Federal University of Pernambuco

Center of Informatics - CIn

PATH TRACING AND CONVOLUTIONAL NEURAL NETWORKS APPROACH FOR PHOTO-REALISTIC RENDERING

MARCEL SANTANA SANTOS

B.Sc. Dissertation

Recife/2018, July

Marcel Santana Santos

PATH TRACING AND CONVOLUTIONAL NEURAL NETWORKS APPROACH FOR PHOTO-REALISTIC RENDERING

A B.Sc. Dissertation presented to the Center for Informatics of Federal University of Pernambuco in partial fulfillment of the requirements for the degree of Bachelor in Computer Engineering.

Advisor: Tsang Ing Ren

Recife
2018, July

Marcel Santana Santos
Path Tracing and Convolutional Neural Networks Approach for Photo-realistic Rendering/ Marcel Santana Santos. – Recife, 2018, July-
56 p. : il. (algumas color.) ; 30 cm.

Advisor Tsang Ing Ren

B.Sc. Dissertation – Universidade Federal de Pernambuco, 2018, July.

1. Palavra-chave1. 2. Palavra-chave2. I. Orientador. II. Universidade xxx. III. Faculdade de xxx. IV. Título

CDU 02:141:005.7

Trabalho de conclusão de curso apresentado por **Marcel Santana Santos** ao programa de Graduação em Engenharia da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título **Path Tracing and Convolutional Neural Networks Approach for Photo-realistic Rendering**, orientada pelo **Prof. Tsang Ing Ren** e aprovada pela banca examinadora formada pelos professores:

Prof. Tsang Ing Ren (Orientador)
Centro de Informática/UFPE

Prof. Silvio de Barros Melos
Centro de Informática/UFPE

Recife
2018, July

I dedicate this work to all that believe that hard-working, dedication, and persistence are the (only) way to success.

Acknowledgements

Firstly, I would like to thank my mother, father, and grandmother for supporting me whenever necessary, doing their best to assist me with anything necessary for my education. I thanks especially to professor Tsang for giving me the necessary support during my final years in graduation and for the development of this work. Also, I would like to thank Thais Aquino, for all positive influence. To the professors, Silvio Melo, Divanilson Campelo, Daniel Cunha and Veronica Teichrieb for all teachings and works developed together. Silvio Melo, in special, is one of the most important influences in my interest in research and Computer Graphics.

Resumo

Renderização fotorrealista é um problema custoso computacionalmente. A técnica conhecida por *Path Tracing*, formulada inicialmente por James Kajiya, demanda um número muito grande de amostras por pixel, mesmo utilizando o processo de integração de Monte Carlo e técnicas de *smart sampling*, para gerar cenas com alto grau de realismo. Isso faz com que o processo de renderização tome um tempo considerável, o que torna proibitivo o seu uso em aplicações de tempo real (como jogos, realidade aumentada e virtual) e encarece a produção de filmes. Por outro lado, o uso de poucas amostras por pixel resulta em uma imagem ruidosa. Uma abordagem para contornar esse problema é renderizar a imagem utilizando poucas amostras por pixel e aplicar técnicas de pós-processamento para obter um resultado melhorado. Contudo, as técnicas tradicionais de pós-processamento são limitadas ou precisam de um ajuste complicado dos seus parâmetros. Assim, a proposta deste projeto é construir uma rede neural convolucional que produz um filtro para imagens geradas por meio de *Path Tracing*. Esse filtro gera imagens sem ruído a partir de imagens ruidosas.

Palavras-chave: Renderização; Integração de Monte Carlo; Redes Neurais Convolucionais

Abstract

Path tracing can deliver beautiful images. However, it needs thousands of samples per pixel to generate good results, even using Monte Carlo integration and smart-sampling techniques. It makes the rendering process take a long time. This delay is prohibitive to real-time applications (such as games, virtual reality, and augmented reality) and increases the cost to the movie industry. On the other hand, using low samples per pixel results in a very noisy image. An approach to deal with this trade off is rendering a low sampled image and so apply post-processing techniques to get a better result. However, the traditional post-processing techniques are either limited or need a very trick parameters tuning. Thus, the purpose of this project is to build a Convolution Neural Network which delivers a filter to Path Traced images. This filter generates noise-free images from noisy ones.

List of Figures

List of Tables

Contents

1

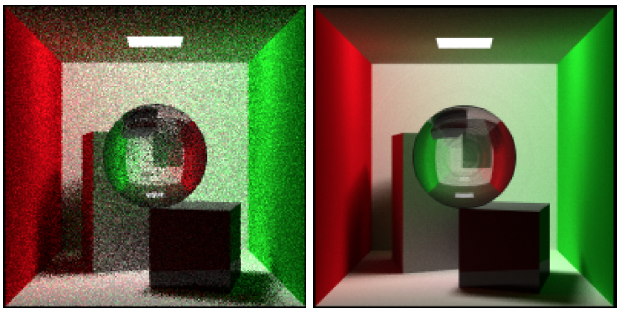
Introduction

The last few years have seen an explosion in the use of computer graphics and realistic image synthesis. This is particularly the case in the movie industry that makes extensive use of computer-generated animation and visual effects that seamlessly integrate with real film footage. Recently, many production studios have switched their rendering pipeline from ad-hoc algorithms to physically-based approaches (KELLER et al., 2015). Computer games are also presenting increasingly realistic worlds in real time. Outside the entertainment industry, synthetic photo-realistic images are also very common in design, architecture, advertising, among others.

Synthesizing this realistic images from a virtual scene means simulating the light transport from light source, through the scene, to the camera. The mathematical model which describes how light interacts with and bounces on materials are well known and have been proposed by KAJIYA (1986) through the rendering equation. Its recursive nature and the integration range evidently yields a high complexity, thus approximations are required for practical applications. Path tracing applying the Monte Carlo (MC) method, also known as MC rendering, is the most straightforward approximation to the rendering equation and is a simple technique to use physically based materials and illumination.

Path tracing consists on casting multiple random light rays from the camera to the scene as samples to integrate the rendering equation for each image pixel by using the Monte Carlo path tracing method to obtain an approximation of each pixel color. The mean intensity of several samples constitutes a noisy estimate of the total illumination. The expectation of this estimate is the ground truth value: the result is unbiased, which means that if the number of samples per pixel is increased, it is guaranteed to converge to the true solution of the rendering equation, taking into account all the lighting effects without any algorithmic trick. However, the variance of the Monte Carlo estimator only decreases linearly with respect to the number of samples per pixel. Thus, it is necessary a high number of samples per pixel in order to obtain the true solution of the rendering equation. In non-trivial scenes, with complex light transport, the convergence of each pixel can easily take over hundred thousands of samples, even with smart sampling techniques. When a scene is under-sampled, a stochastic noise, also called MC noise, usually appears as shown in the Figure 1.1(a), opposed to the same scene with many times more samples in the Figure 1.1(b). This requirement implies it may take from minutes to hours,

depending on the scene complexity, to generate a noise-free image. This delay is prohibitive to real-time applications (such as games, augmented reality, and virtual reality) and increases the cost to the movie industry. The economy of any percentage of the computing costs can yield an economy in the order of millions of dollars for big animation studios.



(a) Scene rendered using 8 spp

(b) Scene rendered using 512 spp

Figure 1.1: Comparison between two images rendered using different number of samples per pixel.

Since convergence to the correct solution using only MC rendering can be extremely slow, a wide variety of noise or variance reduction strategies have emerged over the years. For instance, different path sampling strategies (importance sampling techniques and light paths generation approaches), leading to algorithms such as Bidirectional Path Tracing (LAFORTUNE; WILLEMS, 1993), Photon Mapping (JENSEN, 1996), Metropolis Light Transport (VEACH; GUIBAS, 1997) or Vertex Connection and Merging (GEORGIEV et al., 2012); statistical techniques (quasi-Monte Carlo sampling using low-discrepancy sequences); or denoising methods, to name the most prominent ones. We refer to the reader to the recent survey by ZWICKER et al. (2015) which recovered a very large number of these approaches developed over the last years. Although different path sampling techniques deliver better convergence rates, some difficult effects and zones of the scene (such as caustics, motion blur, glossy reflections, global illumination and depth of field) often remains noisy and still need a lot more time to converge.

The denoisers have been widely used as a post-processing step to improve synthetic images. This allows to obtain a nice-look resulting image but also adds some bias. This bias, however, is tolerable for most visual applications. There are several approaches for MC rendering denoising (MOON et al. (2016); BITTERLI et al. (2016)), some of these uses rendering-specific guiding data, such as the pixel's depth in the scene, normal directions, texture colors etc. These features can help to preserve sharp edges and image texture because they can be extracted with less noise than the final color and also can encode valuable information about the scene composition and geometry. Traditionally, denoising techniques need a very difficult parameters tuning, it motivated KALANTARI; BAKO; SEN (2015) to apply Machine Learning (ML) techniques such as Multilayer Perceptron (MLP) to learn the optimal local parameters of either joint bilateral filter or a joint non-local means.

Most recently, motivated by applications of Convolutional Neural Networks (CNNs) (LECUN et al., 1995) in many computer vision tasks such as object recognition and detections (LECUN; BENGIO; HINTON (2015); RUSSAKOVSKY et al. (2015)), super-resolution (BRUNA; SPRECHMANN; LECUN, 2015) and image inpainting (XIE; XU; CHEN, 2012), several authors have presented results for natural image denoising that compete with the state-of-the-art approaches. Moreover, some works which also applies deep learning to denoise Monte Carlo renderings have been proposed (BAKO et al. (2017); CHAITANYA et al. (2017)). These methods handle the MC denoising by using a learned denoiser which is able to deliver results better or similar to the state-of-the-art. They have reached great image quality results even when a low sample per pixel image is fed into the denoiser.

1.1 Objective

The aim of this work is develop an end-to-end denoiser using CNNs. The system takes a noisy rendered image and several additional rendered features and generates a noise-free version of the image. Thus, we have implemented one of the recent published neural network architecture to MC path tracer denoising BAKO et al. (2017) and evaluated its results on different rendered scenes.

We also evaluated several loss functions and their effect on visual quality of the denoised image. Based on these evaluations, we tested some modifications to the neural network architecture to evaluate the impact on the quality of the final result. Finally, this work resumes the main steps taken to develop an end-to-end system capable to address the noise present in many different scenes rendered using MC path tracers, it also delivers a framework to generate and augment training data.

1.2 Organization of the document

The present document is organized as follows: chapter 2 discusses the basic concepts of realistic image synthesis, MC denoising and machine learning. Chapter 3 introduces our approach to generate the training and evaluation datasets; Chapter 4 discusses the proposed architecture and the pre-processing and post-processing techniques used to overcome some issues related to MC denoising; Chapter 5 discusses the loss functions we found useful for MC denoising; Chapter 6 shows the results and, chapter 7 presents our conclusions and list some potential directions for future work that we think are promising.

2

Background and Related Work

2.1 Introduction

This section discusses the basic concepts of realistic image synthesis, MC denoising and machine learning. Also, the most relevant contributions regarding these field are presented. For MC denoising, we will present only the *a posteriori* methods which treat the renderer as a black box and leverages spatial filters to obtain a noise-free image.

2.2 Realistic Image Synthesis

Realistic Image Synthesis is the process of creating synthetic images which are indistinguishable from images (such as photographs) of the real world. The recent proposed methods, such as Path Tracing, Photon Mapping, and Metropolis Light Transport, archives these results by exploring the physical nature of light. The physically-based simulation of all scattering is called Global Illumination (GI). The goal of global illumination is to simulate all reflections of light in a model and enable an accurate prediction of the intensity of the light at any point in the model. The input to a GI simulation is the scene geometry description, materials and light sources. The job of the global illumination is to compute accurately how light leaving the light sources interacts with the scene and reaches the camera. The basis for all global illumination algorithms is the rendering Equation 2.3. It can be used to compute the outgoing light intensity at any surface location in the models. The outgoing radiance, L_o, is the sum of the emitted radiance, L_e and the reflected radiance, L_r:

$$L_o(x,\vec{\omega}) = L_e(x,\vec{\omega}) + L_r(x,\vec{\omega}) \tag{2.1}$$

$$L_r(x,\vec{\omega}) = \int_\Omega f_r(x,\vec{\omega}',\vec{\omega}) L_i(x,\vec{\omega}')(\vec{\omega}'\cdot\vec{n}) d\vec{\omega}' \tag{2.2}$$

By rewriting the Equations 2.1 and 2.2, we obtain the rendering equation:

$$L_o(x,\vec{\omega}) = L_e(x,\vec{\omega}) + \int_\Omega f_r(x,\vec{\omega_i}',\vec{\omega}) L_i(x,\vec{\omega_i}')(\vec{\omega_i}'\cdot\vec{n}) d\vec{\omega}' \tag{2.3}$$

where L_o represents the outgoing radiance transmitted from a infinitesimal region around the point x into a infinitesimal cone in the direction $\vec{\omega}$. L_e is the emitted radiance, $\vec{n}$ is the normal direction at point x, Ω is the unit hemisphere centered around $\vec{n}$ containing all possible incoming directions $\vec{\omega}_i'$ and L_i represents the incoming radiance from the direction $\vec{\omega}_i'$. The function f_r is the Bidirectional Reflectance Distribution Function (BSDF) which expresses the relationship between incoming flux and reflected flux, in the end the BSDF captures the material properties of an object at x.

The Equation 2.3 is the rendering equation as it is often used in Monte Carlo rendering algorithms such as Path Tracing. Monte Carlo based techniques are the most general class of global illumination methods. They are a straightforward method to solve the rendering equation. Although standard numerical methods like trapezoidal integration and Gaussian quadrature (WALSTON, 1968) are very effective at solving low-dimensional smooth integrals, their convergence for high dimensional integrals that are common in rendering equation is poor. For instance, the number of samples required in quadrature techniques is exponential in the integral dimensions. In Monte Carlo methods, on the other hand, it is used randomness to evaluate integrals and the number of samples is chosen regardless of the dimensions of the integral dimension. Remarkably, as demonstrated in the Appendix A, these MC methods are unbiased, which means that the convergence to the expected result is always guaranteed. However, it is true that using low sampling to evaluate integrals using MC methods such as Path Tracing results in a very noisy image because the standard error of Monte Carlo integration method is proportional to $\frac{1}{\sqrt{N}}$ where N is the number of samples. Thus, a small number of rays is usually not sufficient to render accurately an image. For a mathematical discussion regarding Monte Carlo integration see Appendix A.

Eliminate the noise from the path traced images is a computational intensive task since it is necessary a huge number of samples for each image pixel to generate a noise-free image as shown of Figure 2.1. This requirement implies that it may take from minutes to hours to generate a single frame. Fortunately, there are many variance reduction techniques available. For example, if the shape of the function to be integrated is known, then it is possible to use importance sampling to concentrate the samples in the important parts of the function. In general, if the samples are obtained from a probability distribution that match other factors of the integral (that is, similar in shape to the integrand) present in the rendering equation (the BSDF or the incoming illumination distribution, for instance), the efficiency is improved and the variance is reduced. Other optimization techniques include stratified sampling (KAJIYA, 1986), and Russian roulette. For a complete discussion on rendering and Monte Carlo path tracing, please refer to PHARR; JAKOB; HUMPHREYS (2016).

(a) 1 spp (b) 4 spp (c) 8 spp

(d) 16 spp (e) 32 spp (f) 64 spp

(g) 128 spp (h) 256 spp

Figure 2.1: Convergence of MC renderings. The expected error of the MC estimate decreases as $\sqrt{N}$, where N is the number of samples per pixel (spp).

Unfortunately, even using the techniques mentioned before, the time required to render a plausible frame (with no noise) still is unpredictable for most of the applications (see the Table 2.1 for more details). The values shown in this table refer to the time for rendering a single frame from the *Kitchen* scene (BITTERLI, 2016) on the Tungsten renderer using a 2,2GHz Intel Core i7. As we can see, for generating a frame with 4096 samples per pixel, for example, it takes more than 24 hours. Thus, other techniques may be investigated in order to solve this slow convergence issue.

Table 2.1: The table shows the time (in minutes) to render one frame of the scene *Kitchen* for different samples per pixel.

Samples Per Pixel (SPP)	Render Time (minutes)
1	1
4	3.5
8	6
16	13
32	26
64	49.5
128	98
256	195
512	264
1024	539
4096	2990

2.3 Spatial Filters for Denoising

In order to overcome the slow convergence of MC renderings, several denoising techniques have been proposed to reduce the noise of rendered pixel colors by leveraging spatial redundancy in images. The key idea behind filtering for denoising is that the values of the pixels of the noisy image are updated using a combination of the pixels of the same nature, otherwise, the filter tends to overblur the image (BOUGHIDA; BOUBEKEUR, 2017). Thus, besides accelerating convergence using the techniques introduced previously (Importance Sampling, Stratified Sampling, and Russian Roulette), one can also filter the rendered images as a post-processing step.

2.3.1 Bilateral Filters

Bilateral filters (TOMASI; MANDUCHI, 1998) compute weights of neighbor pixels by comparing both their locations and their values. These filters are an edge preserving extension to the Gaussian filter. Similarly to the Gaussian convolution, the bilateral filter is also defined as a weighted average of pixels. The difference is that the bilateral filter takes into account the variation of intensities to preserve edges. The bilateral filter, denoted by $BF[.]$, is defined by:

$$BF[I]_p = \frac{1}{W_p} \sum_{q \in S_p} G_{\sigma_s}(\|\mathbf{p}-\mathbf{q}\|) G_{\sigma_r}(I_p - I_q) I_q \tag{2.4}$$

where $\|\mathbf{p}-\mathbf{q}\|$ is the spatial L_2 distance between the pixels p and q, $I_p - I_q$ is the difference between the pixels p and q in color and W_p is a normalization factor:

$$W_p = \sum_{q \in S_p} G_{\sigma_s}(\|\mathbf{p}-\mathbf{q}\|) G_{\sigma_r}(I_p - I_q) \tag{2.5}$$

and $G_\sigma(x)$ denotes the two-dimensional Gaussian kernel:

$$G_\sigma(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x^2}{2\sigma^2}\right) \tag{2.6}$$

G_{σ_s} is a spatial Gaussian that decreases the influence of distant pixels, G_{σ_r} a range Gaussian that decreases the influence of pixels q with an intensity value different from the pixel I_p. One major application of bilateral filtering is denoising. One important observation by LIU et al. (2006) is that adapting the range parameter σ_r to the local noise level in the image yields more satisfying results.

2.3.2 Cross Bilateral Filters

Cross Bilateral Filters (EISEMANN; DURAND (2004); PETSCHNIGG et al. (2004)), also known as Joint Bilateral Filters, integrate more dimensions in the weighting kernels to account for additional pixel information (for example depth). Given an image I, the Cross Bilateral Filter smooths the image I while preserving the edges of a second image E:

$$CBF[I,E]_p = \frac{1}{W_p} \sum_{q \in S_p} G_{\sigma_s}(\|\mathbf{p}-\mathbf{q}\|) G_{\sigma_r}(E_p - E_q) I_q \tag{2.7}$$

where Wp is a normalization factor:

$$W_p = \sum_{q \in S_p} G_{\sigma_s}(\|\mathbf{p}-\mathbf{q}\|) G_{\sigma_r}(E_p - E_q) \tag{2.8}$$

Thus, the range weights are calculated based on the difference of the intensities of the pixels in the image E instead of I.

2.3.3 Non-Local Means

Non-Local Means (NL-Means) (BUADES; COLL; MOREL, 2005) is a bilateral filter which uses patches instead of pixels and removes the spatial component in the weights. The use of patches instead of pointwise values has been proposed to improve the denoising quality and robustness. In a nutshell, NL-means averages neighbors with similar neighborhoods. Thus, the Non-Local Means is defined as:

$$NLMF[I]_p = \frac{1}{W_p} \sum_{q \in S_p} G_{\sigma_r}\left(\frac{\|\wp(p) - \wp(q)\|^2}{k^2}\right) I_q \tag{2.9}$$

where $\wp(p)$ and $\wp(q)$ are patches centred on pixels p and q respectively and k is a constant that can be tuned to set the strength of the filter.

2.3.4 MC denoising using Spatial Filters

When filtering MC renderings we can use additional buffers that the renderer can produce (such as depth map, normal map, and diffuse map) instead of using only the pixel colors (MCCOOL, 1999). Based on this, ROUSSELLE; MANZI; ZWICKER (2013) used per-pixel information about caustics and shadows to apply a Cross Bilateral Filter to tackle complex scenes. However, these additional features can be noisy too, especially when dealing with effects like motion blur and depth of field. Thus, a pre-filtering was applied on these additional inputs. MOON; CARR; YOON (2014) also proposed a scheme to overcome the noise present on features buffers. They proposed the use of a weighted regression which robustly computes a subset of features by ignoring noisy features for higher quality results. The proposed method can efficiently and robustly handle a wide variety of rendering effects. Finally, SEN; DARABI (2012) estimate per-pixel bandwidth for each feature buffer of their Cross Bilateral filtering scheme. This is done to reduce the importance of features that depend on random parameters (noisy features) during filtering to reduce MC noise. They find this functional relationship between features and random parameters by using the concept of mutual information.

2.4 Machine Learning

Machine Learning is the application of algorithms that make sense of data. Mitchell (1997) provides a succinct definition: "A computer program is said to learn from a experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E". In supervised machine learning, the goal is to learn a model from labeled training data that allows us to make predictions about unseen data. One could, for example, try to predict if an email is spam or not spam. Following the Mitchell definition, to predict if an email is spam is the task T. To do that we could train a model on a dataset of emails previously classified as spam or not spam (this dataset is the experience E in the Mitchell definition) to predict whether a new email belongs to either of the two categories. This relationship is typically modeled by a function that estimates the response variable y as a function of the input variables (also known as features) x and tunable parameters w that are adjusted to make the model describe the relationship between the input variables and the prediction accurately:

$$\hat{y} = f(x, w) \tag{2.10}$$

The parameters w are learned from data. One of the key ingredients of machine learning algorithms is the objective function (or loss function) $J(w)$ that is to be minimized during the learning process. For a given dataset, we want to find a w that minimizes this cost function:

$$w = arg\,min_w\, J(w) \tag{2.11}$$

A typical loss function for continuous variables is the quadratic or L_2 loss:

$$J(w) = \frac{1}{2m} \sum_{(x_i, \tilde{y}_i) \in \mathrm{D}_{\text{train}}} (\tilde{y}_i - f(x_i, w))^2 \tag{2.12}$$

where $|\mathrm{D}_{\text{train}}| = m$. The $\frac{1}{2}$ term present in the equation above is useful to facilitate the derivations and do not alter the final result of the optimization.

Another fundamental ingredient of machine learning is the optimization algorithm. The aim of the optimization is to find the weights of the learning algorithm that minimizes the loss function. Gradient descent is a versatile tool for optimizing functions for which the gradient can be computed analytically. It is one of the most popular method to optimize neural networks. The weights to optimize are initialized randomly and iteratively updated by following the rule:

$$w_k = w_{k-1} - \alpha \, \nabla J(w_{k-1}), \forall k \in \mathbb{N} \tag{2.13}$$

The α (also known as learning rate) determines the size of steps to reach a (local) minimum. The vanilla implementation of the gradient descent is described below:

Algorithm 1 Gradient descent vanila implementation

1: $weights \leftarrow randomInitialization()$
2: **while** True **do**
3: $\quad weightsGrad \leftarrow evaluateGradientt(lossFun, data, weights)$
4: $\quad weights \leftarrow weights - learningRate * weightsGrad$ ▷ Update the weights
5: **return** *weights*

Other optimization algorithms have been widely used by the deep learning community to overcome several challenges found when training deep architectures such as: accelerating the convergence, choosing the proper learning rate and learning rate schedules; optimizing highly non-convex functions (very common in neural networks) and avoiding getting trapped in their numerous suboptimal local-minima. In particular, mini batch gradient descent, is often used. It iteratvely applies updates with gradients computed on small subset of the dataset. The number of data points in these subsets is called the batch size. An extension of gradient descent is momentum, which can accelerate the convergence by applying exponential smoothing to the gradient update:

$$v_k = (1 - \alpha) v_{k-1} + \alpha \, \nabla J(w_{k-1})$$

$$w_k = w_{k-1} - v_k \tag{2.14}$$

The momentum method is useful when the optimization is close to a local minimum. The surface of this region usually curves much more steeply in one dimension than in another. Hence,

gradient descent presents an oscillatory behavior, making it difficult for the training process. The momentum term increases for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change directions. As result, we obtain a faster convergence.

Some other useful optimizations algorithms are either variants to momentum or apply other tricks to speed up training. These methods includes: ADAM (KINGMA; BA, 2014), Adadelta (ZEILER, 2012), Adagrad (DUCHI; HAZAN; SINGER, 2011), RMSprop, and Nadam (DOZAT, 2016). The ADAM method is also very popular and it is our method of choice in this work.

To evaluate the abilities of a machine learning method, we must define a quantitative measure of performance. Usually this performance P is specific to the task T. For tasks such as classification (as the case of categorizing an email as spam or not spam), we often measure the accuracy of the model, that is, the proportion of correct results that a classifier achieved. For tasks such regression, we often measure the Mean Absolute Error (MAE) or the Mean Square Error (MSE). In general, it is often difficult to choose a performance measure that corresponds well to the desired behaviour of the system. In a future session we show the main performance measurement that may be applied to MC denoising and the difficulties to choose the performance measurement that best fits to this problem.

Finally, the supervised Machine Learning tasks can be classified either as classification or regression. In a supervised learning task with discrete class labels, such as in the previous spam or not-spam example, is also called a classification task. On the other hand, another subcategory of supervised learning is regression. In this category, the outcome is a continuous value. In this work we deal with a regression task.

2.4.1 Neural Networks

Artificial Neural Networks (ANNs) are a class of models with potentially large numbers of parameters that have shown to be very successful in capturing patterns in complex data. The basic concept behind artificial neural networks was inspired on hypothesis and models of how the human brain works to solve complex problems. Although ANNs have gained a lot of popularity in recent years, early studies of neural networks go back to 1940s when the McCulloch-Pitt neuron was first described.

A neural network is composed of a fundamental structure called neuron. This unit, represented in the Figure 2.2 combines the product of inputs by the weights vector and apply a non-linear activation function:

$$y_i = f(b + \sum_{j=1}^{n} x_j w_{ij}) \tag{2.15}$$

where y_i is the neuron output, w is the neuron weight vector, x is the neuron input and b the bias term. Also, $f(\cdot)$ is the activation function which guarantees that the composition of

several neurons can potentially give a non-linear model. The most popular activation functions are softplus, selu, elu, sigmoid function, rectified linear unit (ReLU) and hyperbolic tangent. The Equation 2.15 can be rewritten considering the bias term as an additional weight w_0 and $x_0 = 1$:

$$y_i = f(\sum_{j=0}^{n} x_j w_{ij}) = f(x \cdot w) \tag{2.16}$$

The main limitation of this type of architecture is that it is only capable of learning linearly separable boundaries, this restricts seriously its applications.

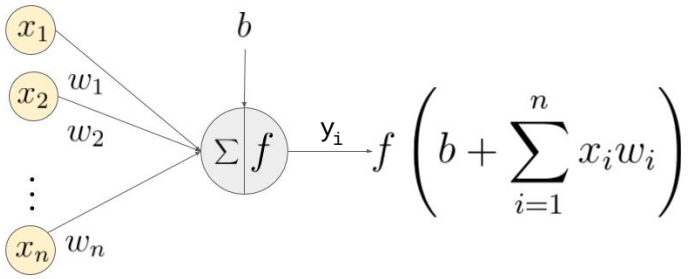


Figure 2.2: The figure illustrates the neuron and its main operations: multiply the input by the neuron weights, combine the result and apply some non-linearity. The neuron is the fundamental unit in neural networks architectures.

We can connect multiple single neurons to obtain a Multilayer Perceptron (MLP) shown in the Figure 2.3. This special type of fully connect network is also known as multilayer feedforward neural network. The advantage of MLP over neural networks with only one layer which we presented before is the fact the MLP is capable of learning both linearly separable and non-linearly decisions boundaries. We can interpret the layers in a neural network as feature extractors. A layer combines the features from the previous layer into 'higher level', more meaningful, features that in the end (in the output layer) enable a more powerful prediction, based on the representation built in the previous layers.

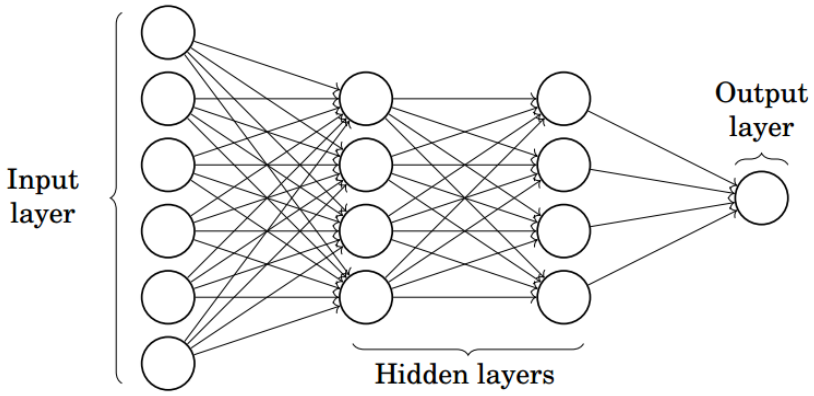


Figure 2.3: A schematics of a MLP with five layers. The MLP depicted has one input layer, two hidden layers, and one output layer.

We can add an arbitrary number of hidden layers to the MLP to create deeper networks architectures. However, the error gradients calculated by the backpropagation algorithm ZHANG (2000), in order to obtain the neurons weights, becomes increasingly small as more layers are added to the network. Hence, the model learning becomes very difficult. This problem is known in the literature as vanishing gradient and several special approaches have been developed in order to help train such deep neural networks (SZEGEDY et al. (2017); HE et al. (2016)).

2.4.2 Convolutional Neural Networks

In recent years, Convolutional Neural Networks (CNNs) (LECUN; BENGIO; HINTON, 2015) have been extremely successful in several practical applications, achieving state-of-the art performance in a diverse range of tasks such as image classification (KRIZHEVSKY; SUTSKEVER; HINTON, 2012), speech processing (VAN DEN OORD et al., 2016) and many others. CNNs consist of a sequence of layers that apply a set of convolution kernels to input from the preceding layer, followed by a non-linear mapping defined as:

$$a_i^l = \sigma\left(\sum_j a_i^{l-1} * k_{i,j}^l + b_i^l\right) \tag{2.17}$$

where $*$ denotes a convolution layer, a_i^l is the ith activation on the lth layer, $k_{i,j}^l$ and b_i^l are the learnable jth 2D convolution kernel and bias term associated with output activation i at level l, respectively, and σ is the activation, typically a rectified linear unit, ReLU, that is defined as $max(0,\cdot)$.

The fundamental difference between a MLP (stack of multiple densely connected layers) and a CNN (stack of multiple convolution layers) is the fact that the dense layers learn global patterns in their input feature space, whereas convolution layers learn local patterns in small 2D windows (the kernels) of the inputs. Thus, CNNs makes neural networks for image processing more tractable by making the connectivity of neurons between two adjacent layers sparse. Therefore, Convolution layers have fewer parameters than fully connected layers.

These key characteristics give CNNs two properties: 1) the patterns they learn are translation invariant. This makes CNNs data efficient, thus CNNs need fewer images to learn good representations. 2) They can learn hierarchical patterns. This means that the firsts convolutions learn small local patterns and the latter convolutions combine these local representations into local objects and finally the last convolutions combine these objects into high-level concepts such as cat, dog, car etc. CNNs have also found applications for a large range of low-level image processing tasks such as denoising (BURGER; SCHULER; HARMELING (2012); JAIN; SEUNG (2009)), inpainting (XIE; XU; CHEN, 2012), superresolution (YANG et al., 2017) and image colorization (LARSSON; MAIRE; SHAKHNAROVICH (2016); IIZUKA; SIMO-SERRA; ISHIKAWA (2016)).

A complete discussion about the basic concepts of CNNs, the last improvements of

CNNs, their recent applications (including image classification, object detection, object tracking, pose estimation, text detection, visual saliency detection, action recognition, scene labeling, speech and natural language processing), the main challenges involved in CNNs and several future research directions, please refer to the recent survey by GU et al. (2017).

2.4.3 Machine Learning Approaches applied to MC denoising

Machine Learning approaches applied to MC denoising have demonstrated huge potential recently. Bitterli et al. employed a first-order model that exploits per-pixel features when relevant and fade out to sample-based filtering when no correlation can be established between the auxiliary features and the pixel color. KALANTARI; BAKO; SEN (2015) observed the Monte Carlo noise present in scenes with many visual effects including motion blur, glossy reflections and global illumination. They built their training dataset containing several images covering these visual effects. The dataset was then used to train a MLP model to predict the proper parameters (the bandwidths) of a spatial filter for generating noise-free images. For each pixel p of the noisy image the neural network outputs the spatial filter parameters. This filter resulting from the parameter choice, when applied to the neighborhood of p, produces the denoised pixel. Besides the noisy color image, the neural network also uses additional buffers extracted from the scene such as depth buffer and normal buffer to gather additional information about the scene composition and then facilitate the neural network training. The main drawback of this approach is that the network learns parameters of an existing filter (a non-local means filter). Although they may produce impressive denoising results, even with a very low number of samples per pixel, it inherits the limitations from the filter used.

CNNs also have attracted the attention thanks to works that tackle the natural image denoising and image super resolution problems (DONG et al. (2014); LEDIG et al. (2017); BURGER; SCHULER; HARMELING (2012)). However, a naive application of convolutional network to MC denoising exposes a range of challenges. First, denoising from only a raw, noisy color buffer causes overblurring since the network cannot distinguish between scene noise and scene detail. Second, the high dynamic range of the image can cause unstable weights during the training, that cause color artifacts in the final image. Moreover, choosing a loss function both coherent with the human visual system and reasonably convex is a hard task. Finally, the training dataset generation demands a manual labor.

The proposed approach is based on a recent work by BAKO et al. (2017). This work use several auxiliary buffers and perform diffuse/specular decomposition to handle previously exposed challenges. The work also overcomes the limitations of KALANTARI; BAKO; SEN (2015) by learning the filter itself and therefore produces better results. CHAITANYA et al. (2017) also applies Deep Learning to denoise Monte Carlo renderings. Their solution uses recurrent connections in a deep autoencoder structure to interactively generate plausible images sequences with global illumination at extremely low sampling budgets (1 spp). Autoencoders

are networks that reconstruct their inputs from an intermediate representation, their autoencoder also removes noise from input. Because of the small sampling budgets available, many images areas have almost only noise, thus they framed the problem as a reconstruction of the final image (rather than denoising). On the other hand, our work focuses more on production-quality renderings instead of interactive renderings with low sample counts.

3

Dataset Construction

3.1 Introduction

Machine learning, most specifically Deep Learning, relies on large datasets in order to get accurate results. One important step when using learned approaches is data acquisition. In the context of machine learning to MC rendering, it is a challenging step. We cannot just corrupt existing images with a particular type of noise and train on these images. We need to render many frames of different scenes with both low and high sample per pixel. Those scenes should cover a wide amount of effects found in path traced scenes such as diffuse reflection, caustics, depth of field, motion blur, glossy reflections and global illumination. Moreover, since our work uses several auxiliary buffers obtained from the scene, such as depth map, normal map, albedo coefficient and specular coefficient, we need a way to access this information for each frame rendered. Thus, the process of scene generation and rendering is very expensive for two main reasons: each scene frame can take hours to be rendered (depending on how many samples per pixel are used to generate the image), and creating different scenes covering many different geometric and optical properties requires considerable manual labor.

This chapter introduces how we tackle the scene generation and augmentation pipeline with both pre-rendering and post-rendering techniques. We also describe the auxiliary buffers stored for each rendered frame and how the data is structured. A typical frame with the additional rendering features is around 100 MB. Table 3.1 gives a summary of our dataset.

Table 3.1: Summary of the dataset developed in this work

	Training Set	Validation Set	Test Set
Number of scenes	7 scenes	2 scenes	3 scenes
Pre-rendering Augmentation	Camera move, field of view, materials (100 images per scene)	No	No
Samples per pixel	32, 4096 spp images	32spp	32 spp
Post-rendering augmentation	Color swapping, shift, mirroring, rotation	No	No

3.2 Scene Generation

Despite the availability of advanced light transport methods, such as Bidirectional Path Tracing (LAFORTUNE; WILLEMS, 1993) and Metropolis Light Transport (VEACH; GUIBAS, 1997), many industrial renderers continue to rely on optimized unidirectional path tracers because they are simple to implement and, compared to bidirectional methods, generate a single (noisy) path integral estimate more quickly. In this work, we used the Tungsten renderer for rendering the scenes to train and evaluate our denoiser. We choose the Tungsten renderer because it is a state of the art Path Tracer able to generate very realistic scenes. Besides that, it supports several optimizations and materials found in modern renderers. It also implements the steps necessary to load a scene and to generate the final image (models loading, texture loading, and image storing). Moreover, the Tungsten is an open-source project and it is possible to modify the source code to extract the auxiliary buffers, a key component of our denoising pipeline. Finally, the Tungsten contains a set of 3D scenes (BITTERLI, 2016) available. Most of these scenes are complex with difficult indirect lighting. In this work, we use extensively these scenes.

The scenes generated are illustrated in Figure 3.1. We generate 1280 x 720 resolution images during the rendering step. These images are chosen to represent a reasonable range of geometric and optical properties. There are many diffuse and specular reflections, indirect illumination, different geometries and different textures. Other effects like motion blur and depth of field are not included in our data rendered by the Tungsten because these effects can be efficiently implemented as a post-processing step and also because they introduce noise in the auxiliary buffers. These effects would be interesting to add in future research.

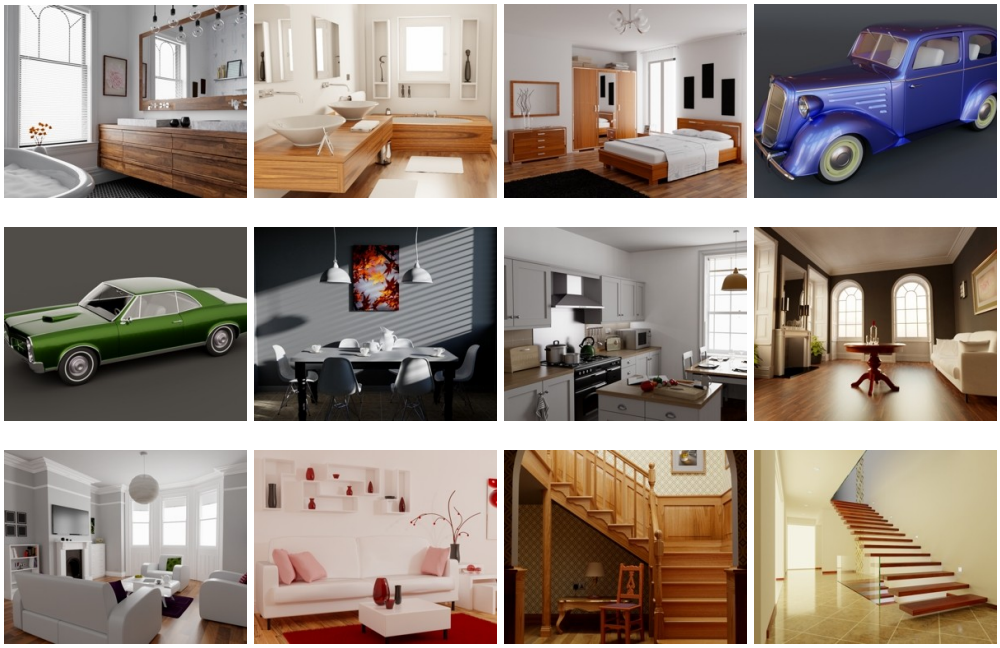


Figure 3.1: Training and Testing Scenes generated using Tungsten.

3.3 Pre-rendering Augmentation

To train a deep neural network, a dataset consisting of only 7 unique images is very limited. Since there would be a limited range of colors and shapes, the generalization behavior of the learned model would be bad. Unfortunately, creating the scenes is an expensive task. It involves several human labor hours to create and position the geometries, textures and light sources in the scene plus the additional time to render low samples per pixel and high samples per pixel (the ground truth) scenes.

To overcome this problem, several data augmentation techniques have been proposed to increase the number of training data and so reduce the model overfitting. However, in this work we are limited to just some techniques. Some augmentation techniques either are not useful to MC denoising or can severally change the noise properties and make it difficult to the learning algorithm extract the important aspects of the MC noise. In this work, in the pre-rendering step, we applied random perturbations to the scenes such as changing lighting, textures, materials, roughness parameters, and camera parameters. In this way, we can generate many visually different views of the same geometry. The scenes augmentation perturbations include:

- Camera move: we generate several images from the same scene by moving the camera to get several point of views.
- Field of view: we generate several images from the same scene by uniformly sampling the field of view parameter of the camera.
- Materials: we generated several images from the same scene by randomly altering the materials (texture, roughness etc.) of the scene geometries.

This approach generates hundreds of perturbations from each scene. Then, we manually prune scenes that are not sensible. As result, we generate several training images. A selection of some generated perturbations is displayed in Figure 3.2. New types of perturbations would be an interesting investigation to add in future research.

Figure 3.2: Some of the images present in our dataset. These images are produced by using different types of perturbations. Each row corresponds to the same scene.

3.4 Post-rendering Augmentation

To artificially increase the size of our training set, we apply a post-rendering process which transforms each image rendered in the previous step pseudo-randomly. The training images are submitted to the following operations:

- Color swapping: the RGB channels of color, color variance, albedo and albedo variance are randomly permuted.
- Shift: all features are randomly translated vertically or horizontally.
- Mirroring: all features are randomly mirrored horizontally or not.
- Rotation: all features are randomly rotated up to a limit of 180 degrees.

The two last operations are relevant when there are no assumptions of symmetry in the images, this is the case of the problem that we are dealing with.

The network input consists of several patches extracted from the rendered images. The Figure 3.3 shows some patches after the post-rendering augmentation.

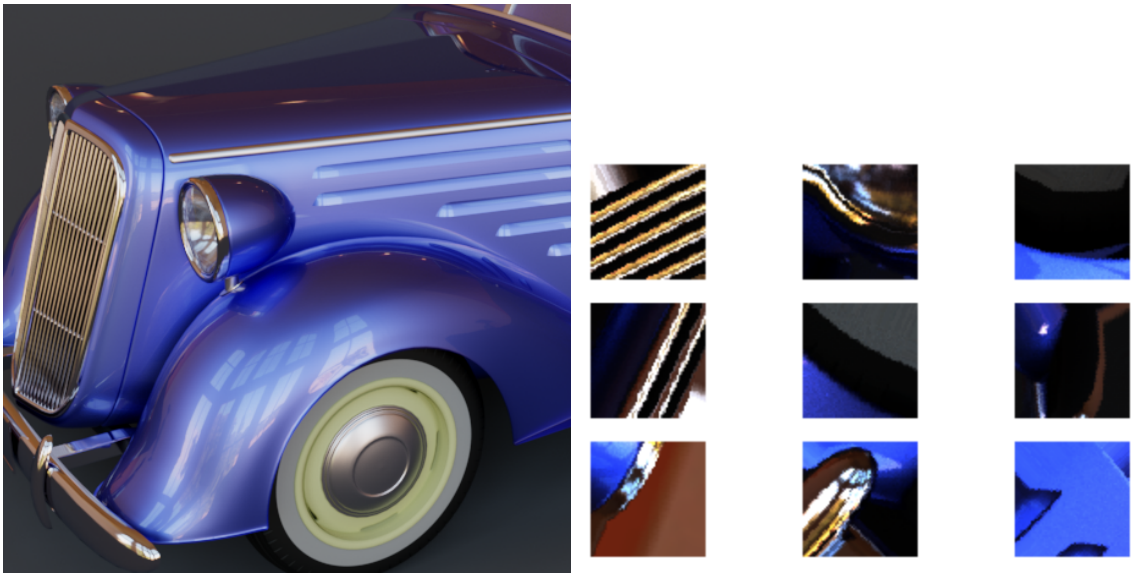


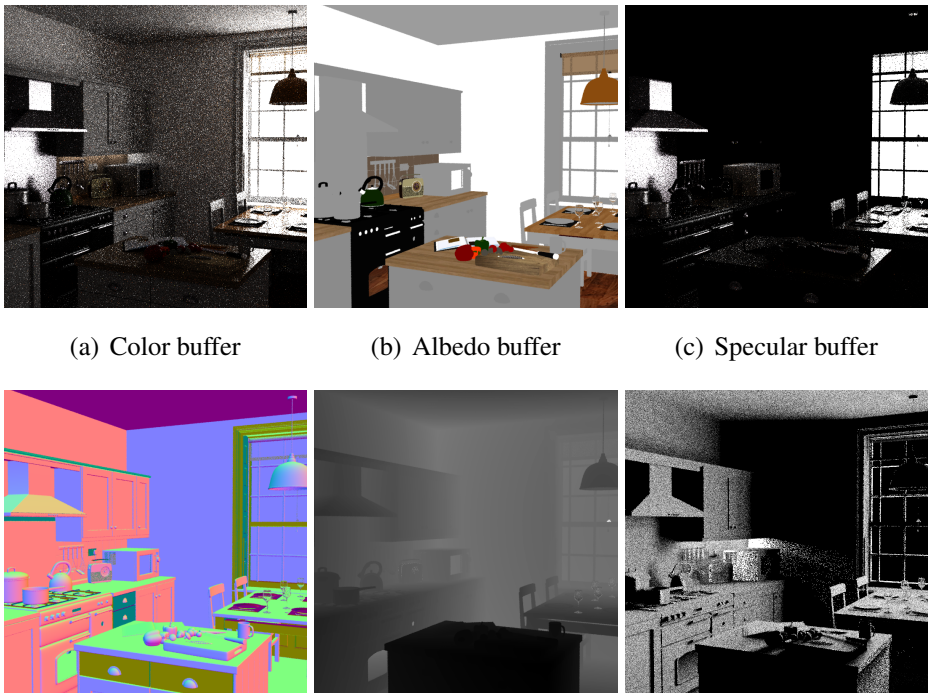
Figure 3.3: Original image and several pacthes extracted from the image. The patches were submitted to the post-rendering pipeline. For each patch it was applied randomly one of the following operations: color swapping, shift, mirroring or rotation.

3.5 Auxiliary Buffers

Our technique relies on several additional buffers (also known as scenes features) such as depth map, normal map, albedo map and specular map to gather rich information about the scene geometry and materials. These maps are less susceptible to noise when compared to the RGB color map, therefore we can leverages this additional maps to provide to the denoiser network more useful features that facilitate learning and convergence.

In our approach we use a specific set of 5 features. For each scene, generated in the scene pre-processing step, the renderer outputs a deep image, which consists of the noisy RGB image plus 5 additional buffers. A visual overview of the features is given in Figure 3.4. The features are:

- Color (3 channels): the noisy RGB image that should be denoised;
- Albedo (3 channels): the albedo map captures texture colors, without lighting applied to them;
- Specular (3 channels): the specular map captures the specular and glossy reflections;
- Normal (3 channels): normal direction at each pixel;
- Depth (1 channel): the distance from the camera to ray's first intersection for each pixel;
- Visibility (1 channels): at the first intersection, a light source is sampled. If there is object between the light source and the intersection point, 1 is sampled, otherwise 0 is sampled.



(a) Color buffer (b) Albedo buffer (c) Specular buffer

(d) Normal buffer (e) Depth buffer (f) Visibility buffer

Figure 3.4: Visual overview of the features buffers outputted by the rendered and used in our solution

All of the described features are means of samples. An estimated variance for this mean is also provided as separate buffers. We convert variance of three channels to a single channel by computing its mean. Thus, we have, for example, two channels for the color variance (for diffuse and specular).

3.6 Reference Images

As we mentioned before, rendering reference images (high samples per pixel) is expensive. It can takes hours to render only one image, depending the scene complexity. In this work, we used 4096 samples per pixel references images rendered using the Tungsten renderer. Some images are shown in the Figure 3.5. Although, some of these images still present some noise artifacts, we found this amount of samples per pixel enough in order to train our model and generate plausible images.

Figure 3.5: Some ground truth images that are present in our dataset.

4

Deep Convolutional Denoising

4.1 Introduction

Our MC denoising technique is a data-driven method that learns a mapping from noisy input image to a noise-free output image based on a large number of training pairs, each one consisting of an example multichannel input (containing the noisy color buffer and many additional feature buffers) and the desired output (i.e. training target). This mapping between noisy input image and noise-free output is modeled using a deep Convolutional Neural Network (CNN). By joining many layers together with activations functions, CNNs are able to learn highly nonlinear mappings of the input features, which are important for obtaining high-quality outputs. For this reason, CNNs are well suited for the denoising task and have indeed been used for traditional image denoising (XIE; XU; CHEN, 2012).

In this section will present our proposed framework for denoising MC images. We first focus on the filtering core of the denoiser (the network architecture and the reconstruction filter) and later describe data decomposition and preprocessing and postprocessing steps that are specific to the problem of MC denoising. The Figure 4.1 illustrates the proposed denoising framework.

4.2 Network Architecture

Our network architecture is based on the kernel-predicting architecture by BAKO et al. (2017) (KPCN). They use a fully convolutional neural network (FCNN) with no fully connected layers to keep the number of parameters reasonably low. This reduces the danger of overfitting and speeds up both training and inference. For all hidden layers in our architecture, we use rectified linear unit (ReLU) activations, $f^l(a) = max(0,a)$, except for the last layer, L, where $f^L(a) = a$. The ReLU activation function has shown excellent performance in recent neural networks architectures. In fact, KRIZHEVSKY; SUTSKEVER; HINTON (2012) shows that ReLU activation greatly accelerates the convergence of stochastic gradient descent compared to the sigmoid/tanh functions. They argue that this occurs because of the linear, non-saturating form of the ReLU activation function.

As discussed early, the weights and biases of the neural network are the trainable

parameters for layers. The dimensions of the weights in each layer (convolutional kernel and biases) and the number of hidden layers are fixed before the training. We noted that after a certain point, the training loss decreases only marginally while the computational cost keeps increasing as we increase the number of hidden layers. Thus, in our implementation, we use nine hidden layers (ten total convolutions, so L = 10) with 100 convolutional kernels of 5x5 in each layer. In a future section, we present an analysis that justifies the use of nine hidden layers. The weights for the network were initialized using the Xavier method (GLOROT; BENGIO, 2010). We implemented our neural network using the TensorFlow framework (ABADI et al., 2016). We optimized the network using the ADAM optimizer with an initial learning rate of 10^{-3}.

Instead of directly outputting a denoised pixel, c_p , the final layer of the network outputs a kernel of scalar weights that is applied to the noisy neighborhood of p to produce c_p. Thus, for each input image pixel, the neural network outputs a k x k filter $\mathcal{N}(p)$. Note that the k parameter is chosen before training and the same filter is applied to each RGB color channel. An additional normalization step is also employed by including a softmax activation function on the network output. Suppose that the network output is $z_p^L \in \mathbb{R}^{k\times k}$, we compute the final normalized filter as:

$$w_{pq} = \frac{exp([z_p^L]_q)}{\sum_{q' \in \mathcal{N}(p)} exp([z_p^L]_{q'})} \tag{4.1}$$

This enforces that $0 \leq w_{pq} \leq 1$ and $\sum_{q'\in\mathcal{N}(p)} w_{pq} = 1$. It brings some benefits such as: avoid color artifacts because the final pixel color always lies in the convex hull of the neighbor pixels; and ensures the gradients of the error with respect to the filter weight are well behaved avoiding oscillatory changes to the network parameters (weights and bias) during the training.

BAKO et al. (2017) show in detail that predicting the filter to be applied in the input noisy image is capable to deliver similar error rates to directly outputting the final denoised image. However, predicting the filter may take the network to converge 5-6x faster. Because of the faster convergence, we prefer to use the kernel predicting architecture for all results and analyses. In our implementation, we used an output kernel with size k = 21 for each pixel of the input image.

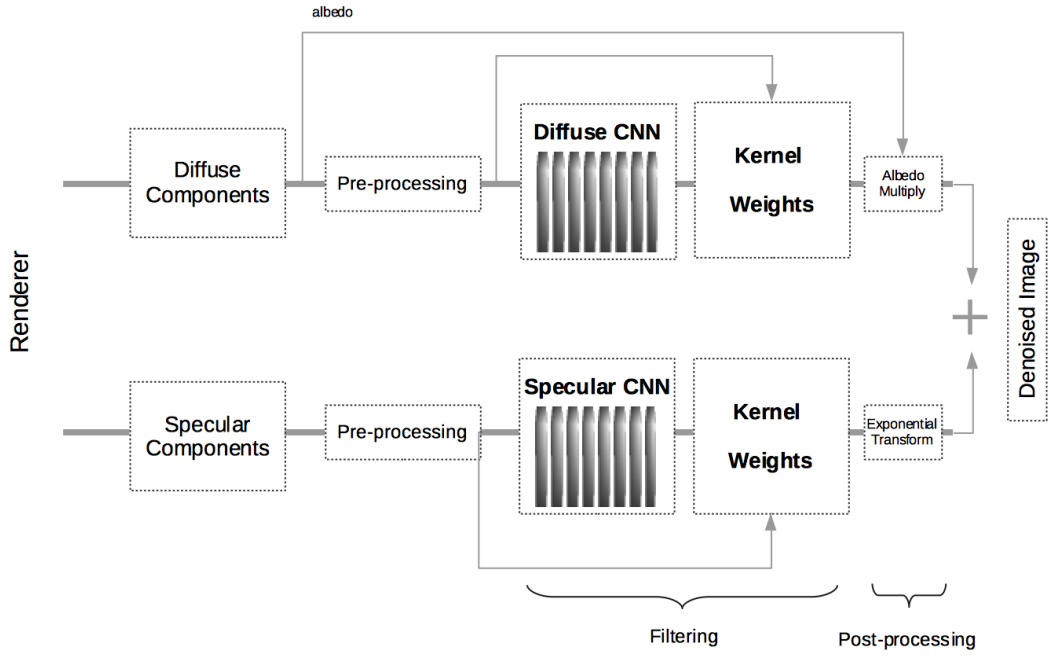


Figure 4.1: Scheme of our denoising framework. We start pre-processing the diffuse and specular components, and then feed the data to two separated neural networks that outputs the kernel for filtering each component. The filtered (noise-free) image is then post-processed and finally combined to obtain the final denoised image.

4.3 Diffuse/Specular Decomposition

The various components of the image have different noise characteristics, which can make it difficult the denoising operation. Because of that, BAKO et al. (2017) propose decomposing the image into diffuse and specular components. These components are independently preprocessed, filtered, and post-processed, before recombining them to obtain the final image, as shown in the Figure 4.1.

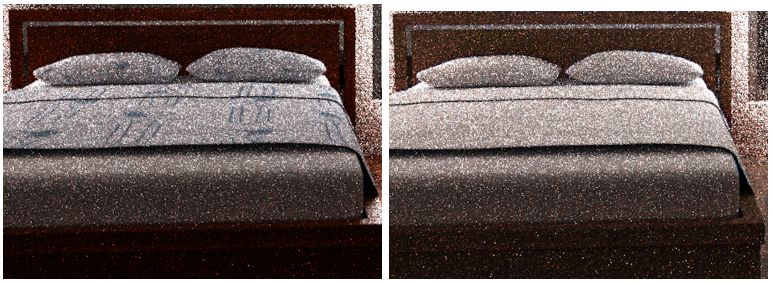
4.3.1 Diffuse Component Preprocessing

We simplify the input diffuse color (the outgoing radiance due to diffuse reflection) by demodulating, in the preprocessing step, the noisy RGB diffuse buffer by the albedo of the directly visible material (ZIMMER et al., 2015):

$$\tilde{c}_{diffuse} = c_{diffuse} \oslash (f_{albedo} + \varepsilon) \tag{4.2}$$

where $\oslash$ is the element-wise division and ε is an small constant to avoid division by zero.

By using this untextured buffer we remove most of the texture complexity from the noisy image, significantly facilitating training and reducing the required network capacity. The Figure 4.2 compares two images before and after demodulating the diffuse buffer by the albedo buffer.



(a) Image before demodulating the diffuse buffer by the albedo buffer.

(b) Image after demodulating the diffuse buffer by the albedo buffer.

Figure 4.2: Comparison between two images before and after demodulating the diffuse buffer by the albedo buffer. Note that after the demodulation the texture details are removed from image.

4.3.2 Specular Component Preprocessing

Denoising the specular component is a challenging problem due to the high dynamic range (HDR) of specular and glossy reflections. The large variations of the pixel amplitude in the specular components make the specular neural network optimization highly unstable. To overcome this problem, we apply a log transform to each color channel of the specular buffer:

$$\tilde{c}_{specular} = log(1 + c_{specular}) \tag{4.3}$$

This transformation can significantly reduce the range of color values and improves results, avoiding artifacts in regions with high dynamic range.

After the two components have been denoised separately, we apply the inverse of the preprocessing transform to the reconstructed output of each network and compute the final denoised image:

$$\hat{c} = \hat{c}_{diffuse} \odot (f_{albedo} + \varepsilon) + exp(\hat{c}_{specular}) + 1 \tag{4.4}$$

where $\odot$ is the element-wise product.

To train the system, we pre-train the specular and diffuse networks separately on the specular and diffuse reference images. After that, we apply the Equation 4.4 and fine-tune the complete architecture by minimizing the error of the final image for additional iterations.

4.4 Network Training

After generating the images using the pipeline described in the chapter 3 and preprocess the specular and diffuse components of the image, we split the images into 65 x 65 patches that are shuffled and used to train the network. Each patch consists of the noisy color image and the corresponding feature buffers. Patches of which most of the pixels present the same color are removed from the dataset. It ensures that we are keeping in our dataset patches very noisy.

All the patches are stored as Tensorflow tensors using the .tfrecord file format. The .tfrecord holds the data in a Protocol Buffer, which have several important features: minimal-size binary strings when serialized, efficient serialization, and efficient interface implementations in multiple languages. The .tfrecord is handy when using Tensorflow to implement the neural network architecture because it facilitates loading and decoding the data and feed into the computation graph.

The specular and diffuse networks are trained independently using L1 loss function. Each network is pre-trained for approximately 600K interactions on a Nvidia GTX 1080 TI. After that, the system is fine-tuned for more 400K iterations using SSIM error metric.

5

Loss Functions

5.1 Introduction

A loss function defines how the error between network output and training targets is computed during training. In our problem of MC denoising, the loss function should approximate a hypothetical perfect human dissimilarity metric well. That is, we need to find a loss function $l(x,y)$, where x is the predicted color and y is the desired output, that describes the human similarity perception. We also need a reasonably convex loss function. Convex loss function will make the neural network training easier avoiding getting trapped in a suboptimal local-minima result. As we mentioned earlier, finding a good loss function is not a simple task. The two criteria mentioned can be contradicting, also it is not straightforward to assess which loss function will give the best result, that is, will give the less perceptual error.

This sections aims to discuss several loss functions we found useful for MC image denoising task.

5.2 Manhattan Distance

The Manhattan distance, also known as L1 loss function, provides a good image metric that is tolerant to outliers. This function is given by the following equation:

$$L1(y,\hat{y}) = \frac{1}{m}\sum_{i=0}^{m} |y_i - \hat{y}_i| \tag{5.1}$$

where y and $\hat{y}$ corresponds, respectively, to the output pixel color and the ground truth value and m is the size of the train dataset.

5.3 Mean Square Error (MSE)

Mean Square error, also known as L2 loss function, is an well known loss function and it is commonly used in linear regression problems. It is also commonly used in image restauration. The L2 equation is given by:

$$MSE(y,\hat{y}) = \frac{1}{2m}\sum_{i=0}^{m}(y_i - \hat{y}_i)^2 \tag{5.2}$$

where y and $\hat{y}$ corresponds, respectively, to the output pixel color and the ground truth value. The $\frac{1}{2}$ term present in the equation above is useful to facilitate the derivations and do not alter the final result of the optimization.

The L2 loss suffer from high sensitivity to outliers. The error blows up quadratically as the distance between two colors increase. Single pixels that are very difficult to denoise and have high error will attract all of the network's "focus", ignoring the other regions that are easier to get right. It is known that using L1 loss instead of L2 can reduce the splotchy artifacts from denoised images (ZHAO et al., 2015).

An alternative to MSE in denoising is the Mean Relative Squared Error (MrSE). Large errors are often made in very bright regions in the image, but they are visually less severe than the same error in a dark region. To correct for this, MrSE divides the squared color difference by the squared ground truth color:

$$MrSE(y,\hat{y}) = \frac{1}{2m}\sum_{i=0}^{m}\frac{(y_i - \hat{y}_i)^2}{\hat{y}_i^2 + \varepsilon} \tag{5.3}$$

where the epsilon term is a small constant to avoid division by zero.

BAKO et al. (2017) introduced to the field of denoising the relative L1 loss function given by equation:

$$RelL1(y,\hat{y}) = \frac{1}{m}\sum_{i=0}^{m}\frac{|y_i - \hat{y}_i|}{|\hat{y}_i| + \varepsilon} \tag{5.4}$$

5.4 Structural similarity (SSIM)

The SSIM index (WANG et al., 2004) is a method for predicting the perceived quality of digital television and cinematic pictures. SSIM is used for measuring the similarity between two images. The index between two images is calculated by moving a window through the two images that we want to compare. Thus, the measure of similarity between these two windows x and y of common size N×N is given by:

$$SSIM(X,Y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{5.5}$$

In the equation μ_x stands for a mean, σ_x for the standard deviation, σ_{xy} for the covariance between X and Y and, finally, c_1 and c_2 are two constants that aims to stabilize the division. The SSIM values lies between -1 and 1, where 1 indicates perfect correspondence. To use SSIM as a loss function, we use:

$$l_{ssim}(x,y) = 1 - SSIM(x,y) \tag{5.6}$$

This metric generally corresponds better to human perceptual similarity than other measures presented so far. However, SSIM is also more difficult to optimize (BAKO et al., 2017). An approach that is commonly used is pre-train the neural network using a more convex loss function such as L1, and then fine tune the model using the SSIM loss.

We found that the L1 loss function is more stable and easy to optimize than L2 and relative L2 functions, as we will show in the Results session. Because of that, we used extensively this loss function in this work.

New types of loss functions and modifications of the functions present earlier would be an interesting investigation to add in future research.

6

Results and Discussion

6.1 Introduction

As already mentioned in the chapter 1, the objective of this work is to develop an end-to-end deep learning framework to overcome the noise present in images rendered using MC path tracing. To do this, we have created a pipeline for data generation and augmentation. Also, we have implemented one of the recently published neural network architecture to MC path tracer denoising and evaluated its results on different rendered images. Finally, we studied and evaluated several loss functions and their effect on visual quality of the denoised image and, based on those evaluations, we tested some modifications to the neural network architecture to evaluate the impact on its quality.

In this chapter, we will discuss the results presented by our framework. Remarkably, in the session 6.2 we show how the loss functions presented in the chapter 5 perform on the network training, in the session 6.3 we study the convergence behaviour of our network by varying its layer count. Finally, in the session 6.4 we highlight the results obtained by our neural network implementation. Please refer to previous chapters to obtain deep information about the denoising architecture and the data generation pipeline. The network development was presented in the chapter 4. Besides that, the results obtained by the data generation pipeline was explored in the chapter 3.

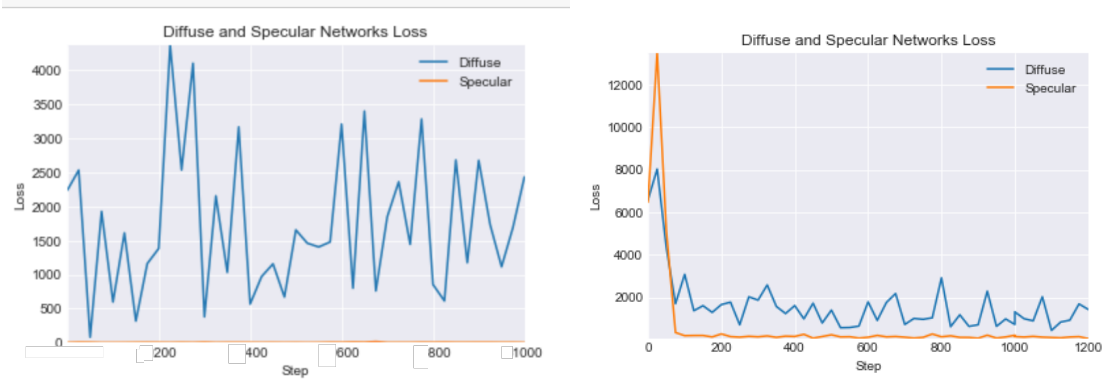
In all experiments we use the Adam optimizer (KINGMA; BA, 2014) applied with batches of 32, patches of size 64 x 64 with a learning rate that decays by a factor 0.96.

6.2 Trained loss

We evaluated multiple loss functions as training loss for our networks. In Section 5, we identified that a good loss function should both correspond well to human perceptual quality and be easy to optimize. As observed early, the SSIM is difficult to optimize. Moreover, ZHAO et al. (2015) recommends using this loss function to fine tune the network trained using another loss function. With this in mind, we evaluate MSE and L1 as candidates for our optimization procedure.

We first evaluated the MSE loss function with a learning rate of 10^{-3}. The specular

network converged easily, however, the diffuse network presented an instability that made the convergence difficult as can be observed in the Figure 6.1(a). By changing the learning rate value, we found out that the reason for the instability was the value too high of the learning rate. As we can see in the figure 6.1(b), the decrease of the learning rate parameter to 10^{-5} improved the neural network convergence.



(a) Learning rate = 10^{-3}

(b) Learning rate = 10^{-5}

Figure 6.1: Comparison of different learning rates. The networks were trained using MSE loss function. We first trained the model using a learning rate of 10^{-3}. We observed that the learning rate was too high resulting in a unstable training (left figure). By changing the learning rate value, we greatly improved the neural network convergence, as we can see in the figure on the right.

We also evaluated the L1 loss with a learning rate of 10^{-3}. As the Figure 6.2 shows, the L1 function arise as good candidates for training the denoiser. Its merit can be explained in the sense that this loss function is more tolerable to outliers very common in noisy images. Also, it does not put too much emphasis on single pixels. Thus, the network learns from all pixels at all times. Compared to the MSE loss function, the L1 converges faster and the training process is smoother. Because of that, the L1 loss is our choice to training the denoiser.

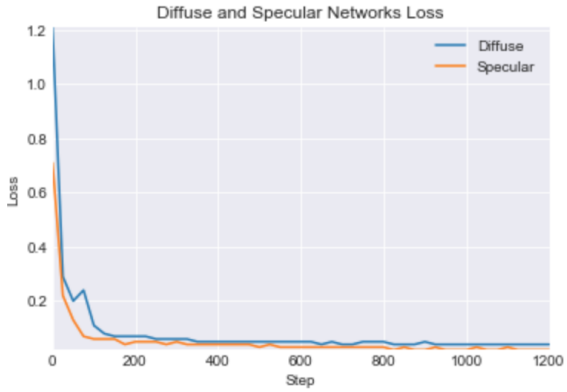


Figure 6.2: We trained the model using a learning rate of 10^{-3} and the L1 loss function. We can observe that this function is a good candidates for training the denoiser. Compared to the MSE function, the L1 converges faster and the training process is more stable.

6.3 Network Design Analysis

In this session, we study the convergence behavior of our network by varying its layer number. Since the space of potential combinations of components is exponential and training a single model takes a long time, we are naturally unable to evaluate all configurations. For this reason, we only evaluated a few possible combinations of the number of layers (see the Table 6.1). Evaluating other possible combinations and network architectures would be an interesting investigation to add in a future research.

Table 6.1: We evaluate the convergence of our network by varying its layer count. We denoted the convolutional layer parameters as "conv(receptive field size)-number of channels", the same notation used by SIMONYAN; ZISSERMAN (2014). In all architectures we use the ReLU activation function, the Adam optimizer and L1 loss function.

Small	**Large 1**	Large 2
conv5-100	conv5-100	conv5-100
conv5-100	conv5-100	conv5-100
conv5-64*64	conv5-100	conv5-100
	conv5-100	conv5-100
	conv5-100	conv5-100
	conv5-100	conv5-100
	conv5-100	conv5-100
	conv5-100	conv5-100
	conv5-100	conv5-100
	conv5-64*64	conv5-100
		conv5-100
		conv5-100
		conv5-64*64

Analyzing the training result of different architectures (outlined in Table 6.1) we observe that after a certain point the loss decreases only marginally while the computation cost keeps increasing, as shown in Figure 6.3. Although the recent evidence that network depth is a crucial ingredient for their success (SIMONYAN; ZISSERMAN (2014); SZEGEDY et al. (2015)), training very deep neural networks become very challenging, leading research on initialization schemes (HE et al. (2016); GLOROT; BENGIO (2010)) and techniques of training networks in multiple stages (ROMERO et al., 2014). HE et al. (2016) observed that training error, and thus test error, can even increase as more layers are stacked. They presented a residual learning framework to ease the training of deeper networks. This architecture can be explored to reduce the error rates of our denoiser.

In this work, we selected the smallest model that gives good results (the model **Large 1**). In total, there are 10 layers in our network with 100 channels of 5x5 in each layer and ReLU

activation function, except the last layer that uses linear activation function and has 64 * 64 channels. Note that 64 * 64 is the size of the filter that will we used in further steps as we described in the chapter 4.

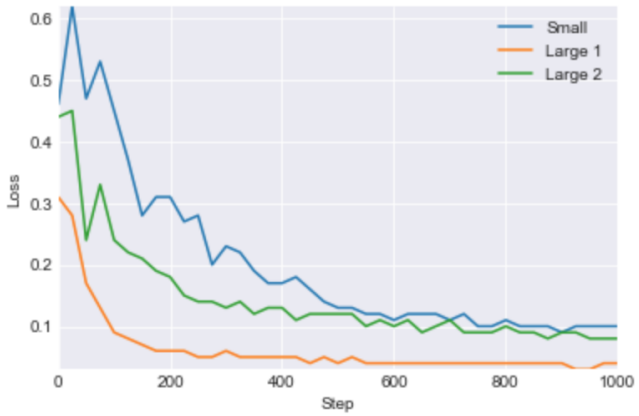
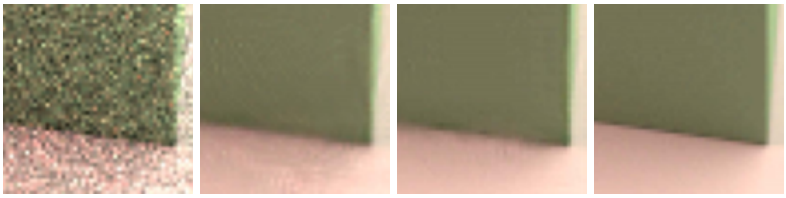


Figure 6.3: Training loss convergence plots for networks trained with different number of layers and using L1 loss function.

6.4 MC images denoising

The previous sections have singled out several of our proposed building blocks and evaluated their contributions to the speed of convergence and final quality. In this section, we evaluate the result of our MC denoiser on different rendered images. To evaluate these images we used the model **Large 1**. It was trained with Adam optimizer and L1 loss with an initial learning rate of 10^{-3} and a learning rate decay of 0.96 every 100000 iterations.

The Figure 6.4 shows the denoising performance of our trained model when applied to a noisy input image (Figure 6.6(a)). We observe a great improvement both visually and in the SSIM metric as well (Figure 6.6(b)). Compared to KPCN (BAKO et al., 2017) (Figure 6.6(c)) and to the reference image (Figure 6.6(d)), we note that our model generates a slightly blurred image. We believe that it happens because we used a smaller and less diverse dataset than BAKO et al. (2017).



(a) Input (SSIM = 0.3308) (b) Our (SSIM = 0.9388) (c) KPCN (SSIM = 0.9669) (d) Reference (SSIM = 1)

Figure 6.4: Comparison of our denoiser performance to our baseline.

Furthermore, as we can observe in the Figure 6.5 and in the Figure 6.6, both denoisers

may wrongly remove important scenes details from the images. For instance, in the Figure 6.5, both the denoisers slightly removes the shadows of the image, however, our framework is more susceptible to this flaw. Moreover, in the 6.6 our denoiser produces some undesirable patterns in the image and removes some fine details of the door, whereas the KPCN filter does note present these problems. The Table 6.2 overviews the SSIM metric error for both scenes.

We would add that the issues observed previously are probably explained by the restricted dataset, since that our architecture is based on the BAKO et al. (2017) architecture. Thus, these problems would be diminished if more scenes and images would be added for training. Besides that, both frameworks wrongly remove the shadows and fine details in some images. Thus, we believe that there is room for improvements on the network architecture and on the dataset generation. Investigating different network architectures may yield improved denoising performance.

(a) Input (b) Our (c) KPCN (d) Reference

Figure 6.5: Comparison of our denoiser performance on the *Modern Living Room* scene. As we can see, both the denoisers wrongly removes the shadows in the image, however our denoiser suffers more with this issue.

(a) Input (b) Our (c) KPCN (d) Reference

Figure 6.6: Comparison of our denoiser performance on the *White Room* scene. As we can see, in this case, our denoiser produces some undesirable patterns in the image and removes fine details of the door. On the other hand, the KPCN filters do not suffer from these stranger patterns at all but we see that the KPCN also removes some details from the image.

It is important to note that the input color image is very noisy and in some case, it is very difficult to distinguish between the scene details and the noise. However, our technique uses several additional input buffers that help the denoising process. These buffers, as we already mentioned in previous chapters, in general, do not suffer too much with noise as the color image

and are useful in the denoising process. These buffers also avoid that the denoiser deliver an over blurred image.

Table 6.2: Error statistics for images from figure 6.5 and figure 6.6

Filter	Modern Living Room (SSIM)	White Room (SSIM)
KPCN	0.9205	0.90858
Our	0.88828	0.62718

We conclude that the end-to-end-learned denoiser for Monte Carlo renderings can deliver good performance even using smaller and less diverse dataset. However, because of these restrictions in the training dataset, the denoiser generates images slightly blurred.

7

Conclusion and Future Works

In this work we have proposed an end-to-end framework for denoising MC rendered images. In order to do this, we have implemented one of the recent published neural network reference to the MC path tracer denoising and evaluating the performance on different rendered scenes. We explored various layers numbers/size and kernel sizes to find settings that work better. We also evaluated several loss functions and compared the final image quality and the training performance in order to choose the loss function capable to deliver the best images and the most stable training performance. Moreover, we developed a framework for generation and augmenting the training dataset.

Despite the promising result in the test scenes, we observed some limitations on the developed denoising framework. Remarkably, it loses some important scene details such as shadows and edges. Even the work by BAKO et al. (2017), which leverages a wider range of images to train the neural network, was not capable of preserving some shadows and fine details of the geometry of the objects. In our case, even though our data generation pipeline was capable to generate more images, the space limitation in our computer hard disk made it impossible training our model on a larger dataset. This observation implies that there is a room for improvements of the proposed model.

The first possible investigation is to include new feature buffers in the input data such as: distance from the primary and secondary ray to the camera and the color buffer in different recursion levels. These features may contain new information regarding the scene geometry and composition, ultimately facilitating the training convergence. Another possible investigation is the neural network architecture. Different architectures and concepts may also yield improved performance. Some generative models, such as varitional autoencoders and generative adversarial have shown great performance for natural image denoising. Recently, CHAITANYA et al. (2017) have applied autoencoders for denoising MC rendered images and showed promissing results. However, a deeper study on applications of these models to MC denoising would be an interesting avenue for future research.

Moreover, other potential topic to investigate is the choice of the loss function. As can be observed, perceptually important features of the image are not well captured by the loss functions presented since that some details in the denoised image are lost. Thus, new loss functions may

increase the model quality. For example, loss functions that penalize the differences in fine details, such as edges, can be explored. Also, we could find useful to combine different loss functions to capture different aspects of the scene. Finally, we demonstrated results for denoising only a single image at a time, however it would be great if we could tackle animated scenes. Animated scenes are present in games, movies, virtual reality and so on, thus having a system capable to generate noise-free animations would benefit these industries. However, this extension is non-trivial since it is necessary develop an architecture capable to guarantee the temporal coherence between the frames.

Compared to the work by BAKO et al. (2017), our solution presents compatible results on validation-scenes that are similar to the training data, as shown. However, to further improve the quality of our denoiser, we should include more images in the training data. We observed that the limited dataset was not enough to generate the same results. Our denoiser removes some details of the image and generate slightly blurred results. Also, to denoise data with effects such as motion blur and depth of field, these effects have to be included in the training data. We believe there is a huge potential in Machine Learning for Computer Graphics, most specifically for denoising MC rendered images.

References

ABADI, M. et al. TensorFlow: a system for large-scale machine learning. In: OSDI. **Anais...** [S.l.: s.n.], 2016. v.16, p.265–283.

BAKO, S. et al. Kernel-predicting convolutional networks for denoising Monte Carlo renderings. **ACM Transactions on Graphics (TOG)**, [S.l.], v.36, n.4, p.97, 2017.

BITTERLI, B. **Rendering resources**. https://benedikt-bitterli.me/resources/.

BITTERLI, B. et al. Nonlinearly Weighted First-order Regression for Denoising Monte Carlo Renderings. In: COMPUTER GRAPHICS FORUM. **Anais...** [S.l.: s.n.], 2016. v.35, n.4, p.107–117.

BOUGHIDA, M.; BOUBEKEUR, T. Bayesian Collaborative Denoising for Monte Carlo Rendering. In: COMPUTER GRAPHICS FORUM. **Anais...** [S.l.: s.n.], 2017. v.36, n.4, p.137–153.

BRUNA, J.; SPRECHMANN, P.; LECUN, Y. Super-resolution with deep convolutional sufficient statistics. **arXiv preprint arXiv:1511.05666**, [S.l.], 2015.

BUADES, A.; COLL, B.; MOREL, J.-M. A non-local algorithm for image denoising. In: COMPUTER VISION AND PATTERN RECOGNITION, 2005. CVPR 2005. IEEE COMPUTER SOCIETY CONFERENCE ON. **Anais...** [S.l.: s.n.], 2005. v.2, p.60–65.

BURGER, H. C.; SCHULER, C. J.; HARMELING, S. Image denoising: can plain neural networks compete with bm3d? In: COMPUTER VISION AND PATTERN RECOGNITION (CVPR), 2012 IEEE CONFERENCE ON COMPUTER VISION. **Anais...** [S.l.: s.n.], 2012. p.2392–2399.

CHAITANYA, C. R. et al. Interactive Reconstruction of Noisy Monte Carlo Image Sequences using a Recurrent Autoencoder. **ACM Trans. Graph.(Proc. SIGGRAPH)**, [S.l.], 2017.

DONG, C. et al. Learning a deep convolutional network for image super-resolution. In: EUROPEAN CONFERENCE ON COMPUTER VISION. **Anais...** [S.l.: s.n.], 2014. p.184–199.

DOZAT, T. Incorporating nesterov momentum into adam. , [S.l.], 2016.

DUCHI, J.; HAZAN, E.; SINGER, Y. Adaptive subgradient methods for online learning and stochastic optimization. **Journal of Machine Learning Research**, [S.l.], v.12, n.Jul, p.2121–2159, 2011.

EISEMANN, E.; DURAND, F. Flash photography enhancement via intrinsic relighting. In: ACM TRANSACTIONS ON GRAPHICS (TOG). **Anais...** [S.l.: s.n.], 2004. v.23, n.3, p.673–678.

GEORGIEV, I. et al. Light transport simulation with vertex connection and merging. **ACM Trans. Graph.**, [S.l.], v.31, n.6, p.192–1, 2012.

GLOROT, X.; BENGIO, Y. Understanding the difficulty of training deep feedforward neural networks. In: OF THE THIRTEENTH INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE AND STATISTICS. **Proceedings...** [S.l.: s.n.], 2010. p.249–256.

GU, J. et al. Recent advances in convolutional neural networks. **Pattern Recognition**, [S.l.], 2017.

HE, K. et al. Deep residual learning for image recognition. In: IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION. **Proceedings...** [S.l.: s.n.], 2016. p.770–778.

IIZUKA, S.; SIMO-SERRA, E.; ISHIKAWA, H. Let there be color!: joint end-to-end learning of global and local image priors for automatic image colorization with simultaneous classification. **ACM Transactions on Graphics (TOG)**, [S.l.], v.35, n.4, p.110, 2016.

JAIN, V.; SEUNG, S. Natural image denoising with convolutional networks. In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS. **Anais...** [S.l.: s.n.], 2009. p.769–776.

JENSEN, H. W. Global illumination using photon maps. In: **Rendering Techniques' 96**. [S.l.]: Springer, 1996. p.21–30.

KAJIYA, J. T. The rendering equation. In: ACM SIGGRAPH COMPUTER GRAPHICS. **Anais...** [S.l.: s.n.], 1986. v.20, n.4, p.143–150.

KALANTARI, N. K.; BAKO, S.; SEN, P. A machine learning approach for filtering Monte Carlo noise. **ACM Trans. Graph.**, [S.l.], v.34, n.4, p.122–1, 2015.

KELLER, A. et al. The path tracing revolution in the movie industry. In: SIGGRAPH COURSES. **Anais...** [S.l.: s.n.], 2015. p.24–1.

KINGMA, D. P.; BA, J. Adam: a method for stochastic optimization. **arXiv preprint arXiv:1412.6980**, [S.l.], 2014.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS. **Anais...** [S.l.: s.n.], 2012. p.1097–1105.

LAFORTUNE, E. P.; WILLEMS, Y. D. Bi-directional path tracing. , [S.l.], 1993.

LARSSON, G.; MAIRE, M.; SHAKHNAROVICH, G. Learning representations for automatic colorization. In: EUROPEAN CONFERENCE ON COMPUTER VISION. **Anais...** [S.l.: s.n.], 2016. p.577–593.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **nature**, [S.l.], v.521, n.7553, p.436, 2015.

LECUN, Y. et al. Learning algorithms for classification: a comparison on handwritten digit recognition. **Neural networks: the statistical mechanics perspective**, [S.l.], v.261, p.276, 1995.

LEDIG, C. et al. Photo-realistic single image super-resolution using a generative adversarial network. **arXiv preprint**, [S.l.], 2017.

LIU, C. et al. Noise estimation from a single image. In: COMPUTER VISION AND PATTERN RECOGNITION, 2006 IEEE COMPUTER SOCIETY CONFERENCE ON. **Anais...** [S.l.: s.n.], 2006. v.1, p.901–908.

MCCOOL, M. D. Anisotropic diffusion for Monte Carlo noise reduction. **ACM Transactions on Graphics (TOG)**, [S.l.], v.18, n.2, p.171–194, 1999.

MOON, B.; CARR, N.; YOON, S.-E. Adaptive rendering based on weighted local regression. **ACM Transactions on Graphics (TOG)**, [S.l.], v.33, n.5, p.170, 2014.

MOON, B. et al. Adaptive polynomial rendering. **ACM Transactions on Graphics (TOG)**, [S.l.], v.35, n.4, p.40, 2016.

PETSCHNIGG, G. et al. Digital photography with flash and no-flash image pairs. In: ACM TRANSACTIONS ON GRAPHICS (TOG). **Anais...** [S.l.: s.n.], 2004. v.23, n.3, p.664–672.

PHARR, M.; JAKOB, W.; HUMPHREYS, G. **Physically based rendering**: from theory to implementation. [S.l.]: Morgan Kaufmann, 2016.

ROMERO, A. et al. Fitnets: hints for thin deep nets. **arXiv preprint arXiv:1412.6550**, [S.l.], 2014.

ROUSSELLE, F.; MANZI, M.; ZWICKER, M. Robust denoising using feature and color information. In: COMPUTER GRAPHICS FORUM. **Anais...** [S.l.: s.n.], 2013. v.32, n.7, p.121–130.

RUSSAKOVSKY, O. et al. Imagenet large scale visual recognition challenge. **International Journal of Computer Vision**, [S.l.], v.115, n.3, p.211–252, 2015.

SEN, P.; DARABI, S. On filtering the noise from the random parameters in Monte Carlo rendering. **ACM Trans. Graph.**, [S.l.], v.31, n.3, p.18–1, 2012.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. **arXiv preprint arXiv:1409.1556**, [S.l.], 2014.

SZEGEDY, C. et al. Going deeper with convolutions. In: **Anais...** [S.l.: s.n.], 2015.

SZEGEDY, C. et al. Inception-v4, inception-resnet and the impact of residual connections on learning. In: AAAI. **Anais...** [S.l.: s.n.], 2017. v.4, p.12.

TOMASI, C.; MANDUCHI, R. Bilateral filtering for gray and color images. In: COMPUTER VISION, 1998. SIXTH INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 1998. p.839–846.

VAN DEN OORD, A. et al. Wavenet: a generative model for raw audio. **arXiv preprint arXiv:1609.03499**, [S.l.], 2016.

VEACH, E.; GUIBAS, L. J. Metropolis light transport. In: COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, 24. **Proceedings...** [S.l.: s.n.], 1997. p.65–76.

WALSTON, D. E. An introduction to numerical analysis. , [S.l.], 1968.

WANG, Z. et al. Image quality assessment: from error visibility to structural similarity. **IEEE transactions on image processing**, [S.l.], v.13, n.4, p.600–612, 2004.

XIE, J.; XU, L.; CHEN, E. Image denoising and inpainting with deep neural networks. In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS. **Anais...** [S.l.: s.n.], 2012. p.341–349.

YANG, W. et al. Deep edge guided recurrent residual learning for image super-resolution. **IEEE Transactions on Image Processing**, [S.l.], v.26, n.12, p.5895–5907, 2017.

ZEILER, M. D. ADADELTA: an adaptive learning rate method. **arXiv preprint arXiv:1212.5701**, [S.l.], 2012.

ZHANG, G. P. Neural networks for classification: a survey. **IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)**, [S.l.], v.30, n.4, p.451–462, 2000.

ZHAO, H. et al. Is L2 a good loss function for neural networks for image processing? **ArXiv e-prints**, [S.l.], v.1511, 2015.

ZIMMER, H. et al. Path-space Motion Estimation and Decomposition for Robust Animation Filtering. In: COMPUTER GRAPHICS FORUM. **Anais...** [S.l.: s.n.], 2015. v.34, n.4, p.131–142.

ZWICKER, M. et al. Recent advances in adaptive sampling and reconstruction for Monte Carlo rendering. In: COMPUTER GRAPHICS FORUM. **Anais...** [S.l.: s.n.], 2015. v.34, n.2, p.667–681.

Appendix

A

Basic Monte Carlo Integration

Suppose we want to evaluate the integral:

$$I = \int_a^b f(x)\, dx \tag{A.1}$$

$$F_N = \frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{p(x_i)} \tag{A.2}$$

x is chosen randomly in some domain of integration D with a probability density $p(x)$. Choosing p intelligently is called importance sampling, because if p is large where f is large, there will be more samples in important regions. For instance, if we choose x uniformly in the domain $[a,b]$ we get:

$$F_N = (b-a)\frac{1}{N}\sum_{i=1}^{N} f(x_i) \tag{A.3}$$

Let us obtain the expectation of F_N:

$$E[F_N] = E[\frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{p(x_i)}] \tag{A.4}$$

$$E[F_N] = \frac{1}{N}\sum_{i=1}^{N} E[\frac{f(x_i)}{p(x_i)}] \tag{A.5}$$

$$E[F_N] = \frac{1}{N}\sum_{i=1}^{N}\int_a^b \frac{f(x)}{p(x)}p(x)dx \tag{A.6}$$

$$E[F_N] = \int_a^b f(x)dx \tag{A.7}$$

The expected value of the estimator is the integral we want evaluate. So, the Monte Carlo method is not biased.