Path to Neural Networks II

Image courtesy Vogel et al. [2018], Gharbi et al. [2019]
Today's Menu

Sample-based denoising

CNN-based approach to generate blue-noise samples

Normalizing Flows

Path guiding using Normalizing Flows
Recap
(a) Input MC (8 spp)  (b) Dependency on \((u, v)\)  (c) Our approach (RPF)
Bilateral Filtering

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

\[ W_p = \sum_{q \in S} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(I_p - I_q) \]

Bilateral filter weights at the central pixel

Spatial weight

Range weight

Multiplication of range and spatial weights

(a) Screen position  (b) Random parameters  (c) World space coords.  (d) Surface normals  (e) Texture value  (f) Sample color

Realistic Image Synthesis SS2021
Bilateral Filtering of Features

\[ w_{ij} = \exp\left[- \frac{1}{2\sigma_p^2} \sum_{1 \leq k \leq 2} (\bar{p}_{i,k} - \bar{p}_{j,k})^2 \right] \times \]

\[ \exp\left[- \frac{1}{2\sigma_c^2} \sum_{1 \leq k \leq 3} \alpha_k (\bar{c}_{i,k} - \bar{c}_{j,k})^2 \right] \times \]

\[ \exp\left[- \frac{1}{2\sigma_f^2} \sum_{1 \leq k \leq m} \beta_k (\bar{f}_{i,k} - \bar{f}_{j,k})^2 \right], \]
Multi-layer Perceptron

\[ y_1 = f(x_1 (w_{11} + w_{10})) \]
\[ y_2 = f(x_1 (w_{21} + w_{20})) \]
\[ y_3 = f(x_1 (w_{31} + w_{30})) \]
Multi-layer Perceptron

\[ y_1 = f(x_1 w_{11} + w_{10}) \]
\[ y_2 = f(x_1 w_{21} + w_{20}) \]
\[ y_3 = f(x_1 w_{31} + w_{30}) \]
Filter weights

For cross Bilateral filters:

\[
d_{i,j} = \exp\left[ -\frac{\|\bar{p}_i - \bar{p}_j\|^2}{2\alpha^2_i} \right] \times \exp\left[ -\frac{D(c_i, c_j)}{2\beta^2_i} \right] \times \prod_{k=1}^{K} \exp\left[ -\frac{D_k(f_{i,k}, f_{j,k})}{2\gamma^2_{k,i}} \right],
\]

Pixel screen coordinates

Mean sample color value

Scene features

(a) Screen position  (b) Random parameters  (c) World space coords.  (d) Surface normals  (e) Texture value  (f) Sample color
Our result with a cross-bilateral filter (4 spp)
Overview on Convolutional Neural Networks (CNNs)

Image Courtesy: Mathworks (online tutorial)
Multi-layer Perceptron vs. CNNs

Multi-layer perceptron

- All nodes are fully connected in all layers.
- In theory, should be able to achieve good quality results in small number of layers.
- Number of weights to be learnt are very high.

CNNs

- Weights are shared across layers.
- Requires significant number of layers to capture all the features (e.g. Deep CNNs).
- Relatively small number of weights required.
Kernel-Predicting Networks for Denoising Monte-Carlo Renderings
Recurrent AutoEncoder for Interactive Reconstruction

Fig. 2. Architecture of our recurrent autoencoder. The input is 7 scalar values per pixel (noisy RGB, normal vector, depth, roughness). Each encoder stage has a convolution and $2 \times 2$ max pooling. A decoder stage applies a $2 \times 2$ nearest neighbor upsampling, concatenates the per-pixel feature maps from a skip connection (the spatial resolutions agree), and applies two sets of convolution and pooling. All convolutions have a $3 \times 3$-pixel spatial support. On the right we visualize the internal structure of the recurrent RCNN connections. $I$ is the new input and $h$ refers to the hidden, recurrent state that persists between animation frames.
Recurrent Neural Networks vs. Simple Feed-Forward NN
Loss Functions

Spatial Loss to emphasize more the dark regions

\[ L_s = \frac{1}{N} \sum_{i}^{N} |P_i - T_i| \]

Temporal loss

\[ L_t = \frac{1}{N} \sum_{i}^{N} \left( \left| \frac{\partial P_i}{\partial t} - \frac{\partial T_i}{\partial t} \right| \right) \]

High frequency error norm loss for stable edges

\[ L_g = \frac{1}{N} \sum_{i}^{N} |\nabla P_i - \nabla T_i| \]

Final Loss is a weighted averaged of above losses

\[ L = w_s L_s + w_g L_g + w_t L_t \]
# Learnable Parameters?

How to compute "learnable" parameters?
How to compute "learnable" parameters?
How to compute "learnable" parameters?
Feed-Forward Neural Network

Image Source: towards-data-science
Feed-Forward Neural Network

(3 \times 5) + (5 \times 2) + (5 + 2) = 17 \text{ parameters}\\ weights \quad biases

Image Source: towards-data-science
Feed-Forward Neural Network

Image Source: towards-data-science
Pixel-space

Kernel Predicting

Denoising

#Learnable Parameters?

Sample-based

MC Denoising
Sample-based Denoising Network

Michael Gharbi, Tzu-Mao Li, Miika Aittala, Jakko Lehtinen, Fredo Durand

SIGGRAPH 2019
Multimodal distribution of sample features
Multimodal distribution of sample features

Input 16spp

Moving sphere

Background

Depth histogram
Multimodal distribution of sample features

Input 16spp

Inset 16spp

Reference

Sen [2012]

Proposed

Depth histogram

Moving sphere

Background

Realistic Image Synthesis SS2021
Reconstruction: Kernel Gather
Reconstruction: Kernel Gather
Reconstruction: Kernel Gather
Reconstruction: Kernel Gather

Kernel gather

2D example

*How should nearby samples influence me?*
Reconstruction: Kernel Splatting

How should nearby samples influence me?
Reconstruction: Kernel Splatting

How should nearby samples influence me?
Reconstruction: Kernel Splatting

How should nearby samples influence me?
Reconstruction: Kernel Splatting

How should nearby samples influence me?

How do I contribute to nearby pixels, given all the samples around me?

Continue splatting kernels for the rest of the samples....
Network: Kernel Gather vs Splatting

input  reference  gather  splat  gather (larger network)

A B A B A B A B A B

gather kernels  splat kernels  gather kernels
(large capacity network)
Permutation Invariance
Permutation Invariance

A model that produces the same output regardless of the order of elements in the input vector
Permutation Invariance: Example

* =
Permutation Invariance: Example
Permutation Invariance: Example

Not Permutation Invariance

\[ \star \Rightarrow \not= \]

\[ \star \Rightarrow \not= \]
Permutation Invariance: Example

Not Permutation Invariance
Permutation Invariance: Architectures

- A standard feedforward neural net such as multilayer perceptron (MLP) is insensitive to order of elements in input vector - so it is inherently permutation insensitive.

- However, both a Convnet and RNNs for instance make full use of input ordering - they are permutation sensitive.
Permutation Invariance: Example

MNIST Dataset

Permuting pixels makes it difficult for humans to understand the images.

However, permutation invariant networks like MLP can detect digits irrespective of the order of pixels.
A graph labeling function $F$ is graph permutation invariant (GPI) if permuting the names of nodes maintains the output. Herzig et al.[2018]
Permutation Invariance

- In MLPs, since each component is connected to each other, the order does not matter.

- In structured convolutions, the order matters and therefore, it is not permutation invariant.
Proposed Network Architecture
Dataset and Training Procedure

Procedurally generated dataset: 300,000 renderings with 128x128 resolution

Also generated input buffer (4, 32 spp), but this time also maintained auxiliary features

Reference was generated for 4096 samples
Splat vs Gather

Input

Reference

per sample gather

per sample splat

per pixel gather

per pixel splat

rMSE = 10.7

rMSE = 0.023

rMSE = 0.044

rMSE = 0.026

rMSE = 0.024

50
Results

input 4spp  [Bako 2017]  ours  ref. 8192spp
Network Architecture Comparisons

reference 8192spp input finetuned [Bako2017] ours reference 8192spp

32spp

16spp

Realistic Image Synthesis SS2021
(Deep) Convolutional Neural Networks
Based on Convolutional Neural Networks

Unstructured data

$\mathbf{x}_1$

$N$ number of point samples
Based on Convolutional Neural Networks

Unstructured data

Convolution

$N$ number of point samples
Based on Convolutional Neural Networks

Unstructured data

$\mathbf{x}_1$

Convolution

$N$ number of point samples
Based on Convolutional Neural Networks

Unstructured data

Convolution $\otimes$

...  

$N$ number of point samples

Loss function

Back-propagate
Based on Convolutional Neural Networks

Unstructured data

Convolution \( \otimes \)

\( N \) number of point samples
Based on Convolutional Neural Networks

Unstructured data

Convolutions

Keep training the network!

$N$ number of point samples
Based on Convolutional Neural Networks

Which Loss function can we use?

Keep training the network!

Unstructured data

Convolution

$N$ number of point samples
Spectral Loss Function

**Spectral Loss** at $i$-th training iteration

$$L_{\text{spectral}} = \left| \left| \langle P_i(\nu) \rangle - \langle P(\nu) \rangle \right| \right|^2$$

Radially averaged power
Training Process

Points

Power Spectrum

56x Slowdown

Kernels for BNOT (de Goes et al. [2012])

Kernels for Step (de Heck et al. [2013])
Architecture: Full pipeline
Results: Spectral Target Spectra

\[ l_1(\text{radSpec}(X), \text{BNOT}) \]
Results: Spectral Target Spectra

![Diagram showing spectral target and results with different constraints: l1(radSpec(X), BNOT) and l1(radSpec(X), Step).](image)
Results: Spectral Target Spectra

- **a)** Target vs. Result for $1l(\text{radSpec}(X), B\text{NOT})$
- **b)** Target vs. Result for $1l(\text{radSpec}(X), J\text{itter})$
- **c)** Target vs. Result for $1l(\text{radSpec}(X), \text{Step})$
- **d)** Target vs. Result for $1l(\text{radSpec}(X), Stair)$
Results: Spectral Target Spectra

- **(a)** Target and Result: $l^1(\text{radSpec}(X), \text{BNOT})$

- **(b)** Target and Result: $l^1(\text{radSpec}(X), \text{Jitter})$

- **(c)** Target and Result: $l^1(\text{radSpec}(X), \text{Step})$

- **(d)** Target and Result: $l^1(\text{radSpec}(X), \text{Stair})$

- **2D**

- **3D**

- **4D**

- **5D**

- **10D**
Spatial Loss Function

PCF Loss at $i$-th training iteration
Spatial Domain

Blue Noise

Samples

Histogram count

Distance
Loss Functions

**PCF Loss** at \( i \)-th training iteration

\[
L_{PCF} = \| \langle r_i(\text{dist}) \rangle - \langle r(\text{dist}) \rangle \|^2
\]
Spatial Target PCFs

g) $l_1(radDDom(X), \text{Step})$

$h) l_1(radDDom(X), \text{Stair})$

Spatial Target PCFs

I) Target

Result

Power

Distance

.125

II) Target

Result

Power

Distance

.125

Spatial Target PCFs

I) Target

Result

Power

Distance

.125
Realistic Image Synthesis SS2021

Spatial Target PCFs

1. **Spectral**
   - **a)** Target, Result
     - $\ell_1(\text{radSpec}(X), \text{BNOT})$
   - **b)** Target, Result
     - $\ell_1(\text{radSpec}(X), \text{Jitter})$
   - **c)** Target, Result
     - $\ell_1(\text{radSpec}(X), \text{Step})$
   - **d)** Target, Result
     - $\ell_1(\text{radSpec}(X), \text{Stair})$

2. **Differential**
   - **e)** Target, Result
     - $\ell_1(\text{radDDom}(X), \text{BNOT})$
   - **f)** Target, Result
     - $\ell_1(\text{radDDom}(X), \text{Jitter})$
   - **g)** Target, Result
     - $\ell_1(\text{radDDom}(X), \text{Step})$
   - **h)** Target, Result
     - $\ell_1(\text{radDDom}(X), \text{Stair})$
3D Point Samples (Different Projection Targets)
3D Point Samples (Different Projection Targets)

Target Spectra

XY (BNOT)

YZ (Jitter)

XZ (Step)
3D Point Samples (Different Projection Targets)

Target Spectra

XY (BNOT)

YZ (Jitter)

XZ (Step)

Our Spectra
Point set: Projections are Preserved

Target Pointset

Our PointSet

XY (BNOT)  

YZ (Jitter)  

XZ (Step)  

Novel Sampling Patterns

Leimkuhler et al. [SIGGRAPH Asia 2019]
Novel Sampling Patterns

\[
\text{Var}(I_N) = \sum_{\Omega} \times \text{Integrand Power spectrum}
\]

Leimkuhler et al. [SIGGRAPH Asia 2019]
Novel Sampling Patterns

\[
\text{Var}(I_N) = \sum_{\Omega} \times
\]

Leimkuhler et al. [SIGGRAPH Asia 2019]
Blue Noise Dithering

Leimkuhler et al. [SIGGRAPH Asia 2019]
Novel Sampling Patterns

using radially averaged loss

Function's power spectrum decay

Sampler's power radial profile

(a) 2D
Novel Sampling Patterns

using radially averaged loss

(a) 2D

(b) 3D

(c) 4D

using radially averaged loss
Blue Noise Dithering
Normalizing Flows
Importance Sampling

\[ I_N = \frac{1}{N} \sum_{k=1}^{N} \frac{f(x)}{p(x)} \]
Importance Sampling

\[ I_N = \frac{1}{N} \sum_{k=1}^{N} \frac{f(x)}{p(x)} \]

\[ p(x) = ??? \]
Normalizing Flows
Normalizing Flows

Technique used in Machine learning to build complex probability distributions by transforming simple ones

Used in the context of generative modeling

Generative modeling: learning without any target (unsupervised)
Complex Probability distributions from simple ones
Complex Probability distributions from simple ones
Normalizing Flows: Basic mathematical framework
Given a continuous variable with a distribution

\[ z \sim p_\theta(z) \]

\[ x = f_\theta(z) = f_k \circ f_2 \circ f_1(z) \]

New distribution obtained

each \( f_i \) is invertible (bijective)
Distributions
Distributions

\[ p(x) = ? \]

\[ f^{-1} \]

\[ f \]

\[ X \]

\[ Z \]
Distributions

\[ z \sim p_\theta(z) \quad \text{Given a continuous variable with a distribution} \]

\[ x = f_\theta(z) = f_k \circ f_2 \circ f_1(z) \]

Each \( f_i \) is invertible (bijective)

\[ p(x) \neq p(f^{-1}(x)) \]
Change of Variables

\( f : Z \to X, \) \( f \) is invertible

\( p(z) \) defined over \( z \in Z \)

Change of variable formula says that:

\[
p(x) = p(f^{-1}(x)) \left| \det \left( \frac{\partial f^{-1}(x)}{\partial x} \right) \right|
\]
Change of Variables

$f : Z \to X$, $f$ is invertible

$p(z)$ defined over $z \in Z$

\[ p(x) = p(f^{-1}(x)) \left| \det \left( \frac{\partial f^{-1}(x)}{\partial x} \right) \right| \]

\[ p(x) = p(z) \left| \det \left( \frac{\partial z}{\partial x} \right) \right| \]
Jacobian Matrix

\[ f : \mathbb{R}^n \rightarrow \mathbb{R}^m \]

\[
J = \begin{bmatrix}
\frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\
\frac{\partial f_2}{\partial x_1} & \cdots & \frac{\partial f_2}{\partial x_n} \\
\vdots & \ddots & \vdots \\
\frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n}
\end{bmatrix}
\]
Jacobian Matrix

\[ f : \mathbb{R}^2 \rightarrow \mathbb{R}^2 \]

Jacobian determinant gives the ratio of the area of the approximating parallelogram to that of the original square.
Jacobian Matrix

\[ f : Z \rightarrow X, f \text{ is invertible} \]
\[ p(z) \text{ defined over } z \in Z \]

\[ p(x) = p(f^{-1}(x)) \left| \det \left( \frac{\partial f^{-1}(x)}{\partial x} \right) \right| \]

\[ p(x) = p(z) \left| \det \left( \frac{\partial z}{\partial x} \right) \right| \]
Invertible and Differentiable Mapping

\[ f : Z \to X, \text{ } f \text{ is invertible} \]

\[ p(z) \text{ defined over } z \in Z \]

\[ \det \left( \frac{\partial x}{\partial z} \right) = 4 \]

\[ p(x) = p(z) = \left| \frac{1}{\det \left( \frac{\partial x}{\partial z} \right)} \right| \]
Maximize Log-likelihood

\[ f : Z \rightarrow X, f \text{ is invertible} \]
\[ p(z) \text{ defined over } z \in Z \]

\[ \log p(x) = \log p(z) + \log \left| \det \left( \frac{\partial f^{-1}(x)}{\partial x} \right) \right| \]
Maximize Log-likelihood

\[ f : Z \rightarrow X, \text{ } f \text{ is invertible} \]
\[ p(z) \text{ defined over } z \in Z \]

\[ \log p(x) = \log p(z) + \log \left| \det \left( \frac{\partial f^{-1}(x)}{\partial x} \right) \right| \]

\[ \log p(x) = \log p(z) + \sum_{i=1}^{K} \log \left| \det \left( \frac{\partial f^{-1}(x)}{\partial x} \right) \right| \]
Jacobian: Lower Triangular Matrix

\( f : Z \rightarrow X, f \) is invertible
\( p(z) \) defined over \( z \in Z \)

\[
\begin{bmatrix}
\ell_{1,1} & & & & \\
\ell_{2,1} & \ell_{2,2} & & & \\
\ell_{3,1} & \ell_{3,2} & \ddots & & \\
& \ddots & \ddots & \ddots & \\
\ell_{n,1} & \ell_{n,2} & \cdots & \ell_{n,n-1} & \ell_{n,n}
\end{bmatrix}
\]
How to ensure lower-triangular Jacobian matrix?

\[ z \in \mathbb{R}^D \]

\[ z_{1:d} \]

\[ z_{d+1:D} \]
How to ensure lower-triangular Jacobian matrix?

Coupling layer

\[ x_{1:d} \quad \Rightarrow \quad g \quad \Rightarrow \quad z_{1:d} \]

\[ x_{d+1:D} \]

\[ \begin{align*}
    z & \in \mathbb{R}^D \\
    m
\end{align*} \]
Neural Importance Sampling

Thomas Müller
Brian McWilliams
Fabrice Rousselle
Markus Gross
Jan Novák

Affiliation:

Work done while at:
What is Light Transport?
Render time: sometimes >100 cpu-hours
What is Light Transport?
What is Light Transport?
Path tracing

Neural path guiding

512 paths per pixel
Path tracing: BSDF sampling
Path tracing: direct-illumination sampling

Multiple Importance Sampling
[Veach and Guibas 1995]
Where is path guiding useful?
Where is path guiding useful?

**Goal:** Sample proportional to incident radiance.
Where is path guiding useful?
Learning incident radiance in a Cornell box
Neural networks as function approximators

**Reference**

SD-tree [Müller et al. 2017]  
Neural Network  
GMM [Vorba et al. 2014]
Neural path guiding overview

Path tracer → Feedback loop → Neural network

Sample

Optimize
Neural path guiding overview
How to draw samples?

Path tracer → Sample → Feedback loop → Optimize → Neural network
Goal: warp random numbers to good distribution with NN

\[ F \approx \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)} \]

Monte Carlo estimator

Need \( p \) in closed form!
Addressed by "normalizing flows"
Parameterizing a bijection allows using the change-of-variable formula

\[ p(x) = p(z) \cdot \left| \det \left( \frac{\partial m(z)}{\partial z^T} \right) \right|^{-1} \]
A chain of simple bijections can model complicated functions

\[ p(x) = p(z) \cdot \prod_{i=1}^{L} \left| \text{det} \left( \frac{\partial m_i(z)}{\partial z^T} \right) \right|^{-1} \]

Our choice, e.g. Gaussian

Squishing/stretching by \( m \)

[Dinh et al. 2016]
How to optimize?
Training with data from the correct distribution is simple

\[ \nabla_\theta \log p(x; \theta) \]

Desired distribution

Neural network

Training data

Optimize
Training from Monte Carlo samples requires careful weighting.

Optimize

\[ \nabla_{\theta} \log p(x; \theta) \]

Arbitrarily distributed

Training data

Neural network
Training with data from the correct distribution is simple

\[ \nabla_{\theta} \log p(x; \theta) \]

min KL-divergence

Desired distribution

Optimize

Neural network

Training data
Training from Monte Carlo samples requires careful weighting

\[ f(x) \quad \frac{\nabla_\theta \log p(x; \theta)}{p(x; \theta)} \]

\[ \min \text{KL-divergence} \]

\[ p(x; \theta) \text{ distributed} \]

Training data

Optimize

Neural network
Training from Monte Carlo samples requires careful weighting

\[
f(x)^2 \frac{\nabla_\theta \log p(x; \theta)}{p(x; \theta)^2} \text{ min variance}
\]

\( p(x; \theta) \text{ distributed} \)
Training from Monte Carlo samples requires careful weighting

\[ f(x)^2 \frac{\nabla_{\theta} \log p(x; \theta)}{p(x; \theta)^2} \min \chi^2 \text{-divergence} \]
Putting it together...
1 path per pixel

Path tracing  Neural path guiding
2 paths per pixel

Path tracing

Neural path guiding
4 paths per pixel

Path tracing

Neural path guiding
8 paths per pixel

Path tracing

Neural path guiding
16 paths per pixel

Path tracing

Neural path guiding
32 paths per pixel

Path tracing

Neural path guiding
64 paths per pixel

Path tracing

Neural path guiding
128 paths per pixel

Path tracing

Neural path guiding
256 paths per pixel

Path tracing

Neural path guiding
512 paths per pixel

Path tracing

Neural path guiding
Product guiding
Product path guiding

\[ L_r(x, \omega_o) = \int L_i(x, \omega_i) f(x, \omega_i, \omega_o) \cos \theta \, d\omega_i \]
Product path guiding

\[ L_r(x, \omega_o) = \int L_i(x, \omega_i) f(x, \omega_i, \omega_o) \cos \theta \, d\omega_i \]
Product path guiding

\[ L_r(x, \omega_o) = \int L_i(x, \omega_i) f(x, \omega_i, \omega_o) \cos \theta \, d\omega_i \]
Product path guiding

\[ L_r(x, \omega_o) = \int L_i(x, \omega_i) f(x, \omega_i, \omega_o) \cos \theta \, d\omega_i \]
Product path guiding

\[ L_r(x, \omega_o) = \int L_i(x, \omega_i) f(x, \omega_i, \omega_o) \cos \theta \, d\omega_i \]
Product path guiding

\[ L_r(x, \omega_o) = \int L_i(x, \omega_i) f(x, \omega_i, \omega_o) \cos \theta \, d\omega_i \]
$L_r(x, \omega_o) = \int L_i(x, \omega_i) f(x, \omega_i, \omega_o) \cos \theta \, d\omega_i$
Product path guiding

\[ L_\text{r}(x, \omega_o) = \int L_i(x, \omega_i) f(x, \omega_i, \omega_o) \cos \theta \, d\omega_i \]
MIS optimization
MIS-aware optimization

\[ p(\omega_i; \theta) = (1 - w) + w \]

Learned distribution

BSDF

X
MIS-aware optimization

\[ p(\omega_i; \theta) = (1 - w) + w \]

Learned distribution

BSDF
MIS-aware optimization

\[ p(\omega_i; \theta) = \left(1 - w(\theta)\right) + w(\theta) \]

Learned distribution

BSDF
Results
Equal time

Path tracing

Müller et al. [2017]

Neural path guiding
Equal spp

Path tracing
Müller et al. [2017]
Neural path guiding
Path tracing

Müller et al. [2017]

Neural path guiding
Conclusion

• Neural networks can drive unbiased MC integration

• Complicated integrands (e.g. product path guiding)

• Computational cost of neural path guiding is high, but quality is state of the art
References

A frequency analysis of light transport, Durand et al. SIGGRAPH 2005

Frequency Analysis and Sheared Reconstruction for Rendering Motion Blur, Egan et al. SIGGRAPH 2009

Temporal Light Field Reconstruction for Rendering Distribution Effects, Lehtinen et al. SIGGRAPH 2011

On Filtering the Noise from the Random Parameters in Monte Carlo Rendering, Sen and Darabi 2012

A Machine Learning Approach for Filtering Monte Carlo Noise, Kalantari et al. SIGGRAPH 2015

Interactive Reconstruction of Monte Carlo Image Sequences using a Recurrent Denoising Autoencoder, Chaitanya et al. SIGGRAPH 2017

Kernel-Predicting Convolutional Networks for Denoising Monte Carlo Renderings, Bako et al. SIGGRAPH 2017

Sample-based Monte Carlo Denoising using a Kernel-Splatting Network, Gharbi et al. SIGGRAPH 2019

NICE: Non-linear Independent Components Estimation

Normalizing Flows: An Introduction and Review of Current Methods

Neural Importance Sampling SIGGRAPH 2019
I would like to thank Thomas Muller and colleagues to make their slides available online.