limit the number of rays/samples in a scene, causing visual artifacts, including aliasing and noise. A number of existing techniques take advantage of frame-to-frame coherence and reconstruct an image from a few samples spread over multiple frames but require the construction of handcrafted heuristics for the accumulation of samples. This results in ghosting artifacts, loss of detail, and temporal instability.

While machine learning-based approaches have shown promise in image reconstruction for offline rendering, they are expensive for games and other interactive media. Using reduced-precision arithmetic to quantize the neural networks can drastically reduce both their computation and storage requirements. However, the use of quantized networks for HDR reconstruction can cause significant quality degradation.

This thesis explores the design space of neural reconstruction methods, considering tradeoffs between quality, complexity, and performance. Our work introduces QW-Net, a neural network for HDR image reconstruction in which 95% of the computations are performed with 4-

bit integer operations. We then demonstrate the capability of this network for supersampling and denoising tasks suitable for real-time rendering. Finally, we combine the idea of supersampling and denoising into a single network amortizing the cost of separate passes. In addition, we also perform super-resolution to further reduce the overall rendering cost. Our network outperforms the state-of-the-art real-time reconstruction techniques in term of quality and performance.

This dissertation is dedicated to my parents,

Dr. K.M Thomas and Dr. Susan Thomas,

for teaching me patience, passion and perseverance can conquer any mountain.

Acknowledgments

I would like to thank:

- My advisor Prof. Angus Forbes, for giving me the courage to explore unknown paths and guiding me throughout the journey to reach my goal.
- My committee members, Prof. James Davis and Prof Adam Smith, for giving great feedback throughout the process that helped me shape my ideas.
- Intel Advanced Rendering Technology (ART) team, especially my mentor Karthik Vaidyanathan for being so kind and patient with me during all the late night brain storming sessions, and my manager Chuck Lingle who made sure that I had proper resources to do my research.
- My collaborators Gabor Liktor and Sungye Kim for the support and feedback they have given me.
- Fellow researchers from the Creative Coding Lab for all the academic and personal help and for the great discussions during lab night dinners.
- Epic Games for the Unreal Engine and the demo scenes, without which my research wouldn't be complete.
- My sister and brother-in-law for supporting all my risky decisions along this journey.
- My niece for entertaining me during tough days.
- My parents for all the moral support they have given me from day one.

Chapter 1

Introduction

The human brain is a highly complex system capable of classifying an experience as real or simulated. Previous experiences and expectations greatly influence visual perception guided by sensory information gathered from photons that fall on the retina. [76]. To create digital media that provide an engaging experience, we need to have a fundamental understanding of how the human visual system works. Essentially, our job as media creators is to control the millions of pixels on the screen in a way that results in meaningful content.

Digital image processing and computer graphics are the two main fields of computing responsible for creating, enhancing, and analyzing visual media. Digital image processing primarily involves acquiring data from a physical sensor and transforming it into human readable form. In each step of the transformation process, we employ knowledge about human perception to design software and hardware to produce memory-efficient, high-quality images and display them as fast as possible. For instance, JPEG, the most commonly used image format, is a form of lossy compression [84]. JPEG discards high-frequency components and exploits the fact that human eyes are more susceptible to luminance than color changes. Contrary to digital image processing, computer graphics uses mathematical models as a basis for synthesis of visual content.

In 3D graphics, a scene description contains the position relative to a coordinate space, material attributes of objects that make up the scene, and light sources' details. Typically, when modeling real-world objects, we take into account the macroscopic properties, such as surface details, but ignore atomic-level details because humans cannot perceive such minute details. Likewise, our light simulation model uses ray optics principles without accounting for optical effects such as diffraction and interference which are less common than reflections and refractions. A further approximation of scene details or light propagation may be necessary with respect to the nature of the media application. For video games, we limit the detail in character models and fake the lighting for better performance, while for movies, we try to add as much detail as possible. This dissertation discusses techniques to create plausible renditions in real-time games instead of running cost-prohibitive light simulations.

Rendering is the process of computing color at each pixel by solving a multi-dimensional integral known as the rendering equation [35]. It is very expensive to evaluating this integral, but it is crucial to obtaining a physically-based high-quality photorealistic image. Approximation techniques usually compromise image quality for speed, resulting in bias in the result. The commonly used fast approximation rendering algorithm in real-time graphics is known as rasterization [86, 26]. To rasterize a scene, we project triangles that make up the geometry to 2D pixel space and mark the pixels that are covered by each triangle. Using material properties and light information from the scene description, we determine how to color each pixel. Each triangle get processed in isolation by dropping all the integrals to simplify the rendering equation. The dedicated hardware projects millions of triangles in parallel, enabling real-time rendering. Due to the lack of information about other triangles in the scene when processing triangles individually, this method is limited in its ability to create shadows, reflections, and other complex light interactions. Approximation tricks are used in order to mimic such effects. For example, shadow maps are created for each light source in order to determine which portions of the scene are in shadow. For indirect illuminations, we bake the light coming from all directions at many points in the scene in an offline step, however, this increases our memory usage as well as causes light leaks. Creating stochastic effects like motion blur and depth of field requires screen space post-processing techniques. A critical constraint of real-time rendering is sampling the continuous scene function to represent it in discrete pixel space [22, 13, 51]. We sample only once per pixel in video games to achieve high frame-rate and trades image quality. With a small sample rate, it is difficult to get all of the edge information of a triangle, resulting in aliasing artifacts in both spatial and temporal dimensions [92].

Ray tracing is a rendering technique that traces rays through the 3D scene and captures the light reflected from the object in a physically based manner[87, 19]. Ray tracing or its variant called path tracing [35] stochastically sample the integral to estimate the rendering equation. In simpler terms, with ray tracing, we trace many rays per pixel and follow the ray's path until it hits a light source or satisfies a termination criterion. Since this technique is physically based, we naturally get light effects such as soft shadows, global illumination, and glossy reflections. A significant disadvantage is that we need a lot of rays or samples to converge to the actual color of the pixel. Otherwise, we end up with a high variance, which visually translates to noise in the final image [98]. The low sample restriction of real-time renderers makes it challenging to use ray tracing for games.

The recent advancements in graphics cards have led to dedicated hardware blocks being designed for accelerating ray tracing for real-time performance [11]. Using this specialized hardware, the current generation of games are able to combine ray tracing effects and raster rendering. Presently, In order to achieve 30 frames per second at 4K resolution, games trace 1 ray per pixel for effects and 0.5 rays per pixel (half resolution) for demanding workloads, leaving considerable noise in the image and negatively affecting the gameplay experience..

Many analytical and statistical reconstruction techniques are available for supersampling, super-resolution and denosing tasks, but they do not achieve results similar to the offline reference, and suffer from visual artifacts, including flickering, ghosting, and over-blurring. For offline rendering, AI denoisers are already incorporated into production renderers for film and visual effects (VFX). For Disney's animation movies, their Hyperion renderer now employs Machine Learning (ML)-based denoisers, instead of their traditional non-local mean denoisers [7, 81]. The open image denoise [2] and Optix denoisers [14] target vfx and archviz renderers with interactive speeds. Because these renderers are not required to meet real-time frame budgets, they can be designed with a heavy network that takes several seconds or minutes to process a single frame. Offline contents generally render the base image with 128 or 256 samples per pixel, which gives the denoiser more information to clean up the image. The learning-based denoisers produce better results than non-ML denoisers since the parameters are optimized according to a loss function defined over a high sampled reference.

In light of the success of ML denoisers in production renderers, and motivated by our own observations(Appendix B, Appendix C), we explore the possibility of developing a neural reconstruction method that would achieve real-time inference speed and produce images similar to ground truth while avoiding reconstruction artifacts commonly seen in modern games.

1.1 Thesis Statement

Our thesis identifies the main challenges with real-time neural reconstruction techniques and discusses the potential applications of quantization as a tool for achieving faster inference for these networks. As our main contribution, we introduce a unified quantized network that combines supersampling, super-resolution, and denoising to enhance the experience of raytraced games. In comparison to existing reconstruction techniques used in games today, our proposed approach shows a significant improvement in image quality.

1.2 Contributions

This thesis makes the following major contributions:

- We present a novel neural network architecture for High Dynamic Range (HDR) image processing, which enables image quantization to INT4[1] precision without any loss of image quality. We show this approach is used for real-time, temporally stable supersampling.

- Our method is further extended for denoised ray traced effects, such as soft shadows, reflections, and global illumination. We also demonstrate how kernel weights can be

[1] 4-bit integer representation

generated for the different lighting and texture components by sharing features from a single network. Additionally, we present a custom perceptual loss network based on photorealistic rendering images that extracts useful additional features compared to natural image-based perceptual loss functions.

- We present a joint denoising and supersampling solution that runs in real-time and produces temporally stable results with significantly better image quality than state-of-the-art techniques currently used in games.

1.3 Outline

The remainder of the thesis is organized as follows. We start by reviewing the basics of light transport, game graphics pipeline, and current state-of-the-art machine learning techniques for graphics in Chapter 2. In Chapter 3, we describe our proposed network architecture for image processing that utilizes quantization, achieving real-time performance without significant image quality degradation. Chapter 4 explains our extension to the quantized network architecture for denoising ray traced effect in real-time. In Chapter 5, we show our unified architecture that combines denoising, supersampling and super-resolution into a single network that produces results close to highly sampled ray traced reference within a real-time budget. Finally, we conclude the thesis in Chapter 6.

Chapter 2

Background

In this chapter, we cover the essential background knowledge needed to delve into the rest of the dissertation. We begin by reviewing the basic concepts of light transport, Monte Carlo rendering, and real-time ray tracing. We then introduce the theory and applications of temporal projection. Finally, we discuss the state-of-the-art machine learning techniques used in rendering and introduce quantization of neural networks.

2.1 Light Transport

The goal of rendering algorithms is to solve the light transport equation, which is also known as the rendering equation [35]:

$$L_o(x, w_o) = L_e(x, w_o) + \int_{\Omega} f(x, w_i, w_o) L_i(x, w_i)(w_i . n) dw_i \tag{2.1}$$

where $L_o(x, w_o)$ is the outgoing radiance from a surface point x in the direction w_o,

$L_e(x, w_o)$ is the radiance emitted from the surface point, $f(x, w_i, w_o)$ is the Bidirectional Distribution Function (BRDF) describes how much light is reflected from a given direction into another direction at a given position. The $\int_\Omega$ represent the hemisphere defined over the point x where the incoming light is sampled. The recursive part of the integral provides a nature solution to indirect effects.

The rendering equation cannot be solved analytically due to the complex nature of the integral. Monte Carlo numerical is used to evaluate the integral by taking random samples from within the integral domain and averaging them. The downside of this techniques is that we require a large number of samples to estimate the integral since the convergence rate is of order $O(n^{-1/2})$. In order to reduce the error by half, we will need to take 4x the number of samples.

In Monte Carlo rendering (path tracing), rays are shot from the camera to the scene. The starting rays correspond to samples in the pixel space, and we take more samples covering more regions in the scene, resulting in a supersampled image. The samples are drawn from a well-studied distribution (usually low-discrepancy sequence) to spread evenly [18]. A ray hits a surface and a new ray is shot in a direction determined by a sampling strategy, and so on until we hit a light source. The radiance from each hit contributes to a pixel's final color.

Regardless of how many samples are used, Monte Carlo rendering produces unbiased output that only contains variance. Variance appears as noise in pixel space. Different variance reduction techniques are available, such as important sampling [80, 40] and path guiding [83, 52, 30].

2.2 Real-time Ray Tracing

Real-time ray tracing follows a hybrid approach between traditional rasterization and ray traced effects (shadows, reflection and global illumination). The first intersection point in the scene can be found from the G-Buffer [71] from which rays are traced into the scene. Generating shadows begins with finding the world-space position of the surface reconstructed from the depth buffer. For hard shadows, a single ray is traced toward a light source, and if the ray hits an object, the surface point is in shadows. For soft shadows, a number of rays are shot following a uniform cone sampling where the cone angle determines the softness of the penumbra. A spatiotemporal filter is required to clean the noise after tracing the shadow rays. Rays are traced similarly for reflections but follows BRDF importance sampling [80]. For better performance reflections are traced half resolution and reconstructed in the filtering step. Ray traced reflections are only performed for certain objects in the game and screen space reflections are added to bring addtional details. Global illumination (GI) or indirect lighting accumulate light contribution from the nearby diffuse surfaces. To produce GI effect rays are traced with a cosine distribution of the hemisphere over the surface point.

2.3 Temporal Reprojection

The idea behind temporal reprojection is to resolve subpixel details by distributing jittered samples over many consecutive frames and accumulating them to get a supersampled result. This technique exploits the fact that the geometry and shading from one frame to the next are minimal [92, 60]. A number of real-time applications utilize this technique, such as antialiasing, denoising, and upsampling. The algorithm works as follows:

- A jitter offset is used to shift the viewport to distribute samples in the pixel area during rendering.
- For every pixel in the current frame I, we find the corresponding location in the previous frame I-1 after compensating for the scene motion.
- The location in the previous frame is resampled to get the history color.
- A history validation is done to check for disocclusion, surface consistancy and illumination changes. If the history data is valid then it is blended with the current sample otherwise it is rejected.
- An optional history rectification step modify the rejected history samples into a more consistent form and blended with the new sample.

Jittering samples are generated by adding an offset to the projection matrix of the camera. The spread of the samples is dictated by the quality of the offset. Usually they are drawn from a well-distributed sample sequences such Halton sequence [57]. To compensate for scene motion, velocity vectors are computed from transforming the geometry using the current and previous projection matrix and storing the difference as texture. During reprojeciton, this vector is used to locate the position in the previous frame for resampling. The samples are accumulated using an exponential moving average instead of storing all the sample information from previous frames. A weight parameter determines whether the previous or the current sample should contribute more to towards the pixel color. For example, in the case of denoising if the current sample is a firefly that should be suppressed then the previous accumulated sample will get more weight. The weight can be fixed or adaptive depending on the variance or other user defined criteria. If we pick a poor weighting strategy, we end up with temporal lag and blurry frames.

Due to illumination changes or disocclusion, the temporal data can become invalid and it should be rejected or rectified. For occlussion changes comparing the G-buffer data such as normal, depth and object ID can be used to determined if the previous sample is valid for accumulation. In the case of illumination changes, statistics on the neighbouring pixel colors provide a hint as to whether the previous sample should be accumulated. The history rectification works around the idea of clipping and clamping the sample color with the help of either a AABB built using the neighbour pixels (3x3) or using a variance clipping.

Various visual artifacts may be introduced by the temporal reprojection process due to scenes changing rapidly or resampling and/or rectification errors [91]. A one to one mapping of the history sample is very unlikely, so some kind of filtering is needed to obtain the sample color at the fractional pixel location. Consequently, the high-frequency components are removed from the image, resulting in significant blurring. The filter error accumulates over time and produces a significantly blurred image. Due to the aggressiveness of history rectification, details thin wires and grass objects are lost. Ghosting appears when new objects are brought to the scene as a result of other moving objects. However, when rejection and rectification fail, we see a second version of the moving object blended with the disoccluded one.

Despite the shortcomings of temporal reprojection, it is still powerful in accumulating samples from history frames helping in antialiasing and denoising tasks. When coupled with machine learning based techniques, the rejection and rectification strategies are also learned based on the loss function over reference.

2.4 Machine Learning for Rendering

In recent years, machine learning has reshaped a large portion of graphics sub-domains due to it's ability to learn complex non-linear functions and creates plausible results that are sufficient for majority of the use-cases. As we plan to discuss neural techniques for reconstruction throughout the thesis, we will now explore other areas of neural rendering.

Simulations involving particles for smoke, water or fire requires computationally heavy physics calculations. A common approach in simulation rendering is to compute the physics on the CPU and then transfer the data to GPU for rendering. Physics calculation can take hours or days depending on the variables and complexity. Rendering takes similar amount of time if physically based renderers are used. Unless the simulation is targeted for scientific research, we can deviate from the actual representation and make the computation faster.

Xie et al. [88] introduced tempoGAN, the first approach to synthesize four-dimensional physics fields with neural networks. A lower resolution fluid simulation is faster to compute producing a coarse appearance while a higher resolution version of the same fluid will looks detailed and realistic but the time for computation grows linearly. The key observation in this work is that a neural network can be used to infer a high-resolution configuration from a low resolution data. The main contribution of this work includes a novel temporal discriminator providing a better temporal coherence, a feature loss for better spatial details and using velocity as the input to make the learning process simpler. The generator network takes a low resolution density field as the input and tries to predict its higher resolution version. The predicted density field is given to a discriminator which classify as real or fake density field. During training the generator learns to fool the discriminator and at the same time discriminator learns to classify correctly.

While tempoGAN was designed for realistic fluid simulation, Kim et al. [39] presented a transport-based neural style transfer network for smoke simulations giving artistic control over fluids. As opposed to tempoGAN, here the input is the velocity vector, which facilitates stylization since the directional information is pre-encoded. This approch make use of the pre-trained CNNs used in style transfer for images. The 2D multiview images of smoke simulations are passed to the pre-trained CNNs to analyze for style and content loss. Using a differentiable renderer, the gradient flows to the 3D volumetric structures. This method takes 10 minutes per frame for 200x300x200 voxels.

A rapidly growing area is neural representations for rendering. Neural primitive representation using a fully connected network captures more details in a compressed format than traditional mathematical representation [50, 54]. NeRF essentially converts a discrete sampling function to a continuous function providing a smoother result. Granskog [27] explored creating a neural scene graph where neural transformation and neural light description are applied to create scene representations.

2.5 Kernel Prediction Network

Kernel prediction network are neural networks that produces a series of weights instead of the final output. The weights are then applied onto the input to obtain final result. Schmidhuber et al. [75] proposed a two network approach for sequence learning where the first network learns the output weight change for the second ”fast network” that serves as a short term memory block. Essentially they store temporal information in the form of fast weight rather than having a separate storage blocks. David et al [28] introduces the concept of hypernetwork where a small network (hypernetwork) generate weights for the main larger network. While training for each layer in the main network, the weight are converted into a embedding vector that goes into the hypernetwork. The hypernetwork outputs the new weight for the layer and thereby reducing the search space. Mildenhall et al [49] used a kernel prediction network to predict a per pixel kernel weights for denoising camera handheld photos. The kernel predicting network for images are observed to have faster convergence, no ringing artifacts and avoids color shifts commonly seen in direct prediction networks.

2.6 Quantization for Neural Networks

A common practice to accelerate performance is by applying quantization [93]. The idea is to convert float values to integers and perform the operations on faster hardware. This is mostly used in embedded devices without float point hardware. As an example (Fig 2.1), we could instead of using the float operations to find the product of float values, convert floats to integers using a mapping that we define and take advantage of faster integer hardware to perform the exact calculation. Our final step is to convert the result back to floating point. Ultimately we trade precision for speed.

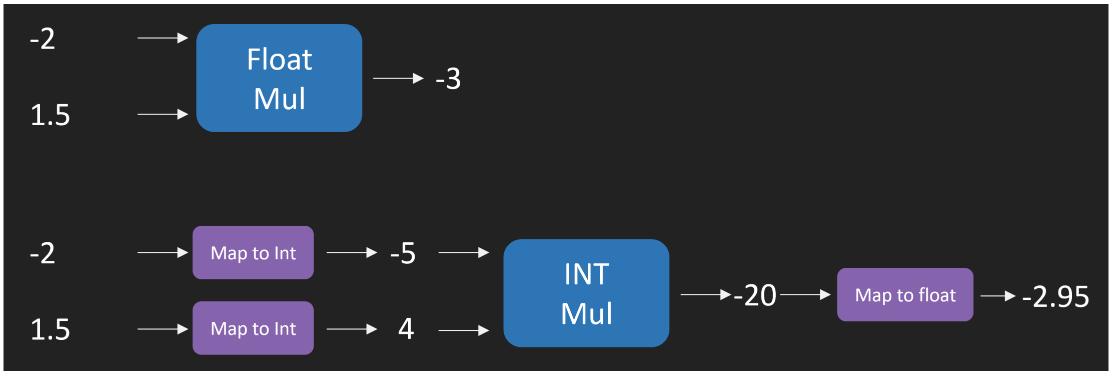

Figure 2.1: An example of a product operation using quantization

In neural networks, the weights and activation parameters can be quantized so that the convolution operation can be performed on integer hardware as shown in Figure 2.2. The result is dequantized before passing to the next layer. After training, the quantized weights are fixed and can be fused with linear operations to further improve performance.

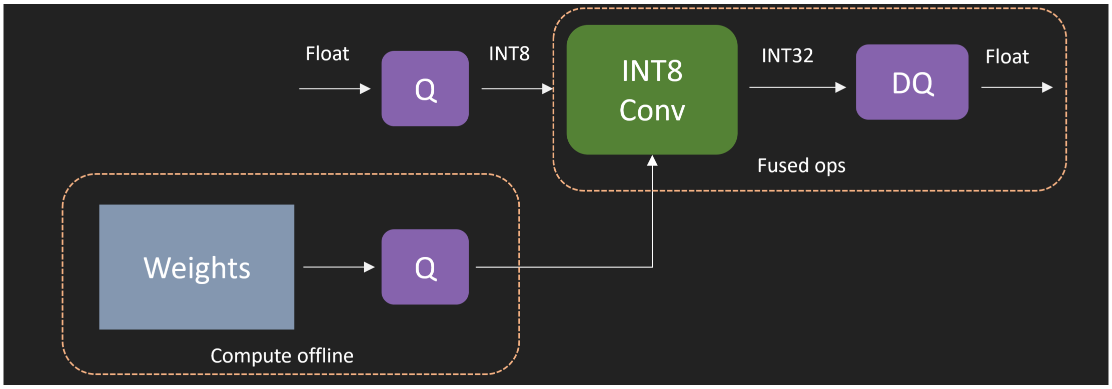

Figure 2.2: Quantization applied on weights and activation of a nerual network

A quantization scheme is defined by the selection of:

- Mapping - Symmetric with linear mapping and zero point mapped to zero or asymmetric with affine mapping.
- Granularity - Quantize per layer or per channel.
- Range - Maximum range if the data is evenly distributed or tighter range if the data is clustered around some point giving more precision

A full-precision network can be quantized either post-training or by training the network with simulated quantization. With *post-training quantization*, a network is quantized using the distribution of trained weights and by measuring the distribution of activations on a sample dataset, a process known as *calibration*. For example, TensorRT [48] quantizes weights and activations by minimizing the Kullback-Leibler (KL) divergence between the quantized and un-quantized distributions. TensorFlow [32] on the other hand maps the range of the weight tensor and the average range of the activations computed over several batches, to the range of the quantized format. Unfortunately post training quantization shows as significant degradation in accuracy with 4-bit integers (INT4) and lower precisions [32, 41].

Jacob et al. [23] proposed an alternate *quantization-aware training* approach that simulates quantization errors during training, achieving significantly better accuracy. Simulated quantization introduces quantization errors in the forward pass by quantizing the weights and activations and dequantizing them back to the original range. However, in the backward pass, weights and biases are updated with floating point precision without any loss in precision.

The choice of threshold values for quantizing weights and activations greatly impacts the accuracy of the network. While Jacob et al. [32] derived the quantization thresholds based on a measurement of the tensor range, Jain et al. [33] and Esser et al. [23] showed that the quantization thresholds could be trained to further improve accuracy.

They derived the gradient of the quantization function applying a *straight through estimator* [10] for the gradient of round/ceil operations but without approximating these operations with an identity function. This allowed the thresholds to grow (favoring larger dynamic range) or shrink (favoring higher precision) based on the gradients.

A majority of existing work on reduced precision networks use a uniform quantization scheme where the quantized values are evenly spaced [53, 95, 44, 96]. A uniform quantizer can be further classified into *asymmetric* and *symmetric* quantizers. An asymmetric quantizer applies an affine transformation with a scale and a zero-point to map values in the floating-point range to the integer range. In a symmetric quantizer, the zero point is set to 0 reducing the affine mapping to a linear mapping. The symmetric quantizer avoids the overhead of handling zero-points making it computationally efficient. A large body of work explores binary and ternary neural networks, where the weights and/or activations are quantized to binary or ternary values [20, 21, 62, 97]. These networks push quantization to its limits but also lose a significant amount of accuracy.

NVIDIA introduced a dedicated chip called *Tensor Core* for ML inference with the Volta GV100 GPU. An enhanced version of *Tensor Core* made its way into consumer level Turing architecture GPUs [38]. The Turing *Tensor Core* includes INT8 and INT4 precision modes supporting quantized networks. *Tensor Core* follows a systolic architecture favoring fast matrix-matrix multiplication. Availability of *Tensor Core* in GeForce gaming GPUs makes it feasible to use deep learning techniques in game applications.

The new Xbox Series X supports machine learning for games through *DirectML*, a ML stack of its graphics API *DirectX*. Xbox Series X have over 97 TOPS (Trillion Integer Operation per Second) of 4-bit int performance. Microsoft has already started testing ML based SDR to HDR conversion for older games but implementation details are unknown at this moment.

2.7 Evaluating Reconstruction Quality

The reconstruction quality in real-time rendering is measured by comparing similarity with reference image rendered with high sample count unbiased production quality pathtracer. The two commonly used quality metrics are Peak Signal to Noise Ratio (PSNR) and Structural Similarity (SSIM). The PSNR is used in digital signal processing to determine the quality of transmission. It represents the maximum power of a signal vs the maximum power of corrupting noise. It is often used to measure the fidelity of transmission of digital signals. The downside of psnr is that it is a pixel to pixel comparison and does not accurately reflect the human perception. SSIM on the other hand is a perceptual metrics that penalizes for perceived changes in the structural information. SSIM is the product of 3 components: luminance, contrast and structure.

Chapter 3

Reduced-Precision Network for Supersampling

The first challenge towards building a neural reconstruction technique is to find an image processing network architecture that can run in real-time and produces high-quality images. In this chapter, we first introduce the problem of aliasing and reconstruction in graphics, then we discuss legacy and state of the art reconstruction techniques used in video games. Finally, we present our work towards a quantized real-time learned solution for image reconstruction tasks without any degradation in image quality as shown in Figure 3.1.

3.1 Introduction

Synthesizing computer-generated imagery involves two key functions: sampling the incident radiance on an image plane and applying a reconstruction filter to produce the final image [19]. If the incident radiance is undersampled, high-frequency components resulting from

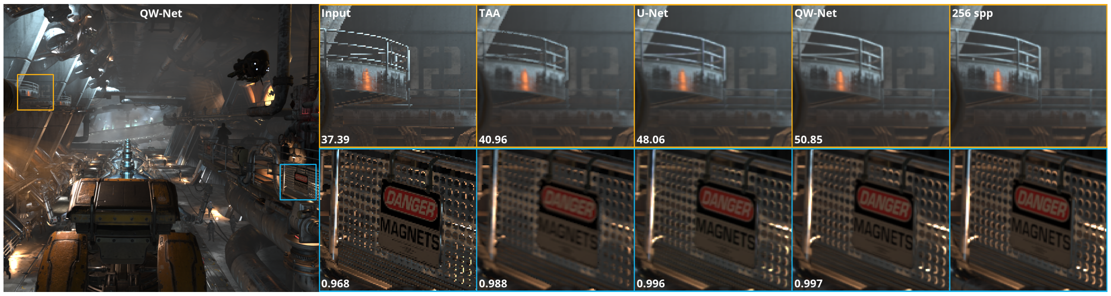

Figure 3.1: A comparison of different reconstruction techniques on a frame from the INFILTRATOR scene with PSNR (top row) and SSIM (bottom row) metrics. TAA has the worst reconstruction quality with excessive blurring followed by direct prediction with U-Net. QW-Net (ours) produces the best results with quality comparable to brute force sampling with 256 samples per pixel.

visibility discontinuities or specular lighting can appear as aliasing in the reconstructed image. To reduce aliasing, production renderers typically apply supersampling with a large number of samples per pixel. On the other hand, real-time applications trade image quality with high frame rate and resolution. As a result, video games always suffered from aliasing artifacts which commonly manifests as jagged edges in the spatial domain and as flickering in the temporal domain. This led to the development of specialized image reconstruction techniques to suppress the aliasing effect in real-time scenarios. Temporal antialiasing (*TAA*) [92, 34, 37] is an image reconstruction technique that is widely used in games today. TAA uses renderer-generated motion vectors to gather and accumulate samples from previous frames, effectively increasing the number of samples per pixel. However, TAA is susceptible to ghosting artifacts, loss of detail, and temporal instability [90].

Convolutional neural networks offer a promising alternative for image reconstruction, and have been successfully applied to closely related fields such as denoising [8, 15, 82]. The U-Net [65] architecture is particularly well suited for such tasks as it processes features at

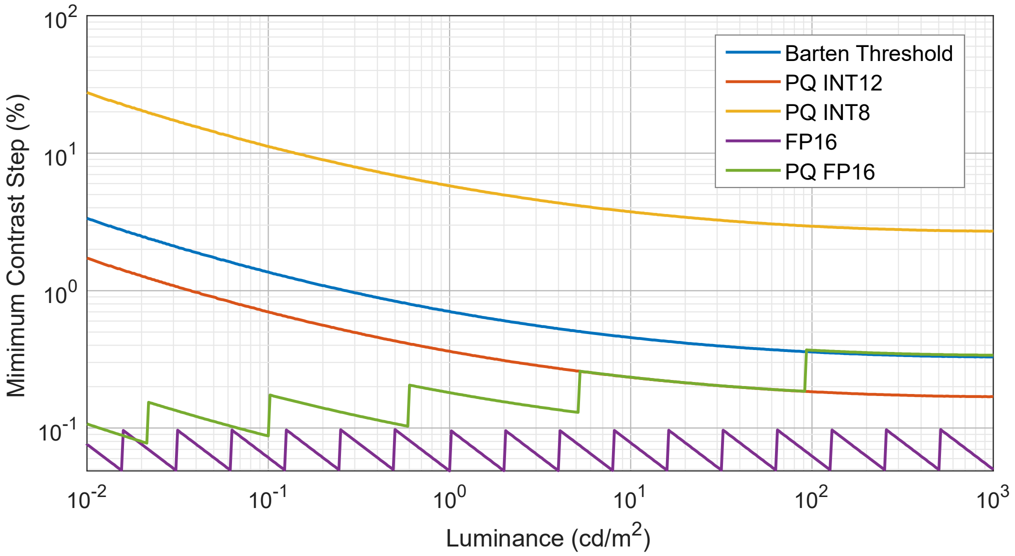

Figure 3.2: Minimum contrast step for a 1000 nit display with different numeric formats. The step size with 8-bit integers exceeds the human contrast sensitivity threshold even with perceptual quantization (PQ) to minimize the perceived error [77].

multiple scales, achieving a large receptive field and more effective filtering with a relatively low computational cost.

With the hardware acceleration of 8-bit and 4-bit tensor computations on GPUs [56], quantizing the weights and activations of such a network can be key to achieve real-time performance, without compromising its expressive power. However, quantization errors can severely impact the image quality, especially with high-dynamic-range content. We can see from Figure 3.2 that the minimum contrast step with 8-bit integers, far exceeds the human contrast sensitivity threshold [9] on a high-dynamic-range display. In the case of 4-bit integers, the available precision is insufficient even for standard-dynamic-range displays.

We introduce a novel network called QW-Net that addresses this issue by using a combination of two networks, a feature extraction network and a filtering network. The feature extraction network is based on U-Net and can be quantized to 4-bit integers as feature detection is more resilient to quantization errors. The filtering network is another U-shaped network that uses the extracted features at each scale to predict filters. The filtering operations require a higher precision but involve significantly fewer computations than the feature extraction network. The name QW-Net is motivated by the coupled U-shaped networks resembling a W, as well as the quantized nature of our feature extraction network.

Our network combines the advantages of U-Net and kernel prediction networks [8]. It achieves a large receptive field with just a few layers, similar to the U-Net, while avoiding artifacts like color shift and requiring significantly less training time, similar to kernel prediction networks [82]. Moreover, our network temporally refines the reconstructed image by recurrently warping and accumulating previous frames.

Although our motivation for using 4-bit convolutions is improved inference performance, realizing the full potential of a quantized network requires extensive exploration of various implementation aspects such as tensor layouts, tile sizes, kernel fusion and constraints imposed by the hardware, which are beyond the scope of this work. Our goal is to show the feasibility of a heavily quantized network for image reconstruction using a novel network topology that preserves image quality. To the best of our knowledge, this is the first 4-bit network applied to image processing. We evaluate image quality using simulated quantization and include an analysis of the computational cost and the performance of 4-bit convolutions in section 3.4.2.

3.2 Background

An image can have several different representations. An *analog image* is a 2D region with varying light intensity captured through a film camera. If the image is transmitted, it can be converted into an electrical signal. An *analog image* is both continuous and physical quantity (photograph or voltage). A *descriptive image* is expressed through mathematical expressions, for example $f(x,y) = sin(x+y)$ produces a 2D sine wave pattern. *Descriptive images* are continuous but not physical quantities like analog images. A *digital image* is a finite grid of colors called pixels. They are neither continuous nor physical and are made up of discrete values with finite precision.

In computer graphics, we define a 3D scene using a set of mathematical expressions forming a *descriptive image* and then we convert it to a *digital image* to get the final output image on our display system. The *descriptive image* is continuous meaning we can get the radiance at any granular level but the *digital image* is finite. In order to represent a continuous signal in a discrete format, we take samples at regular intervals and tries to reconstruct the signal back in the digital space. The sampling and reconstruction process involves approximation that introduces error known as **aliasing**. This error occurs due to the inability to capture all the information from a continuous radiance signal.

There are two main strategies to solve the aliasing problem in signal processing:

- **Supersampling** - taking a high number of samples

- **Prefiltering** - eliminating the high frequency components before sampling.

Figure 3.3 shows how sampling once per pixel causes jagged edges. The top left

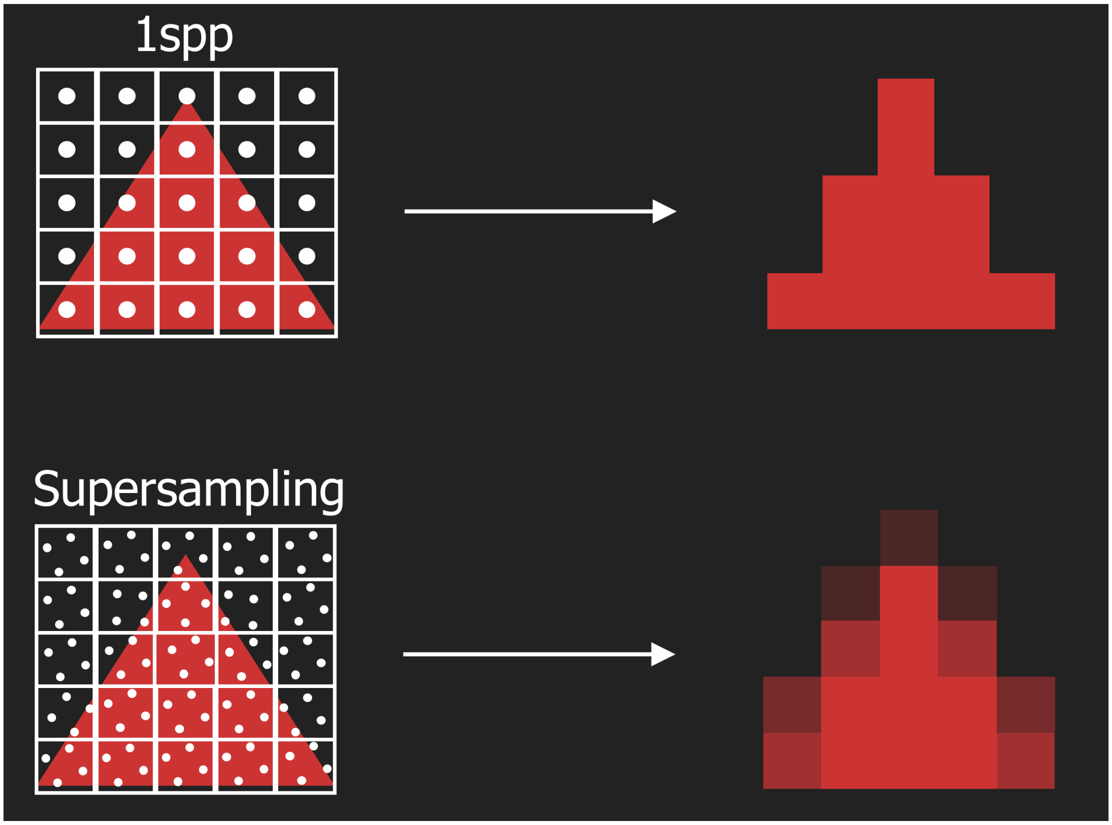

Figure 3.3: Left image shows how sampling is done for each pixel and their results. Right shows an example of aliasing effect seen in a commercial game.

image shows a red triangle that we are trying to reconstruct on the pixel grid. The pixels are shaded red only if the edge is visible to the center sample. The bottom left image shows how taking more samples per pixel (supersampling) can reduce the effect of aliasing. *Supersampling* can improve the jagginess in a scene but it is not enough to completely eliminate it. In fact, no amount of resolution or samples can truly represent an edge in the pixel space as it is not band-limited. During animation or in an interactive scene the jagged edge will crawl through the pixel producing an artifact which look like ants moving across the scene.

Apart from jagged edges, other aliasing issues occur when we want to show small objects (or small part of a bigger object). In one frame the small object is visible to the sample and in the next frame due the scene motion small object misses the sample, this causes a flickering effect. Temporal aliasing or wagon wheel effect cause jerky movements in a game. This is due to the undersampling of the temporal domain. A common solution for temporal aliasing is motion blur which blend one or more previous frames and thereby smoothing the motion.

3.3 Related Work

3.3.1 Prefiltering (Anlytical area sampling)

A pure *prefiltering* is not practical in computer graphics. It requires the elimination of high-frequency information in the target function before sampling. It's not trivial to represent an image function explicitly. This means that we don't have any idea of the objects that cover a pixel or their color before sampling. In fact, sampling is a method we use to query the visibility and color information. The approximation of prefiltering method in graphics translates to finding the intensity (color) over the pixel area whenever an edge passes through it. In this way, we can get a smooth gradient of the edge. The pixel acts as a window to the 3D scene, and each polygon that covers this area contributes to the final pixel color. Prefiltering for visibility is not common but was explored by researchers decades ago [22, 13, 6]

3.3.2 Multi-Sampling Antialiasing (MSAA)

In 1993, a new graphics system was designed by Silicon Graphics Computer Systems known as *RealityEngine* [5]. The system was built for general purpose high-end graphics problem including real-time renderings. It was capable of rendering 1 million antialiased triangles per second. They introduced a simpler version of *supersampling* in which multiple samples are used per pixel for checking triangle coverage but the shading calculation is done only on one subsample (subsample is an individual sample inside a pixel). The result is shared with all the samples per pixel which falls inside the triangle. This technique is called Multi-Sampling Antialiasing (MSAA).

MSAA is built upon the observation that shading operation is expensive when using mulitple samples per pixel but coverage (visibility) test is not. Figure 3.4 shows the working of MSAA. On the left, the triangle is sampled at the center of the pixel but misses the triangle, hence the pixel color becomes background color. On the right, the triangle is sampled using 4 samples and tested for coverage. The coverage results are stored in a bitmask. Apart from coverage, the depth of each sample is also stored for occlusion tests. Shading operation is done only for one sample that is covered by the triangle and the result is shared with all the samples that falls inside the triangle. There are several way to combine the samples to get the final pixel color but the most common way is to use a 1-pixel-wide box filter.

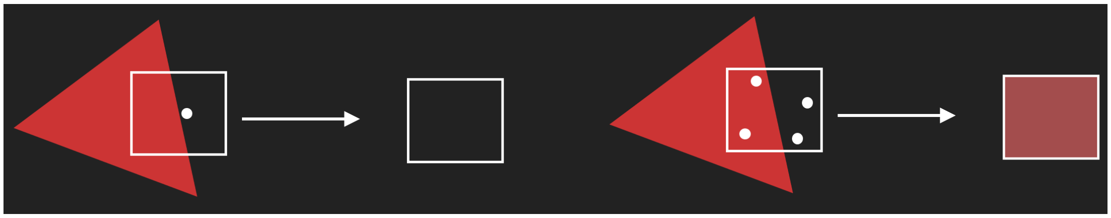

Figure 3.4: Sampling a triangle with 1 sample per pixel (left) and 4 samples per pixel with MSAA (right)

Although MSAA provides good image quality, it's rarely used in games nowadays since it doesn't support modern rendering techniques such as deferred shading. In deferred shading, the geometry is decoupled with the lighting computation. In the two-pass algorithm, the first-pass process only the geometry and stores the relevant information in multiple buffers known as G-buffers. In the second pass, lighting computation is done using the G-buffer data. The advantage comes from performing lighting computation only for the visible fragments. Using MSAA with deferred shading increases the memory requirement since now we have store the G-buffer for each subsamples. It demands a lot of memory that many games cannot afford.

3.3.3 Post-Processing Anti-Aliasing

The cost of performing antialiasing in the sample space motivated industry and academia to explore alternate techniques involving post-processing filters. The general idea with post-processing techniques is to render the frames with 1 sample per pixel that will create an image with artifacts and then use a filter to smooth the aliased edges by combining the nearby pixel values. This was first shown through a CPU-based technique known as Morphological Antialising (MLAA) [63]. MLAA opened doors for a new wave of post-processing performance oriented antialiasing techniques. All techniques follows the general idea and differs in the design and usage of reconstruction filters.

Working of MLAA is shown in Figure 3.5. First, pixel discontinuities are found by comparing two adjacent rows and two adjacent columns. A separation line is defined using two farthest vertices. A second step involves identifying secondary separation lines forming a pattern. The predefined patterns can take 3 shapes Z, U and L. Z and U patterns can be

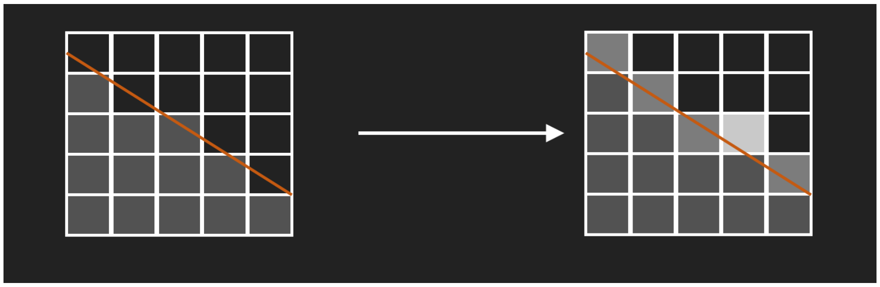

Figure 3.5: Steps involved in MLAA: 1) finding separation lines (top left) 2) computing the area of influence for each lines (bottom) 3) blending colors of opposite pixels (top right)

decomposed into multiple L's making the process easier. The blending weight are calculated as the area of a trapezoid formed from the vertices of the L shape. The colors of the opposite sides of the separation lines are blended using the formula:

$$c_{new} = (1-a)c_{old} + a \times c_{opposite} \tag{3.1}$$

MLAA is simple, fast and flexible but there are several limitations. The edge detection fails on certain edges when the scene is in motion. When an edges rotates, the same patterns are detected triggering blending to happen when it should have been avoided. As the edge covers the sample point of the pixel, it is shaded (no blending) without a smooth transition causing a flickering effect.

In order to improve edge detection and sub-pixel accuracy of MLAA, Jimenez et al. [34] introduced Enhanced Subpixel Morphological Antialiasing (SMAA). SMAA rebuilds MLAA by using a local contrast analysis for better edge detection. This method shows the first attempt to couple MLAA with a temporal reprojection [55] for subpixel features. Once an edge is resolved, the color of that pixel for the previous frame is reprojected to the current frame and blended together to remove flickering. Special care is given to avoid ghosting or disocclusion.

3.3.4 Temporal Antialiasing (TAA)

Temporal Antialiasing (TAA) is the state of the art technique for suppressing spatial and temporal aliasing in modern games. The general idea with TAA is based on the assumption that there is minimal change in terms of geometry, lighting and camera motion from frame to frame. With this assumption we can reuse the sample data from the previous frames in the current frame and get the effect of *supersampling*. Initially we start with one sample per pixel and accumulate samples over time to get subpixel details and produce quality output free of aliasing artifacts. The working of TAA is shown in Fig. 3.6.For every pixel, we first sample at a new location and reprojects the history sample using renderer generated motion vector. If we can reuse the history data we blend them (corresponding to subpixel features) with the new sample to get the final pixel color. Since 60 frames per second is required for games, the players are not likey going to see the artifact from initial frames formed from low samples. TAA runs reasonably fast producing very high quality images compared to MLAA techniques.

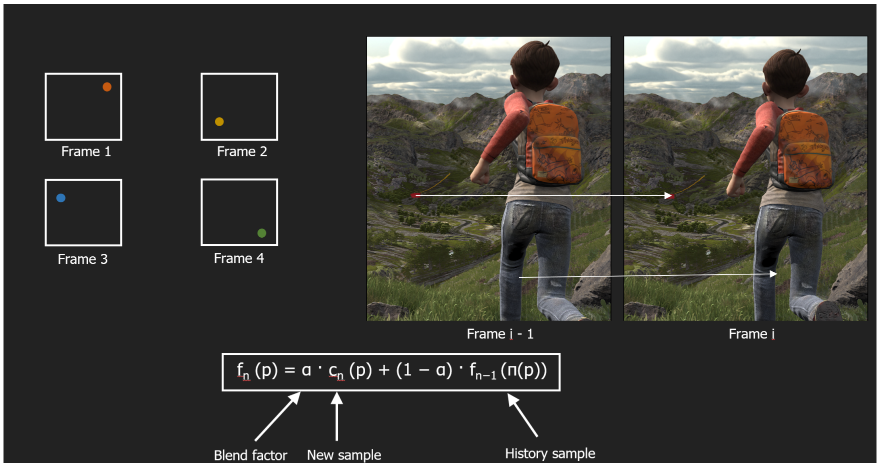

Figure 3.6: Reprojection of pixel center in the current to the previous frame is shown here.

Yang et al. [92] introduced the first reprojection based antialiasing technique. This technique is used for shader aliasing but is scalable for geometry and temporal aliasing. *Halo: Reach* and *Crysis2* were the first commercial games to implement temporal data for antialiasing. SMAA (discussed earlier) combined TAA reprojection with MLAA but as TAA evolved various variants were born that showed improvement over SMAA in terms of quality and performance.

Subpixel features are obtained by sampling at different locations in the pixel. To offset the sample location withing a pixel the camera is jittered over time. The pattern of distribution of the sample is correlated with the convergence of a pixel. When the scene is static, that is when there is no motion, reprojecting from previous frame is trivial but that's not the case for dynamic scenes. To assist in finding the exact location of a pixel in the previous frame, we make use of motion vectors. Motion vectors are 2D textures with the offest information of pixel location. The samples from the previous frame are not stored separately due to the memory constraints. In practice, only one history buffer is stored which contains an exponential average of samples from a previous frames [55, 73].

A naive TAA implementation that doesn't check for validity of the data from history buffer will result in ghosting artifacts. This can occur when there is a change in occlusion, lighting and shading. A validation and rectification steps are required to avoid reprojection errors appearing as artifacts (flickering and over-blurring) in the final image.

3.3.5 Deep Learning based Supersampling

The success of TAA techniques depends on the quality of heuristics and rectification mechanism. Some of the problems in using TAA include over-blurring due to resampling the history buffer, ghosting and temporal artifacts from history rectification. Recent advancements in machine learning in image processing motivated researchers to experiment a learned approach to perform history rectification. This improved the image quality significantly. As of now there is only one ML based supersampling technique from NVIDIA. Their system is a blackbox and the implementation details are unknown but looking at the available data we can speculate that it has elements of TAA coupled with a neural network for learning the rectification step.

3.4 QW-Net

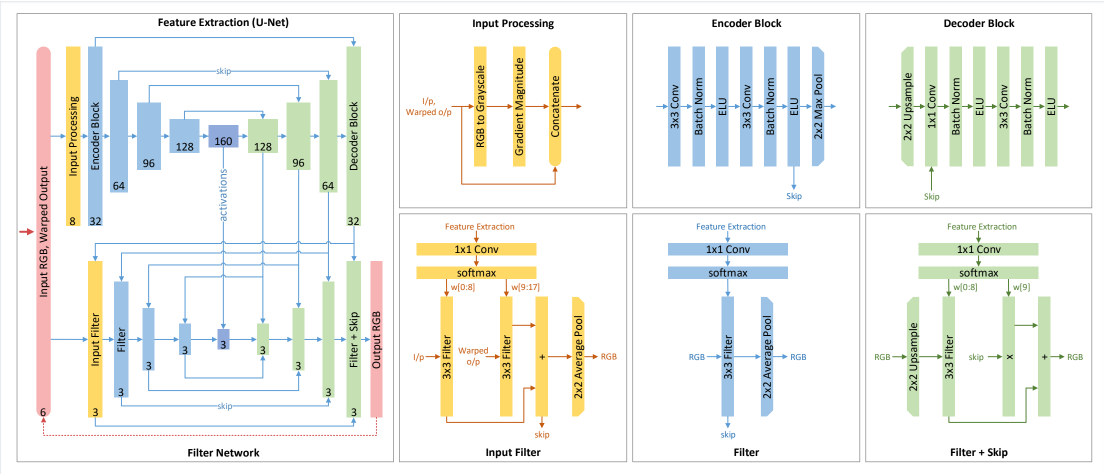

Figure 3.7: The QW-Net architecture including the feature extraction network (top left) and the filtering network (bottom-left). The number of output channels is shown at the bottom of each stage of the network. The details of the encoder and decoder blocks as well as the filter stages are shown on the right.

3.4.1 Network Architecture

Figure 3.7 shows the details of the QW-Net architecture, comprising the feature extraction network and the filtering network. The input to our network is a sequence of images and per-pixel motion vectors generated by the renderer. The network processes images in tone-mapped space, similar to the approach of Bako et al. [7]. However, instead of the log transform, we use the inverse of the perceptual Electro-Optical Transfer Function (EOTF) [77] which is a better match for the human contrast sensitivity model. We map the EOTF to a luminance range of up to a 1000 nits.Besides achieving better results with high-dynamic-range content, tone mapping also reduces the overall perceptual error with 8-bit and 4-bit integer formats.

Similar to TAA, the input images are rendered with a sub-pixel jitter sequence producing a spatial distribution of samples over multiple frames. In order to leverage this distribution and temporally accumulate samples, we apply the frame-recurrent approach of Sajjadi et al. [72], where the previously reconstructed frame is warped and concatenated with the input frame, forming the current input to the network.

3.4.1.1 Feature Extraction

The feature extraction network is based on the U-Net architecture which includes a series of encoder blocks that downsample the image followed by decoder blocks that reverse this process. The first stages in the network convert the input images $\boldsymbol{I_a}$ and $\boldsymbol{I_w}$ to grayscale and compute their gradient magnitudes. The two gradient magnitude images are concatenated with $\boldsymbol{I_a}$ and $\boldsymbol{I_w}$ forming the input for the first convolution layer. The gradients highlight aliased regions in the image which aids training [7].

Each encoder block has two convolution layers with a 3×3 spatial footprint, each followed by batch normalization [31] and Exponential Linear Unit (ELU) activation [17].The last stage in the encoder block is downsampling with 2×2 max pooling. We increase the number of channels (tensor depth) by 32 at each successive block starting with 32 in the first encoder block and reaching 160 at the bottleneck. Encoder blocks have skip connections to the corresponding decoder blocks relaying high-frequency details to the decoder. The bottleneck is similar to an encoder block but excludes max pooling and skip connections.

The first stage in the decoder block is a 2×2 nearest-neighbor upsampling operation. The upsampled activations are concatenated with the skip connection and projected to the same size as the encoder output using a 1×1 convolution layer [79]. The decoder block includes a single 3×3 convolution layer resulting in three such layers at each scale, excluding the bottleneck, which has two convolution layers.

3.4.1.2 Filter Network

The filter network has a similar topology to the feature extraction network with a series of downsampling filters followed by upsampling filters with skip connections between them. The pair of downsampling and upsampling filters at each scale are coupled to the output of the corresponding decoder block in the feature extraction network.

Each filter uses the activations from the decoder block to predict a 3×3 kernel that is applied to the input image. Similar to Bako et al. [7] we use a 1×1 convolution layer with softmax activation to predict the kernel resulting in normalized weights. The input filter predicts 18 normalized filter weights corresponding to two 3×3 filters with 9 weights each. These filters

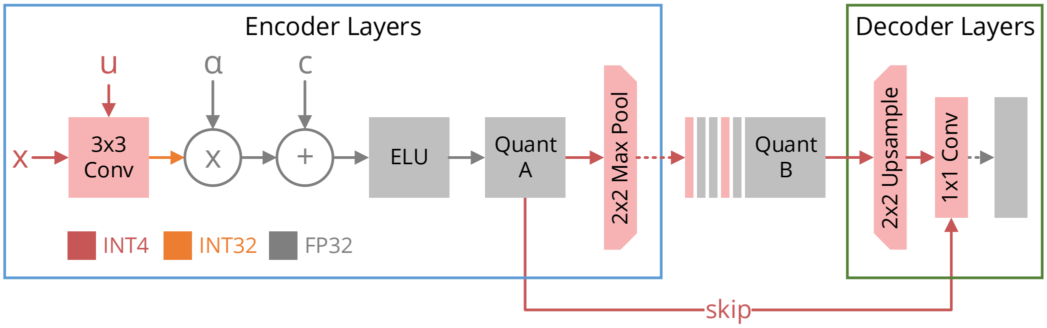

Figure 3.8: Convolution layers with 4-bit activations and weights. When the outputs of quantizers are concatenated together (e.g. quantizers A and B) we ensure that they use the same quantization range.

are applied to $\boldsymbol{I_a}$ and $\boldsymbol{I_w}$ respectively and the results are summed to produce a single image. The subsequent downsampling filters apply a 3×3 kernel to a single image. The last stage in each downsampling filter is a 2×2 average pooling operation. The bottleneck filter excludes this pooling operation.

The first stage in each upsampling filter is bilinear upsampling, following which the image is filtered and combined with the skip connection. The upsampling filters use 10 filter weights, 9 weights for the 3×3 filter kernel and one for scaling the skip connection. We use average pooling and bilinear upsampling in the filtering network as it results in better image quality. On the other hand we use max pooling and nearest neighbor upsampling in the feature extraction network as they are computationally cheaper and do not significantly impact feature extraction.

3.4.2 Quantization

We quantize all layers of the feature extraction network to use 4-bit weights and activations, except the first convolution layer, which uses 8-bit weights and 4-bit activations. As previously observed with other networks [97, 94, 16], having the input layer at a higher precision leads to a significant improvement in the loss. We use per-channel symmetric quantization for the weights and affine per-layer quantization for the activations [41]. This approach achieves good results with 4-bit quantization with a relatively small overhead.When the outputs of two layers are concatenated together, we use the same quantization range for both the activations as shown in Figure 3.8, ensuring a uniform quantization range for the input to the next convolution layer.

The mapping of quantized integer weights and activations u, v to their real values w, x is given by

$$w = s_w u, \tag{3.2}$$

$$x = s_x(v - z), \tag{3.3}$$

where s_w is a per-channel scale (step size) for the quantized weights, s_x is a per-layer scale for the quantized activations and z is the zero point. We can then represent the convolution for a single channel as:

$$y = \sum w_i x_i + b_i,$$

where b_i is the bias. Substituting with Equations 3.2 and 3.3 we get:

$$\begin{aligned} y &= s_w s_x \left(\sum u_i v_i - \sum u_i z\right) + b_i, \\ &= \alpha \sum u_i v_i + c_i, \end{aligned}$$

where $\alpha = s_w s_x$ and $c_i = b_i - \alpha \sum u_i z$, represents a single-precision floating point bias that can be precomputed.

Figure 3.8 shows the convolution and activation layers for a single channel along with their numeric formats. The convolutions constitute the majority of the computations but can be mapped to the tensor cores on Turing that can efficiently evaluate 4-bit multiplications and accumulate the result in a 32-bit integer. An additional floating-point MAC scales the convolution output by α and introduces the bias c, following which an ELU activation is computed with single-precision floating point. A final MAC operation then maps the floating-point activation back to a 4-bit integer.

We initially train our network with full precision and then quantize the weights and activations by fine tuning the network with simulated quantization [32]. Training with simulated quantization achieves significant improvements over post-training quantization, especially with 4-bit precision [41]. Simulated quantization applies a quantization followed by a de-quantization in the forward pass, introducing quantization errors while maintaining and updating the weights as un-quantized variables.

The activations are quantized using a function $\mathrm{Q}(x,l,u)$ that maps an activation x in the range $[l,u]$ to a quantized integer x_q in the range $[M_l, M_u]$. For a b-bit integer, $M_l = -2^{b-1}$, $M_u = 2^{b-1} - 1$ and $M = M_l + M_u$ is the number of quantization levels. Following

Jacob et al. [32], we adjust the quantization range such that a zero value gets quantized without error, preserving zero-padded data. The adjusted range $[l_a, u_a]$ is derived as follows:

$$l_a = s\left\lfloor \frac{l}{s} \right\rceil, \tag{3.4}$$

$$u_a = l_a + sM, \tag{3.5}$$

where $s = \frac{u-l}{M}$ is the quantization step size and a common term for both s_w and s_x. The quantization function Q is then given by

$$x_q = \begin{cases} \left\lfloor \frac{x}{s} \right\rceil + z & \text{if } l_a \leq x \leq u_a \\ M_l & \text{if } x < l_a \\ M_u & \text{if } x > u_a, \end{cases} \tag{3.6}$$

where $z = M_l - \frac{l_a}{s}$ is the zero point introduced earlier in Equation 3.3.

The simulated quantization function $\mathrm{Q}'(x,l,u)$ can be derived from Equations 3.3 and 3.6:

$$\mathrm{Q}'(x,l,u) = s\left(\mathrm{Q}(x,l,u) - z\right) \tag{3.7}$$

While Jacob et al. [32] use the moving averages of the minimum and maximum activations in a batch to derive the quantization thresholds l and u, we adopt a more recent approach [33, 23] where the quantization thresholds are derived from trained variables.

For the filtering network, We quantize the weights of this layer to 8-bit using trained quantization but we do not quantize the activations (filter kernel) as all filtering operations are computed with single-precision floating-point.

Although the focus of this project is high-quality reconstruction with quantization, we provide some data to support the feasibility of high performance 4-bit inference. Table 3.1 lists the number of MAC operations per pixel for the QW-Net network and the corresponding numeric formats.

We also list the additional overhead for quantization which is less than 1% of the total computations. This includes one MAC per channel for scaling and biasing the convolution output and another MAC for quantizing the activations. The overhead of kernel prediction and filtering is close to 2%, while 4-bit convolutions account for 95% of the computations. Table 3.1 also lists the number of operations for direct prediction with a comparable U-Net network.

To further study the feasibility of a 4-bit network, we implemented a convolution layer in CUDA with 32 input and output channels. This represents the second and last layer in the feature extraction network which are the most expensive. We also benchmarked a similar convolution layer in NVIDIA TensorRT which is a general-purpose inference runtime. Both these implementations were evaluated on a TITAN RTX GPU and utilized tensor cores. Table 3.2 shows the execution time with different numeric formats at a 1080p resolution. We see a 2× performance gain going from 8-bit integers to 4-bit integers using our custom kernel, which is in-line with the expected performance on a Turing GPU. A similar trend is observed with the 8-bit and half precision convolutions in TensorRT. Unfortunately, TensorRT does not support 4-bit convolutions.

3.4.3 Results

3.4.3.1 Dataset

We train and evaluate our network (QW-Net) and a comparable direct prediction network (U-Net) using images rendered with Unreal Engine 4 (UE4) [25]. The direct prediction U-Net is the same as the feature-extraction network but with an additional 1×1 convolution layer at the output to directly produce the reconstructed color.

We generated our training and test datasets using a modified version of UE4. Obtaining large-scale datasets that are representative of modern game workloads is a challenging task because most game engines cannot produce reference images with the sampling density that we require. Therefore, we render an image sequence with one sample per pixel and repeat this multiple times with different sub-pixel viewport offsets. The frames from each render are then weighted by a 2D Blackman-Harris window and accumulated to produce a supersampled reference sequence. This accumulation is performed in a perceptually tone-mapped space [77] in order to suppress “fireflies” caused by high-energy outliers in the reference image. The accumulated result is inverse tone mapped to bring it back into linear space. This is analogous to what most TAA implementations do when accumulating samples.

We prepared our datasets from four cinematic scenes publicly available for UE4. ZENGARDEN has large, smooth surfaces with well defined edges in an outdoor setting and a slow panning camera, making it the least challenging for reconstruction. INFILTRATOR features dark indoor scenes with several light sources, and highly specular materials, as well as an outdoor cityscape. KITE is a brightly-lit landscape sequence with a large amount of alpha-

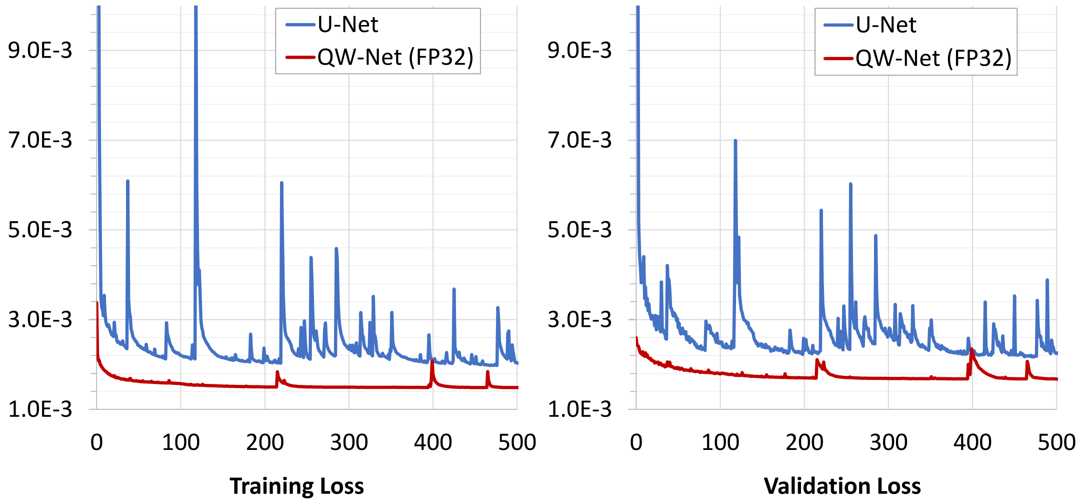

Figure 3.9: Loss versus epochs during full-precision training.

tested foliage. Finally, SHOWDOWN is another city scene with several reflective materials, which was not included in the training data. Both the training and test datasets consist of several image sequences, each having 120 continuous frames. The training set has 6 sequences from INFILTRATOR, 4 from KITE and 2 from ZENGARDEN. The test set contains 2 sequences of each of these three scenes, and an additional sequence from the SHOWDOWN demo which was excluded from training. We also set aside a sequence from the KITE demo for validation.

3.4.3.2 Network Analysis

Figure 3.9 shows the loss profile for QW-Net and U-Net with full-precision training. This aligns with earlier findings [7, 82] showing faster convergence with kernel prediction compared to direct prediction.

We derived the quantized weights by first training our network for 500 epochs and

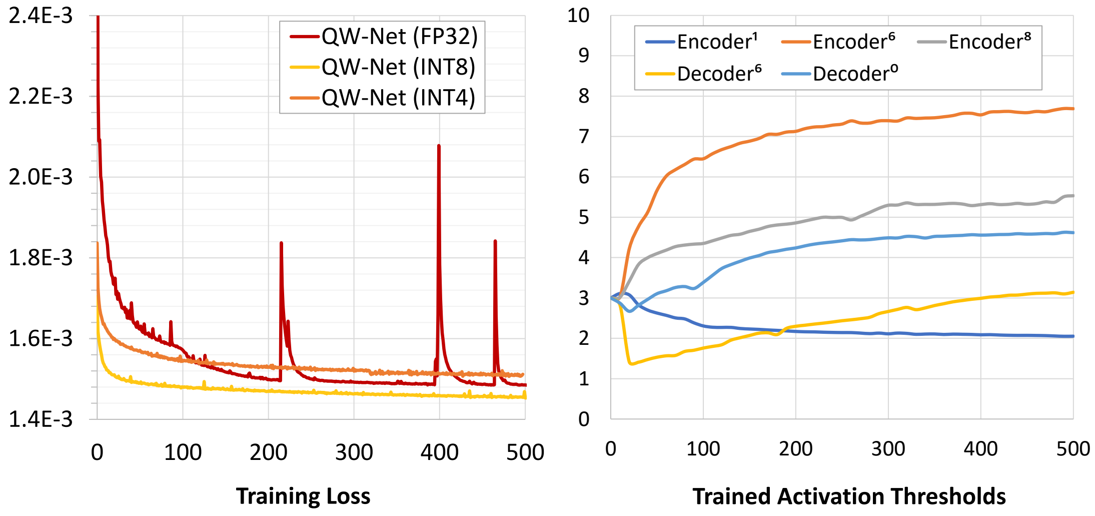

Figure 3.10: On the left, we show that training loss for both 4-bit and 8-bit quantization converges quickly to a value that is close to the full precision loss. On the right, we show how the upper quantization thresholds change during reduced precision re-training. The encoder and decoder layers are numbered from the input to the output. Layers closer to the inputs and outputs (Encoder 1, Decoder 6) have a lower threshold compared to inner layers (Encoder 6, Decoder 0).

then re-training the network with simulated quantization. Figure 3.10 shows that re-training converges quickly to a loss that is close to the full precision loss. With 8-bit quantization the loss converges to a value that is slightly below the full precision loss while with 4-bit quantization it remains slightly above. This small difference in loss values has little impact in terms of image quality as we show in the next section. Figure 3.10 also shows the quantization threshold for the activations from a few selected layers in the network. The layers closest to the input and the output seem to adapt to a lower threshold compared to the inner layers.

To verify that our network is not over-parameterized, we evaluated a network called QW-Net-R, where we set the tensor depth of the first stage to 24 (instead of 32) and increased it by 24 at each following stage. As a result, the computational complexity of QW-Net-R

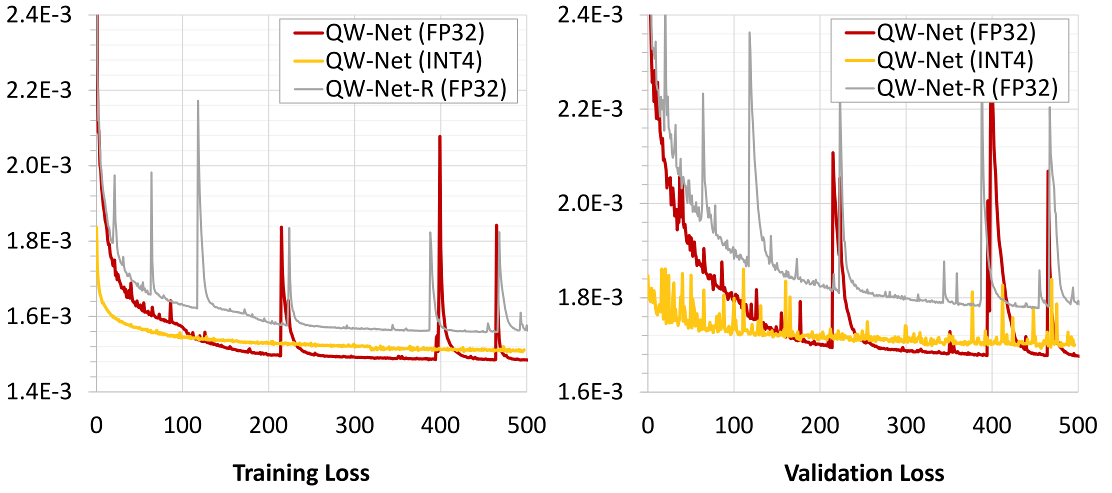

Figure 3.11: QW-Net-R trained with single-precision float versus QW-Net re-trained with 4-bit quantization.

is approximately half of QW-Net. Figure 3.11 shows that QW-Net quantized to 4-bits still converges to a significantly lower loss than QW-Net-R without any quantization.

We also studied the impact of reduced precision at the input layer of the feature extraction network, by quantizing this layer to 4 bits instead of 8 bits, while the remaining layers were quantized to 4 bits as usual. This resulted in a significant increase in loss as shown in in Figure 3.12, highlighting the importance of higher input precision.

3.4.3.3 Image Quality

We include a qualitative and quantitative comparison of the reconstructed images using the default TAA implementation in UE4, direct prediction (U-Net) and our network (QW-Net). Unless otherwise noted, all QW-Net images were produced with 4-bit quantization. QW-Net was trained for 500 epochs at full precision followed by trained quantization for 1000 epochs while U-Net was trained for 1200 epochs at full precision. For each network we use the

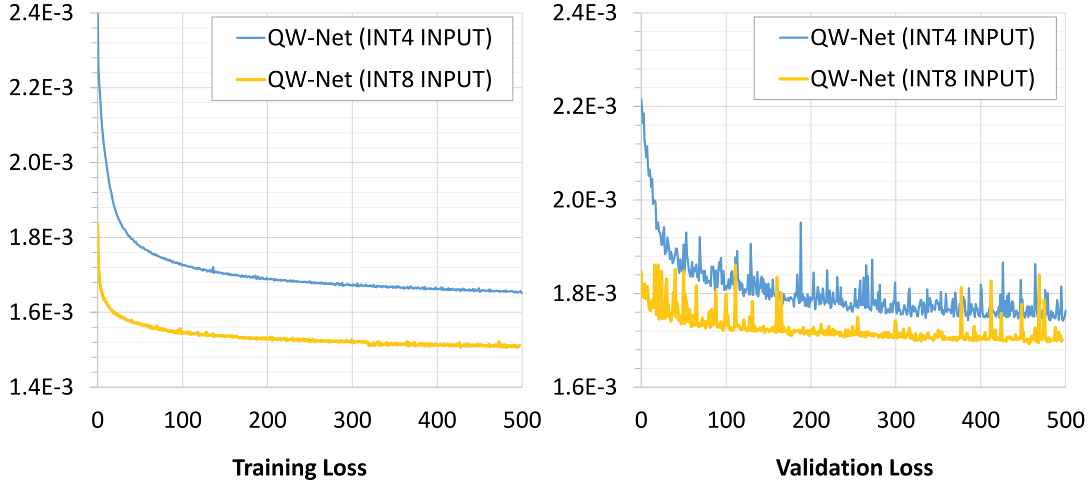

Figure 3.12: QW-Net re-trained with 4-bit and 8-bit quantization at the input of the feature extraction network.

weights from the epoch with the minimum training loss. We use highest-quality settings when rendering images with TAA.

For the quantitative analysis we use two widely-used metrics: the Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity (SSIM) using a reference image rendered with 256 samples per pixel. Unfortunately, neither of these metrics sufficiently penalizes local phenomena such as jaggies or ghosting. However, we still find them to be useful as they clearly show an order of quality across different techniques. The problem of quantitatively analyzing aliasing artifacts in synthetic images was studied by [59], where they proposed a neural network to generate a metric for aliasing.

Figures 3.1 and 3.13 show a few representative frames from our dataset. With the exception of Figure 3.1, all results shown in this paper are from the test dataset and were not used during training. In Figure 3.1, TAA blurs the text on the sign significantly and loses details on the railings as a result of color clamping. In Figure 3.13, blurring is present in almost all

TAA examples, with ZENGARDEN also displaying severe ghosting artifacts.

The differences between QW-Net and U-Net on the other hand are more subtle. However, looking at the quality metrics, we can see that QW-Net consistently outperforms U-Net, and these differences can also be perceived with some visual scrutiny. While U-Net can remove aliasing in many cases, it fails on the railings in Figure 3.7. The results of QW-Net are also noticeably sharper, especially in the KITE scene. The bottom row of the INFILTRATOR scene in Figure 3.13 also shows a color shift around the collar. QW-Net on the other hand cannot produce a color shift, since it predicts a single set of kernels that are applied to all color channels. The top row of the SHOWDOWN scene shows a region where some ghosting is observed with all reconstruction techniques but to a lesser degree with QW-Net.

While these figures show selected frames, we observe superior quality metrics for all sequences with QW-Net. Figure 3.14 plots the per-frame metrics for an example sequence from KITE. It shows a notable drop in quality around frame 100, which is the result of a large disoccluded region where temporal supersampling has to discard pixel history. Even in this scenario, QW-Net stays above U-Net in quality and both networks recover quickly after a few frames. Figure 3.15 shows a region in the frame that becomes blurry at the disocclusion event and regains sharpness in a few frames.

Inspired by the evaluation of video super-resolution methods [72, 12], we also show *temporal profiles* in Figure 3.16. These images highlight temporal discontinuities by tracing a column of pixels over a sequence of frames. While the temporal profiles with QW-Net appear smooth and relatively close to the reference, U-Net shows visible aliasing on the railing in INFILTRATOR and temporal noise in the KITE scene.

3.5 Conclusion

Quantization can be a key to overcome the computational barriers of deep neural networks for their wider application in real-time rendering. It has particularly good potential on modern GPU architectures that accelerate reduced-precision tensor operations. In this project, we demonstrated a network where most of the convolutions can be mapped to 4-bit operations without a significant loss in dynamic range or quality. While preserving image quality with 4-bit quantization has been the main focus of our research, an optimized implementation for real-time inference remains future work.

Table 3.1: Multiply-Accumulate (MAC) operations per pixel.

Network	**INT4**	**INT8**	**FP32**	**% Total**
U-Net				
Input Conv			2304	3.13
Encoder			37344	50.73
Decoder			33472	45.47
Pool / Upsample			404	0.55
Output Layer			96	0.13
Total			70912	
QW-Net				
Input Conv		2304		3.07
Encoder	37344			49.80
Decoder	33472			44.63
Quantization			387	0.52
Kernel Prediction		1358		1.81
Pool / Upsample			28	0.04
Filter			99	0.13
Total	70816	3662	514	

Table 3.2: Execution time for a single convolution operation.

Precision Level	**TensorRT(ms)**	**Custom(ms)**
FP16	1.03	
INT8	0.61	0.29
INT4		0.14

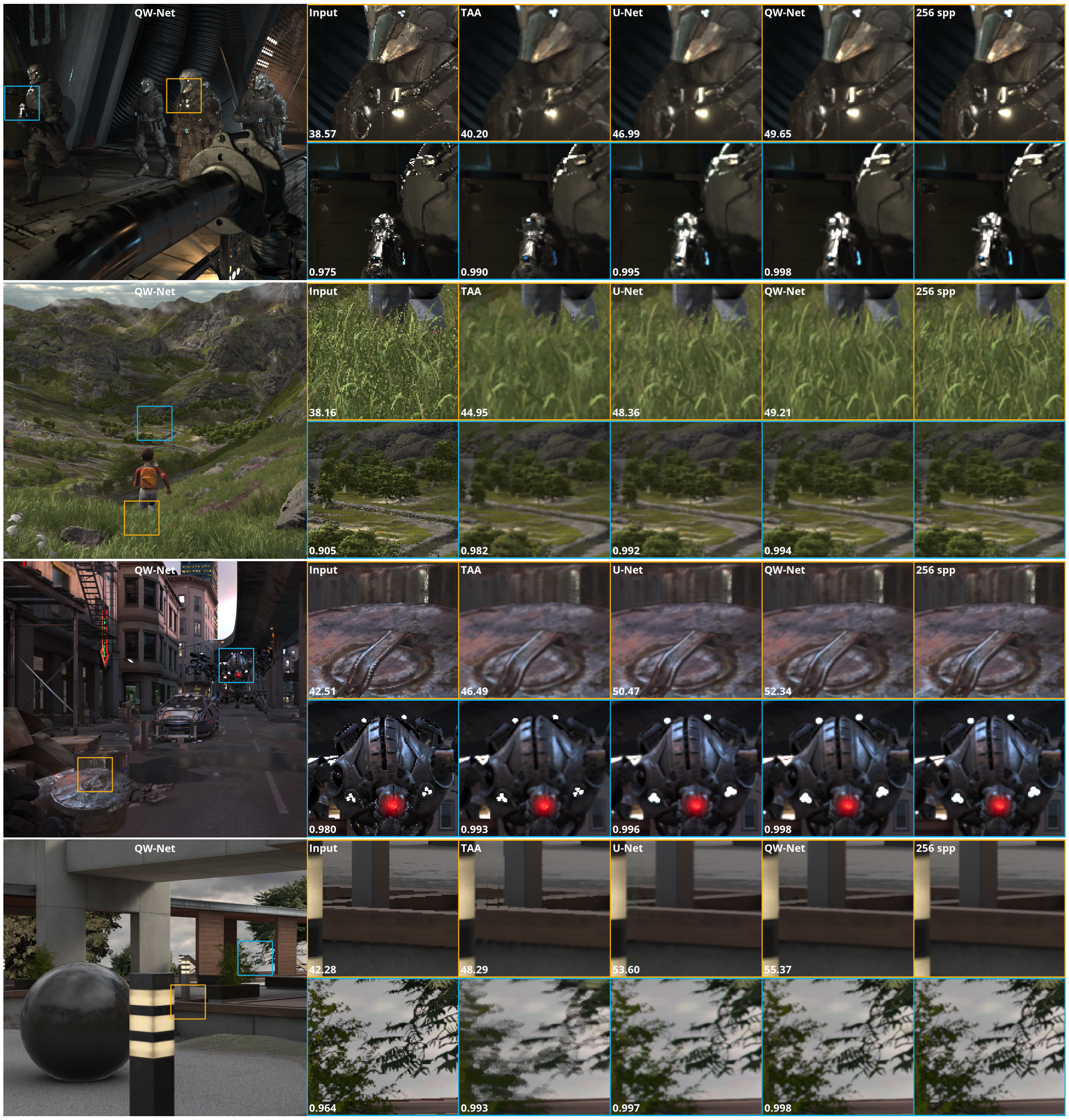

Figure 3.13: Representative frames from our test datasets. We compare the visual quality of TAA, U-Net, and QW-Net (INT4), compared to the 256× supersampled reference. For each frame we magnify two areas of interests for easier comparison. We also show the PSNR metrics in the upper comparison rows and the SSIM values in the lower rows. TAA produces significantly lower metrics for all test frames, losing sharpness, and sometimes producing noticeable ghosting (e.g. ZENGARDEN, top row). U-Net closely approximates the reference, but is also consistently below QW-Net in quality metrics, and sometimes produces color shifts (e.g. INFILTRATOR, bottom row).

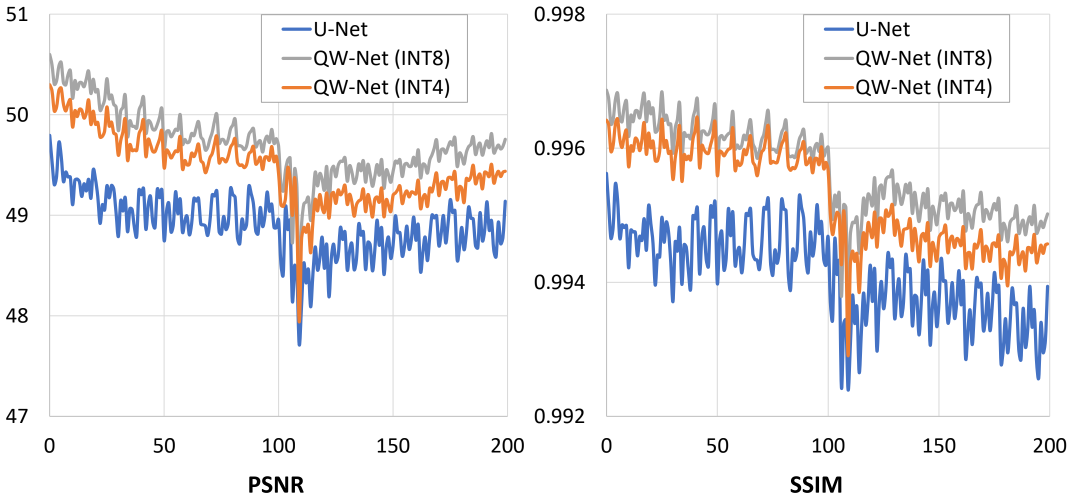

Figure 3.14: Per-frame quality metrics on a continuous frame sequence from the KITE dataset. QW-Net consistently outperforms U-Net, with a slight degradation in the case of INT4.

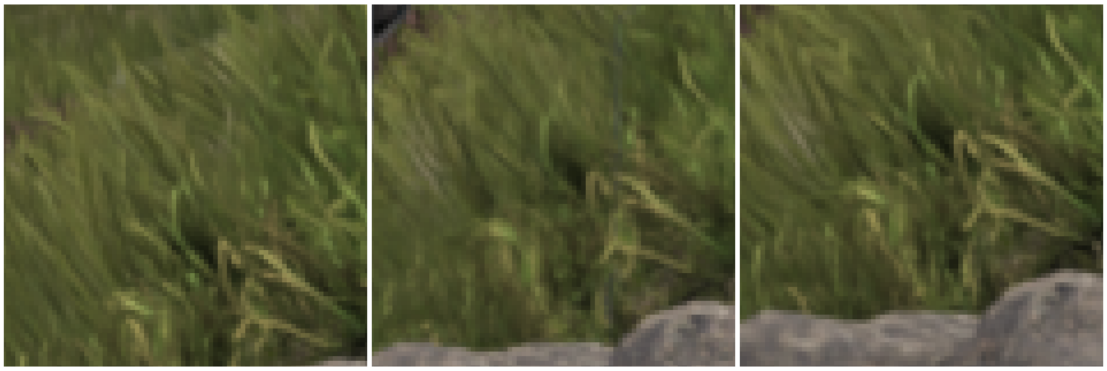

Figure 3.15: The loss of history samples due to disoccluded regions results in a short-term reduction in image quality. Before occlusion (left), at disocclusion (middle), 10th frame after disocclusion.

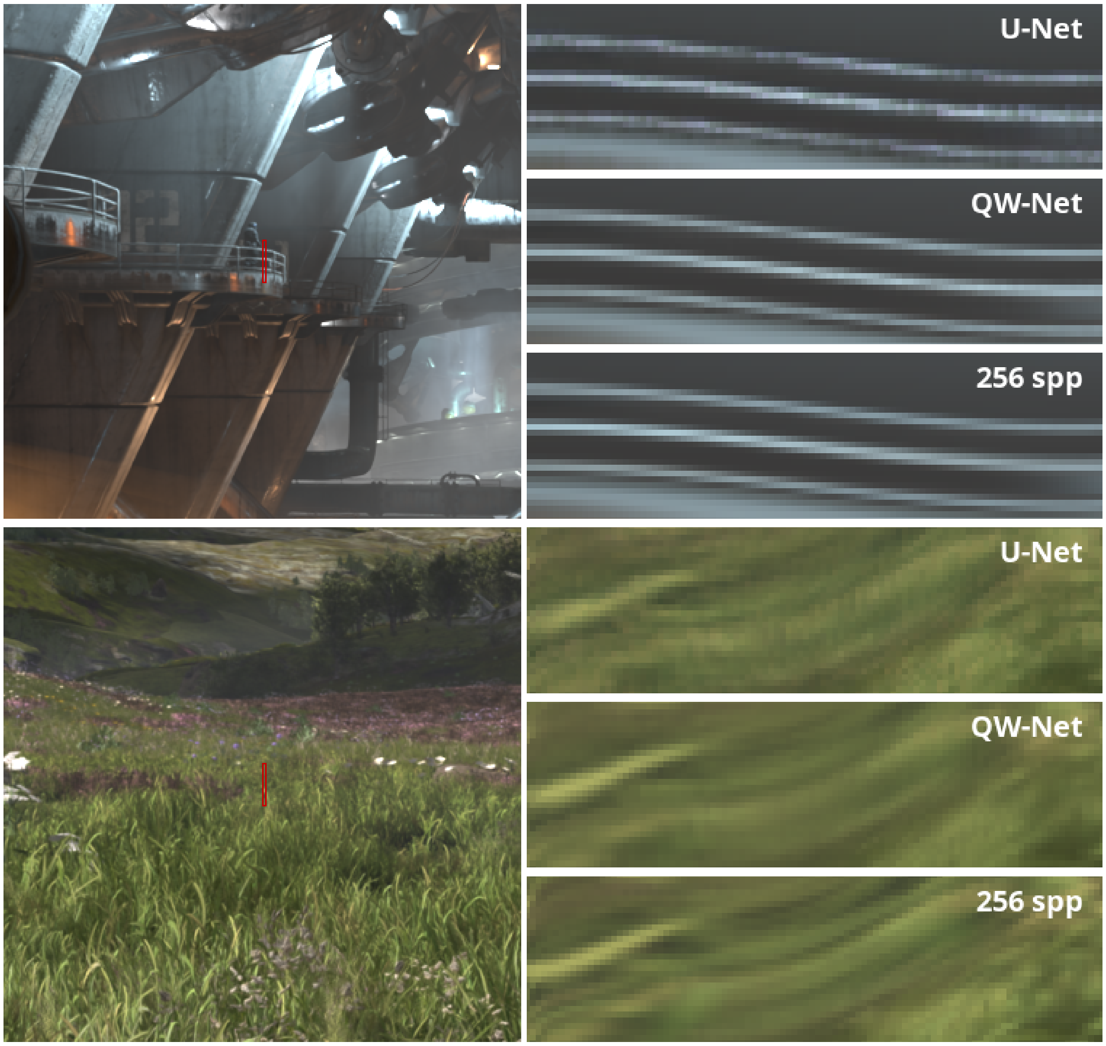

Figure 3.16: Temporal Profiles from the INFILTRATOR (top) and KITE (bottom). QW-Net has a significantly smoother profile than U-Net, indicating better temporal coherence.

Chapter 4

Neural Denoiser for Real-time Ray tracing

4.1 Introduction

Physically-based light transport is the key to achieve photorealistic effects in rendering. The general idea is to simulate real-world light transport as close as possible avoiding approximation tricks and tweaks to model realistic lights effects such as soft shadows, reflections, and so on without any extra effort [61]. In 1986 Kajiya [35] formulated a recursive integral known as the *rendering equation* to describe the light transport.

$$L_o(x, w_o) = L_e(x, w_o) + \int_{\Omega} f(x, w_i, w_o) L_i(x, w_i)(w_i.n) dw_i \tag{4.1}$$

$L_o(x, w_o)$ is the radiance leaving the from a position x in the direction w_o. The outgoing radiance $L_o(x, w_o)$ is made up of radiance that the surface at that point emits($L_e(x, w_o)$) towards w_o and the integral of reflected radiance from all possible direction over the hemisphere. $f(x, w_i, w_o)$ is the BRDF (bidirectional reflectance distribution function) that describes

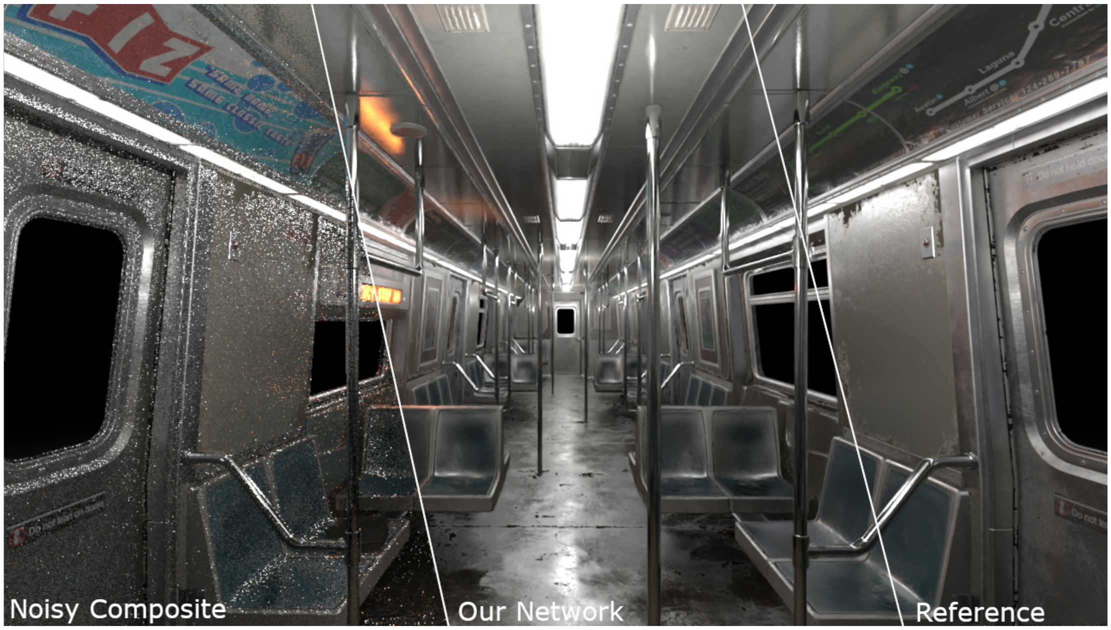

Figure 4.1: Our network(middle) is able to denoise a 1 ray-per-pixel ray traced frame (left) producing results comparable to the reference (right)

the material of the reflected surface.

With rasterization and traditional ray tracing, we can simplify the rendering equation by dropping the integral, and only light from a surface to the source is modelled without any inter-surface reflections. Path tracing is a technique that utilizes Monte Carlo Integration to approximate the rendering equation. The estimate of the integral is determined by combining the stochastic samples of the function. The expected value is the actual value of the integral. When the number of samples approaches infinity, the variance decreases, and we get the true value of the integral.

In simpler terms, we shoot lots of rays through the pixel to the 3D scene and follow its path until the ray hits a light source or termination criteria that we define to get the true color of that pixel. Here the variance caused by undersampling of the integral shows up in the final image as noise. Unlike the noise we see in a photograph, which is addictive in nature, here, the noise is part of the solution. To minimize the variance, we need a large number of samples per pixel. This is computationally intensive and slow for real-time use cases.

For video games, we trace 0.5 or 1 sample per pixel(spp) to achieve realistic frame-rates. With the current hardware capabilities, we are only limited to a certain effect like soft shadows, reflections or ambient occlusion. The trend is towards fully path-traced video games, but such games will require special techniques to deal with the noise caused by low sample counts. In the offline rendering space, various variance reduction techniques exist, including importance sampling [80] and next event estimation [24]. For real-time rendering, these techniques alone cannot generate noise-free images.

Filtering is a popular method to remove noise in rendering. The idea is to combine similar estimates to reduce variance. In image space, this means blending nearby pixels with similar spatial features to get the final pixel value. Sample space or path space filtering is also possible, but image space filtering is preferable due to its low computational and memory expense. The high-frequency details in the scene need to be taken into account when performing filtering.

Recently, machine learning based filtering have become the state of the art technique in denoising for offline and interactive path tracing applications. These techniques require either a deep network to cover a large footprint or 4-8 samples per pixel inputs. Such requirements are not practical in real-time applications due to performance reasons. Furthermore, these networks are not designed for low sample inputs; therefore, they fail to produce high-quality results.

We present an extension to the QW-Net that is designed specifically for denoising real-time ray traced effects such as soft shadows, reflections and global illumination. Typically, these effects are denoised separately using domain-specific, analytically designed filters, with some of the filters needing to be generated per light source, increasing the overall cost of denoising. Our goal is to replace separate denoisers with a single network but filtering each illumination component separately. With our network we combine the lighting components into a low-frequency diffuse signal, and a high-frequency specular signal. Our results are significantly better in terms of image quality than the denoising techniques used in modern games.

To further enhance the quality of the output, we train a new network based perceptual loss function that is specially designed for extracting features from a rendered image. This loss function reduces the structural artifacts (neural hallucinations) typically found in VGG-based loss functions [85] that are trained using natural images. Using a photorealistic synthetic dataset rendered for understanding indoor scenes, we train our custom loss network to solve segmentation tasks.

4.2 Related Work

Monte Carlo denoising is a well-explored topic with respect to offline rendering and now re-exploring for real-time use cases. A survey by Zwicker [98] on adaptive sampling and reconstruction of Monte Carlo rendering divides the techniques into *a priori* and *a posteriori* methods. A priori method (analytical denoisers) derives filters based on the analysis of light transport, whereas a posteriori method treats the renderer as a black box and works only with the outgoing radiance.

4.2.1 Classical Filter Methods

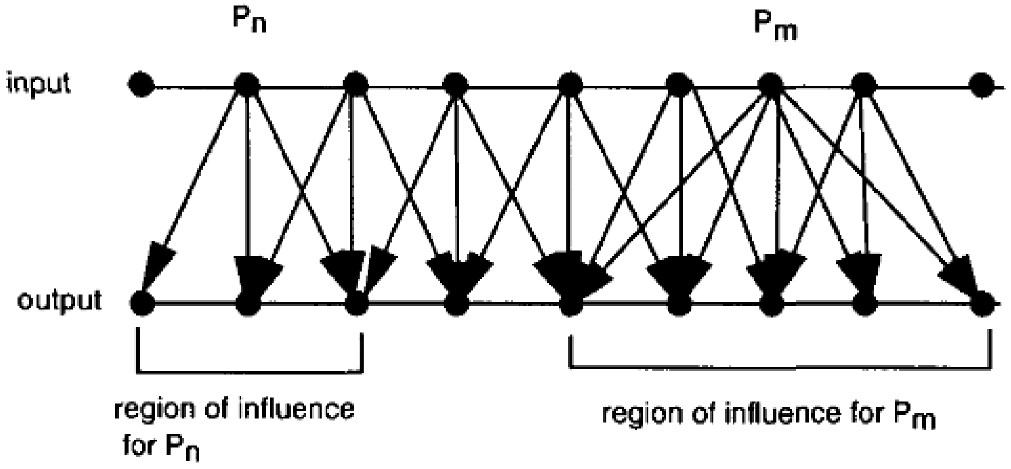

Figure 4.2: Energy of pixel P_n is spread to the neighboring pixels

Lee and Redner [42] made the observation that linear filters can cause problems when filtering Monte Carlo renderings. If the filter is small, there will be residual noise and if the filter is large, high frequency details such as edges are blurred. They proposed a median and an alpha-trimmed mean filter where the outlier pixel values are thrown out. In path tracing, as we discussed earlier there is no outlier samples and every sample counts towards the final result. Removing samples will cause energy drop and bias the resulting image.

Rushmeier and Ward [70] proposed a nonlinear filter that preserves energy shown in Figure 4.2 Instead of removing outliers, they spread the energy to the neighboring pixels. This avoids energy loss but energy can still spread over edges. McCool et at[45] used an anistropic diffusion to smooth the noisy images. To avoid blurring the edge details they used noise-free auxiliary buffers such as normals and depth map to steer the filter. The technique is an iterative filtering process and the image gets smoother for each iteration. Here the difficulty is to find the optimal number of iteration before introducing blur in the image.

Bilateral filter for rendering was introduced by Xu and Pattanailk [89]. Bilateral filter was extended to cross-bilateral filter where the filter weights not only depends on the pixel values but also looks at the noise-free auxiliary buffers. Bilateral filters cannot completely eliminate all the fireflies in the render and sometimes used after applying a Gaussian filter to smooth the image first. Unfortunately with Gaussian filters sharp features are hard to retain.

4.2.2 Filter Selection using Error Estimation

To improve the quality of Gaussian or Bilateral filters, a filter bank consisting of multiple filters with different parameters are stored. We could then select a filter per pixel that minimizes an error estimate as shown in Figure 4.3. Rousselle et al. [67] make use of a Greedy Error Minimization (GEM) approach where from a bank of Gaussian filters with different kernel size, they greedily choose larger kernels until the error approximate increases. They approximate pixel Mean Square Error (MSE) using the bias term and variance of coarse and fine filters. The region with edge details will use finer filter kernels while smooth gradient region uses large filters kernels. This method was improved by Li et al. [43] by replacing the error estimate with Stein's Unbiased Risk Estimator (SURE) [78] that can estimate MSE from a variety of unbiased estimators. This allowed them to use cross-bilateral filters in the filter bank which improved the denoising quality. Later Rousselle et al. [68, 69] combined the previous

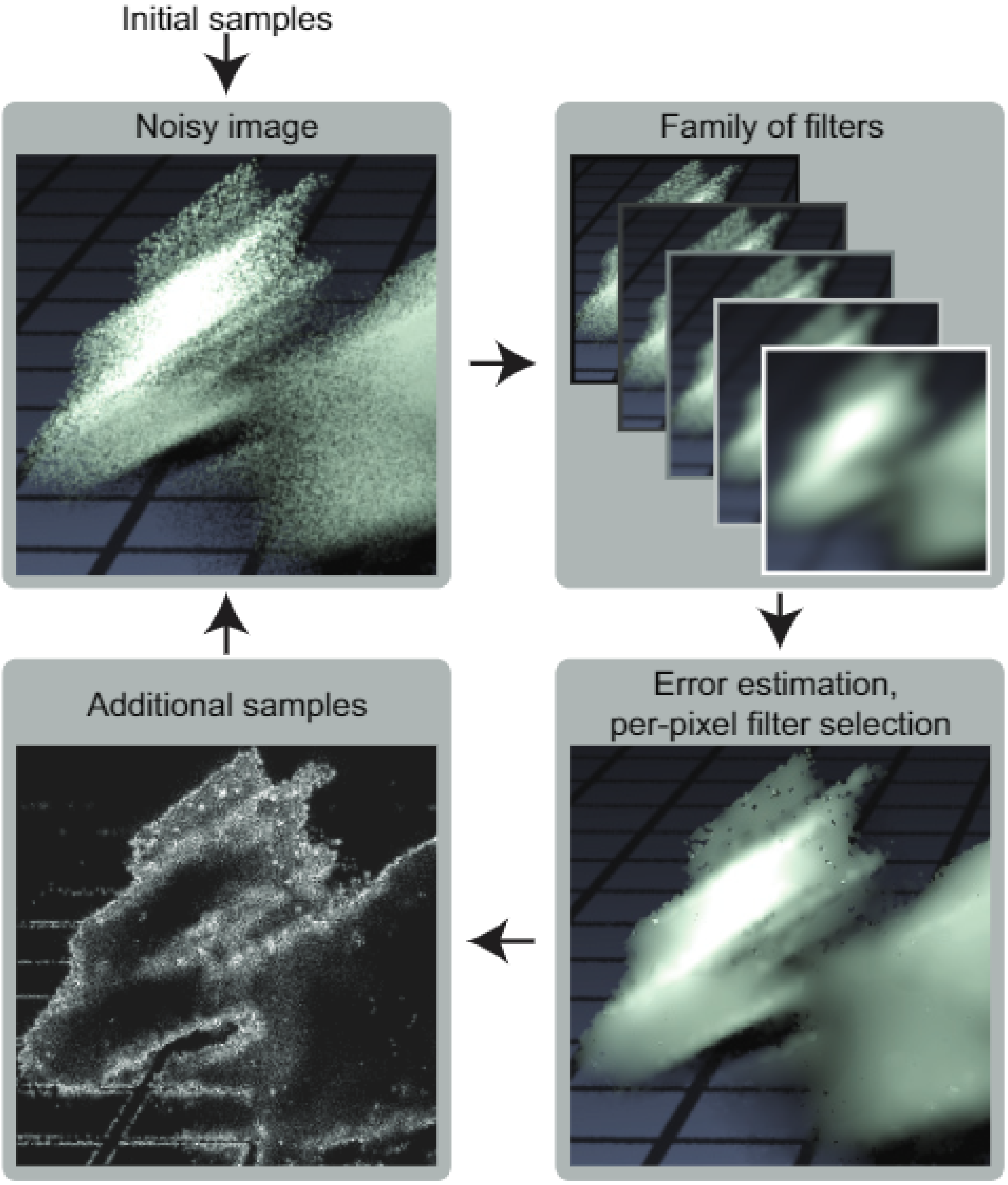

Figure 4.3: A per pixel filter is selected based on an error estimation

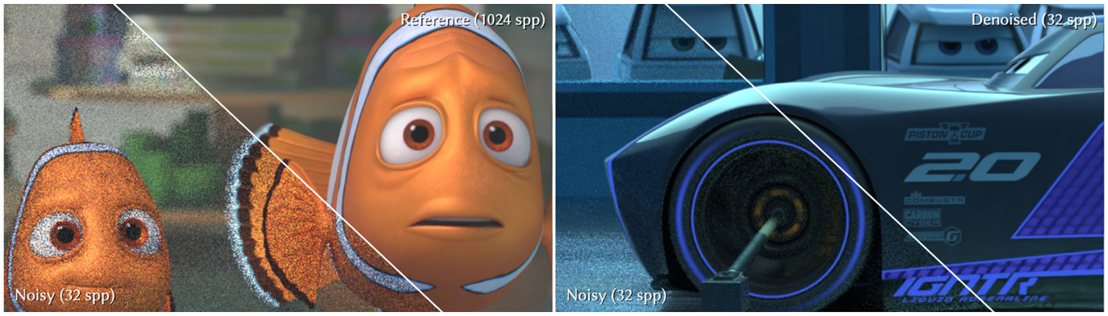

Figure 4.4: Input and output from the kernel predicting denoiser network from Disney's Hyperion renderer

techniques and improved it by introducing non-local means (NLM) filtering where filter kernels are determined based on the patches centered on the two pixels. NLM's are more expensive to compute and are limited to offline rendering applications.

4.2.3 Machine Learning based Filtering

Kalantari et al. [36] used a fully connected network to learn the parameters of feature-based denoising filters for Monte-Carlo renderings. Previous filtering method required hand tuning for the filter parameters which can be challenging depending on the content. In this work, they propose a neural network based regression model to find the ideal filter parameters. The primary features such as color, position, normals are used to extract secondary features including mean and variance. The secondary feature is then given as the input to the fully connected network and the output is the filter parameters. The network is trained using a loss function defined between predicted output pixel from the network and the ground-truth pixel from a path tracer.

Bako et al. [7] proposed a network that predict the filter itself resulting in a significantly better image quality. Letting the neural network predict the kernel for each local region of the image is powerful because now the model has the freedom to design complex filters per small patch. In this work, they designed two neural network based models, one for denoising diffuse illumination and specular illumination. First the input data goes through a preprocessing step in which the albedo (texture without light contribution) is decoupled from the diffused component. This step helps the network to filter aggressively without blurring the texture. Both diffuse and specular data pipeline goes through a normalization step which is required since the HDR image data can have infinitely high or low light contribution. Once the kernel is predicted filtering is done on both network paths and the result is combined to get the denoised output. A fully convolutional network is used with no fully connected layers. A deep network will have large receptive field covering more context and have long-range dependencies. The side by side comparison of with the noisy input it shown in Figure 4.4.

The most successful network for image processing application is the U-Net [66]. The U-Net consist of a series of encoders that extracts the significant features from the input. At each encoder stage the input is downsampled to increase the receptive field so that the filters in the next stage covers a large area. The encoders are followed by series of decoders that performs feature decompression and upsampling. Skip-connections exist between the corresponding encoder and decoder state. These are useful in transferring the high-frequency information that are lost during the downsampling. Chaitanya et al. [14] extended this architecture for denoising Monte-Carlo renderers. They made the U-net temporally coherent by adding recurrent connection within each layer. Recurrent layers are only used in the encoders and each recurrent layer

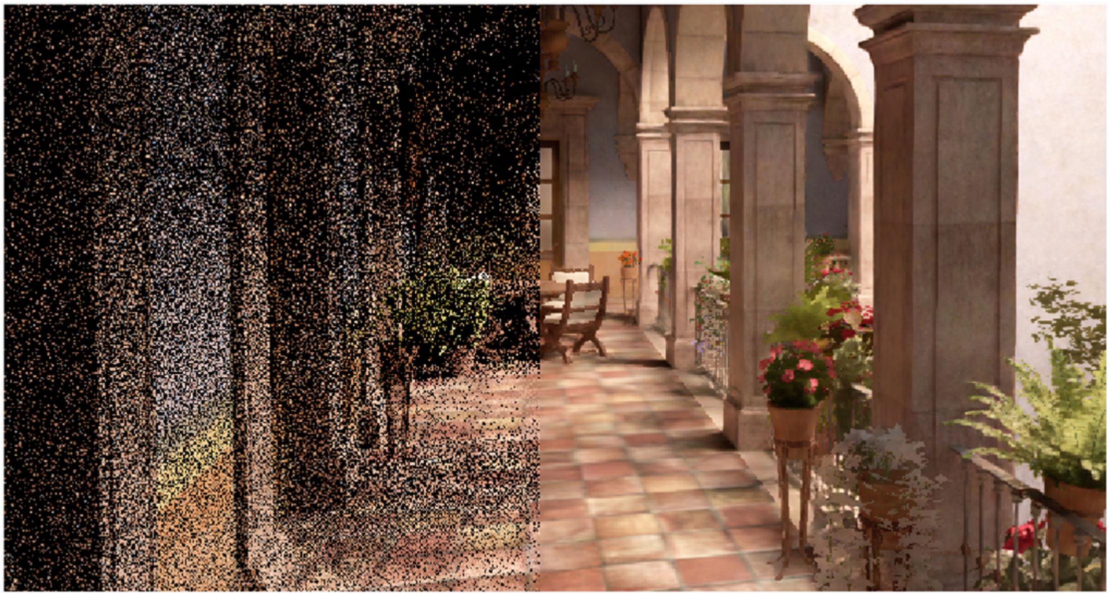

Figure 4.5: The input (left) and output(right) from a recurrent autoencoder based denoiser

produce a hidden state which is combined with features of the next frame. This passing of information makes the network temporally stable. The loss function is a combination of three loss functions. A spatial loss computed using mean absolute distance between the image produced and the ground truth. Spatial loss is useful in retaining structural information. A gradient loss is used for penalizing the difference in fine details. This is computed as a absolute difference between image gradients of predicted image and ground truth. Finally, temporal loss function is defined as the absolute difference between temporal gradient of adjacent predicted image and the ground-truth. The results from this network is shown in Figure 4.5.

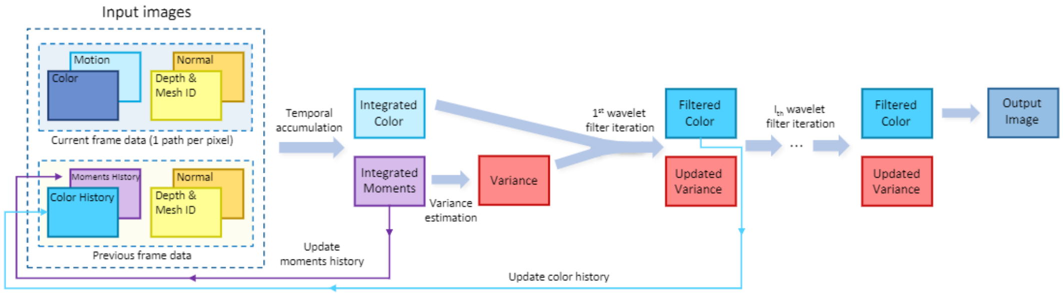

Figure 4.6: An overview of SVGF approach. Temporal variance is estimated if the history sample is valid otherwise fall back to spatial estimate and this controls the spatial filter weights that is applied iteratively.

4.2.4 Spatiotemporal Variance Guided Filtering (SVGF)

Schied et al. [74] proposed a fast edge-avoiding A-Trous wavelet filter approach specifically aimed for real-time. Schied noted that real-time can only afford one sample per pixel or less and reconstruction at very low sampling rate is challenging. It is difficult to find the source of the noise when we sample a high frequency region. In SVGF, to make the job of the denoiser easy they first decouple the noisy signal into direct and indirect light signals. They also demodulte the albedo containing texture details so that an aggressive filter can be applied without overblurring the textures.

The core idea is to analyze the noisy input over time and apply blur based on the variance of a region. The temporal accumulation step is the same as the temporal integration in TAA (Sec 3.3.4) and use G-buffer to check for consistency. The current sample is reprojected to the previous frame which effectively increases the amount of samples used. The local variance is estimated by computing the first and second raw moments. However for the first few frames, there exist only few samples to estimate the temporal variance and so a spatial variance estimate is used until enough samples are accumulated. The temporal filtering is followed by spatial filtering that utilizes an A-Trous wavelet transform for hierarchical filtering over multiple iterations. In every iteration the receptive field gets larger but the number of kernel weights used remains the same.

There are several limitations to this approach:

- **Heuristics based temporal accumulation** - The validity of the reprojected sample is determined using handmade heuristics that doesn't cover all the edge cases causes artifacts like ghosting and flickering to appear similar to TAA.

- **Residual noise** - Even with the iterative filtering, the technique fails to get rid of all the noise in the scene when the variance estimate is unreliable especially in low lit regions, or if stochastic depth of field or motion blur is used. Furthermore, in the case of extreme aliasing in a high frequency region, the technique has difficulties figuring out the noise and hence the noise remains in the final image.

- **Over-blurring** - During the initial filtering the scene features are preserved but lot of noise will remain, as we do more spatial filtering, the filters can over-blur the sharp features.

- **Detached shadows** - When there is motion in the scene, a delay in the movement of shadows can been seen as a result of slow update and validation of history samples.

4.2.5 Analytical Denoisers

Analytical denoisers are designed based on the observation that the noise characteristics is different over a local region per effect [47]. The illumination components are split into different buffers and separate custom filters are designed to filter the noise more aggressively. Unlike statistical techniques, analytical denoisers take advantage of scene properties such light radius, hit distance and BRDF lobe and so on.

The shadow denoiser for filtering area light shadows generate kernels based on a frequency analysis of local occlusion [46]. This technique takes into account the type of light, its size, shape and hit distance to derive an anisotropic filter. The spatial filtering is followed by a temporal component to increase the effective sample count. The temporal component work similar to TAA. Since the denoiser requires certain information from the light source, a separate pass is required for each light source increasing the denoising cost.

The reflection denoiser derives a screen space anisotropic kernel that takes BRDF, hit distance, roughness and normal into account![29]. The spatial filter is followed by a temporal filter however the velocity vector based on the geometries are not valid for reflections. To overcome this limitation, the motion vector of the reflected objects are computed by modelling a local pixel area as a thin lens and using the lens equation.

The diffuse indirect denoiser (GI) applies a world-space kernel based on frequency analysis of indirect light-field using an axis-aligned filtering approach reducing it to a image space blurring [47]. The filter size is adaptive with respect to the hit distance avoiding over blurring. Similar to other denoisers, the spatial pass is followed by a temporal pass.

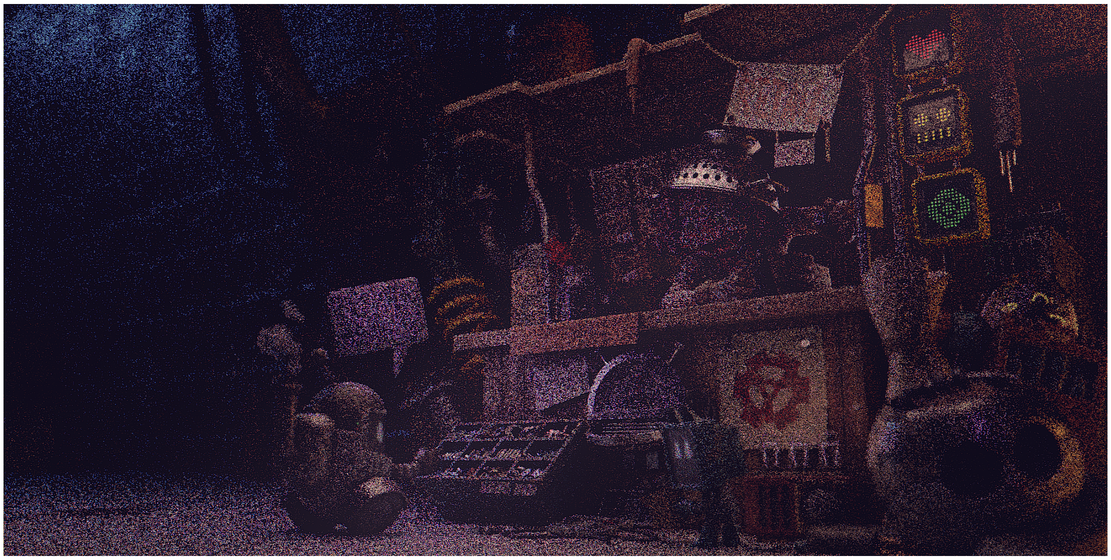

Figure 4.7: A simple scene rendered with 1 sample per pixel in Blender.

4.3 Real-time Neural Denoiser

The main goal of the our denoising network is to overcome the limitations of the existing real-time denoising solutions and produce high-quality temporally stable images. The network-based approach replaces the validation and reification of the history sample with learned temporal blending weights producing less ghosting and flickering in the final image, similar to QW-Net for antialiasing. The autoencoder-based filter network provides a high receptive field with low computational cost and produces complex filters that remove noise while retains high-frequency details. The quantized nature of our network enables the denoiser to be a real-time solution.

A trivial experiment is to replace the training dataset for QW-Net with noisy input, and its path traced reference and retrain the network. Unfortunately, this approach doesn't work well

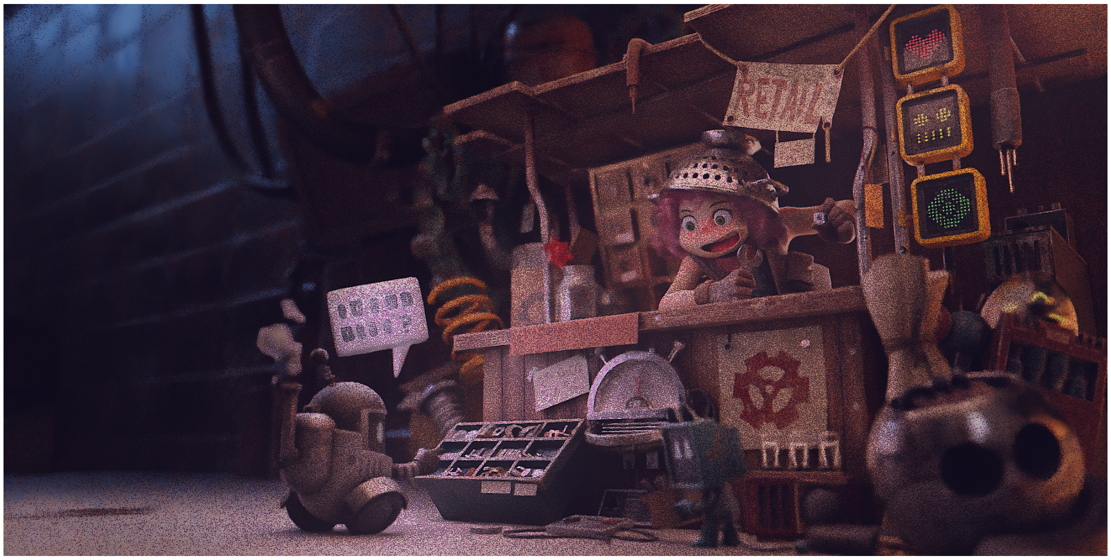

Figure 4.8: Open Image denoise output from 1 sample per pixel render.

in practice. Figure 4.7 shows a noisy image rendered with 1 sample per pixel using Blender's cycles renderer. In video games, with ray tracing, even 1 sample per pixel is a tough requirement to meet forcing developers to go even further down to 0.5 sample or half resolution (less data available for the reconstruction techniques). Intel Open Image Denoiser Network (OIDN) is a state-of-the-art AI denoiser used in production renderers for VFX and archviz. OIDN is based on the recurrent autoencoder-based denoiser [14] but without the recurrent connections and is trained on a large dataset with varying sampling rates. Figure 4.8 shows the output when the noisy input image is passed through OIDN. As seen from the output image, most of the scene is reconstructed, but there is still a lot of residual noise left, which is not acceptable for games. With animation sequences, the problem becomes even worse as noise moves around producing flickering and loss of details. With a base image rendered with 32 samples per pixel or more, OIDN works well, but with low sample counts, it fails to reconstruct the image. This shows that simply replacing the dataset and retraining QW-Net is not enough for extending the network for denoising.

When the input image is extremely noisy, the offline denoiser networks have difficulties in figuring out whether a sample is from a high-frequency detail or from a smooth gradient. From SVGF(4.2.4), We know that it is easier to decouple each noisy channel and denoise it separately. By doing so, we can filter aggressively on the more noisy channel, while retaining details in the less noisy channel. Motivated by this finding, we split light contributions into diffuse and specular.

To achieve real-time performance, we extend the QW-Net architecture to have a single feature extraction network that takes all the noisy input buffers and the auxiliary features, extracts and shares the features with the multiple filter network. The filter network can either mix the features or choose to ignore them when predicting the per pixel filter weight at each filter stage. Sharing features has the advantage that when edge features are noisy in one buffer, less noisy features can be used to drive the filter generation. Even though this architecture requires the use of multiple filter networks, the computational overhead is considerably lower than that of the feature extraction network.

4.3.1 Network Architecture

Figure 4.9 shows the architecture of our denoising network. Similar to QW-Net for supersampling, the denoising network consist of a single U-Net based feature extraction with a series of encoders followed by decoders and multiple filter stages.

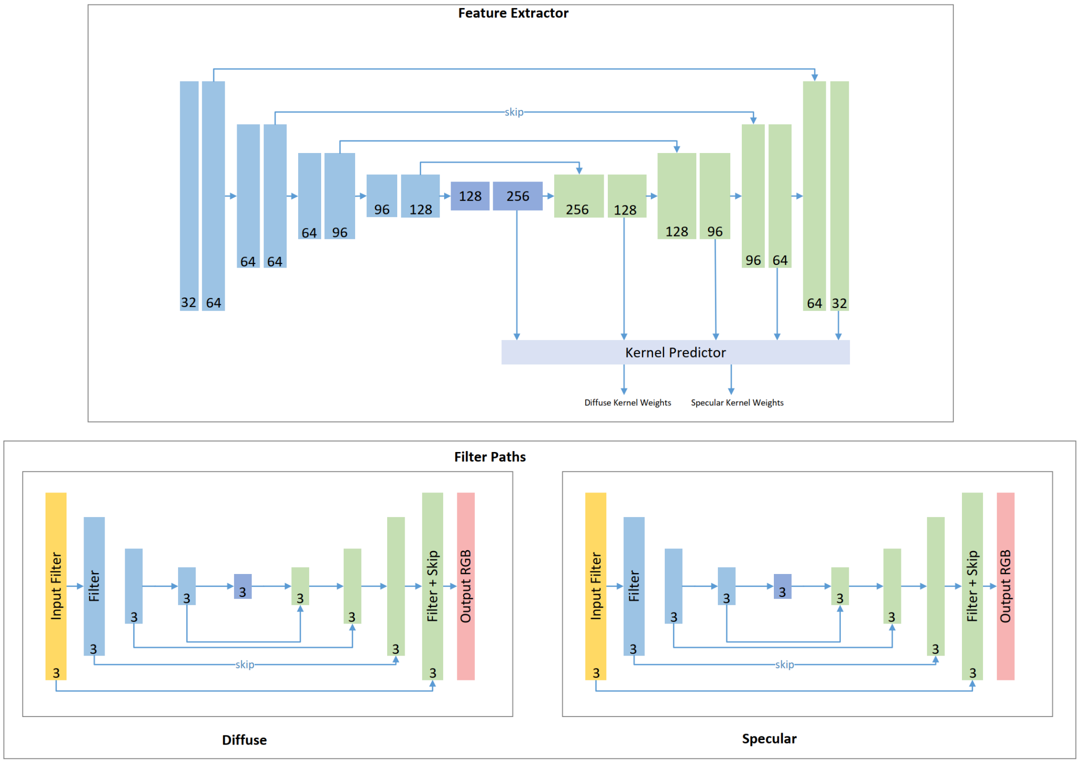

Figure 4.9: The denoising network architecture including the feature extraction network (top) and the two filtering networks (bottom). The number of output channels is shown at the bottom of each stage of the network.

The main difference with the supersampling feature extractor is the higher capacity of the network and the use of two convolution layers in each encoder stages. Additional layers and parameters are necessary to extract features from sparsely sampled input as shown in Figure 4.10. The input to the feature extractor is the noisy illumination components and the geometry buffers. To compress the dynamic range inputs, a tonemapper is applied before sending to the first encoder stage. Diffuse and specular input are filtered separately by two filtering stages. The input to the filter stage is the current noisy radiance and the corresponding reprojected previous denoised buffer. The filtering is applied in the linear space since applying tonemapping function to a noisy input results in energy loss which darkens the denoised output. Both filter stages are hierarchical in nature and generates kernel weights from the corresponding decoder stages in the feature extraction. In addition to spatial filters, a temporal filter and blend weights are also generated to accumulate previous samples and intermediate images after filtering.

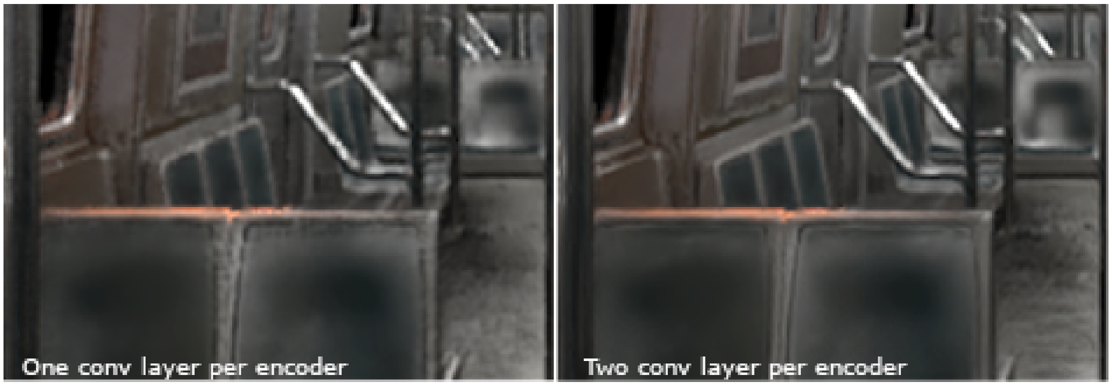

Figure 4.10: Using two convolution layer per encoder (right) produces cleaner image compared to one convolution layer per encoder (left)

The final composited image is obtained by modulating the albedo with the denoised diffuse and combining it with the specular component:

$$OutputDenoisedFrame = Albedo * Diffuse + Specular \tag{4.2}$$

4.3.2 Result

4.3.2.1 Dataset

The Unreal Engine (UE4) shaders were modified to extract seperate signals for each effect. The noisy inputs are captured with jittered one sample per pixel. The direct illumina-

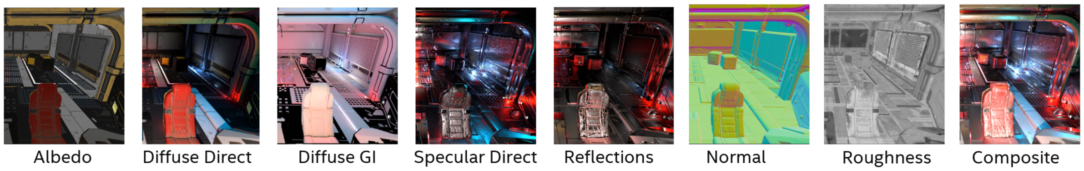

Figure 4.11: Seperate buffers extracted from Unreal Engine. The compositing is done in a post-capture step. The composited image is only used in the training

tion effect such as diffuse direct and specular direct are captured using 1 ray per light source while indirect effects are captured with 1 ray per pixel. The reference signals are captured with 256 jittered samples followed by an accumulation step. The G-buffers are extracted from the traditional deffered rendering pipeline which is useful in hybrid rendering to get the first hit point in ray tracing. All shaders in the pipeline that follow ray tracing are disabled, including supersampling and post-processing effects, in order to isolate the effects captured.

We captured seven cinematic scene that are publicly available with different geometry, light sources and materials. Every sequence in a scene includes camera animation and one that includes character animation. We captured five sequences with 60 frames for each scene which is split into a training set and a test set. The capture resolution was 1280x720 pixels and took around two weeks to complete. For training, 256x256 pixels patches from 8 consecutive frames were created with a spatial stride of 192 pixels and a temporal stride of 6 frames for creating overlap in scene regions over different batches.

4.3.2.2 Network Analysis

Our network was implemented in PyTorch [58] and trained for 500 epochs in full-precision mode. Once the training is complete, we turn the quantization mode (INT8) on and continue the training for another 500 epoch. A ranger optimizer with exponential decay learning rate is used for both precision mode training. In addition, we introduce a new perceptual loss function for synthetic content, Photorealistic Image Feature Extraction (PIFX).

Figure 4.12 shows the result from training with different loss functions. In our experiments, we found that L1 introduces significant blur and loses scene details with low sampled inputs, despite previous denoising algorithms that successfully utilized per pixel L1 loss functions. In recent image super-resolution networks it was observed that using a perceptual loss function that operated on feature maps produced a sharper image [85].Using a weighted VGG network loss instead of the L1, we found that the claim is true, but introduces a pattern-like artifact across the image. Since VGG network is trained on natural images with real world structural patterns, it encourages the network to amplify these artifacts especially for flat surfaces. Ideally, a loss function should have the same perceptual qualities as a VGG network but trained on synthetic images, avoiding these artifacts.

To this end, we trained a network from scratch on Apple's hypersim dataset containing photorealistic renderings of indoor scenes [64]. Dataset includes 77K images with G-buffers, illumination buffers, and per-pixel semantic instance segmentation labels. Our loss network (PIFX) is a U-Net with skip connection trained for predicting a segmentation mask which is close to a human level perceptual task. This loss is defined over a feature map of the first three layers and is weighted so that each layer contributes equally. The first layer is intended to focus more on high frequency features such as edges, while the additional two layers will extract abstract shapes and textures.

Our denoising network applies the custom perceptual loss separately to two filtering paths, and another loss over the composite output and reference output. Our loss network is trained on the composited image of the hypersim dataset, but they also provides individual buffers, so we can train loss networks optimized for each signal separately. Our initial experiment with this approach shows promise as shown in Figure 4.13. We pair the perceptual with a L1 temporal loss giving equal contribution to the total loss.

4.3.2.3 Image Quality

Figure 4.14 to 4.16 shows noisy, denoised and reference composited frames from the Subway scene that resides in the training set. We do not require composite inputs or produce a composite output. The compositing happens in a separate stage after the network inference according to eq 4.2. Our network produce denoised results that comparable to the reference composited captures. The breakdown of each signal with it's corresponding denoised version and reference is shown in Figure 4.17 to 4.22. Our network also shows the ability to generalize across test scenes as shown in Figure 4.24. Compared to the extremely challenging noisy input (Fig 4.23), our approach reconstructs a plausible output that is close to the reference shown in Figure 4.25.

We show comparisons between analytical denoisers and our results for Dock and Scifi scene in Figure 4.26 and Figure 4.27. The analytical denoisers were captured similarly to our noisy input capture pipeline except we turn on the shaders responsible for per effect denoisers. All denoisers rely on TAA for better temporal support and suppressing fireflies. In a real game, the TAA shader is applied to the composited image. In our capture, we apply TAA to each effect after denoising, resulting in better image quality. In addition, we modulate the reference albedo with the denoised diffused component to prevent aliasing in the results, thereby enhancing their quality. Essentially, the result that we present here for comparison is the highest quality analytical denoiser that is not feasible for a game. Native analytical denoising results in a lot of residual noise with a loss of details. While TAA improves this result, it does not create a clean image. Our result is clean, sharp with scene details close to the reference. Here the Dock scene is from the training set while the Scifi scene is from the test set. The Dock and Scifi temporal profiles illustrate how we achieve temporal stability through our network, whereas analytical approaches have discontinuities, as shown in Figure 4.28. We show PSNR and SSIM metrics for quantitative comparison of different techniques in Figure 4.29. Our network consistently outperforms the analytical denoiser with and without TAA. While there is a dip in the metrics for a few frames, visual inspection showed no degradation in the image quality.

Lastly we show how our network overcomes some of shortcomings of analytical denoisers. Figure 4.30 shows three consecutive frames after both techniques were applied. In the animation, the character's hand is moving across the frame covering the part of the plant. As the hand moves away from the visible region, the plant is disoccluded with no history information to recover from. Analytical denoisers with TAA does not recover from disoccluded samples, while with our technique we generate filter weights to increase the blurring in those regions by utilizing temporal feedback. When TAA pass is added to the denoiser, it is likely to cause reprojection errors that result in ghosting as shown in Figure 4.31. Our network learns to generate temporal filter weights along with per-pixel blending weights that avoids ghosting. TAA tends to blur the output a lot even after modulating with the reference albedo. By using our custom perceptual loss and kernel weights of range [-1,1], our filter kernel weights can also act as a sharpness filter for preserving high-frequency details.

4.3.2.4 Performance

As a preliminary estimate of performance, we take into account that modern games use 100-1000 light sources and that with recent advancement in sampling, the number is now increasing to millions. As stated earlier, several analytical denoisers need to be executed per light, which increases the overall cost proportional to the number of lights affected by the pixel. In contrast, our network's cost remains the same regardless of the scene configuration in real-time as shown in Figure 4.33. To extract the cost in an actual game requires an engineering effort to develop optimized kernels. We consider such implementation outside the scope of our work.

4.4 Conclusion

Denoising a ray traced image using a neural network can produce undesired result if the input is heavily undersampled. One way to improve the result is to split the light contribution into different frequency component and design specialized networks to denoise these signals separately. Unfortunately, running separate network for each signal is not practical for real-time usecase. In this work, we show that with a single feature extractor, we can generate kernel weights for multiple filter paths making the network real-time friendly. Our network produces denoised results that are significantly better that the analytical denosiers used in ray traced games.

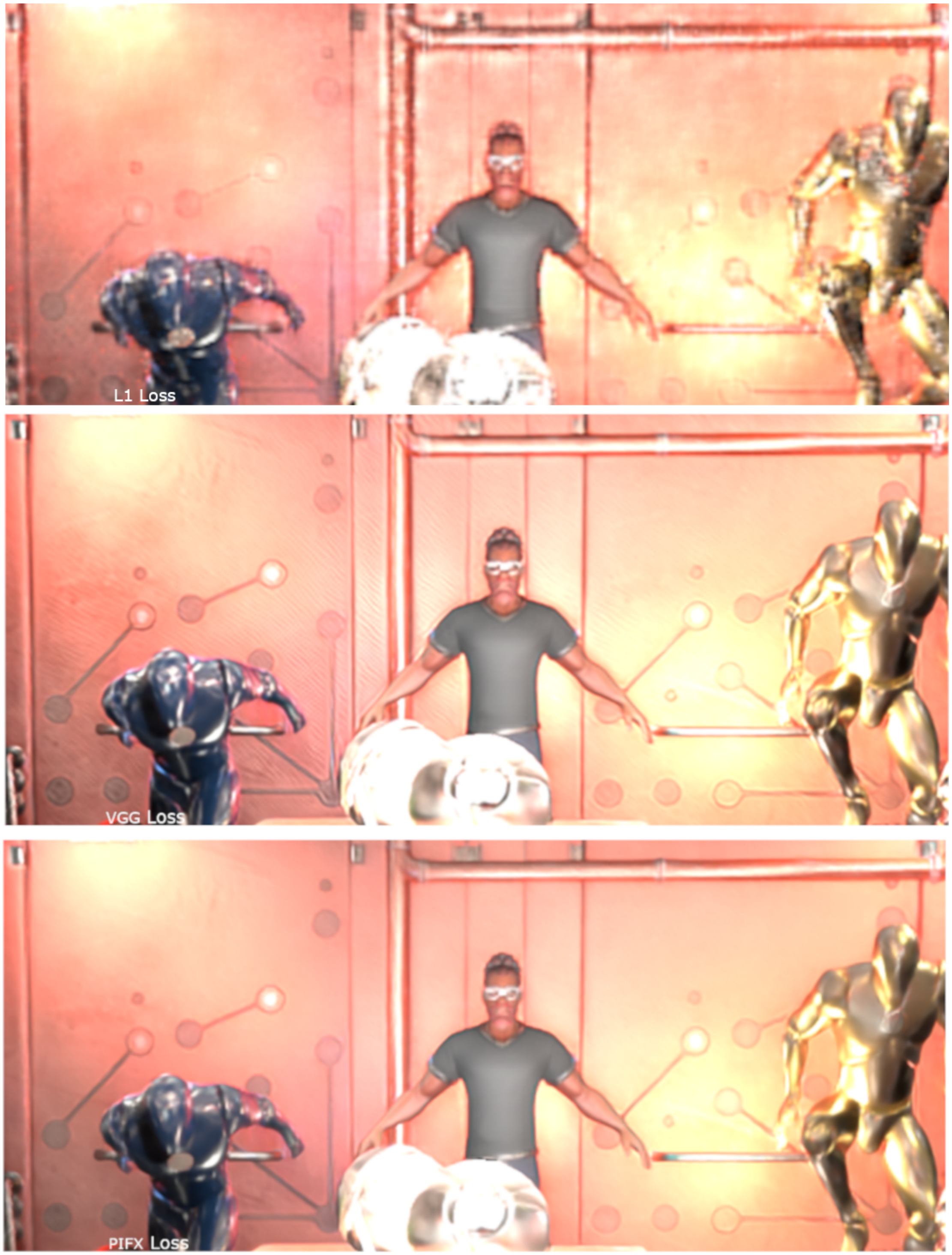

Figure 4.12: Comparison showing results from training using different loss functions. PIFX loss is able to produce sharp images without pattern artifacts

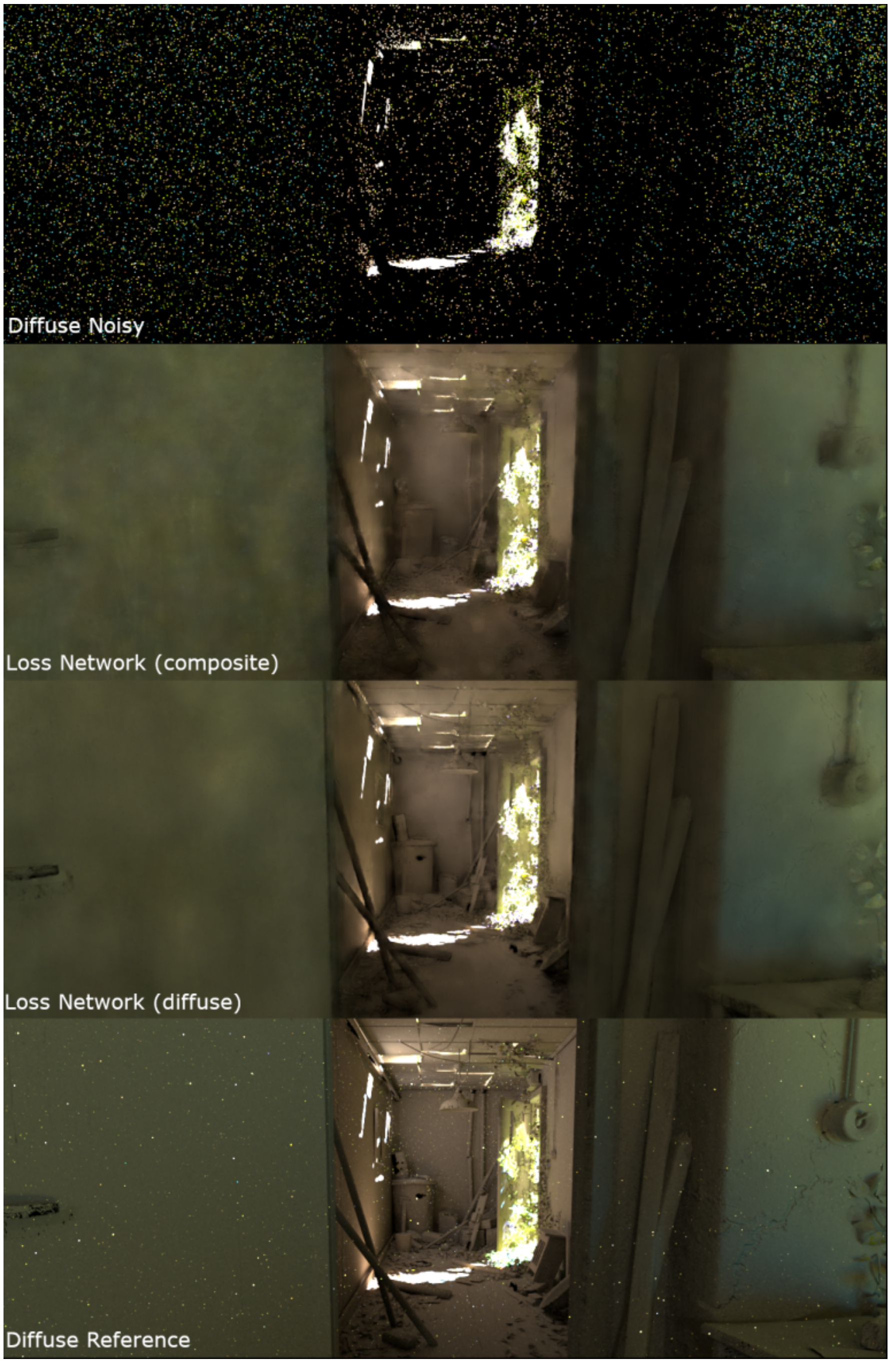

Figure 4.13: Noisy diffuse signal (first row), denoised result using a composite loss network (second row), denoised result using a diffuse loss network (third row) and the diffuse reference (last row)

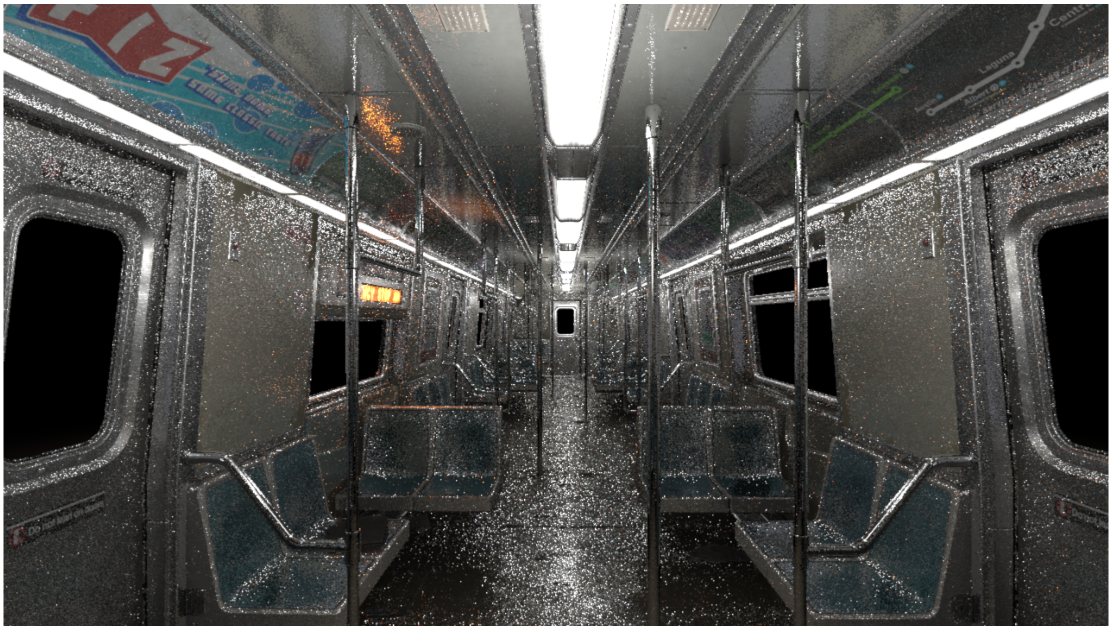

Figure 4.14: The noisy composite. This is a composite from the noisy buffers and not an input to the network.

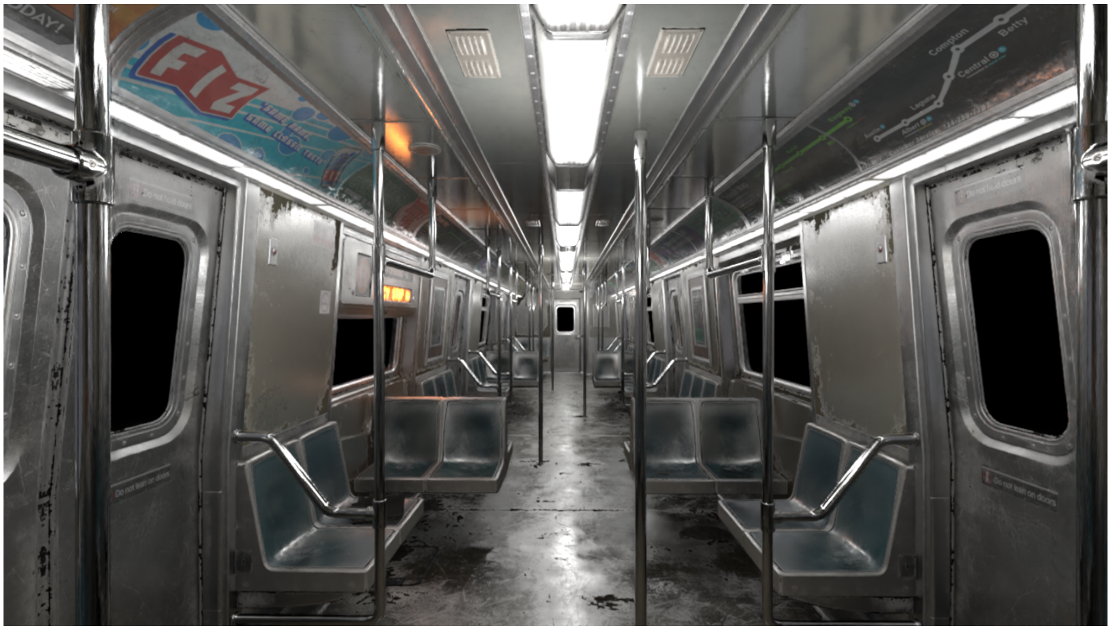

Figure 4.15: The denoised composite. The composite is obtained from combing multiple denoised buffers.

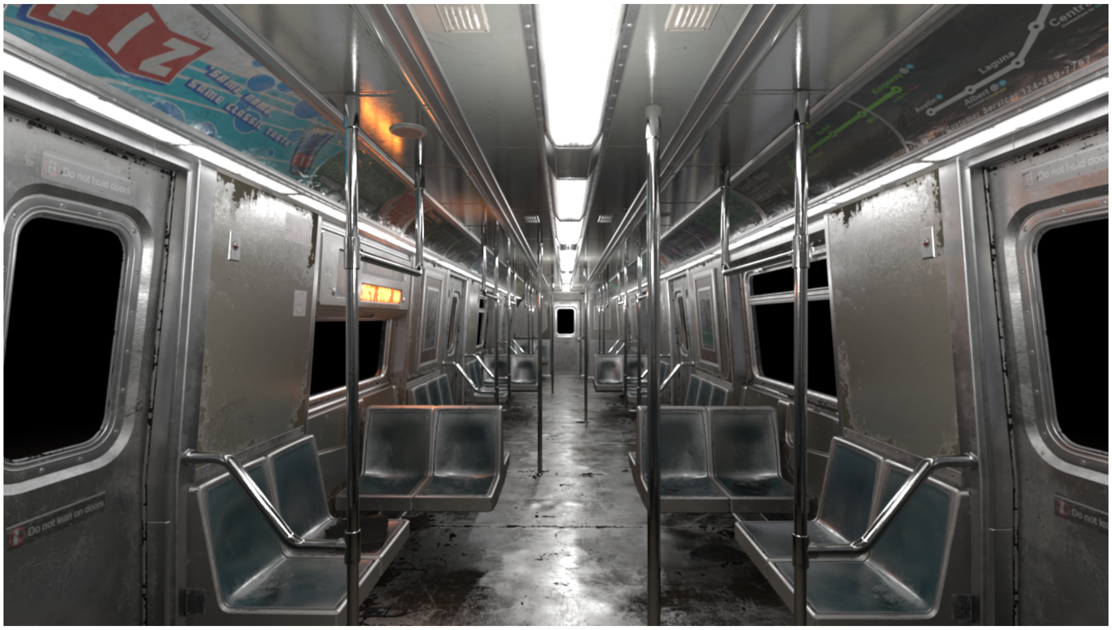

Figure 4.16: The composited reference

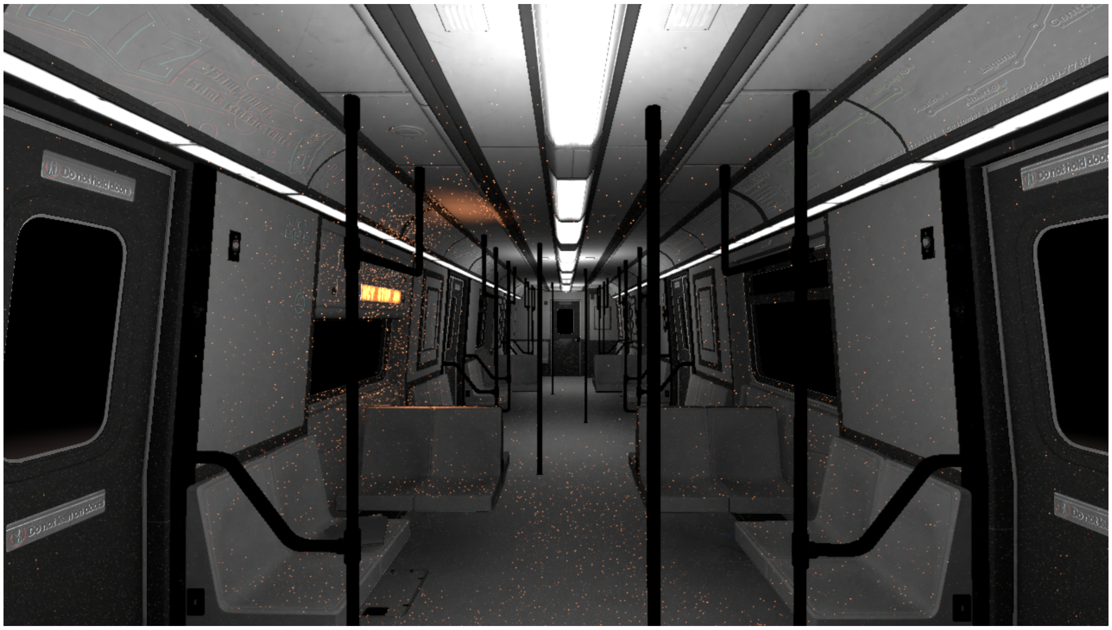

Figure 4.17: Noisy diffuse signal. The signal is obtained by combining diffuse direct and diffuse indirect buffers.

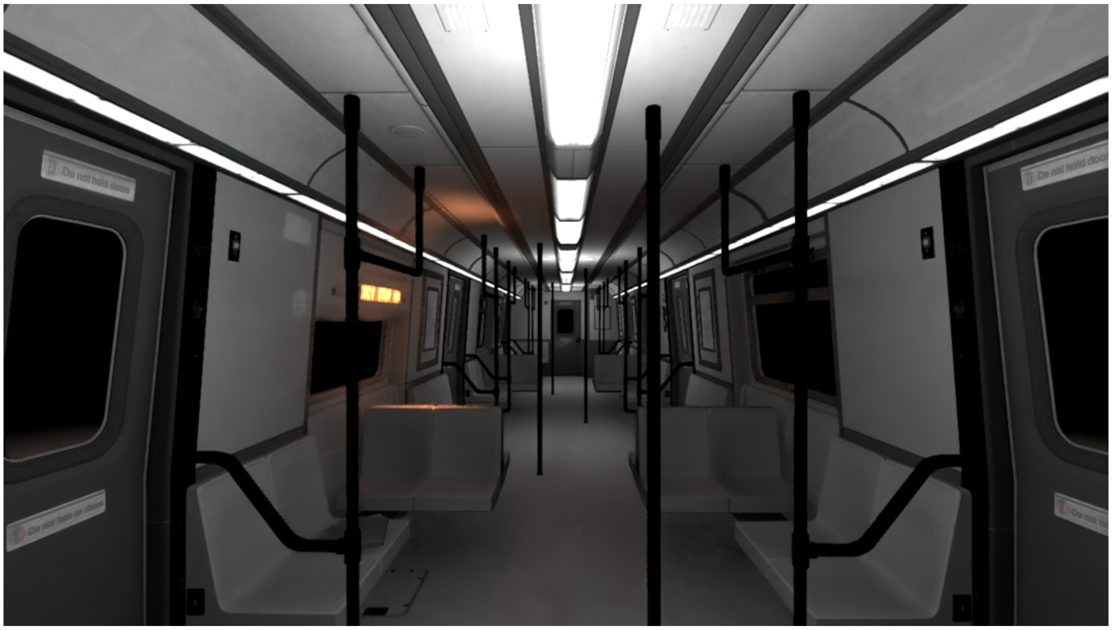

Figure 4.18: The denoised output from the diffuse filter path of our network.

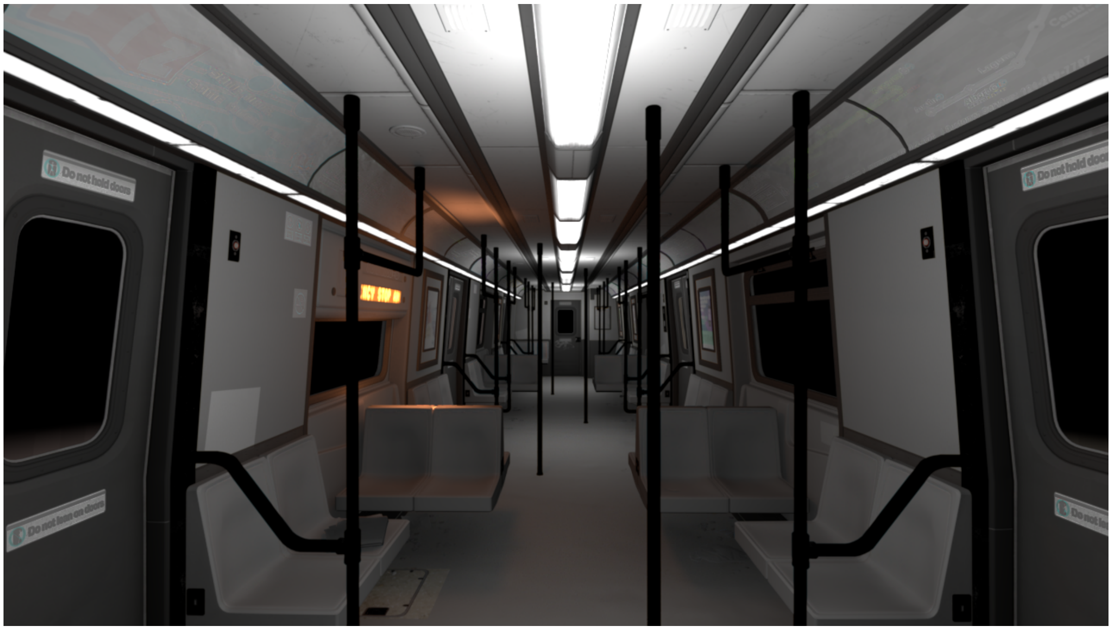

Figure 4.19: Reference capture for diffuse signal

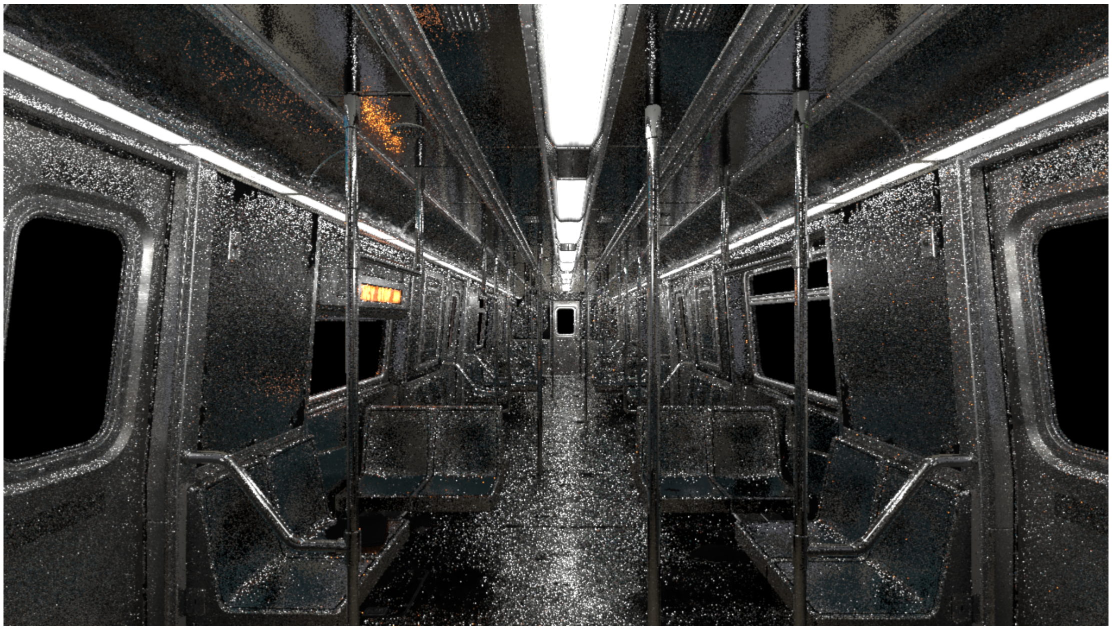

Figure 4.20: Noisy specular signal. The signal is obtained by combining specular direct and reflection buffers.

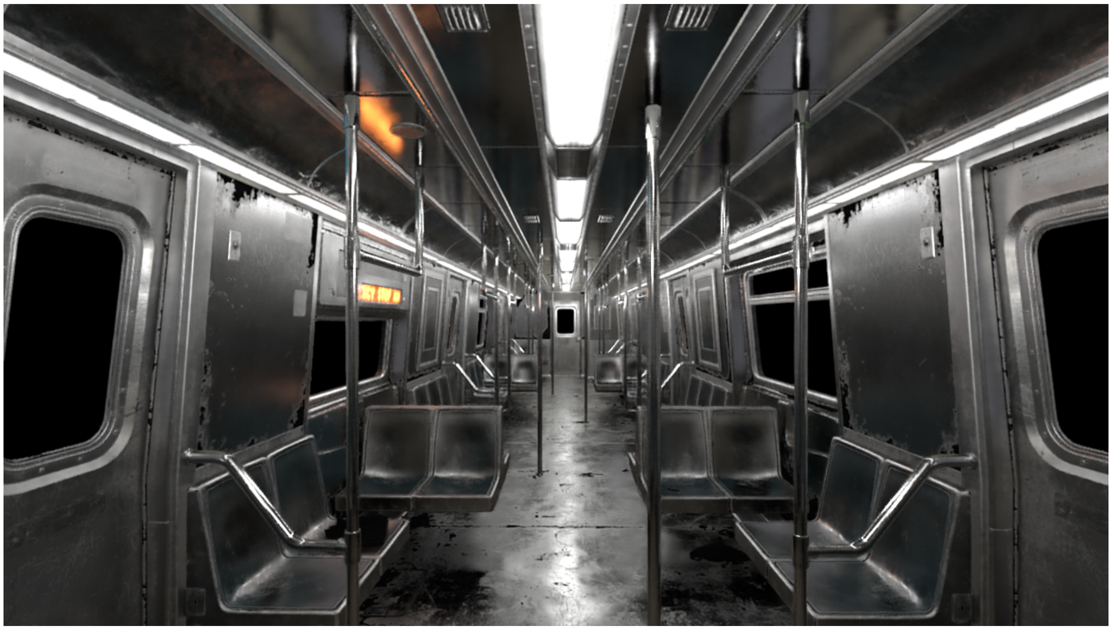

Figure 4.21: The denoised output from the specular filter path of our network.

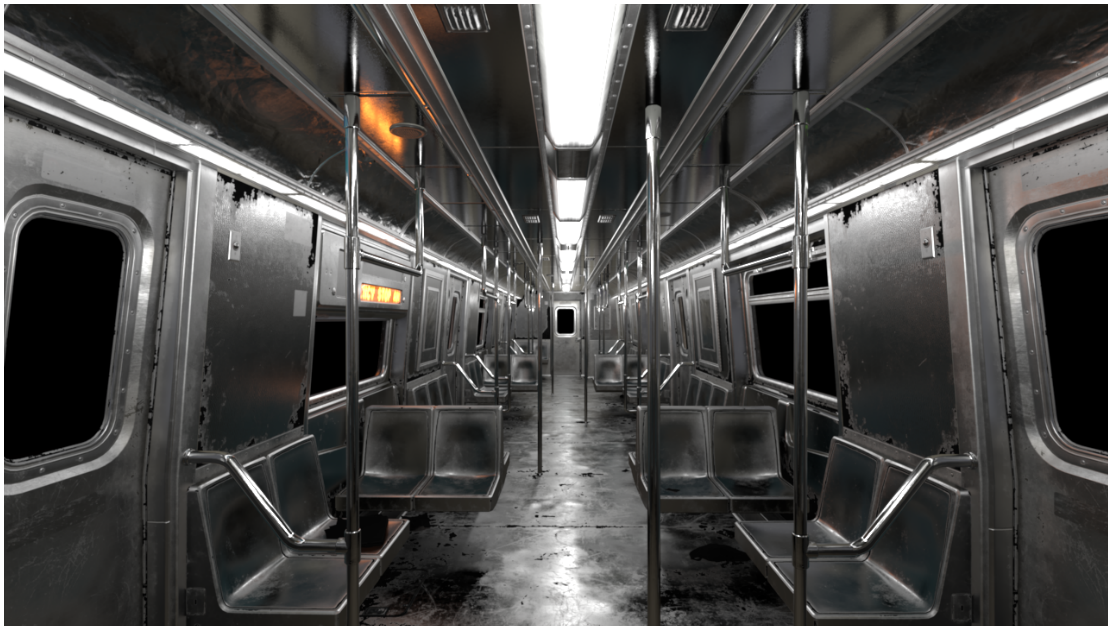

Figure 4.22: Reference capture for specular signal

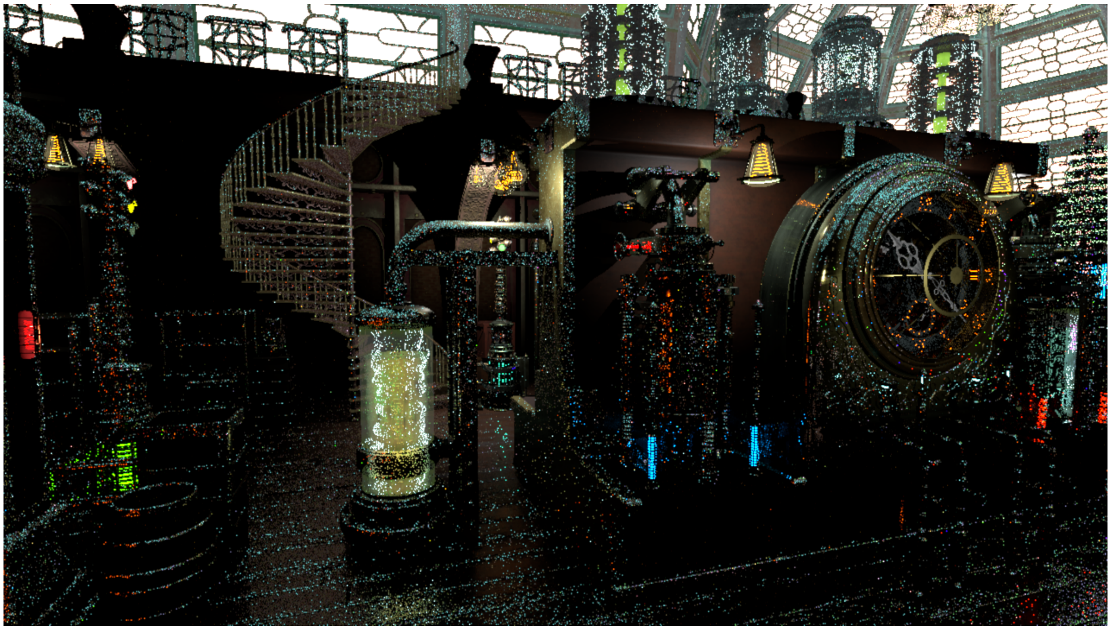

Figure 4.23: Noisy composite frame from Lab scene. The sequence is in the test dataset

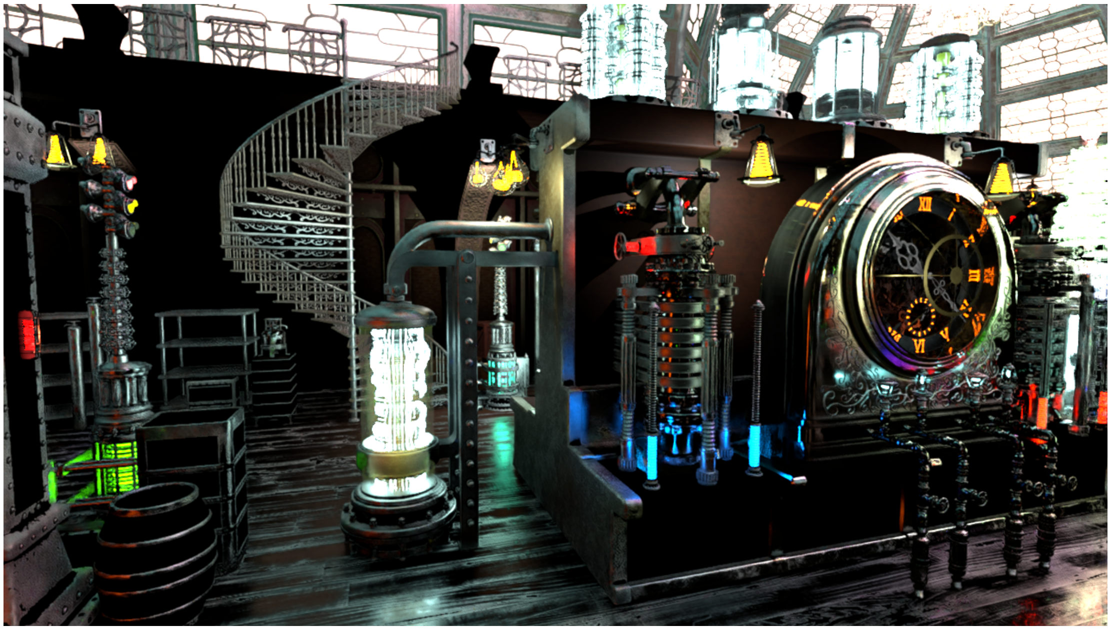

Figure 4.24: Denoised composited output from our network for the frame from Lab scene.

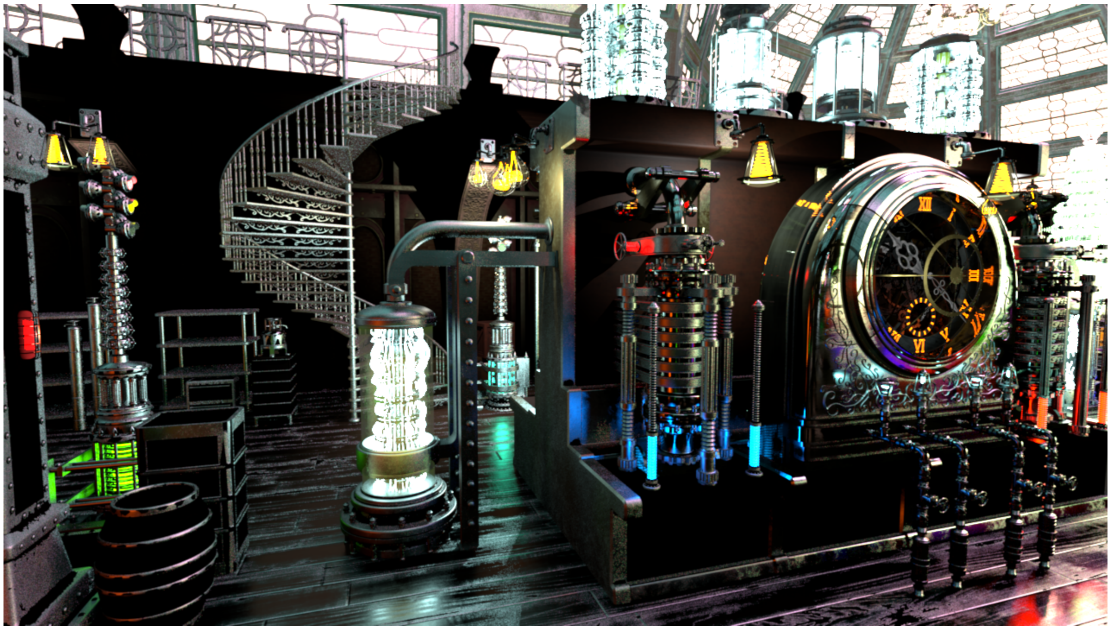

Figure 4.25: Reference Composite for the Lab scene

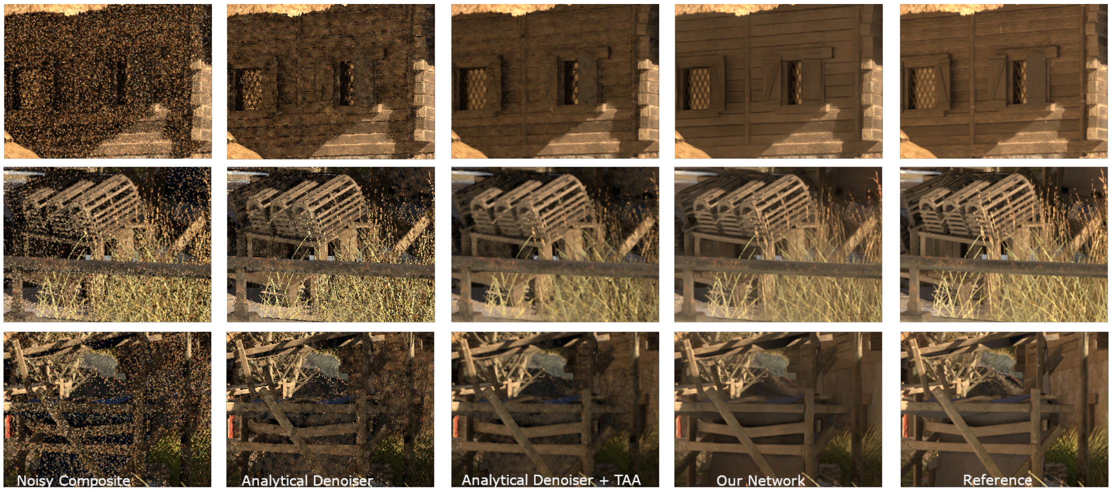

Figure 4.26: Side by side comparison of our technique with analytical denoiser with and without TAA applied for Dock scene

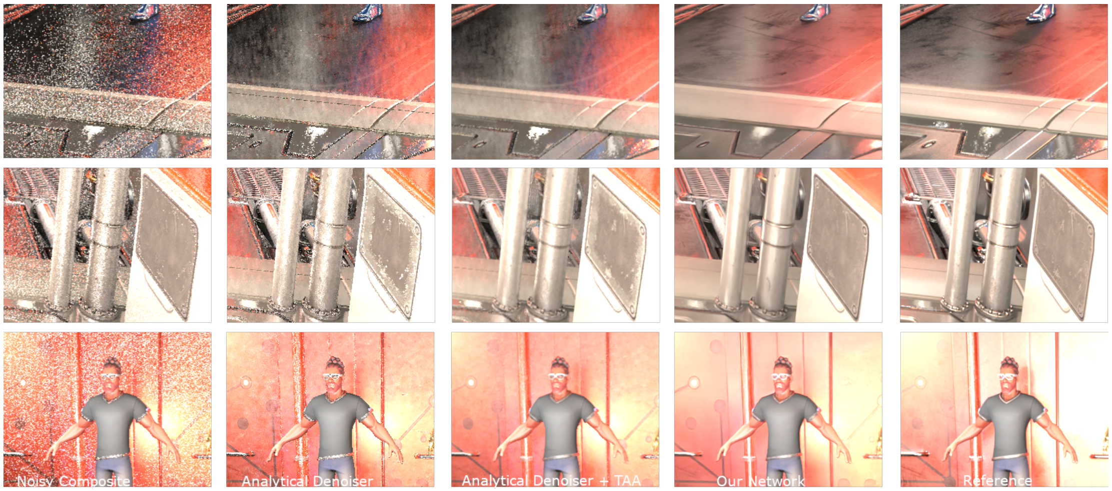

Figure 4.27: Side by side comparison of our technique with analytical denoiser with and without TAA applied for Scifi scene

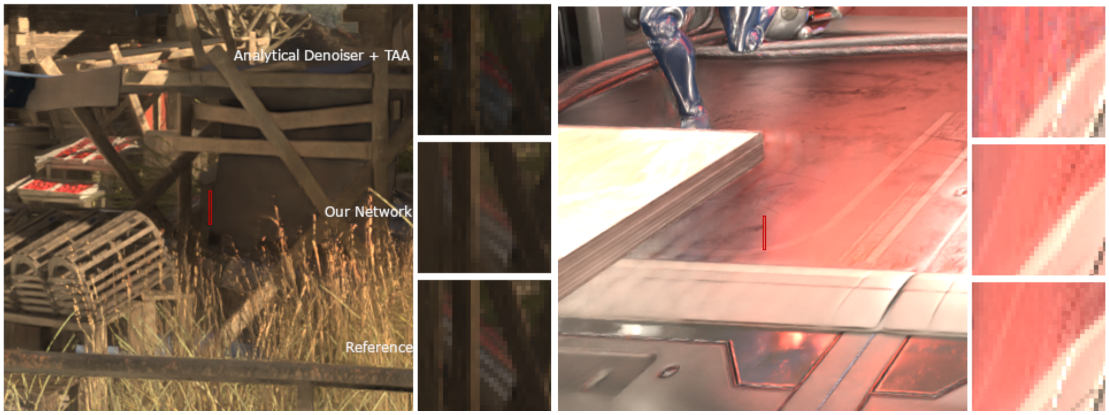

Figure 4.28: Time slice shown for a small segment in Dock and Scifi scene. Our network produce a smoother result while analytical solution contains discontinuities.

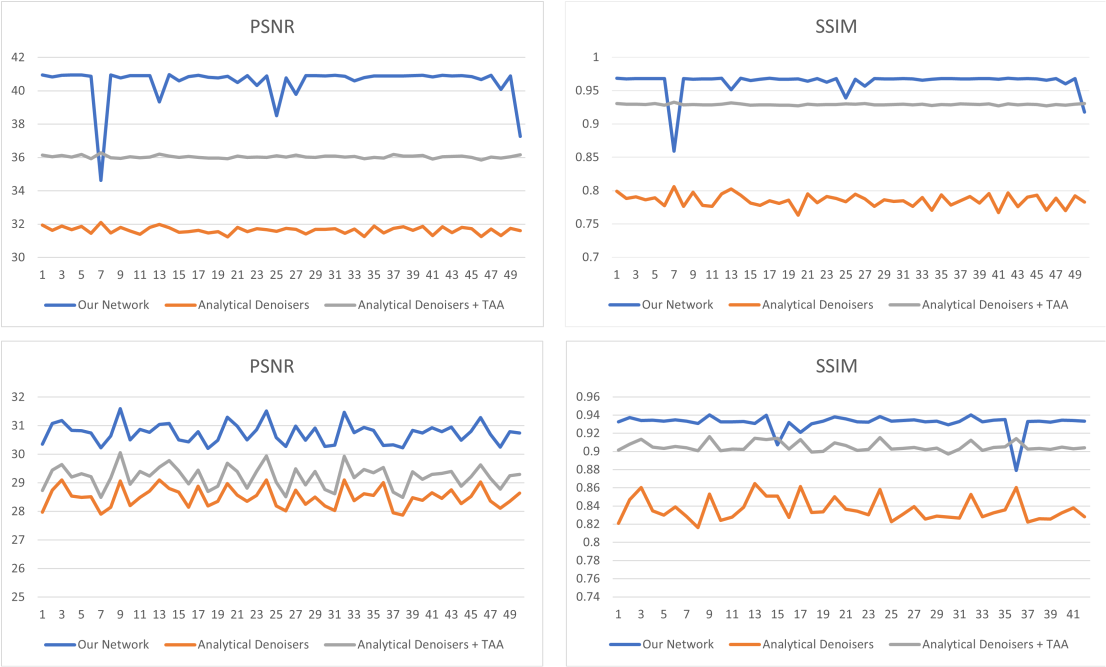

Figure 4.29: Per-frame quality metrics on a continuous frame sequence from the Dock and Scifi dataset. Our network consistently outperforms analytical denoisers.

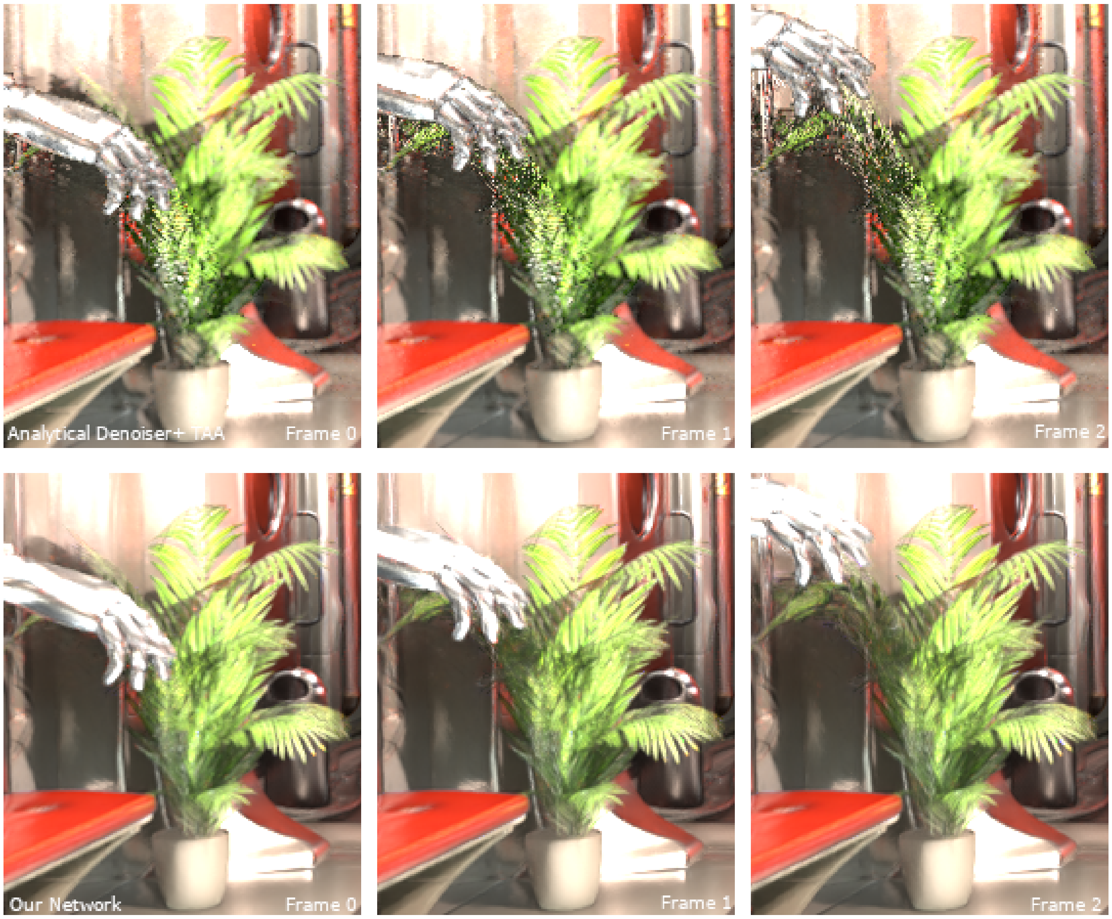

Figure 4.30: Handling disocclusion is challenging for a denoiser. Our network (bottom) blurs more in the disoccluded region while analytical denoiser with TAA (top) failed to detect the noisy regions.

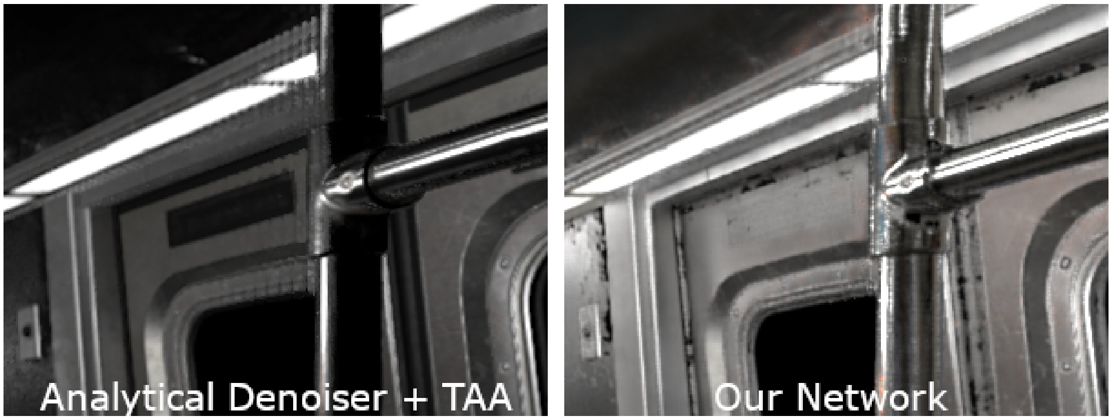

Figure 4.31: TAA improves analytical denoisers but introduces ghosting artifact (left). On the other hand our network show less ghosting (right)

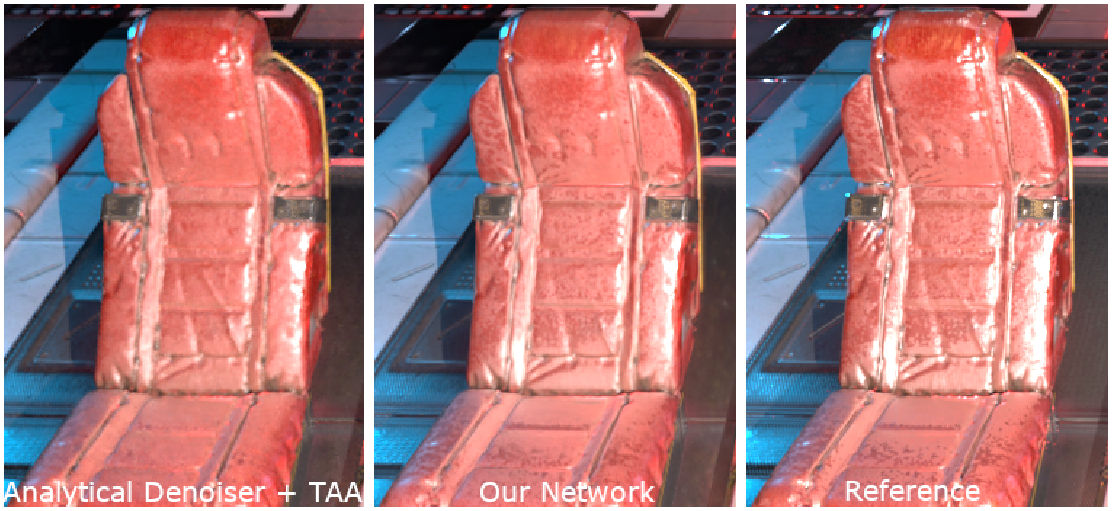

Figure 4.32: Our network is able to create complex filters that can retain high frequency detail giving a more sharper look (middle) than the analytical denoisers(left). Our results are comparable to the high ray count reference (right).

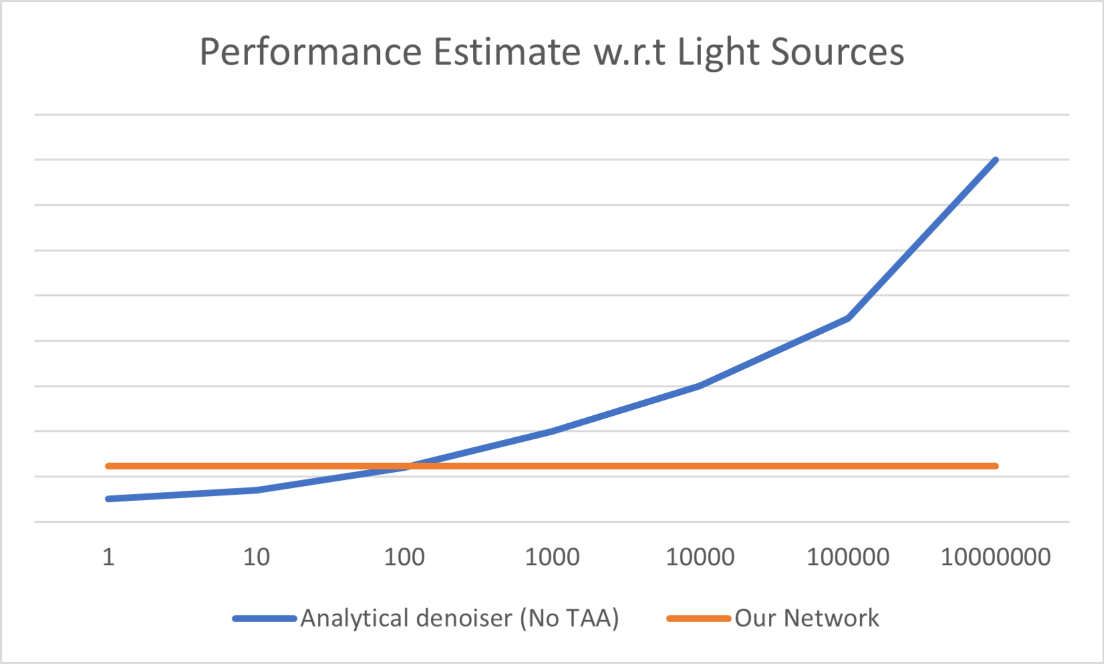

Figure 4.33: Relative performance of our network compared to analytical denoiser as the number of light source in the scene affecting a pixel increases.

Chapter 5

Joint Neural Denoising and Supersampling

5.1 Introduction

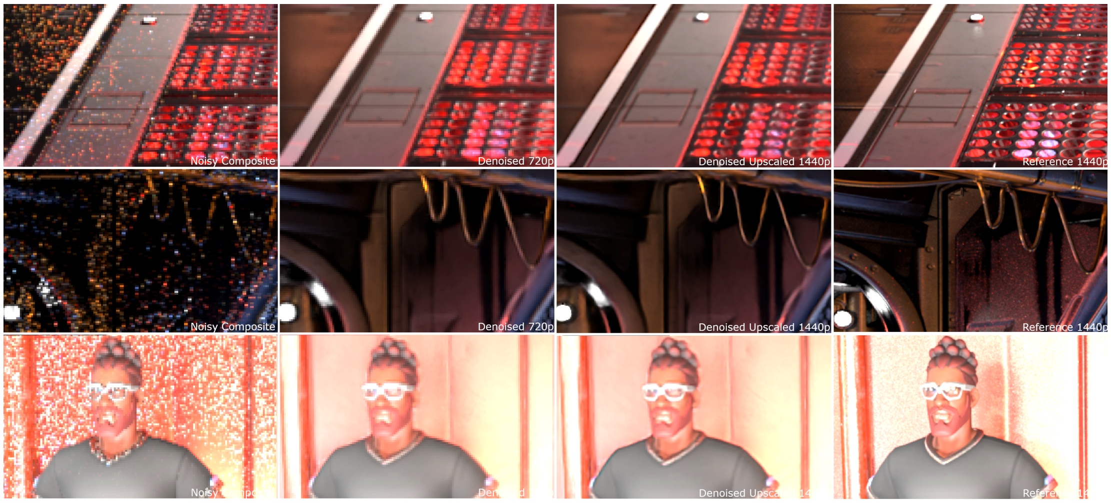

Figure 5.1: Our joint neural denoising supersampling network produces significantly sharper high resolution output (third column) compared to the our neural denoiser results. Input composite is 720p, output composite is 1440p. (Zoom to see more details)

Ray tracing adds realistic light effects to games producing high fidelity imagery comparable to offline renderers used in films [29]. Rendering a high-quality ray traced image in real-time budget with high spatial and temporal resolution is extremely challenging even with the specialized hardware due it's computational cost. To achieve real-time performance, the signal is undersampled increasing the variance and discontinuities that is visually translated to as noise and aliasing in the image [91, 98]. Many clever techniques exist that utilizes samples over time to reconstruct a high-quality final image. Doman specific spatiotemporal filters combines the sparse samples reducing the variance by trading scene details. An additional temporal filtering is usually required to clean the residual noise. The widely adopted Temporal Antialiasing (TAA) technique reprojects samples from previous frame using the renderer generated motion vector and apply heuristics to accumulate with the current frame samples[37]. However, when heuristics fails, TAA suffers from visual artifacts in the form of ghosting, over blurring and flickering.

Neural reconstruction techniques are gaining popularity among rendering community due to their ability to produce high-quality images from low sampled inputs through an end-to-end learning process. DLSS [4] and XeSS [3] takes aliased low-resolution input and reconstruct a high resolution antialiased output. Though the details of these networks aren't publicly available, they are designed to work in real-time and require input to be denoised. Interactive temporal neural denoisers requires at least 2-8 samples per pixel (spp) to produce convincing results making it not suitable for games. Moreover, chaining a neural denoiser and a neural supersampling technique would be impractical in most scenarios as this would increase the cost of rendering a single frame. We introduce a novel real-time neural reconstruction technique that performs denoising and supersampling jointly using a single network. We achieve significantly better performance than using multiple analytical denoisers and a separate supersampling pass without any image quality degradation. Our frame recurrent network takes 1spp low resolution input, which is noisy and aliased, and reconstruct it to a high resolution denoised supersampled output. By leveraging reduced precision training, we attain fast inference performance making this technique suitable for game workloads.

Our techniques consist of a single feature extraction network and several filtering stages. The computationally heavy U-Net based feature extractor is set to INT8 precision and runs on a lower resolution aiding in faster execution. The features are shared with multiple filtering stages that attend to different components of the input signal. The input signal is decomposed into demodulated diffuse, specular and albedo and each filter stage is responsible for one component. At each filter stage, we generate a set of kernel weights that are hierarchically applied to achieve high spatial footprints. We also accumulate samples from warped previous frame using motion vectors generated by the renderer to produce temporally stable results.

5.2 Related Work

To our best knowledge, there exist no previous work that combines denoising, supersampling and super-resolution in a single pass for game rendering. In the earlier chapters we discuss previous works on denoising and supersampling. We will cover existing techniques used for upsampling in video games in this section.

5.2.1 Checkerboard Rendering

Checkerboard rendering (CBR) is usually used in console platforms to get a 4K target resolution [1]. The idea is to shade half of the pixels in every other frames in a checkerboard pattern. The missing pixel is filled using spatial neighbourhood and temporal reprojection. The history clamping and rectification from temporal reprojection is also applicable in checkerboard rendering.

5.2.2 Temporal Upsampling

Temporal upsampling (TAAU) is an extension to the TAA algorithm where we reduce the sampling rate and modify the sample accumulation step [91]. The samples are accumulated into a high resolution buffer and since there is no one to one mapping between the pixels we need to upscale the input samples to the target resolution using Gaussian kernel over a pixel neighbourhood.In addition to the confidence weight, an additional confidence weight is derived to eliminate low quality upscaled samples that could increase blurring in the final output.

5.2.3 FidelityFX Super Resolution

FidelityFX super resolution (FSR) is spatial upsampling technique with no temporal feedback. The basic idea of FSR is to find and reconstruct high-resolution edges from a low resolution input. To do so an Edge-Adaptive Spatial filter is generated by examining the gradients of the neighbour pixel. The intensity of the gradient determines the filter kernel weights. The spatial filter is followed by a sharpness filter to amplify the details in the reconstructed image.

5.2.4 Neural Super-resolution

Nerual network based techinques such as DLSS [4] and XeSS [3] take a low resolution aliased input and produces a high resolution antialiased output frame. Both techniques assumes the input to be denoised before passing it the network. There is no details publicly available information regarding the working of both techniques.

5.3 Joint Neural Denoising and Supersampling

5.3.1 Network Architecture

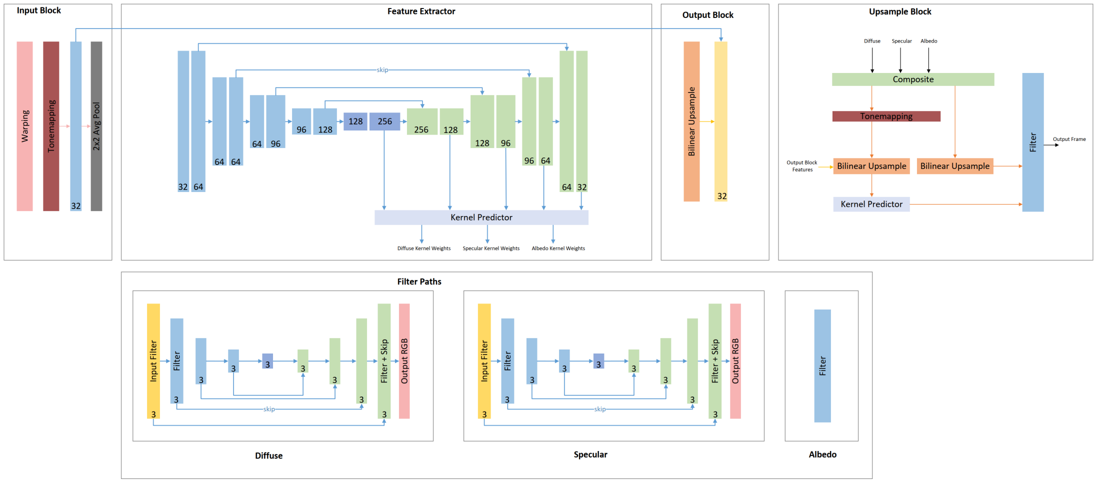

Figure 5.2: Network Architecure of our Joint Neural Denoising and Supersampling technique

Figure 5.2 shows the details of our Joint Neural Denoising and Supersampling architecture, comprising of an input block, feature extractor, filtering stages, output block and an upsample block. The input to the network is a set of untextured illumination components and auxiliary features along with per-pixel motion vector from the renderer. The noisy illumination components are combined into a low frequency diffuse signal and a high frequency specular signal similar to our neural denoiser (Ch. 4). The auxiliary features include albedo, normal and roughness that guides the network to distinguish between scene detail and noise avoiding overbluring. The input block, output block and the filtering stages operates in the native resolution while the feature extraction runs in a sub-native resolution. The upsample block filters the composited result from the filtering stage in the target resolution. The weights and activations of all the non-filter blocks are set to INT8 precision mode leveraging quantization-aware training to support fast inference performance.

5.3.1.1 Input Block

A frame recurrent approach is adopted to combine the history samples that are spatially distributed over multiple frames. To align the previous output signals with the current input, a bicubic warp function is used with the motion vector and the result is concatenated with the input passed to the feature extraction. The high dynamic range inputs are clamped to the range [0, 65535] and then compressed with a tonemapper function:

$$x' = log(x+1)^{1/2.2} \tag{5.1}$$

A 1x1 projection layer is used to get the per-pixel features followed by a set of 3x3 convolution kernels that are applied on the input to extract the high-level features. These features are skipped to the output block to recover the details lost during the pooling operation. A 2x2 average pool is used to downsample the input to the feature extraction. This downsampling step is essential in reducing the number of Multiply-Accumulate (MAC) operations per pixel to facilitate real-time inference.

5.3.1.2 Feature Extraction

The feature extraction is based on a standard U-Net architecture consisting of several encoder blocks followed by decoder blocks. The input to this network is the downsampled features from the input block. Each encoder block consists of two 3x3 convolution layers followed by ReLU activation and a maxpool operation before passing to the next stage. The decoder blocks start with a bilinear upsampling of the previous activation followed by two 3x3 convolution layers and ReLU activation function. Skip connections from the encoders are concatenated with the upsampled activation before the convolution kernel are applied to help recover the high frequency details. The activations from the decoder and the bottleneck blocks are shared by two 1x1 convolution kernels per block generating hierarchical filter weights for diffuse and specular signals that are send to their corresponding filtering stage. We don't predict kernels for albedo at this stage since large receptive field is not necessary for aliased signals.

5.3.1.3 Output Block

The output from the feature extractor is bilinearly upsampled and concatenated with the activation skipped from the input block. A set of 3x3 convolution kernels are applied before predicting the filter kernel weights. The output block generates per pixel 3x3 spatial and temporal filter kernel weights at the native resolution completing the hierarchical filters for diffuse and specular signal. Additionally, it predicts a per pixel 3x3 kernel weights for filtering albedo.

A per pixel temporal blend weight is also predicted to control the contribution of the history samples. We apply linear kernels for diffuse and specular and is clamped to [-1, 1] improving the overall sharpness while the kernels for albedo are normalized.

5.3.1.4 Filter Stages

A receptive field with large spatial footprint is necessary for filtering sparse samples. A multi-scale filtering layout similar to a U-Net architecture is used to achieve a large receptive field with lower cost instead of predicting and applying a single large filter. Unlike U-Net there are no learnable parameters in the filter stages since all filter weights are predicted from the decoder of the feature extractor and the output block. The multi-scale filter layout consists of series of downsampling filters followed by upsampling filters with skip connection. The temporal filter is applied to the warped previous output and blended with the current input before the downsampling filters. The downsampling filters apply a 3x3 kernel to the input image and downsample the result using a 2x2 average pool. The upsampling filter uses a bilinear upsample and then apply the predicted 3x3 kernel which is combined to the scaled skip image from the corresponding downsample stage. The diffuse and specular filter paths follow the same hierarchical filtering layout with unique filter kernel weights while albedo is filtered only in the native resolution.

5.3.1.5 Upsample block

The output images from the filter stages are composited following eq 4.2. The resulting composite is bilinearly upsampled twice to get a tonemapped and a linear version. The tonemapped composite is concatenated with the features from the output block and is passed to a kernel prediction stage. Similar to other kernel prediction blocks, a 3x3 per pixel kernel is predicted in the target resolution. The predicted weights and the linear upsampled composite is sent to the final filter block where the kernel weights are applied to get the final outptut image.

5.3.1.6 Quantization

The quantized layers are set to use INT8 weight and activations where the weight follow a per-channel symmetric quantization while activation uses an affine per-layer quantization. When the skip layers are combined, we use the same quantization range. The quantitation threshold for weight is set to it's maximum absolute value per channel with symmetric range. For activations, we train the threshold by setting a straight through estimator for the gradient non differentiable functions. Since we don't use batchnorm in our network, no folding of weights are required.

5.3.2 Result

5.3.2.1 Image Quality

Figure 5.1 shows the results from our neural denoiser and our joint neural reconstruction solution on scenes from test dataset. Our network is capable of producing supersampled high resolution denoised images from a low resolution noisy input with sharp details comparable to the native high resolution reference. Kernel weights clamped to [-1, 1] can effectively bring out high frequency details in the target resolution without introducing ringing artifacts.

When comparing to native 1440p rendering with analytical denoiser and TAA ap-

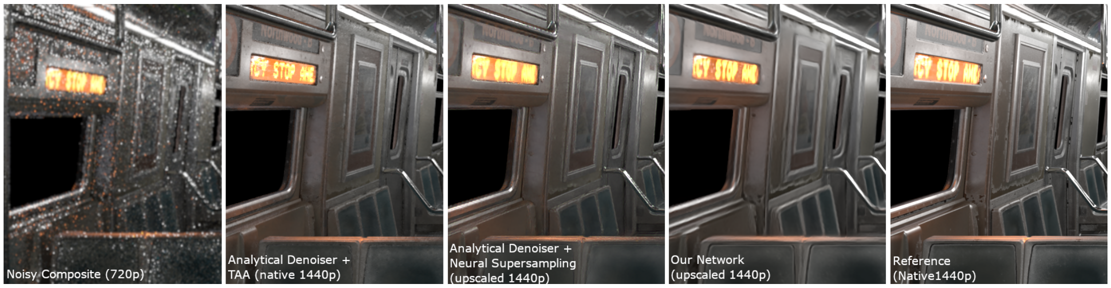
Noisy Composite (720p)
Analytical Denoiser + TAA (native 1440p)
Analytical Denoiser + Neural Supersampling (upscaled 1440p)
Our Network (upscaled 1440p)
Reference (Native1440p)

Figure 5.3: A side by side comparison of our joint technique with other native rendering and chained techniques. Our result show superior image quality despite the input being noisy and low-resolution

plied, our technique produces cleaner and sharper output with 720p noisy input as shown in Figure 5.3. Furthermore, the cost of rendering and denoising native 1440p is about four times that of 720p, making our technique desirable for games targeting higher resolution frames The PSNR, SSIM plot show that native rendering performs well when compared with analytical denosier coupled with neural supersampling (Fig 5.4). Our joint network and two reconstruction networks (Neural Denoiser and Neural Supersampling) combined outperforms native and supersampled version of anaylitical denoiser. The plot also shows dips, as we observed in the denoising network, but after inspecting the frame, we see no artifacts.

We also show that our network is generalizable over different composite functions as shown in Figure 5.5. During the training process we used a fixed composite function (eq 4.2) to run the loss network and here we show that the upsample block can adapt to frame composition that uses different weighting function for illumination signals.

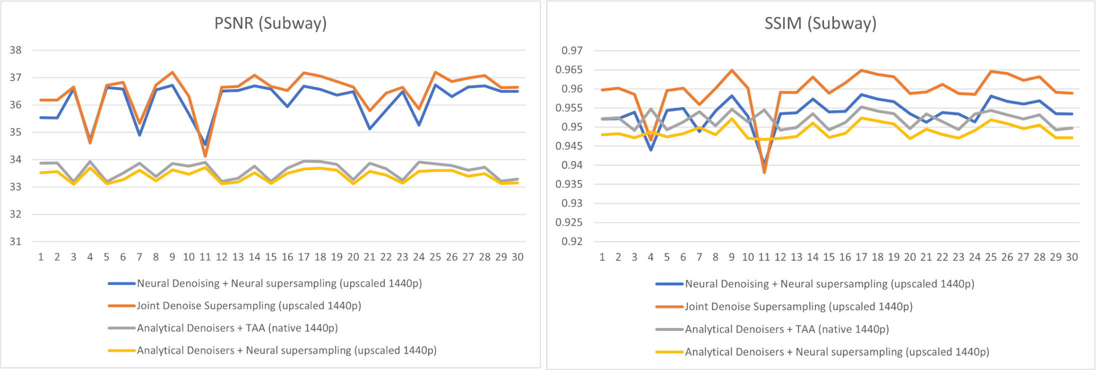

Figure 5.4: Per-frame quality metrics on a continuous frame sequence from the Dock and Scifi dataset. Our network consistently outperforms other tecniques

5.3.2.2 Performance

Similar to the performance evaluation for our denoising network, here we compare how the run-time varies when the number of light sources affecting a pixel is increased in a native 1440p rendering with analytical denoiser and TAA applied and with our joint neural denoising and supersampling network. We see the benefit of using a quantized network that combine different tasks into a single network amortizing the cost as seen in Figure 5.6.

5.4 Conclusion

In the earlier chapters we describe our neural techniques for solving individual reconstruction problems such as supersampling and denoising. In this chapter, we combine the findings into a single network reducing the cost of running two neural reconstruction techniques chained together. We also show that by adding additional compositing and upsample block we can perform super-resolution reducing the rendering cost even further.

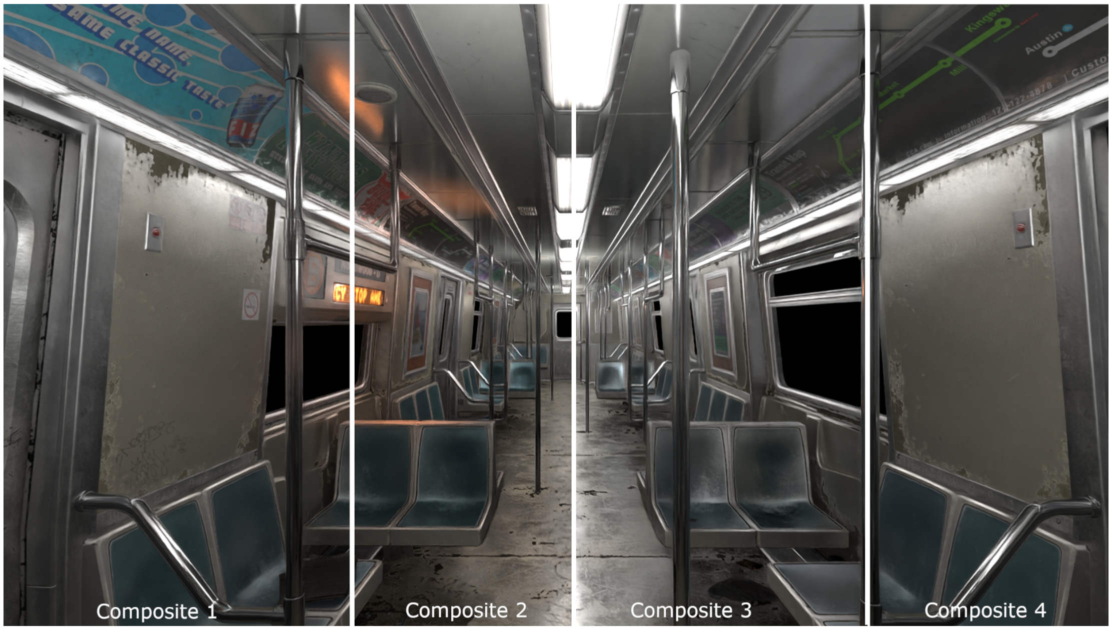

Figure 5.5: Our network generalizes to work with a number of composite weights

In a similar vein to spatial supersampling, it is beneficial to include temporal supersampling in the same network. This can double the frame rate in computationally heavy games. In future, we plan to explore along these direction to boost the rendering while providing an high-quality visual experience.

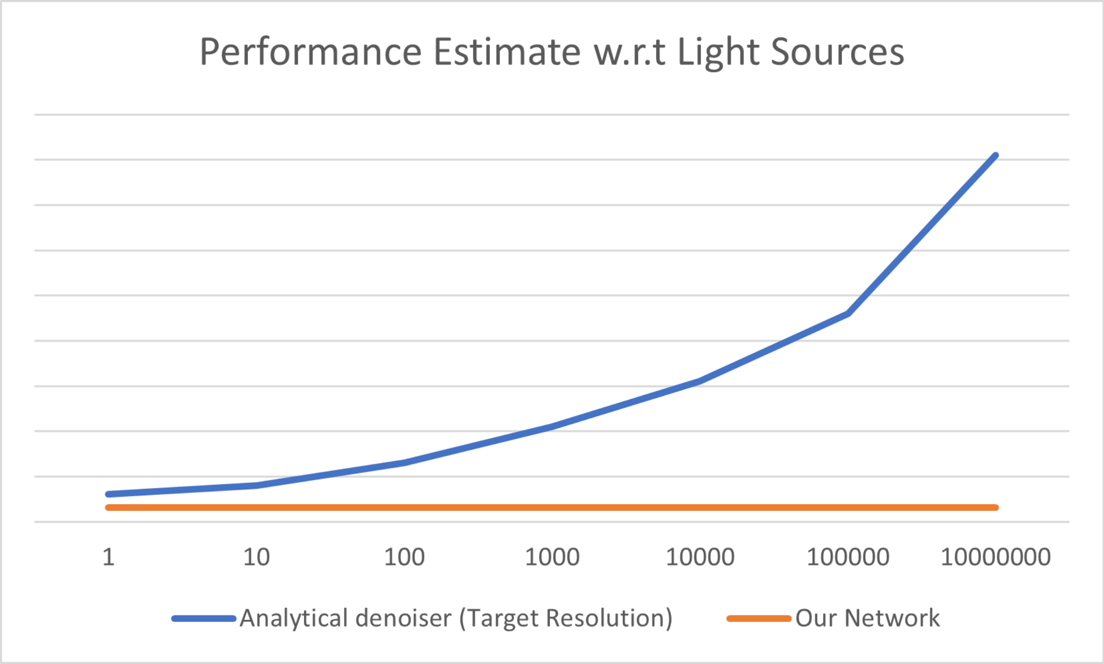

Figure 5.6: Relative performance plot show great benefit in using a single network to jointly perform different reconstruction tasks

Chapter 6

Conclusion

In this thesis, we first studied the challenges in designing a network based reconstruction task for real-time games. To overcome these challenges, we proposed a novel network where most of the operations are set to a lower precision mode. We show that a naive quantization of weights are not enough to produce high-quality HDR images. Instead we leverage quantization-aware training to quantize the weights using a per channel quantizer and a per layer quantizer for the activation. We find that choosing an arbitrary threshold for the activation quantizer limits the range and so we adaptively learn the threshold in the training process. We then demonstrate the capability of this network for a supersampling task. Our network consist of a U-Net based feature extractor and a similar topology filtering path. The hierarchical filtering is useful in achieving large receptive field providing more information about the neighbouring pixel for filtering.

In Chapter 4, we extend the idea of quantized network for a real-time denoising task. Our main contribution is in sharing a single feature extractor for multiple filtering passes. We also describe the design and implementation of a custom perceptual loss network trained on rendered content. We show that the loss network is successful in capturing high frequency details in regions where a standard perceptual loss fail and forms pattern artifact. Our network outperforms the analytical denoisers in term of quality and performance.

Finally, In Chapter 5, we combine the idea of supersampling and denoising into a single network amortizing the cost of separate passes. In addition, we also perform super-resolution to further reduce the overall rendering cost. The input to this network is low resolution one sample per pixel noisy frame and it's corresponding G-buffer and the output is high resolution supersampled denoised frame that is temporally stable. We achieve this by extending our denoising network by adding carefully designed and tested input, output and composite blocks. The feature extraction is same as the denoising network but runs in sub-native resolution to speed up the inference. To recover the high frequency information lost in the pooled feature extractor, we use a per-projection layer and a single convolution layer to extract features in the native resolution. The target resolution convolution and filtering only happens in the composite block. In addition to two denoising paths, there is a per-pixel 3x3 filter for albedo to filter the aliasing.

Our key contribution to the field of neural reconstruction is the demonstration of a novel network architecture for real-time image reconstruction. We believe that this thesis represents a new paradigm for game rendering that does not require a full image to be formed by the renderer, but instead generates partial information about the image, which is then reconstructed by neural networks to produce a plausible image

Bibliography

[1] Checkerboard rendering.

[2] Intel® open image denoise.

[3] Intel® xe super sampling.

[4] Nvidia dlss.

[5] Kurt Akeley. Reality engine graphics. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 109–116, 1993.

[6] John Amanatides. Ray tracing with cones. *ACM SIGGRAPH Computer Graphics*, 18(3):129–135, 1984.

[7] Steve Bako, Thijs Vogels, Brian McWilliams, Mark Meyer, Jan Novák, Alex Harvill, Pradeep Sen, Tony Derose, and Fabrice Rousselle. Kernel-predicting convolutional networks for denoising monte carlo renderings. *ACM Trans. Graph.*, 36(4):97–1, 2017.

[8] Steve Bako, Thijs Vogels, Brian Mcwilliams, Mark Meyer, Jan Novák, Alex Harvill, Pradeep Sen, Tony Derose, and Fabrice Rousselle. Kernel-predicting convolutional net-

works for denoising monte carlo renderings. *ACM Transactions on Graphics*, 36(4), July 2017.

[9] Peter G. J. Barten. Formula for the contrast sensitivity of the human eye. In Yoichi Miyake and D. Rene Rasmussen, editors, *Image Quality and System Performance*, volume 5294, pages 231–238. International Society for Optics and Photonics, SPIE, 2003.

[10] Yoshua Bengio. Estimating or propagating gradients through stochastic neurons. *arXiv preprint arXiv:1305.2982*, 2013.

[11] John Burgess. Rtx on—the nvidia turing gpu. *IEEE Micro*, 40(2):36–44, 2020.

[12] Jose Caballero, Christian Ledig, Andrew Aitken, Alejandro Acosta, Johannes Totz, Zehan Wang, and Wenzhe Shi. Real-time video super-resolution with spatio-temporal networks and motion compensation. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4778–4787, 2017.

[13] Edwin Catmull. A hidden-surface algorithm with anti-aliasing. *ACM SIGGRAPH Computer Graphics*, 12(3):6–11, 1978.

[14] Chakravarty R Alla Chaitanya, Anton S Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila. Interactive reconstruction of monte carlo image sequences using a recurrent denoising autoencoder. *ACM Transactions on Graphics (TOG)*, 36(4):1–12, 2017.

[15] Chakravarty R. Alla Chaitanya, Anton S. Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila. Interactive reconstruction of monte

carlo image sequences using a recurrent denoising autoencoder. *ACM Transactions on Graphics*, 36(4), July 2017.

[16] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*, 2018.

[17] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). *arXiv preprint arXiv:1511.07289*, 2015.

[18] Robert L Cook. Stochastic sampling in computer graphics. *ACM Transactions on Graphics (TOG)*, 5(1):51–72, 1986.

[19] Robert L Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 137–145, 1984.

[20] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015.

[21] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[22] Franklin C Crow. The aliasing problem in computer-generated shaded images. *Communications of the ACM*, 20(11):799–805, 1977.

[23] Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. Learned step size quantization. *arXiv preprint arXiv:1902.08153*, 2019.

[24] Luca Fascione, Johannes Hanika, Marcos Fajardo, Per Christensen, Brent Burley, and Brian Green. Path tracing in production-part 1: Production renderers. In *ACM SIGGRAPH 2017 Courses*, pages 1–39. 2017.

[25] Epic Games. Unreal engine 4.24 on github, 2019.

[26] Nader Gharachorloo, Satish Gupta, Robert F Sproull, and Ivan E Sutherland. A characterization of ten rasterization techniques. In *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, pages 355–368, 1989.

[27] Jonathan Granskog, Fabrice Rousselle, Marios Papas, and Jan Novák. Compositional neural scene representations for shading inference. *ACM Transactions on Graphics (TOG)*, 39(4):135–1, 2020.

[28] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

[29] Eric Haines and Tomas Akenine-Möller. *Ray Tracing Gems: High-Quality and Real-Time Rendering with DXR and Other APIs*. Springer, 2019.

[30] Sebastian Herholz, Oskar Elek, Jiří Vorba, Hendrik Lensch, and Jaroslav Křivánek. Product importance sampling for light transport path guiding. In *Computer Graphics Forum*, volume 35, pages 67–77. Wiley Online Library, 2016.

[31] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[32] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2704–2713, 2018.

[33] Sambhav R Jain, Albert Gural, Michael Wu, and Chris H Dick. Trained quantization thresholds for accurate and efficient fixed-point inference of deep neural networks. *arXiv preprint arXiv:1903.08066*, 2019.

[34] Jorge Jimenez, Jose I Echevarria, Tiago Sousa, and Diego Gutierrez. Smaa: enhanced subpixel morphological antialiasing. In *Computer Graphics Forum*, volume 31, pages 355–364. Wiley Online Library, 2012.

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

[36] Nima Khademi Kalantari, Steve Bako, and Pradeep Sen. A machine learning approach for filtering monte carlo noise. *ACM Trans. Graph.*, 34(4):122–1, 2015.

[37] Brian Karis. High Quality Temporal Supersampling. SIGGRAPH 2014 Advances in Real-Time Rendering in Games course, 2014.

[38] Emmett Kilgariff, Henry Moreton, Nick Stam, and Brandon Bell. Nvidia turing architecture in-depth, 2018.

[39] Byungsoo Kim, Vinicius C Azevedo, Markus Gross, and Barbara Solenthaler. Transport-based neural style transfer for smoke simulations. *arXiv preprint arXiv:1905.07442*, 2019.

[40] Arjan JF Kok and Frederik W Jansen. Adaptive sampling of area light sources in ray tracing including diffuse interreflection. In *Computer Graphics Forum*, volume 11, pages 289–298. Wiley Online Library, 1992.

[41] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*, 2018.

[42] M. Lee and R. Redner. A note on the use of nonlinear filtering in computer graphics. *IEEE Computer Graphics and Applications*, 10:23–29, 1990.

[43] Tzu-Mao Li, Yu-Ting Wu, and Yung-Yu Chuang. Sure-based optimization for adaptive sampling and reconstruction. *ACM Transactions on Graphics (TOG)*, 31(6):1–9, 2012.

[44] Darryl D. Lin, Sachin S. Talathi, and V. Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks. In *Proc. International Conference on Machine Learning (ICML)*, page 2849–2858, 2016.

[45] Michael D. McCool. Anisotropic diffusion for monte carlo noise reduction. *ACM Trans. Graph.*, 18(2):171–194, April 1999.

[46] Soham Uday Mehta, Brandon Wang, and Ravi Ramamoorthi. Axis-aligned filtering for interactive sampled soft shadows. *ACM Transactions on Graphics (TOG)*, 31:1 – 10, 2012.

[47] Soham Uday Mehta, Brandon Wang, Ravi Ramamoorthi, and Fredo Durand. Axis-aligned filtering for interactive physically-based diffuse indirect lighting. *ACM Transactions on Graphics (TOG)*, 32(4):1–12, 2013.

[48] Szymon Migacz. 8-bit inference with TensorRT. In *GPU Technology Conference (GTC)*, 2017.

[49] Ben Mildenhall, Jonathan T Barron, Jiawen Chen, Dillon Sharlet, Ren Ng, and Robert Carroll. Burst denoising with kernel prediction networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2502–2510, 2018.

[50] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, pages 405–421. Springer, 2020.

[51] Don P Mitchell and Arun N Netravali. Reconstruction filters in computer-graphics. *ACM Siggraph Computer Graphics*, 22(4):221–228, 1988.

[52] Thomas Müller, Markus Gross, and Jan Novák. Practical path guiding for efficient light-transport simulation. In *Computer Graphics Forum*, volume 36, pages 91–100. Wiley Online Library, 2017.

[53] Prateeth Nayak, Degan Zhang, and Sek Chai. Bit efficient quantization for deep neural networks. *arXiv preprint arXiv:1910.04877*, 2019.

[54] Thomas Neff, Pascal Stadlbauer, Mathias Parger, Andreas Kurz, Chakravarty R Alla Chaitanya, Anton Kaplanyan, and Markus Steinberger. Donerf: Towards real-time rendering of neural radiance fields using depth oracle networks. *arXiv e-prints*, pages arXiv–2103, 2021.

[55] Diego Nehab, Pedro V Sander, Jason Lawrence, Natalya Tatarchuk, and John R Isidoro. Accelerating real-time shading with reverse reprojection caching.

[56] NVIDIA. Nvidia turing gpu architecture whitepaper, 2018.

[57] Ryutarou Ohbuchi and Masaki Aono. Quasi-monte carlo rendering with adaptive sampling. *IBM Tokyo Research Laboratory*, 1, 1996.

[58] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[59] Anjul Patney and Aaron Lefohn. Detecting aliasing artifacts in image sequences using deep neural networks. In *Proc. High-Performance Graphics (HPG)*, 2018.

[60] Lasse Jon Fuglsang Pedersen. Temporal reprojection anti-aliasing in inside. In *Game Developers Conference*, volume 3, page 10, 2016.

[61] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2016.

[62] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Im-

agenet classification using binary convolutional neural networks. In *European conference on computer vision*, pages 525–542. Springer, 2016.

[63] Alexander Reshetov. Morphological antialiasing. In *Proceedings of the Conference on High Performance Graphics 2009*, pages 109–116, 2009.

[64] Mike Roberts, Jason Ramapuram, Anurag Ranjan, Atulit Kumar, Miguel Angel Bautista, Nathan Paczan, Russ Webb, and Joshua M Susskind. Hypersim: A photorealistic synthetic dataset for holistic indoor scene understanding. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10912–10922, 2021.

[65] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *arXiv preprint arXiv 1505.04597*, 2015.

[66] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[67] Fabrice Rousselle, Claude Knaus, and Matthias Zwicker. Adaptive sampling and reconstruction using greedy error minimization. *ACM Transactions on Graphics (TOG)*, 30(6):1–12, 2011.

[68] Fabrice Rousselle, Claude Knaus, and Matthias Zwicker. Adaptive rendering with non-local means filtering. *ACM Transactions on Graphics (TOG)*, 31(6):1–11, 2012.

[69] Fabrice Rousselle, Marco Manzi, and Matthias Zwicker. Robust denoising using feature

and color information. In *Computer Graphics Forum*, volume 32, pages 121–130. Wiley Online Library, 2013.

[70] Holly E. Rushmeier and Gregory J. Ward. Energy preserving non-linear filters. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '94, page 131–138, New York, NY, USA, 1994. Association for Computing Machinery.

[71] Takafumi Saito and Tokiichiro Takahashi. Nc machining with g-buffer method. *ACM Siggraph Computer Graphics*, 25(4):207–216, 1991.

[72] Mehdi SM Sajjadi, Raviteja Vemulapalli, and Matthew Brown. Frame-recurrent video super-resolution. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6626–6634, 2018.

[73] Daniel Scherzer. Pixel-correct shadow maps with temporal reprojection and shadow test confidence.

[74] Christoph Schied, Anton Kaplanyan, Chris Wyman, Anjul Patney, Chakravarty R Alla Chaitanya, John Burgess, Shiqiu Liu, Carsten Dachsbacher, Aaron Lefohn, and Marco Salvi. Spatiotemporal variance-guided filtering: real-time reconstruction for path-traced global illumination. In *Proceedings of High Performance Graphics*, pages 1–12. 2017.

[75] Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4:131–139, 1992.

[76] Peggy Seriès and Aaron Seitz. Learning what to expect (in visual perception). *Frontiers in human neuroscience*, 7:668, 2013.

[77] SMPTE. St 2084:2014 - smpte standard - high dynamic range electro-optical transfer function of mastering reference displays. *ST 2084:2014*, pages 1–14, 2014.

[78] Charles M Stein. Estimation of the mean of a multivariate normal distribution. *The annals of Statistics*, pages 1135–1151, 1981.

[79] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proc. IEEE Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[80] Eric Veach. *Robust Monte Carlo methods for light transport simulation.* Stanford University, 1998.

[81] Thijs Vogels, Fabrice Rousselle, Brian McWilliams, Gerhard Röthlin, Alex Harvill, David Adler, Mark Meyer, and Jan Novák. Denoising with kernel prediction and asymmetric loss functions. *ACM Transactions on Graphics (TOG)*, 37(4):1–15, 2018.

[82] Thijs Vogels, Fabrice Rousselle, Brian Mcwilliams, Gerhard Röthlin, Alex Harvill, David Adler, Mark Meyer, and Jan Novák. Denoising with kernel prediction and asymmetric loss functions. *ACM Transactions on Graphics*, 37(4), July 2018.

[83] Jiří Vorba, Johannes Hanika, Sebastian Herholz, Thomas Müller, Jaroslav Křivánek, and

Alexander Keller. Path guiding in production. In *ACM SIGGRAPH 2019 Courses*, pages 1–77. 2019.

[84] Gregory K Wallace. The jpeg still picture compression standard. *IEEE transactions on consumer electronics*, 38(1):xviii–xxxiv, 1992.

[85] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Yu Qiao, and Chen Change Loy. Esrgan: Enhanced super-resolution generative adversarial networks. In *Proceedings of the European conference on computer vision (ECCV) workshops*, pages 0–0, 2018.

[86] Gary S Watkins. A real time visible surface algorithm. Technical report, UTAH UNIV SALT LAKE CITY SCHOOL OF COMPUTING, 1970.

[87] Turner Whitted. An improved illumination model for shaded display. In *Proceedings of the 6th annual conference on Computer graphics and interactive techniques*, page 14, 1979.

[88] You Xie, Erik Franz, Mengyu Chu, and Nils Thuerey. tempogan: A temporally coherent, volumetric gan for super-resolution fluid flow. *ACM Transactions on Graphics (TOG)*, 37(4):1–15, 2018.

[89] Ruifeng Xu and Sumanta N. Pattanaik. A novel monte carlo noise reduction operator. *IEEE Comput. Graph. Appl.*, 25(2):31–35, March 2005.

[90] Lei Yang, Shiqiu Liu, and Marco Salvi. A Survey of Temporal Antialiasing Techniques. *Computer Graphics Forum*, 39:607–621, 2020.

[91] Lei Yang, Shiqiu Liu, and Marco Salvi. A survey of temporal antialiasing techniques. In *Computer Graphics Forum*, volume 39, pages 607–621. Wiley Online Library, 2020.

[92] Lei Yang, Diego Nehab, Pedro V Sander, Pitchaya Sitthi-Amorn, Jason Lawrence, and Hugues Hoppe. Amortized supersampling. *ACM Transactions on Graphics (TOG)*, 28(5):1–12, 2009.

[93] Randy Yates. Fixed-point arithmetic: An introduction. *Digital Signal Labs*, 81(83):198, 2009.

[94] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *Proc. European Conference on Computer Vision (ECCV)*, pages 365–382, 2018.

[95] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. In *International Conference on Learning Representations (ICLR)*, 2017.

[96] Shuchang Zhou, Zekun Ni, Xinyu Zhou, He Wen, Yuxin Wu, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

[97] Chenzhuo Zhu, Song Han, Huizi Mao, and William J. Dally. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*, 2016.

[98] Matthias Zwicker, Wojciech Jarosz, Jaakko Lehtinen, Bochang Moon, Ravi Ramamoorthi, Fabrice Rousselle, Pradeep Sen, Cyril Soler, and S-E Yoon. Recent advances in adaptive

sampling and reconstruction for monte carlo rendering. In *Computer graphics forum*, volume 34, pages 667–681. Wiley Online Library, 2015.

Appendix A

Derivation for Quantization Function gradiants

In this section, we derive the gradients for our simulated quantization function. Expanding Equation 3.7 we get the simulated quantization function as:

$$x'_q = \begin{cases} s\left\lfloor \frac{x}{s} \right\rceil & \text{if } l_a \leq x \leq u_a \\ l_a & \text{if } x < l_a \\ u_a & \text{if } x > u_a, \end{cases} \tag{A.1}$$

where s is the quantization step, l_a and u_a are the adjusted range as defined in Equation 3.4 and 3.5. The local gradient with respect to trainable lower bound t_l is given by:

$$\frac{\partial x'_q}{\partial t_l} = \begin{cases} \frac{l}{M}\left(\frac{x}{s} - \left\lfloor \frac{x}{s} \right\rceil\right) & \text{if } l_a \leq x \leq u_a \\ \frac{l}{M}\left(\frac{l}{s} - \left\lfloor \frac{l}{s} \right\rceil + M\right) & \text{if } x < l_a \\ \frac{l}{M}\left(\frac{l}{s} - \left\lfloor \frac{l}{s} \right\rceil\right) & \text{if } x > u_a \end{cases} \tag{A.2}$$

Similarly, the local gradient with respect to trainable upper bound t_u is:

$$\frac{\partial x'_q}{\partial t_u} = \begin{cases} \frac{u}{M}\left(\lfloor \frac{x}{s} \rceil - \frac{x}{s}\right) & \text{if } l_a \leq x \leq u_a \\ \frac{u}{M}\left(\lfloor \frac{l}{s} \rceil - \frac{l}{s}\right) & \text{if } x < l_a \\ \frac{u}{M}\left(\lfloor \frac{l}{s} \rceil - \frac{l}{s} + M\right) & \text{if } x > u_a \end{cases} \tag{A.3}$$

Appendix B

Approximating Dynamic Global Illumination with GANs

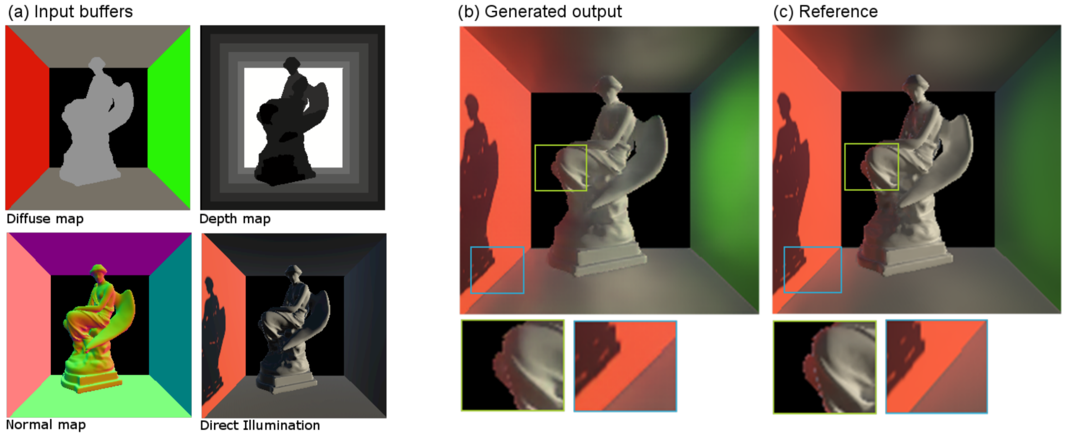

Figure B.1: The Deep Illumination technique produces global illumination output by training generative adversarial networks. Once the network is trained, inputting G-buffers along with the direct illumination buffer will add high-quality indirect illumination in real-time, including for novel configurations of a scene, such as new positions and orientations of lights and cameras, and entirely new objects.

Computing global illumination effects to reproduce real-world visual phenomena, such as indirect illumination, soft shadows, reflections , crepuscular rays, and caustics, remains a challenge for real-time applications, even with the support of modern graphics hardware. Accurate physically-based global illumination computation is costly due to the difficulties in computing visibility between arbitrary points in 3D space and computing illumination at a point by taking the integration of light information from different directions. In real-time applications, we are bounded by frame rates to maintain interactivity. Perceptually plausible illumination at real-time frame rates are often achieved by using approximation. Global illumination (GI) approximation techniques can be classified as static or dynamic. In static approximation, light information is precomputed and stored as texture data, called lightmaps. The advantage of using lightmaps is that they store the outputs of computationally expensive GI algorithms for determining light information, which are first computed offline and then readily available via fast texture lookups. The downside is that lightmaps are effective only for the static objects in a scene, or which the time-consuming computations only need to be done once. On the other hand, dynamic approximations support changes in lighting, soft shadows, and a range of other lighting effects. Screen space algorithms provide dynamic approximation, operating on G-buffers in image space rather than on 3D objects in model space. The G-buffers contain data about visible surfaces such as color, normal, depth, and position. The buffers are created during the first pass and then the actual shading is deferred to a second pass that uses only the buffer information. Screen space algorithms are well suited for GPUs, which are designed for fast texture fetching. Although such algorithms can provide fast and dynamic GI solutions, image quality, level of detail, and accuracy may be compromised.

Recent developments in deep learning and neural networks have been used to solve image-to-image translation problems using a generative model called Generative Adversarial Networks (GANs) and its conditional version. Both GANs and cGANs have shown promise in density estimation in an implicit fashion. So far, to the best of our knowledge, generative models have been used primarily in the domain of image processing, for example, to generate a plausible improvement in the resolution of an image, or to transform black-and-white images to color images, or to automatically remove noise from image. They have also been used to transfer features learned from a set of images (the "style") onto another image. In a somewhat analogous manner, with this work we introduce the use of GANs to solve problems related to graphics rendering. We can view effective GI approximation using image space buffers as a translation problem in which the network is expected to generate the GI output and where the image space buffers act as the input conditions.

We present a novel technique, Deep Illumination,that computes indirect illuminations and soft shadows using a deep learning approach. The main idea is to train a network to learn high quality GI using an effective offline technique or online approximation technique. That is, similar to approaches for static approximation, we first precompute the lighting and make it available for retrieval when rendering frames in real time. However, we model this information in a neural network, where the network has learned a function to convert from one probability distribution to another. The key difference between our approach and a static approach, like generating lightmaps, is that we sample light information from a range of different possible configurations in terms of light direction, the position and orientation of the camera, and the position of objects in the scene we are interested in. We then extract the G-buffers and the direct illumination buffer for the same configuration and provide it as an input to a deep neural network in order to learn a mapping from these buffers to the computed GI. After training on a sufficient number of dynamic configurations, we show that our network has the ability to effectively approximate indirect illumination for object-light-camera configurations that it has not encountered during training. That is, it produces realistic GI even when the camera, lights, and objects in the scene are placed in new locations and/or have different orientations. It also provides strong temporal coherence in dynamic scenes so that the applied GI does not flicker from frame to frame. Not only does it produce effective GI for objects used during training in arbitrary positions, but good results are generated even for new objects that were not provided during training. Additionally, it learns to soften the hard shadows produced by direct illumination.

We introduce a “one network for one scene” approach for training the network to learn GI features for a particular 3D scene. This approach is highly relevant for scenes that are dynamic, but which nonetheless have an clearly recognizable style with features that can be learnt by a neural network. For example, video games generally distinguish themselves with a distinct artistic style, and a different Deep Illumination network could be created for each game. In video games where each level is artistically different from each other in terms of shading style and light setup, a Deep Illumination network could be created for each level.

There are two phases to our approach: 1) the training phase, and 2) the runtime phase. For the training phase, we extract G-buffers (depth map, normal map and diffuse map), the direct illumination buffer, and the output from an any expensive GI technique, which we use as our “ground truth,” from a 3D scene for different possible light-camera-object configurations in terms of position and direction. (We have used trained on the output of path tracing and

Voxel-based Global Illumination, but our technique will work on any GI technique.) The GAN network is then trained in a semi-supervised fashion where the input to the network is the G-buffers and direct illumination buffer. The ground truth, that is, the output from the chosen GI technique, is used by the network to learn the distribution with the help of a loss function. Once training is complete, we can then use the trained network for the runtime phase. During the runtime phase, the network can generate indirect illuminations and soft shadows for new scene configurations at interactive rates, with a similar quality to the ground truth GI technique used for training, and using only image space buffer information.

We prepared the dataset for two different scenarios (that is, we create a different GANs for each of the two scenes). In the first scenario, the scene has a fixed camera, and consisted of a Cornell box with a directional light rotating along an axis. A 3D model is placed in the center of the Cornell box, which rotates along multiple axes. For training and validation, we used only 3D models of a sphere, cylinder, cube, the Stanford bunny, and the Stanford dragon. We extracted 8000 pairs of G-buffers (depth, normal, and diffuse maps), the direct illumination buffer, and the ground truth from Voxel-based GI solution and GPU path tracing. All image pairs in the dataset have a resolution of 256x256 pixels. For testing, we extract 2000 image pairs with a statue model and the Happy Buddha model, both of which were not used in training.

The second scenario presents a custom low-poly world, traversed with a moving camera and containing moving objects that are emissive in nature, along with particle effects. The light in the scene simulates the sun, producing a day-night cycle. We extracted 12,000 pairs of image space buffers. For training validation and testing, we used the same objects (unlike in the first scenario), but here the camera moves in a different path not encountered during training. The validation set is thus a mix of both unseen configurations of the scene and intermediate scenes from the training set. (Additionally, another portion of this scenario is used for evaluation purposes.

We use generative adversarial networks, which consists of a generator and a discriminator. The generator network takes G-buffers and the direct illumination buffer as the input and attempts to map it to its corresponding GI output. The discriminator network takes either the ground truth GI output or the generated GI output from the generator, and classifies them as real or fake, where "real" means that the input is from the distribution of images produced by the expensive GI technique. Thus the generator and discriminator network play a min-max adversarial game where the generator tries its best to fool the discriminator into thinking that the generated output is from the real distribution and the discriminator tried to learn from the real and generated image to classify them. This game will continue until discriminator is unable to distinguish between real and generated images.

Our generator is a U-Net, which is a deep convolutional network consisting of an encoder and decoder with skip connections. The encoder consists of a convolution followed by an activation function, compressing the data into its high-level representation. The decoder does the opposite of the encoder, consisting of a deconvolution and an activation function. Both the encoder and the decoder undergo batch normalization, except for the first and last layers of the network. LeakyReLU is used as the activation function of the encoder part, and ReLU and tanh (for the last layer only) are used in the decoder. Skip connections combine the output of an encoder to the input of the decoder counterpart. These skip connections are useful in recovering the spatial information lost during compression. The input to the generator network is the G-buffers and direct illumination buffer concatenated into a 12 channel image, and the output is a 3-channel RGB image. We use 8 encoders and 8 decoders with a varying number of base layers (K=32, 64, 128) for evaluations.

Our discriminator is a patchGAN which is also made up of deep convolutional layers, but patchGAN has the advantage that it gives a patch-wise probability of being real or fake, which is then averaged to get the output. The discriminator consists of 5 encoders (no decoders) with LeakyReLU and sigmoid (only for the last layer) as the activation function. Similar to the generator network, the discriminator encoder undergoes batch normalization throughout, except for the first and last layers. The discriminator takes two sets of inputs, conditional image space buffers combined with the target image, or instead with the generated image. The discriminator then decides whether the given set is generated or not. The input to the discriminator is image space buffers and an image (real or generated) making it a 15 channel input, and output is a probability of how much of the input is real. We used a base layer k=64 for the discriminator.

In addition to the original GAN loss function, we also use an L1 loss, which is a pixel-wise absolute value loss on the generated output. Unlike the traditional GANs, where the goal is to produce novel content as output, we care about the translation from image buffers to a final image rather than novel content generation. If we use GAN loss alone, we will end up with an image from the distribution we are looking for, but which lacks accuracy in terms of structure and light information. This is because the discriminator is only designed to check whether the generated image is from the target probability distribution. If we use L1 loss alone, i.e., only a generator, then the network tends to produce blurry images, since it does well at low frequencies but fails at high frequencies that are needed to provide crispness, edges, and detail. The combination of these two loss function ensures that the network will produce crisp images that are from the desired distribution and that are accurate when compared with ground truth.

In this work, we presented Deep Illumination, the first application of generative adversarial networks to the problem of global illumination, producing temporally coherent indirect illumination and soft shadows from image space buffers. There are several interesting areas to explore as future work for this approach. For example, we showed how a generative adversarial network can be used learn a mapping from G-buffer and direct illumination buffer to any global illumination solution for scenes with diffuse materials. For future exploration, we would like to study how the network behaves when introducing different materials, and we also believe that our technique will be effective at learning other global illumination effects, such as motion blur, depth of field, and caustics.

Although our experiment shows promising results when introducing an unknown object into the scene, it will also be interesting to investigate more complex scenarios containing a wider variation of models, cameras, and lights in different positions and orientation. One interesting direction is to explore global illumination in procedurally generated environments, where the network is trained on random scenes that contain the building blocks of the environment. In terms of network structure, we plan to perform more experiments on scenes of different sizes and scales, and with different numbers of objects and lights in order to find an optimal number for the base layer, which directly impacts the runtime.

Finally, an interesting avenue of exploration is to automate the data acquisition process for training. That is, given a particular scenario, we should be able to automatically configure the camera within possible scenes in order to extract frames for training that contain significant light-object variations. Future work will investigate optimizing the training of the network to both speed up the training process and to reduce artifacts when generating global illumination effects.

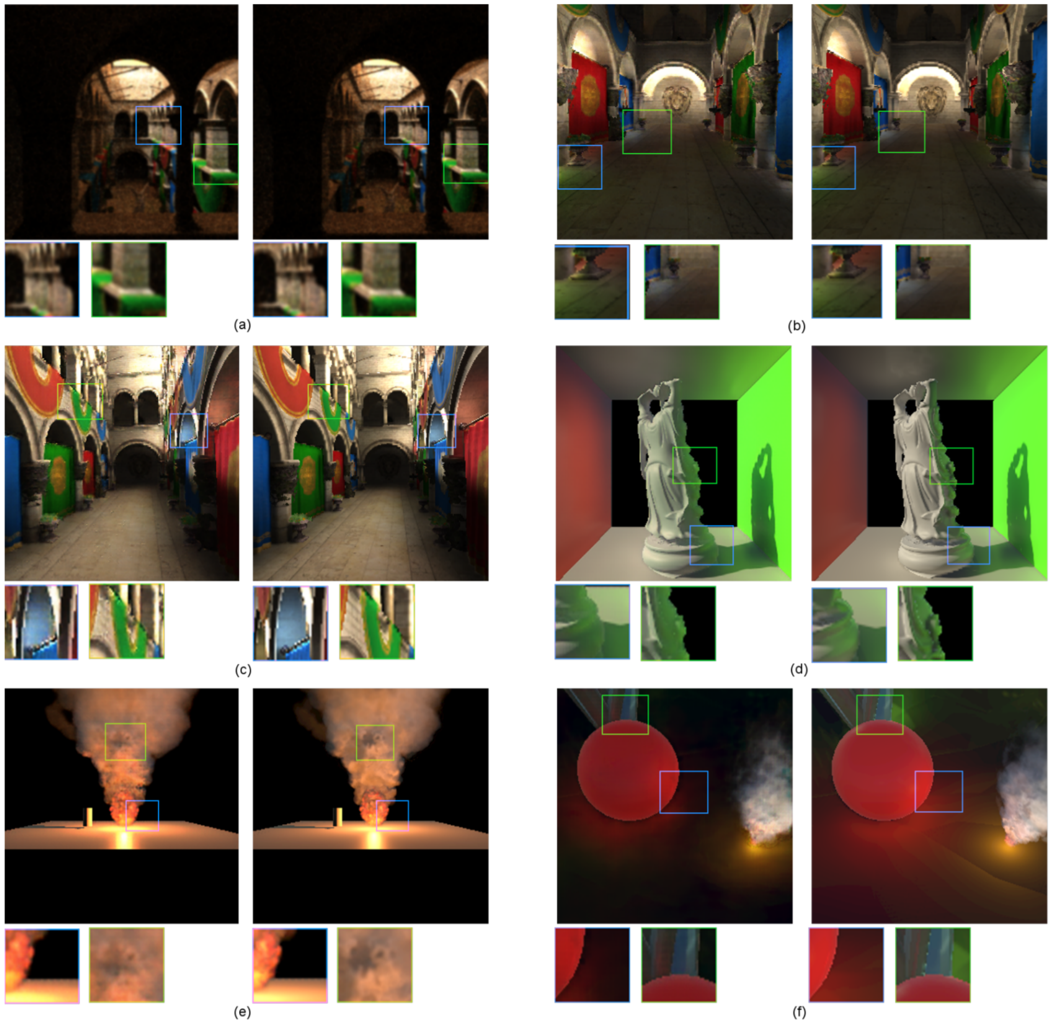

Figure B.2: This figure shows the application of Deep Illumination to a wide number of real world scenarios, in which the subtleties of GI are captured nearly perfectly. In all examples, the left side is the Deep Illumination output, and the right side is the ground truth. We show close-ups to highlight comparisons showing interesting features in each scene. Pathtracing is used for example (a) and Voxel Cone Tracing is used for the rest of the examples. In (b), for example, we see that Deep Illumination can successfully generate the reflections of colored lights on the floor (blue square and green square). The example in (d) shows that our system can generate realistic global illumination ever for new objects that it was not trained on.

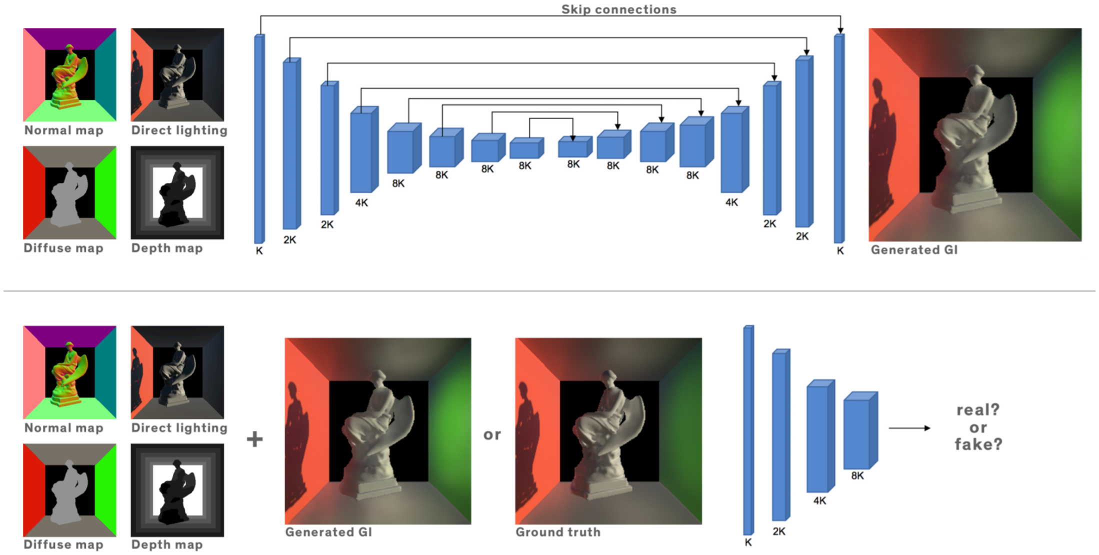

Figure B.3: Figure shows a schematic of the generator network (top) and discriminator network (bottom) used in Deep Illumination to learn a mapping from image space buffers to the target global illumination solution.

Appendix C

Learning Patterns in Sample Distributions for Monte Carlo Variance Reduction

Monte Carlo (MC) integration is the de-facto standard for physically based image synthesis, stemming mainly from its broad applicability and solid mathematical foundations. On the downside, the stochastic nature of MC methods causes the resulting solution to contain a certain amount of error that fully vanishes only in the theoretical limit case of infinitely many evaluated samples. This error is the residual variance of the used MC estimator, often referred to as “Monte Carlo noise”. This nomenclature is somewhat unfortunate, since “noise”—in the signal processing context— typically refers to an undesired distortion of a signal caused by instruments or otherwise. In contrast, MC variance is an inherent phenomenon, inseparable from, and specific to any MC estimator.

Every image sample produced by an MC estimator carries relevant information about the underlying distribution from which it is drawn. These distributions—specific to every point in the image—will be denoted as characteristic distributions. Our proposition is to identify the characteristic distributions in the context of MC image synthesis, and use that knowledge to produce better estimates for every image pixel individually, rather than simply averaging the samples generated for that pixel. In that sense, our method belongs to the a-posteriori category. This is in contrast to apriori methods which attempt to derive a variance reduction strategy from theoretical principles; although we do ground our reasoning in theory as well, whenever possible.

We start with this simple interior scene illuminated by a single closed window; most of the illumination is therefore indirect. There is a good variation of diffuse and glossy materials. The transport in this scene is moderately difficult for a path tracer—after 2k SPP we get a reasonably converged image with only a few fireflies.

Diffuse objects - CHD tend to be quite symmetric in the log domain, close to Gaussian in fact. In the linear (radiance) domain, CHD exhibit long tails, strong skew, with mode typically lower than mean—resembling the Gamma-type distribution quite well.

Glossy objects - Shapes surprisingly similar to diffuse objects, although in this case the different RGB channels' CHD are shifted w. r. t. each other (as a consequence of the underlying brown-wood texture). Interestingly, their spread differs across color channels, suggesting that their width is a function of the incoming radiance itself

Noisy reflections - All of the pixels in the reflection, in spite of their high estimate variance, share virtually the same CHD shape. This is in full agreement with the assumptions of previous work regarding viable pixel similarity measures.

Direct illumination - Very well resolved, low-noise CHD—in spite of their very significant spread and long tail.

Indirect illumination - Noisy CHD with very long tails (high kurtosis). High number of null samples (around 60 %) and significant proportion of dark samples, showing inverted (negative) skew in the log domain (similar to the noisy reclection case).

Our prototype model is a multi-layer perceptron network with a width-32 input layer (for the radiance samples L), three 32-neuron densely connected layers, and a single output unit representing the estimate. In accordance with the findings of our study we trained a separate network for each tested scene.

The training process works with the sample data obtained from rendering the ground truth images using 2k samples per pixel. For each training run, we uniformly selected 6400 pixels and divided their radiance samples into batches of 32 samples each; then separated out 10 iterations with the learning rate of 0.005, the training took 20 minutes in Tensorflow on NVIDIA RTX 2080 Ti GPU. The runtime of this model on an 800×800 RGB image with 3×32 SPP is under 1 minute in the unoptimized development mode.

The ability of our model to improve upon the baseline estimates is visually demonstrated in Figure for the BATHROOM and MEASUREONE scenes. These results are promising, as our model yields smoother images even without exchaning any information between neighboring pixels. In particular, a majority of the fireflies in the baseline is suppressed, a behavior similar to the results obtained by previous works. This has the benefit of regularizing the image to yield a more balanced variance distribution, which means that lateral denoising approaches would have an easier time using our model's output as the starting point.

To evaluate our model numerically, we first examined the quality of the results during training; this is shown in Figure for the two result scenes. We see a reasonably stable behavior for both scenes, although surprisingly the model does not improve markedly after the initial iteration batch. This indicates a space for future optimization of the training time, so that potentially the model could learn on the fly—that is, during the rendering process itself, to adapt to the current scene and rendering configuration.

The variance reduction model we present shows promising results, in spite of the fact that it has no information about the surrounding pixels. Our plan is to continue our investigation and develop a truly robust model that is scene-sensitive, adapts to the particular rendering configuration, and runs progressively in lockstep with the rendering process.

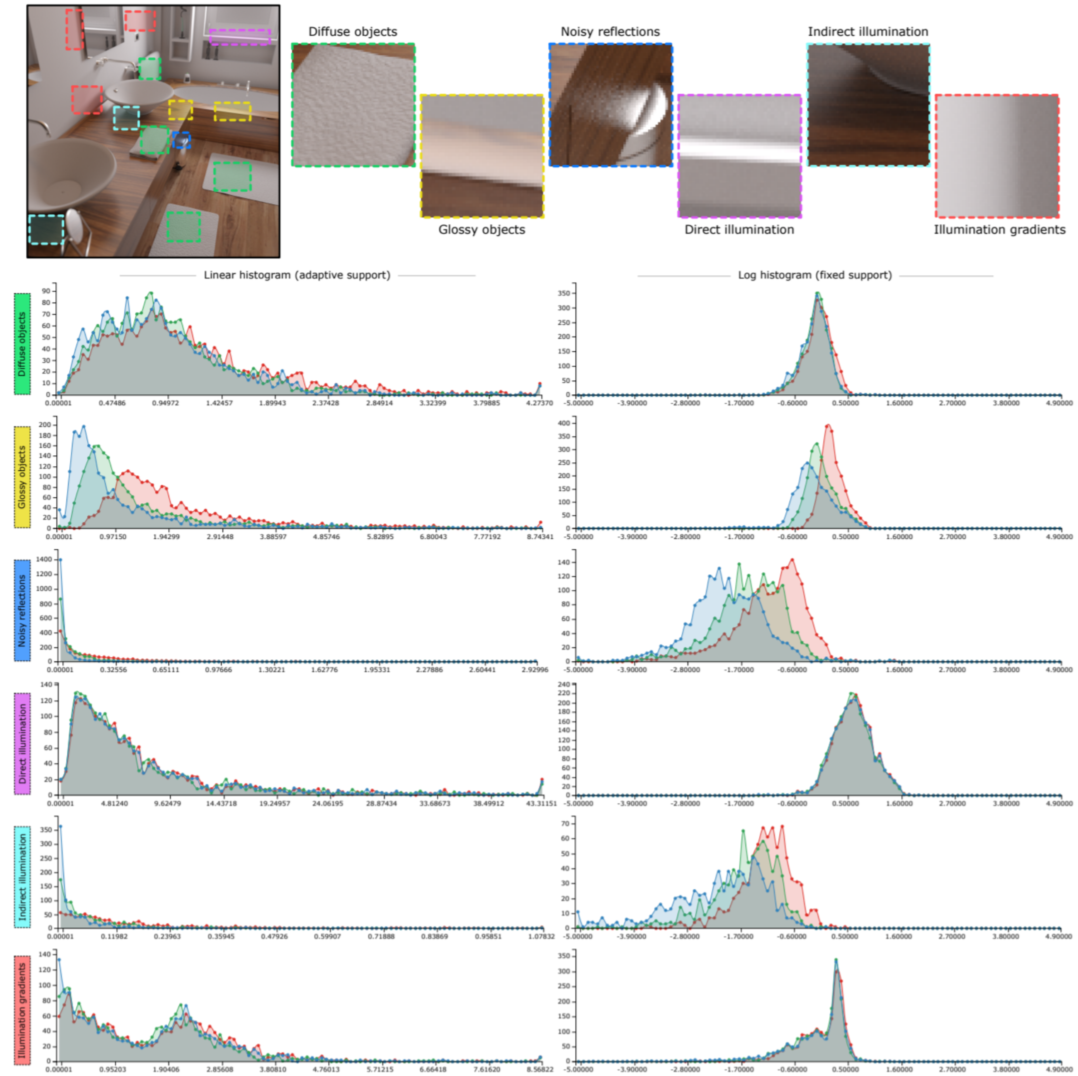

Figure C.1: The regions of interest are selected using different colors, which then mark the corresponding magnified insets as well as the characteristic sample distributions below

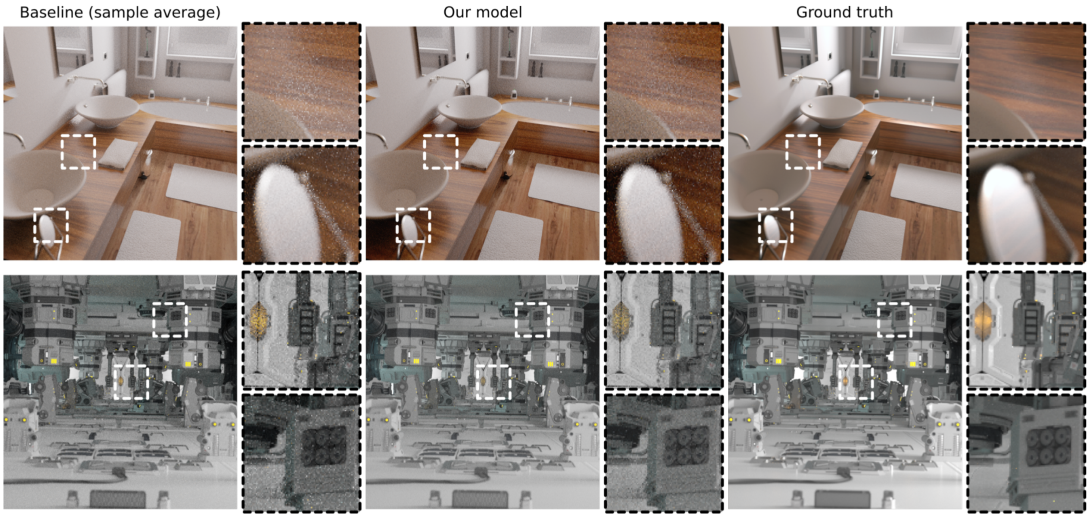

Figure C.2: Variance reduction results for the BATHROOM and MEASUREONE scenes. We compare the baseline estimate from 32 SPP and the result of our model for the same sample set, against the ground-truth images rendered with 2k SPP.

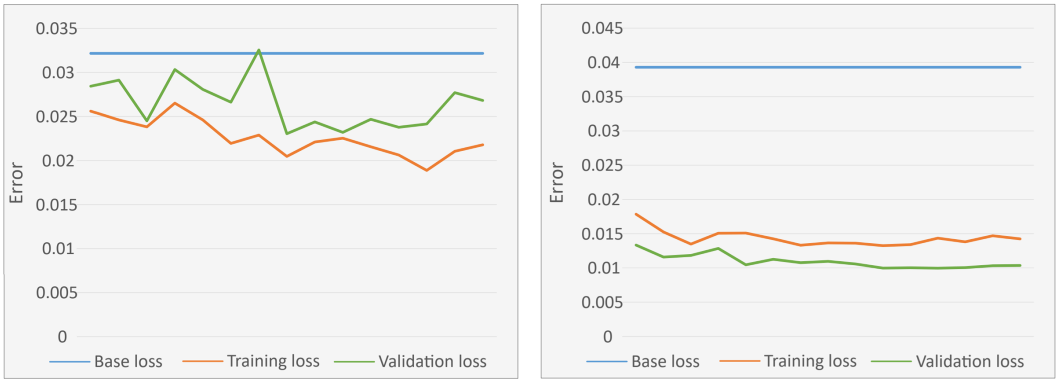

Figure C.3: The loss function values obtained during the 30k training iterations for the BATHROOM scene (left) and the MEASUREONE scene (right). The loss for the training and validation data is plotted separately, against the baseline error

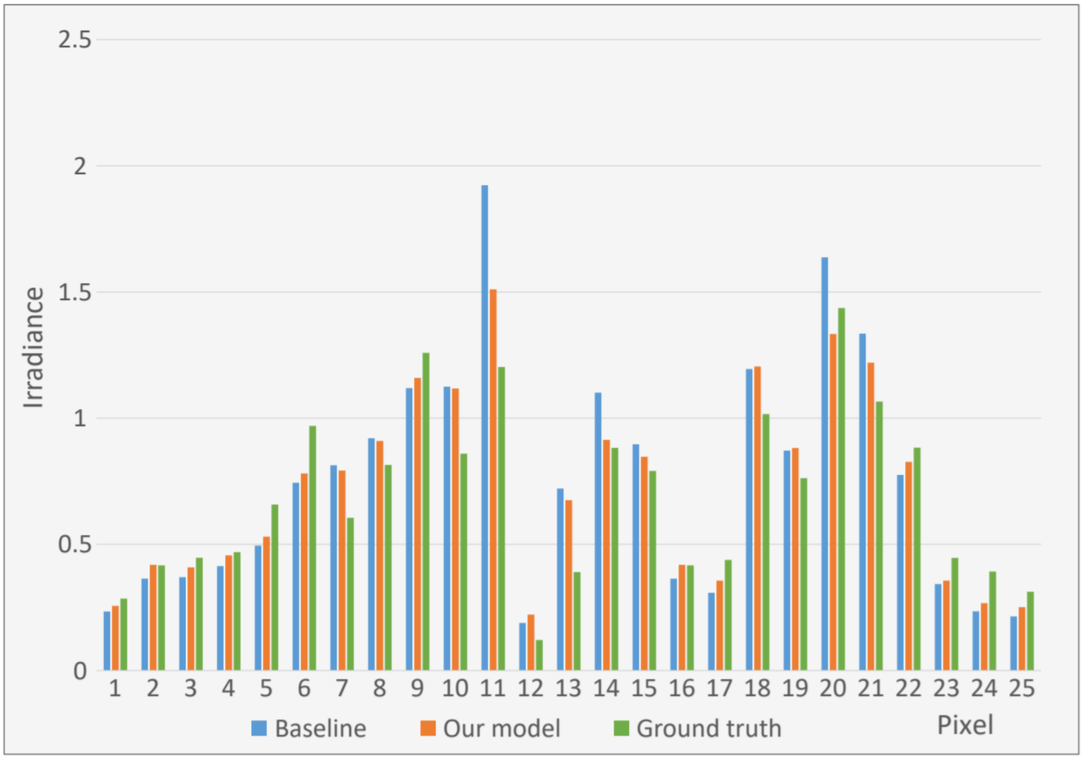

Figure C.4: Numerical evaluation for the BATHROOM scene (top) and the MEASUREONE scene (bottom), shown for 25 randomly selected pixels from the respective images.