석 사 학 위 논 문
Master's Thesis

# 렌더링 노이즈 제거를 위한 픽셀 단위 가이던스 기반 보조 피처 활용법

Enhancing Monte Carlo Denoising with Pixel-level Guidance of Exploiting Auxiliary Features

2023

한 규 범 (韓 奎 帆 Han, Kyu Beom)

한 국 과 학 기 술 원

Korea Advanced Institute of Science and Technology

석 사 학 위 논 문

# 렌더링 노이즈 제거를 위한 픽셀 단위 가이던스 기반 보조 피처 활용법

2023

한 규 범

한 국 과 학 기 술 원

전산학부

# 렌더링 노이즈 제거를 위한 픽셀 단위 가이던스 기반 보조 피처 활용법

한 규 범

위 논문은 한국과학기술원 석사학위논문으로
학위논문 심사위원회의 심사를 통과하였음

2023년 6월 7일

심사위원장 윤 성 의 (인)

심 사 위 원 김 민 혁 (인)

심 사 위 원 박 진 아 (인)

# Enhancing Monte Carlo Denoising with Pixel-level Guidance of Exploiting Auxiliary Features

Kyu Beom Han

Advisor: Sung-Eui Yoon

A dissertation submitted to the faculty of
Korea Advanced Institute of Science and Technology in
partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

Daejeon, Korea
June 7, 2023

Approved by

---

Sung-Eui Yoon
Professor of Computer Science

The study was conducted in accordance with Code of Research Ethics[1].

[1] Declaration of Ethical Conduct in Research: I, as a graduate student of Korea Advanced Institute of Science and Technology, hereby declare that I have not committed any act that may damage the credibility of my research. This includes, but is not limited to, falsification, thesis written by someone else, distortion of research findings, and plagiarism. I confirm that my thesis contains honest conclusions based on my own careful research under the guidance of my advisor.

MCS

한규범. 렌더링 노이즈 제거를 위한 픽셀 단위 가이던스 기반 보조 피처 활용법. 전산학부 . 2023년. 27+iv 쪽. 지도교수: 윤성의. (영문 논문)
Kyu Beom Han. Enhancing Monte Carlo Denoising with Pixel-level Guidance of Exploiting Auxiliary Features. School of Computing . 2023. 27+iv pages. Advisor: Sung-Eui Yoon. (Text in English)

**초 록**

기하 정보나 광선 경로 정보와 같은 렌더링 보조 피처들은 몬테카를로 노이즈 제거에 상당한 성능 향상을 가져다주었다. 그러나 최근 기법들은 노이즈 제거 딥러닝 모델의 학습 과정에서 보조 피처 활용에 대한 정보를 주지 않아 보조 피처들을 효과적으로 활용하기 어렵다. 이러한 문제를 해결하기 위해 본 연구는 픽셀 단위로 보조 피처 활용에 대한 가이던스를 제공하는 노이즈 제거 기법을 소개한다. 먼저 기하 정보와 광선 경로 정보를 서로 다른 노이즈 제거 모델 학습에 각각 활용한다. 그런 다음 앙상블 네트워크를 통해 각 픽셀당 앙상블 가중치 맵을 얻는다. 이 맵은 각 픽셀을 복원하는 대에 어떤 보조 피처를 우세하게 활용해야 하는지를 나타내는 가이던스로서 역할을 한다. 해당 가중치 맵을 통해서 두 개의 노이즈 제거 모델의 결과를 앙상블하여 최종 노이즈 제거 결과를 얻는다. 이에 더해 픽셀 단위 가이던스를 노이즈 제거 모델에 전파하기 위해 노이즈 제거 모델과 앙상블 모델을 동시에 학습하여, 기하 정보 혹은 광선 경로 정보가 상대적으로 중요한 영역에 노이즈 제거 모델이 집중하도록 한다. 기하 정보와 광선 경로 정보를 동시에 활용하는 기존 노이즈 제거 모델들에 제안하는 프레임워크를 적용하여 수치와 시각적 성능이 개선됨을 확인하였다.

**핵 심 낱 말** 몬테카를로 광선추적법, 심층 신경망, 앙상블 학습, 이미지 노이즈 제거

**Abstract**

The utilization of auxiliary features such as geometric buffers (*G-buffers*) and path descriptors (*P-buffers*) has greatly enhanced the denoising process in Monte Carlo (MC) techniques. However, recent methods let the neural network to implicitly learn how to exploit these auxiliary features, which may result in suboptimal utilization of each type. To address this issue, we propose a denoising framework that incorporates explicit pixel-wise guidance for leveraging auxiliary features. Our approach involves training two separate denoisers, each trained with a specific auxiliary feature (*G-buffers* or *P-buffers*). We then employ an ensembling network to generate per-pixel ensembling weight maps, which serve as guidance for determining the dominant auxiliary feature for reconstructing each individual pixel. These weight maps are used to combine the outputs of the two denoisers. Additionally, we propagate the pixel-wise guidance to the denoisers by jointly training them with the ensembling network, encouraging the denoisers to focus on regions where G-buffers or P-buffers are more relevant for denoising. Our experimental results demonstrate significant improvement in denoising performance compared to a baseline model that utilizes both *G-buffers* and *P-buffers*.

**Keywords** Monte Carlo ray tracing, deep neural network, ensemble learning, image denoising

# Contents

# List of Tables

# List of Figures

# Chapter 1. Introduction

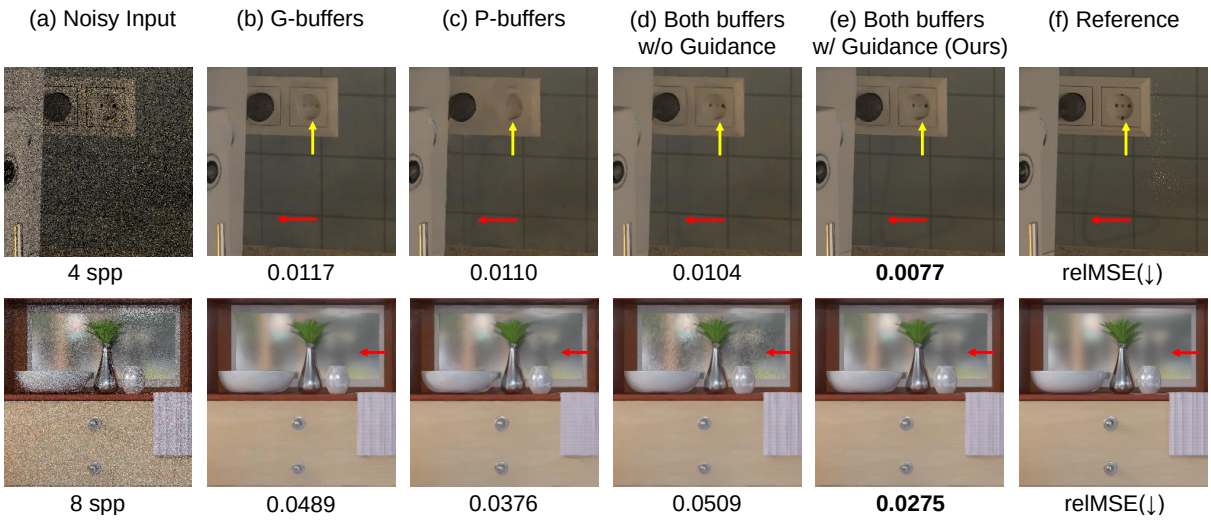


Figure 1.1: We propose a novel guidance framework that generates pixel-wise guidance for exploiting auxiliary features in MC denoising. Our pixel-wise guidance effectively utilizes two types of auxiliary features, geometric features (*G-buffers*) and path features (*P-buffers*). Our framework (e) effectively removes noise on geometric details (yellow arrows) and complex lighting effects (red arrows) compared to previous works using both auxiliary features (d) via simple concatenation and using a single type of auxiliary features (b, c).

Monte Carlo (MC) rendering [15] is widely used in production rendering for generating photorealistic images. The power of MC rendering comes from its unbiased nature. The unbiasedness allows MC rendering to provide good approximate for global illumination. On the other hand, the stochasticity of MC rendering makes the approximation converge slowly, demanding MC rendering use thousands of samples per pixel to achieve a clean image. Thorough research has been done to accelerate MC rendering by using a few samples per pixel for generating clean, photorealistic images.

Among various strategies, MC denoising techniques [20] are widely studied for their effectiveness and simplicity. The goal of these techniques is to accelerate the rendering process by removing the noise from the noisy image generated from MC rendering using few samples per pixel (*i.e.,* 2-64 spp) rather than shooting thousands of samples. These post-processing approaches are easy to deploy on renderers, requiring minimal implementation to acquire auxiliary features, which serve as valuable cues for denoising.

Geometric features (*i.e., G-buffers*) are common choice as auxiliary features for MC denoising. They usually consist of albedo (texture), normal, and depth of the first bounce of the light sample, which are aggregated into image space [26]. *G-buffers* provide strong geometric constraints for denoising, showing robust performance in reconstructing geometric details and diffuse reflections. However, they often fail to capture information for complex lighting effects such as multi-bounce reflections and caustics.

Recent works propose path features (*i.e., P-buffers*) to deal with noise from complex lighting effects [10][19][6]. *P-buffers* contain optical and material information on each bounce of light samples,

such as sampling probabilities and light directions. Furthermore, Cho et al. [6] introduce a manifold-learning technique that can embed high-dimensional *P-buffers* in low-dimensional space based on the similarity in radiances. The proposed manifold-learning increases the utilization of *P-buffers* on various learning-based denoisers [2][10][21].

However, previous works using *G-buffers* and *P-buffers* [10][6] simply combine both auxiliary features as an input to the denoiser. A simple concatenation of auxiliary features makes it challenging for the denoising neural network to capture the different characteristics of these auxiliary features. Consequently, the denoiser struggles to effectively utilize various types of auxiliary features to remove noises from various light transport phenomena, as in the cases of Figure. 1.1.

To address this issue, we propose a novel pixel-wise guidance framework for MC denoising. Our framework provides guidance on which auxiliary features (*e.g., G-buffers* or *P-buffers*) should be more effectively exploited in a pixel-wise manner. We first consider two denoising networks, each using only a single type of auxiliary features. We design a deep learning-based ensembling network that estimates pixel-wise ensembling weight map, which is used to ensemble the two denoised results from the denoisers. Additionally, we train the ensembling network jointly with the denoisers to provide pixel-wise guidance to the denoisers. Our joint-training guides the denoisers to focus on denoising regions where the weight maps assign more emphasis.

We build our framework based on two existing denoisers, KPCN [2] and AdvMCD [30], and show that our work can benefit various learning-based denoisers. We also compare our work with the existing optimization-based ensembling approach for MC denoising [34]. Finally, we present an analysis of our ensembling framework with extensive studies.

# Chapter 2. Related Works

## 2.1 MC Denoising and Auxiliary Features

The goal of MC denoising is to accelerate the rendering process by removing the noise from the rendered result when using few samples per pixel (*e.g.,* 2-64 spp). Such post-processing is faster than using a sufficient number of samples to get the clean image. Traditional MC denoising methods estimate robust denoising kernels to remove the noise. Some works deploy existing denoising kernels such as non-local means [25][26], A-Trous filter [7], and bilateral filters [17]. However, unlike general image denoising, these works use auxiliary features to create a robust denoising kernel that can be easily obtained during rendering. The common auxiliary features are geometric features (*i.e., G-buffers*), which contains 3D geometric information (*e.g.,* albedo, normal, depth) of the first bounce of samples in image-space [20]. They provide strong geometric constraints when estimating denoising kernels.

Recent works apply machine learning for MC denoising. These works train a neural network to estimate the denoised pixel values or denoising kernels from a large training data of various scenes. Pioneering work from Kalantari et al. [16] train multi-layer perceptrons(MLPs) to estimate parameters for bilateral filters. Bako et al. [2] have proposed a kernel-predicting convolutional network (KPCN) that estimates a denoising kernel directly. Xu et al. [30] propose an adversarial training approach that helps the denoising neural network preserve high-frequency details. Finally, Yu et al. [32] suggest a self-attention mechanism guided by auxiliary features to capture relationships between features effectively. Overall, learning-based approaches have shown strength in extracting useful features from input noisy image and *G-buffers* for denoising.

However, *G-buffers* suffer from reconstructing complex lighting effects because they only contain geometric information on the first bounce. Recent works overcome the problem by introducing new auxiliary features called path features or *P-buffers* [10][19][6]. *P-buffers* include radiometric information of each bounce of light samples which are beneficial for dealing with noise from complex lighting effects such as multi-bounce reflections and caustics. Meanwhile, other kinds of auxiliary features are used for volume rendering [33] or hair rendering [24] depending on their purposes of rendering.

While a wide variety of auxiliary features can be obtained and used for denoising, exploiting high dimensional auxiliary features is still an ill-posed problem. Cho et al. [6] argue that high dimensionality of *P-buffers* makes utilization of *P-buffers* ineffective. They propose a path manifold module that embeds *P-buffers* to low-dimensional space before feeding to the denoising network. On the other hand, Zhang et al. [33] show that using a subset of effective auxiliary features gives better denoising performance than using a complete set of auxiliary features. They suggest an automatic feature selection that selects a subset of effective features that shows the best performance and efficiency.

Our framework also addresses the high-dimensionality problem when using both *G-buffers* and *P-buffers*. We use two denoising networks, each dealing with a single type of auxiliary features, and ensembles the results of the two denoising networks.

## 2.2 Ensembling for MC Denoising

Recent works argue that various MC denoising methods have systematic biases due to hand-crafted designs, choice of auxiliary features, or training data [34][1]. To solve the problem, they suggest a post-correction method for denoising that ensembles the rendered and denoised results. Back et al. [1] propose a deep-learning method that estimates ensembling kernels to ensemble the noisy and denoised results. Future works enhance the deep learning-based ensembling using error estimation techniques [9][13] for robust ensembling between noisy and denoised results. On the contrary, Zheng et al. [34] propose an optimization-based ensembling method, ensembling denoised results from multiple denoisers. Their method optimizes pixel-wise ensembling weight maps to reduce the estimated error of the ensembled result.

These works are built to post-process the results of existing denoisers and thus do not modify the denoisers. Our framework, on the other hand, shows the benefit of modifying the denoisers during ensembling by jointly training the denoisers with our ensembling network to provide an enhanced ensembled result. We provide ablation studies on test performance based on the training strategies (Chapter. 5.4) and comparison with previous ensembling method [34] (Chapter. 5.3) to show the effectiveness of our deep learning-based ensembling and joint-training.

# Chapter 3. Method

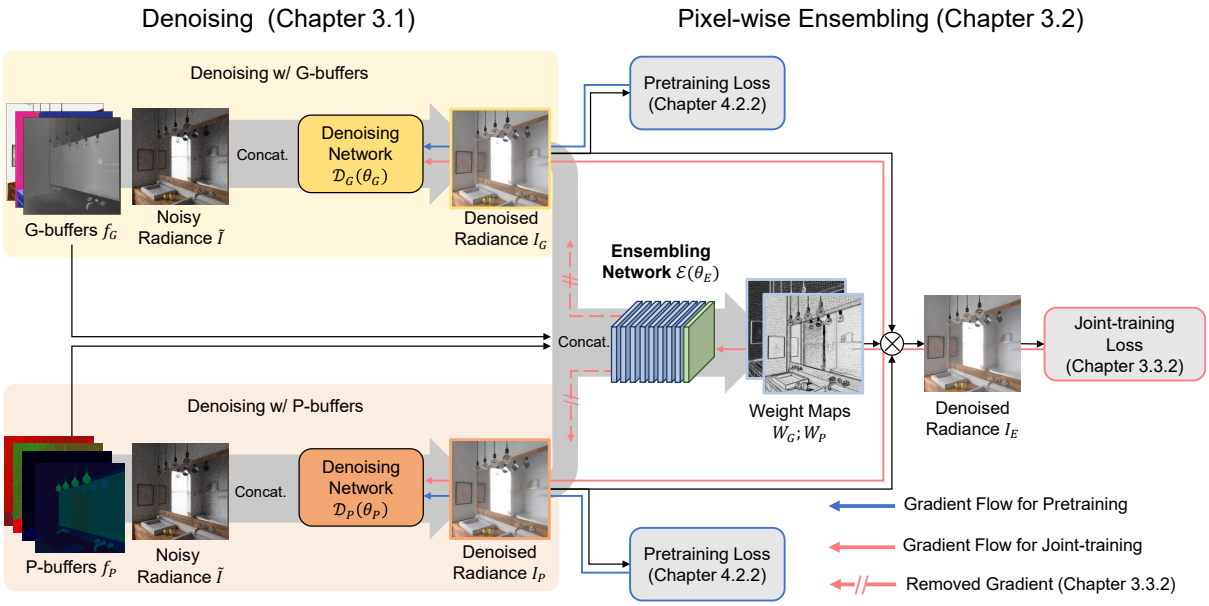


Figure 3.1: Visualization of our denoising framework. We first denoise the noisy input with only a single type of auxiliary features (*G-buffers* $f_G$ or *P-buffers* $f_P$) using independent denoising networks, $D_G$ and $D_P$. Then the ensembling network estimates the ensembling weight maps from the two denoised results $(I_G, I_P)$ and the auxiliary features $(f_G, f_P)$. The ensembling weight maps further ensemble the two denoised results as the final ensembled output $I_E$. The path manifold module that generates low-dimensional *P-buffers* is omitted in this figure for simplicity. Please refer to the details of the path-manifold module in Chapter 8.1.3.

Our novel guidance framework provides explicit pixel-wise guidance for utilizing auxiliary features in MC denoising. We demonstrate the details of our framework and the training strategies in this section.

## 3.1 Denoising with *G-* and *P-buffers*

We set two denoisers, $\mathcal{D}_G$ and $\mathcal{D}_P$, each denoising with only a single type of auxiliary features, *G-buffers* and *P-buffers*. We separate the denoising process to avoid the high-dimensionality problem of auxiliary features and to let each neural network learn different characteristics of *G-buffers* and *P-buffers* separately.

The following equations can summarize this stage:

$$I_G = \mathcal{D}_G(\theta_G; \tilde{I}, f_G), \tag{3.1}$$

$$I_P = \mathcal{D}_P(\theta_P; \tilde{I}, f_P), \tag{3.2}$$

where $\tilde{I}$ is the input noisy image, $f_G$ and $f_P$ are *G-buffers* and *P-buffers*, respectively, and $I_G$ and $I_P$ are the denoised images of the corresponding denoising networks. $\theta_G$ and $\theta_P$ are parameters of the

denoising networks $\mathcal{D}_G$ and $\mathcal{D}_P$, respectively. The denoising networks share the same structure with a simple modification in the input channel. We change the input channel of the first layer to fit the channel dimension of the *G-buffers* and *P-buffers*. We can deploy any denoising network architecture for $D_G$ and $D_P$, where we choose KPCN [2] and AdvMCD [30] for this work. We provide details of the configuration and channel number of *G-buffers* and *P-buffers* in Appendix 8.1. Please refer to the details of the network architectures from the works above.

## 3.2 Deep-learning based Pixel-wise Ensembling

In this stage, we ensemble the denoised result $I_G$ and $I_P$ of the denoisers $D_G$ and $D_P$ in a per-pixel manner. To do so, we design an ensembling network, $\mathcal{E}$, that estimates pixel-wise weight maps for ensembling from the denoised results ($I_G$, $I_P$) and the auxiliary features ($f_G$, $f_P$). We design the architecture of our ensembling network similar to KPCN [2], where we use only eight 5x5 convolutional layers with fifty hidden channels, each layer followed by a ReLU activation function. The last convolution layer has two channels followed by a sigmoid activation to estimate the weight maps with values normalized to $[0, 1]^{H\times W}$, resembling the soft masks. We normalize the weightmaps (*i.e.,* $W_G + W_P = \mathbf{1}^{H\times W}$) for stable ensembled results during training and testing [34].

In summary, estimating pixel-wise ensembling weight maps $\{W_G; W_P\}$ can be summarized as below:

$$\{W_G; W_P\} = \mathcal{E}(\theta_E; I_G, I_P, f_G, f_P), \tag{3.3}$$

where $\theta_E$ is the parameter of our ensembling network $\mathcal{E}$.

Then we ensemble the two denoised results after applying their respective weight maps as shown below:

$$I_E = I_G \otimes W_G + I_P \otimes W_P, \tag{3.4}$$

where $\otimes$ is the Hadamard product. Thus, $I_E$ is the final product of our guidance framework.

Our ensembling weight maps represents pixel-wise importance of auxiliary features when ensembling. Intuitively, given the denoised results $I_G$ and $I_P$, our weight maps should give more weight to the result that shows better reconstruction to produce an enhanced ensembled result. This reflects that the regions where our weight maps highlight (*i.e.,* assign more weights) are regions where the corresponding auxiliary features are more useful for reconstruction than another type of auxilary features. We further analyze the ensembling weight maps in Chapter 5.2

## 3.3 Jointly Training for Pixel-wise Guidance

To make further use of our ensembling weight maps for enhanced denoising, we jointly train the denoising networks and the ensembling network to provide pixel-wise guidance to the denoising networks. In this section, we describe the details and the effects of our joint-training.

### 3.3.1 Pretraining Denoisers

Training our whole framework from scratch would make our ensembling network deal with poorly denoised images from the denoising networks that are not fully trained. This may guide our ensembling network to estimate suboptimal weight maps for ensembling. To circumvent the problem, we pretrain our two denoising networks before jointly training them with our ensembling network. We describe

the detailed pretraining strategy in Chapter. 4.2.2, where strategies differ by the baseline we choose for denoising networks. We also show empirical results on how our pretraining enhances the performance of our framework in Chapter. 5.4.

### 3.3.2 Joint-Training

We jointly train the denoising networks $D_G$, $D_P$, and the ensembling network $\mathcal{E}$ to minimize the reconstruction loss of the final ensembled result $I_R$. As mentioned in the previous chapter, our ensembling network learns to assign higher weights to the denoised result that shows better quality in a pixel-wise manner.

However, even though we pretrain the denoising networks, jointly training our ensembling network and the denoising networks may unintentionally entangle the objectives of these networks. Even though the final ensembled result is plausible, the denoising networks may not properly denoise the noisy input, and the ensembling network may not estimate proper ensembling weight maps. To prevent the entanglement, we remove the gradients on $I_G$ and $I_P$ when passing them to the ensembling network as inputs [5]. By removing the gradient, the denoising networks only gets training signals from the ensembling process (see the red line in Figure. 3.1). This disentangles the objectives of our networks and get legitimate and interpretable ensembling weight maps and denoised results.

With the proper gradient removal, jointly training our denoising networks with an ensembling network enhances the denoisers to focus on regions where their corresponding auxiliary features are important. The ensembling weight maps $W_G$ and $W_P$ mask the gradients that flow to their corresponding denoising networks $D_G$ and $D_P$, similar to soft-masking.

The derivation of the gradient masking scheme is when using $\ell_1$ loss for reconstruction is shown below:

$$\frac{\partial}{\partial \theta_G}\left|I_E - \hat{I}\right| = \frac{\partial}{\partial \theta_G}\left|(I_G \otimes W_G + I_P \otimes W_P) - \hat{I}\right| \propto W_G \otimes \mathrm{sgn}(I_E - \hat{I}) \otimes \frac{\partial I_G}{\partial \theta_G}, \tag{3.5}$$

$$\frac{\partial}{\partial \theta_P}\left|I_E - \hat{I}\right| = \frac{\partial}{\partial \theta_P}\left|(I_G \otimes W_G + I_P \otimes W_P) - \hat{I}\right| \propto W_P \otimes \mathrm{sgn}(I_E - \hat{I}) \otimes \frac{\partial I_P}{\partial \theta_P}, \tag{3.6}$$

where $\mathrm{sgn}(\cdot)$ is a sign function. The aforementioned gradient removal zeros the following gradients: $\frac{\partial W_P}{\partial \theta_G} = 0$ and $\frac{\partial W_G}{\partial \theta_P} = 0$. Note that the gradient masking scheme works similarly for different reconstruction loss functions such as MSE, relativeMSE, relativeL1, and SMAPE [27].

The masked gradients enhance the denoising networks $D_G$ and $D_P$ by providing guidance on regions where each network should focus when denoising. Weight maps $W_G$ and $W_P$ contain pixel-wise information on which auxiliary features more successfully reconstruct the pixel radiance. As the weight maps mask the gradients, they leave training signals on regions where *G-buffers* or *P-buffers* are important. For instance, $W_G$ highlights pixels where *G-buffers* are vital for denoising with high weights. Thus the denoiser $D_G$ will get training signals based on those highlighted pixels during joint training. Similarly, the denoiser $D_P$ will get training signals on pixels where *P-buffers* are vital for denoising based on the highlighted pixels of $W_P$. Therefore, our joint-training strategy provides pixel-wise guidance to the denoising networks on which region they should focus for training. Chapter. 5.2 provides empirical evidence with visualization of how our joint-training enhances the denoisers.

# Chapter 4. Experiment Setups

In this section, we discuss details of our experiment settings including dataset configuration, evaluation , and training details for our framework and baseline denoisers. All experiments are done on NVIDIA RTX 3090 GPU for KPCN [2] models and NVIDIA RTX 8000 GPU for AdvMCD [30]-based models.

## 4.1 Datasets and Evaluation

### 4.1.1 Dataset Generation

We generate our own dataset using 18 scenes gathered from publicly available resources [3] and Blend Swap, and render them with our OptiX-based renderer [22] for fast generation. For training and validation dataset, we randomize the camera (*e.g.,* position, up direction, look direction), material parameters (*e.g.,* BRDF type, albedo, roughness, index of refraction), and lighting (*e.g.,* Environment map, light source) to train with diverse light transport. Note that we fix parameters that are crucial for physical correctness, such as refraction indices and specular reflectance. We render 25 randomized shots per scene with a resolution of $1280 \times 1280$, and a single shot per scene with the same resolution for validation. 256 patches with a resolution of $128 \times 128$ are randomly sampled for training and validation. Noisy images for training and validation are rendered with 2 to 8 spp in order to train our denoiser to deal with spp-dependent variance in noisy radiances and *P-buffers*. The reference images are rendered with 8000 spp. On the other hand, we render test sets with 12 scenes not used for training and validation. Note that we do not apply randomization and random patch sampling for test sets for evaluation. Noisy images for the test set are rendered with 2, 4, 8, 16, 32, and 64 spp for concise evaluation on various noise levels. The reference images for the test set are rendered with 16000 spp or 32000 spp, depending on the convergence of the scene.

We store noisy radiance, *G-buffers*, and *P-buffers* for each rendered shot of our denoising. We follow the descriptions of the previous denoising works using *G-buffers* and *P-buffers* for MC denoising [2][30][10][6]. We deliver specific configuration of *G-buffers* and *P-buffers* in Appendix 8.1

### 4.1.2 Evaluation Metrics

We apply gamma tone mapping [30] to the radiances for numerical and perceptual comparison on low dynamic range (LDR). To evaluate the performance on test sets, we use two error metrics: relative mean square error (relMSE) [25] and structural dissimilarity ($1-$SSIM) [28]. We also use relMSE to evaluate the performance on the validation set. However, the error rate varies depending on the scene. To compare numerical error across the scenes, we report the errors normalized by the error of 2-spp noisy inputs for each scene and then compute the average among the scenes [2][27][6].

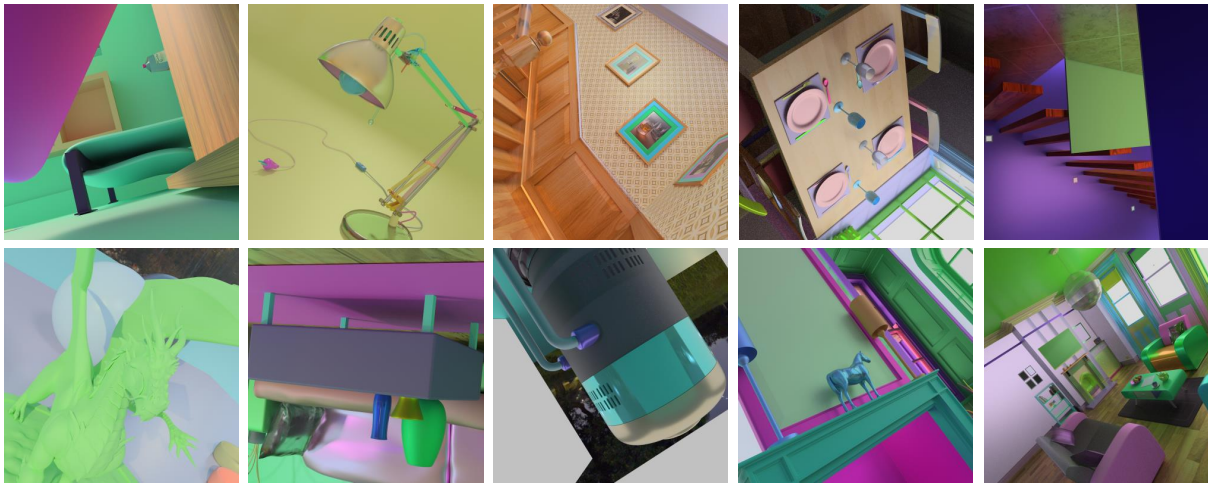

Figure 4.1: Examples of clean shots from our training set. We randomize various scene parameters to generate various light transport phenomena for training.

## 4.2 Baseline and Training Details

### 4.2.1 Baseline

We choose two learning-based MC denoising models, KPCN [2] and AdvMCD [30], to show that our method benefits various denoising models. To embed the path descriptors as *P-buffers*, we use the path-manifold module from Cho et al. [6].

### 4.2.2 Pretraining Denoisers

For KPCN using *G-buffers*, we use $\ell_1$ loss between the reference $\hat{I}$ and the denoised radiance $I$ as a reconstruction loss to train the diffuse and specular branches of KPCN with a learning rate of 1e-4. Then, we finetune both branches to minimize the $\ell_1$ loss between the reference and the final denoised radiance (diffuse and specular combined) with a learning rate of 1e-6. Both training and finetuning are done until the validation loss stops decreasing.

For AdvMCD using *G-buffers*, we use the generator of AdvMCD as the baseline denoiser for $D_G$ and $D_P$. Denoiser $D_G$ and $D_P$ have their own critic networks, which is required for adversarial training. We pretrain the denoisers with both L1 reconstruction loss and adversarial loss and gradient-penalty loss weighted with parameters $w_{adv} = 0.005$ and $w_{gp} = 10.0$. The learning rate is initially set to 1e-4, where we halve the learning rate every two epochs. We train the denoisers up to 8 epochs. Please check the detailed architecture of the denoising network, critic networks, and losses of AdvMCD in Xu et al. [30].

When using *P-buffers* as auxiliary features for the denoisers, we add a path-manifold module and jointly train the denoiser and the path-manifold module with the reconstruction loss and the path disentangling loss [6]. The two path-manifold modules are added to each of the diffuse and specular branches to provide embedded *P-buffers* for each branch. The final loss term for training KPCN using *P-buffers* is as below:

$$\mathcal{L}(I, f_P, \hat{I}) = \ell_1(I, \hat{I}) + w_{path}\mathcal{L}_{path}(f_p, I), \tag{4.1}$$

where $w_{path}$ is the balancing weight applied for path disentangling loss $\mathcal{L}_{path}(f_p, I)$ [6]. Similariy, the

final loss term for training AdvMCD using *P-buffers* is as below:

$$\mathcal{L}(I, f_P, \hat{I}) = \ell_1(I, \hat{I}) + w_{adv}\mathcal{L}_{adv}(I, \hat{I}) + w_{gp}\mathcal{L}_{gp}(I) + w_{path}\mathcal{L}_{path}(f_p, I). \tag{4.2}$$

We use the same training objective of Eq. 4.1 to train the denoisers using *G-buffers* and *P-buffers*, where these buffers are concatenated and fed to denoiser.

All models use ADAM optimizer [18], and we use a *P-buffers* size of 12 and $w_{path}$ of 0.1 which have shown the best performance for our work [6].

### 4.2.3 Applying Our Framework with Joint-training

We construct two of our denoising frameworks for each diffuse and specular branches [2][30], and train each framework to minimize the $\ell_1$ loss of the ensembled result. Note that we do not use the critic network, adversarial loss, and gradient-penalty loss for joint training when using AdvMCD as a baseline. We use a learning rate of $10^{-5}$ to train the ensembling network $\mathcal{E}$ while we use a learning rate of $10^{-6}$ to train the pretrained denoisers and the path manifold module. We train our framework until the validation loss stops decreasing to prevent overfitting.

Our implementation is based on a public code of baseline works, which are based on PyTorch [23]. All convolutional layers of our model and baselines are initialized using the Xavier method [11].

# Chapter 5. Results and Discussions

In this section, we first deliver the overall performance and analysis of our denoising framework by comparing it with the baseline denoisers KPCN [2] and AdvMCD [30]. We then provide analysis on ensembling weight maps and ablation studies on our framework. Moreover, we show comparison of our work with an optimization-based ensembling method [34]. Lastly, we discuss the computational cost of our work with additional analysis.

## 5.1 Comparison with Baseline Models

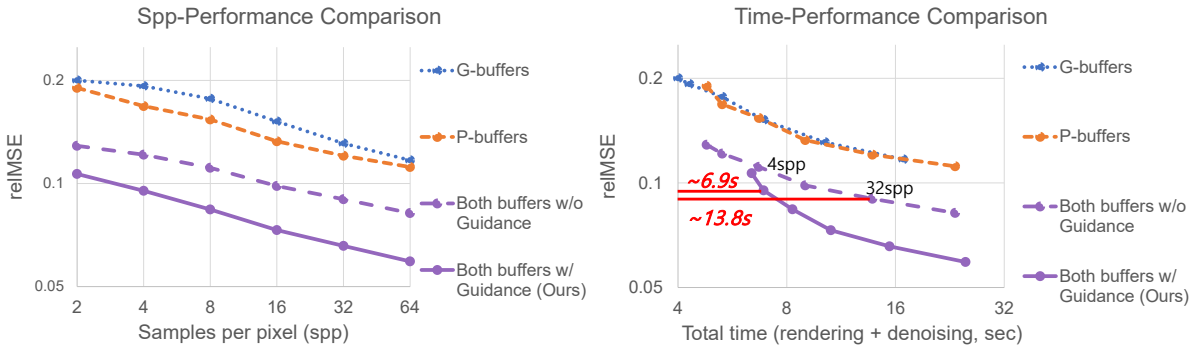


Figure 5.1: Numerical comparison of our guidance framework with baseline (KPCN [2]). Our model outperforms the baseline models using various combinations of features in denoising performance (left plot) and in efficiency (right plot).

| Time cost (secs) v.s. spp | 2 | 4 | 8 | 16 | 32 | 64 |
|---|---|---|---|---|---|---|
| Ensembling Network $\mathcal{E}$ & Multiplication (for each branch) | 0.15 | 0.15 | 0.15 | 0.15 | 0.15 | 0.15 |
| $\mathcal{D}_G$ (*i.e., G-buffers*) | 1.3 | 1.3 | 1.3 | 1.3 | 1.3 | 1.3 |
| $\mathcal{D}_P$ (*i.e., P-buffers* & Both buffers w/o Guidance) | 2.1 | 2.3 | 2.7 | 3.4 | 4.9 | 7.9 |
| Both buffers w/ Guidance (Ours) | 3.7 | 3.9 | 4.3 | 5.0 | 6.5 | 9.5 |

Table 5.1: Runtime cost breakdown of our framework with KPCN baseline. Our framework takes more time to process the same inputs due to requiring inferences of two denoising networks and our ensembling method. Note that our framework ensembles results for each diffuse and specular radiance, requiring two ensembling processes.

We first report the numerical comparison of our guidance framework and the baseline models based on KPCN [2]. The left plot of Figure 5.1 shows denoising performance on various noise levels (2-64 spp). Our framework of pixel-wise guidance outperforms the baseline models using a combination of *G-buffers* and *P-buffers* throughout all noise levels. The right plot shows denoising performance with respect to total rendering time. This plot shows that our guidance framework also outperforms the time-performance efficiency, even though our framework requires additional computation overhead for inferencing two denoising networks and ensembling (see Table 5.1 for specific analysis). Such overhead can be seen in the right plot of Figure 5.1, where the solid purple line of our framework is slightly

shifted to the right compared to the baselines. Nevertheless, for moderate sample counts (*i.e.,* 4-64 spp), our framework shows improved efficiency. For instance, our framework can reach a similar denoising performance using half of the total time (6.9 seconds) compared to the baseline model using both buffers (13.8 seconds).

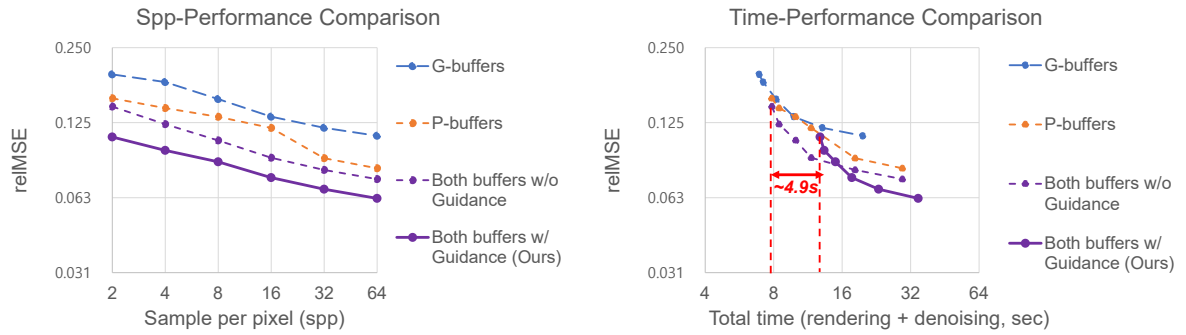


Figure 5.2: We compare the numerical results of our denoising framework with the baseline models (AdvMCD) using different combinations of auxiliary features.

We also provide a numerical comparison of our denoising framework and the baseline models based on AdvMCD [30]. Our framework also outperforms other baseline models using different combinations of auxiliary features. However, our model cannot outperform the baseline models regarding efficiency in low sample counts (*e.g.,* 2-8 spp) due to a large denoising network of AdvMCD. Because AdvMCD has more parameters than KPCN, running the denoising network takes longer. This increases our model's computational cost, which requires two passes of the denoising network before ensembling. Thus, our framework is more effective when applied to a lightweight denoising network in terms of efficiency. We provide a more thorough analysis of the computational cost of our work in Chapter. 5.5.

Figure 5.3 shows a visual comparison of the denoised result of our framework and baseline models based on KPCN. Our framework shows a better reconstruction of geometric details and textures (*e.g.,* woven basket and pasta) than the baseline models (see first and second row). Our framework also shows enhanced results on reconstructing complex lighting effects like reflected details (*e.g.,* reflection on the glossy floor and glass) compared to the baseline model (see third and fourth row). Especially, the refracted pipe of the hookah scene on the fourth row is hardly reconstructed when using both buffers without guidance, while it is fairly reconstructed by baseline using only *P-buffers*. Our framework enhances the reconstruction of the refracted pipe with visible edges, showing less numerical error on the scene. Thus, our work can effectively utilize both *G-buffers* and *P-buffers* for denoising.

## 5.2 Analysis on Pixel-wise Guidance for Utilizing Auxiliary Features

We analyze how our pixel-wise guidance framework and joint-training strategy combine the advantages of *G-buffers* and *P-buffers* in this section. Figure 5.4 shows the error maps of the denoisers $D_G$ (first row) and $D_P$ (second row) before and after joint-training of our framework. The scene on the first row contains various geometric details, where *G-buffers* plays more role in denoising. Our weight map $W_G$ highlights regions where geometric information is important such as edges and textures. By joint training, denoiser using *G-buffers*, $D_G$, is further trained to denoise with a focus on the highlighted

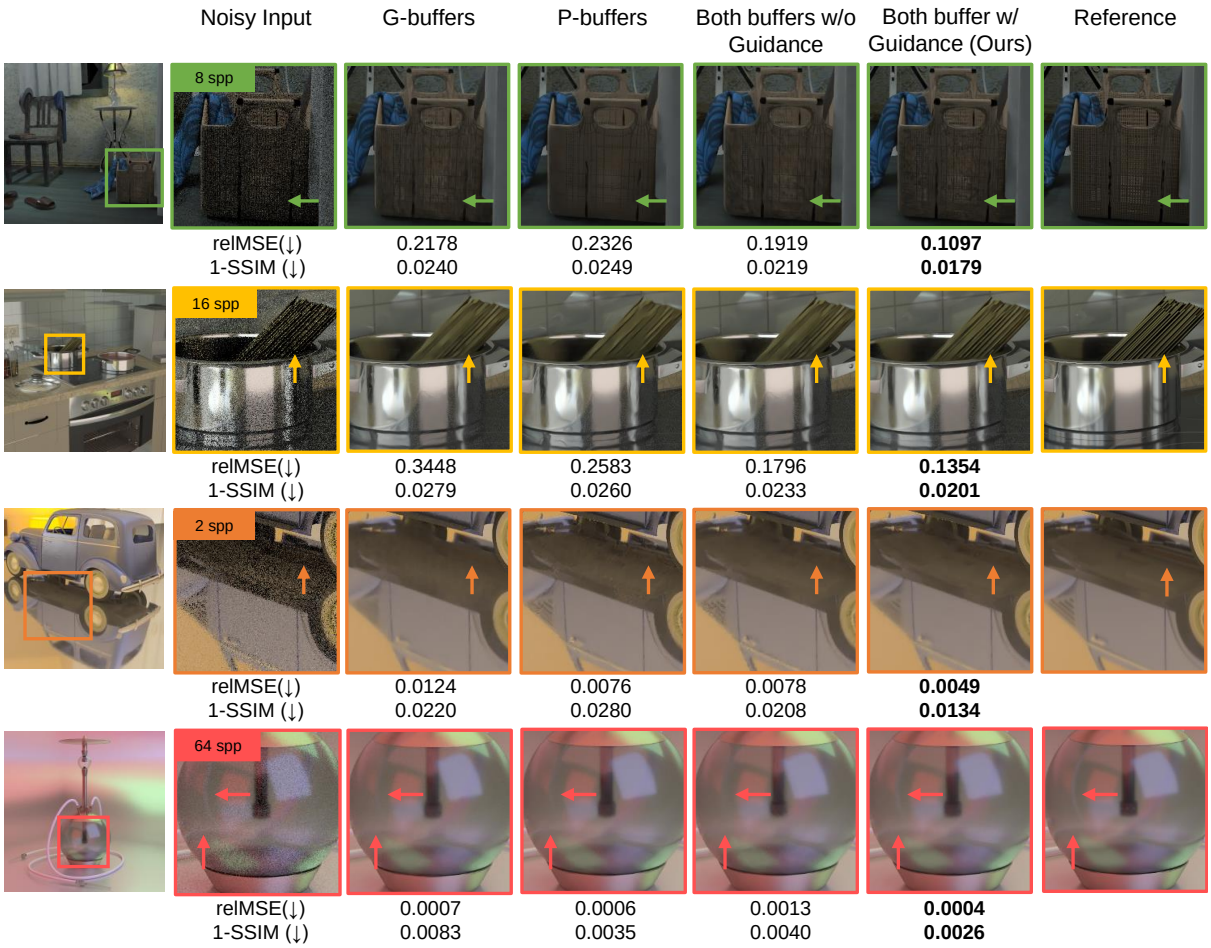


Figure 5.3: Visual comparison of our denoising framework and baseline models. Our pixel-wise guidance shows enhanced reconstruction on both geometric details (first and second row) and complex lighting effects (thrid and fourth row). We further provide the comparison of the diffuse and specular branches in Appendix 8.2. Best viewed with zoom-in.

regions on the weight map $W_G$, showing less error on those regions than before. A similar phenomenon happens for denoising using *P-buffers*, $D_P$. The scene on the second row contains various reflection details, where *P-buffers* plays more role. $D_P$ is jointly-trained to denoise with a focus on regions where *P-buffers* are vital with guidance from the weight map $W_P$. These regions with lower errors are further ensembled with higher weights, enhancing the final ensembled result.

Further analysis shows that our guidance framework can utilize *G-buffers* and *P-buffers* in a complementary manner, resulting in robust denoising on various noise levels (spp). On low sample counts, *P-buffers* tend to be noisy because the radiometric values of the light sample depend on whether the sample reaches the light source or not. On the other hand, *G-buffers* provide relatively clean features throughout sample counts because the geometric information on the first bounce of light samples is relatively stationary. Thus, denoising with *P-buffers* on low sample counts (*i.e.,* 4 spp) results in noisy reconstruction compared to denoising with *G-buffers*, even though the target scene in Figure 5.5 contains reflected details on the glass door (see first row). On the other hand, the noise in *P-buffers* reduces when using an abundant number of samples (*i.e.,* 64 spp), showing improved reconstruction result compared to denoising with *G-buffers* (see second row).

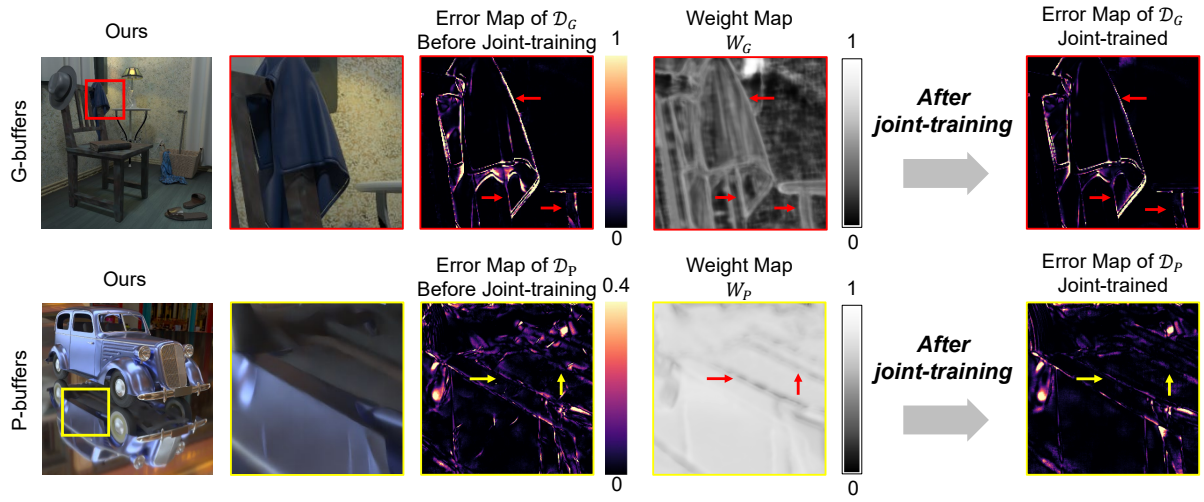


Figure 5.4: Analysis on our joint-training strategy on denoising networks. Our joint-training reduces the errors of each denoising network based on the highlighted regions of their corresponding weight maps, where the highlighted regions represent the importance of *G-buffers* or *P-buffers* when denoising.

However, our framework gives more emphasis on better-denoised results for ensembling. Our framework gives more emphasis on the denoised result using *G-buffers* for low sample counts or on the denoised result using *P-buffers* for high sample counts. Thus, in both cases, our framework provides an enhanced ensembled result.

## 5.3 Comparison with Optimization-based Ensembling Method

We further compare our work with an optimization-based ensembling approach for MC denoising (ED) [34] to show the effectiveness of our deep learning-based ensembling framework and our joint-training strategy. ED ensembles result of multiple denoisers without manipulating the denoising methods. We prepared a new test dataset applicable to our method and ED, supporting half-buffers techniques for ED's optimization. We also modified ED to estimate per-pixel weight maps instead of per-channel (weights for RGB channels each) for a fair comparison with our work.

Numerical comparison between our guidance framework and ED on Figure. 5.6 shows that our framework is more powerful and efficient for ensembling. The left plot shows that our guidance framework outperforms ED in all noise levels, showing marked improvement by using more samples per pixel. Our deep learning-based ensembling with joint-training shows more robust ensembling compared to the optimization-based method that does not modify the denoisers. In addition, our framework outperforms ED regarding time-performance efficiency (right plot). ED's optimization approach requires heavy computation using three radiances (one from full buffers, two from each half-buffers) and auxiliary features for optimizing ensembling weight maps for every scene. On the other hand, our framework can denoise with inferences of denoising and ensembling networks, which is computationally efficient compared to ED.

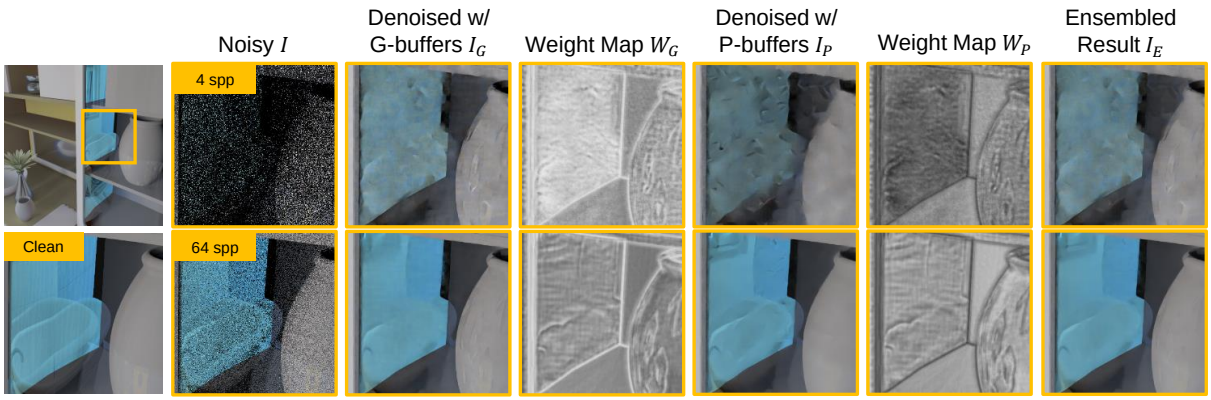


Figure 5.5: Comparison of ensembling weight maps obtained for low sample counts (4 spp) and high sample counts (64 spp). The quality of $I_P$ depends on the noise in *P-buffers*, which the noise is severe in low sample counts. Our ensembling network gives more emphasis on better-reconstructed results between $I_G$ and $I_P$ and returns the enhanced ensembled result, reducing the effect of the noise in *P-buffers* .

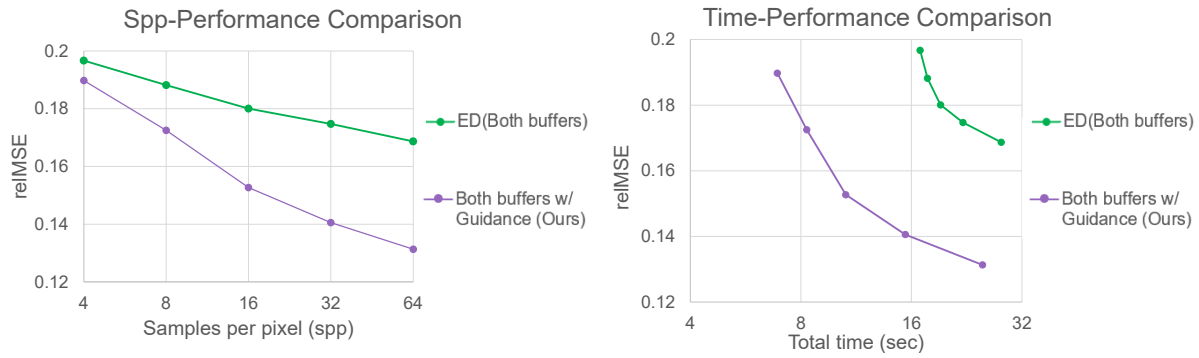


Figure 5.6: Numerical comparison of our guidance framework with optimization-based ensembling method (ED) [34]. Our framework outperforms ED in both denoising performance (left plot) and efficiency (right plot), showing the benefit of deep learning-based ensembling and joint-training of the denoising networks.

## 5.4 Ablation Studies

In this section, we deliver ablation studies about our training strategy for our framework and input configuration for our ensembling network. We summarize numerical results for ablation studies in Table. 5.2.

### 5.4.1 Training Stratagies

We apply various training strategies besides our joint-training for our guidance framework and compare the denoising performance (see Ablation #1 in Table 5.2). Our joint-training strategy shows the best test performance on both relMSE and structural dissimilarity metrics. Training only the ensembling network after pretraining the denoising networks (*Fix-N-Train*) shows improved test performance than the baseline denoiser using both buffers but does not outperform our framework jointly trained. This reflects the effectiveness of our joint-training strategy that enhances the denoisers based on the ensembling weight maps. In addition, training all networks of our framework from scratch (*Full-N-Full*) shows worse

Ablation #1 Training Strategies for Our Denoising Framework

| Method | Description | relMSE(↓) | 1−SSIM(↓) |
|---|---|---|---|
| Baseline | Baseline denoiser using both buffers | 0.10529 | 0.05151 |
| *Full-N-Full* | Train all models from scratch | 0.11245 | 0.04957 |
| *Fix-N-Train* | Fix pretrained denoisers & Train ensembling model | 0.09652 | 0.04699 |
| ***Joint-training* (Ours)** | **Train pretrained denoisers & ensembling model** | **0.07983** | **0.04411** |

Ablation #2 Input Configuration for Our Ensembling Network

| Input Configuration | relMSE(↓) | 1−SSIM(↓) |
|---|---|---|
| Denoised radiances only ($I_G$, $I_P$) | 0.10520 | 0.04846 |
| + ***G-buffers*** **($f_G$),** ***P-buffers*** **($f_p$) (Ours)** | **0.07983** | **0.04411** |

Table 5.2: Ablation study on training strategies for our denoising framework (upper table) and input configuration for our ensembling network $\mathcal{E}$ (lower table).

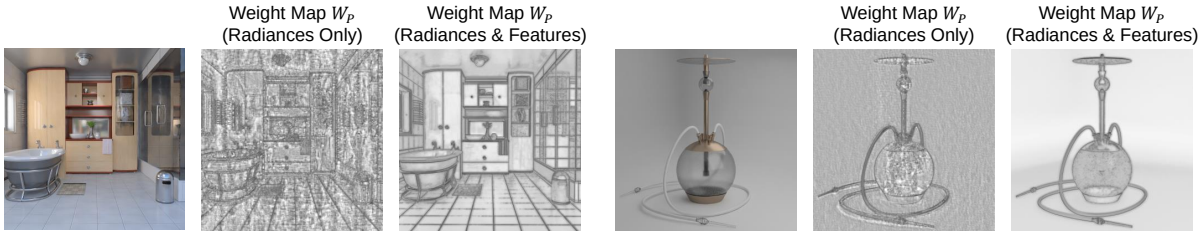


Figure 5.7: Visual comparison of weight maps obtained using different input configurations. Weight maps estimated from radiances and auxiliary features are cleaner than the estimated result with only radiances. Cleaner weight maps provide more reliable pixel-wise guidance for ensembling and enhancing denoisers during joint-training.

results than the baseline, showing that the pretraining stage is required for stable and optimal training (see Chapter. 3.3.1).

### 5.4.2 Input Configuration

We further test different input configurations for our ensembling network and compare the denoising performance (see Ablation #2 in Table 5.2). Estimating ensembling weight maps only with denoised radiances show worse performance than estimating with denoised radiances and auxiliary features. The visualization of estimated weight maps from Figure. 5.7 shows that weight maps estimated only with denoised radiances are noisy compared to the ones estimated using both radiances and auxiliary features. We conjecture that the auxiliary features provide practical guidance on estimating clean weight maps, which is beneficial for ensembling and enhancing denoisers during joint-training. Such analysis is similar to Zheng et al. [34], where they applied cross bilateral filters based on *G-buffers* to remove noises in the estimated ensembling weight maps for smooth ensembled results.

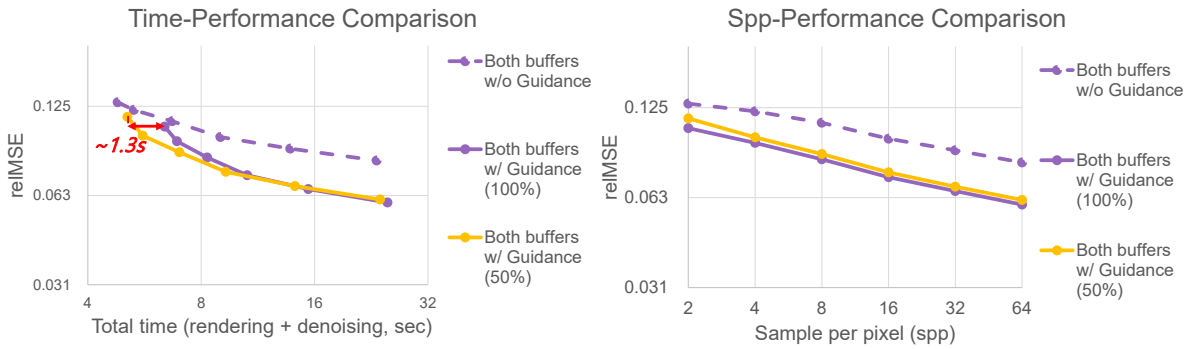


Figure 5.8: Numerical comparison of our reduced framework (50%) with our original framework (100%) and baseline denoiser (KPCN) using both buffers. Our reduced framework shows better efficiency than our original framework, especially on low sample counts (left plot). However, using few parameters leads to slightly reduced denoising performance (right plot).

## 5.5 Analysis on Computational Cost and Performance

In Chapter. 5.1, we have discussed that the performance of our framework comes with an additional computational cost. The right plot of Figure. 5.1 shows that our framework based on KPCN is less favorable in dealing with low spp inputs, where our framework takes 1.6 more seconds to process the 2 spp input compared to baseline KPCN using both buffers. Also, the computational overhead increases as we use a large denoising network (*i.e.,* AdvMCD). The right plot of Figure. 5.2 shows that the computational overhead of our framework based on AdvMCD to process 2 spp input is increased to 4.9 seconds compared to baseline AdvMCD using both buffers.

To find the correlation between the capacity of our framework and the time-performance efficiency, we reduced the number of learnable parameters of our framework based on KPCN to 50% by reducing the hidden layers of the denoising networks and ensembling network to half. Then we compared the test performance and the time-performance efficiency with our original framework based on KPCN and baseline KPCN using both buffers.

Our reduced framework shows improved efficiency compared to our original framework, saving 1.3 seconds for processing the same input and requiring only 0.3 seconds to process the input compared to the baseline denoiser using both buffers (left plot of Figure. 5.8). We conjecture that our framework may be more beneficial when using lightweight denoising networks for baseline, such as works that target real-time MC denoising [4][21][8].

On the other hand, using few parameters results in reduced test performance. Our reduced framework shows slightly reduced (3.7-7.8%) denoising performance throughout various noise levels (right plot of Figure. 5.8). However, still, our reduced framework highly outperforms the baseline denoiser in denoising performance and efficiency due to our pixel-wise guidance and joint-training.

# Chapter 6. Limitations and Future Works

Despite of additional experiments on Chapter. 5.5, our framework is still slower than using a single denoising network because our framework involves an ensembling process and inference of two denoising networks. We have shown that the computational overhead increases as we use a large baseline denoising network. We have also shown that we can increase efficiency by reducing the capacity of our framework, but it comes with a tradeoff in denoising performance.

To this end, maintaining the denoising performance while reducing the computation of our framework will be an interesting future work. Recent works for ensemble learning focus on reducing the computation and capacity of child models, analogous to the denoisers of our framework, for efficient ensembling. For instance, Whitaker et al. [29] create multiple randomly pruned child models from a pretrained parent model and then finetune the child models based on the ensembled result. Similarly, we can prune the denoisers and reduce the computational cost of denoising while enhancing the final ensembled result for denoising. Moreover, our framework can provide guidance in utilizing auxiliary features to learning-based MC denoisers using a single column of a neural network via knowledge distillation [12][31].

Another direction for the future will be expanding our framework to deal with a wide variety of auxiliary features. There are domain-specific auxiliary features for volume rendering and hair rendering, where our pixel-wise guidance can be very useful.

# Chapter 7. Conclusion

We have proposed a novel guidance framework that provides pixel-wise guidance of utilizing auxiliary features to the denoisers. Our guidance is generated as pixel-wise ensembling weight maps that are used to ensemble the results of two denoisers using *G-buffers* and *P-buffers* each. We also applied joint-training of the ensembling network and the denoisers to propagate the pixel-wise guidance to the denoisers. Our framework is applicable to various learning-based MC denoisers with slight modification and training. Despite the additional computational cost, our framework has shown a considerable increase in both denoising performance and time-performance efficiency. We hope that our guidance can be further applied to effectively use various auxiliary features for rendering.

This work is an extended version of the published work [14] with additional experiments on baseline denoiser AdvMCD [30].

# Chapter 8. Appendix

## 8.1 Contents for *G-buffers* and *P-buffers*

### 8.1.1 *G-buffers* Configuration for KPCN [2]

We acquire *G-buffers* following the instructions of Bako et al. [2] as below:

- Albedo of the first non-specular bounce (3), its derivatives of $x$ and $y$ (2×3), and variance (1),
- Normal of the first non-specular bounce (3), its derivatives of $x$ and $y$ (2×3), and variance (1),
- Depth of the first non-specular bounce (1), its derivatives of $x$ and $y$ (2×1), and variance (1).

Results to a total 24 channels per pixel for each diffuse and specular branch of KPCN.

### 8.1.2 *G-buffers* Configuration for AdvMCD [30]

We acquire *G-buffers* following the instructions of Xu et al. [30] as below:

- Albedo of the first non-specular bounce (3),
- Normal of the first non-specular bounce (3),
- Depth of the first non-specular bounce (1).

Results to a total 7 channels per pixel for each diffuse and specular branch of AdvMCD.

### 8.1.3 Path Descriptor Configuration and Path Manifold Module for *P-buffers*

We follow the logic and the process of Cho et al. [6] to acquire path descriptors and embedding them to *P-buffers* using the path manifold module. Note that our OptiX-based renderer sets maximum bounce of light sample to $k = 5$. Therefore the maximum number of vertices of each sample path is 6.

The path descriptors are as below:

- Radiance undivided by the sampling probability per sample (3)
- Photon energy per sample (3)
- Sampling probability per sample (1)
- Attenuation factor of each color channel per vertex (3×6)
- Material-light interaction tag per vertex, *i.e.,* reflection, transmission, diffuse, glossy, specular (1×6)
- Roughness of BSDF per vertex (1×6)

Results to a total 36 channels for path descriptors. These path descriptors are embedded as *P-buffers* via path-manifold module [6] (see Fig. 8.1).

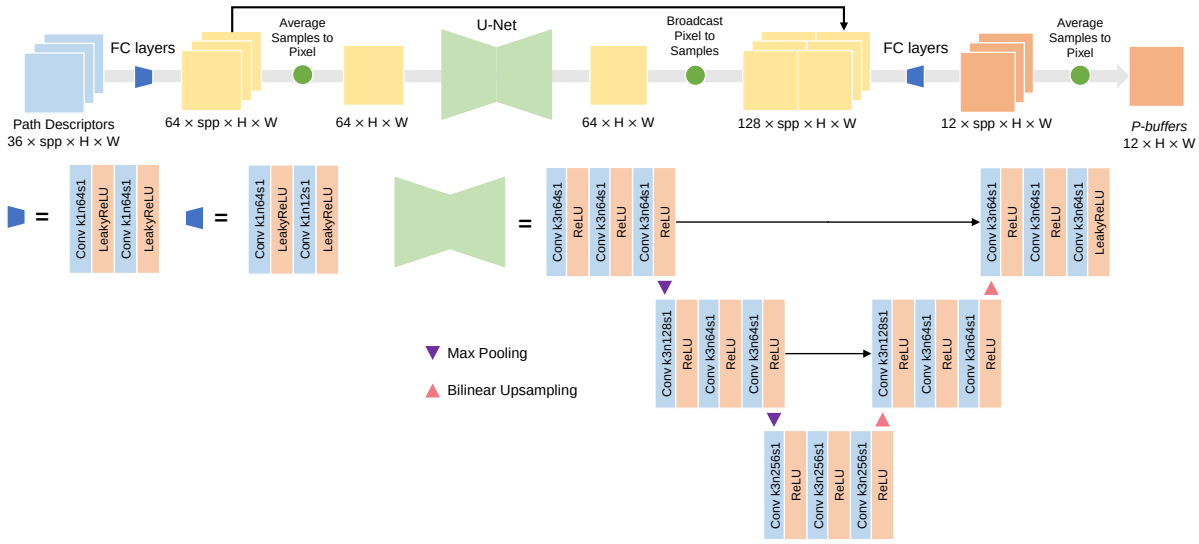


Figure 8.1: Demonstration of the path-manifold module [6]. The module embeds high-dimensional path descriptors for each sample into a lower dimensional space. Then it averages the embeddings into a pixel level to be applied to the pixel-space denoiser (*e.g.,* KPCN). The network is annotated in order of kernel size(k), number of channels(n) and stride(s) [30]. We add padding to preserve spatial dimension when needed, and use 0.01 as the slope for LeakyReLU.

## 8.2 Qualitative Results of Diffuse and Specular Branches

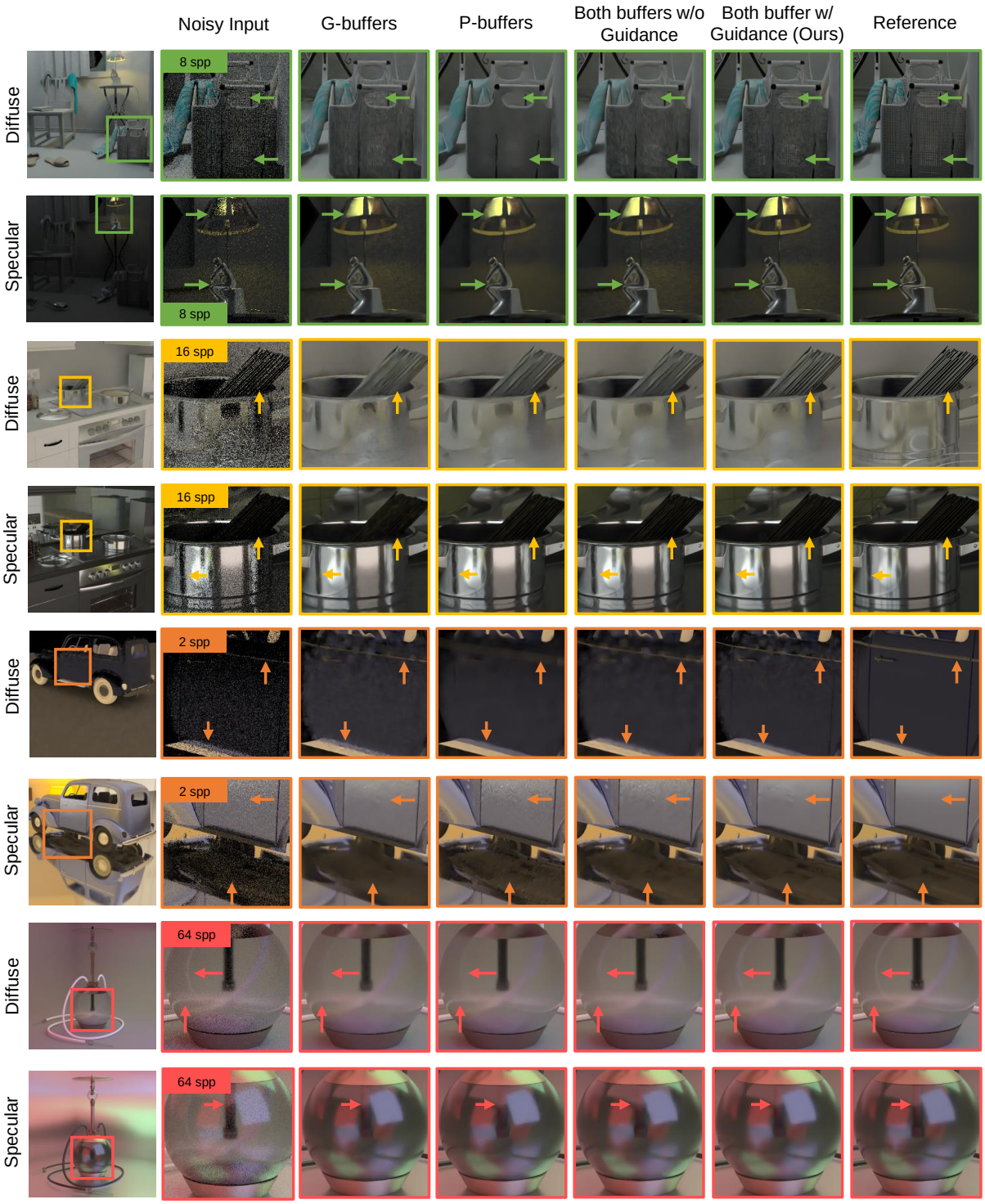


Figure 8.2: Denoised results of both diffuse and specular branch of our framework and baseline models on test scenes from Fig. 5.3. Our results demonstrate that our pixel-wise guidance effectively exploits *G-buffers* and *P-buffers* on denoising both diffuse and specular radiances compared to the baseline.

# Bibliography


[1] Jonghee Back, Binh-Son Hua, Toshiya Hachisuka, and Bochang Moon. Deep combiner for independent and correlated pixel estimates. *ACM Trans. Graph.*, 39(6):242–1, 2020.

[2] 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.

[3] Benedikt Bitterli. Rendering resources, 2016. https://benedikt-bitterli.me/resources/.

[4] 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.

[5] Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15750–15758, 2021.

[6] In-Young Cho, Yuchi Huo, and Sung-Eui Yoon. Weakly-supervised contrastive learning in path manifold for monte carlo image reconstruction. *ACM Trans. Graph.*, 40(4):38–1, 2021.

[7] Holger Dammertz, Daniel Sewtz, Johannes Hanika, and Hendrik PA Lensch. Edge-avoiding a-trous wavelet transform for fast global illumination filtering. In *Proceedings of the Conference on High Performance Graphics*, pages 67–75. Citeseer, 2010.

[8] Hangming Fan, Rui Wang, Yuchi Huo, and Hujun Bao. Real-time monte carlo denoising with weight sharing kernel prediction network. In *Computer Graphics Forum*, volume 40, pages 15–27. Wiley Online Library, 2021.

[9] Arthur Firmino, Jeppe Revall Frisvad, and Henrik Wann Jensen. Progressive denoising of monte carlo rendered images. 41(2):1–11, 2022.

[10] Michaël Gharbi, Tzu-Mao Li, Miika Aittala, Jaakko Lehtinen, and Frédo Durand. Sample-based monte carlo denoising using a kernel-splatting network. *ACM Transactions on Graphics (TOG)*, 38(4):1–12, 2019.

[11] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[12] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129(6):1789–1819, 2021.

[13] Jeongmin Gu, Jose A Iglesias-Guitian, and Bochang Moon. Neural james-stein combiner for unbiased and biased renderings. *ACM Transactions on Graphics (TOG)*, 41(6):1–14, 2022.

[14] Kyu Beom Han, Olivia G Odenthal, Woo Jae Kim, and Sung-Eui Yoon. Pixel-wise guidance for utilizing auxiliary features in monte carlo denoising. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 6(1):1–19, 2023.

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

[16] 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.

[17] Nima Khademi Kalantari and Pradeep Sen. Removing the noise in monte carlo rendering with general image denoising algorithms. 32(2pt1):93–102, 2013.

[18] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[19] Weiheng Lin, Beibei Wang, Jian Yang, Lu Wang, and Ling-Qi Yan. Path-based monte carlo denoising using a three-scale neural network. In *Computer Graphics Forum*, volume 40, pages 369–381. Wiley Online Library, 2021.

[20] Michael D McCool. Anisotropic diffusion for monte carlo noise reduction. *ACM Transactions on Graphics (TOG)*, 18(2):171–194, 1999.

[21] Jacob Munkberg and Jon Hasselgren. Neural denoising with layer embeddings. 39(4):1–12, 2020.

[22] Steven G Parker, Heiko Friedrich, David Luebke, Keith Morley, James Bigler, Jared Hoberock, David McAllister, Austin Robison, Andreas Dietrich, Greg Humphreys, et al. Gpu ray tracing. *Communications of the ACM*, 56(5):93–101, 2013.

[23] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[24] R Ramon Currius, U Assarsson, and E Sintorn. Real-time hair filtering with convolutional neural networks. *Neural Networks*, 5:1, 2022.

[25] 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.

[26] 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.

[27] 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.

[28] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[29] Tim Whitaker and Darrell Whitley. Prune and tune ensembles: Low-cost ensemble learning with sparse independent subnetworks. *arXiv preprint arXiv:2202.11782*, 2022.

[30] Bing Xu, Junfei Zhang, Rui Wang, Kun Xu, Yong-Liang Yang, Chuan Li, and Rui Tang. Adversarial monte carlo denoising with conditioned auxiliary feature modulation. *ACM Trans. Graph.*, 38(6):224–1, 2019.

[31] Lucas D Young, Fitsum A Reda, Rakesh Ranjan, Jon Morton, Jun Hu, Yazhu Ling, Xiaoyu Xiang, David Liu, and Vikas Chandra. Feature-align network with knowledge distillation for efficient denoising. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 709–718, 2022.

[32] Jiaqi Yu, Yongwei Nie, Chengjiang Long, Wenju Xu, Qing Zhang, and Guiqing Li. Monte carlo denoising via auxiliary feature guided self-attention. *ACM Transactions on Graphics (TOG)*, 40(6):1–13, 2021.

[33] Xianyao Zhang, Melvin Ott, Marco Manzi, Markus Gross, and Marios Papas. Automatic feature selection for denoising volumetric renderings. 41(4):63–77, 2022.

[34] Shaokun Zheng, Fengshi Zheng, Kun Xu, and Ling-Qi Yan. Ensemble denoising for monte carlo renderings. *ACM Transactions on Graphics (TOG)*, 40(6):1–17, 2021.

# Acknowledgments in Korean

WIP

# Curriculum Vitae in Korean

이　　름: 한 규 범

## 학　　력

2014. 2. – 2017. 2.　KAIST 부설 한국과학영재학교

2017. 2. – 2021. 8.　한국과학기술원 전산학부 및 전기및전자공학부(학사,부전공)

2021. 8. – 2023. 8.　한국과학기술원 전산학부 (석사)

## 경　　력

2018. 12. – 2019. 2.　SK 하아닉스 QRA 인턴

2019. 12. – 2020. 3.　휴멜로(Humelo) 데이터 엔지니어링 인턴

2021. 8. – 2022. 2.　한국과학기술원 전산학부 조교 (CS482, 대화형컴퓨터그래픽스)

2022. 2. – 2023. 8.　한국과학기술원 전산학부 조교 (CS206, 데이타구조)

## 연 구 업 적

1. **Kyu Beom Han**, Olivia G. Odenthal, Woo Jae Kim, and Sung-Eui Yoon, "Pixel-wise Guidance for Utilizing Auxiliary Features in Monte Carlo Denoising", *Proc. ACM Comput. Graph. Interact. Tech.*, vol. 6, issue. 1, no. 11, pp. 1-19, May. 2023
2. **Kyu Beom Han** and Sung-Eui Yoon, "반사 공간 정보를 이용한 몬테카를로 렌더링 이미지 노이즈 제거", *한국정보과학회 2022 한국컴퓨터종합학술대회 (KCC 2022).*, vol. 49, no. 1, pp. 1438-1440, Jun. 2022