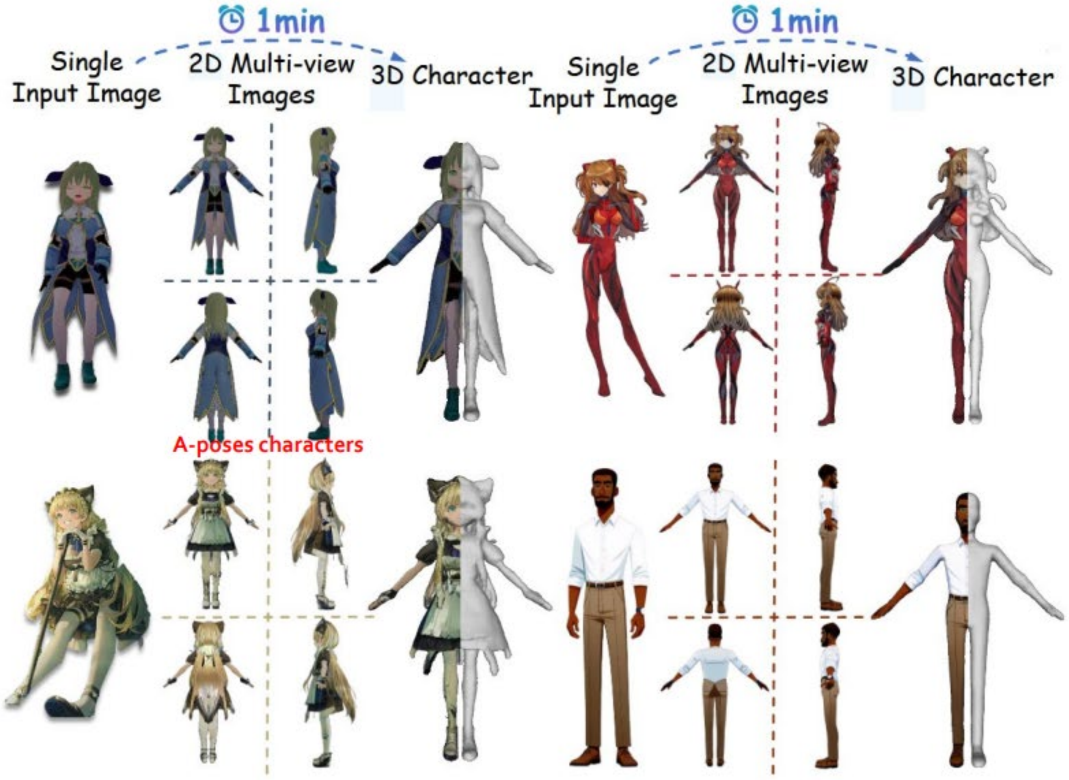
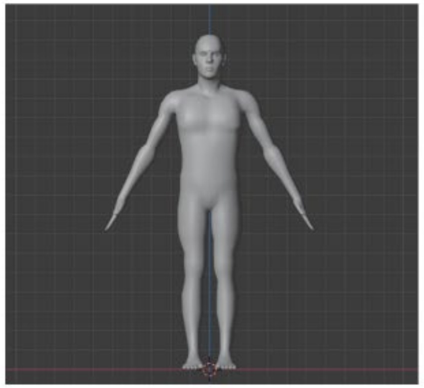
CharacterGen: Efficient 3D character generation from single images with multi-view pose calibration





A-poses

EUROGRAPHICS 2022

43RD ANNUAL CONFERENCE OF
THE EUROPEAN ASSOCIATION FOR COMPUTER GRAPHICS

Université de Reims Champagne-Ardenne

REIMS · FRANCE
APRIL 25·29 / 2022

Progressive Denoising of Monte Carlo Rendered Images

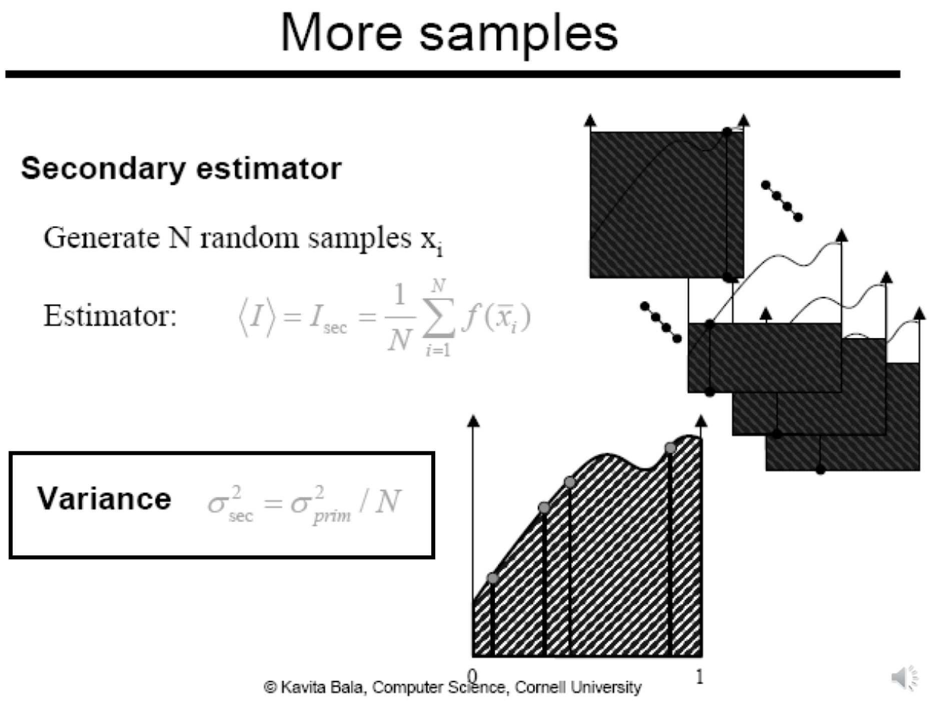
Arthur Firmino, Jeppe Revall Frisvad, and Henrik Wann Jensen

April 26th, 2022
Centre des Congrès de Reims

LUXION

DTU Compute
Department of Applied Mathematics and Computer Science

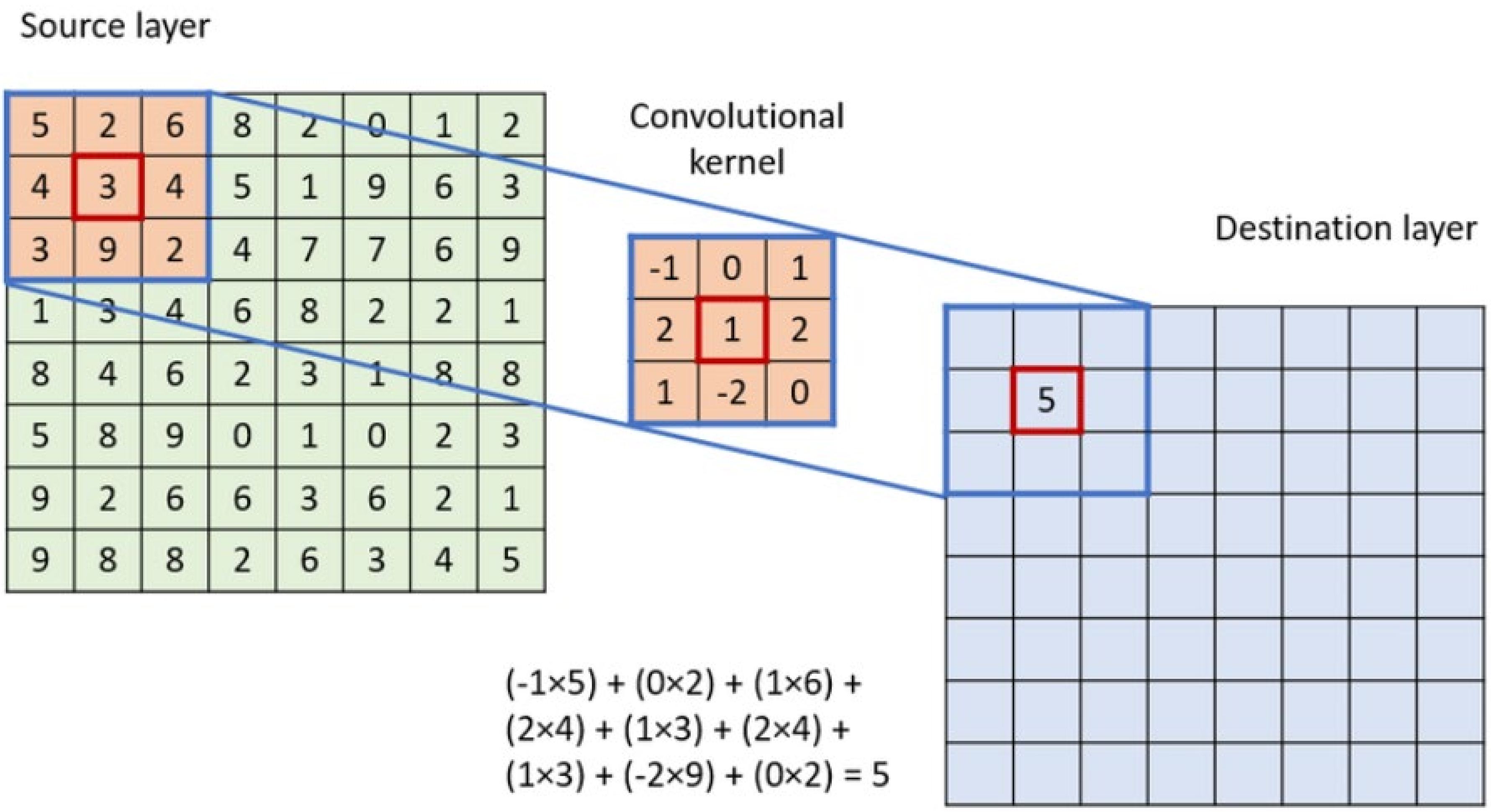
Backgrounds – Denoisers



Lecture 04. Montecarlo Integration

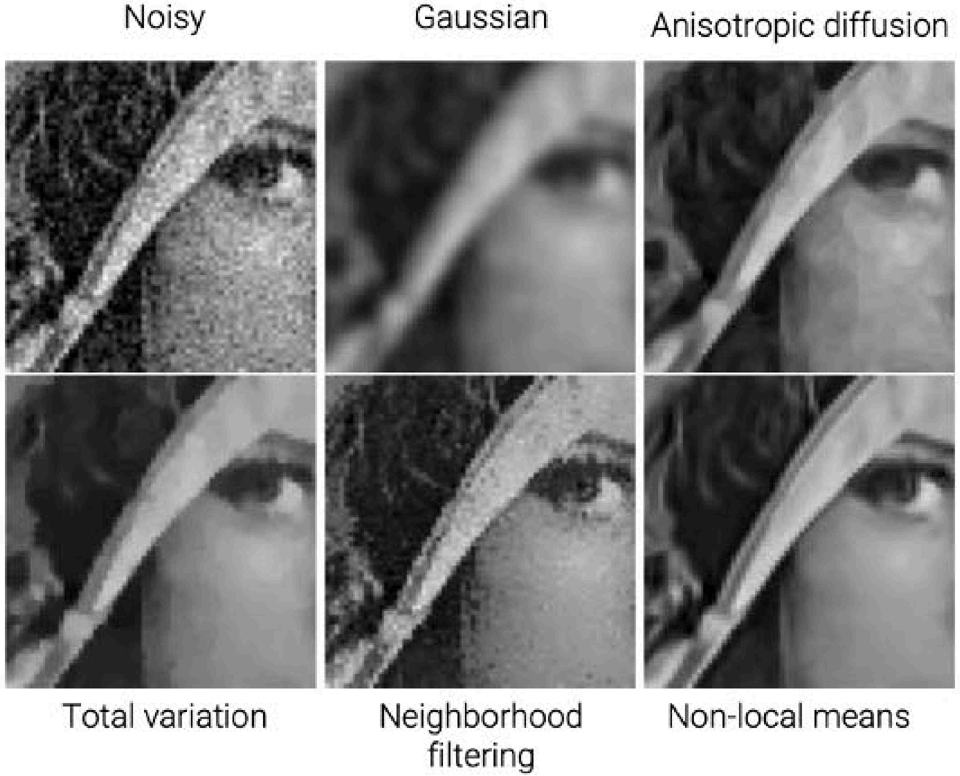
Special Lecture. Monte Carlo Noise Reduction

Background – Image Filters

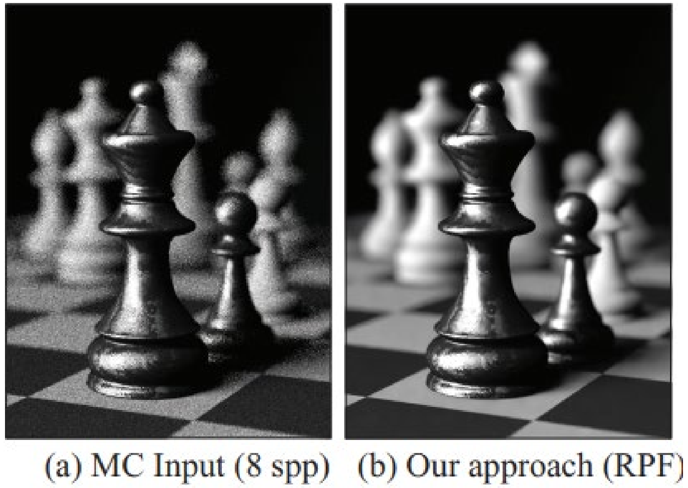


$(-1\times5) + (0\times2) + (1\times6) +$
$(2\times4) + (1\times3) + (2\times4) +$
$(1\times3) + (-2\times9) + (0\times2) = 5$

Background – Image Filters



Background – Image Filters



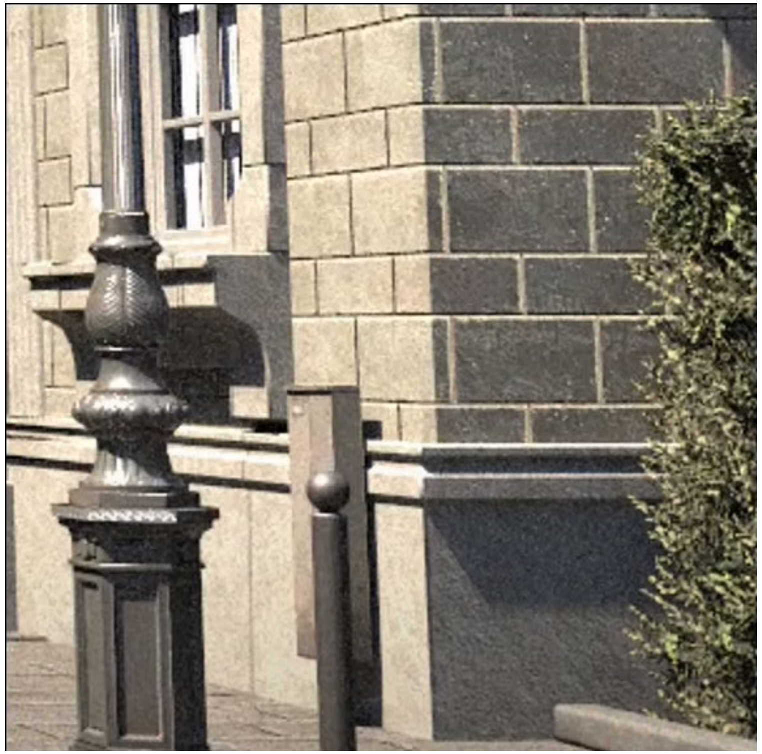
Cross-Bilateral Filter

High-order filter

SEN P., DARABI S.: On filtering the noise from the random parameters in Monte Carlo rendering.

BITTERLI B et al. Nonlinearly weighted first-order regression for denoising Monte Carlo renderings.

Problem – Loss of Detail

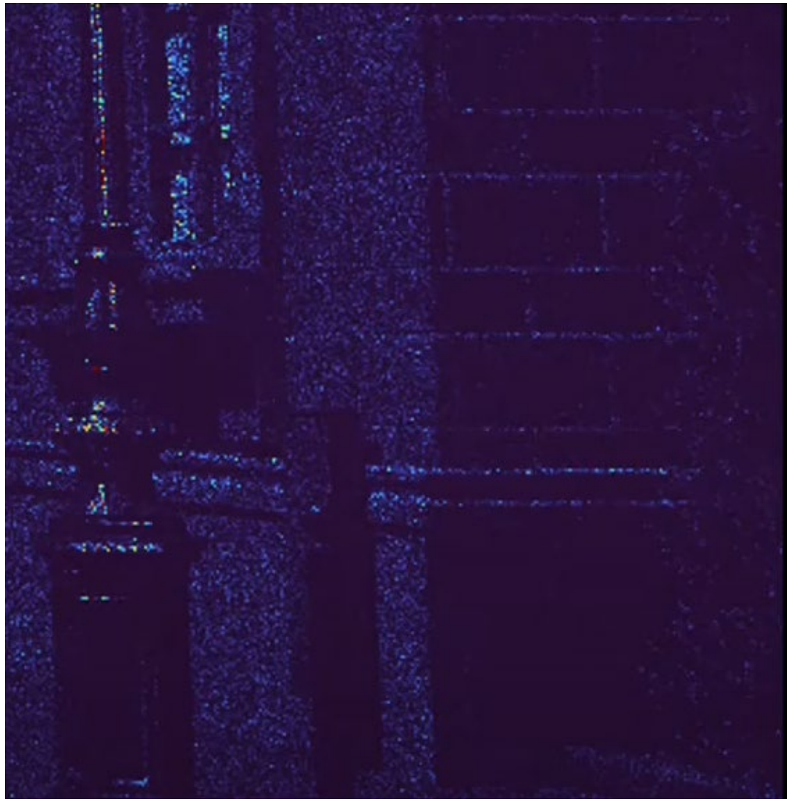


Rendered Image

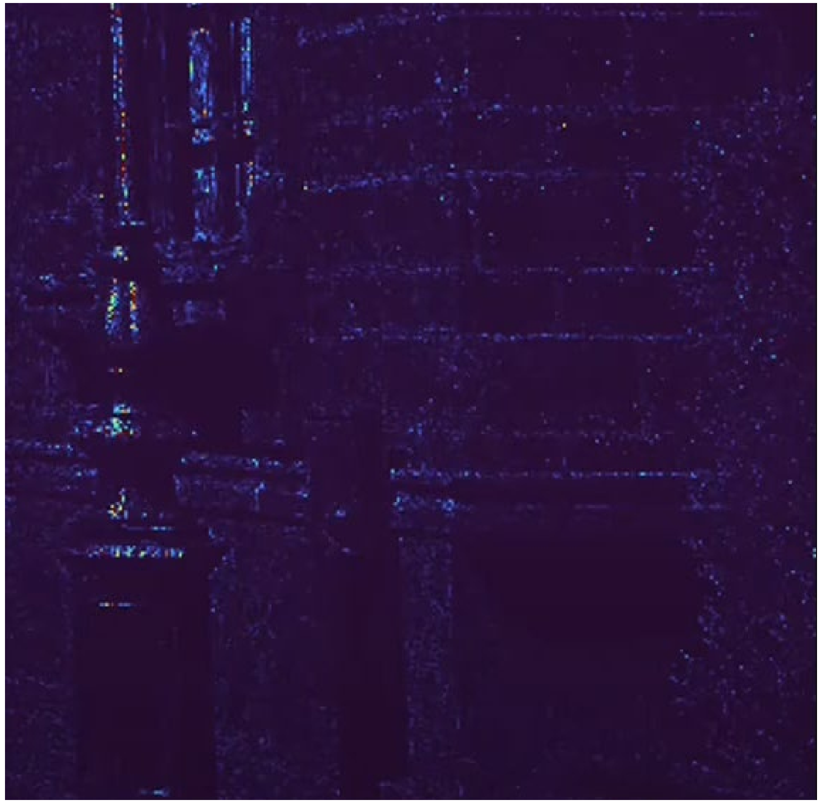
Denoised Image

Reference

Problem – Loss of Detail



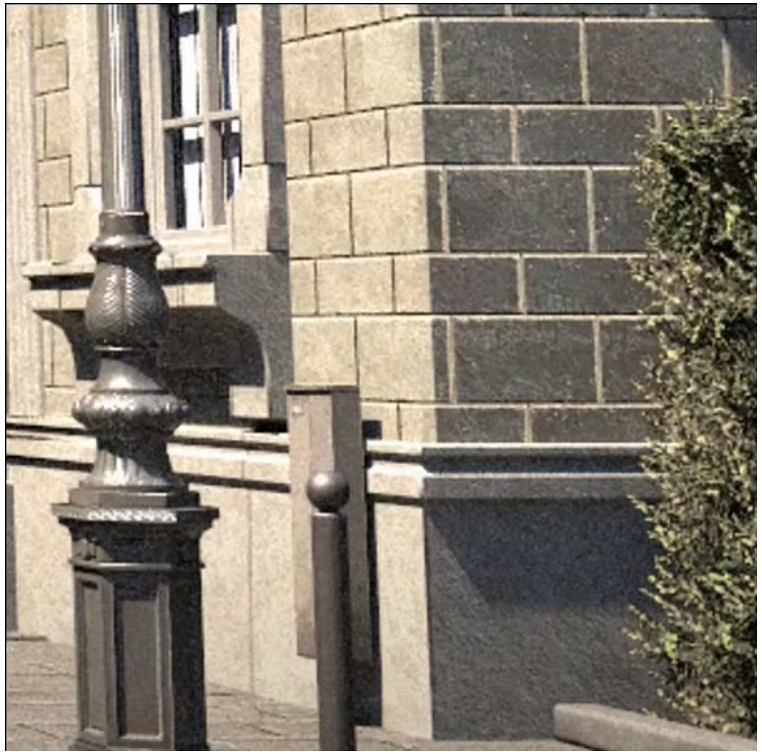
Rendered Image



Denoised Image

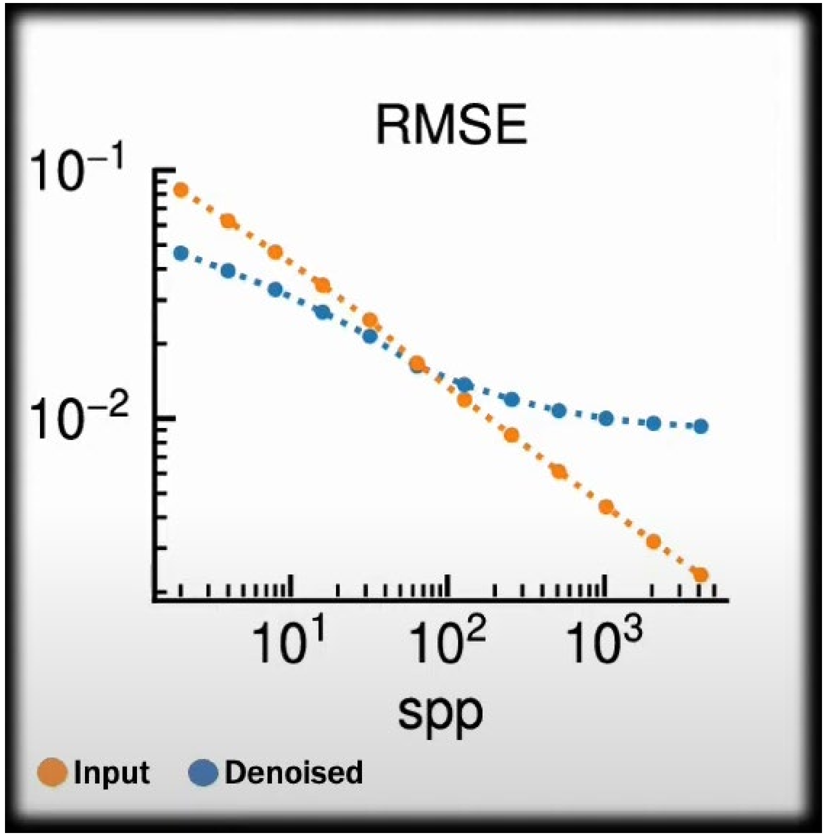
Reference

Problem - Non-converging



Rendered Image

Denoised Image



Method

Goal: To fix two major problems by mixing parameter α

Problems:

1. **Loss of detail**
2. **Non – converging**

Method

$(1-\alpha)$  $+\ \alpha \cdot$  $=$ Good Image!

Rendered Image

Denoised Image

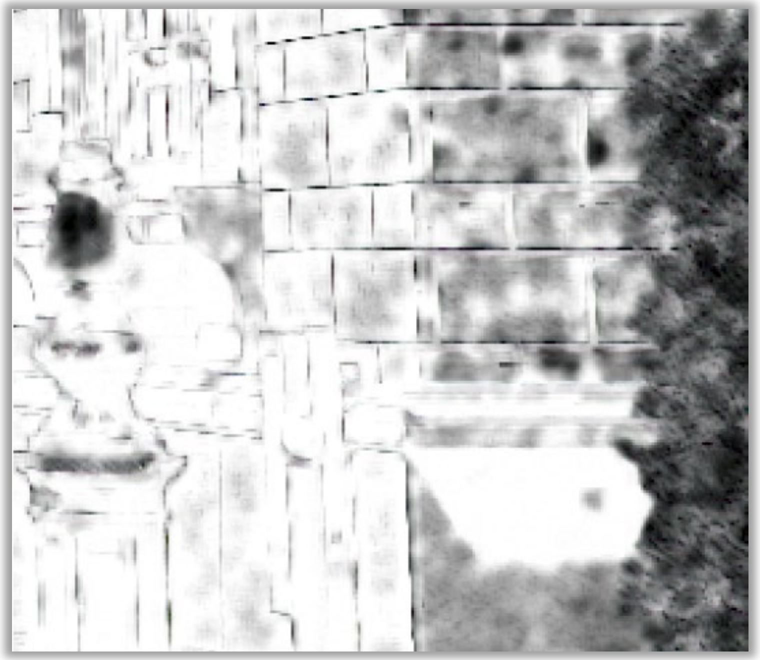
Method

$(1-\alpha)$

Rendered Image

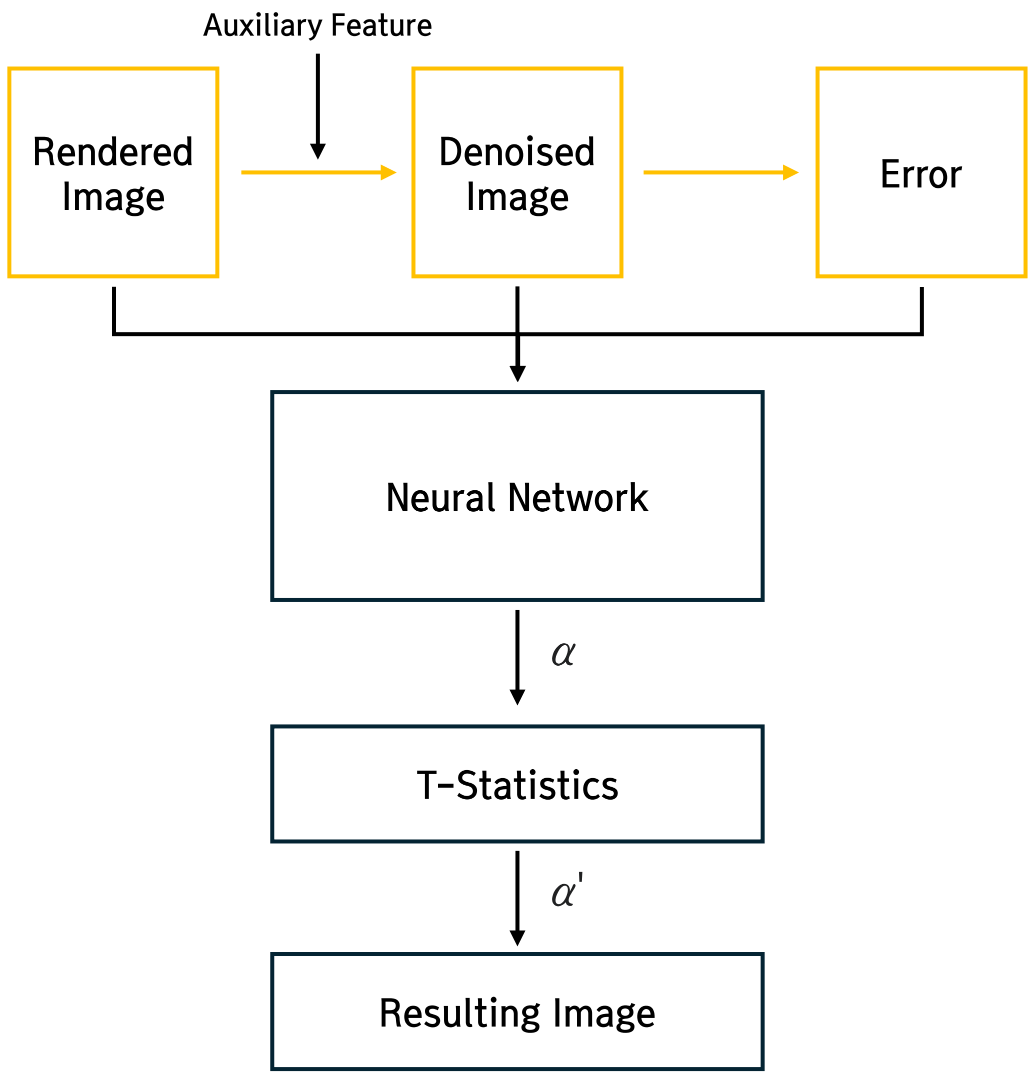
$+\ \alpha \bullet$

Denoised Image



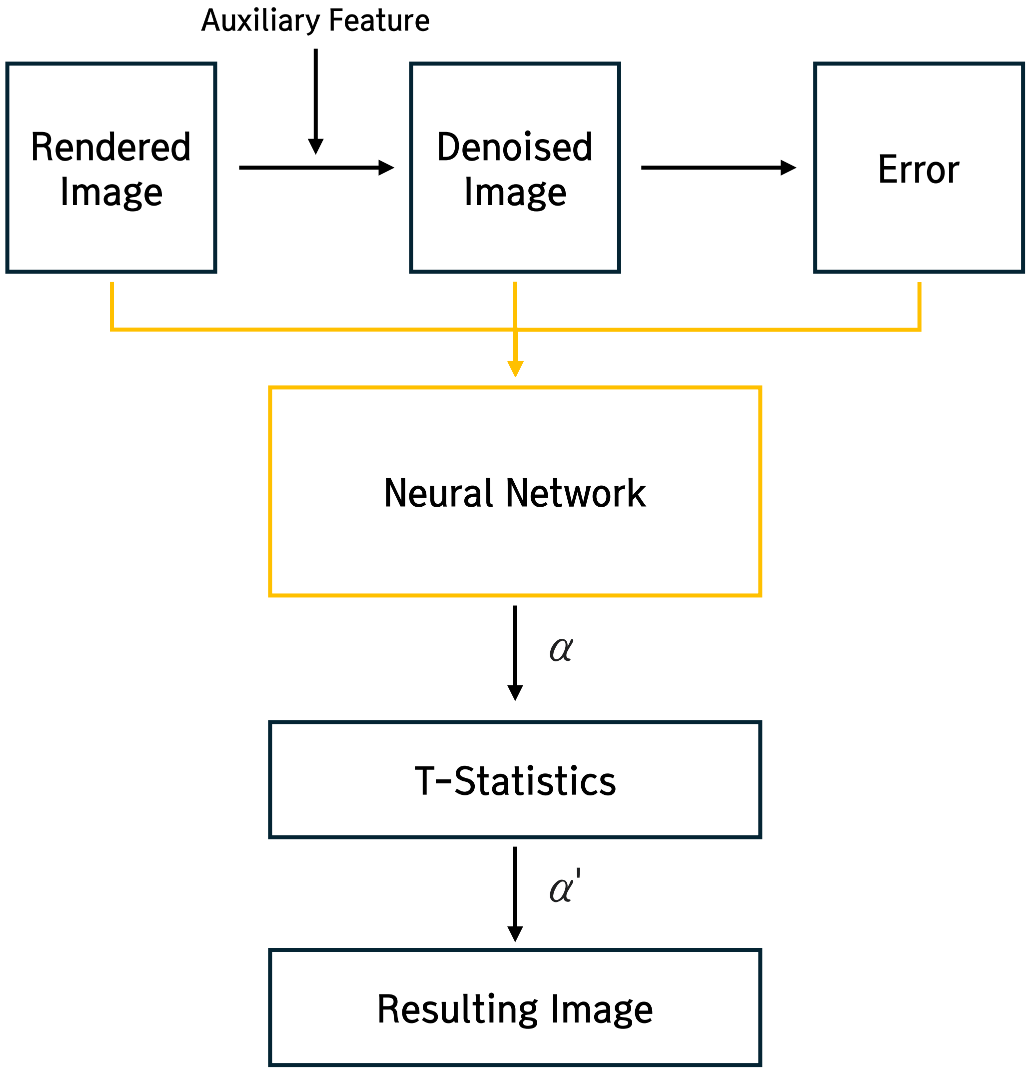
Resulting α

Pipeline



1. Generate denoised image from rendered image. Calculate error from denoised image.
2. Feed rendered image, denoised image and error to neural network.
3. Receive α as output.
4. Rescale α with t-statistics.
5. Generate resulting image.

Pipeline



1. Generate denoised image from rendered image. Calculate error from denoised image.

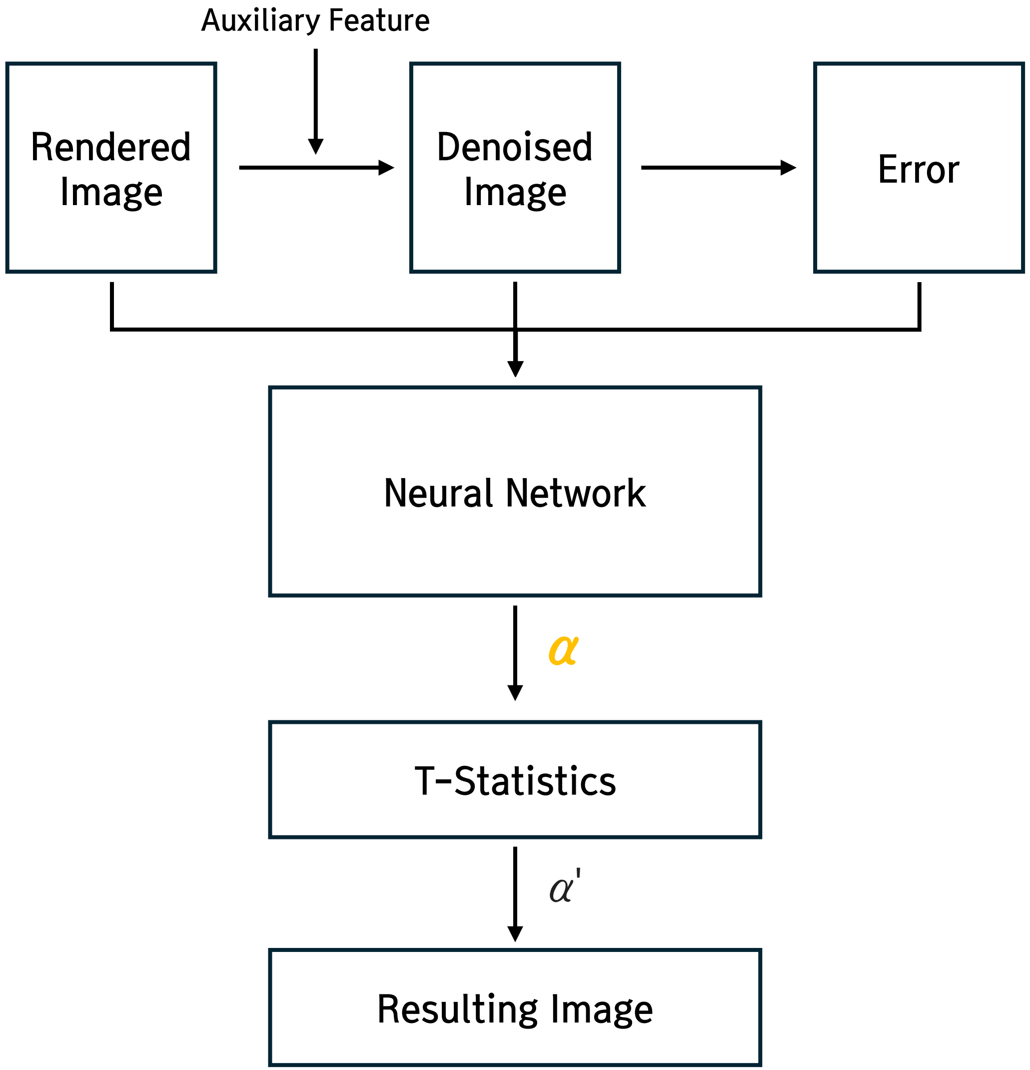
2. **Feed rendered image, denoised image and error to neural network.**

3. Receive α as output.

4. Rescale α with t-statistics.

5. Generate resulting image.

Pipeline



1. Generate denoised image from rendered image. Calculate error from denoised image.

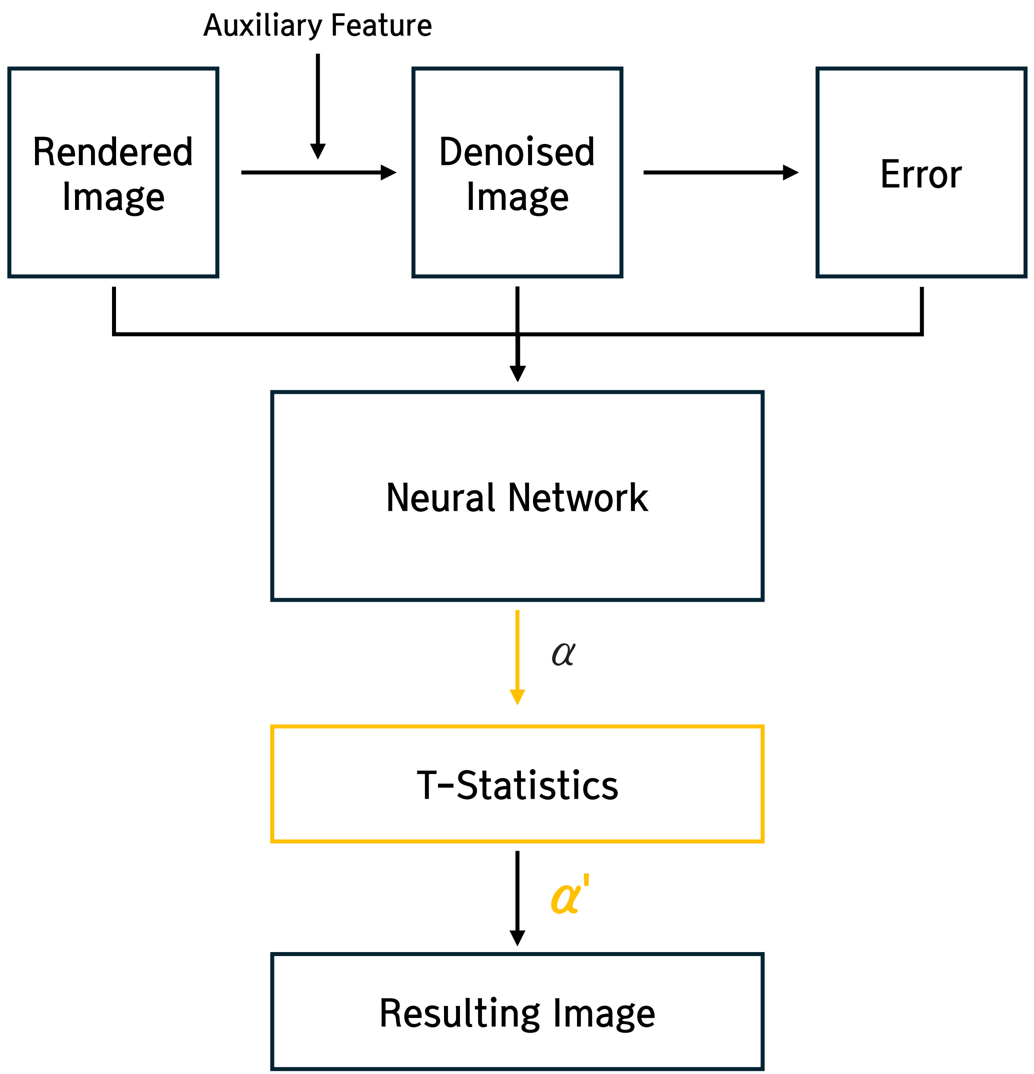
2. Feed rendered image, denoised image and error to neural network.

3. **Receive α as output.**

4. Rescale α with t-statistics.

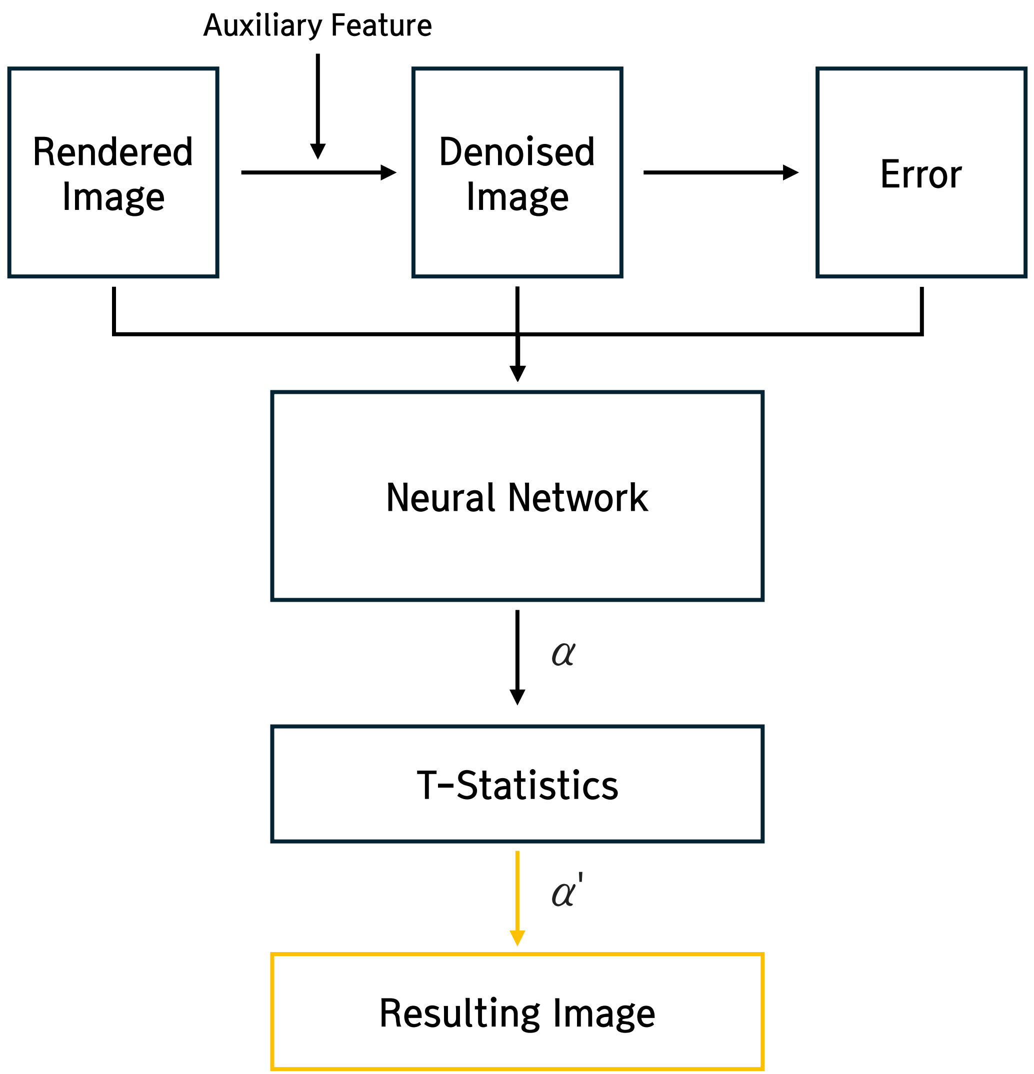
5. Generate resulting image.

Pipeline



1. Generate denoised image from rendered image. Calculate error from denoised image.
2. Feed rendered image, denoised image and error to neural network.
3. Receive α as output.
4. **Rescale α with t-statistics.**
5. Generate resulting image.

Pipeline

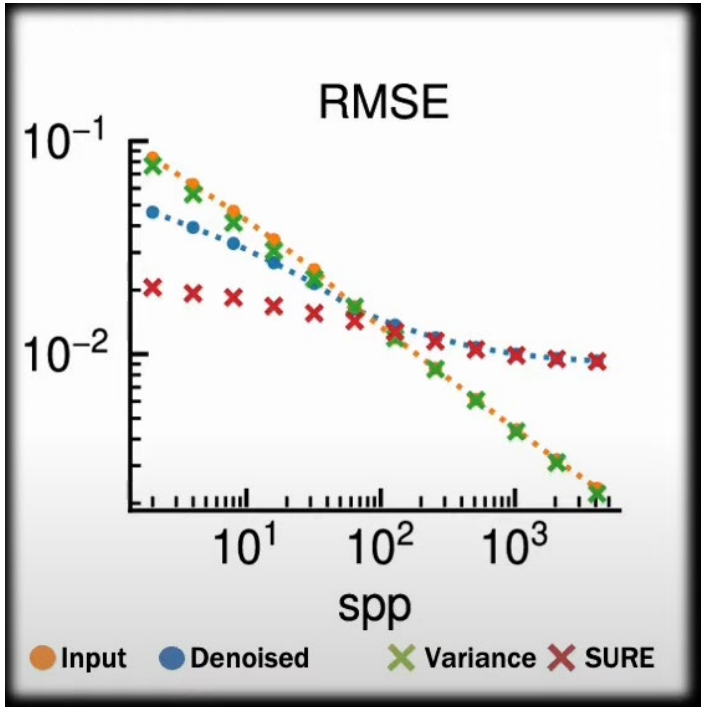


1. Generate denoised image from rendered image. Calculate error from denoised image.
2. Feed rendered image, denoised image and error to neural network.
3. Receive α as output.
4. Rescale α with t-statistics.
5. **Generate resulting image.**

Error Estimation – SURE

1. Generate denoised image from rendered image. Calculate **error** from denoised image.

$$\mathrm{SURE}(F,x) = \frac{1}{d}\left(\|F(x)-x\|^2 + 2\mathrm{tr}(J_F(x)\cdot\Sigma) - \mathrm{tr}(\Sigma)\right)$$



Error Estimation – SURE

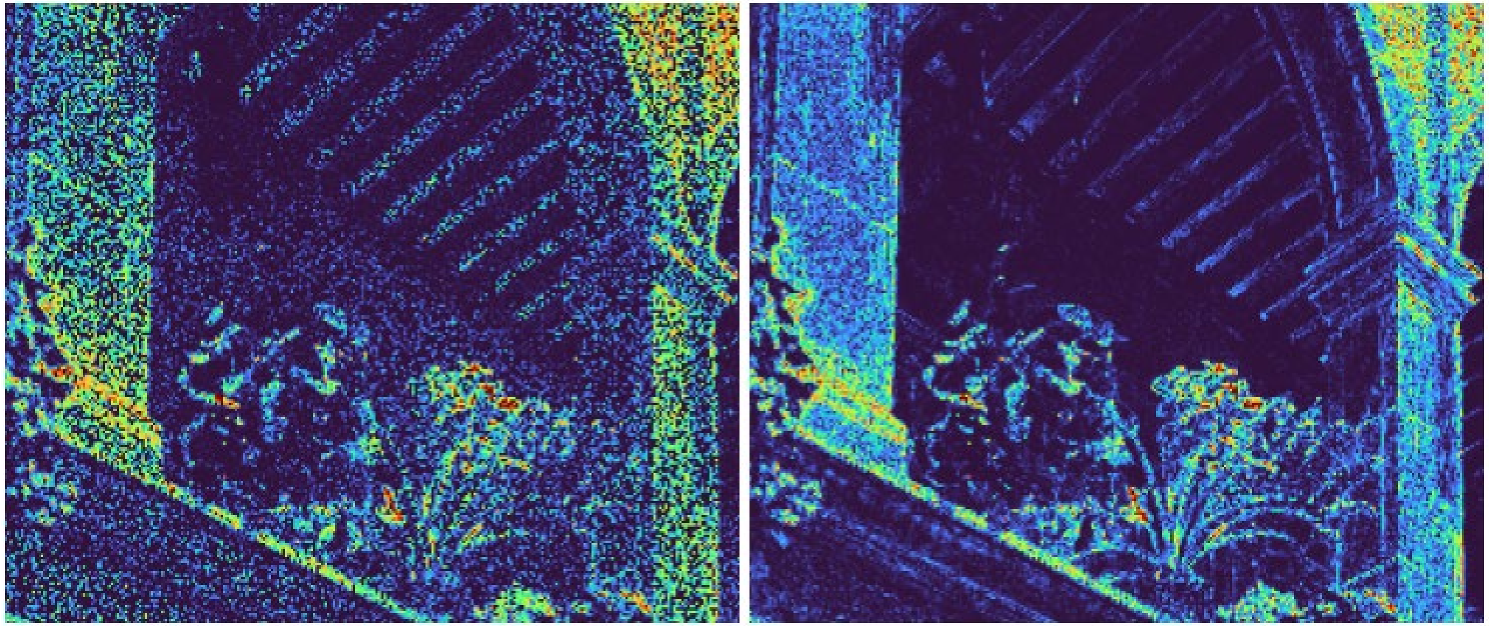


Figure 3: *Per-pixel squared error estimate of a denoised image using SURE (left), and its actual squared error (right).*

Input swapping

2. Feed rendered image, denoised image and error to neural network.
3. Receive α as output.

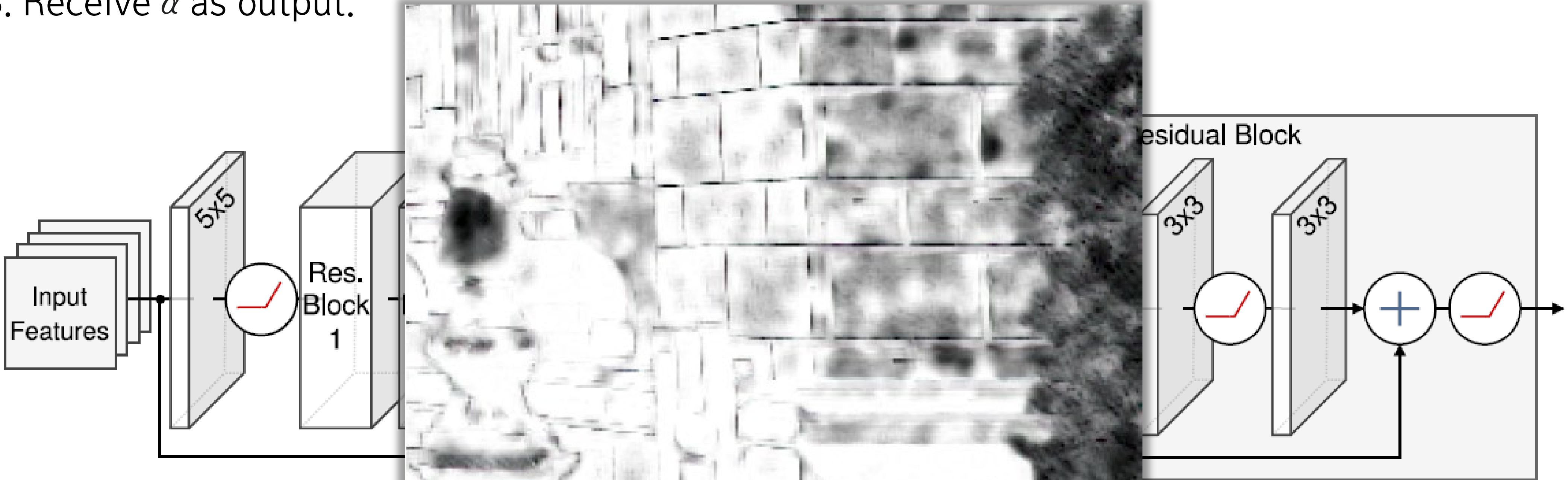


Figure 6: *Overview of our network* … *denoised counterpart y using per-pixel mixing parameters* α'*. The initial 5×* … *l convolutional layer. ReLU activation follows all but the final convolution* … *de the rendered and denoised images, along with estimates of their error (S* … *of Vogels et al. [VRM* 18].*

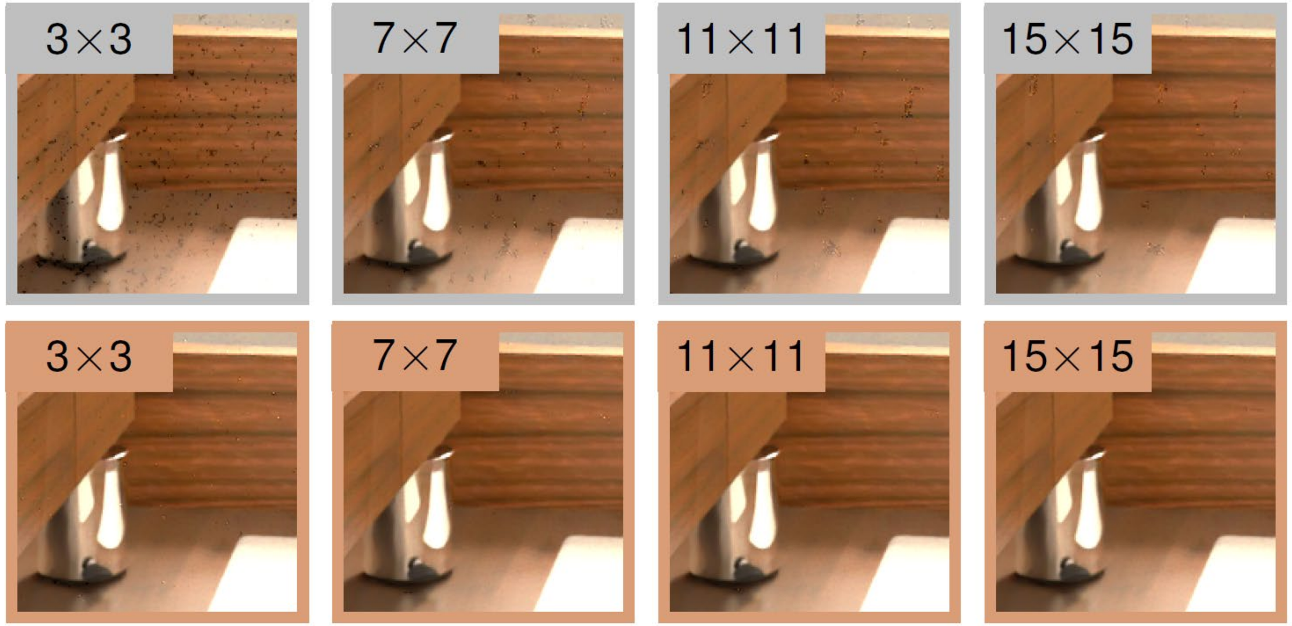
T-statistics

4. Rescale α with t-statistics.

$$t_p = \frac{\bar{z}_p - \bar{x}_p}{\sqrt{\mathrm{Var}[\bar{x}_p] + \varepsilon}}$$

x_p: averages around pixel p in rendered image
z_p: averages around pixel p in mixed image

Large t_p → Decrease alpha

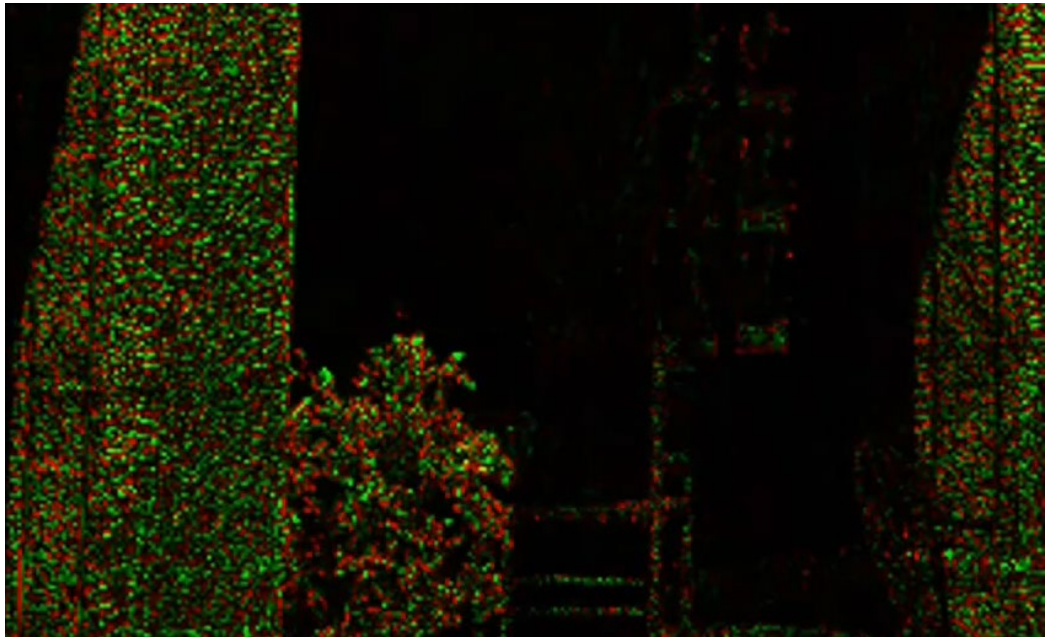


Results

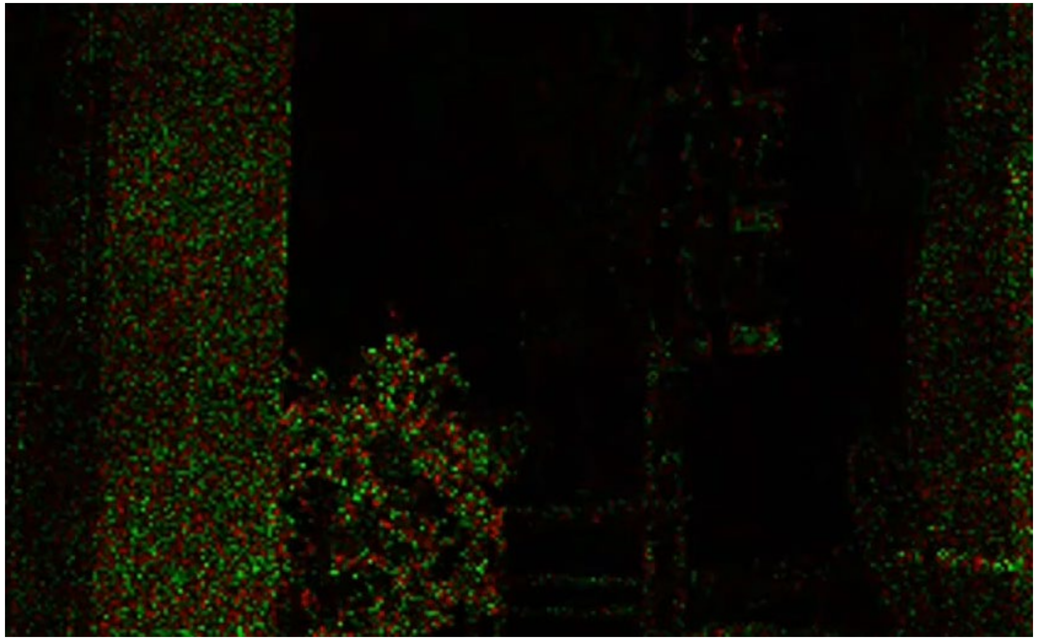
Denoised Image

Mixed Image

Results

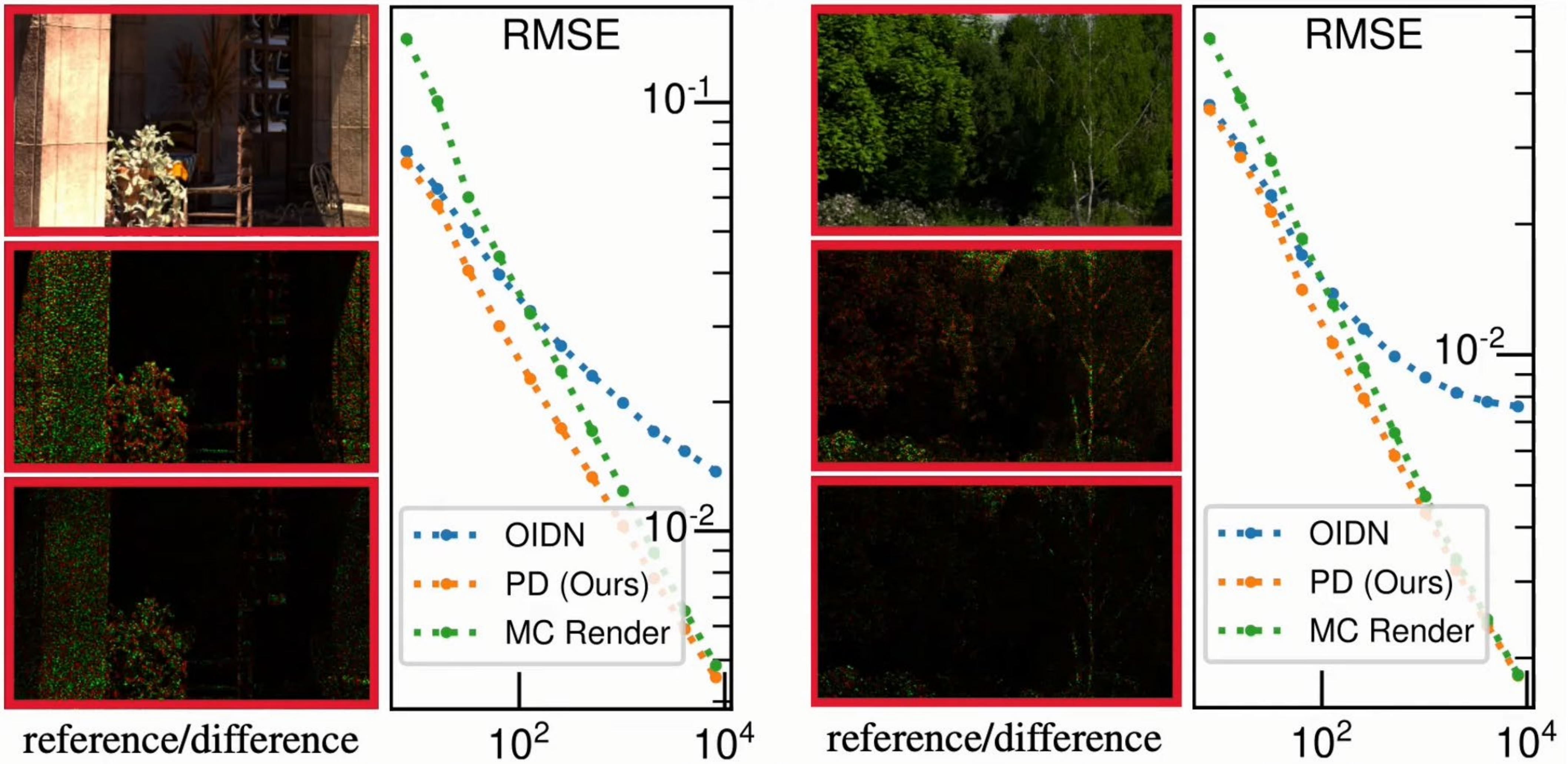


Denoised Image



Mixed Image

Results

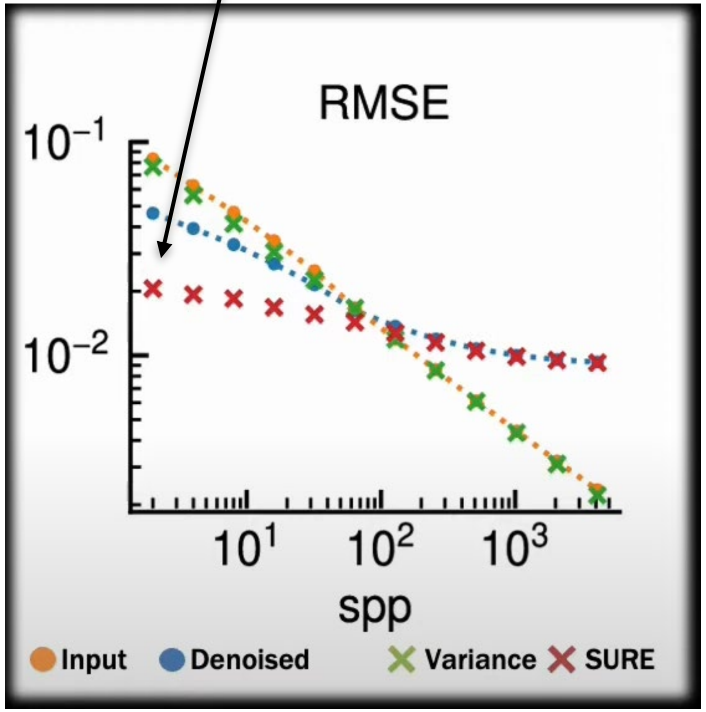


RMSE
10^{-1}
10^{-2}
OIDN
PD (Ours)
MC Render
reference/difference
10^2
10^4
RMSE
10^{-2}
OIDN
PD (Ours)
MC Render
reference/difference
10^2
10^4

Limitations

Figure 13: *Limitation of our method at very low sample counts, 2spp in this example, arising from insufficiently accurate sample variance estimates.*

This part



Quiz! :D

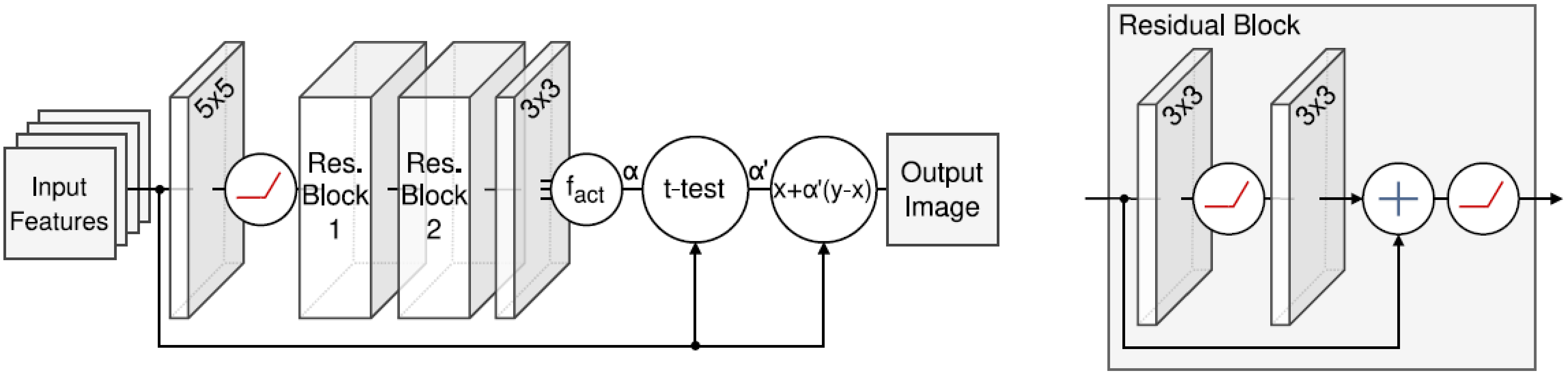


Figure 6: *Overview of our network $h_{NN}(x, y, \ldots; \Theta_h)$ for mixing an unbiased MC rendering x with its denoised counterpart y using per-pixel mixing parameters α'. The initial 5×5 convolutional layer is followed by two residual blocks and a final convolutional layer. ReLU activation follows all but the final convolution, which uses f_{act} (see Sections 4.1 and 4.3). Input features include the rendered and denoised images, along with estimates of their error (Section 4.3). The use of residual blocks is inspired by the networks of Vogels et al. [VRM*18].*

$$
\begin{aligned}
\bar{x}_p &= x_{M_p} + \sum_{q \in \mathcal{N}_p} \kappa_{p,q} x_q, && (12) \\
\mathrm{Var}[\bar{x}_p] &= \mathrm{Var}[x_{M_p}] + \sum_{q \in \mathcal{N}_p} \kappa_{p,q}^2 Var[x_q] && (13) \\
\bar{z}_p &= z_{M_p} + \sum_{q \in \mathcal{N}_p} \kappa_{p,q} (x_q + \alpha_q (y_q - x_q)) && (14) \\
\kappa_{p,q} &= \frac{1}{\sigma_s \sqrt{2\pi}} \exp\left(- \frac{\|p-q\|^2}{2\sigma_s^2} \right) && (15) \\
M_p &= \arg\max_{q \in \mathcal{N}_p} \mathrm{Var}[x_q]. && (16)
\end{aligned}
$$

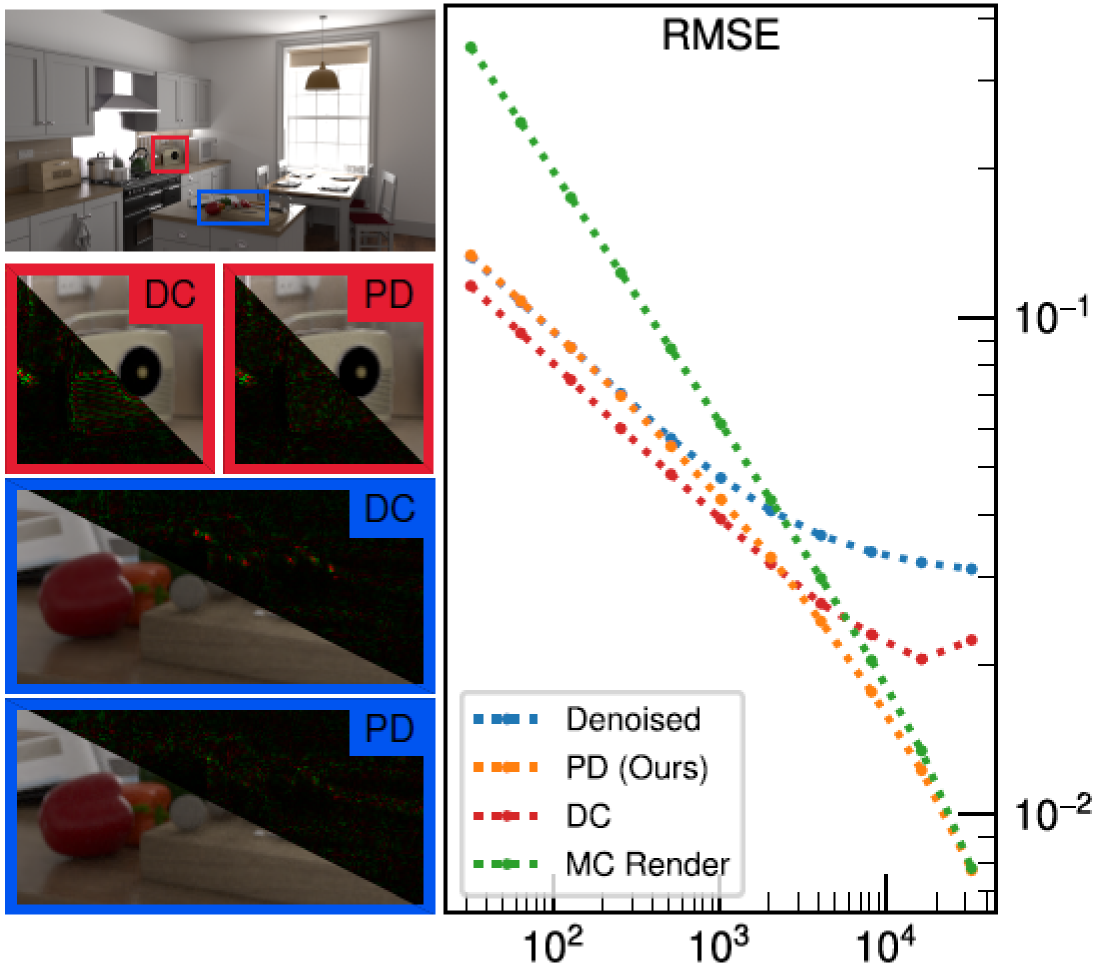


Figure 11: *Comparison of our method, Progressive Denoising (PD), with Deep Combiner (DC), kitchen test scene. While DC continually improves upon the denoised image, our method performs the best as the quality of the MC rendered input increases.*

inputs	model	error		
		RMSE	SMAPE	FLIP
HDR	den	0.1964	0.0344	0.0816
	pro-den	0.0784	0.0284	0.0726
HDR, VAR	den	0.1022	0.0292	0.0736
	pro-den	0.0587	**0.0277**	**0.0713**
HDR, ALB, NRM	den	0.1247	0.0393	0.0947
	pro-den	**0.0523**	0.0298	0.0789
HDR, ALB, NRM, VAR	den	0.0971	0.0326	0.0838
	pro-den	0.0536	0.0278	0.0742
	mc-render	0.0841	0.0669	0.1049

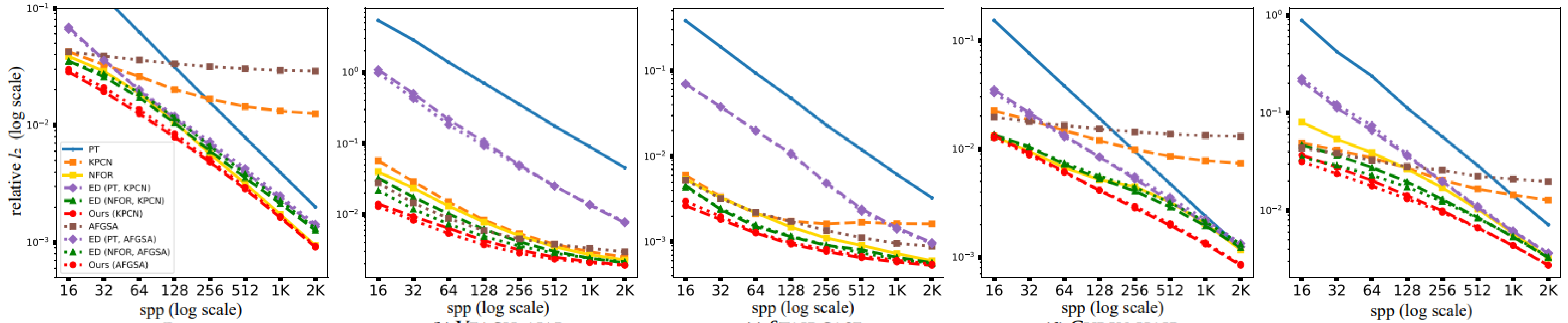
Table 1: *Comparison of our approach (pro-den) with simple denoising (den) for different input feature combinations.*

inputs	model	error		
		RMSE	SMAPE	FLIP
HDR, ALB, NRM, VAR	den-kpcn	0.1342	0.0362	0.0990
	pro-den	**0.0640**	**0.0285**	**0.0788**
	mc-render	0.0841	0.0669	0.1049

Table 2: *Our approach (pro-den) applied to a denoiser based on a kernel predicting network (den-kpcn). The improvement is similar as to when applied to the U-Net based denoisers of Table 1.*

inputs	model	error		
		RMSE	SMAPE	FLIP
HDR	oidn	0.0499	0.0278	0.0735
	pro-den	**0.0394**	0.0251	**0.0685**
HDR, ALB, NRM	oidn	0.0586	0.0265	0.0743
	pro-den	0.0489	**0.0248**	0.0705
	mc-render	0.0841	0.0669	0.1049

Table 3: *Applying our method (pro-den) to a pre-trained denoiser, Intel Open Image Denoise (oidn). Despite the already high-quality of OIDN, our method is still able to lower the overall error.*



(a) DRAGON (b) VEACH-AJAR (c) STAIRCASE (d) CURLY-HAIR (e) GLASS-OF-WATER

Fig. 13. The ED, which takes a pair of unbiased and biased images, i.e., ED (PT, KPCN) and ED (PT, AFGSA), shows much higher errors than the other biased results, including ours. This MSE-based method can be more robust when it takes only biased inputs, i.e., ED (NFOR, KPCN) and ED (NFOR, AFGSA), but produces higher errors than its input NFOR for the DRAGON (from 256 to 2K spp) and CURLY-HAIR (from 16 to 128 spp and 2K spp) scenes. Our technique, however, robustly improves our input denoisers and shows the best errors over the tested ranges.