Learning to Importance Sample in Primary Sample Space

Quan Zheng[1], Matthias Zwicker[1]

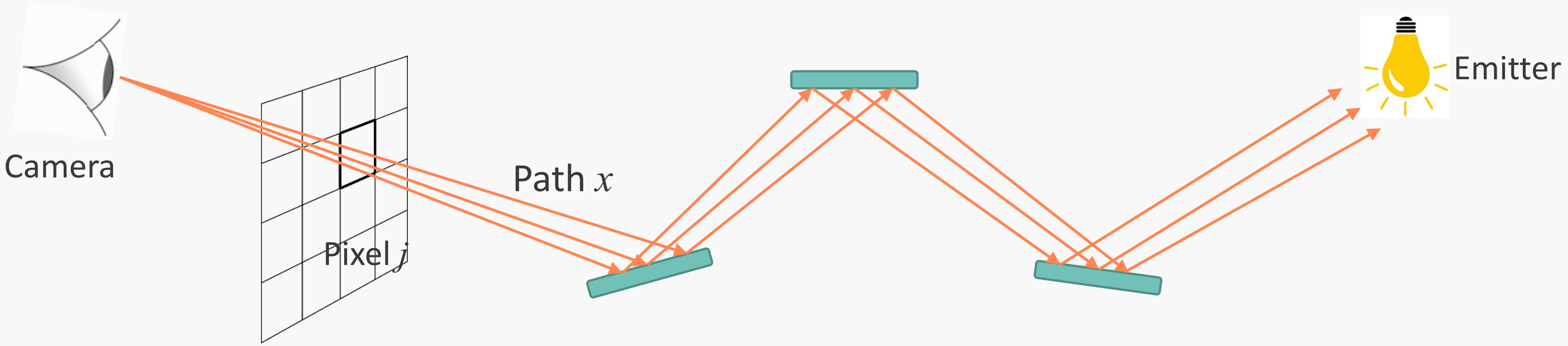
[1] University of Maryland, College Park, USA

Slides adapted from conference presentation:

EG2019

The 40° Annual Conference of the European Association for Computer Graphics

Motivation: Monte Carlo Rendering

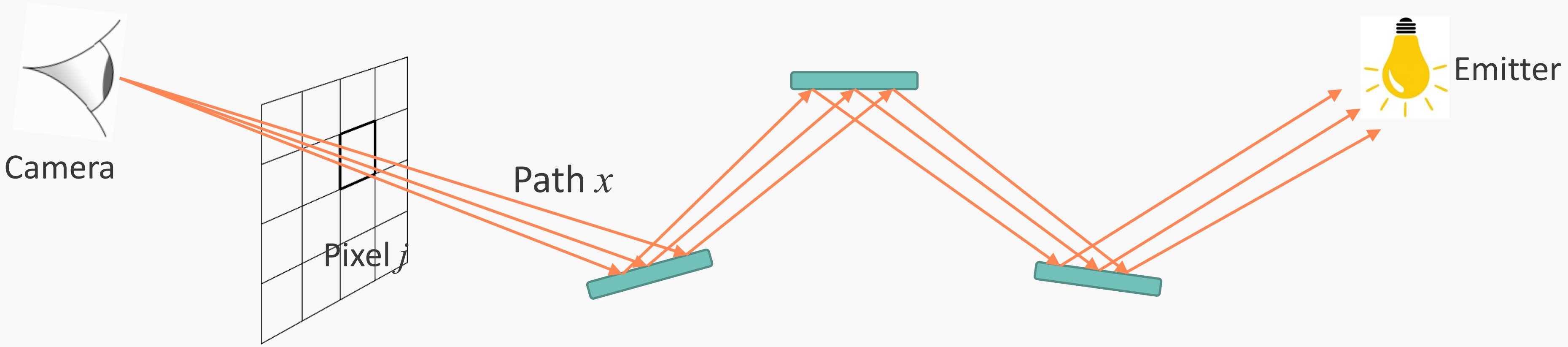


Motivation: Monte Carlo Rendering

Rendering equation: integral over all light paths through pixel j

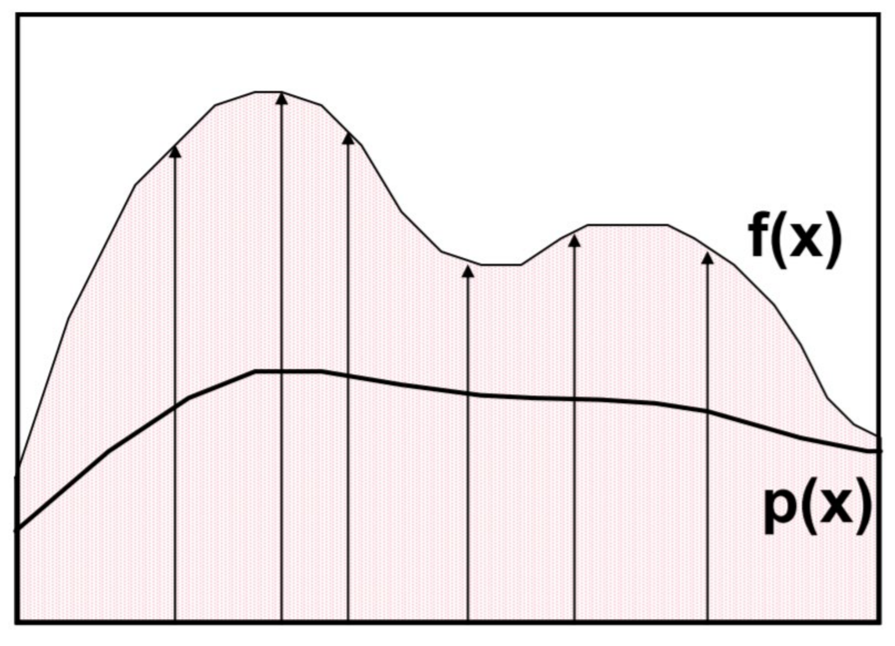
$$I_j = \int_{\Omega} f_j(x)\, du(x)$$

Monte Carlo integration: $I_j \approx \sum_{i=1}^{N} \frac{f_j(x_i)}{p(x_i)}$ where x_i is a random sampled path



Motivation: Importance Sampling

- Reduce variance by designing sampling density p proportional to integrand f, as much as possible



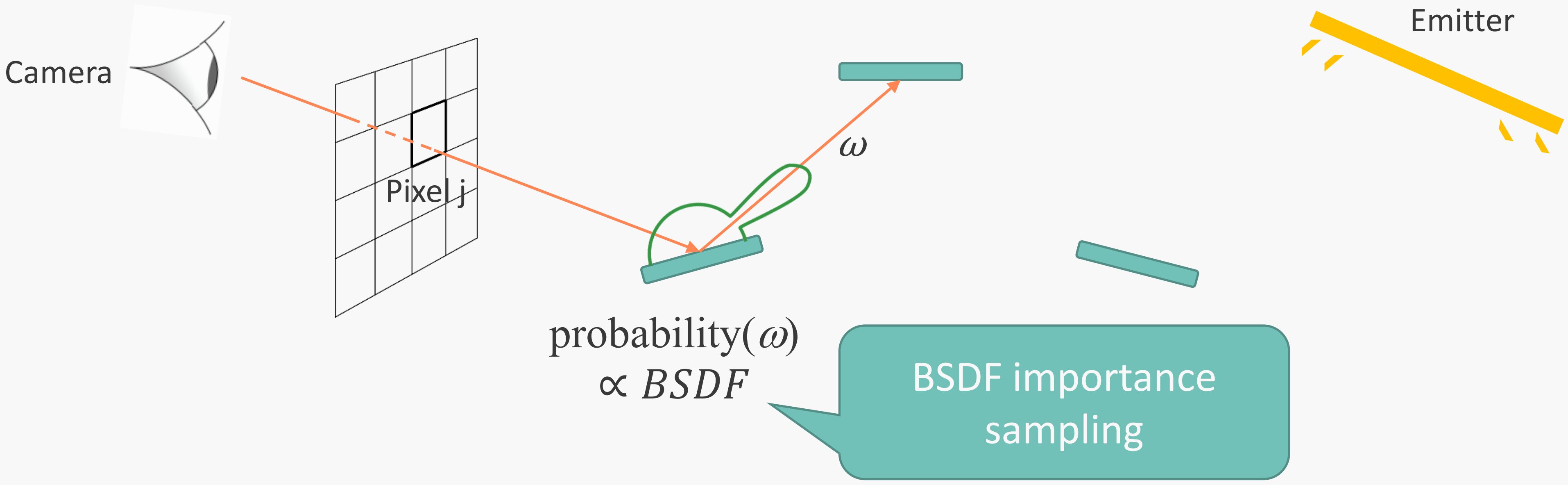
Variance

$$V\left[\frac{1}{N}\sum_{i=1}^{N}\frac{f_j(x_i)}{p(x_i)}\right] = \frac{1}{N}V\left[\frac{f_j(x)}{p(x)}\right]$$

Design density p so this is constant, as much as possible

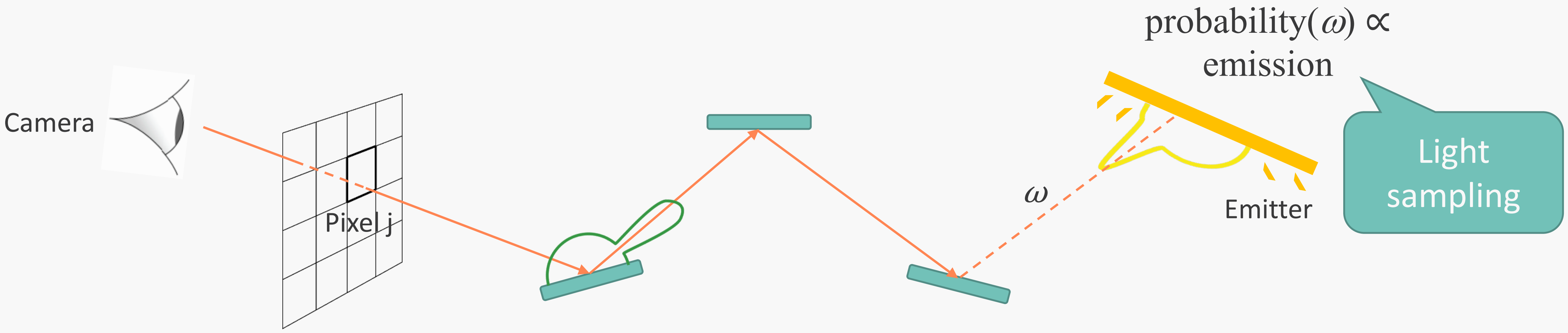
Standard Importance Sampling

- Construct paths step by step
- In each step, choose new direction using either **BSDF sampling**, or **light sampling**



Standard Importance Sampling

- Construct paths step by step
- In each step, choose new direction using either **BSDF sampling**, or **light sampling**



Standard Importance Sampling

Disadvantages

- Does not consider full complexity of integrand (importance samples at each bounce separately, instead of importance sampling entire paths)
- Approximate models of BSDFs and emitters for importance sampling
- Ignores occlusions (visibility function)
- Does not include non-local effects such as chains of near specular reflections
- Requires separate techniques for effects such as motion blur or depth of field

Standard Importance Sampling

Disadvantages

- Does not consider full complexity of integrand (importance samples at each bounce separately, instead of importance sampling entire paths)
- Approximate models of BSDFs and emitters for importance sampling
- Ignores occlusions (visibility function)
- Does not include non-local effects such as chains of near specular reflections
- Requires separate techniques for effects such as motion blur or depth of field

Our goals

- Use higher dimensional target sampling density that more accurately represents integrand
- Learn how to sample from target density by leveraging neural network
- Unified approach that treats renderer as black box, is applicable to any light transport effect

Related Work: Importance Sampling

- A priori
  - Use analytical representations of the integrand [Clarberg05]

Related Work: Importance Sampling

- A priori
  - Use analytical representations of the integrand [Clarberg05]
- A posteriori
  - Acquire small set of “training” samples of integrand, then fit target density to acquired training samples
    - Path guiding by caching [Jesen95, Hey02]
    - Online learning path guiding [Vorba14, Müller17]
    - Path guiding with reinforced method [Dahm17]
    - Kd-tree based path guiding [Guo2018]
- A combination
  - Bayesian approach to sample direct illumination [Vevoda18]

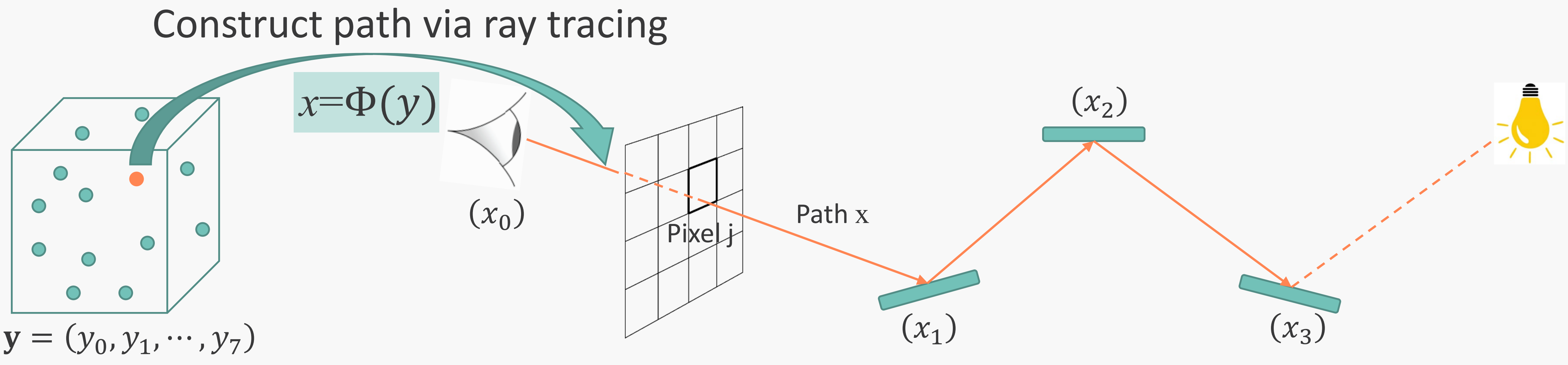
Related Work: Markov Chain Monte Carlo

- Metropolis light transport (MLT) [VG97]
- Various mutation techniques and path parameterisations [JM12, KHD14, LLR15]
- Primary sample space MLT (PSSMLT) [KSKAC02]
- Multiplexed MLT (MMLT) and combine different space parameterizations [HKD14, OKH17, BJNJ7]

Related Work: Deep Learning for Rendering

- Denoise Monte Carlo rendering [BVM*17, CKS*17, VRM*18]
- Indirect lighting approximation [RWG*13]
- Cloud radiance prediction [KMM*17]
- **A concurrent work for importance sampling [MMR*18]**
  https://arxiv.org/abs/1808.03856

Preliminaries: Primary Sample Space Integral

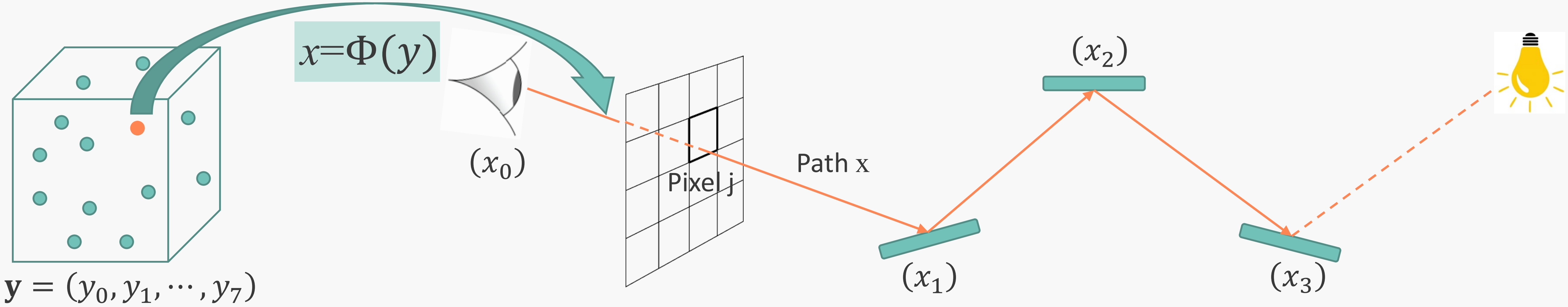


Primary sample space (PSS)
(unit hypercube; samples of random number generator)

Path space Ω, path x given by vertex locations x_i (surface form of rendering equation)

Preliminaries: Primary Sample Space Integral

Construct path via ray tracing



Primary sample space (PSS)

(unit hypercube; samples of random number generator)

Path space Ω

$$I_j = \int_{\Omega} f_j(x)\, du(x) = \int_{PSS} f_j(\Phi(\mathbf{y})) \boxed{\left|\frac{\partial \Phi(\mathbf{y})}{\partial \mathbf{y}}\right|} d\mathbf{y}$$

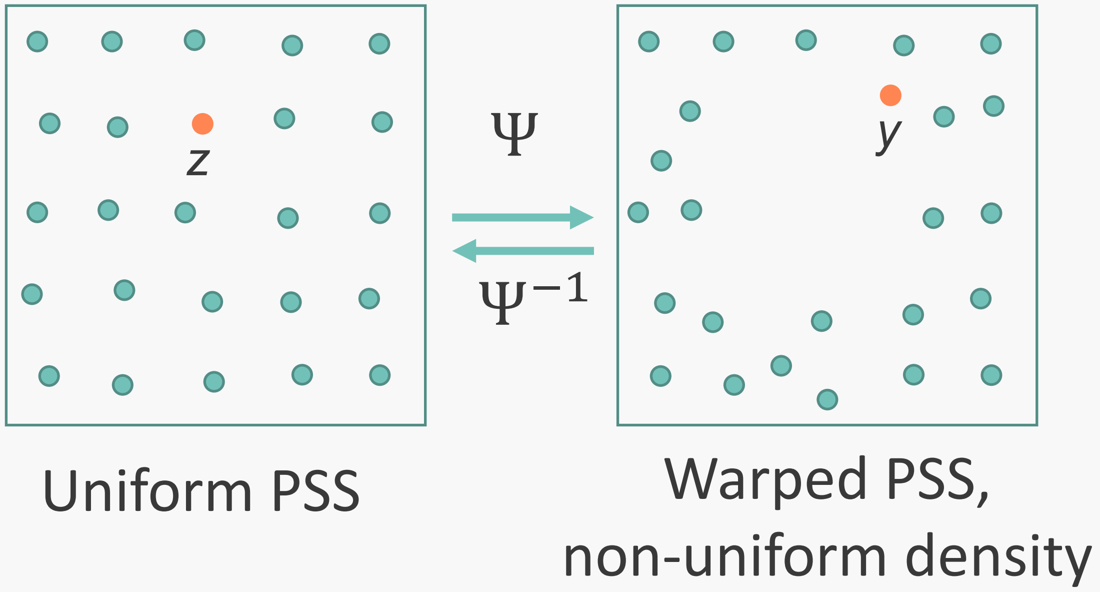
Path construction from primary samples corresponds to change of integration variables $x = \Phi(y)$

$$I_j \approx \frac{1}{N} \sum_{i=1}^{N} \frac{f_j(\Phi(\mathbf{y}))}{\left|\frac{\partial \Phi(\mathbf{y})}{\partial \mathbf{y}}\right|^{-1}}$$

Importance sampling expressed as change of integration variables

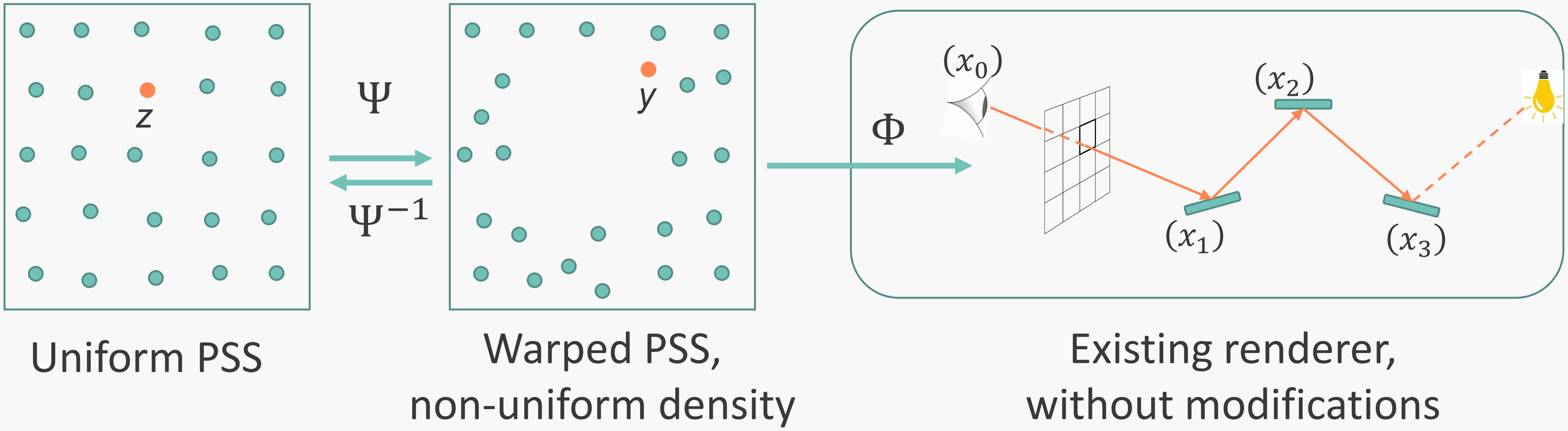
Our core idea

- Improve importance sampling using primary sample space warping Ψ
- Non-linear warp Ψ leads to warped PSS with non-uniform density

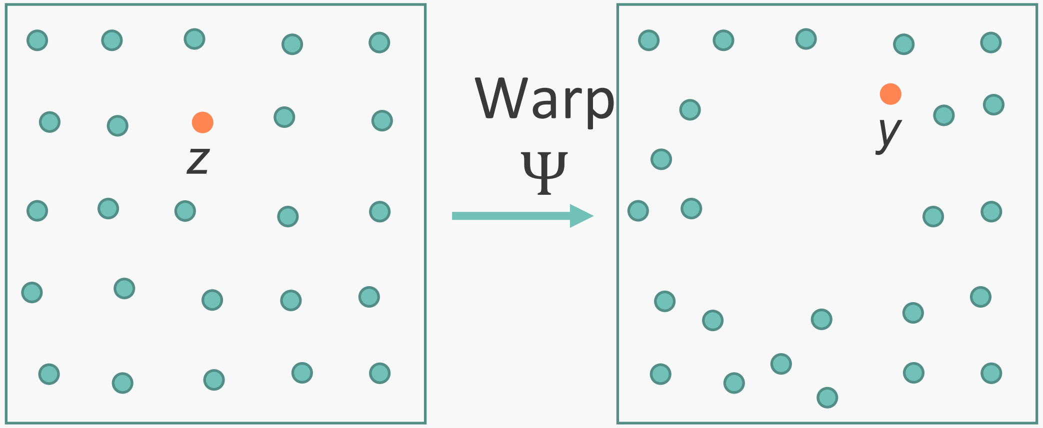


Our core idea

- Improve importance sampling using primary sample space warping Ψ
- Non-linear warp Ψ leads to warped PSS with non-uniform density



Non-linear PSS warp and importance sampling



Uniform PSS

Warped PSS,
non-uniform density

$$I_j = \int_\Omega f_j(x)\, du(x)$$

$$= \int_{PSS} f_j(\Phi[\Psi(\mathbf{z})]) \left|\frac{\partial\Psi(\mathbf{z})}{\partial\mathbf{z}}\right| \left|\frac{\partial\Phi[\Psi(\mathbf{z})]}{\partial\Psi(\mathbf{z})}\right| d\mathbf{z}$$

Chain rule including warp

Existing renderer as black box

$$I_j \approx \frac{1}{N}\sum_{i=1}^{N} \frac{f_j(\Phi[\Psi(\mathbf{z}_i)])}{\left|\frac{\partial\Psi(\mathbf{z}_i)}{\partial\mathbf{z}_i}\right|^{-1} \left|\frac{\partial\Phi[\Psi(\mathbf{z}_i)]}{\partial\Psi(\mathbf{z}_i)}\right|^{-1}}$$

Design Ψ to improve importance sampling

PSS target density

Ideal goal:
warp Ψ so that density proportional
to each pixel integrand j in existing renderer

$$p(y) = \left|\frac{\partial \Psi(z)}{\partial z}\right|^{-1} \propto \frac{f_j(\Phi(y))}{\left|\frac{\partial \Phi(y)}{\partial y}\right|^{-1}}$$

- PSS target density denoted as $p(y)=p(\Psi(z))$
  - Goal: make it proportional to integrand in PSS
  - Challenge: each pixel has its own integrand
- Simplification: one global target density, instead of per pixel
  - Using path throughput g, instead of image contribution function f_j

Throughput Pixel filter

$$f_j(\Phi(y)) = \underbrace{g(\Phi(y))}_{\text{Throughput}} \cdot \underbrace{W_j(\Phi(y))}_{\text{Pixel filter}}$$

- Global target density

Desired warp

$$p(y) = \left|\frac{\partial \Psi(z)}{\partial z}\right|^{-1} \propto \frac{g(\Phi(y))}{\left|\frac{\partial \Phi(y)}{\partial y}\right|^{-1}}$$

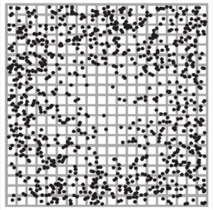
Practical goal:
warp Ψ so density proportional
to path throughput g in existing renderer

Challenges

- Target density and its representation
- Representation of the warp
- Learning the warp

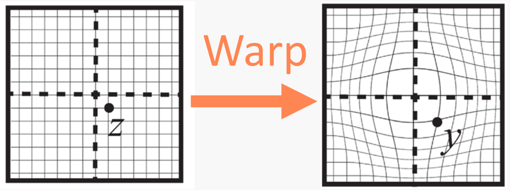
Overview

1



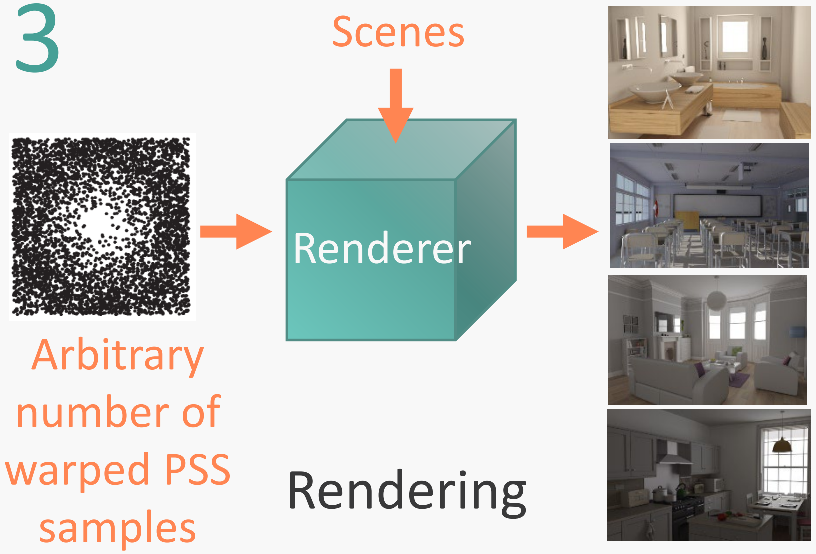
Acquire PSS target density as small set of samples

2



Represent warp as neural network, train to match target density

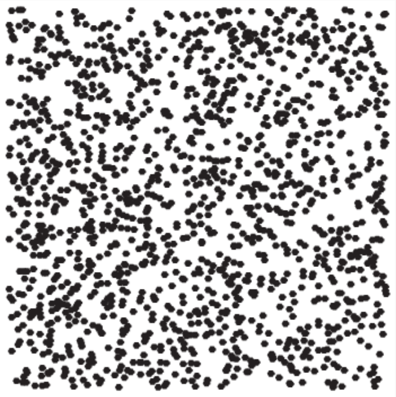
3



Represent target density using set of samples

- Build a candidate set **T,** uniformly sampled in PSS
  - Each sample stores value of target density
- Resample a subset **S** from **T** that follows target density [TCE05 https://dl.acm.org/doi/10.5555/2383654.2383674], **sample-importance resampling**

Candidate set, uniformly distributed in PSS, each sample stores value of target density

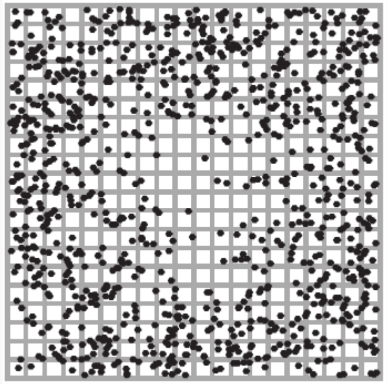


$|T|=\alpha \cdot |S|, \alpha>1$

Resample

(discard samples with probability inversely proportional to target density)

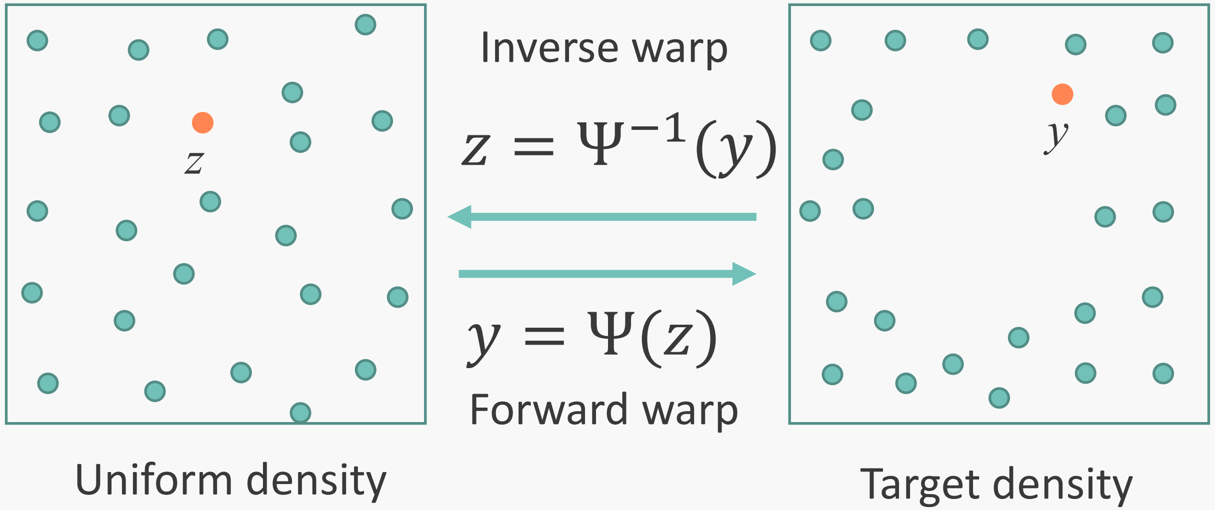
Resampled set, distributed according to target density



$|S|$

Target density as invertible warp

- Find inverse warp $z = \Psi^{-1}(y)$, such that samples $y = \Psi(z)$ follow distribution of target density



Maximum likelihood estimation

- Inverse warp parametrized by θ
- Consider density as function of warp parameters θ

$$p(\theta; y) = \left| \frac{\partial \Psi^{-1}(y;\ \theta)}{\partial y} \right|$$

Density corresponds to determinant of Jacobian of mapping
https://en.wikipedia.org/wiki/Random_variable#Functions_of_random_variables

- Objective: find the optimal θ to maximize likelihood of samples y_i of target density
  - "Find warp that is most likely to produce target samples"

$$\theta^* = arg \max_{\theta} \frac{1}{N} \sum_i \log\big(p(\theta; y_i)\big)$$

Warp representation

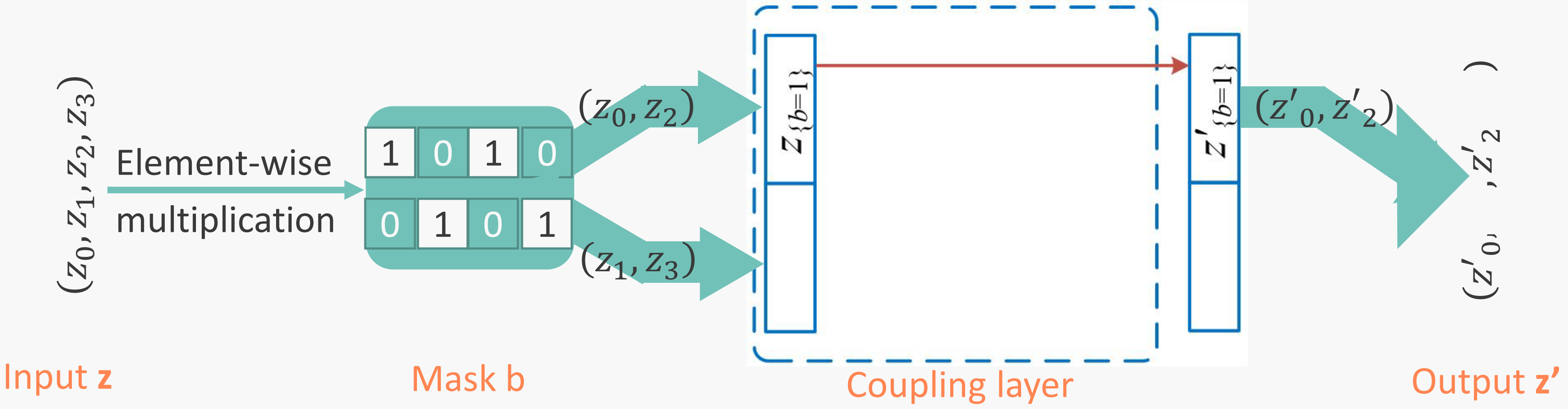
- Requirements of warp Ψ
  - Express complex, non-linear mappings
  - One-to-one mapping
  - Easy to invert
  - Evaluate Jacobian determinant efficiently

Key idea: normalizing flows (overview see https://arxiv.org/pdf/1908.09257.pdf)

- Represent warp as deep neural network
- Leverage normalizing flow architecture, here ‘real NVP’, Dinh et al., ICLR 2017
  - Satisfies all requirements

Real NVP

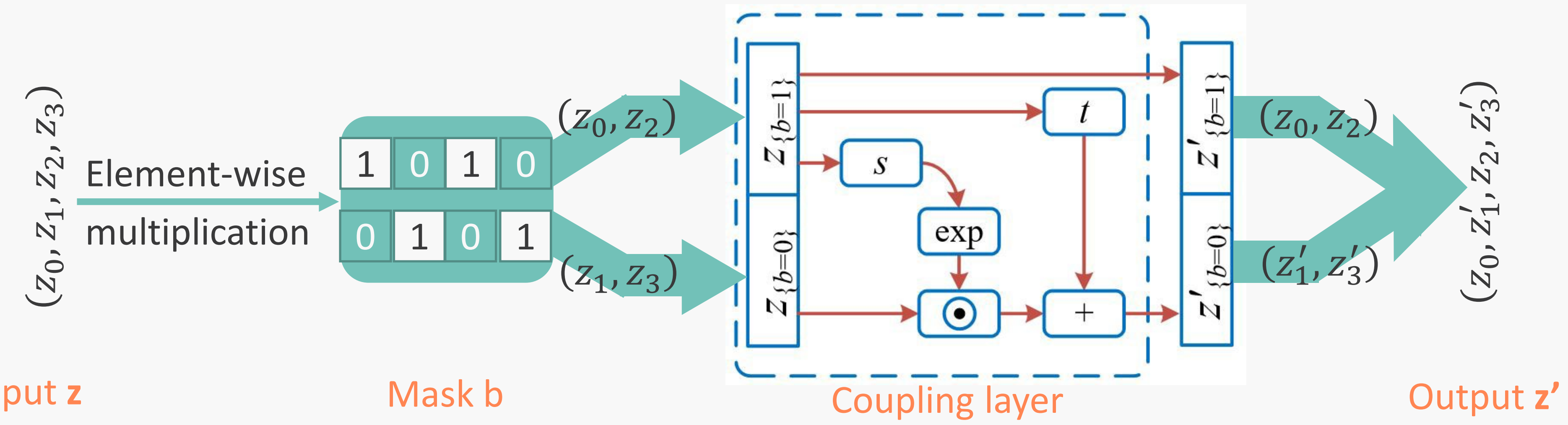
- Affine coupling layer (forward mapping)



$$z'_{\{b=1\}} = z_{\{b=1\}}$$

Real NVP

- Affine coupling layer (forward mapping)

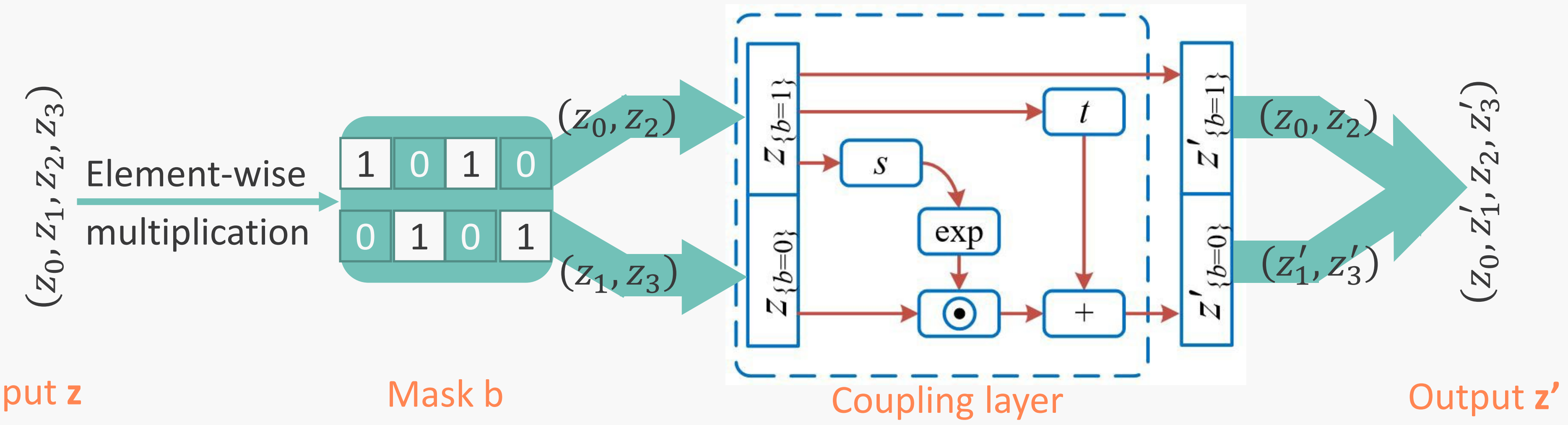


$$z'_{\{b=1\}} = z_{\{b=1\}}$$

$$z'_{\{b=0\}} = z_{\{b=0\}} \odot \exp(s(z_{\{b=1\}})) + t(z_{\{b=1\}})$$

Real NVP

- Affine coupling layer (forward mapping)



$$z'_{\{b=1\}} = z_{\{b=1\}}$$

$$z'_{\{b=0\}} = z_{\{b=0\}} \odot \exp(s(z_{\{b=1\}})) + t(z_{\{b=1\}})$$

$$\mathbf{J} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \dots & \dots & e^{s(z_{\{b=1\}})_0} & 0 \\ \dots & \dots & 0 & e^{s(z_{\{b=1\}})_1} \end{bmatrix}$$

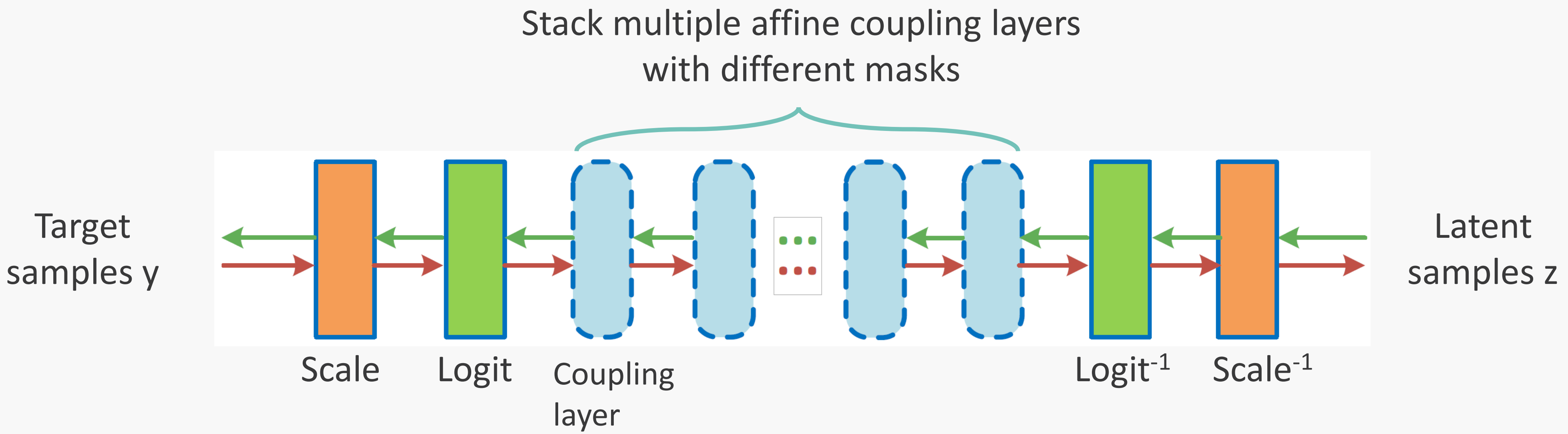
$$|\mathbf{J}| = \exp\left(\sum_k s(z_{\{b=1\}})_k\right)$$

Real NVP

- Inverse of coupling layer
  - Trivial to compute due to structure of coupling layer
  - Does not require inverting functions s and t
- Implementing functions s and t
  - Subject to few constraints
  - Neural networks
  - Warp parameters θ are trainable network weights

Achieving complicated warps

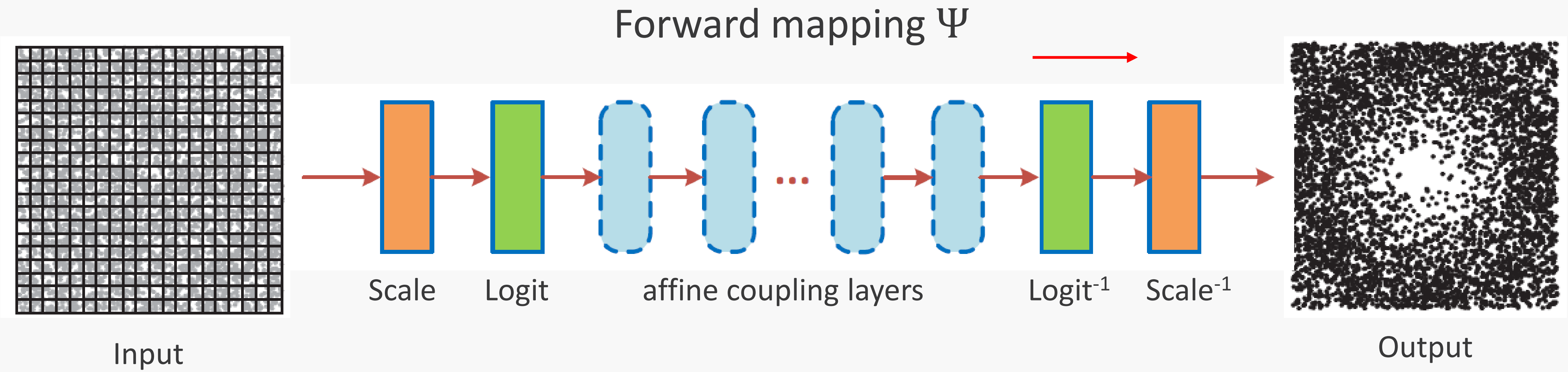
- Multiple coupling layers



- Training under maximum likelihood objective

Generating samples for rendering

- Use forward mapping Ψ
- Input: uniform random PSS vectors
- Output: target PSS vectors for rendering



Training details

- Network initialization
  - Pretrain the neural network to achieve an identity warp
  - Reuse the trained weights as initialization
- Optimization method
  - End-to-end scene-dependent training
  - Adam optimizer with learning rate 10^{-4}
  - Batch size 2000

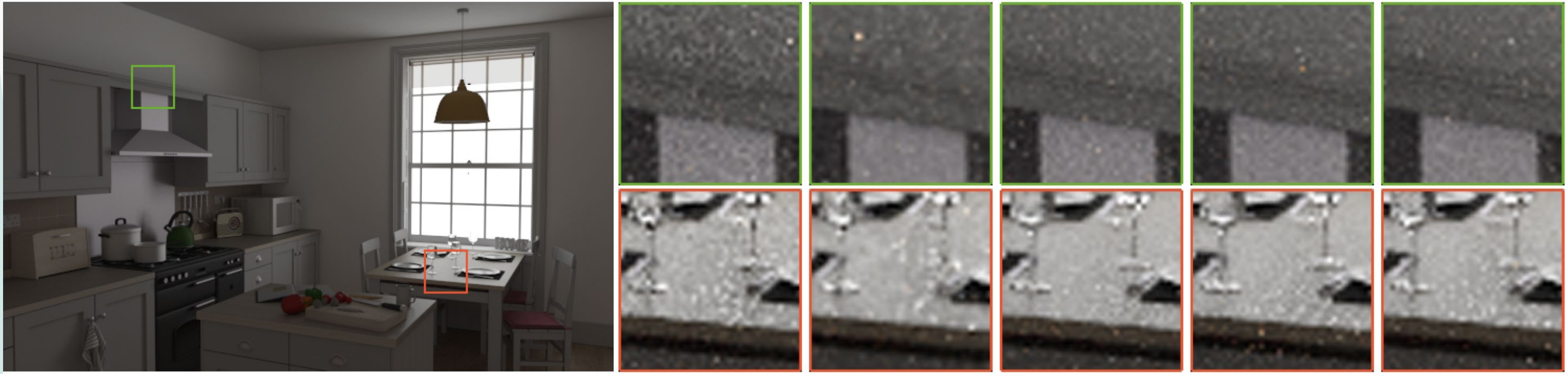
Results

Results

- Practical simplification
  - Warp at most the first 3 bounces (8D warp, 6D + 2D image plane)
  - Continue to trace further bounces using uniform PSS parameters
- Training dataset sizes
  - Epp-k (k x 200 x 160 training samples)
- Training time
  - Ranges from 9 to 20 minutes on Nvidia GTX 1070 GPU
  - We show equal-time comparison. To see timing results, please refer to the paper.

Equal sample count comparison (128spp)

- Country Kitchen scene



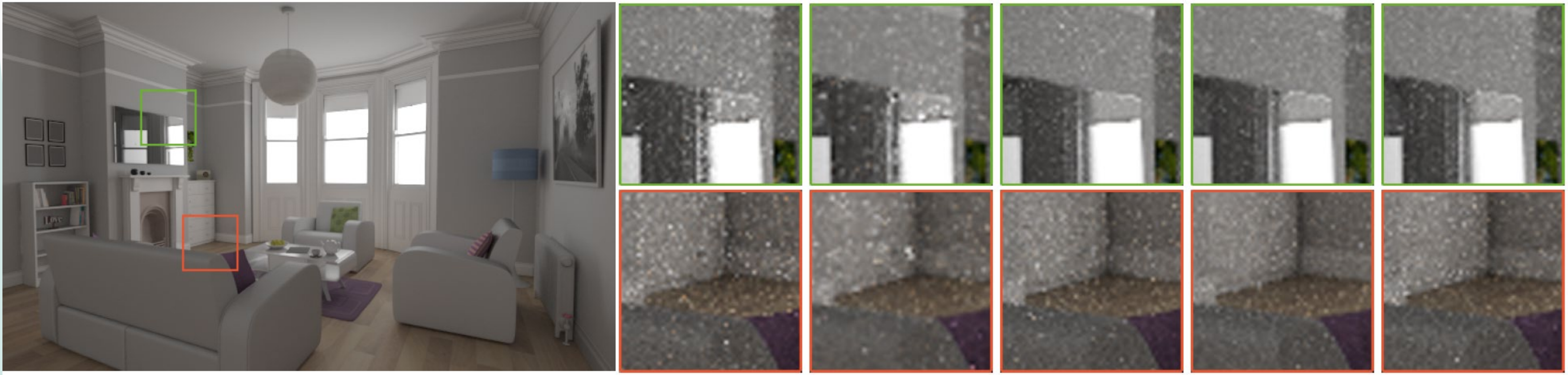
Reference	PT no warp	[GBBE18] Kdtree warp	Our 4D	Our 6D	Our 8D
MSE	0.0401	0.0220	0.0170	0.0172	0.0205

$\underbrace{\text{Our 4D, Our 6D, Our 8D}}_{\textbf{Training with epp-16}}$

Equal sample count comparison (128spp)

- White Room scene

Reference	PT no warp	[GBBE18] Kdtree warp	Our 4D	Our 6D	Our 8D



MSE	0.2155	0.1235	0.0820	0.0677	0.0832

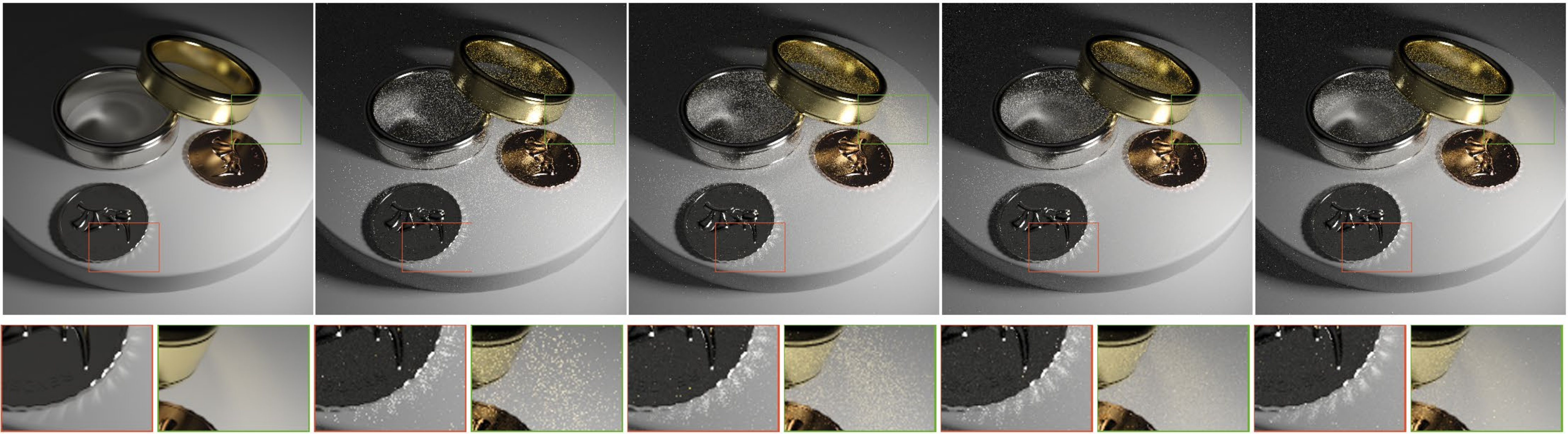
$\underbrace{\text{Our 4D, Our 6D, Our 8D}}_{\textbf{Training with epp-16}}$

Training with epp-16

Small illumination features

- Increasing training data size improves our results, captures small illumination features

Reference	PT no warp	Ours with training data size epp=1	Ours with training data size epp=4	Ours with training data size epp=64



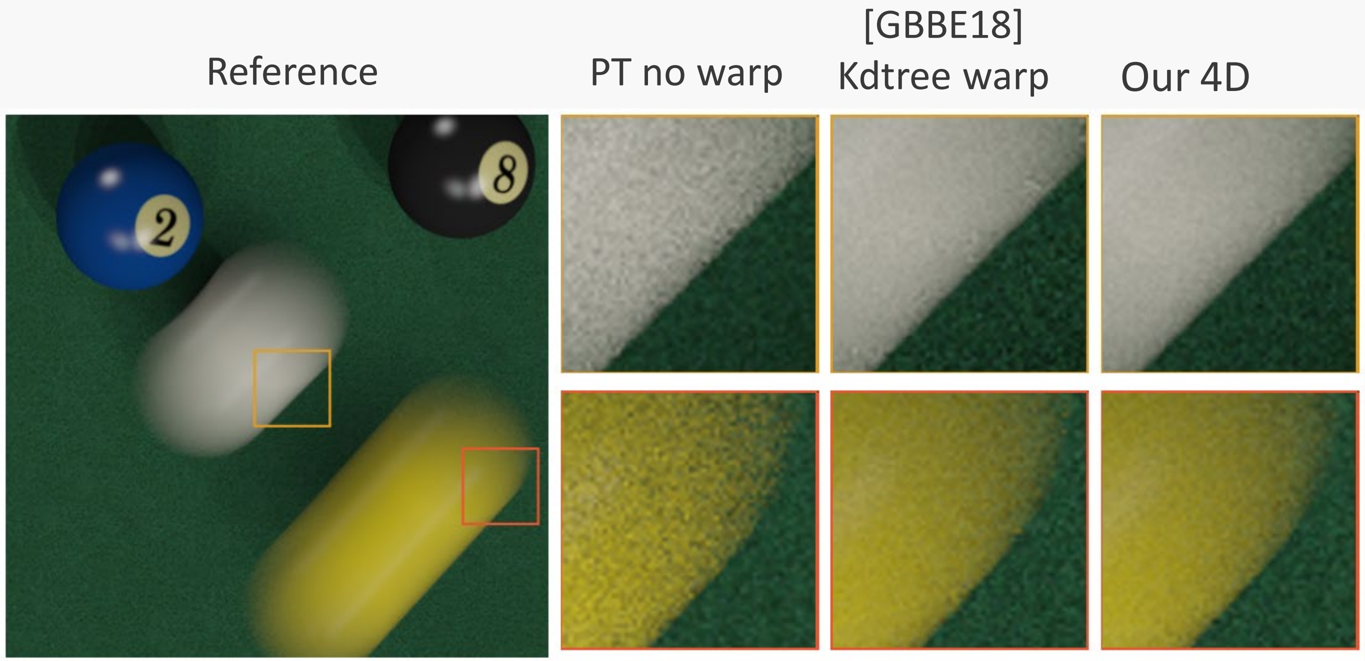
MSE	0.09275	0.06863	0.05701	0.05038

Training data size = epp x 100^2

(Equal sample count 128 spp comparisons)

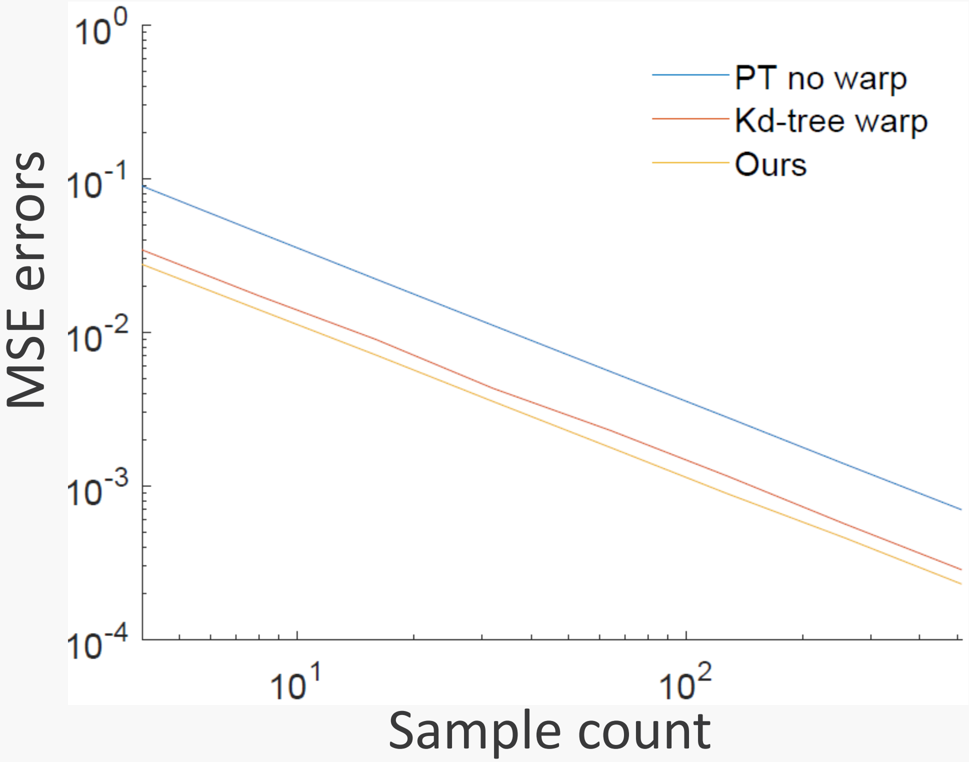
Distribution ray tracing

- Pool ball with motion blur



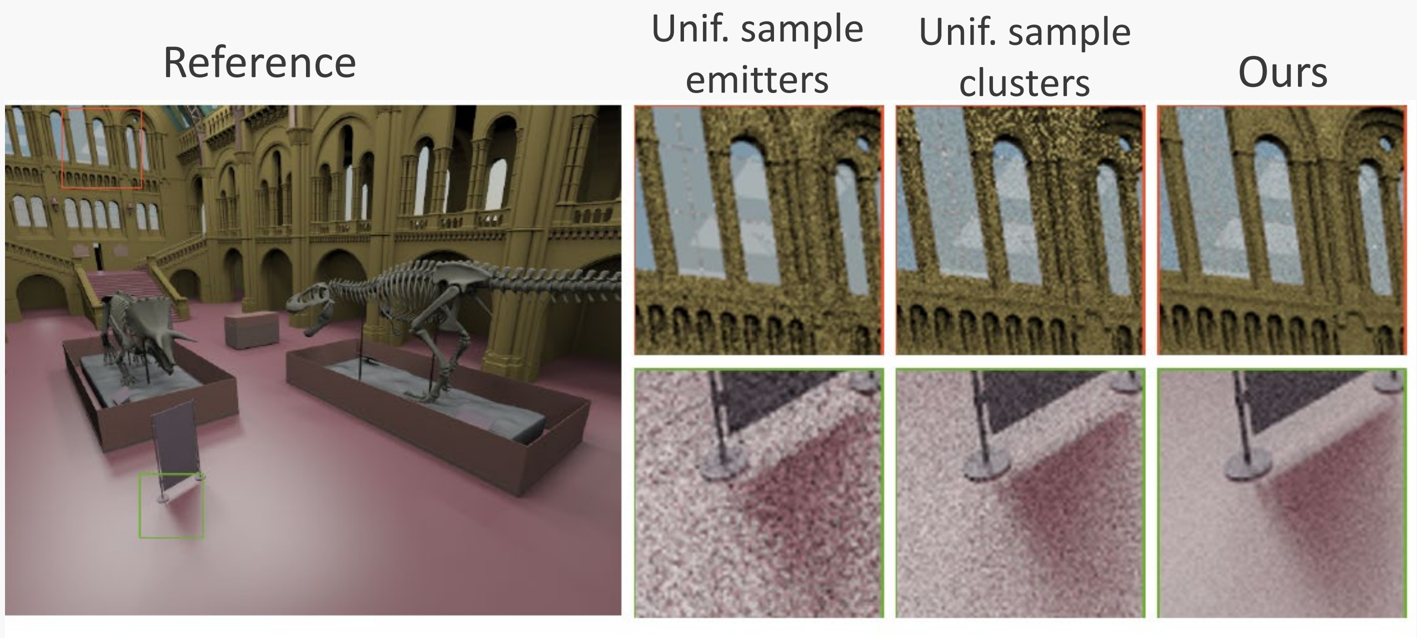
Training with epp-16

Error plot

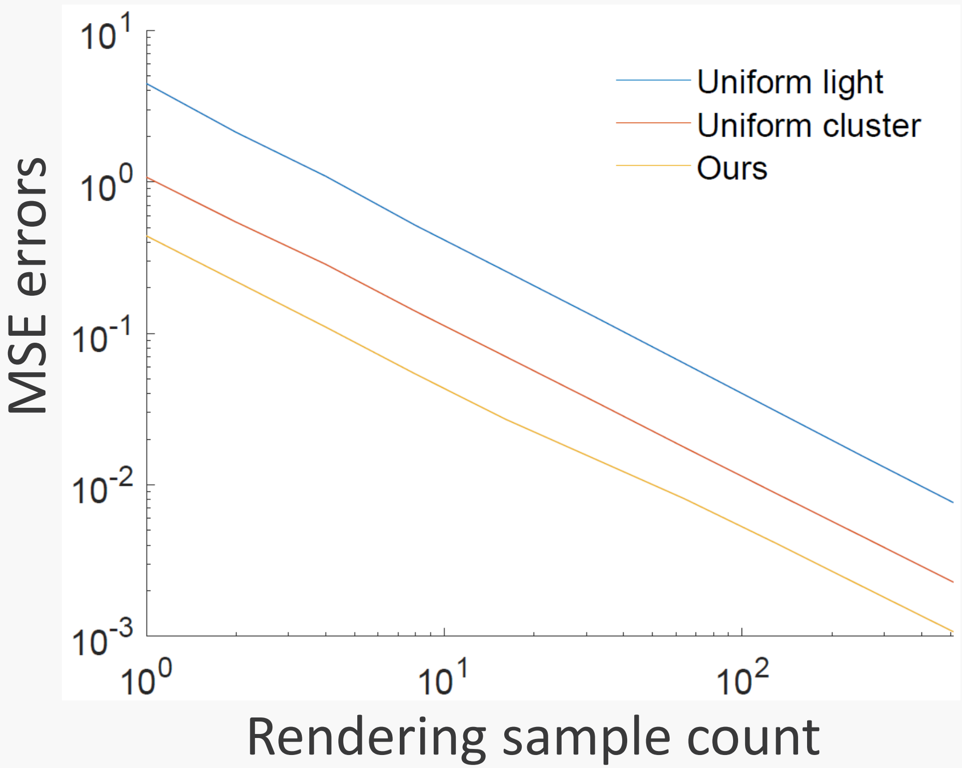


Light cluster sampling

- Sampling light cluster for direct lighting
  - Natural History Museum scene with 93 emitters

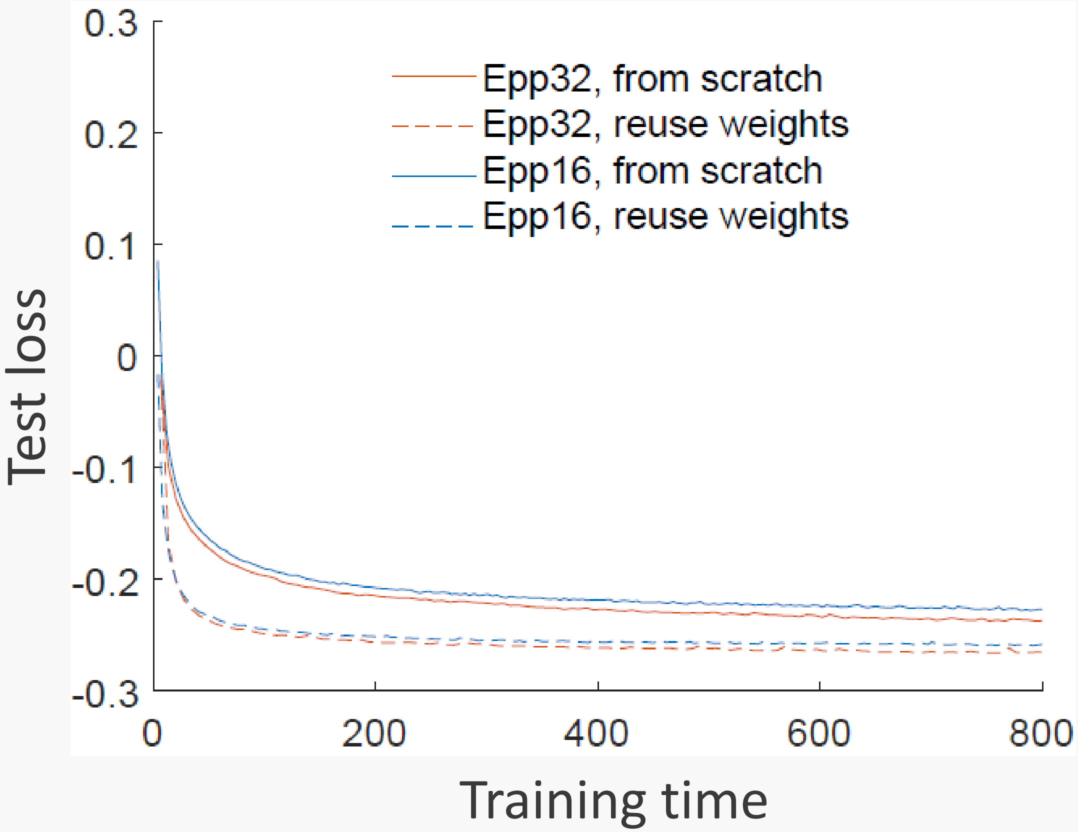


Error plot



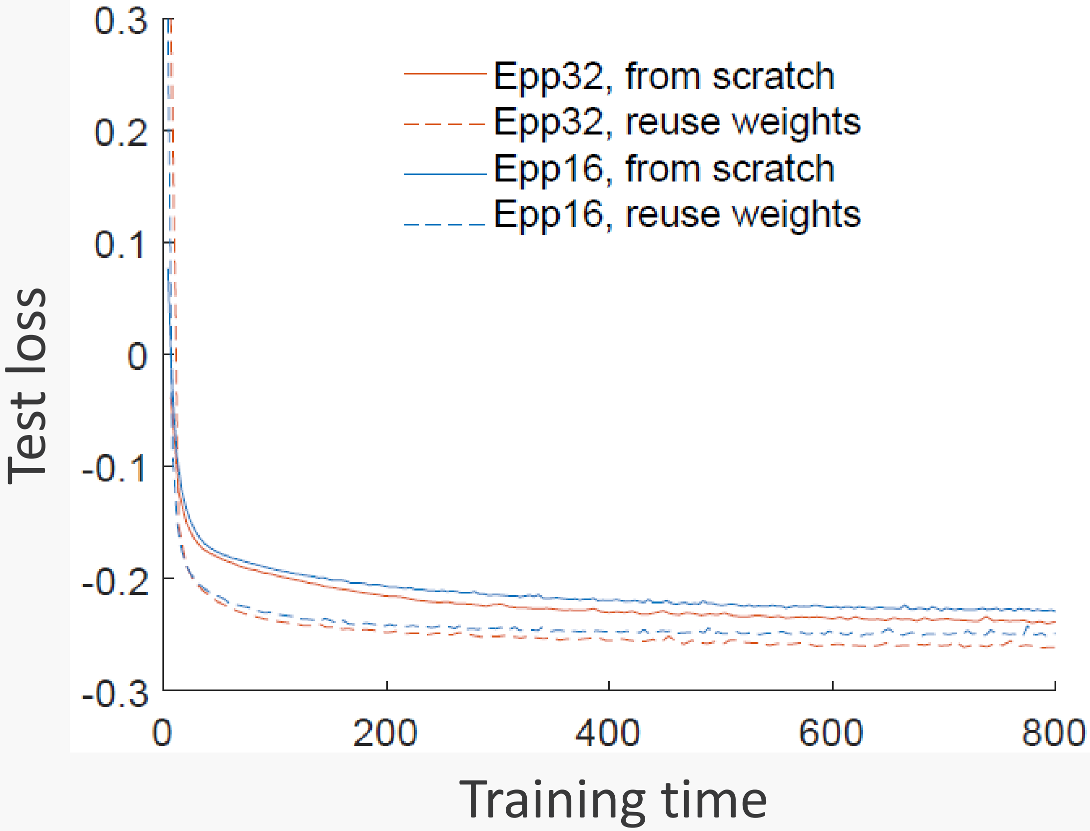
Rendering animation sequences

- Network reusing to render new camera views
- Amortize training costs over several views



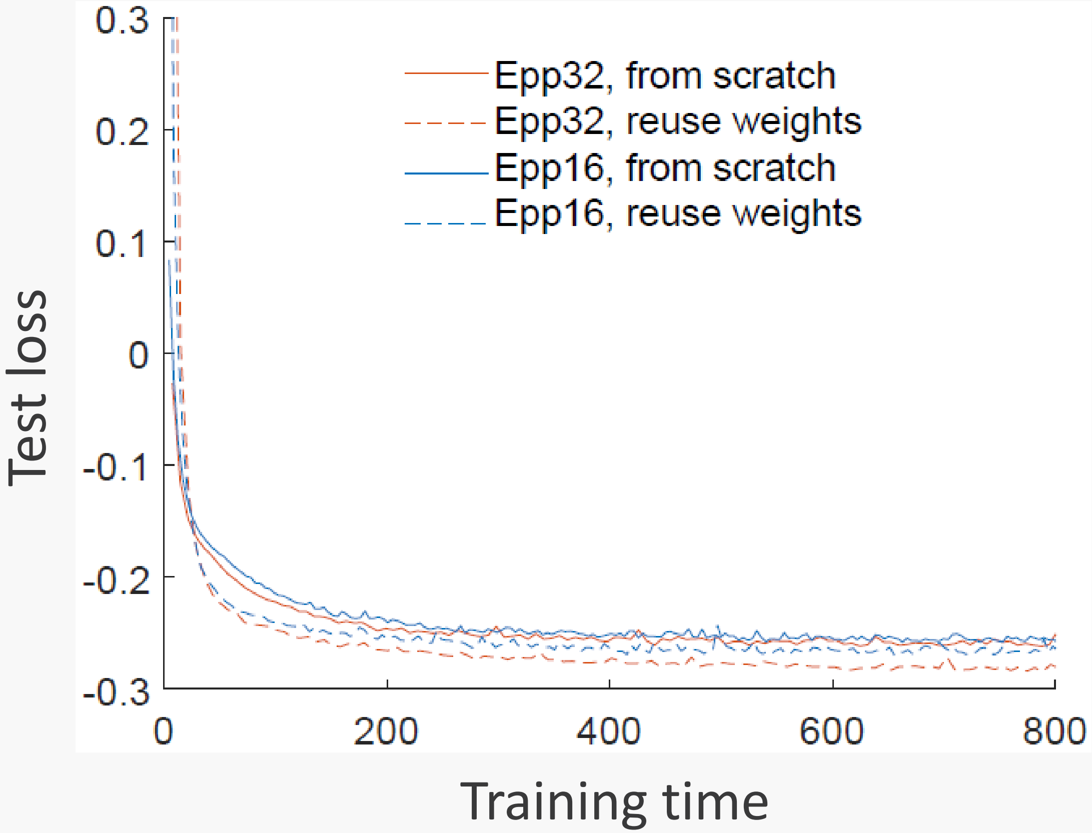
Rendering animation sequences

- Network reusing to render new camera views
- Amortize training costs over several views



Rendering animation sequences

- Network reusing to render new camera views
- Amortize training costs over several views



Limitations

- Computation cost
  - Acquiring data, and training requires orders of minutes per image
  - Related to the size of training data, and the scale of neural network
- Higher dimensional warps
  - Experiments up to 12 dimensions
  - Require large amounts of training data, time
  - Improvements under equal sample count rendering, but prohibitive training cost

Conclusions

- We proposed a novel approach to learn importance sampling
- Leverage neural network to perform non-linear warp
  - Generalize to different tasks
  - Treat existing renderer as a black box
  - Effective to reduce variance

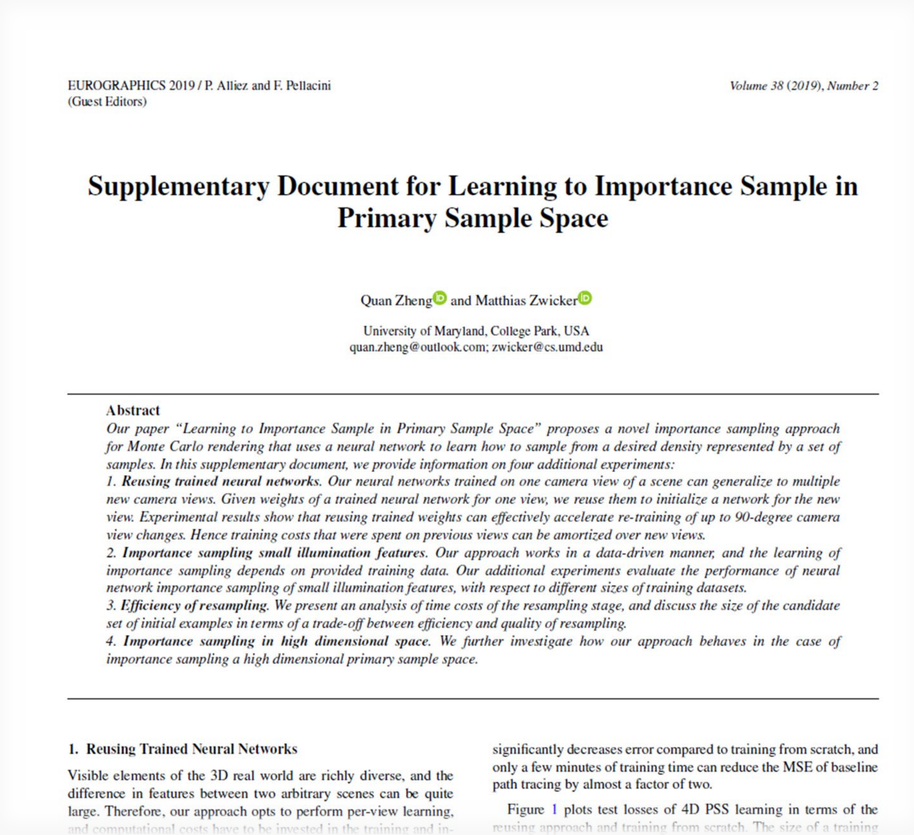
- Why not represent radiance function directly as neural network, instead of probability density used for sampling?
  - Advantage of using probability density: can do unbiased Monte Carlo sampling
  - Advantage of using radiance function: may be possible to come up with different type of algorithm to solve the rendering equation (different from series expansion and Monte Carlo integration)

Quan Zheng, qzhengcs@cs.umd.edu

Matthias Zwicker, zwicker@cs.umd.edu

More information:

https://tinyurl.com/LIS-PSS



EUROGRAPHICS 2019 / P. Alliez and F. Pellacini (Guest Editors) — *Volume 38 (2019), Number 2*

Supplementary Document for Learning to Importance Sample in Primary Sample Space

Quan Zheng and Matthias Zwicker

University of Maryland, College Park, USA
quan.zheng@outlook.com; zwicker@cs.umd.edu

Abstract

Our paper "Learning to Importance Sample in Primary Sample Space" proposes a novel importance sampling approach for Monte Carlo rendering that uses a neural network to learn how to sample from a desired density represented by a set of samples. In this supplementary document, we provide information on four additional experiments:
1. ***Reusing trained neural networks.*** *Our neural networks trained on one camera view of a scene can generalize to multiple new camera views. Given weights of a trained neural network for one view, we reuse them to initialize a network for the new view. Experimental results show that reusing trained weights can effectively accelerate re-training of up to 90-degree camera view changes. Hence training costs that were spent on previous views can be amortized over new views.*
2. ***Importance sampling small illumination features.*** *Our approach works in a data-driven manner, and the learning of importance sampling depends on provided training data. Our additional experiments evaluate the performance of neural network importance sampling of small illumination features, with respect to different sizes of training datasets.*
3. ***Efficiency of resampling.*** *We present an analysis of time costs of the resampling stage, and discuss the size of the candidate set of initial examples in terms of a trade-off between efficiency and quality of resampling.*
4. ***Importance sampling in high dimensional space.*** *We further investigate how our approach behaves in the case of importance sampling a high dimensional primary sample space.*

1. Reusing Trained Neural Networks

Visible elements of the 3D real world are richly diverse, and the difference in features between two arbitrary scenes can be quite large. Therefore, our approach opts to perform per-view learning, and computational costs have to be invested in the training and in-

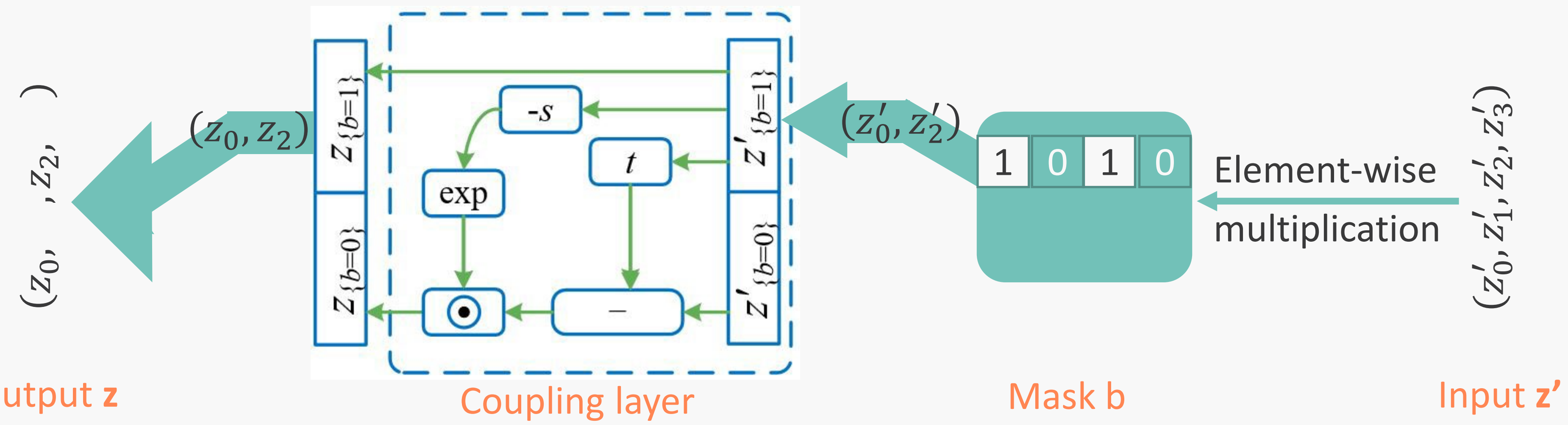
significantly decreases error compared to training from scratch, and only a few minutes of training time can reduce the MSE of baseline path tracing by almost a factor of two.

Figure 1 plots test losses of 4D PSS learning in terms of the reusing approach and training from scratch. The size of a training

Thank you for your attention!

Real NVP

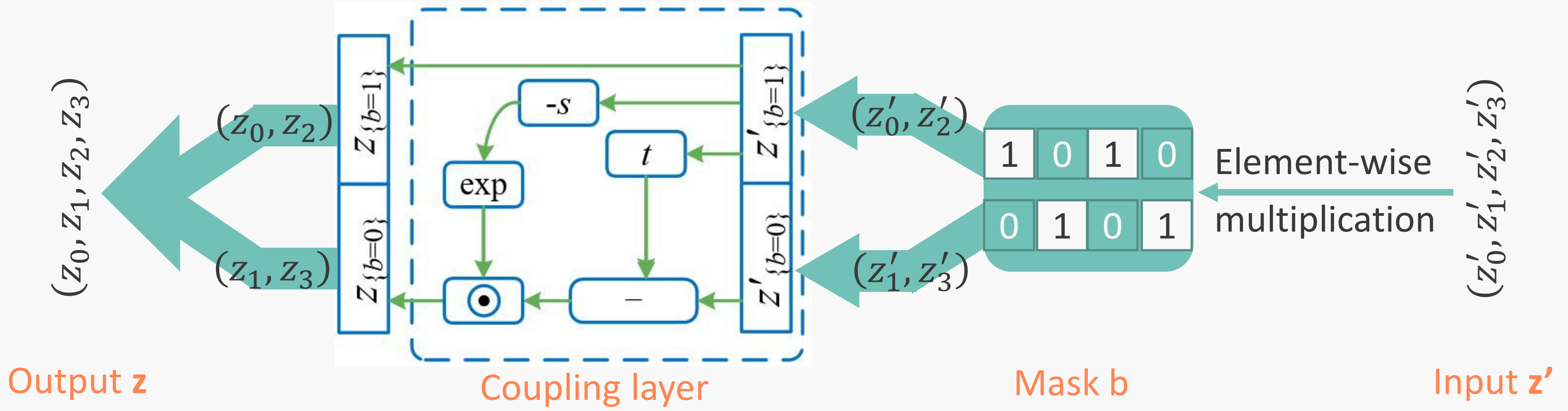
- Affine coupling layer (inverse mapping)



$$z_{\{b=1\}} = z'_{\{b=1\}}$$

Real NVP

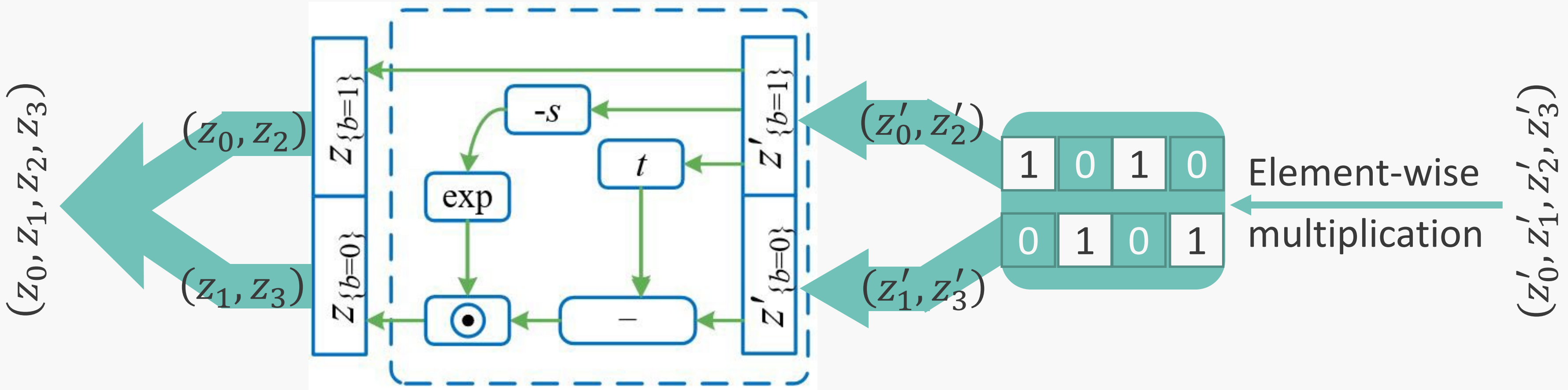
- Affine coupling layer (inverse mapping)



$$z_{\{b=1\}} = z'_{\{b=1\}}$$
$$z_{\{b=0\}} = \left(z'_{\{b=0\}} - t(z_{\{b=1\}})\right) \odot \exp(-s(z_{\{b=1\}}))$$

Real NVP

- Affine coupling layer (inverse mapping)



Output **z** Coupling layer Mask b Input **z'**

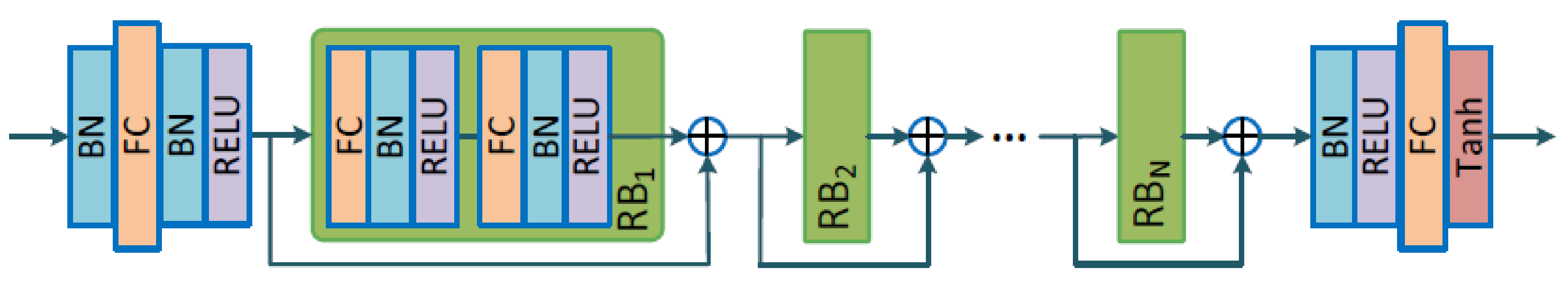
$$z_{\{b=1\}} = z'_{\{b=1\}}$$
$$z_{\{b=0\}} = \left(z'_{\{b=0\}} - t(z_{\{b=1\}})\right) \odot \exp(-s(z_{\{b=1\}}))$$

$$|\mathbf{J}| = \exp\left(-\sum_k s(z_{\{b=1\}})_k\right)$$

Note: Do not require inverse of s and t

Implementation of Functions s and t

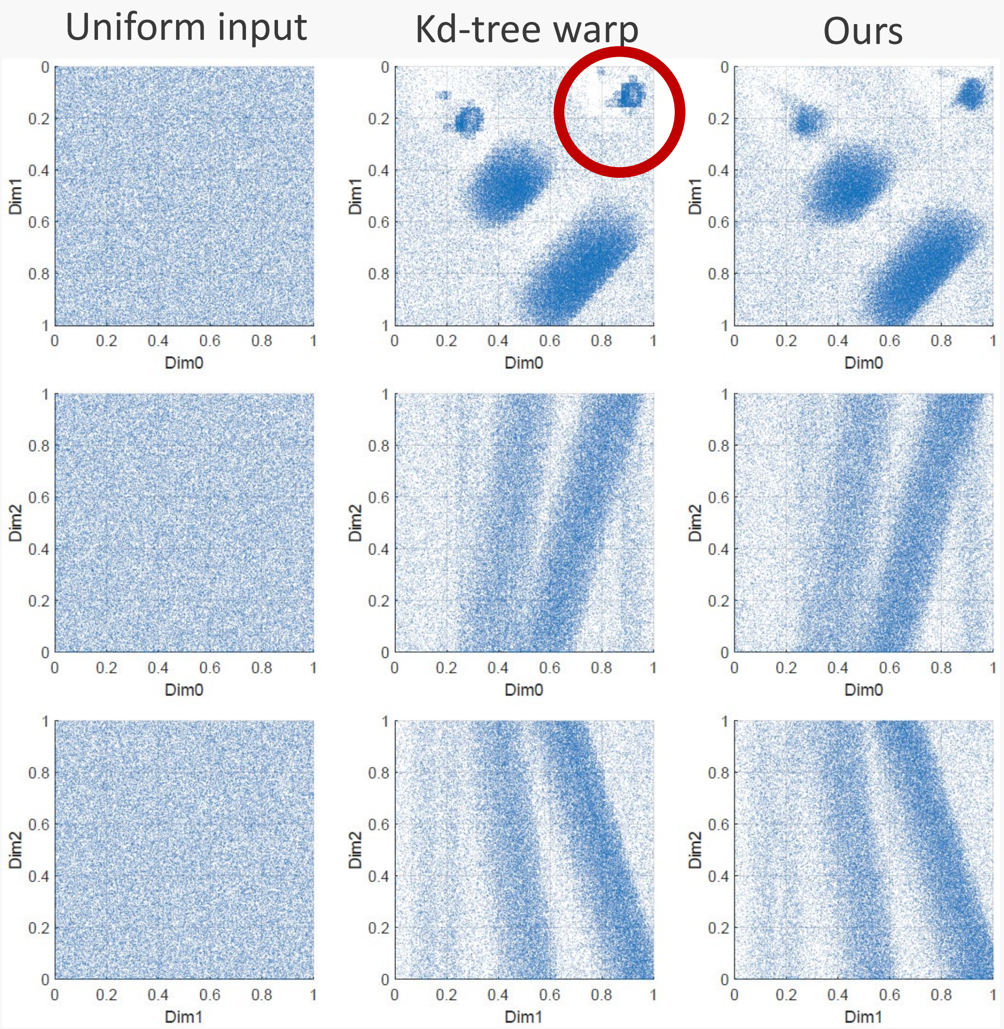
- Subject to few constraints
- We design them as neural networks
- Warp parameters θ are trainable network weights



FC: Fully-connected layer
BN: Batch normalization
RB_i: i-th residual block
RELU: Rectified linear unit

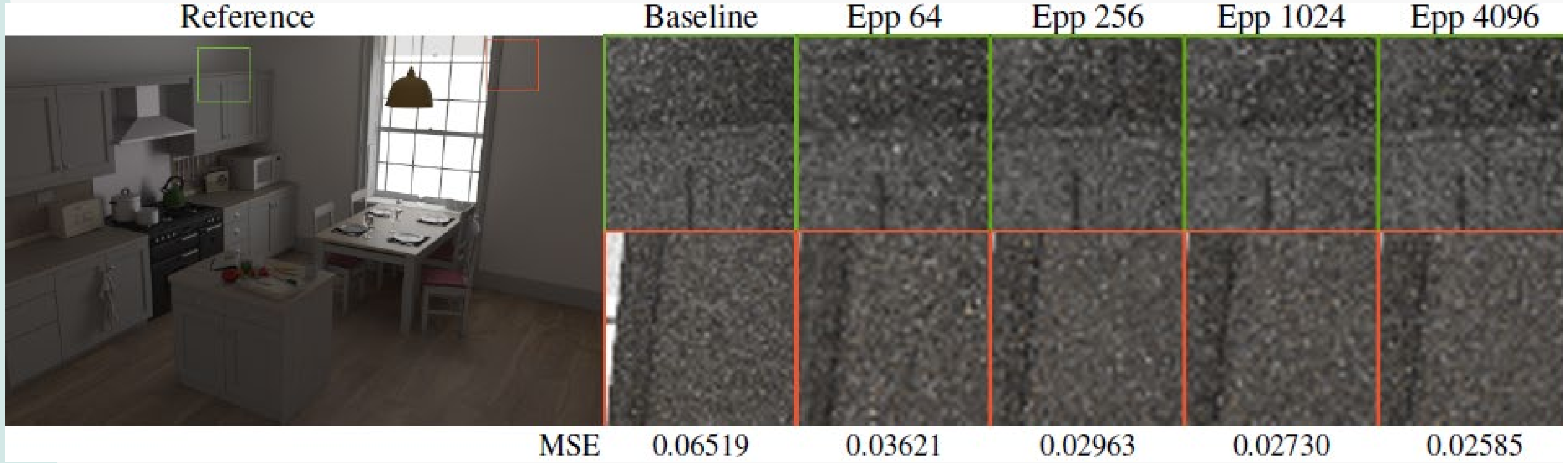
Visualization of sample warping

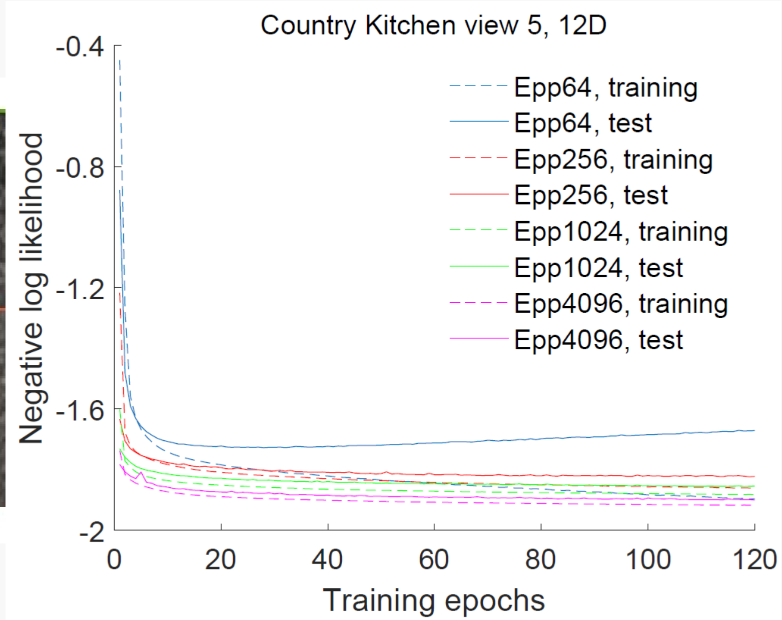
- Sample distribution comparison
  - Pool Ball scene with motion blur
  - 3D warping (image plane, time)
- Comparison
  - Ours achieves desired density
  - Sample distribution of kd-tree warping is blocky



Limitations

- Higher dimensional warp
  - Country Kitchen scene, 12D learning
  - Require much more training data
  - Epp 4096 dataset took 26h to train





(Equal sample count 128 spp comparisons)

Rendering animation sequences

- Network reusing to render new camera views
- Amortize training costs over several views

View 1
Original

View 2
10 degrees

View 3
45 degrees

View 4
90 degrees

Rendering animation sequences

- Network reusing
  - Generalize to multiple new camera views
  - Amortize spent training costs over new views