Neural Importance Sampling

by Müller et al., 2018

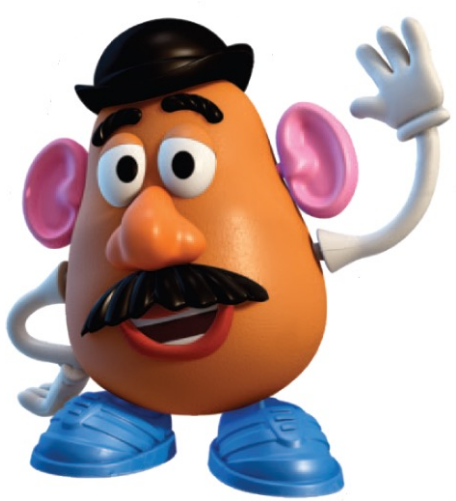
Jesus Solano and Federico D'Agostino

Machine Learning Seminar – Paper Presentation

University College London

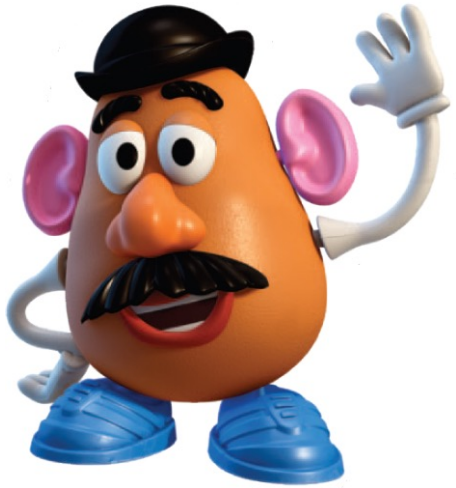
March 2022

What do these two have in common?



$$I = \int_{\mathcal{P}} L_{\mathrm{e}}(\mathbf{x}_0, \mathbf{x}_1) T(\bar{x}) W(\mathbf{x}_{k-1}, \mathbf{x}_k) \mathrm{d}\bar{x}$$

What do these two have in common?



$$I = \int_{\mathcal{P}} L_{\mathrm{e}}(\mathbf{x}_0, \mathbf{x}_1) T(\bar{x}) W(\mathbf{x}_{k-1}, \mathbf{x}_k) \mathrm{d}\bar{x}$$

And why does **Disney** need a ML research division?

… Let's first back up a bit to answer this

Contents

- What's in an animated motion picture movie?
- The problem of integration
- Integrating with neural networks: NICE (***Nonlinear Independent Components Estimation***, Dinh et al. 2014)
- Extending NICE: Neural Importance Sampling
- Experiments and applications to rendering
- Evaluation and outlook

Nonlinear Independent Components Estimation
Dinh, Krueger and Bengio, 2014

How are cartoons made?

Pre-production → Modelling → Texturing → … → **Rendering** → Post-production

Light-Transport simulation

- One of the main challenges in computer graphics

Path Sampling and Path Guiding

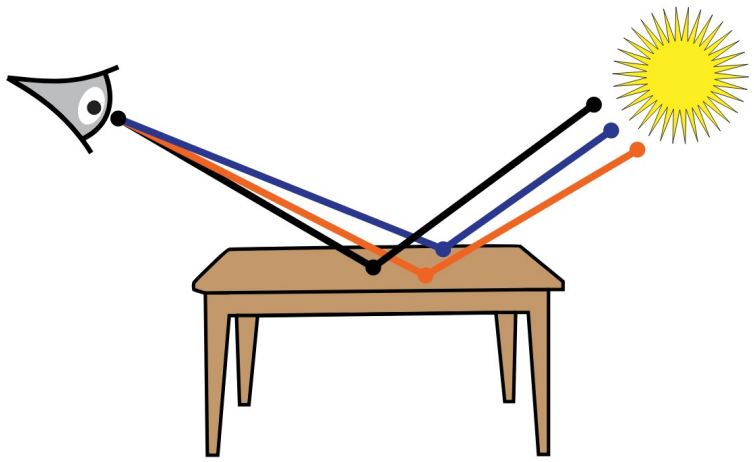
Reduce estimation variance by "guiding" the construction of light paths

Path Sampling

Path Guiding

Primary-Sample-Space Path Sampling

GOAL → Estimate amount of light reaching the camera after taking **ANY** of the possible paths



Approach → Integrating over all paths in the scene

Path Throughput

Camera Sensor response

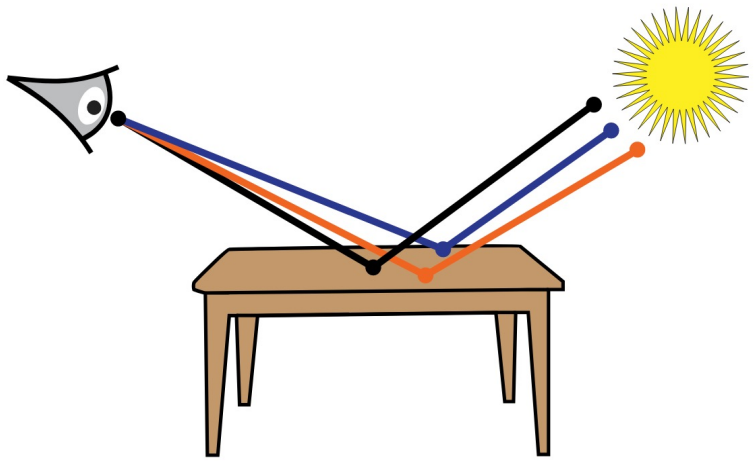
$$I = \int_{\mathcal{P}} L_e(\mathbf{x}_0, \mathbf{x}_1) T(\bar{x}) W(\mathbf{x}_{k-1}, \mathbf{x}_k) \mathrm{d}\bar{x}$$

Emitted Radiance

Chain of positions of a **single light path** with k vertices

Primary-Sample-Space Path Sampling

GOAL → Estimate amount of light reaching the camera after taking **ANY** of the possible paths



Approach → Integrating over all paths in the scene

$$I = \int_{\mathcal{P}} L_{\mathrm{e}}(\mathbf{x}_0, \mathbf{x}_1) T(\bar{x}) W(\mathbf{x}_{k-1}, \mathbf{x}_k) \mathrm{d}\bar{x}$$

How → Use Monte Carlo Integration *

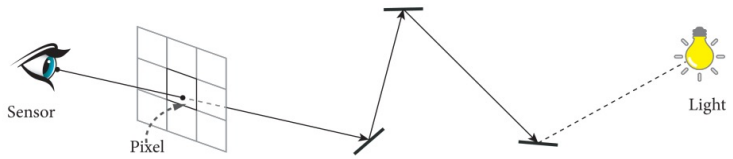
$$\langle I \rangle = \frac{1}{N} \sum_{j=1}^{N} \frac{L_{\mathrm{e}}(\mathbf{x}_{j0}, \mathbf{x}_{j1}) T(\bar{x}_j) W(\mathbf{x}_{jk-1}, \mathbf{x}_{jk})}{q(\bar{x}_j)}$$

→ Joint probability density of generating all k vertices of a path

Path Guiding

GOAL → Estimate radiative equilibrium of every surface point *x* in the direction ω_0

Approach → Integrating over all possible reflecting angles



Emitted Radiance

Camera Sensor response

$$L_o(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \int_\Omega L(\mathbf{x}, \omega) f_s(\mathbf{x}, \omega_o, \omega) |\cos \gamma| \, d\omega$$

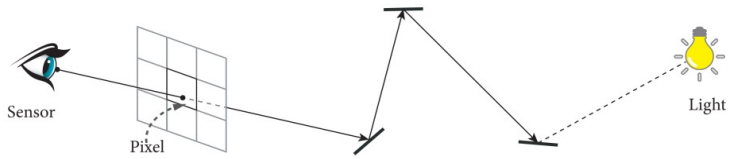
Reflected Radiance

Incident Radiance

Angle between w and surface normal

Path Guiding

GOAL → Estimate radiative equilibrium of every surface point x in the direction w_o



Approach → Integrating over all possible reflecting angles

$$L_{\mathrm{o}}(\mathbf{x}, \omega_{\mathrm{o}}) = L_{\mathrm{e}}(\mathbf{x}, \omega_{\mathrm{o}}) + \int_{\Omega} L(\mathbf{x}, \omega) f_{\mathrm{s}}(\mathbf{x}, \omega_{\mathrm{o}}, \omega) |\cos \gamma| \mathrm{d}\omega$$

How? → Use Monte Carlo Integration *

$$\langle L_{\mathrm{r}}(\mathbf{x}, \omega_{\mathrm{o}}) \rangle = \frac{1}{N} \sum_{j=1}^{N} \frac{L(\mathbf{x}, \omega_j) f_{\mathrm{s}}(\mathbf{x}, \omega_{\mathrm{o}}, \omega_j) |\cos \gamma_j|}{q(\omega_j \mid \mathbf{x}, \omega_{\mathrm{o}})}$$

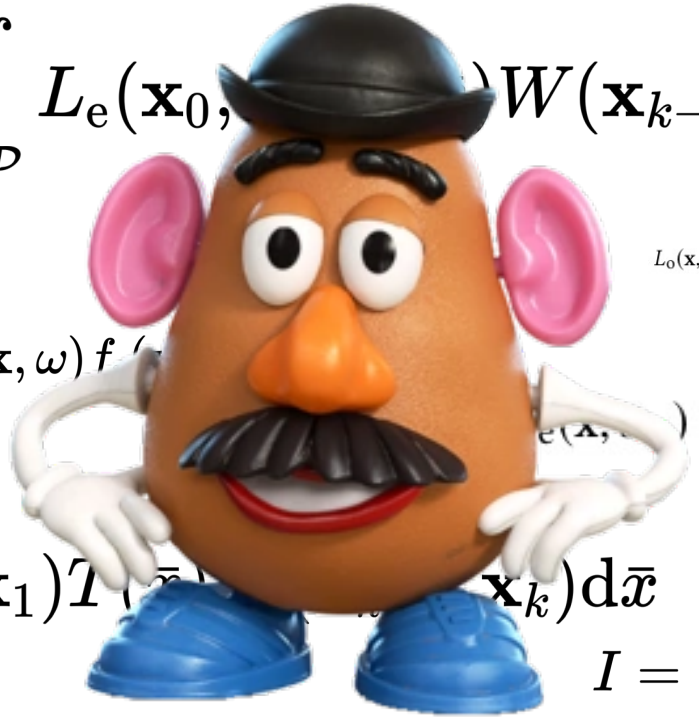
→ Density conditioned on position and angle (>5D)

In summary...

$$L_o(\mathbf{x}, \omega_o, \lambda, t) = L_e(\mathbf{x}, \omega_o, \lambda, t) + \int_\Omega f_r(\mathbf{x}, \omega_i, \omega_o, \lambda, t) L_i(\mathbf{x}, \omega_i, \lambda, t)(\omega_i \cdot \mathbf{n}) \mathrm{d}\omega_i$$

$$I = \int_{\mathcal{P}} L_e(\mathbf{x}_0, \ldots) W(\mathbf{x}_{k-1}, \mathbf{x}_k) \mathrm{d}\bar{x}$$

$$L_o(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \int_\Omega L(\mathbf{x}, \omega) f_s(\mathbf{x}, \omega_o, \omega) |\cos \gamma| \, \mathrm{d}\omega$$



$$I = \int_{\mathcal{P}} L_e(\mathbf{x}_0, \mathbf{x}_1) T(\bar{x}) W(\mathbf{x}_{k-1}, \mathbf{x}_k) \mathrm{d}\bar{x}$$

Integrals everywhere!

The problem of integration

- Recall from previous lectures: integration is everywhere! (unfortunately)
  - Not only in ML!
- Analytical solutions may be intractable or non-existent
- Dimensionality only adds to the problem
  - In any strategy we previously encountered as the dimensionality of the integrand increases the approximation deteriorates
- Some approximation strategies are situation-specific
  - We can work our way around that…

Background:
MC with importance sampling

- The usual **importance sampling scenario**:

$$\mathbb{E}[f(x)] = \int f(x)p(x)dx = \int f(x)\frac{p(x)}{q(x)}q(x)dx \approx \frac{1}{N}\sum_{i=1}^{N} f(X_i)\frac{p(X_i)}{q(X_i)}$$

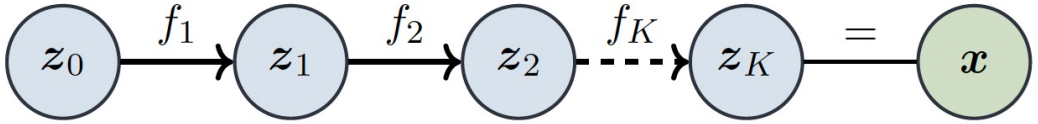
Where $f(x)$ follows the distribution $p(x)$, and $q(x)$ is a proposal distribution.

- **Our scenario:**

$$F = \int_{\mathcal{D}} f(x)\mathrm{d}x = \int_{\mathcal{D}} \frac{f(x)}{q(x)}q(x)\mathrm{d}x = \mathbb{E}\left[\frac{f(X)}{q(X)}\right] \approx \langle F\rangle_N = \frac{1}{N}\sum_{i=1}^{N}\frac{f(X_i)}{q(X_i)}$$

Background: Normalizing Flows (NFs)

- Normalizing flows:



with $z_0 \sim p_0$, a “simple” base distribution and $z_k = f_k(z_{k-1})$

Through the **change of variables trick**, we can express this in terms of probability distributions (*where we swapped f with h to be consistent with our previous notation*):

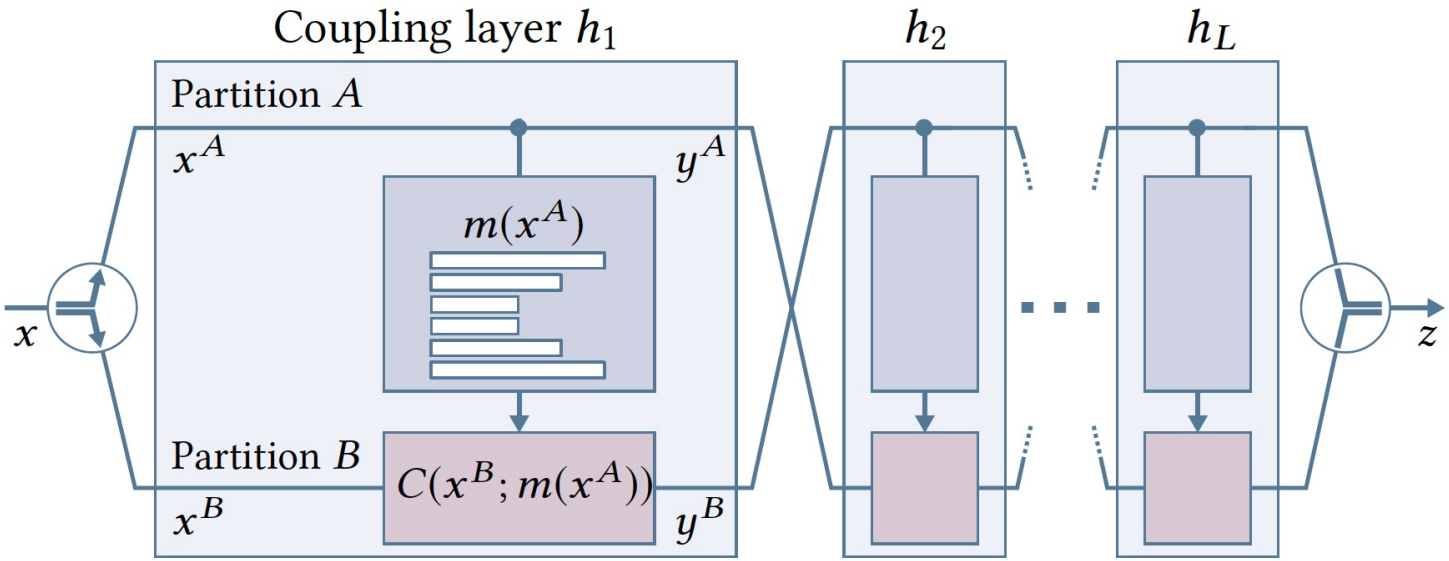
$$q(x;\theta) = \int \delta(x - h(z;\theta))q(z)\mathrm{d}z = q(z)\left|\det\left(\frac{\partial h(z;\theta)}{\partial z^T}\right)\right|^{-1}$$

What do we want from an ideal integration algorithm?

- A parametric proposal distribution
  - Additional MC over q introduces bias
- Applicability to high dimensionality
  - i.e. an expressive proposal
- Efficient sampling and evaluation of the proposal
- If using NFs, a mapping that is easily invertible
  - i.e. whose Jacobian determinant can be computed efficiently

MC Integration with NICE: Neural Importance Sampling

$$\langle F \rangle_N = \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{q(X_i;\theta)} = \frac{1}{N} \sum_{i=1}^{N} \frac{p(X_i)F}{q(X_i;\theta)}$$



Let's break this down component by component

Coupling Layer
The trick to add Neural Networks

The **cost** of computing the determinant grows **super-linearly**

What can be done to reduce the computational cost? → Use transformations whose Jacobian is diagonal, lower **triangular** or upper **triangular**.

How to use this observation in Normalizing Flows and MC integration?

1. Neural network with triangular weight matrices and bijective activation functions → **BUT** this will constrain too much the possible architectures!

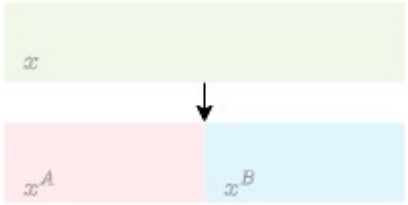
2. **Coupling Layers** → Family of functions with **triangular Jacobian**

Coupling Layer

The trick to add Neural Networks

What is a **coupling layer**? → Specific family of transformations whose determinant reduce to the product of diagonal terms

1. Split the input vector into two **disjoint partitions** (x^A, x^B) of its dimensions →



2. Apply the coupling transform *C* to ONLY one of the partitions →

$$y^A = x^A$$
$$y^B = C\left(x^B; m\left(x^A\right)\right)$$

3. Build output of **coupling layer** by concatenation →

$$h(x) = y = \left(y^A, y^B\right)$$

Coupling Layer

The trick to add Neural Networks

Coupling Layer →

$$y^A = x^A$$
$$y^B = C\left(x^B; m\left(x^A\right)\right)$$

→

$$y = \left(y^A, y^B\right)$$
$$= h(x)$$

Trivial inversion of h:

$$x^A = y^A$$
$$x^B = C^{-1}\left(y^B; m\left(x^A\right)\right) = C^{-1}\left(y^B; m\left(y^A\right)\right)$$

What are the **basic** Coupling transforms?

1

Additive Coupling Transform

Translation of signal

$$C\left(x^B; t\right) = x^B + t$$

2

Multiply-Add Coupling Transform

Element-wise product plus translation

$$C\left(x^B; s, t\right) = x^B \odot e^s + t$$

NICE ideas, summarized

Nonlinear Independent Components Estimation

Dinh, Krueger and Bengio, 2014

- Composable transformations: $h = h_i \circ \ldots \circ h_2 \circ h_1$
- Disjoint partitions of :

$$h_i(x) = y = \left(y^A, y^B\right) \longrightarrow \quad y^A = x^A \qquad y^B = C\left(x^B; m\left(x^A\right)\right)$$

With: $C : \mathbb{R}^{|B|} \times m\left(\mathbb{R}^{|A|}\right) \to \mathbb{R}^{|B|}$

- Tractable inverses thanks to smart use of disjoint partitions:

$$x^A = y^A$$

$$x^B = C^{-1}\left(y^B; m\left(x^A\right)\right) = C^{-1}\left(y^B; m\left(y^A\right)\right)$$

Piecewise-Polynomial Coupling Layers

Coupling transforms (C) could be **more complex**!

$$C\left(x^B; m\left(x^A\right)\right) = \left(C_1\left(x_1^B; m\left(x^A\right)\right), \cdots, C_{|B|}\left(x_{|B|}^B; m\left(x^A\right)\right)\right)^T$$

→ **Transform each dimension independently**

We can interpretate this warping function as a *cumulative distribution function* (CDF)

How to produce each C_i ? → Neural Network to output the corresponding unnormalized probability density (q_i) → Construct C_i by integration

1 **Piecewise-Linear Coupling Transform**

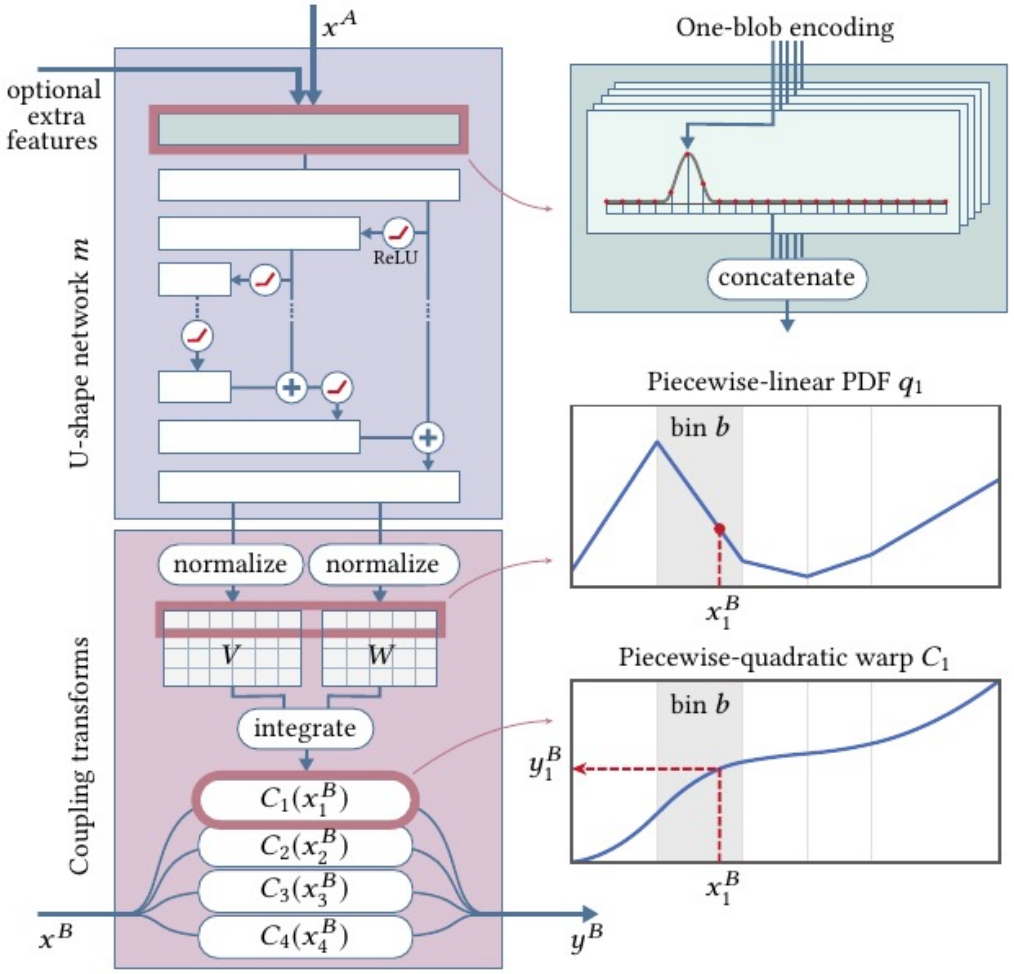
$$C_i\left(x_i^B; Q\right) = \int_0^{x_i^B} q_i(t)\mathrm{d}t = \alpha Q_{ib} + \sum_{k=1}^{b-1} Q_{ik}$$

2 **Piecewise-Quadratic Coupling Transform**

$$C_i\left(x_i^B; W, V\right) = \frac{\alpha^2}{2}\left(V_{ib+1} - V_{ib}\right)W_{ib} + \alpha V_{ib} W_{ib} + \sum_{k=1}^{b-1} \frac{V_{ik} + V_{ik+1}}{2} W_{ik}$$

Piecewise-Polynomial Coupling Layers

How does the whole **coupling layer** look like?

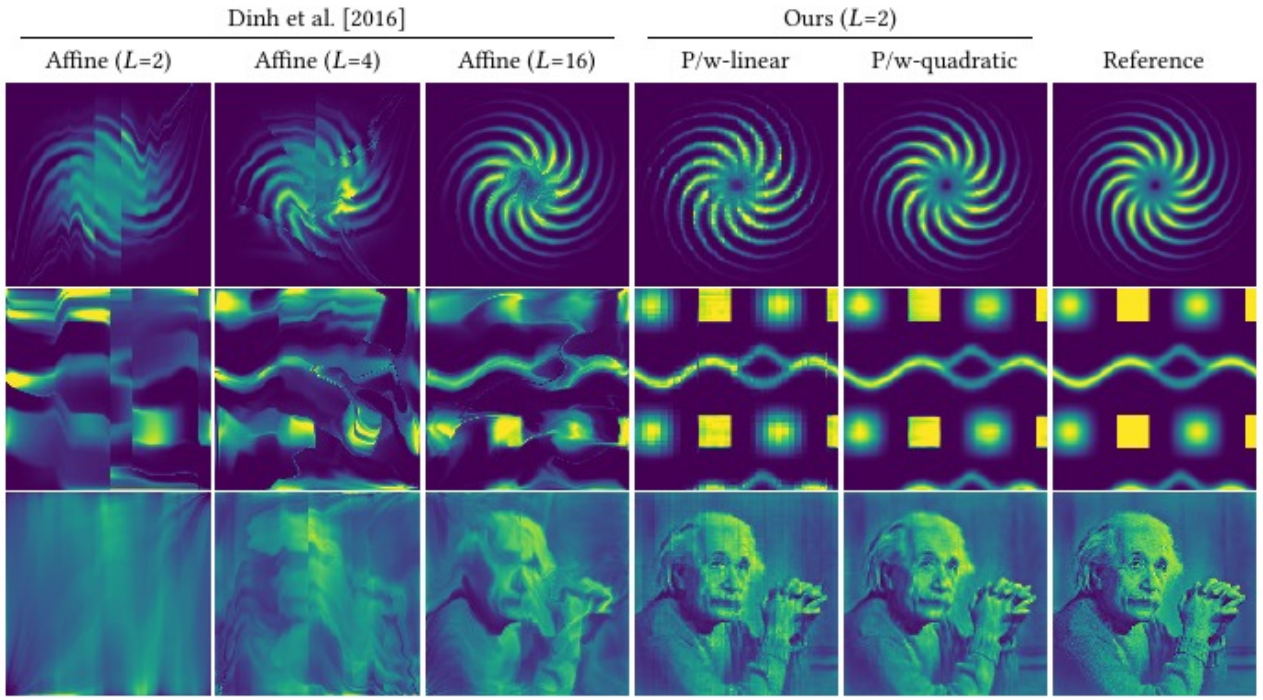


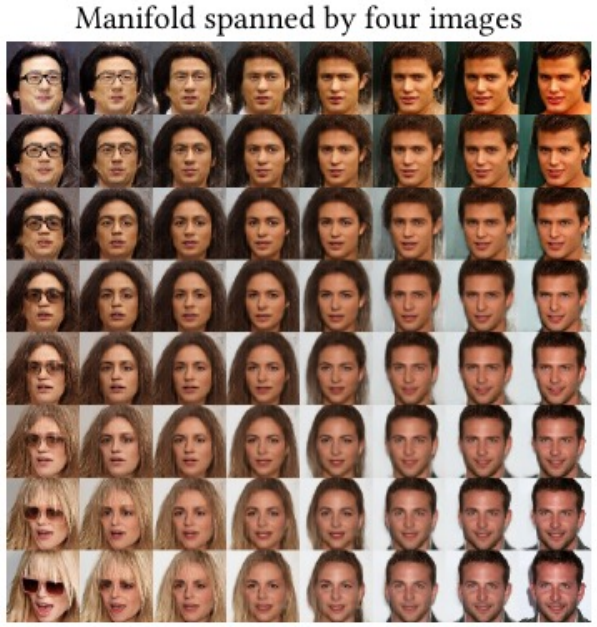
Generalization of one-hot encoding

A kernel is used to activate multiple adjacent matrix instead of a single one

Performance of Coupling Layers

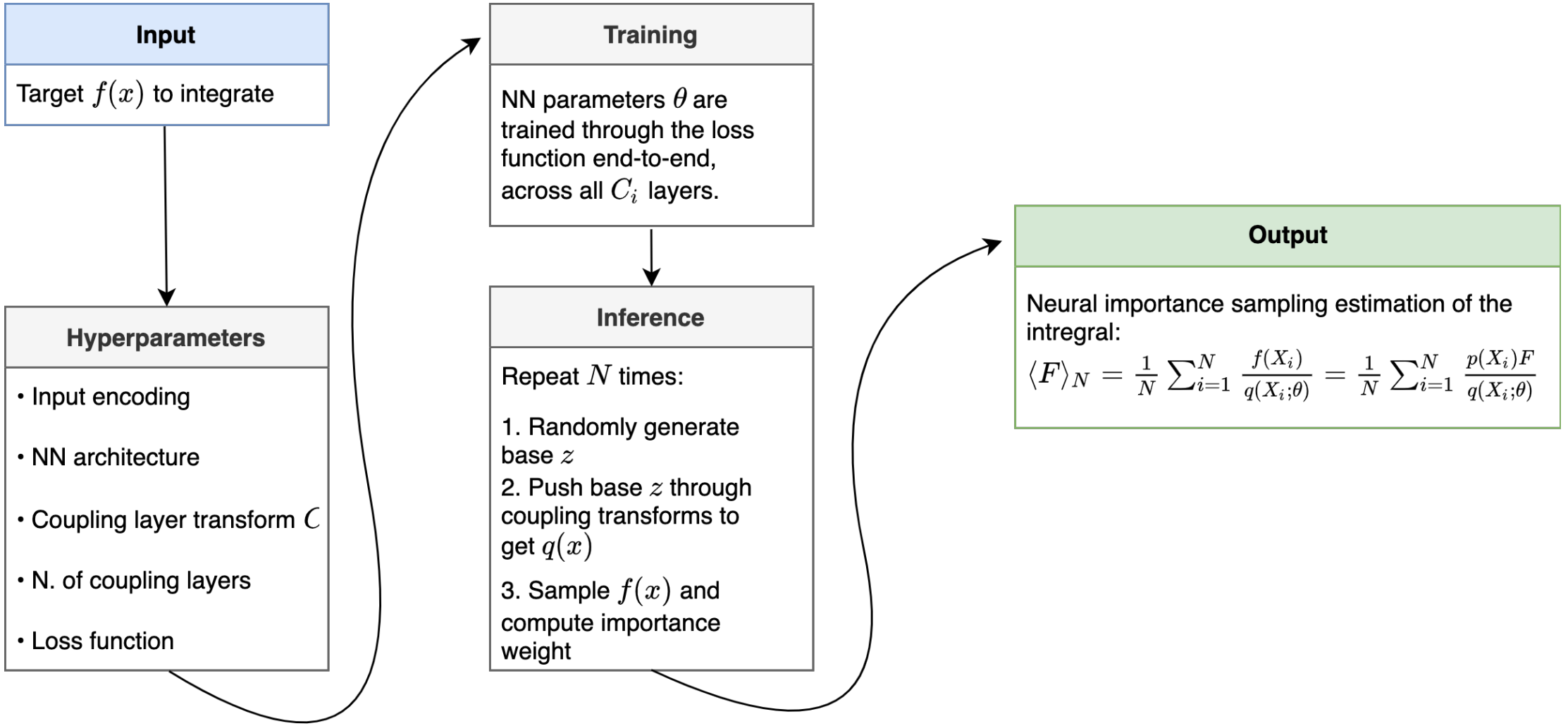
Piecewise coupling layers achieve superior performance compared with affine transforms on **low-dimensional** regression problems





Learning CelebFaces attributes with Piecewise-quadratic coupling layers: a **high-dimensional** density-estimation problem

The full algorithm



Evaluation

Data → Reduce the variance of rendered scenes (Mitsuba2)

Experimental Setup → Rendering procedure is implemented as a hybrid CPU/GPU algorithm

- CPU traces the rays in large batches
- GPU performs Neural Network optimization

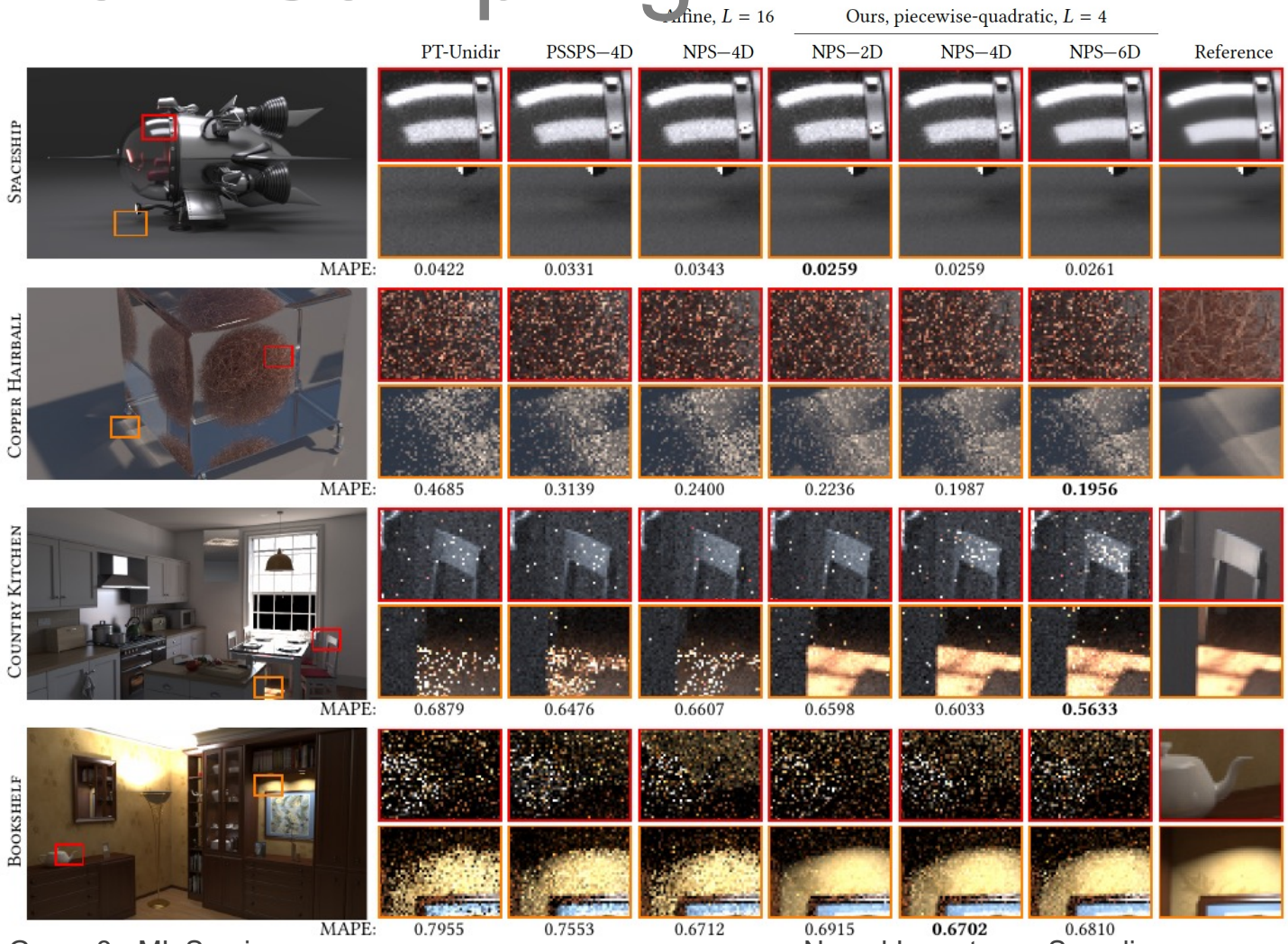
Metric → Mean Absolute Percentage Error (MAPE)

$$\frac{1}{N}\sum_{i=1}^{N}|v_i - \hat{v}_i|/(\hat{v}_i + \epsilon)$$

Avoid singularities and down-weight close-to-black pixels

Results

Path Sampling



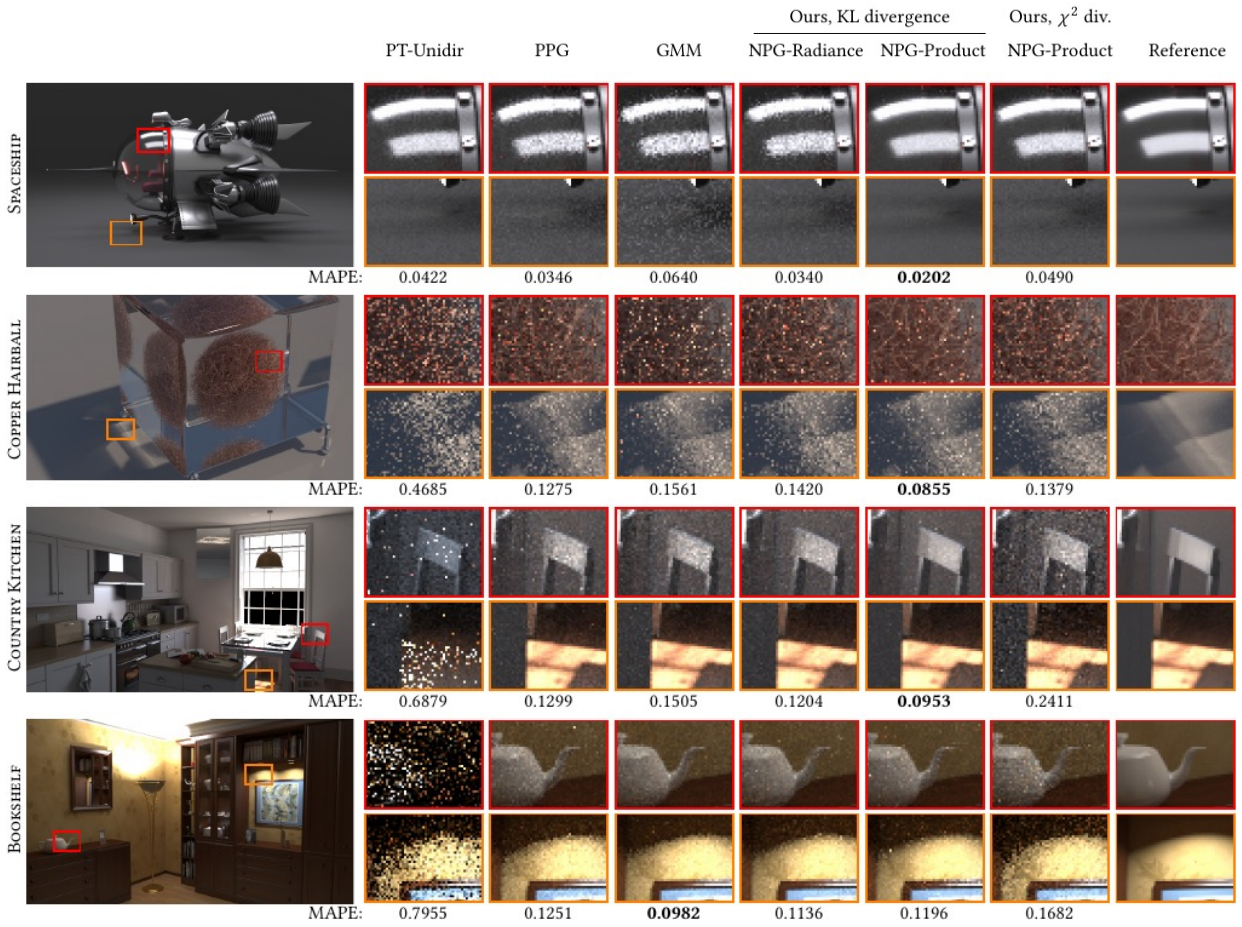
1. **Neural path sampling outperforms** previous approaches in terms of MAPE, **variance is reduced** for challenging scenes

2. **Neural path sampling only** offers improvement **beyond the 4D** case if paths stay Coherent

Results
Path Guiding

1. **Neural path guiding** outperforms previous models even without considering texture properties of materials *(NPG-radiance),* but only overall radiance.

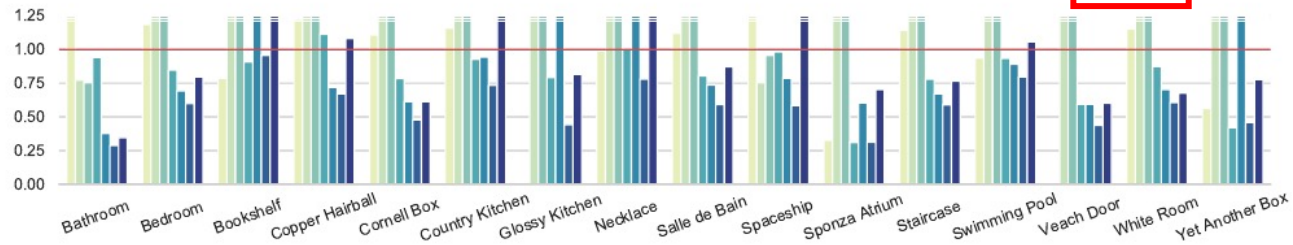
2. **Neural path guiding** performs even better while accounting for texture properties. This is called *NPG-Product* because it applies multiple importance sampling to combine different densities capturing properties of the materials



Advantages of Neural Importance Sampling

- A collection of **Neural Networks** can parametrize **complex** sampling **densities**.
- Piecewise-polynomial transforms further **improve** overall **expressiveness**
- The **performance gain** in challenging scenarios could be **big.**

		PT-Unidir	PPG	GMM	PSSPS	Ours, KL divergence: NPS one-blob	Ours, KL divergence: NPG-Rad. one-blob	Ours, KL divergence: NPG-Product scalar	Ours, KL divergence: NPG-Product one-blob	Ours, χ^2 div.: NPG-Product one-blob
Bathroom	236 MS	0.147 88s	0.189 2.3m	0.272 9.1m + 48s	0.143 89s	0.146 3.5m	0.178 9.3m	0.071 11m	**0.054** 12m	0.066 15m
Bedroom	236 MS	0.078 75s	0.053 1.8m	0.063 34m + 60s	0.068 77s	0.068 3.5m	0.045 6.4m	0.037 7.2m	**0.032** 7.7m	0.042 10m
Bookshelf	236 MS	0.796 74s	0.125 2.5m	**0.098** 16m + 66s	0.755 78s	0.681 3.5m	0.114 8.1m	0.250 10m	0.120 10m	0.168 12m
Copper Hairball	472 MS	0.468 2.0m	0.128 1.8m	0.156 11m + 2.0m	0.314 2.0m	0.196 6.9m	0.142 4.9m	0.092 15m	**0.086** 16m	0.138 17m
Cornell Box	268 MS	0.185 23s	0.044 82s	0.049 6.2m + 24s	0.130 26s	0.109 4.0m	0.035 6.6m	0.027 8.5m	**0.021** 10m	0.027 9.1m
Country Kitchen	236 MS	0.688 48s	0.130 81s	0.151 13m + 33s	0.648 49s	0.563 3.5m	0.120 5.4m	0.123 6.9m	**0.095** 7.8m	0.241 8.1m
Glossy Kitchen	236 MS	1.476 78s	0.308 87s	–	1.452 77s	1.491 3.5m	0.243 5.6m	0.810 11m	**0.136** 11m	0.251 13m
Necklace	236 MS	0.344 29s	0.163 42s	0.161 3.2m + 15s	0.333 31s	0.268 3.5m	0.163 2.6m	0.249 10m	**0.127** 11m	0.310 10m
Salle de Bain	236 MS	0.185 51s	0.071 89s	0.080 11m + 37s	0.161 54s	0.158 3.5m	0.057 5.4m	0.052 5.7m	**0.042** 6.1m	0.062 7.9m
Spaceship	236 MS	0.042 27s	0.035 56s	0.064 4.4m + 4.6m	0.033 28s	0.026 3.5m	0.034 2.8m	0.027 3.5m	**0.020** 3.9m	0.049 4.1m
Sponza Atrium	236 MS	1.709 84s	0.353 91s	0.115 11m + 53s	1.692 81s	1.616 3.5m	**0.109** 7.4m	0.213 9.0m	0.110 11m	0.247 11m
Staircase	236 MS	0.163 57s	0.057 81s	0.065 14m + 41s	0.138 58s	0.130 3.5m	0.044 5.5m	0.038 5.7m	**0.033** 6.3m	0.043 7.7m
Swimming Pool	236 MS	0.688 40s	0.082 63s	0.077 29m + 22s	0.494 41s	0.198 3.5m	0.077 3.0m	0.073 4.5m	**0.066** 5.1m	0.087 5.1m
Veach Door	236 MS	0.910 38s	0.227 70s	–	0.903 41s	0.716 3.5m	0.135 9.0m	0.135 12m	**0.099** 13m	0.137 12m
White Room	236 MS	0.102 79s	0.066 1.9m	0.076 24m + 55s	0.090 79s	0.093 3.5m	0.058 7.0m	0.046 7.8m	**0.040** 8.3m	0.045 11m
Yet Another Box	1073 MS	1.228 1.9m	0.141 4.7m	0.079 6.0m + 1.7m	1.200 2.1m	1.108 15m	**0.059** 23m	0.222 30m	0.065 36m	0.110 33m



Neural path sampling performs **better** in almost every scene

Drawbacks

- Expensive compared to other rendering models
  - A different neural network per coupling layer
  - More expensive coupling transforms than previous research
- Application scenarios are limited
  - Need to satisfy cost/benefits trade-off
  - Prohibitive for simple tasks
- Does not extend to multiple integrals
  - Cannot neglect the normalising factor

$$\langle F \rangle_N = \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{q(X_i;\theta)} = \frac{1}{N} \sum_{i=1}^{N} \frac{p(X_i)F}{q(X_i;\theta)}$$

- More difficult to examine failure cases of the model
  - Relevant in the rendering case

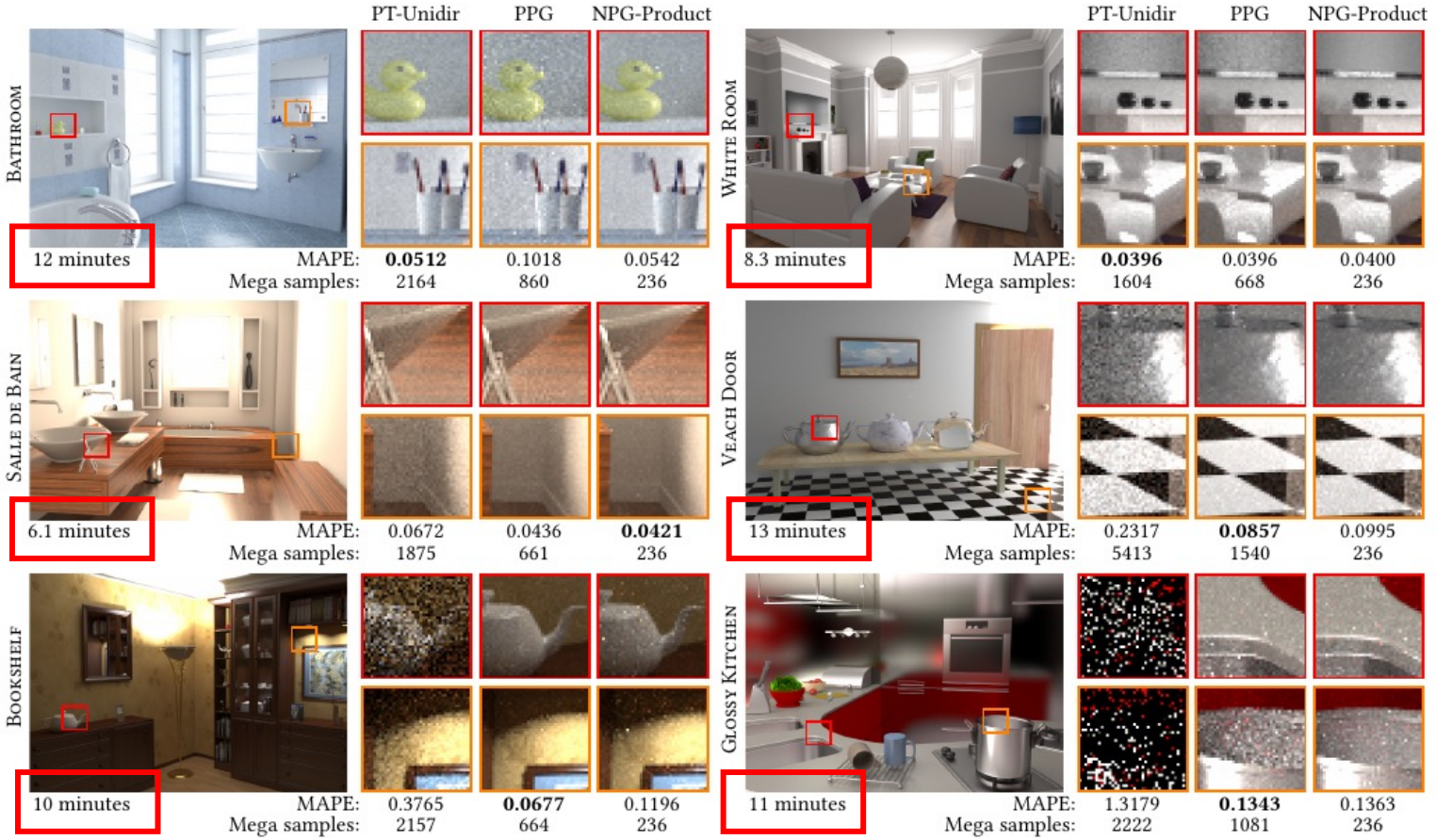
Let's do a fair comparison...

Neural Importance Sampling **performs better....**

BUT

What is the computational **overhead** of Neural Importance Sampling?

Let's run all the approaches the **same amount of time**, **which** one performs **better?**



Further Extensions

1

Extend the piecewise-polynomial coupling layer to higher degree

↓

Piecewise-**cubic** warp!

Cubic-spline Flows
Durkan et al., 2019

2

Improve density estimation by proposing a coupling layer in the rational domain

↓

Warping made of the **quotient of two quadratic polynomials**

Neural Spline Flows
Durkan et al., 2019

Takeaways

- **Integration** is a problem in a surprising variety of fields
  - **ML can help** solve these problems!
- Monte Carlo Integration with NICE **reduces the estimated variance** of the integrand.
- **NFs** can be made **more expressive** without overloading the Jacobian determinant term
  - The idea of coupling layers can be extended to **Neural Networks**!
  - Coupling layers can be of **various types**, adding expressiveness
- While Neural Importance Sampling is impractical, it outperforms the state-of-the-art approaches!

References

- Dinh, L., Krueger, D., & Bengio, Y. (2015). NICE: Non-linear Independent Components Estimation. *ArXiv:1410.8516 [Cs]*. http://arxiv.org/abs/1410.8516
- Durkan, C., Bekasov, A., Murray, I., & Papamakarios, G. (2019a). Cubic-Spline Flows. *ArXiv:1906.02145 [Cs, Stat]*. http://arxiv.org/abs/1906.02145
- Durkan, C., Bekasov, A., Murray, I., & Papamakarios, G. (2019b). Neural Spline Flows. *ArXiv:1906.04032 [Cs, Stat]*. http://arxiv.org/abs/1906.04032
- Müller, T., McWilliams, B., Rousselle, F., Gross, M., & Novák, J. (2019). Neural Importance Sampling. *ArXiv:1808.03856 [Cs, Stat]*. http://arxiv.org/abs/1808.03856

Extra: Why should we transform only one partition when building coupling layers?

1. Split the input vector into two **disjoint partitions** (x^A, x^B) of its dimensions →

2. Apply the coupling transform *C* to ONLY one of the partitions →

$$y^A = x^A$$
$$y^B = C\left(x^B; m\left(x^A\right)\right)$$

3. Build output of **coupling layer** by concatenation →

$$h(x) = y = \left(y^A, y^B\right)$$

Notice that the partition A remains **unchanged !**

↓

Allows

Trivial inversion of mapping (h):

$$x^A = y^A$$
$$x^B = C^{-1}\left(y^B; m\left(x^A\right)\right) = C^{-1}\left(y^B; m\left(y^A\right)\right)$$

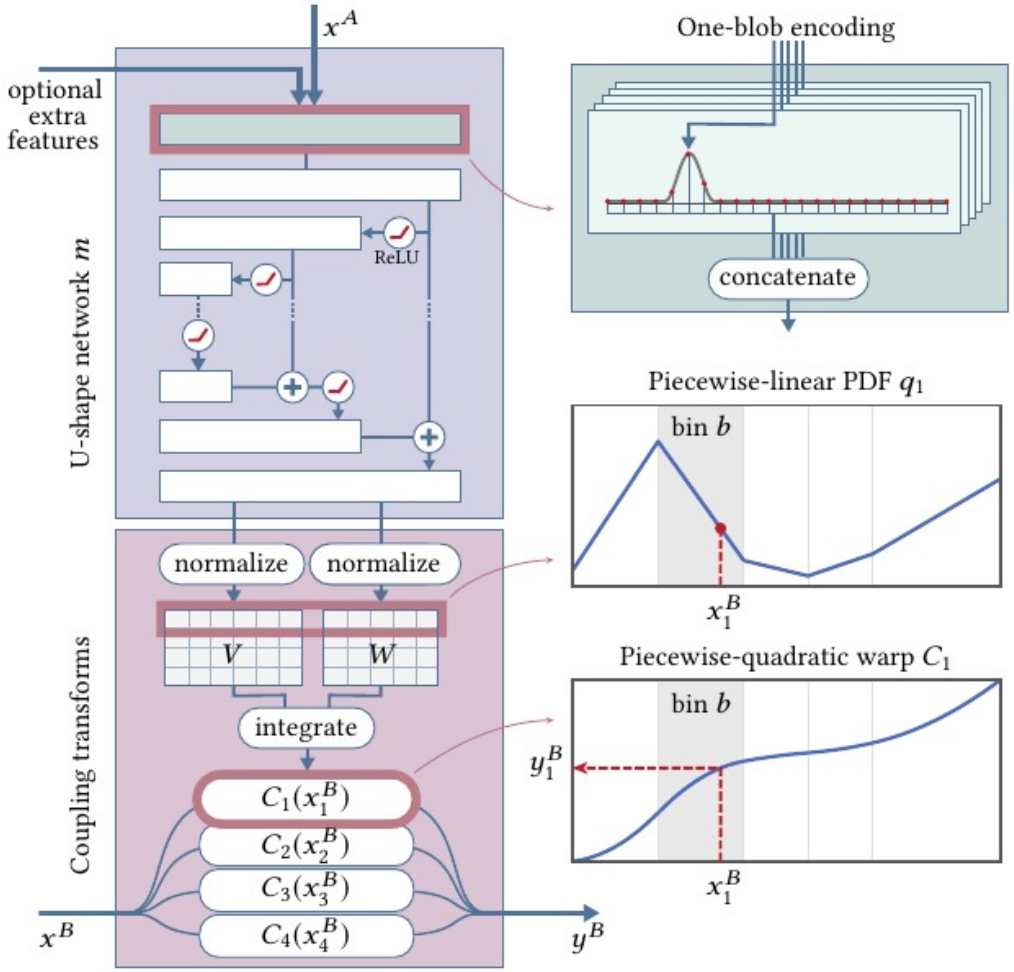
Notice that **C is separable by construction** (disjoint partitions of dimensions)

↓

Ensures

Jacobian Determinant is triangular!

Extra: More details about One-blob Encoding



Generalization of one-hot encoding

Gaussian Kernel
sigma = 1/k , where k = # of bins

A **kernel** is used to activate multiple adjacent matrix instead of a single one

Fancy properties...

- It shuts down certain parts of the linear path of the network, allowing it to **specialize the model on various sub-domains** of the input.
- In contrast to one-hot encoding, the **one-blob encoding is lossless**; it captures the exact position of the continue variable.

Extra: the effect of different loss functions on the model

(1) Kullback-Leibler-Divergence

- Notice the **importance weight** before the gradient of the log likelihood
- **Minimizing the KL** divergence via gradient descent is equivalent to **minimizing** the negative log **likelihood weighted by MC** estimates of F.
- This is perhaps the most “natural” loss to use, but **it does not explicitly control the generation of outliers** which will increase the variance of the estimator.

$$
\begin{aligned}
D_{\mathrm{KL}}(p\|q;\theta) &= \int_{\Omega} p(x)\log\frac{p(x)}{q(x;\theta)}\mathrm{d}x \\
&= \int_{\Omega} p(x)\log p(x)\mathrm{d}x - \underbrace{\int_{\Omega} p(x)\log q(x;\theta)\mathrm{d}x}_{\text{Cross entropy}}.
\end{aligned}
$$

$$
\begin{aligned}
\nabla_\theta D_{\mathrm{KL}}(p\|q;\theta) &= -\nabla_\theta \int_{\Omega} p(x)\log q(x;\theta)\mathrm{d}x \\
&= \mathbb{E}\left[-\frac{p(X)}{q(X;\theta)}\nabla_\theta \log q(X;\theta)\right]
\end{aligned}
$$

Extra: the effect of different loss functions on the model

(2) χ^2-Divergence

- Notice how the gradient is weighted by the **square** of the importance weight
- **Minimising the variance** of the estimator is **equivalent**, upon close inspection, to **minimising the Pearson chi-squared divergence** between the proposal and ground truth
- This estimate is **more conservative than DKL**, i.e. it penalises more strongly low densities of q in regions where p is high density.

$$
\begin{aligned}
\mathbb{V}\left[\frac{p(X)}{q(X;\theta)}\right] &= \mathbb{E}\left[\frac{p(X)^2}{q(X;\theta)^2}\right] - \mathbb{E}\left[\frac{p(X)}{q(X;\theta)}\right]^2 \\
&= \int_\Omega \frac{p(x)^2}{q(x;\theta)}\mathrm{d}x - \underbrace{\left(\int_\Omega p(x)\mathrm{d}x\right)^2}_{1}. \\
\nabla_\theta \mathbb{V}\left[\frac{p(X)}{q(X;\theta)}\right] &= \nabla_\theta \int_\Omega \frac{p(x)^2}{q(x;\theta)}\mathrm{d}x \\
&= \int_\Omega p(x)^2 \nabla_\theta \frac{1}{q(x;\theta)}\mathrm{d}x \\
&= \int_\Omega -\frac{p(x)^2}{q(x;\theta)} \nabla_\theta \log q(x;\theta)\mathrm{d}x \\
&= \mathbb{E}\left[-\left(\frac{p(X)}{q(X;\theta)}\right)^2 \nabla_\theta \log q(X;\theta)\right].
\end{aligned}
$$