WAMnet for Image Denoising

Aaron Bies[1], Adarsh Djeacoumar[1], and Shailesh Mishra[1]

Saarland University

Abstract. Photo-realistic images generated using path tracing often contain a significant amount of noise as it involves Monte-Carlo integration. Raising the total number of samples in an image reduces the amount of noise but increases the computational cost of the algorithm. It is desirable to reduce the noise and generate high-quality renders at a low computational cost. We present a lightweight novel MLP based filter architecture that denoises images rendered at 1 sample per pixel and its corresponding normal. Furthermore, this lightweight network makes use of fragment shaders and delivers a run-time performance of roughly 11 frames per second. This architecture significantly reduces the computation cost required to render high-quality images. Our network reduces noise by approximately 40% and improves the resulting structure by over 200%. We found that using the corresponding normal maps of the input image in training significantly enhances the performance. Therefore, we believe extension based on the concepts from computer graphics can improve the performance of denoising networks.

Keywords: Ray Tracing · Image Denoising · Neural Networks.

1 Introduction

Path tracing is the state of the art method for generating photo-realistic images in computer graphics and has been for many years [1]. In its most common form, it is a method based on Monte Carlo integration, which works by shooting rays from the camera into the scene, randomly bouncing them off surfaces, and sampling light sources to iteratively approximate the brightness received by each pixel of our virtual camera [1]. Intuitively it traces the path a ray of light would take in the real world on its way from a light source to the camera, only in reverse.

To produce high-quality renders, these methods perform sampling on each pixel and average the incoming radiance from all samples on that pixel. Despite being a robust method, i.e., being free from bias, it requires large computation time, and the computation explodes with the increase in resolution. Therefore, computer vision algorithms are applicable in reducing the noise for a rendered image obtained using low sample counts. In this report, we highlight the contribution that allows us to denoise the input images rendered using 1 sample per pixel (SPP) in real-time. The filter we use is a multi-layer perceptron network that is lightweight and can run on a single fragment shader.

2 Prior Work

Chaitanya et al. [2] described an Autoencoder with recurrent connects that improved the temporal stability for sequences of sparsely sampled input images. They attempt to remove a class of noise present in Monte Carlo rendering by reconstructing the global illumination in the scene, using Convolutional Neural Networks. They demonstrate that their method models relationships, such as depth and normals, with the input channels.

Müller et al. [4] in their paper study non-linear independent components estimation (NICE) and its extensions to generate samples in Monte Carlo integration using deep neural networks. They demonstrate learning of joint path-sampling densities in the primary sample space and importance sampling of multi-dimensional path prefixes. Their approach extracted and leveraged the conditional directional densities for path guiding. In a way, their experiments revealed that Neural path guiding performed on-par or better than other path tracing techniques at equal sample count.

Zhang et al. in [6] introduced the Denoising Convolutional Neural Network (DnCNN) for denoising images. Their experiments demonstrate that the DnCNN can handle Gaussian noise of unknown noise level (blind Gaussian denoising). They adopt the residual learning formulation to train a residual mapping $R(y) \approx v$, and then $x = y - R(y)$ gives the predicted noise for an input given by $y = x + v$. They address various problems such as Gaussian denoising, single image super-resolution, and JPEG image deblocking.

Isola et al. in [3] illustrate their model pix2pix, a conditional GAN, for image-to-image translation tasks. They validate their model on settings in graphics (photo generation), and vision (semantic segmentation). Community access to the pix2pix model further proved its robustness as a general-purpose solution for image-to-image translation tasks.

We take inspiration from these works to construct a lightweight network that can be a part of a graphics pipeline and fit in a fragment shader, and can perform on-par or close to the state-of-the-art networks. To this end, we propose the "Wait a minute network" (WAMnet). WAMnet is a multi-layer perceptron network that is scene-specific, fits in a pipeline, and can generate a 3D scene in real-time. Note that while we do not train our model on different orientations, the pipeline also generates the scene for unknown orientations equally good as the scene that was trained.

3 Architecture of WAMnet

Given an NxN patch from a noisy input image as well as the corresponding normal map, our network predicts the color of the center pixel of that chunk. This allows the network to take neighbouring pixels, information from the normal maps and inferred surface properties into account. The network, WAMnet, is a 5 layer Multi-layer perceptron each with hidden layers consisting of 32 neurons. These neurons are activated by a periodic activation function, inspired by the SiREN network [5]. The overall architecture is depicted in Fig. 1.

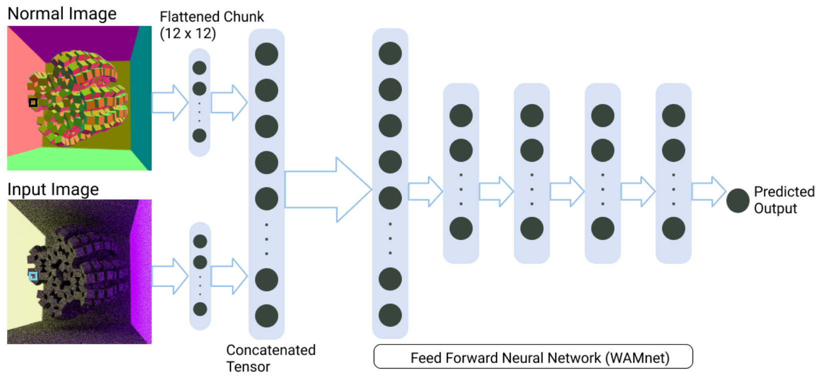


Fig. 1: Architecture of WAMnet

4 Experiments

4.1 Dataset

We start our experiment by designing a custom scene in a fragment shader. Using this custom scene, we render our training sets as well as testing sets. The training set consists of 1000 rendered images along with their corresponding normals. The input image is rendered at 1 SPP while the ground truth is rendered at 1024 SPP. Our testing dataset consists of 100 rendered images at the same settings as the training set. For both testing and training set, we set up our image resolution to be 1024×1024 shown in Appendix A.

4.2 WAMnet

We train our model for a single scene using a single pair of input and output images. For each scene, we create $20,000$ training samples by randomly sampling an input chunk of size 12×12 from 1024×1024 input image. Out of these $20,000$, we pass $10,000$ random chunks to our network in one iteration. We repeat the training for $20,000$ iterations. This approach of training our neural net on a single scene requires approximately 90 secs for a given scene. We designed a series of experiments to evaluate the performance of our model.

First, we train WAMnet by giving 3 channel (RGB) input chunk and its corresponding output chunk (WAMnet\normals). In another experiment, we append the normal map generated during the pipeline to our input chunk creating a 6 channel input chunk and compare with the same output chunk (WAMnet). We noticed considerable improvements in the geometry of the shapes in the scene when using normals. When trained without a normal map, WAMnet fails to maintain the sharp geometry and works closer to a blur filter. In both cases, we use Adam optimizer for minimizing the mean squared error (MSE) loss described as:

$$L(y, \tilde{y}) = \frac{1}{N}\sum_{i=1}^{N}||(y - \tilde{y})||^2 \tag{1}$$

4.3 pix2pix

We split our dataset into smaller patches (128×128) and pass it to the network. We use the U-Net that is 6 layers deep and also takes Gaussian noise (z) as input to prevent deterministic outputs, as our encoder-decoder architecture for our generator (G). The discriminator (D) architecture is the PatchGAN network, which is a convnet and is 3 layers deep. Note that we are trying to optimize our architecture to run on a fragment shader and hence we design our models as light as possible. We primarily use MSE (L2 norm) and reconstruction loss (L1 norm) for our custom loss function. We observed that penalizing the MSE and reconstruction loss in the ratio of 1:100 gave the best results. Our custom loss function is described below:

$$\begin{aligned} L_D &= \frac{1}{2}\left(L(D(x), target_{real}) + L(D(G(x,z)), target_{fake})\right) \\ L_G &= L(D(G(x,z)), y) \\ L_{L1} &= \frac{1}{N}\sum_{i=1}^{N}||(D(G(x,z)) - target_{real})|| \\ L_{GAN} &= L_G + 100 * L_{L1} \end{aligned}$$

4.4 DnCNN

We resize our dataset from its original 1024×1024 to 256×256 to fit in a GPU. The DnCNN architecture we use is a convnet of depth 16. [6] describe the DnCNN architecture to be able to learn the residual map $R(y) \approx v$, where v is the noise. Then, the output image is calculated by $x = y - R(y)$, where x is the clean image. We adopt the averaged MSE between the desired and estimated residual mappings as the loss function:

$$L(\phi) = \frac{1}{2N}\sum_{i=1}^{N}||(R(y_i, \phi) - (y_i - x_i))||^2 \tag{2}$$

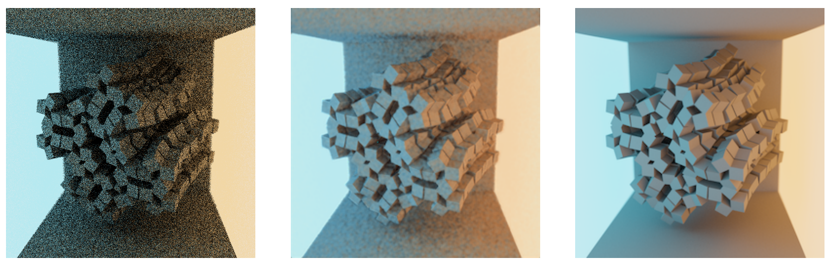
5 Results

5.1 Evaluation Metrics

To assess the performance of the neural network we employ the following metrics: MSE, PSNR, and SSIM. By definition, the lower the MSE, the higher the PSNR, and closer the predicted image is to the ground truth. SSIM, which stands for Structural Similarity Index Measure, compares the luminance, contrast, and structure between the predicted image and the ground truth. The SSIM score is present in $[0, 1]$ and the closer the score is to 1, the better the image.

Table 1: **Performance comparison with relevant methods**. Please note despite pix2pix showing superior metrics, the overall results introduces unnecessary artifacts.

Test Network	MSE	PSNR	SSIM
Input Images	82.93	66.67	0.21
DnCNN	79.03	67.16	0.44
WAMnet	48.04	72.17	0.68
pix2pix	39.28	74.18	0.70



(a) Input (1 SPP) (b) Predicted result (c) GT (1024 SPP)

Fig. 2: Result showing prediction from WAMnet

5.2 Performance

Table 1 shows the performance of the networks that we trained on our dataset. We can observe that the input images rendered at 1 SPP have a high MSE, low PSNR, and poor SSIM score. Our input images fail to maintain the structure of the desired scene view. Given such a scene, our network can reconstruct the geometry (edges of the mini cubes are clear) and denoise it resulting in a structure that is 68% similar to the ground truth. We also observe that using normal maps in the input sequence reconstructs better than in the absence of normals.

Furthermore, our network outperforms DnCNN by a significant margin in all three metrics. Finally, pix2pix performs better than WAMnet by 2%. While this is the case, the complexity of the pix2pix model poses implicit constraints on the fragment shader and hence is unsuitable for our pipeline. Moreover, since we train pix2pix on patches sampled from the input images, we observe blocking artifacts when we reconstruct the image from the noisy input image. In this sense, WAMnet is not only good but also consistent and fits our purpose perfectly.

Fig. 2 shows how our network can retain the scene geometry as well as soft shadows. Since we feed in the normal map, we believe that the network is able to estimate the light contribution on the missing pixels using the information from noisy input as well as the normal map.

Fig. 3 compares results of WAMnet, pix2pix and DnCNN for one image from our test set. We observe that WAMnet and pix2pix perform extremely better

(a) WAMnet (b) pix2pix (c) DnCNN

Fig. 3: Comparison of predictions with different methods

than DnCNN. While pix2pix looks better in low resolution, the inherent block artifacts that come as a result of patch training are visible in higher resolutions.

6 Further Studies

We also conducted further experiments to test out the generalization capability of our network. While networks such as DnCNN and pix2pix generalize for the entire training set and are capable of working independently on the test set, WAMnet is a more specialised network and can only work on the scene that it is trained on. Considering the fact that the model contains only 5 layers, while compared to the deep layers in pix2pix and DnCNN, the trade-off is justified as, just by training on one particular scene for 90 seconds, our pipeline can generate a 3D scene that varies in time at 11 frames per second.

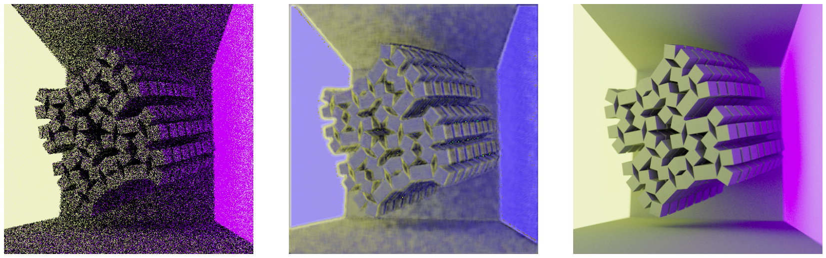
Moreover, the resulting scene is comparable to a scene that is rendered at 30 SPP. To put it in perspective, the renderer takes about a minute to generate an image at 1 SPP, while it takes about 7 minutes to generate an image at 30 SPP. Our pipeline aims to achieve generating a scene at higher SPP given 1 SPP (the lowest possible).

6.1 Different lights for training and testing

Instead of training and testing on the same scene with the same light, we changed the light colors on training as well as testing the image. For the same scene when trained on an arbitrary set of light colors $\boldsymbol{A}$, and tested on a different set of light colors $\boldsymbol{B}$, we noticed that our network fails to reproduce the original color. This limitation arises from the size of our network. The results can be observed in Fig. 4

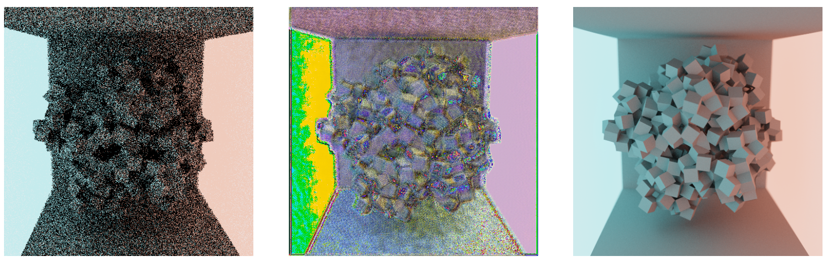
6.2 Training on multiple lights

We also conducted experiments where we trained on different sets of lights. When trained on multiple sets of lights, we found that our model tries to produce the



(a) Input (1 SPP) (b) Predicted result (c) Original (1024 SPP)

Fig. 4: Result showing effect of different colors for training and testing on same scene



(a) Input (1 SPP) (b) Predicted result (c) Original (1024 SPP)

Fig. 5: Result showing effect of use of multiple colors in training phase

colors but is not able to reproduce them successfully. The visualization can be seen from Fig. 5

7 Discussion

From our experiments, we notice that passing in the normals allows the MLP to learn the scene geometry. Since we were constrained to run it on a fragment shader as a part of our graphics pipeline, we didn't go for large and deep networks as they can not be compiled using the fragment shader. Therefore, we sacrifice generalization and work on specialization because of the budget constraint imposed by the engineering design.

8 Conclusions

With our work, we have successfully designed a neural network solution that is capable of running inside a fragment shader as a part of a graphics pipeline. Our network is compact and fast enough to run inside the GPU without using any interface from CUDA API. Since it is not possible to address a variety of scenes, we have traded generalization for specialization and fast training time. Because of the short training time, our network can fit the cases where it is expensive to generate the animation for a given scene at high quality.

9 Future Works

Our work is one of its kind to run a denoising network inside a fragment shader. From this work, we notice that running a neural network requires extensive optimization and we encourage the readers to pursue this path. If it is not possible to run on the fragment shader, one can optimize the graphics pipeline by introducing CUDA calls in between. By doing so, one can run large models that can generalize to a large variety of scenes. Further, we also noticed that to achieve a significant performance gain, one shouldn't just be limited to vision algorithms but can also introduce the concepts from graphics. One such direction is passing the ray direction along with normals and the input image.

References

1. Realistic image synthesis, https://graphics.cg.uni-saarland.de/courses/ris-2021/index.html
2. Chaitanya, C.R.A., Kaplanyan, A.S., Schied, C., Salvi, M., Lefohn, A., Nowrouzezahrai, D., Aila, T.: Interactive reconstruction of monte carlo image sequences using a recurrent denoising autoencoder. ACM Trans. Graph. **36**(4), 1–12 (2017)
3. Isola, P., Zhu, J.Y., Zhou, T., Efros, A.A.: Image-to-image translation with conditional adversarial networks. CVPR (2017)
4. Müller, T., McWilliams, B., Rousselle, F., Gross, M., Novák, J.: Neural importance sampling. ACM Trans. Graph. **38**(5), 1–19 (2019)
5. Sitzmann, V., Martel, J., Bergman, A., Lindell, D., Wetzstein, G.: Implicit neural representations with periodic activation functions. Advances in Neural Information Processing Systems **33** (2020)
6. Zhang, K., Zuo, W., Chen, Y., Meng, D., Zhang, L.: Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising. IEEE Transactions on Image Processing **26**(7), 3142–3155 (2017)

A Dataset

We show a sample of our input (Fig. 6), normal (Fig.7) and output (Fig. 8) images rendered using our ray tracing algorithm.

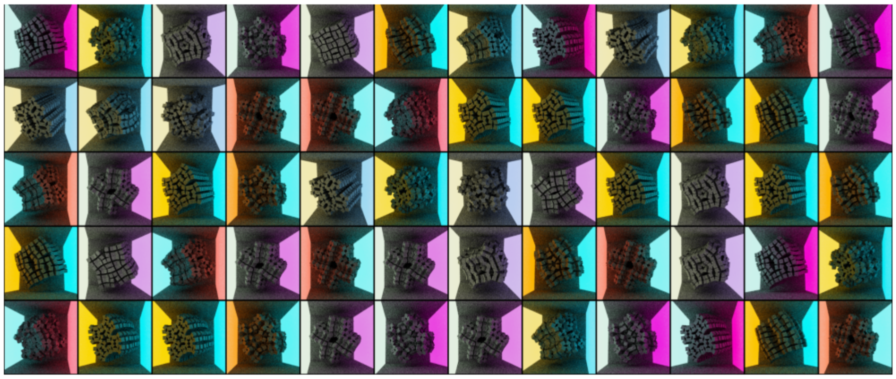


Fig. 6: Input Images

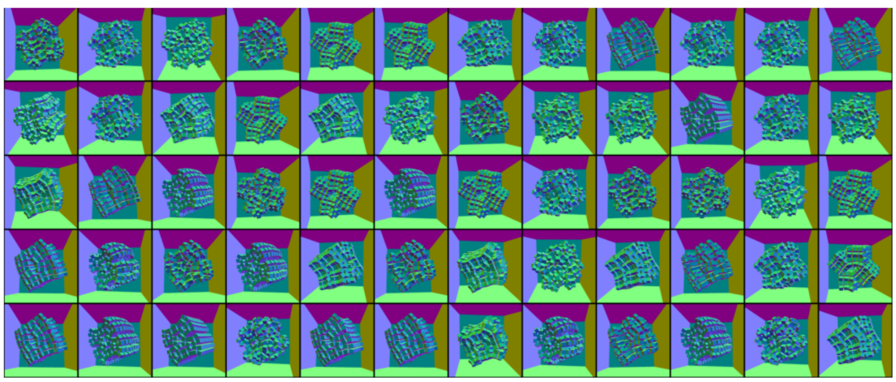


Fig. 7: Normal Images

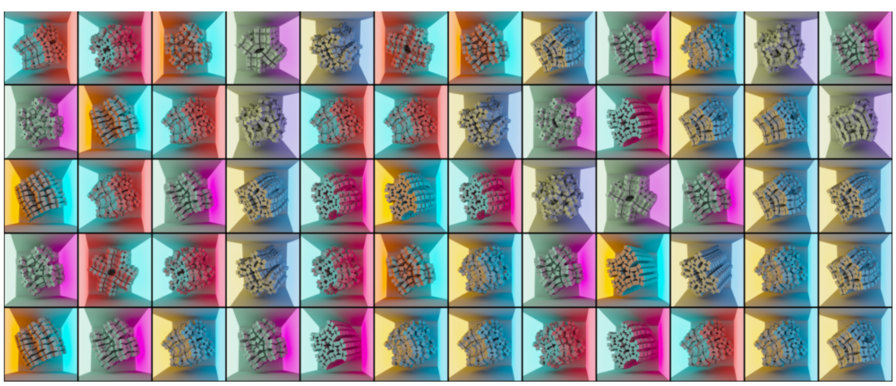


Fig. 8: Output Images