Path to Neural Networks

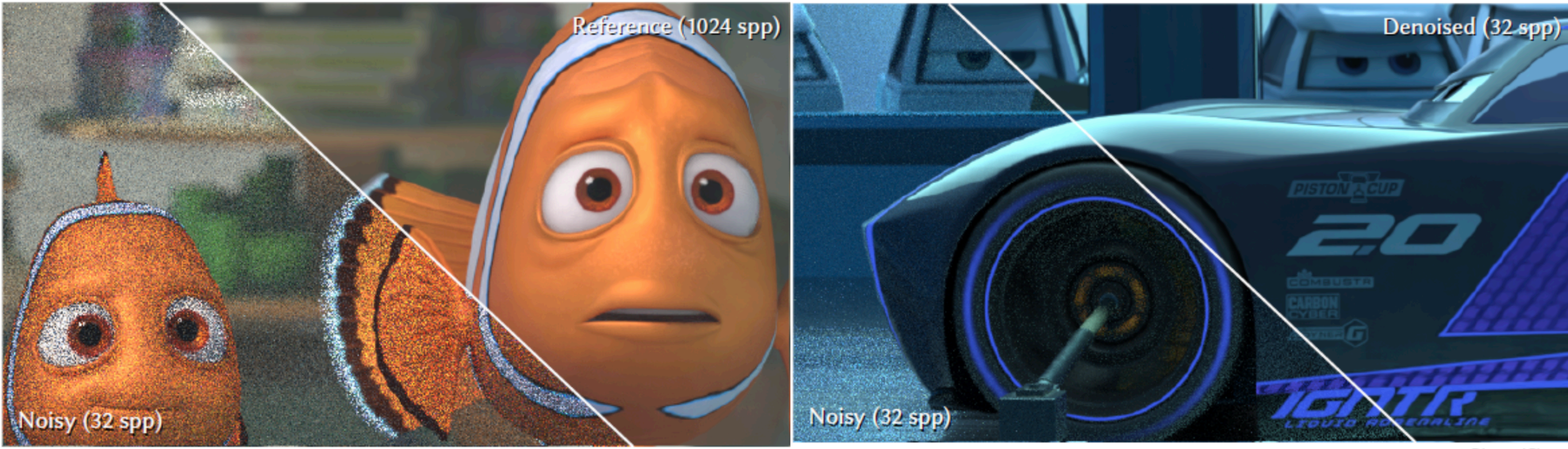


Image courtesy Bako et al. [2017]

Gurprit Singh

Introduction
Denoising using Data

Introduction
Denoising using Data

MLP based Denoising

Introduction
Denoising using Data

MLP based Denoising

Neural Importance Sampling

Filtering Monte Carlo Noise From Random Parameters

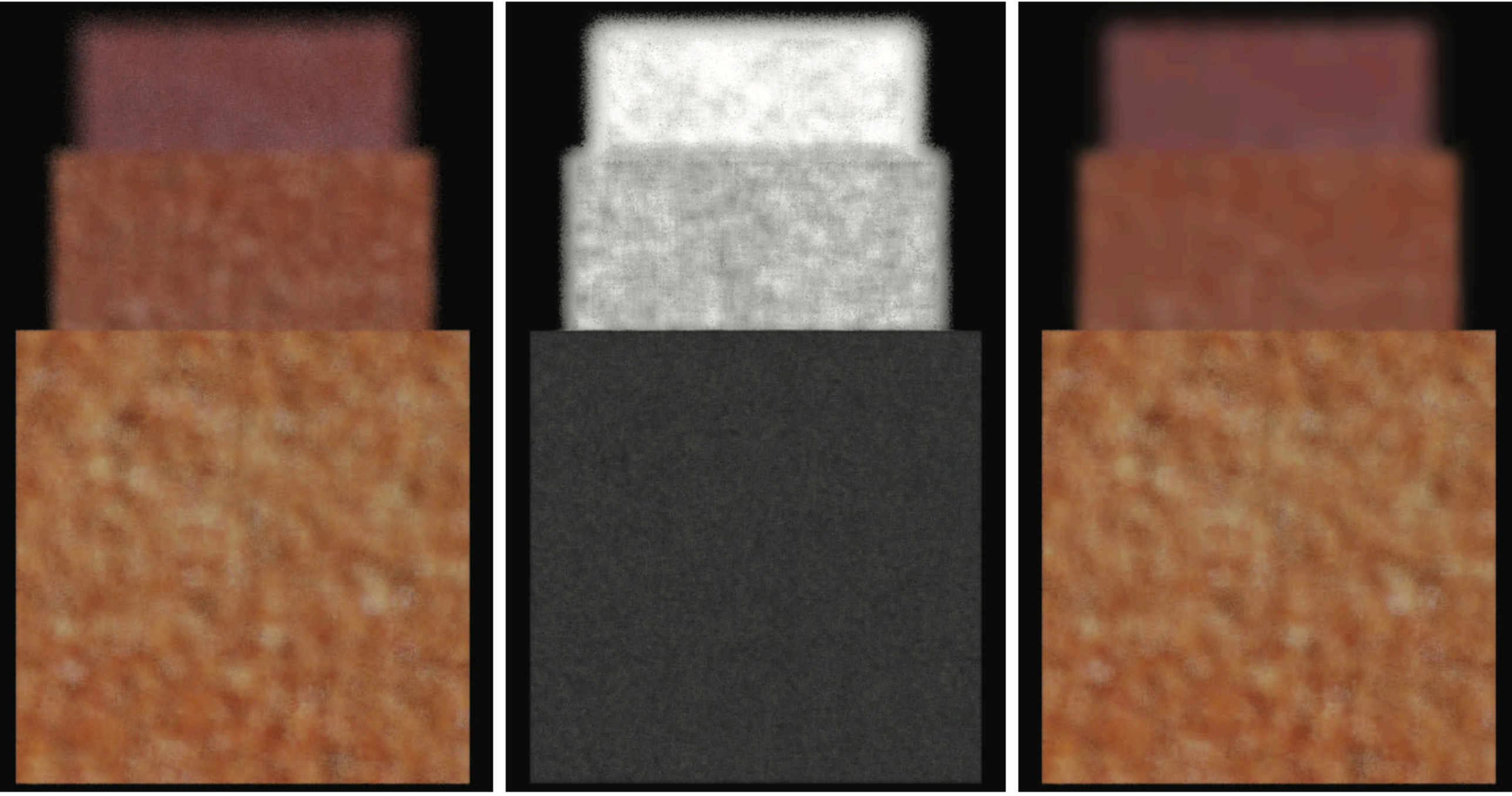
Sen and Darabi [2012]

input Monte Carlo (8 samples/pixel)

after RPF (8 samples/pixel)

High-dimensional Monte Carlo Integration

$$I(i,j) = \int_{i-\frac{1}{2}}^{i+\frac{1}{2}} \int_{j-\frac{1}{2}}^{j+\frac{1}{2}} \cdots \int_{-1}^{1} \int_{-1}^{1} \int_{t_0}^{t_1} f(x, y, \cdots, u, v, t)\, dt\, dv\, du \cdots dy\, dx$$



(a) Input MC (8 spp) **(b)** Dependency on (u, v) **(c)** Our approach (RPF)

Parameters in Monte Carlo estimator

Random parameters: $\mathbf{r} = \{r_1, r_2, \ldots, r_n\}$

Color: $\mathbf{c}_i \Leftarrow f(\underbrace{\mathbf{p}_{i,1}, \mathbf{p}_{i,2}}_{\text{screen position}}; \underbrace{\mathbf{r}_{i,1}, \mathbf{r}_{i,2}, \ldots, \mathbf{r}_{i,n}}_{\text{random parameters}})$

Random Parameters Classification

Random parameter for each pixel :

$$\mathbf{x}_i \Leftarrow f(\mathbf{p}_{i,1}, \mathbf{p}_{i,2}; \mathbf{r}_{i,1}, \mathbf{r}_{i,2}, \ldots, \mathbf{r}_{i,n})$$

$$\mathbf{x}_i = \{\underbrace{\mathbf{p}_{i,1}, \mathbf{p}_{i,2}}_{\text{screen position}}; \underbrace{\mathbf{r}_{i,1}, \ldots, \mathbf{r}_{i,n}}_{\text{random parameters}}; \underbrace{\mathbf{f}_{i,1}, \ldots, \mathbf{f}_{i,m}}_{\text{scene features}}; \underbrace{\mathbf{c}_{i,1}, \mathbf{c}_{i,2}, \mathbf{c}_{i,3}}_{\text{sample color}}\}$$

Random Parameters Classification

Random parameter for each pixel :

$$\mathbf{x}_i \Leftarrow f(\mathbf{p}_{i,1}, \mathbf{p}_{i,2}; \mathbf{r}_{i,1}, \mathbf{r}_{i,2}, \ldots, \mathbf{r}_{i,n})$$

$$\mathbf{x}_i = \{\underbrace{\mathbf{p}_{i,1}, \mathbf{p}_{i,2}}_{\text{screen position}}; \underbrace{\mathbf{r}_{i,1}, \ldots, \mathbf{r}_{i,n}}_{\text{random parameters}}; \underbrace{\mathbf{f}_{i,1}, \ldots, \mathbf{f}_{i,m}}_{\text{scene features}}; \underbrace{\mathbf{c}_{i,1}, \mathbf{c}_{i,2}, \mathbf{c}_{i,3}}_{\text{sample color}}\}$$

Random Parameters Classification

Random parameter for each pixel :

$$\mathbf{x}_i \Leftarrow f(\mathbf{p}_{i,1}, \mathbf{p}_{i,2}; \mathbf{r}_{i,1}, \mathbf{r}_{i,2}, \ldots, \mathbf{r}_{i,n})$$

$$\mathbf{x}_i = \{\underbrace{\mathbf{p}_{i,1}, \mathbf{p}_{i,2}}_{\text{screen position}}; \underbrace{\mathbf{r}_{i,1}, \ldots, \mathbf{r}_{i,n}}_{\text{random parameters}}; \underbrace{\mathbf{f}_{i,1}, \ldots, \mathbf{f}_{i,m}}_{\text{scene features}}; \underbrace{\mathbf{c}_{i,1}, \mathbf{c}_{i,2}, \mathbf{c}_{i,3}}_{\text{sample color}}\}$$

Random Parameters Classification

Random parameter for each pixel :

$$\mathbf{x}_i \Leftarrow f(\mathbf{p}_{i,1}, \mathbf{p}_{i,2}; \mathbf{r}_{i,1}, \mathbf{r}_{i,2}, \ldots, \mathbf{r}_{i,n})$$

$$\mathbf{x}_i = \{\underbrace{\mathbf{p}_{i,1}, \mathbf{p}_{i,2}}_{\text{screen position}}; \underbrace{\mathbf{r}_{i,1}, \ldots, \mathbf{r}_{i,n}}_{\text{random parameters}}; \underbrace{\mathbf{f}_{i,1}, \ldots, \mathbf{f}_{i,m}}_{\text{scene features}}; \underbrace{\mathbf{c}_{i,1}, \mathbf{c}_{i,2}, \mathbf{c}_{i,3}}_{\text{sample color}}\}$$

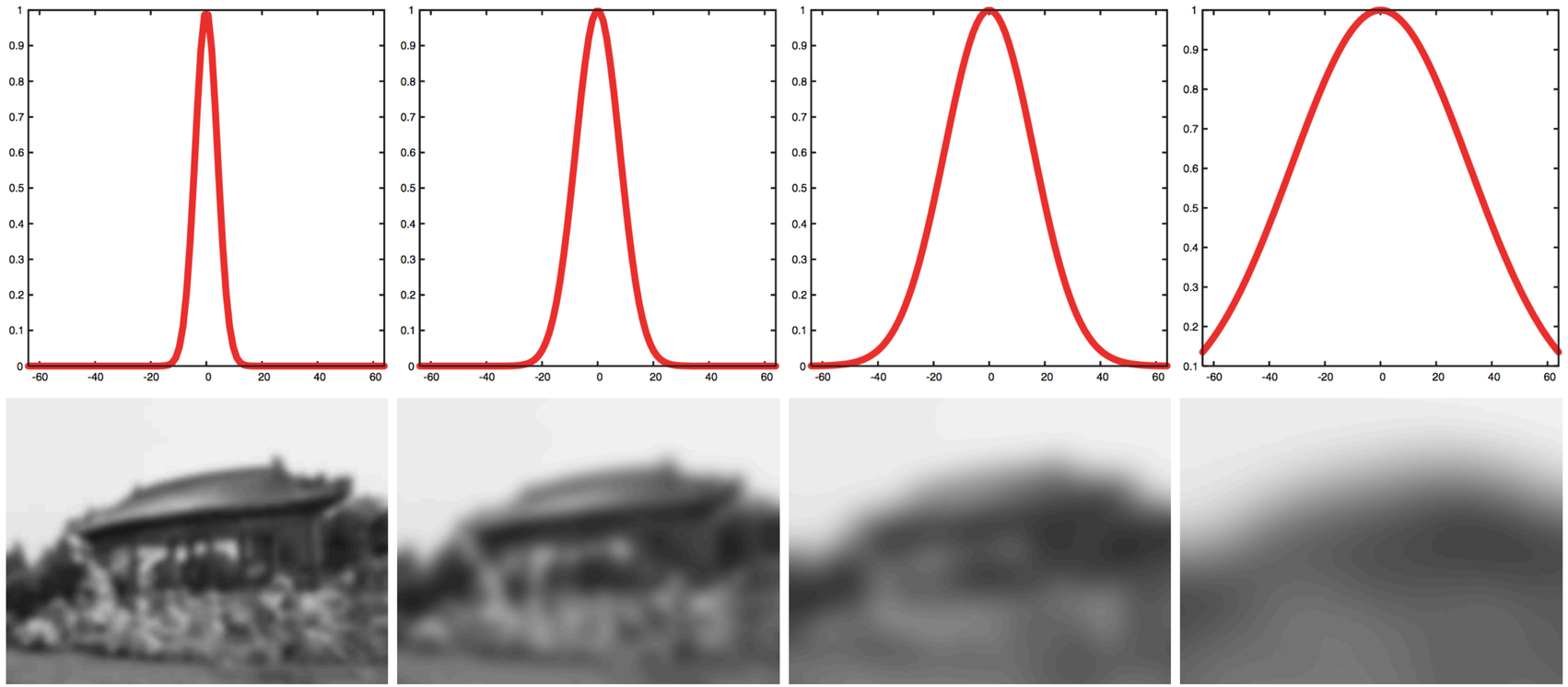
Random Parameters Classification

Random parameter for each pixel :

$$\mathbf{x}_i \Leftarrow f(\mathbf{p}_{i,1}, \mathbf{p}_{i,2}; \mathbf{r}_{i,1}, \mathbf{r}_{i,2}, \ldots, \mathbf{r}_{i,n})$$

$$\mathbf{x}_i = \{\underbrace{\mathbf{p}_{i,1}, \mathbf{p}_{i,2}}_{\text{screen position}}; \underbrace{\mathbf{r}_{i,1}, \ldots, \mathbf{r}_{i,n}}_{\text{random parameters}}; \underbrace{\mathbf{f}_{i,1}, \ldots, \mathbf{f}_{i,m}}_{\text{scene features}}; \underbrace{\mathbf{c}_{i,1}, \mathbf{c}_{i,2}, \mathbf{c}_{i,3}}_{\text{sample color}}\}$$

Gaussian Filtering

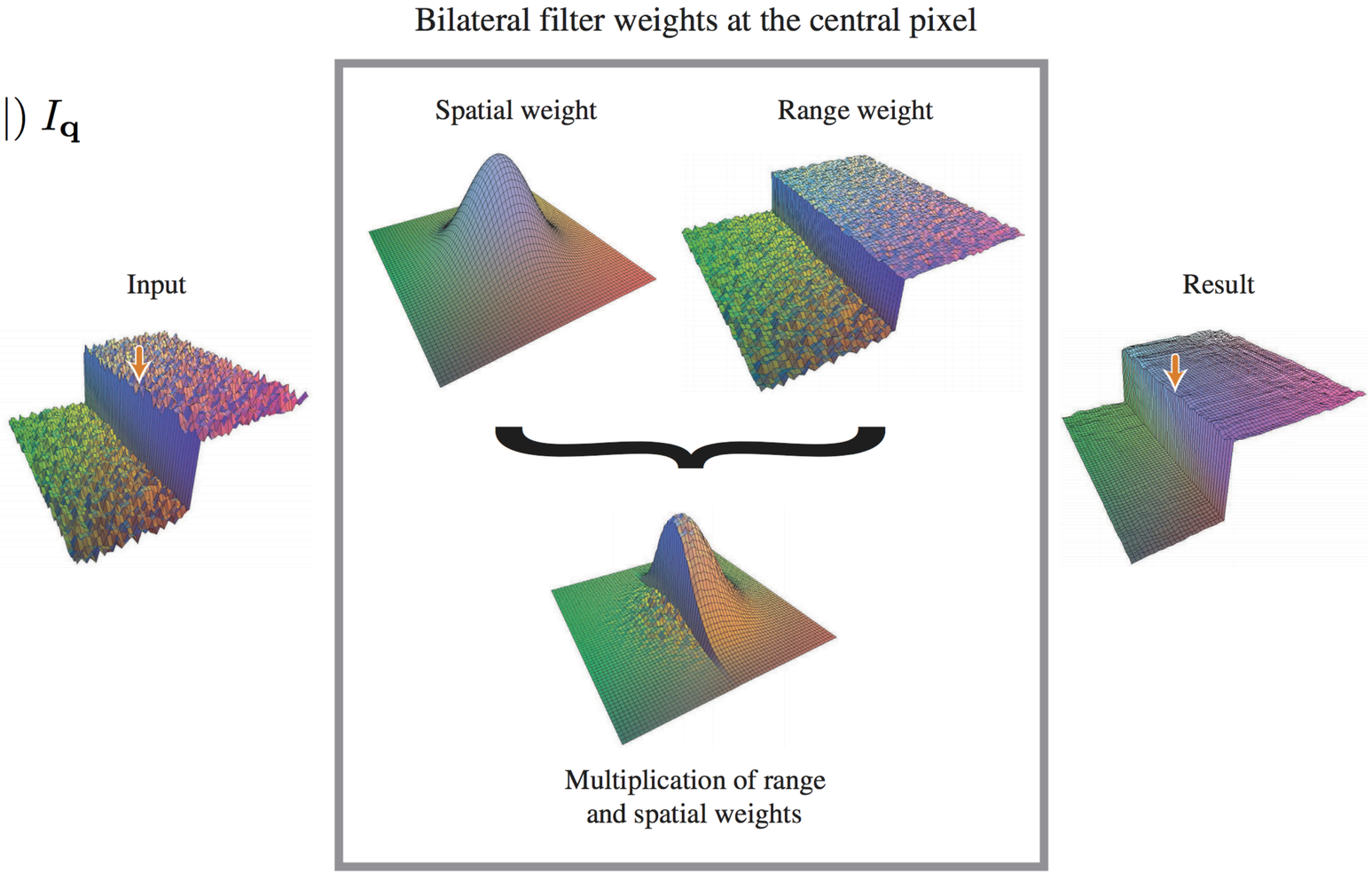


$$GC[I]_{\mathbf{p}} = \sum_{\mathbf{q} \in \mathcal{S}} G_\sigma(\|\mathbf{p} - \mathbf{q}\|)\, I_{\mathbf{q}}, \qquad G_\sigma(x) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2}{2\sigma^2}\right)$$

Bilateral Filtering

$$BF[I]_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q} \in \mathcal{S}} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|)\, G_{\sigma_r}(|I_{\mathbf{p}} - I_{\mathbf{q}}|)\, I_{\mathbf{q}}$$

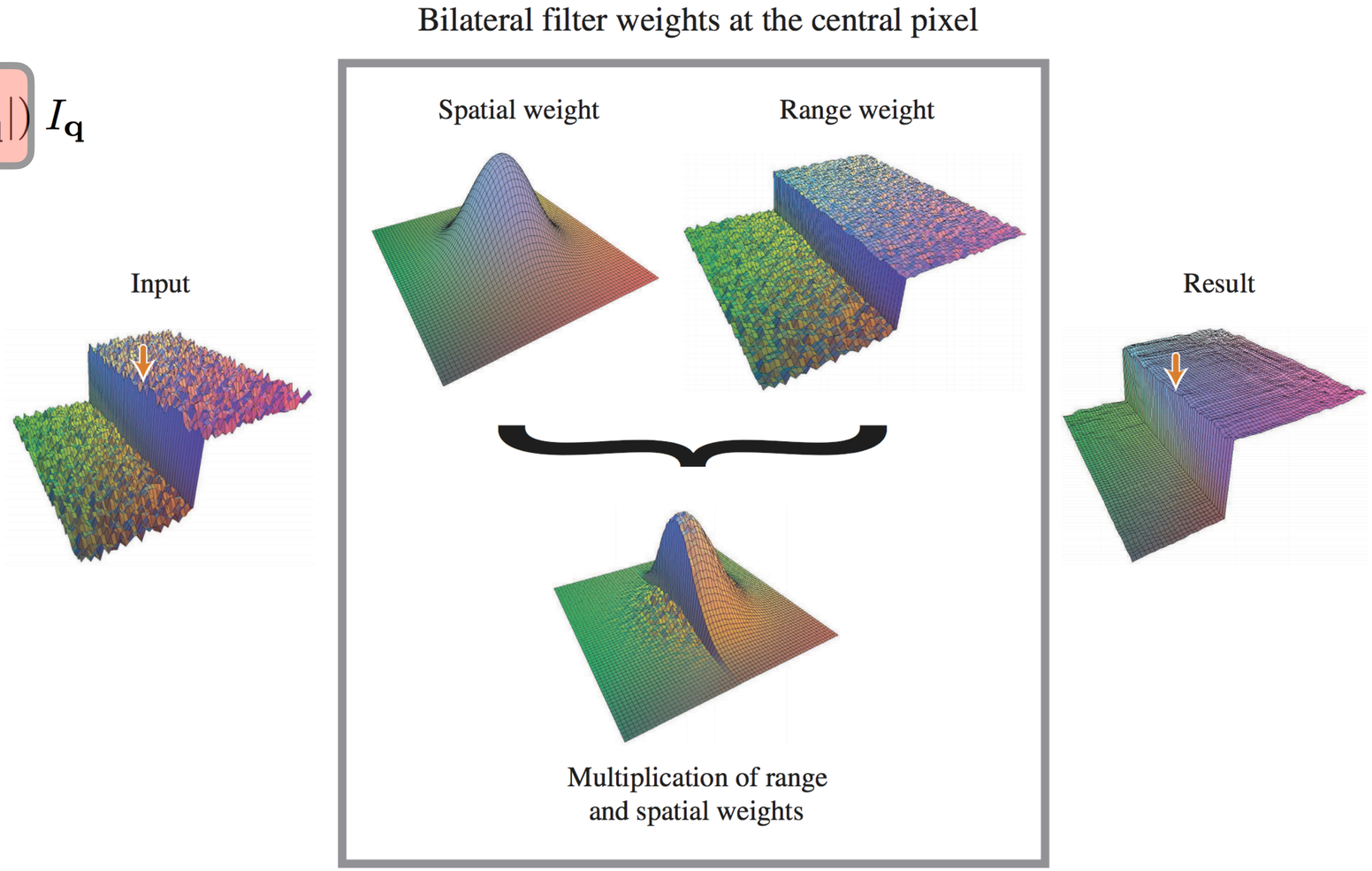
$$W_{\mathbf{p}} = \sum_{\mathbf{q} \in \mathcal{S}} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|)\, G_{\sigma_r}(|I_{\mathbf{p}} - I_{\mathbf{q}}|)$$



Bilateral Filtering

$$BF[I]_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q} \in \mathcal{S}} G_{\sigma_\mathrm{s}}(\|\mathbf{p} - \mathbf{q}\|)\, G_{\sigma_\mathrm{r}}(|I_{\mathbf{p}} - I_{\mathbf{q}}|)\, I_{\mathbf{q}}$$

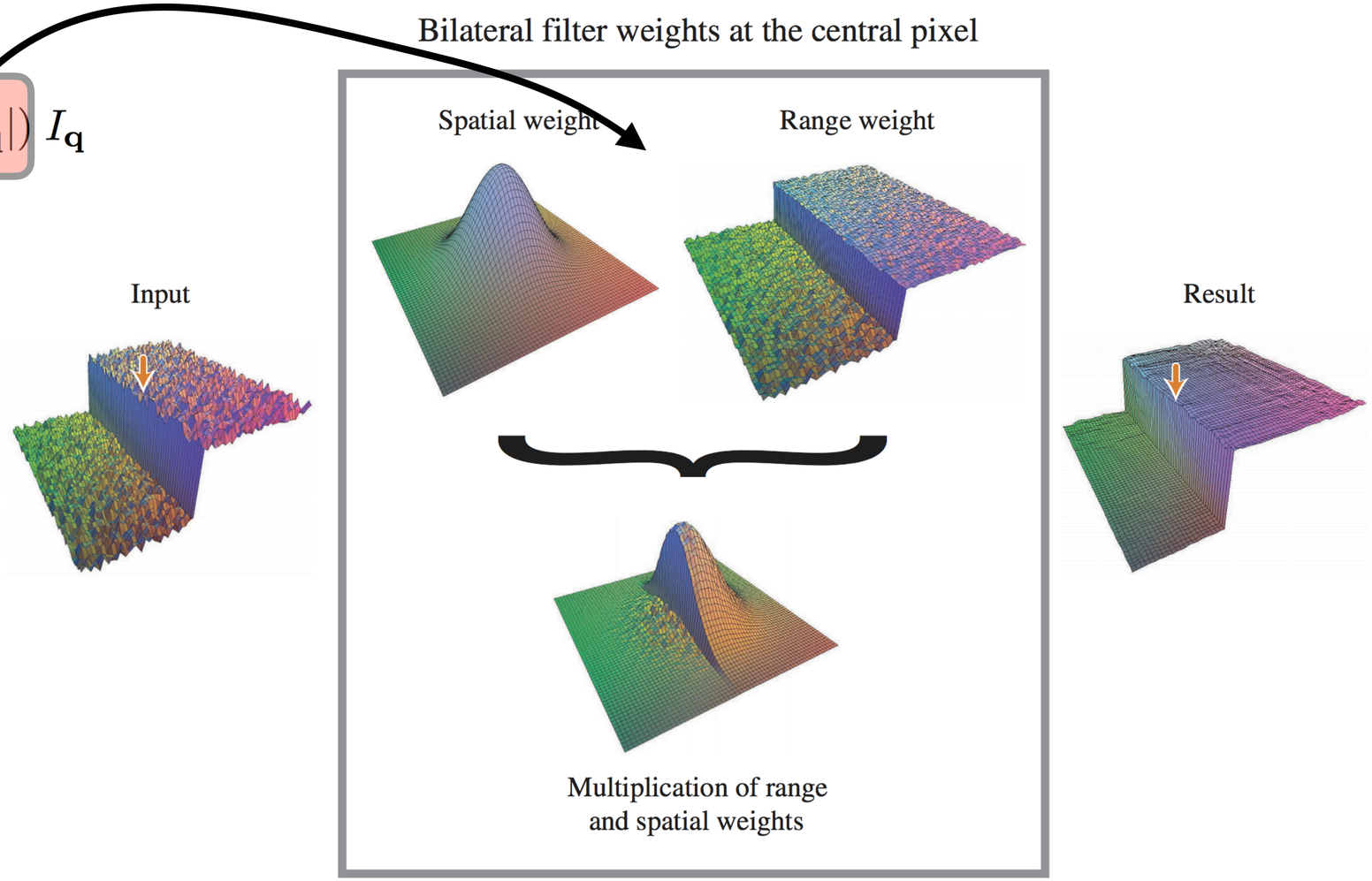
$$W_{\mathbf{p}} = \sum_{\mathbf{q} \in \mathcal{S}} G_{\sigma_\mathrm{s}}(\|\mathbf{p} - \mathbf{q}\|)\, G_{\sigma_\mathrm{r}}(|I_{\mathbf{p}} - I_{\mathbf{q}}|)$$



Bilateral Filtering

$$BF[I]_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q} \in \mathcal{S}} G_{\sigma_\mathrm{s}}(\|\mathbf{p} - \mathbf{q}\|)\, G_{\sigma_\mathrm{r}}(|I_{\mathbf{p}} - I_{\mathbf{q}}|)\, I_{\mathbf{q}}$$

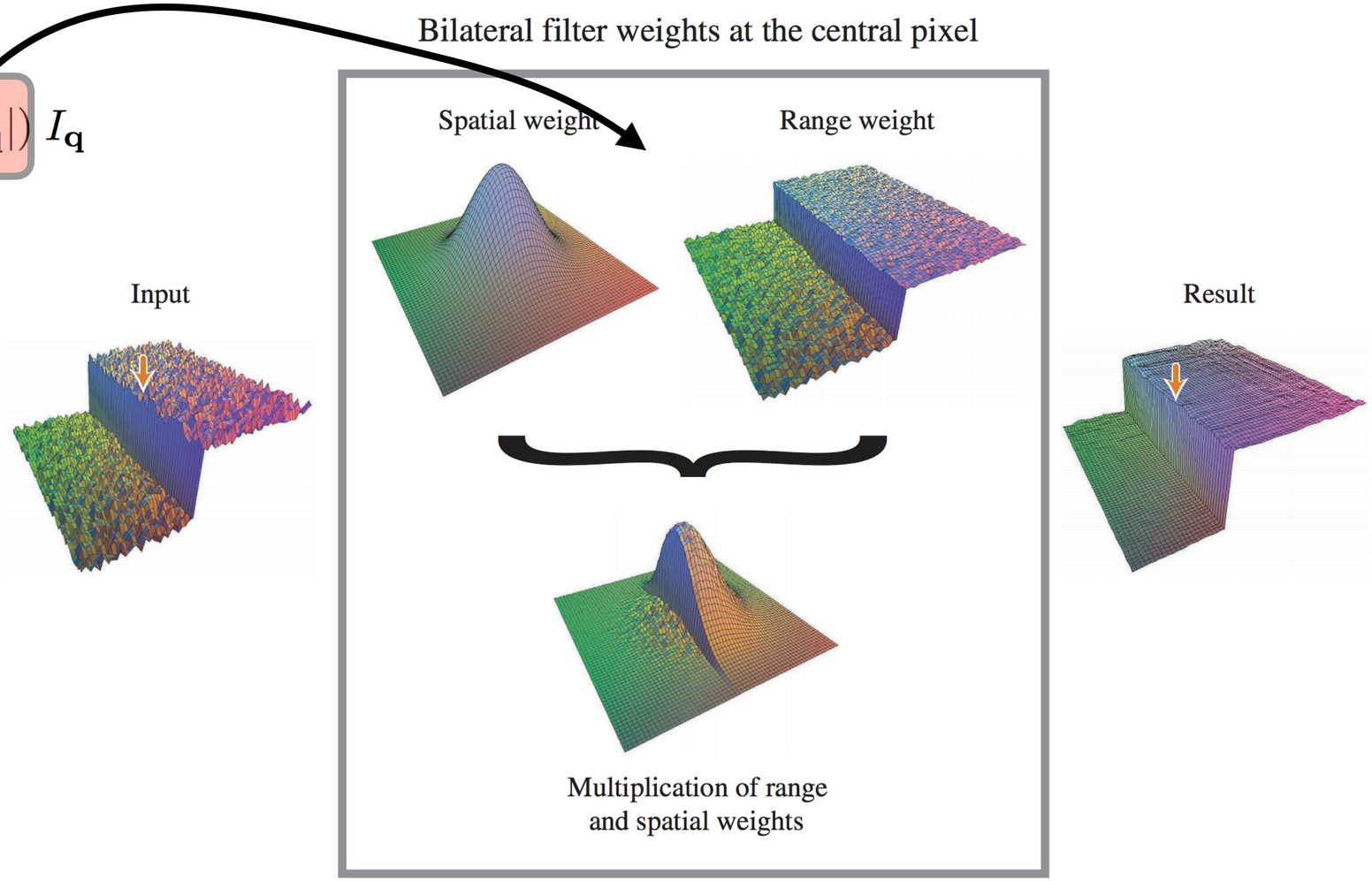
$$W_{\mathbf{p}} = \sum_{\mathbf{q} \in \mathcal{S}} G_{\sigma_\mathrm{s}}(\|\mathbf{p} - \mathbf{q}\|)\, G_{\sigma_\mathrm{r}}(|I_{\mathbf{p}} - I_{\mathbf{q}}|)$$



Bilateral Filtering

$$BF[I]_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q} \in \mathcal{S}} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|)\, G_{\sigma_r}(|I_{\mathbf{p}} - I_{\mathbf{q}}|)\, I_{\mathbf{q}}$$

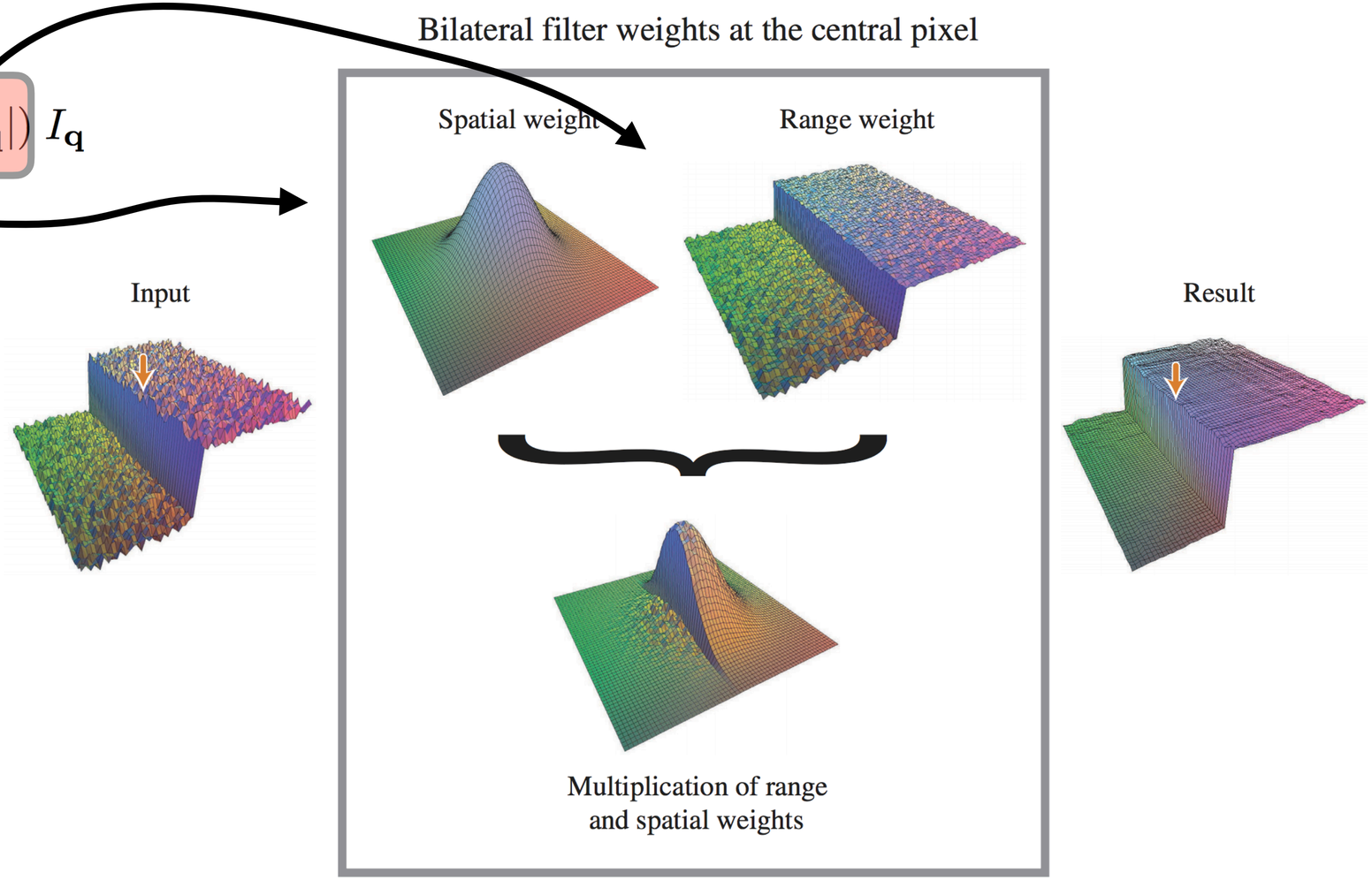
$$W_{\mathbf{p}} = \sum_{\mathbf{q} \in \mathcal{S}} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|)\, G_{\sigma_r}(|I_{\mathbf{p}} - I_{\mathbf{q}}|)$$



Bilateral Filtering

$$BF[I]_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q} \in \mathcal{S}} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|)\, G_{\sigma_r}(|I_{\mathbf{p}} - I_{\mathbf{q}}|)\, I_{\mathbf{q}}$$

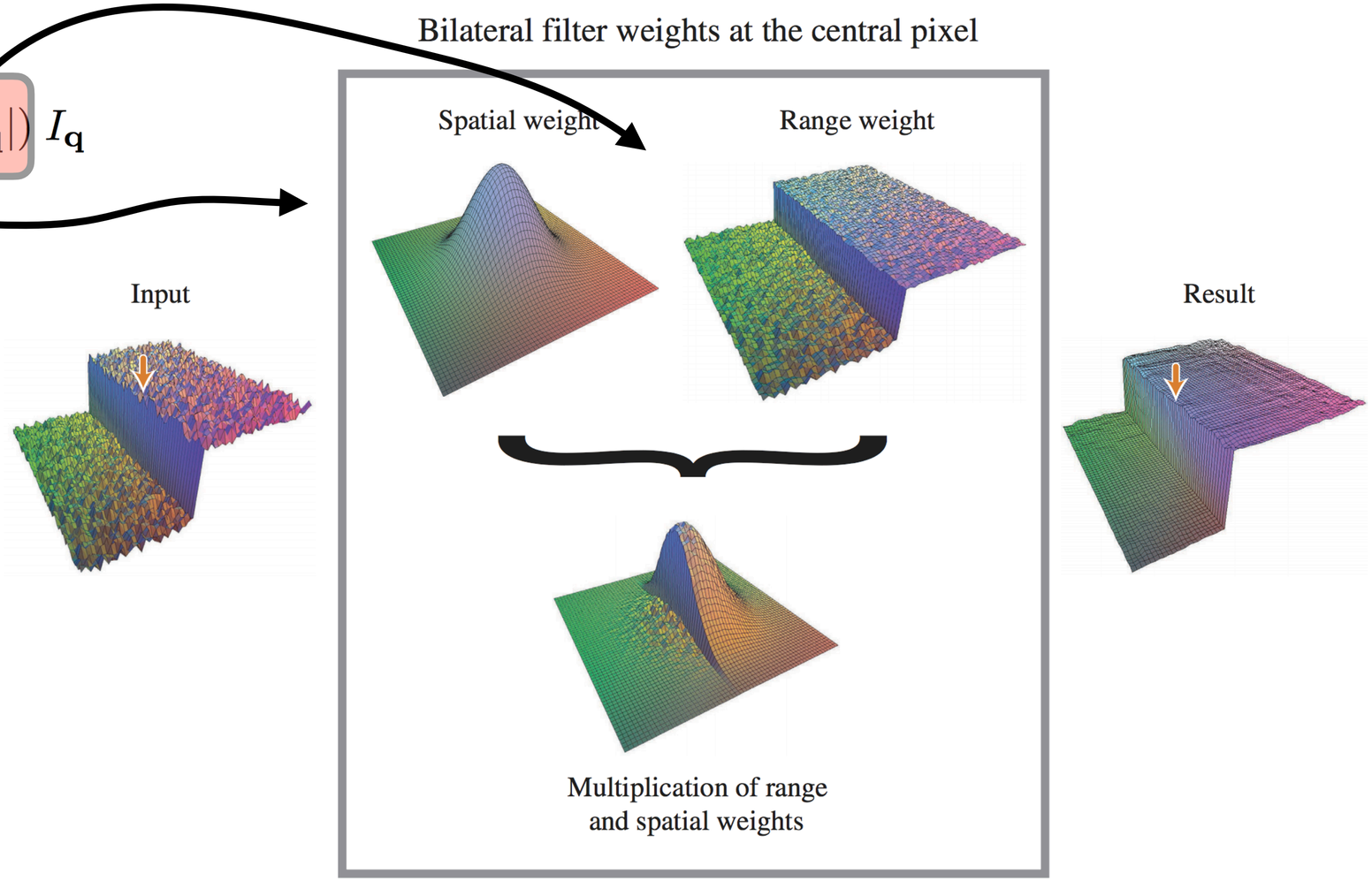
$$W_{\mathbf{p}} = \sum_{\mathbf{q} \in \mathcal{S}} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|)\, G_{\sigma_r}(|I_{\mathbf{p}} - I_{\mathbf{q}}|)$$



Bilateral Filtering

$$BF[I]_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q} \in \mathcal{S}} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|)\, G_{\sigma_r}(|I_{\mathbf{p}} - I_{\mathbf{q}}|)\, I_{\mathbf{q}}$$

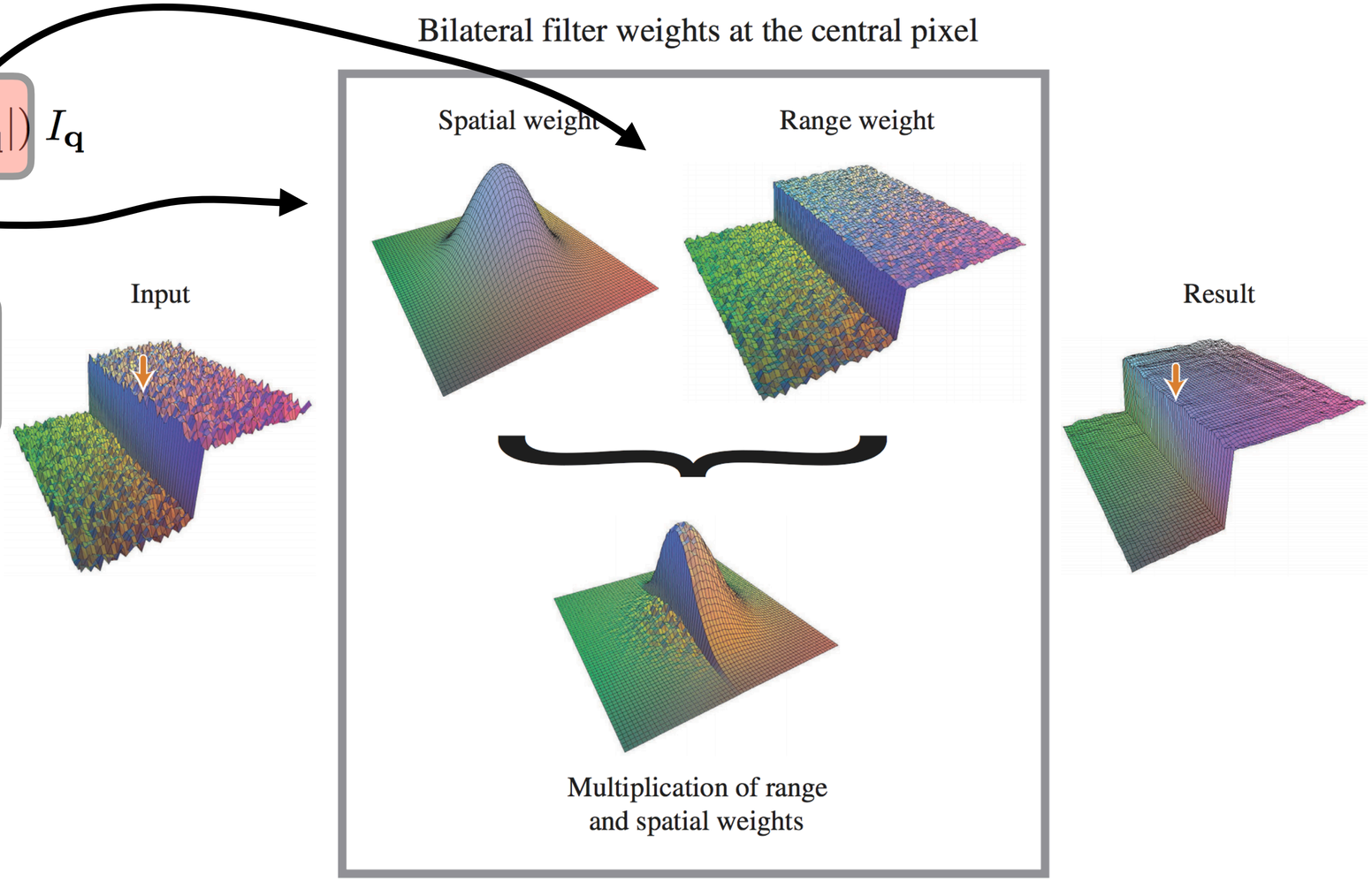
$$W_{\mathbf{p}} = \sum_{\mathbf{q} \in \mathcal{S}} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|)\, G_{\sigma_r}(|I_{\mathbf{p}} - I_{\mathbf{q}}|)$$



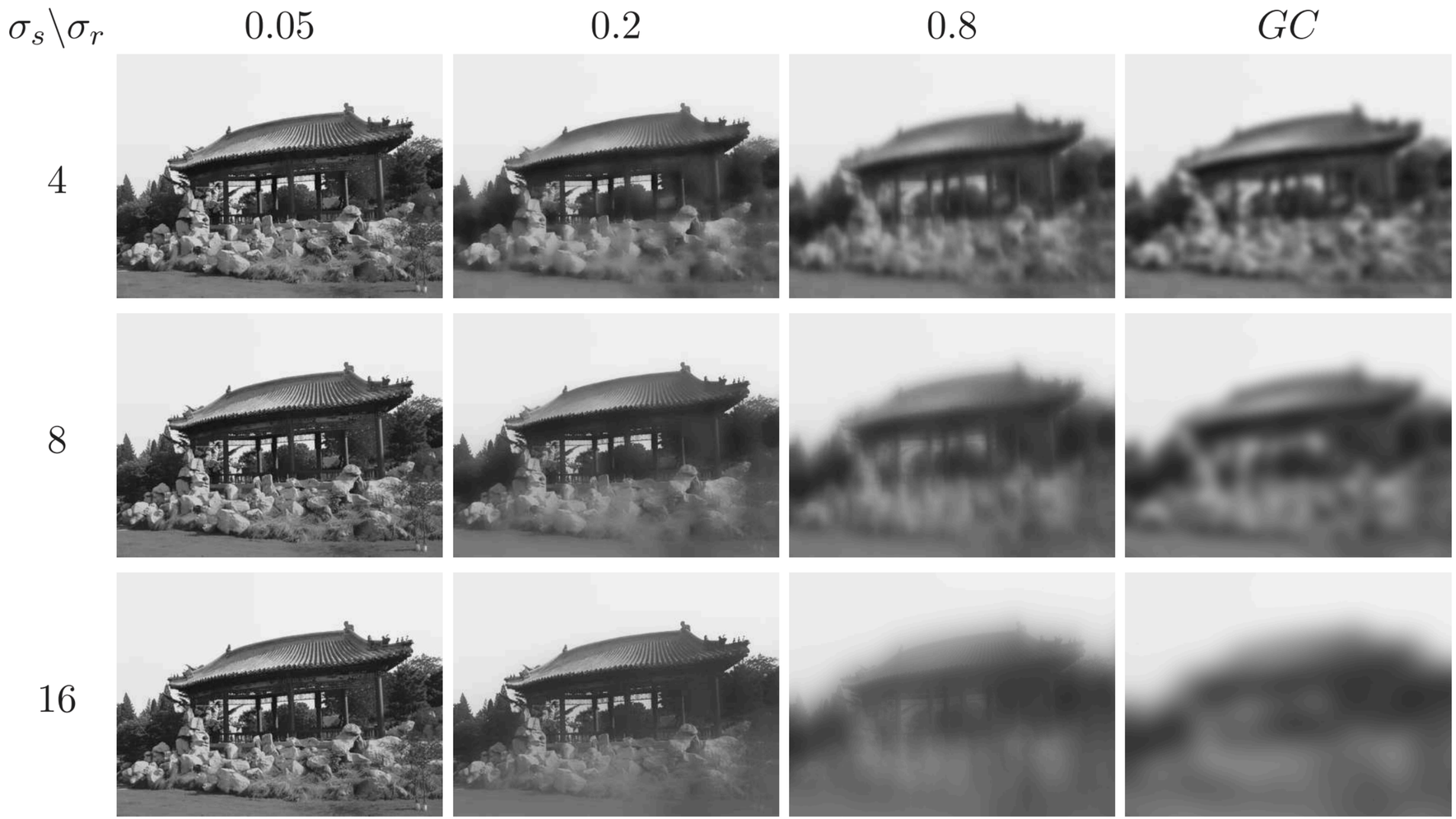
Bilateral Filtering

$$BF[I]_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q} \in \mathcal{S}} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|)\, G_{\sigma_r}(|I_{\mathbf{p}} - I_{\mathbf{q}}|)\, I_{\mathbf{q}}$$

$$W_{\mathbf{p}} = \sum_{\mathbf{q} \in \mathcal{S}} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|)\, G_{\sigma_r}(|I_{\mathbf{p}} - I_{\mathbf{q}}|)$$



Bilateral vs Gaussian Filtering



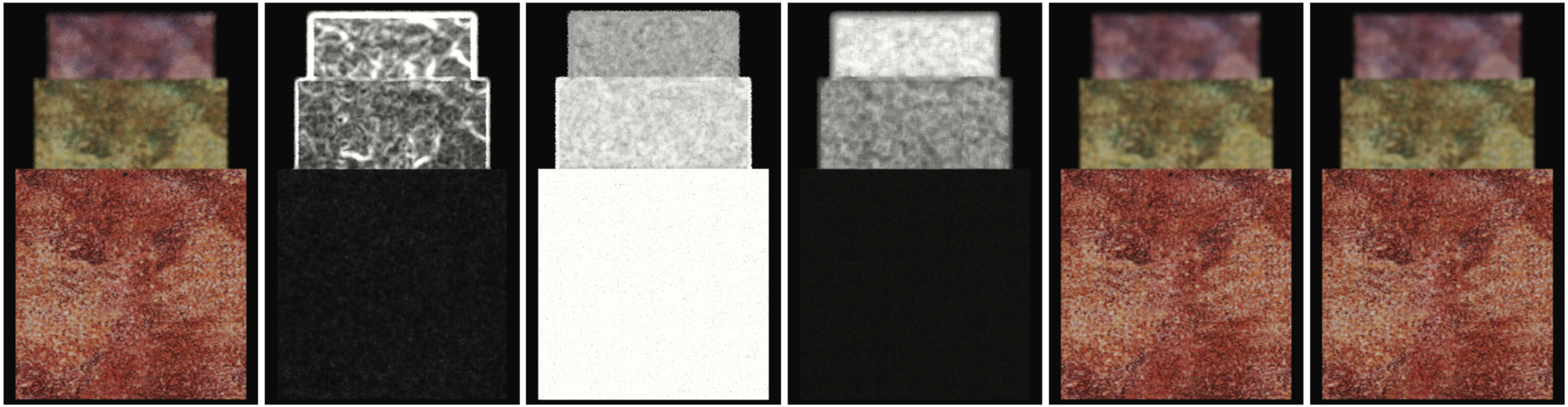
Bilateral Filtering of Features

$$w_{ij} = \exp[-\frac{1}{2\sigma_{\mathbf{p}}^2} \sum_{1 \le k \le 2} (\bar{\mathbf{p}}_{i,k} - \bar{\mathbf{p}}_{j,k})^2] \times$$

$$\exp[-\frac{1}{2\sigma_{\mathbf{c}}^2} \sum_{1 \le k \le 3} \alpha_k (\bar{\mathbf{c}}_{i,k} - \bar{\mathbf{c}}_{j,k})^2] \times$$

$$\exp[-\frac{1}{2\sigma_{\mathbf{f}}^2} \sum_{1 \le k \le m} \beta_k (\bar{\mathbf{f}}_{i,k} - \bar{\mathbf{f}}_{j,k})^2],$$

Dependency on Random Parameters



Input Monte Carlo (8 spp) | Dependency of color on random parameters ($D_{\mathbf{c}}^{\mathbf{r}}$) | Dependency of color on screen position ($D_{\mathbf{c}}^{\mathbf{p}}$) | Fractional dependency on random parameters ($W_{\mathbf{c}}^{\mathbf{r}}$) | Our approach (RPF) | Reference MC (512 spp)

Bilateral Weights

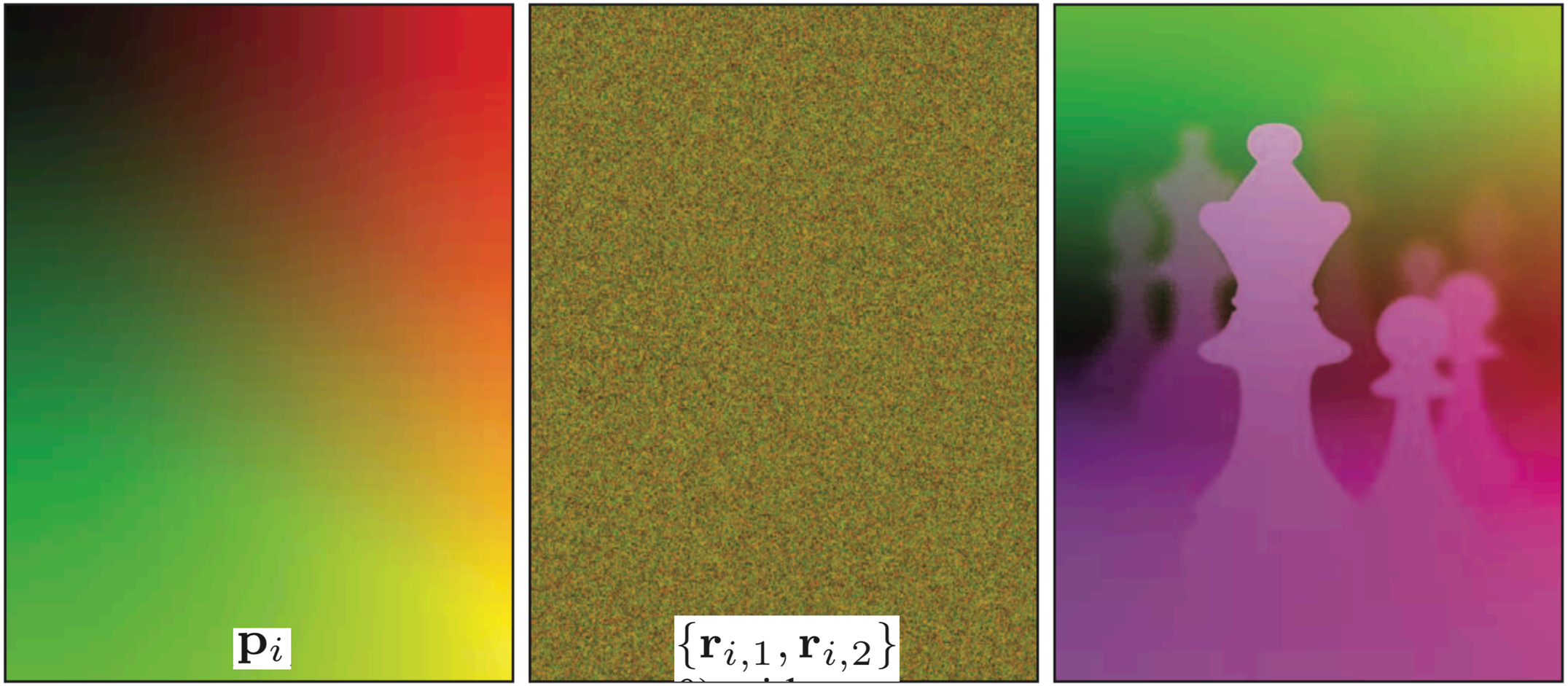
$$W^{\mathbf{r}}_{\mathbf{f},k} = \frac{D^{\mathbf{r}}_{\mathbf{f},k}}{D^{\mathbf{r}}_{\mathbf{f},k} + D^{\mathbf{p}}_{\mathbf{f},k}}$$

$$\beta_k = 1 - W^{\mathbf{r}}_{\mathbf{f},k}$$

$$W^{\mathbf{r}}_{\mathbf{c},k} = \frac{D^{\mathbf{r}}_{\mathbf{c},k}}{D^{\mathbf{r}}_{\mathbf{c},k} + D^{\mathbf{p}}_{\mathbf{c},k}}$$

$$\alpha_k = 1 - W^{\mathbf{r}}_{\mathbf{c},k}.$$

Pixels,Random Params,Features



(a) Screen position **(b)** Random parameters **(c)** World space coords.

Pixels,Random Params,Features

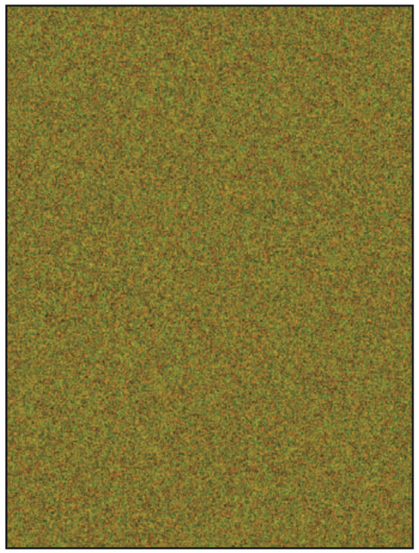
(d) Surface normals

(e) Texture value

(f) Sample color

Pixels,Random Params,Features

(a) Screen position



(b) Random parameters

(c) World space coords.

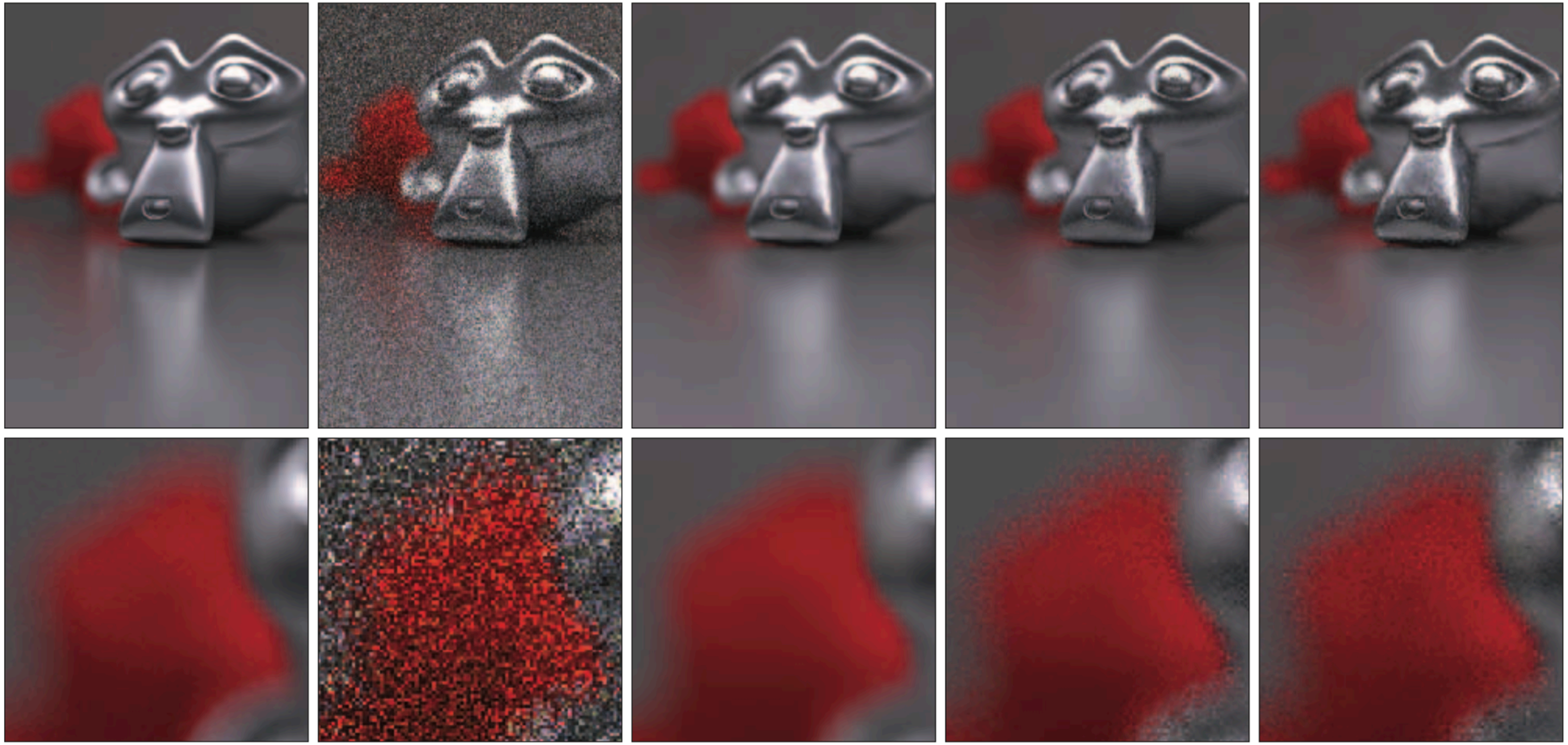
(d) Surface normals

(e) Texture value

(f) Sample color

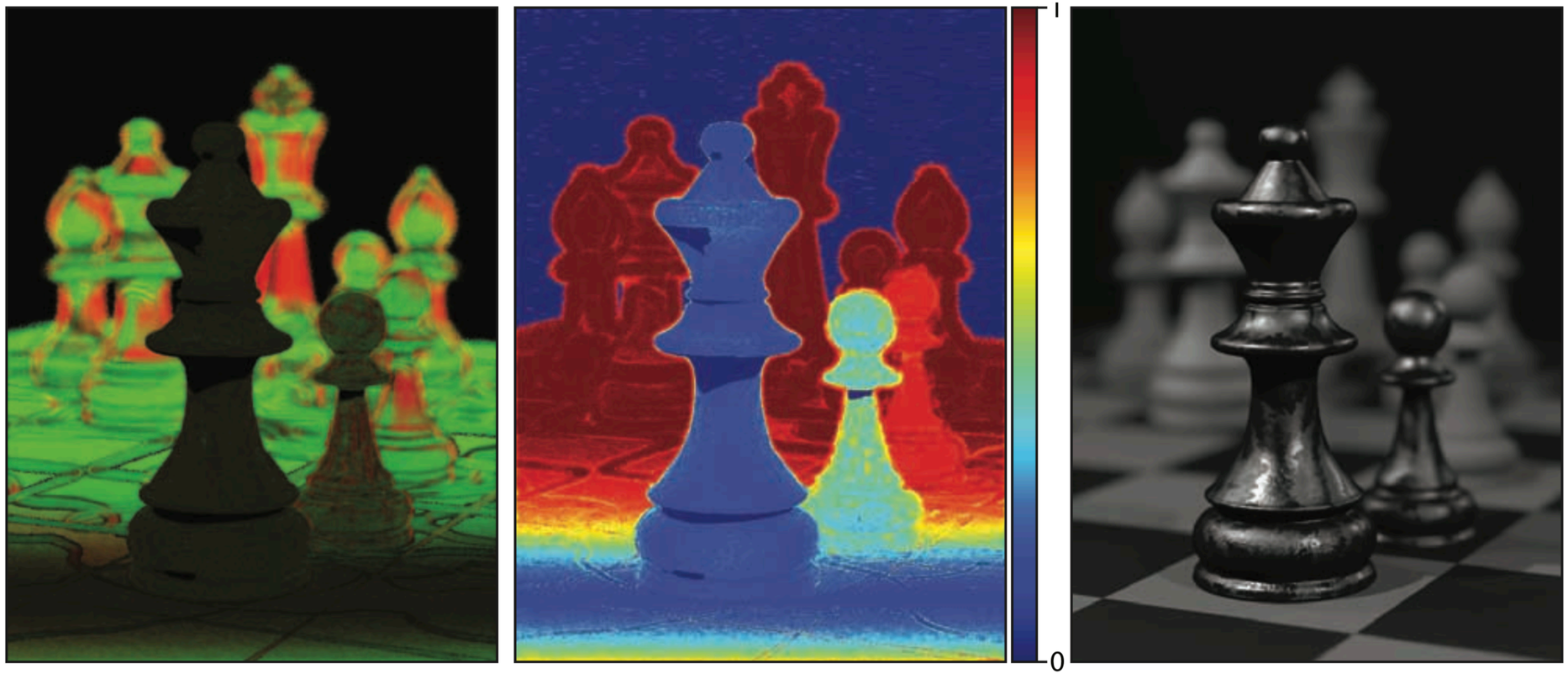
The algorithm computes the statistical dependency of **(c-f)** on the random parameters in (**b**)

Random Parameter Filtering



(a) Reference **(b)** MC Input **(c)** RPF **(d)** no clustering **(e)** no DoF params

Random Parameter Filtering



(a) $W^{\mathbf{r},1}_{\mathbf{c},k}$ and $W^{\mathbf{r},2}_{\mathbf{c},k}$ **(b)** $W^{\mathbf{r}}_{\mathbf{c},k}$ **(c)** Our output (RPF)

Statistical Dependency

Mutual information between two random variables:

$$\mu(X;Y) = \sum_{x \in X} \sum_{y \in Y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)}$$

where, these probabilities are computed over
the neighborhood of samples around a given pixel

Statistical Dependency

Functional dependency of the k-th scene parameter:

$$D^{\mathbf{r}}_{\mathbf{f},k} = \sum_{1 \le l \le n} D^{\mathbf{r},l}_{\mathbf{f},k} = \sum_{1 \le l \le n} \mu(\bar{\mathbf{f}}_{\mathcal{N},k}; \bar{\mathbf{r}}_{\mathcal{N},l})$$

$$D^{\mathbf{p}}_{\mathbf{f},k} = \sum_{1 \le l \le 2} D^{\mathbf{p},l}_{\mathbf{f},k} = \sum_{1 \le l \le 2} \mu(\bar{\mathbf{f}}_{\mathcal{N},k}; \bar{\mathbf{p}}_{\mathcal{N},l}),$$

$$D^{\mathbf{r}}_{\mathbf{c},k} = \sum_{1 \le l \le n} D^{\mathbf{r},l}_{\mathbf{c},k} = \sum_{1 \le l \le n} \mu(\bar{\mathbf{c}}_{\mathcal{N},k}; \bar{\mathbf{r}}_{\mathcal{N},l}),$$

$$D^{\mathbf{p}}_{\mathbf{c},k} = \sum_{1 \le l \le 2} D^{\mathbf{p},l}_{\mathbf{c},k} = \sum_{1 \le l \le 2} \mu(\bar{\mathbf{c}}_{\mathcal{N},k}; \bar{\mathbf{p}}_{\mathcal{N},l}).$$

Statistical Dependency

$$D^{\mathbf{r}}_{\mathbf{f},k} = \sum_{1 \le l \le n} D^{\mathbf{r},l}_{\mathbf{f},k} = \sum_{1 \le l \le n} \mu(\bar{\mathbf{f}}_{\mathcal{N},k}; \bar{\mathbf{r}}_{\mathcal{N},l})$$

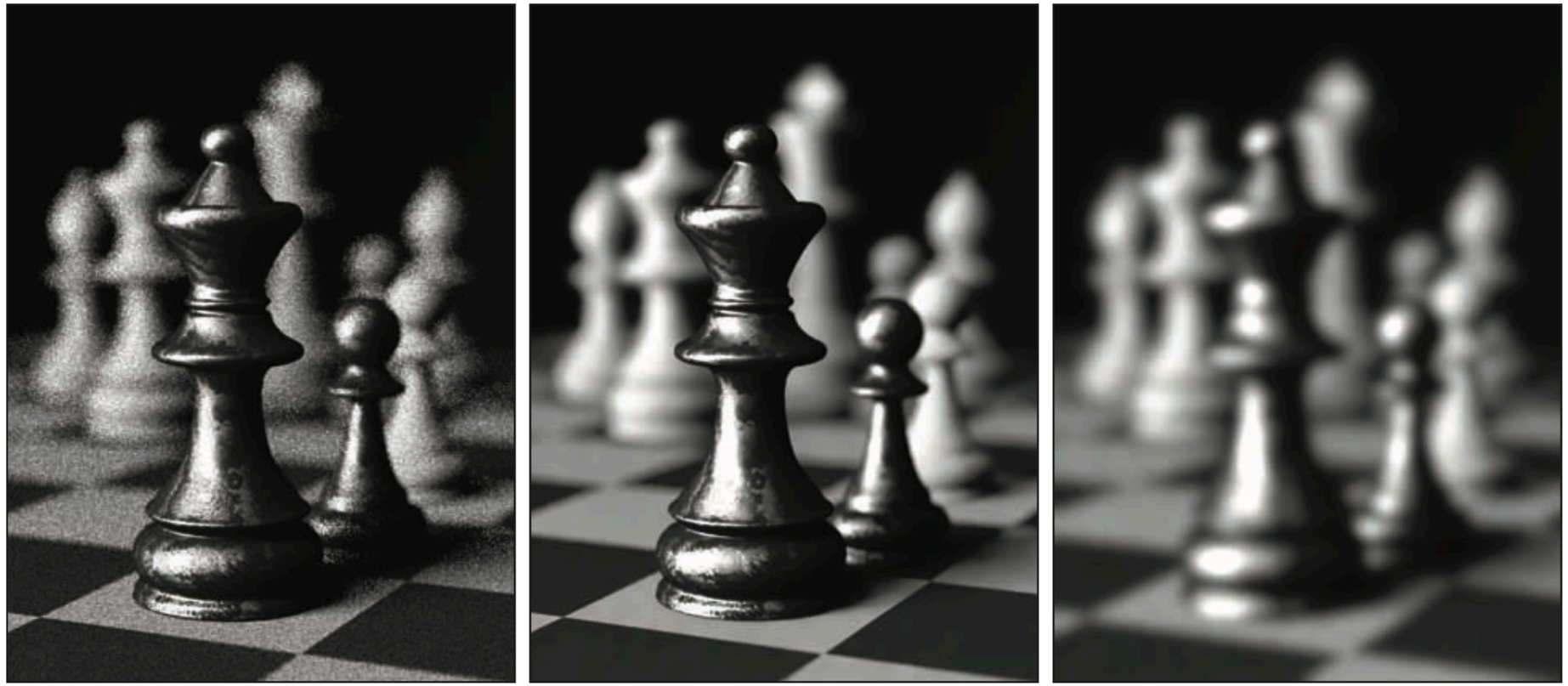
$$W^{\mathbf{f},k}_{\mathbf{c}} = \frac{D^{\mathbf{f},k}_{\mathbf{c}}}{D^{\mathbf{r}}_{\mathbf{c}} + D^{\mathbf{p}}_{\mathbf{c}} + D^{\mathbf{f}}_{\mathbf{c}}}$$

$$D^{\mathbf{r}}_{\mathbf{c}} = \sum_{1 \le k \le 3} D^{\mathbf{r}}_{\mathbf{c},k}, \qquad D^{\mathbf{p}}_{\mathbf{c}} = \sum_{1 \le k \le 3} D^{\mathbf{p}}_{\mathbf{c},k}, \qquad D^{\mathbf{f}}_{\mathbf{c}} = \sum_{1 \le k \le 3} D^{\mathbf{f}}_{\mathbf{c},k}$$

Weighted Average Bilateral Filtering

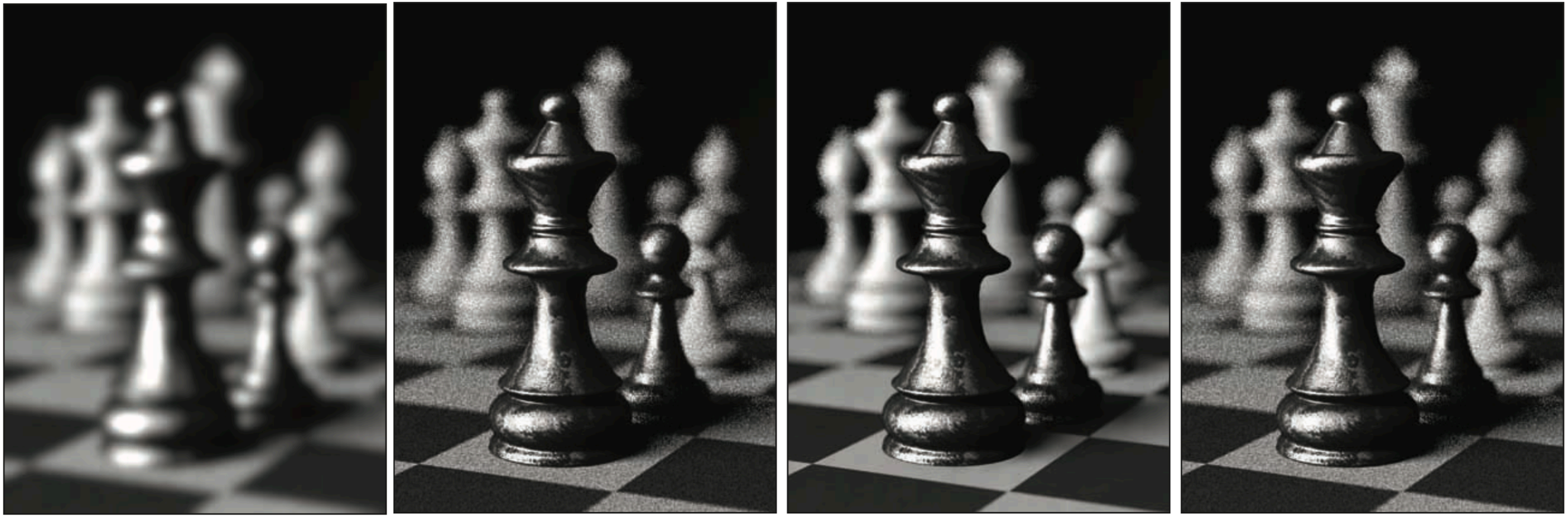
$$\mathbf{c}'_{i,k} = \frac{\sum_{j \in \mathcal{N}} w_{ij} \mathbf{c}_{j,k}}{\sum_{j \in \mathcal{N}} w_{ij}}$$

Results



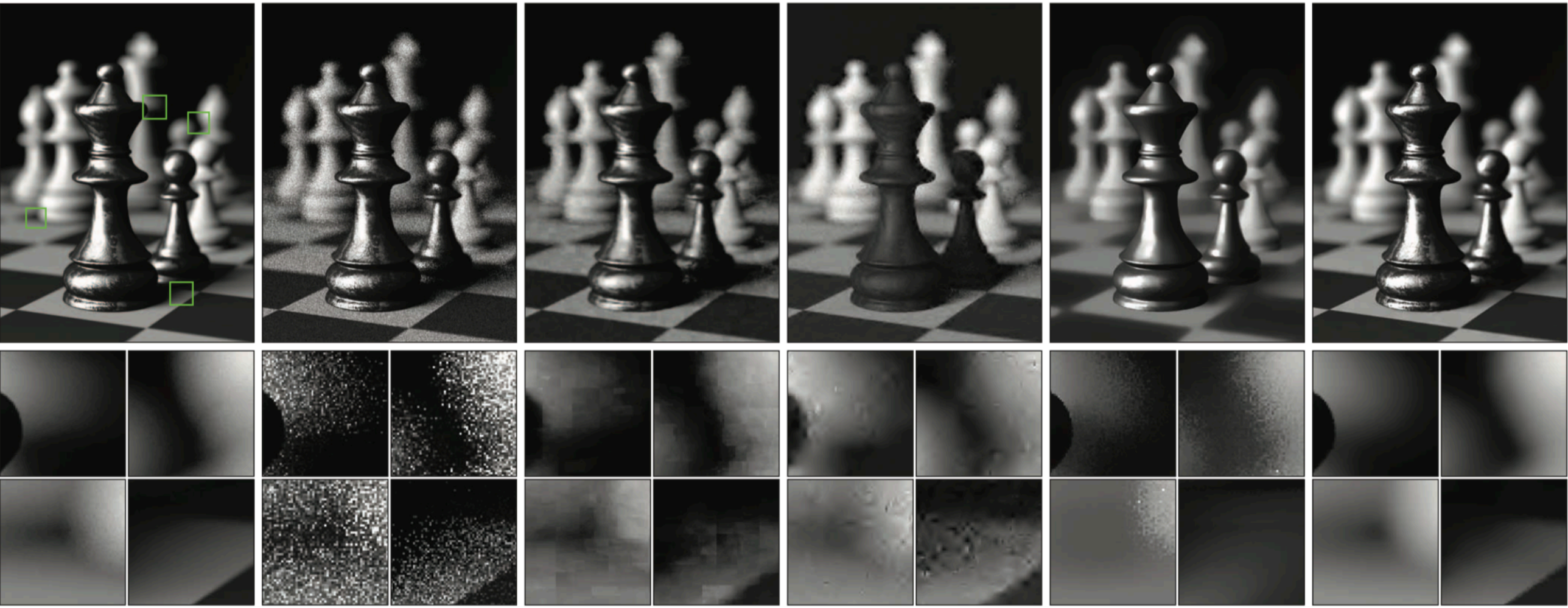
(a) MC Input (8 spp) **(b)** Our approach (RPF) **(c)** $\alpha_k = 0, \beta_k = 0$

Results



) **(c)** $\alpha_k = 0, \beta_k = 0$ **(d)** $\alpha_k = 1, \beta_k = 0$ **(e)** $\alpha_k = 0, \beta_k = 1$ **(f)** $\alpha_k = 1, \beta_k = 1$

Results



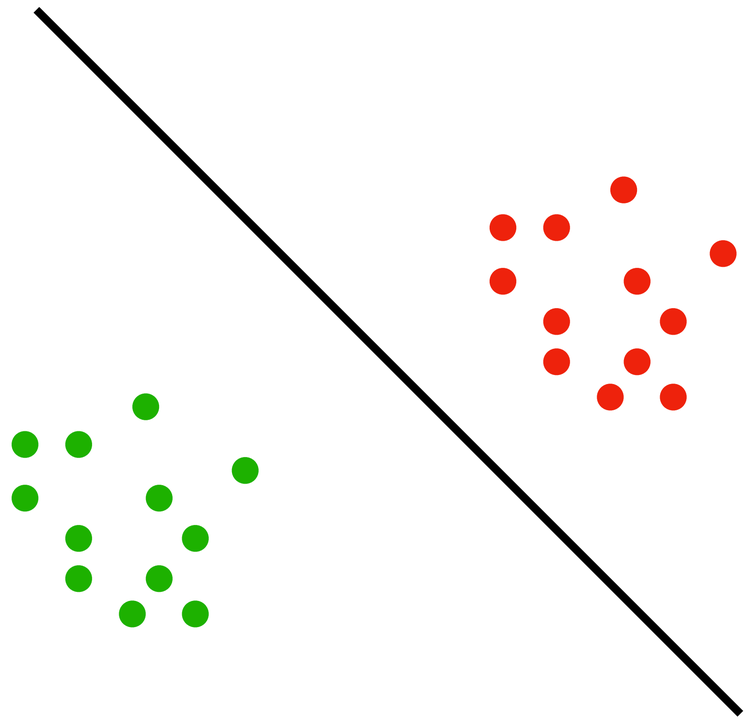
Reference (8,192 spp) Input Monte Carlo (8 spp) MDAS AWR À-Trous Our approach (RPF)

Multi-Layer Perceptrons

History of Neural Networks

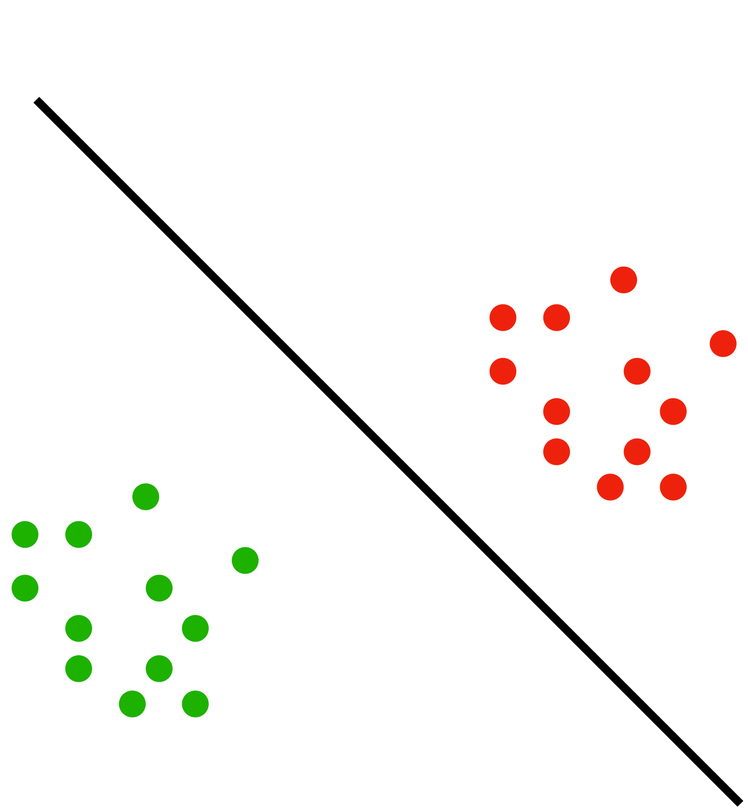
- In 1943, McCulloch and Pitts created a computational model for neural networks
- In 1975, Werbos's back propagation algorithm generally accelerated the training of multi-layer networks.
- In 1980s, Recurrent Neural Networks were developed

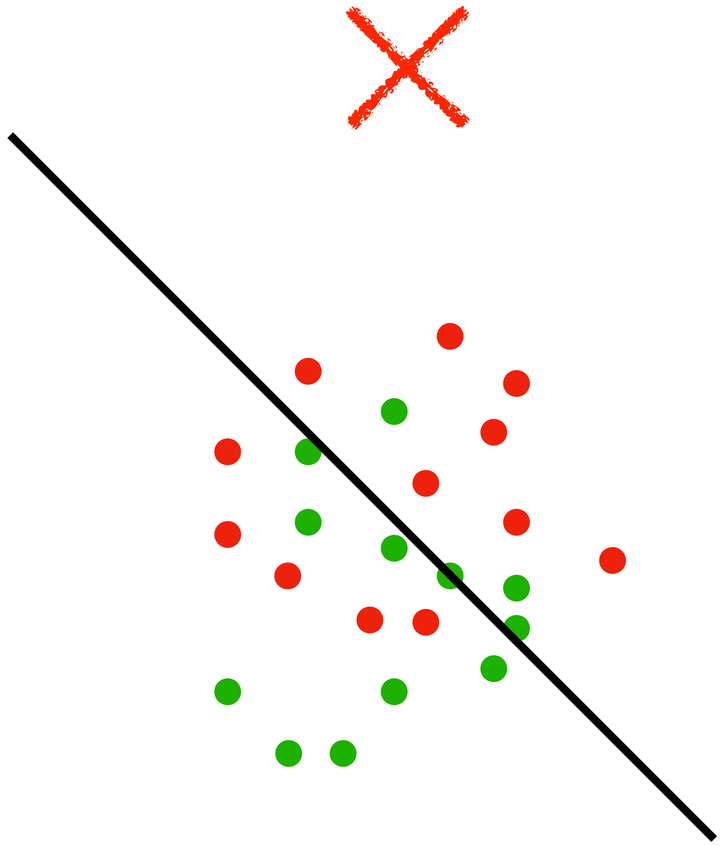
Classifiers



$$y_j = f(w_j x_j + b_j)$$

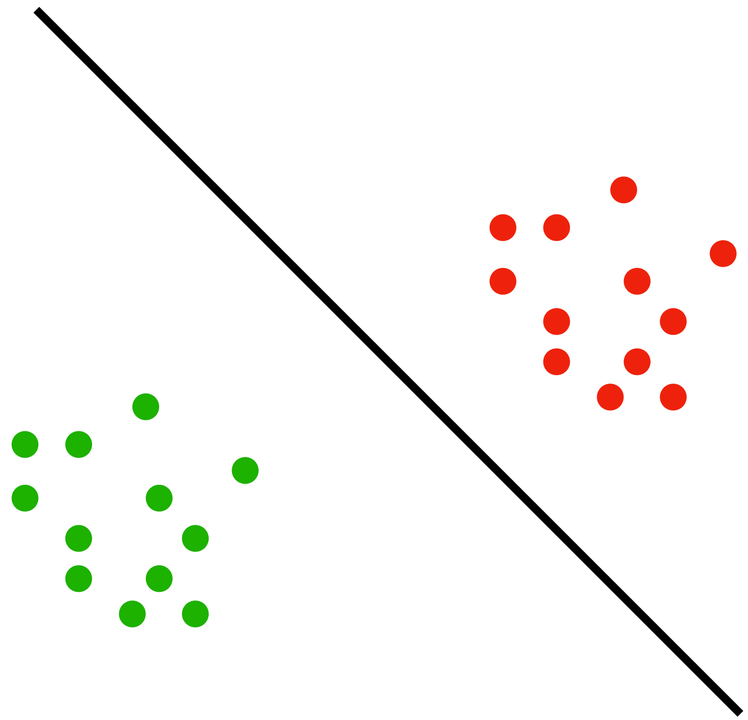
Classifiers



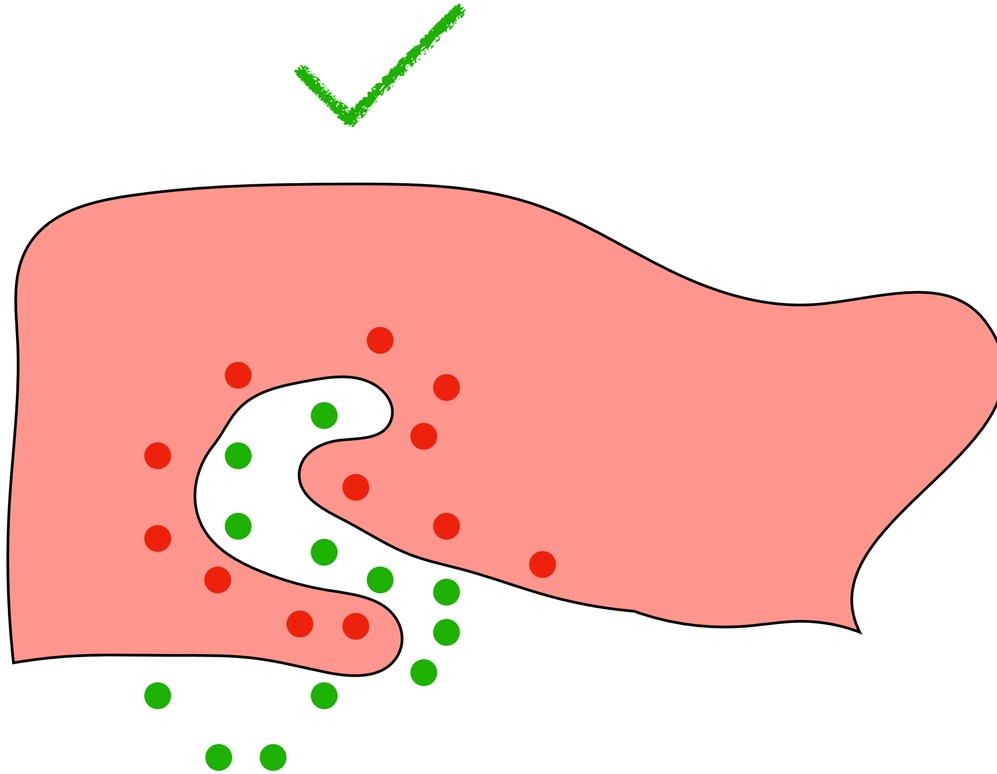


$$y_j = f(w_j x_j + b_j)$$

Complex Classifiers



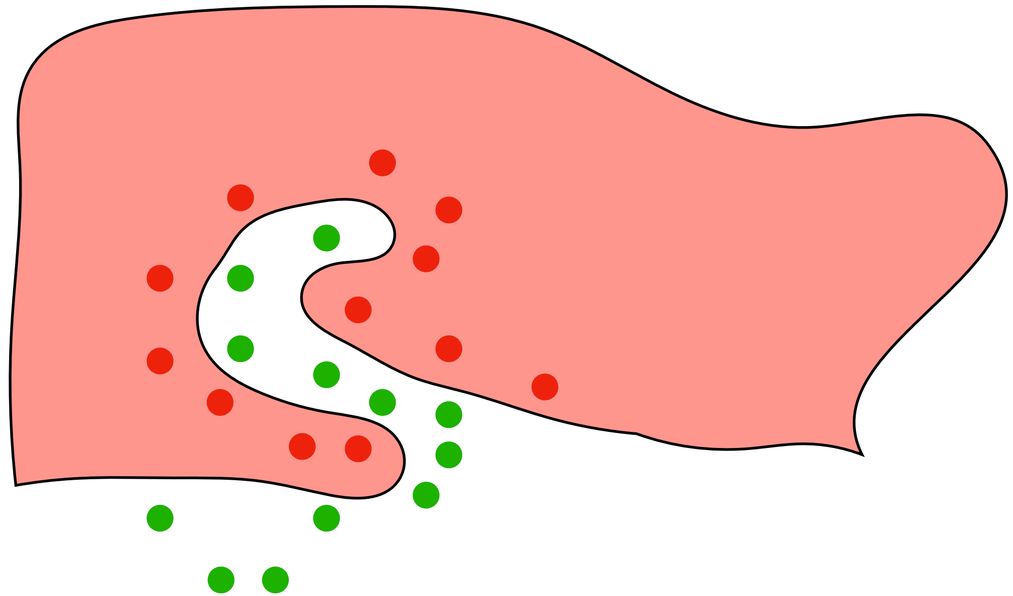
$$y_j = f(w_j x_j + b_j)$$



Complex classifier

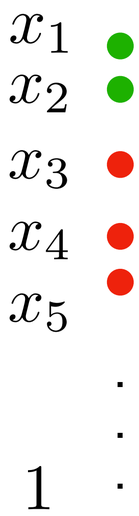
Complex Classifiers

Complex classifier

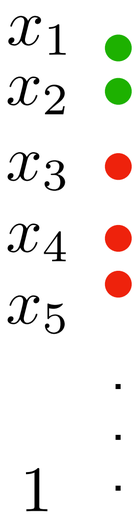


What features can produce this decision rule ?

Perceptron Classifier

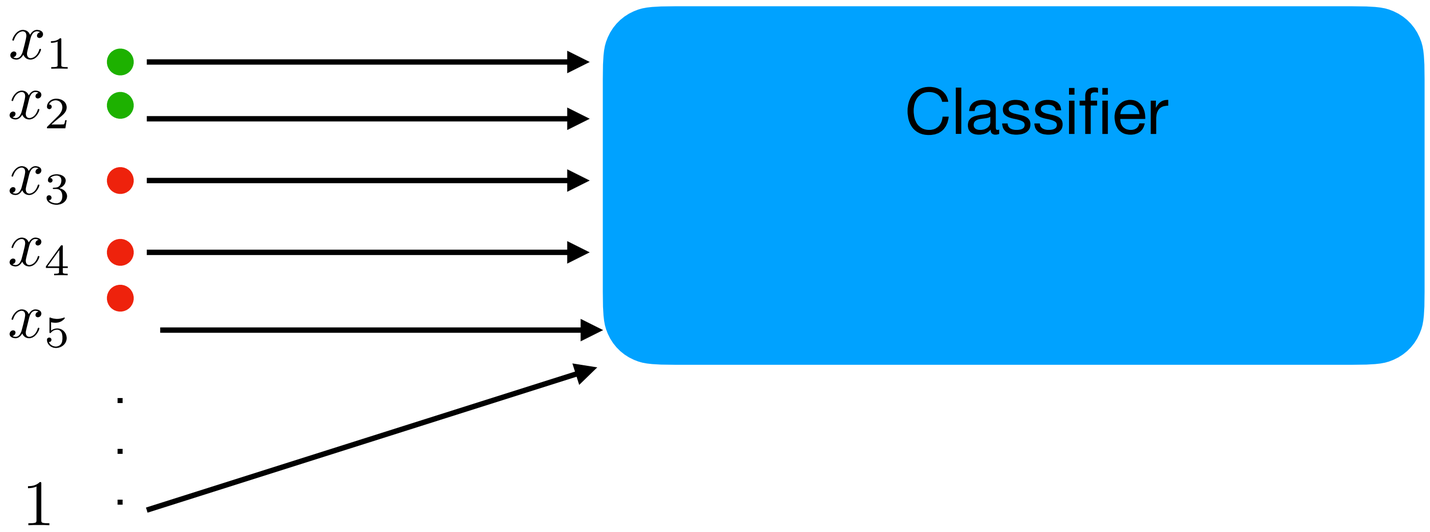


Perceptron Classifier

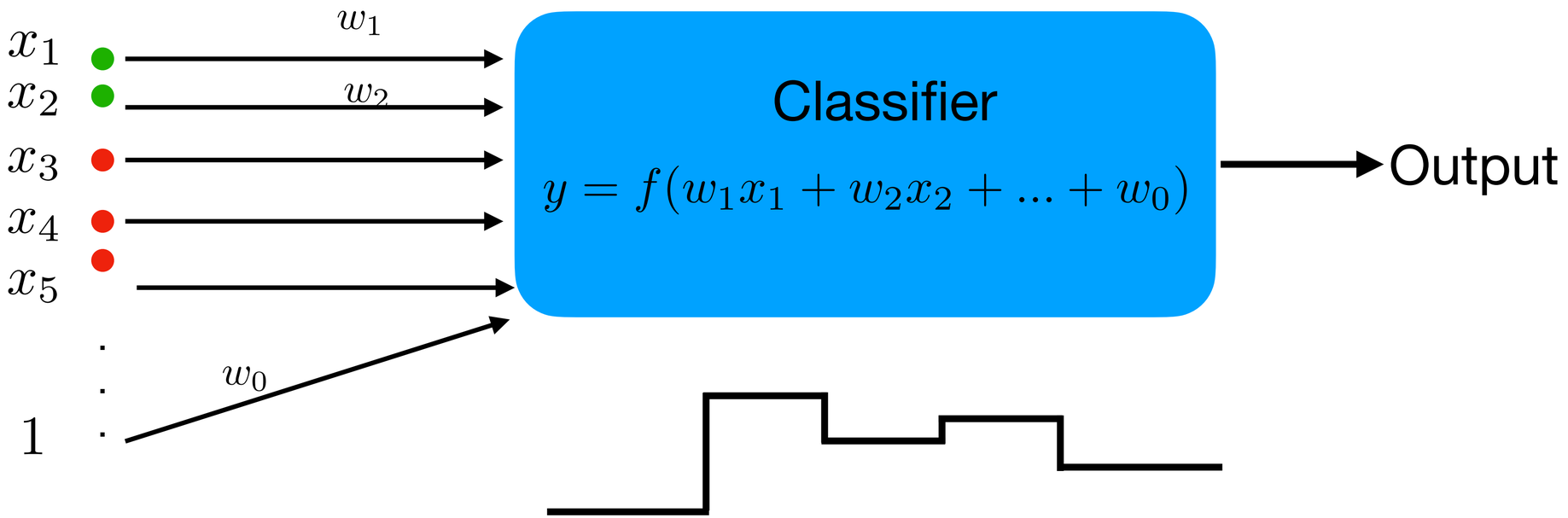


Classifier

Perceptron Classifier



Perceptron Classifier

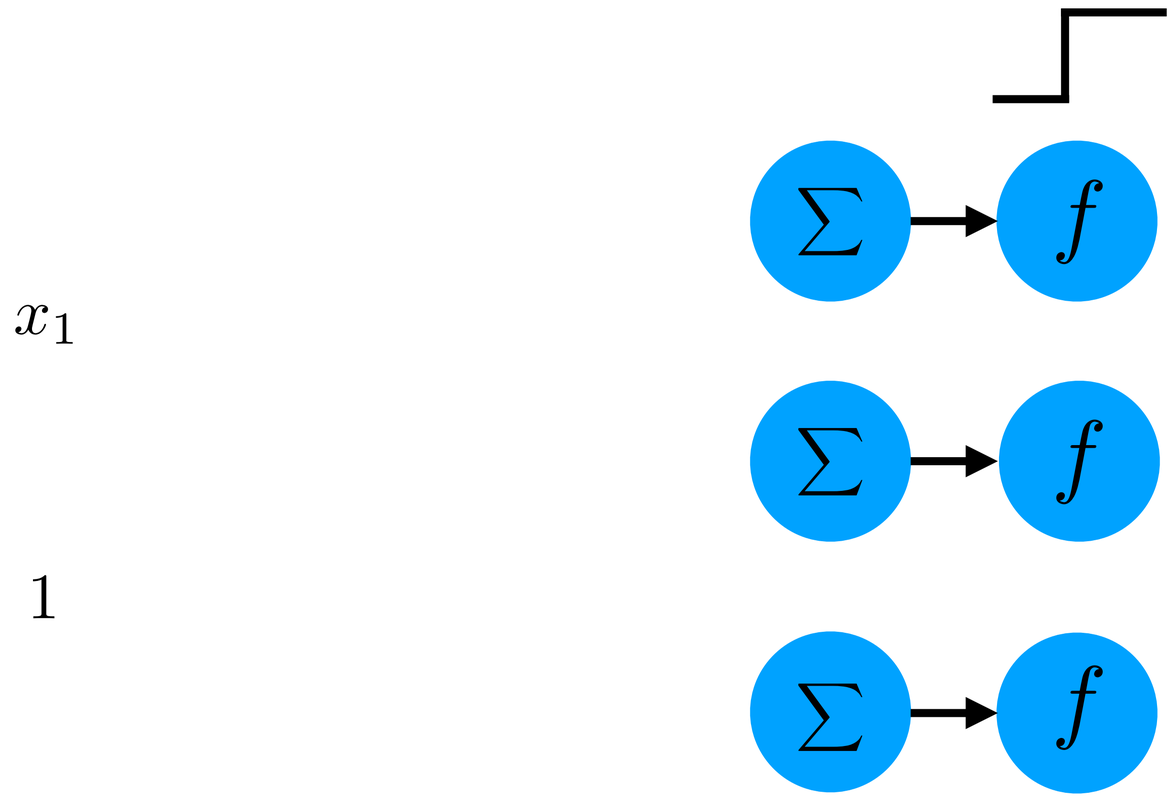


Multi-layer Perceptron

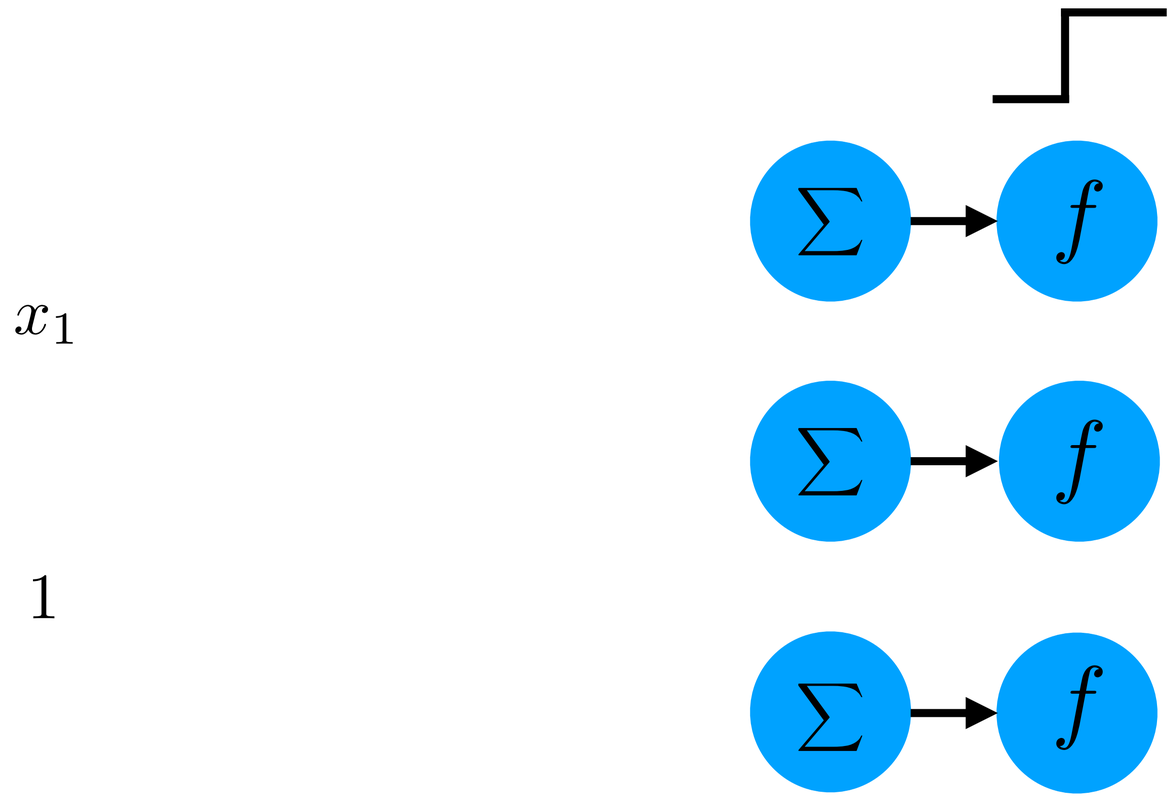
x_1

1

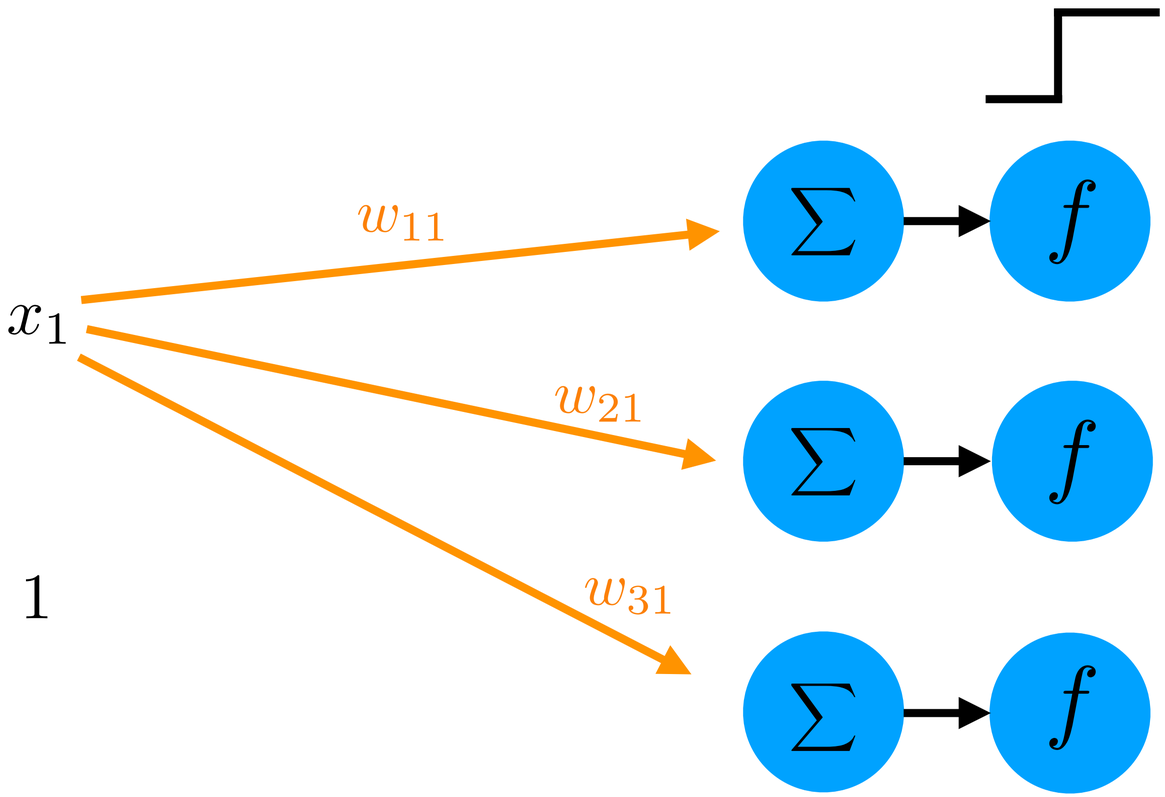
Multi-layer Perceptron



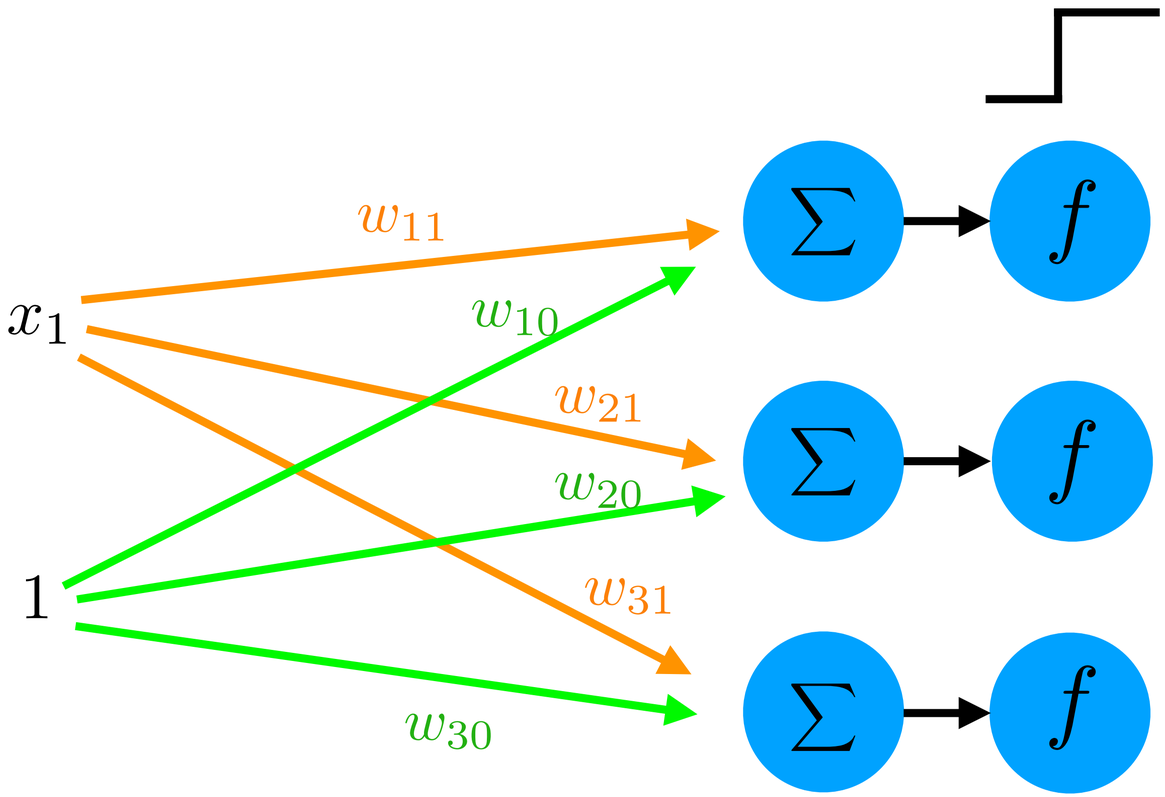
Multi-layer Perceptron



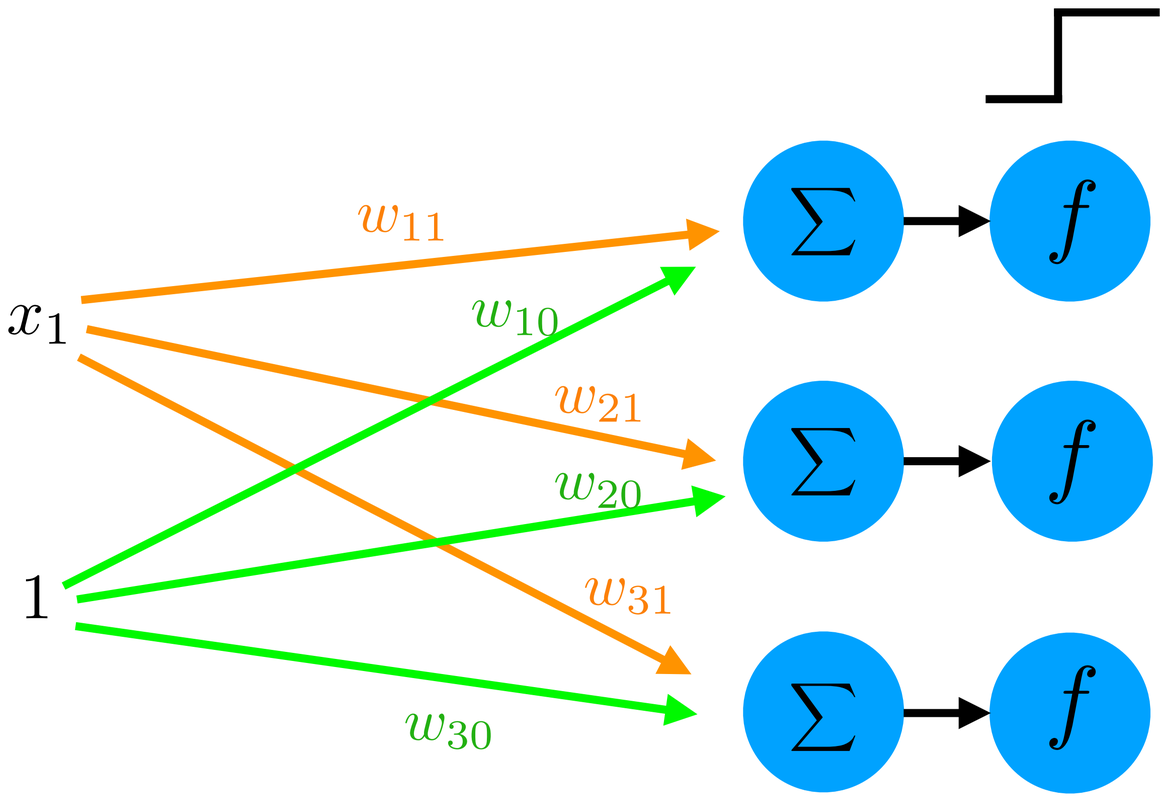
Multi-layer Perceptron



Multi-layer Perceptron

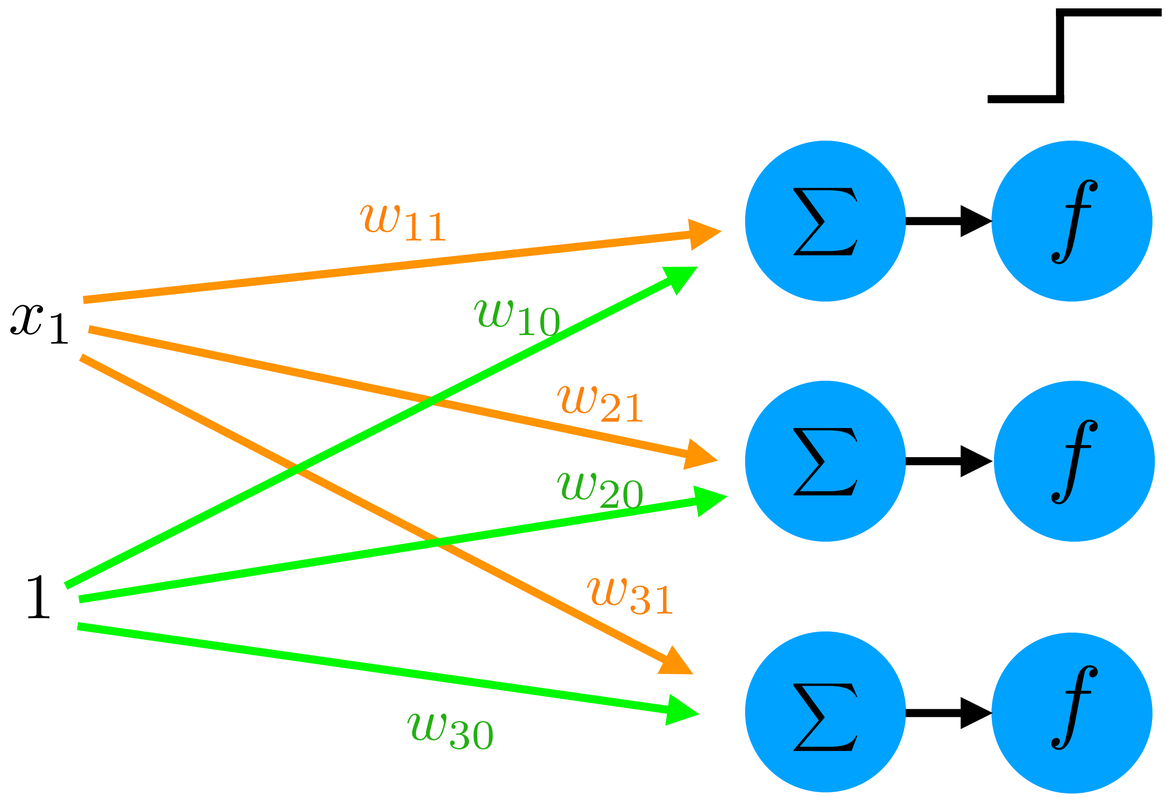


Multi-layer Perceptron



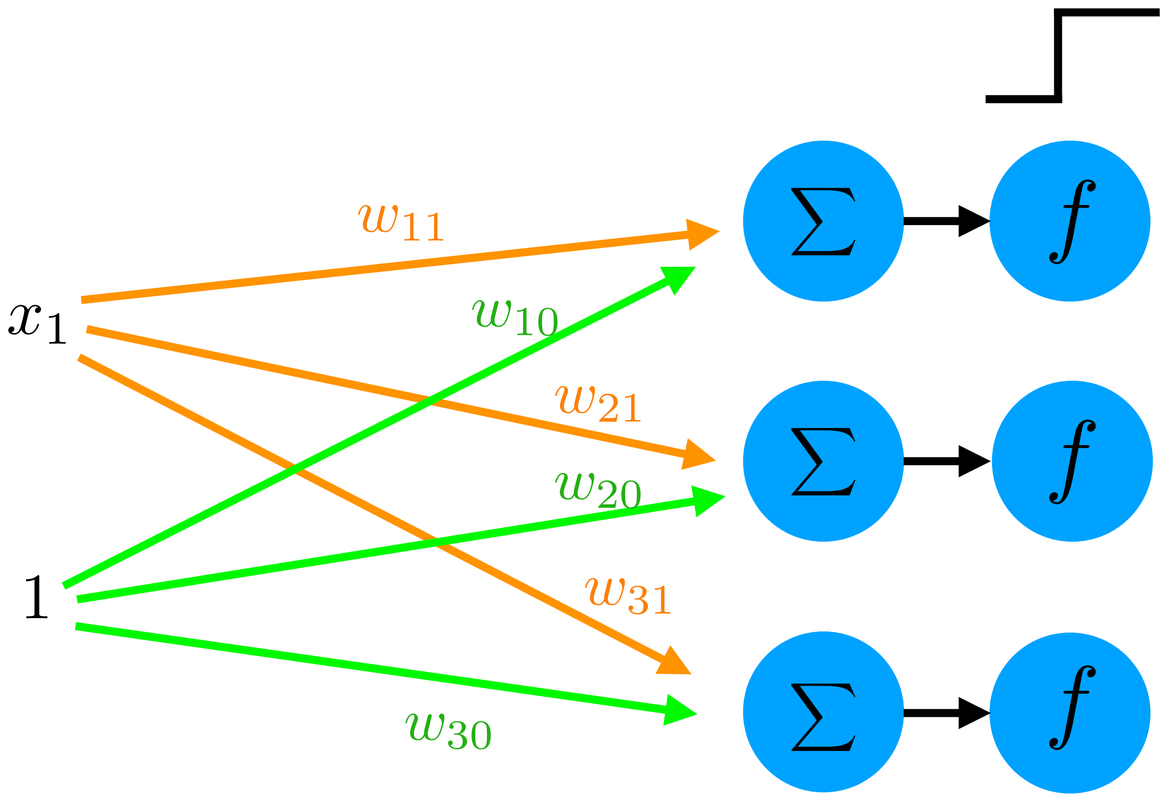
$$x_1\, w_{11} + w_{10}$$
$$x_1\, w_{21} + w_{20}$$
$$x_1\, w_{31} + w_{30}$$

Multi-layer Perceptron



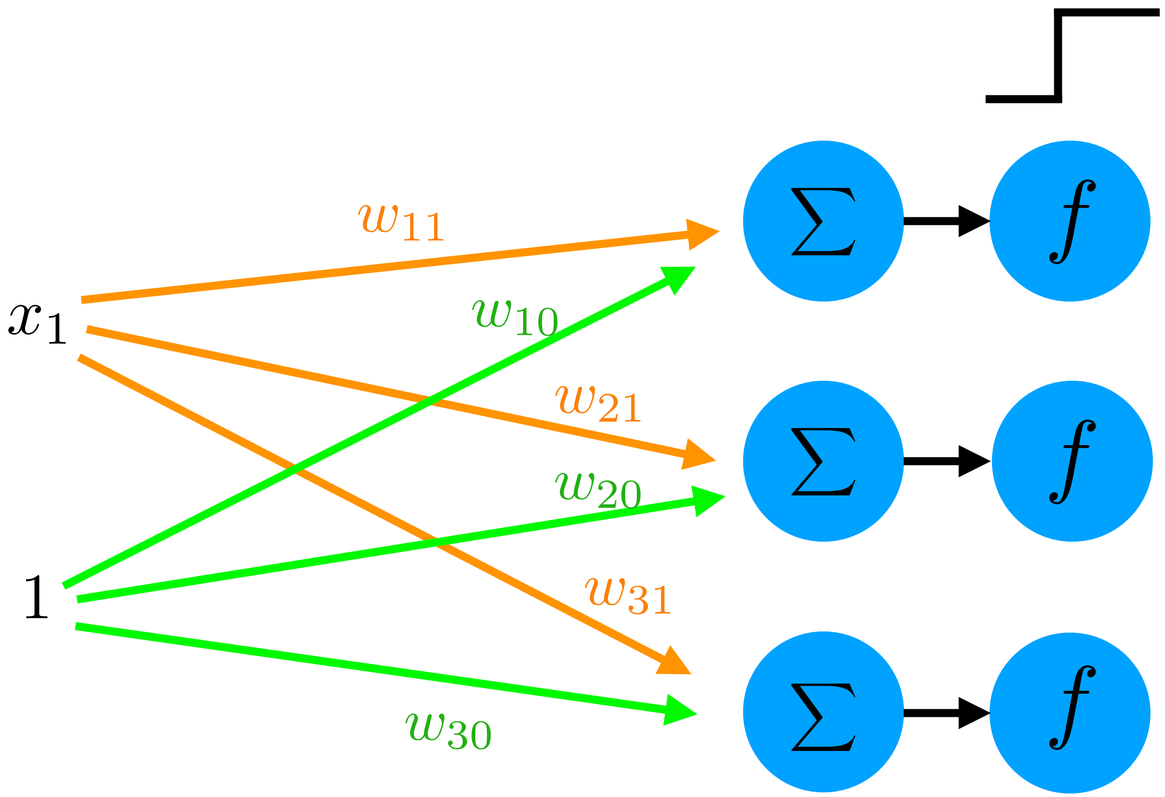
$$y_1 = f(x_1 w_{11} + w_{10})$$
$$y_2 = f(x_1 w_{21} + w_{20})$$
$$y_3 = f(x_1 w_{31} + w_{30})$$

Multi-layer Perceptron



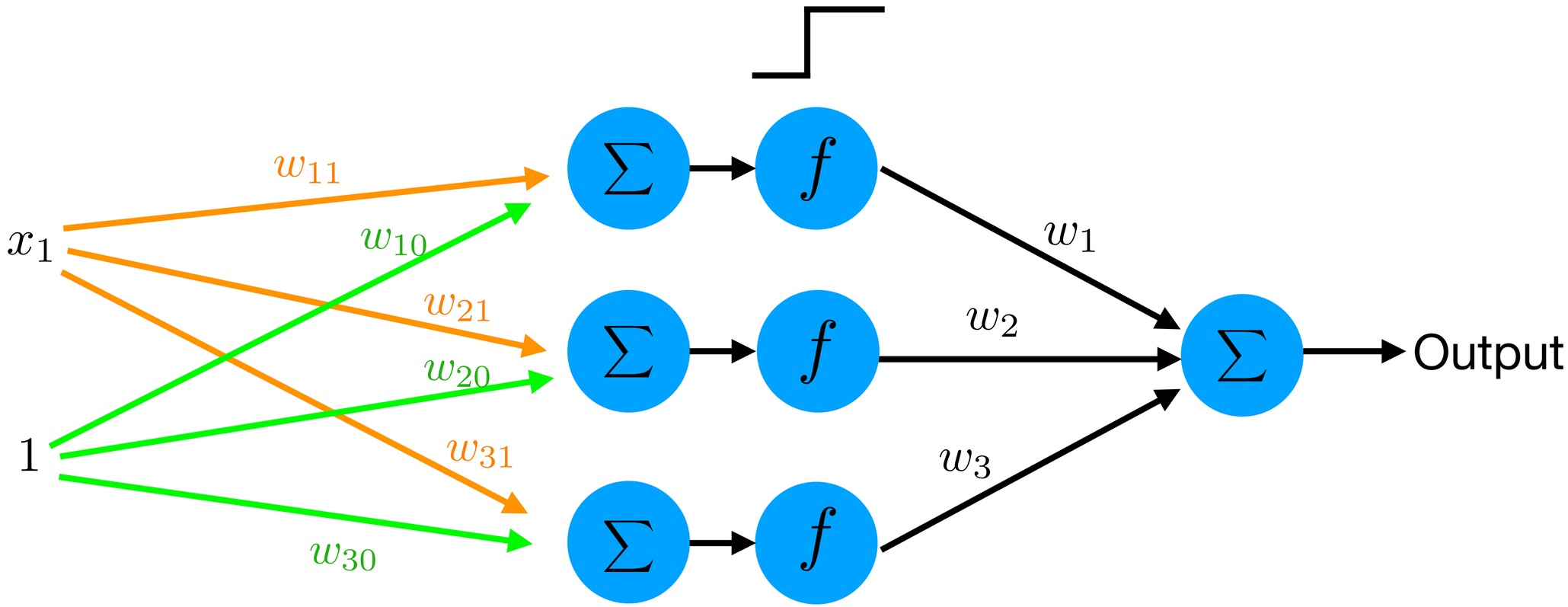
$$
\begin{aligned}
y_1 &= f(x_1 w_{11} + w_{10}) \\
y_2 &= f(x_1 w_{21} + w_{20}) \\
y_3 &= f(x_1 w_{31} + w_{30})
\end{aligned}
$$

Multi-layer Perceptron



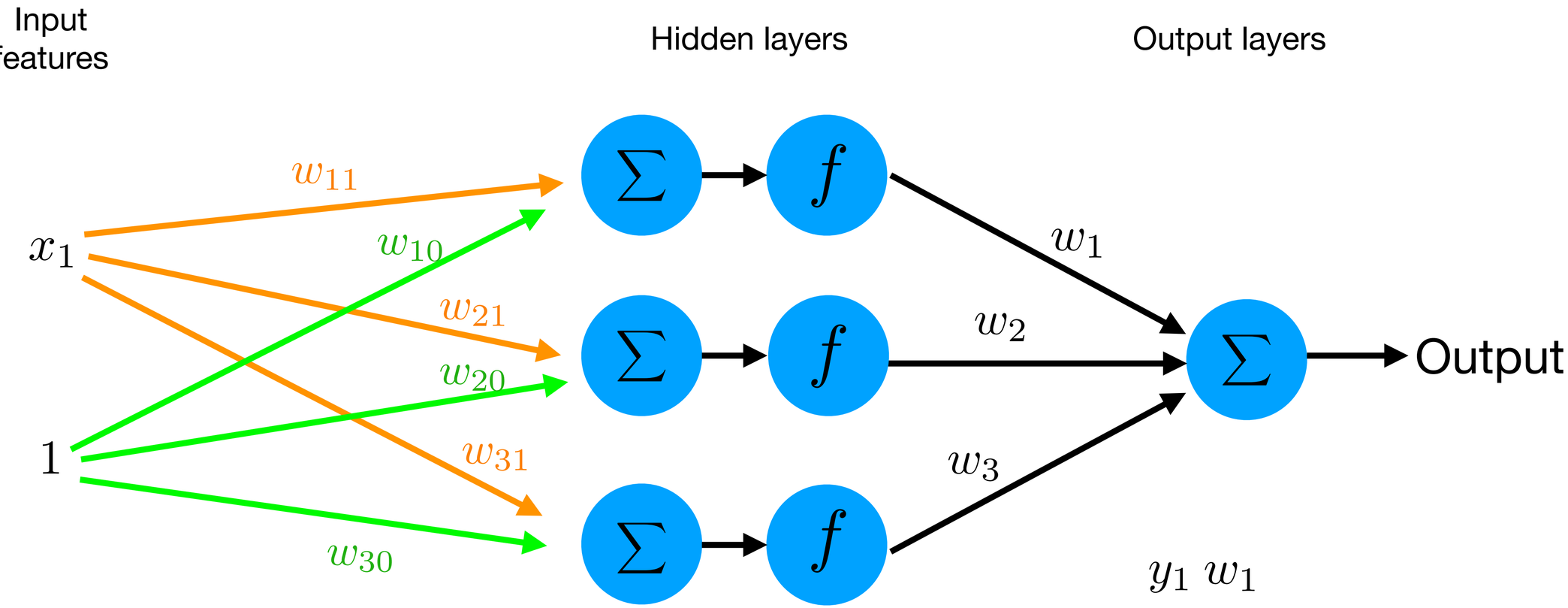
$$
\begin{aligned}
y_1 &= f(x_1 w_{11} + w_{10}) \\
y_2 &= f(x_1 w_{21} + w_{20}) \\
y_3 &= f(x_1 w_{31} + w_{30})
\end{aligned}
$$

Multi-layer Perceptron

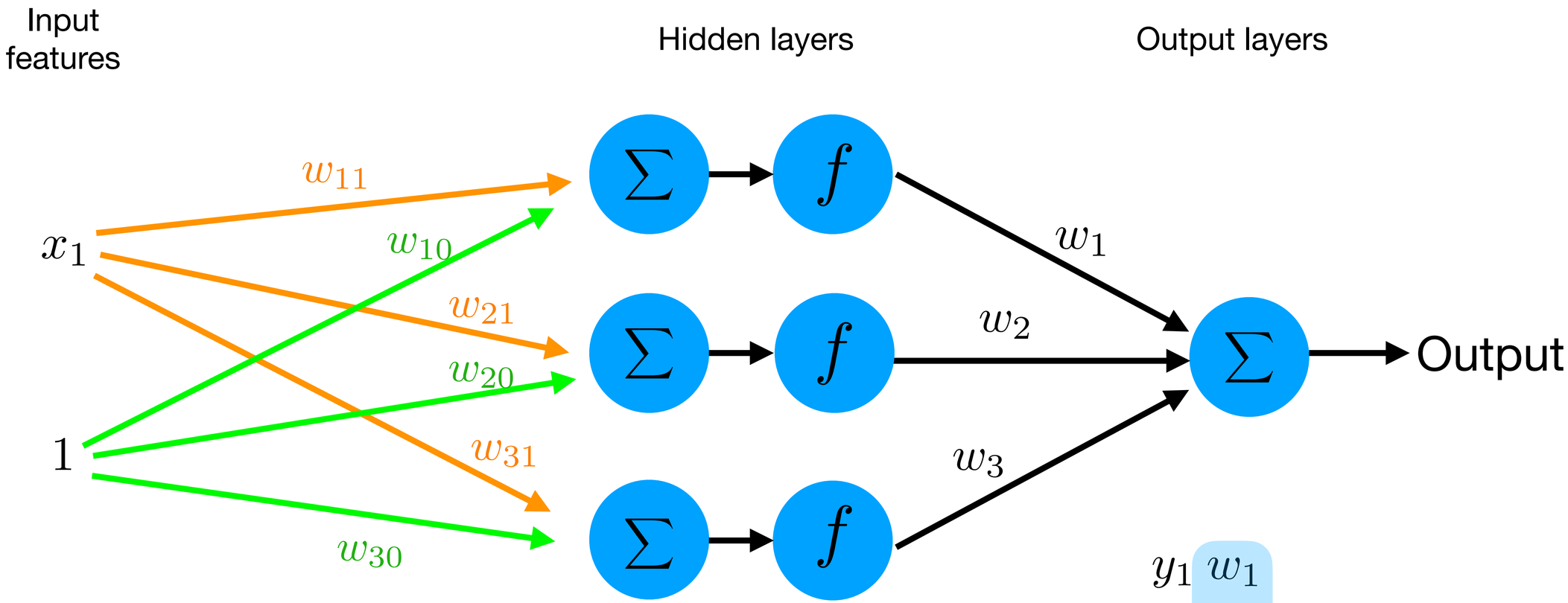


$$
\begin{aligned}
y_1 &= f(x_1 w_{11} + w_{10}) \\
y_2 &= f(x_1 w_{21} + w_{20}) \\
y_3 &= f(x_1 w_{31} + w_{30})
\end{aligned}
$$

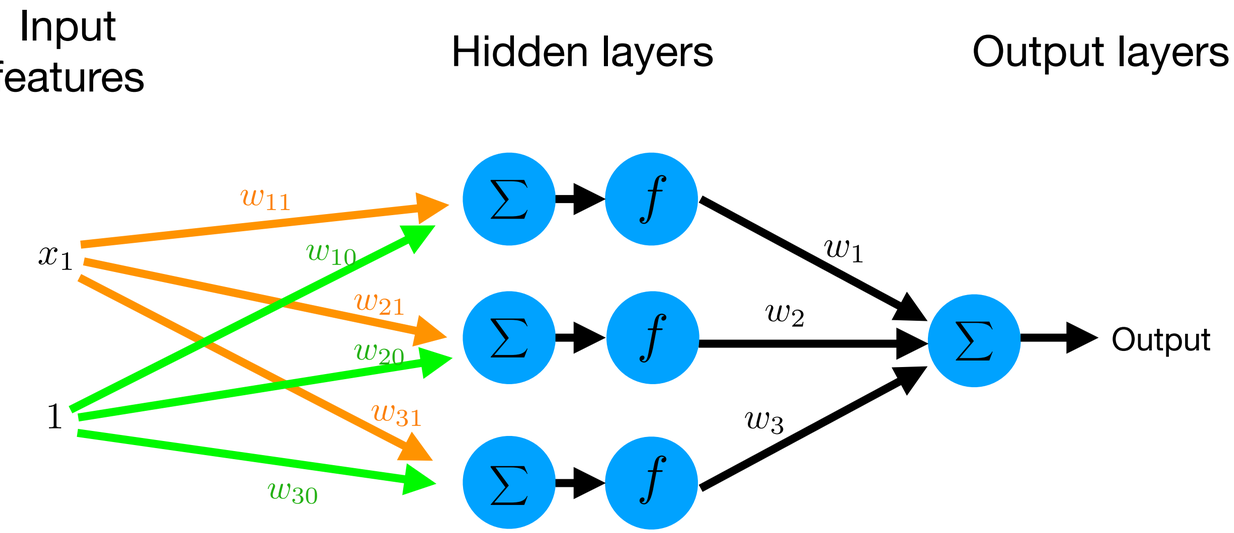
Multi-layer Perceptron



Multi-layer Perceptron



Multi-layer Perceptron



"Features" are outputs of perceptrons

Matrix of first layer weights

$$\begin{matrix} w_{11} & w_{10} \\ w_{21} & w_{20} \\ w_{31} & w_{30} \end{matrix}$$

Matrix of second layer weights

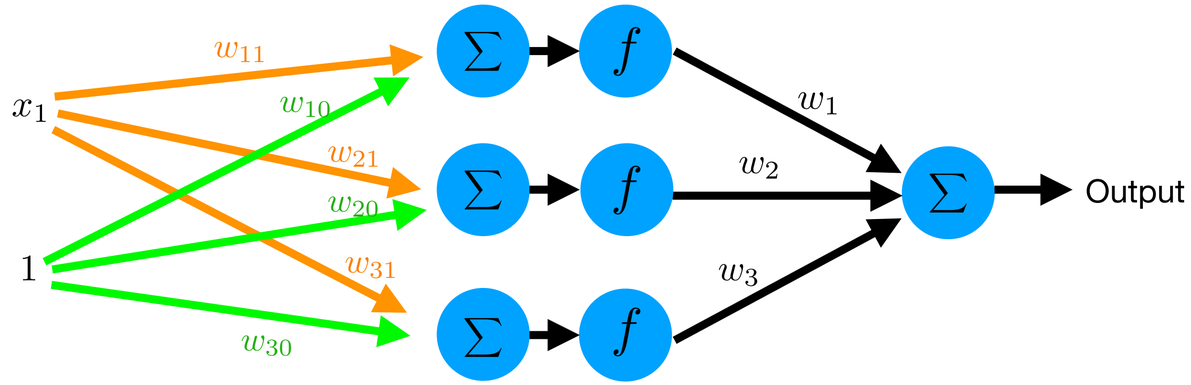
$$\begin{matrix} w_1 \\ w_2 \\ w_3 \end{matrix}$$

Multi-layer Perceptron

Input features

Hidden layers

Output layers



"Features" are outputs of perceptrons

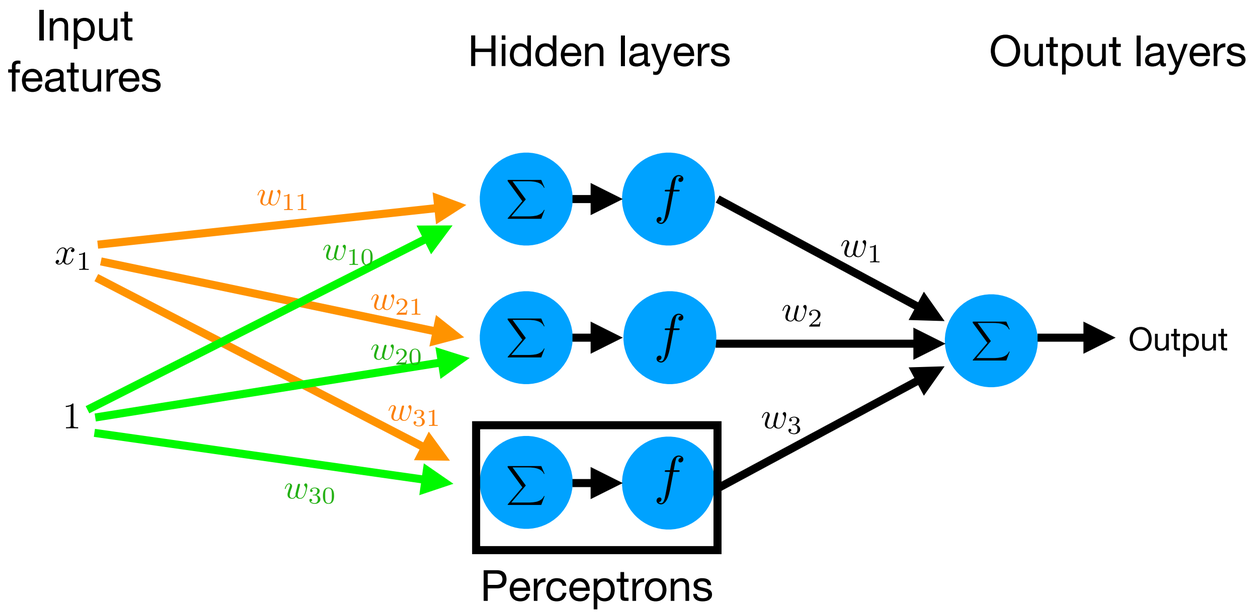
Matrix of first layer weights

$$\begin{bmatrix} w_{11} & w_{10} \\ w_{21} & w_{20} \\ w_{31} & w_{30} \end{bmatrix}$$

Matrix of second layer weights

$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

Multi-layer Perceptron



"Features" are outputs of perceptrons

Matrix of first layer weights

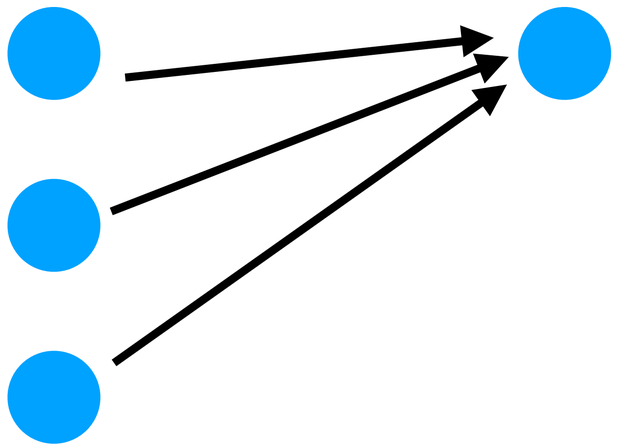
$$\begin{matrix} w_{11} & w_{10} \\ w_{21} & w_{20} \\ w_{31} & w_{30} \end{matrix}$$

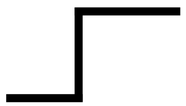
Matrix of second layer weights

$$\begin{matrix} w_1 \\ w_2 \\ w_3 \end{matrix}$$

Features of MLPs

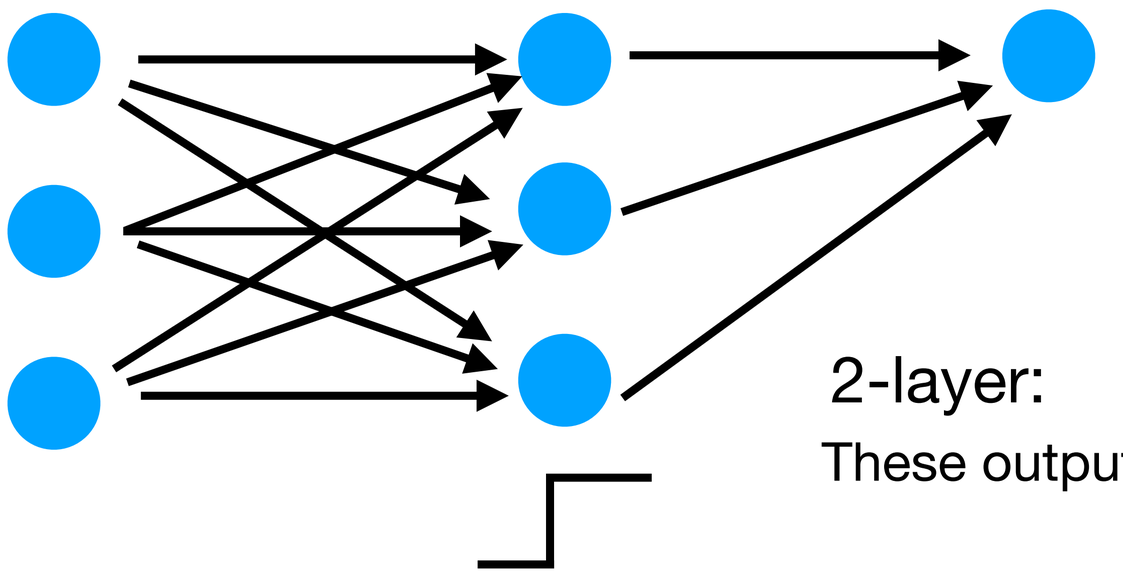
Input features





Perceptron: Step function with linear decision boundary

Features of MLPs

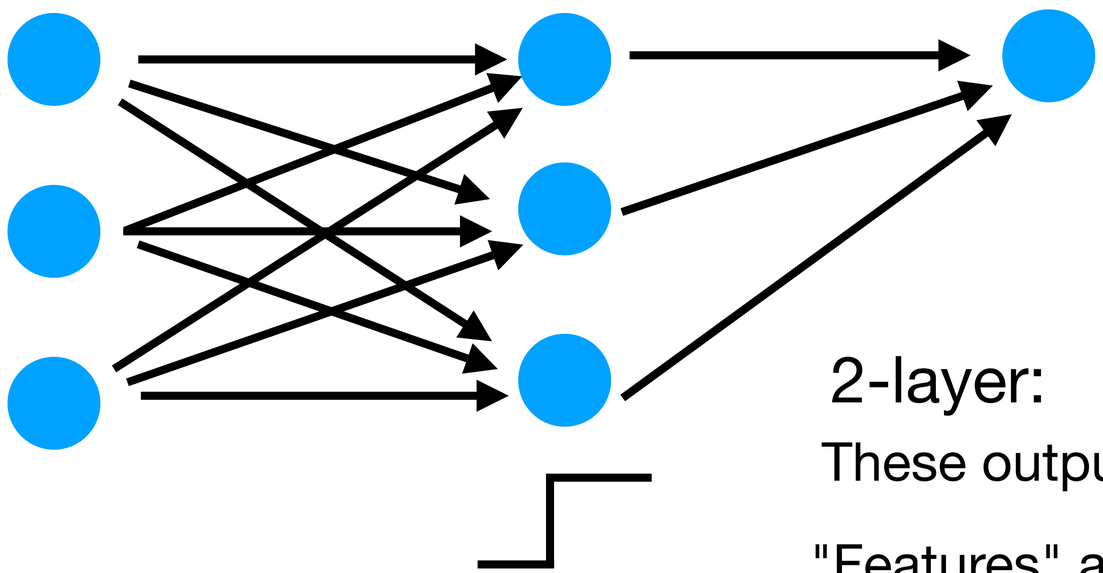


Layer 1

2-layer:

These outputs are now input features to the next layer

Features of MLPs



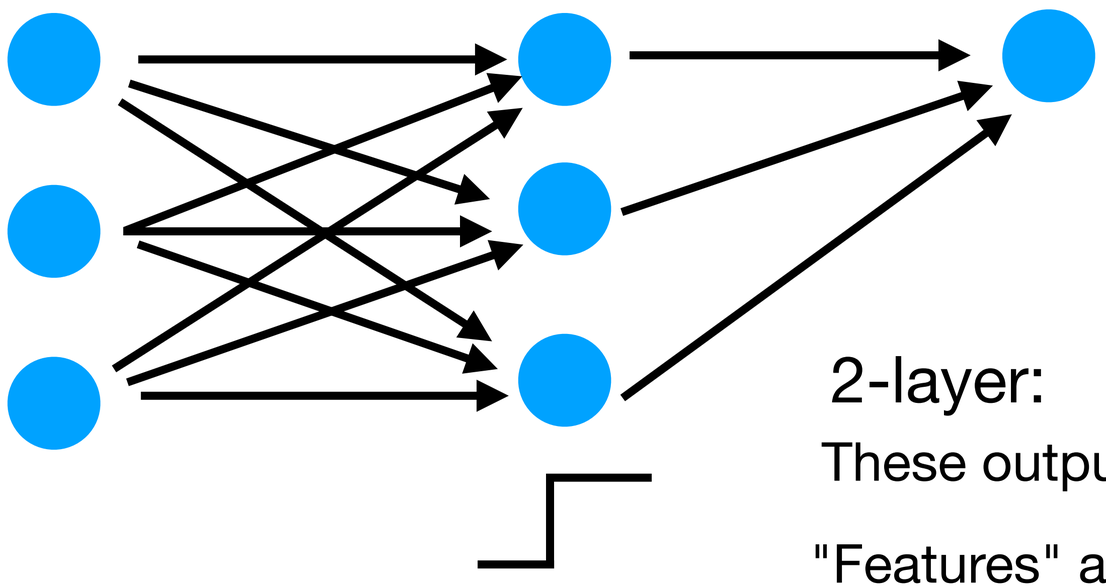
Layer 1

2-layer:

These outputs are now input features to the next layer

"Features" are now decision boundaries (partitions)

Features of MLPs



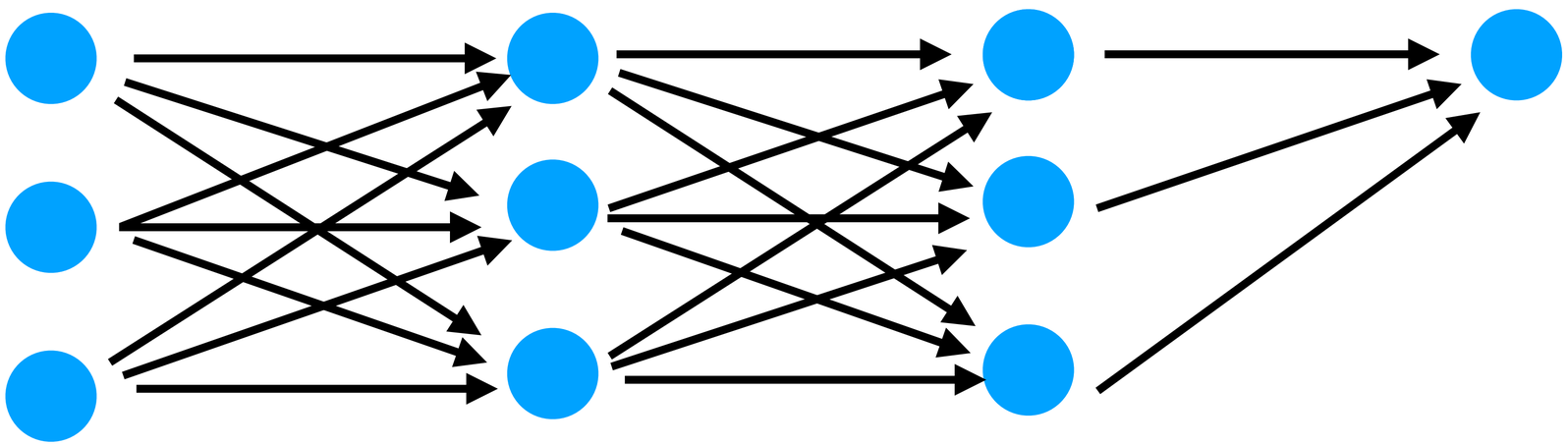
2-layer:

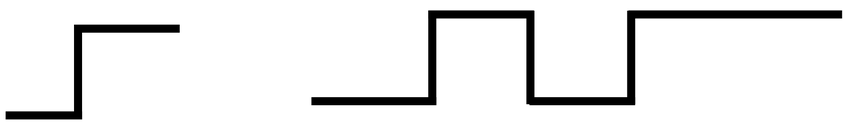
These outputs are now input features to the next layer

"Features" are now decision boundaries (partitions)

All linear combination of those partitions give complex partitions

Features of MLPs

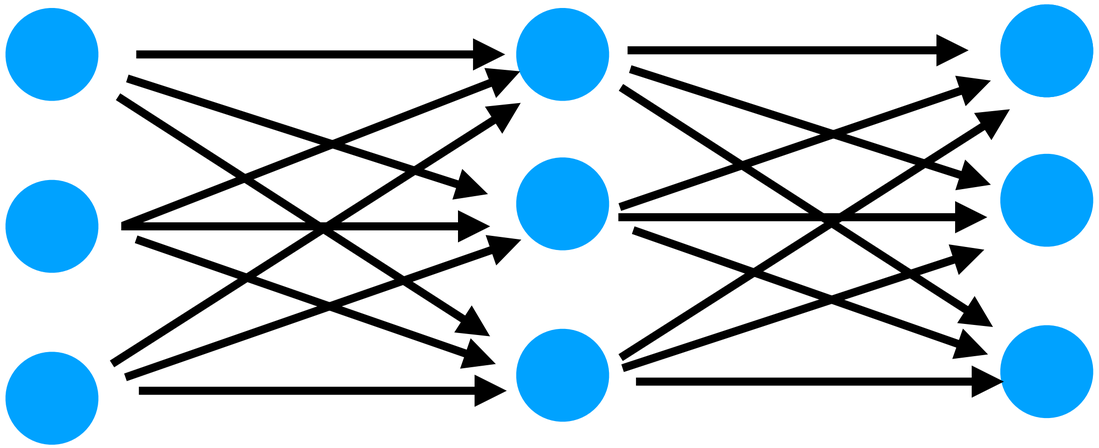




Layer 1 Layer 2

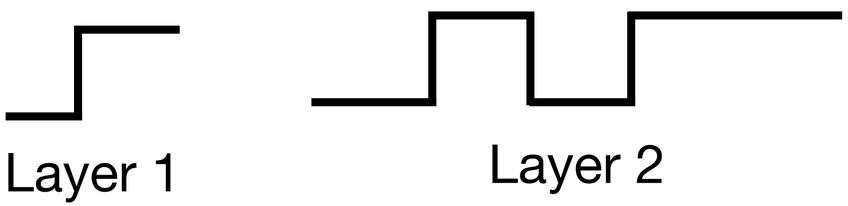
These complex outputs become the features for the new layer

Features of MLPs



• • •

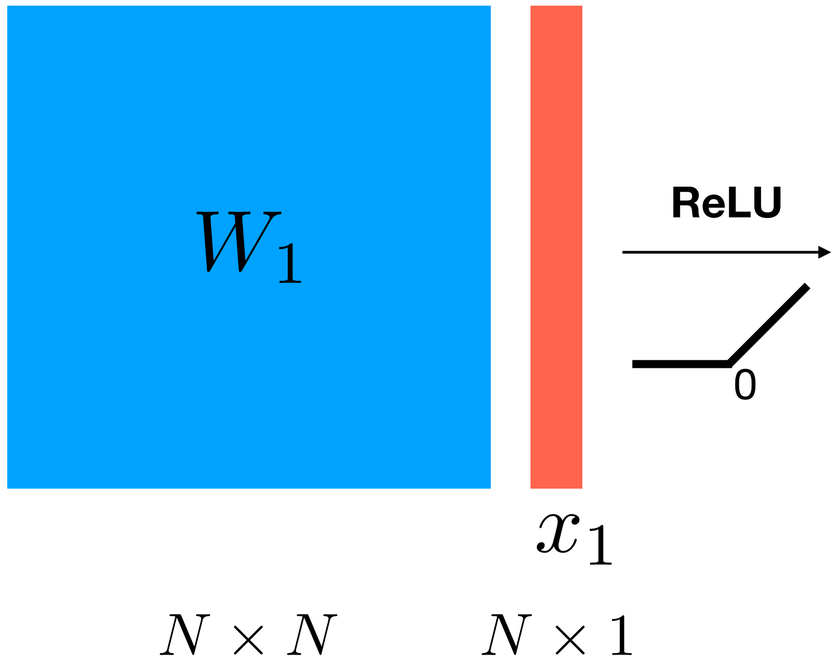
Deep Neural Networks



Computational Graph representation of Neural Networks

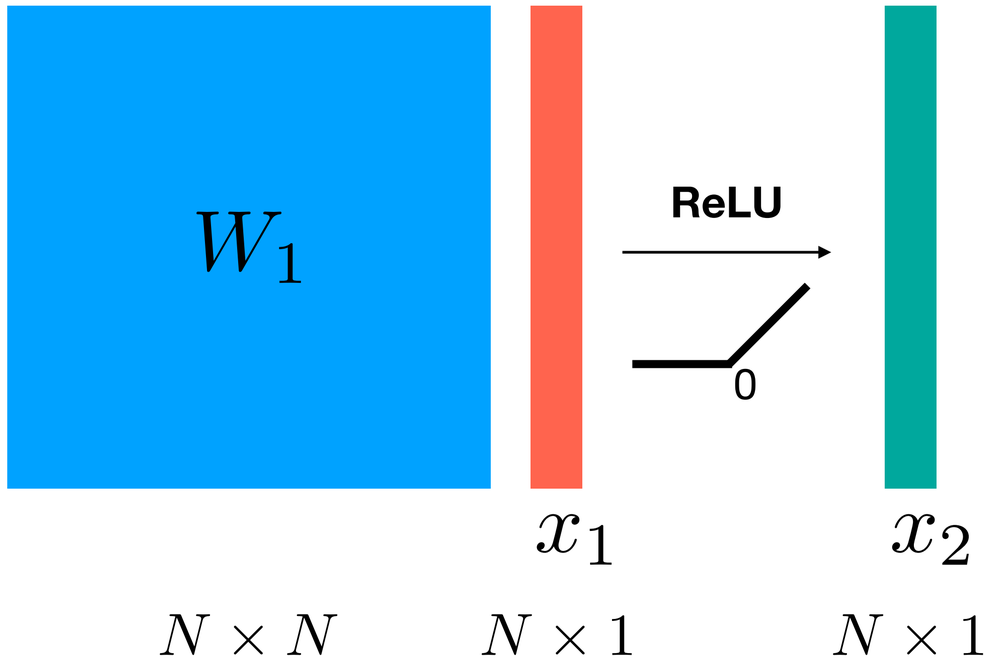
Neural Networks

Fully connected layers



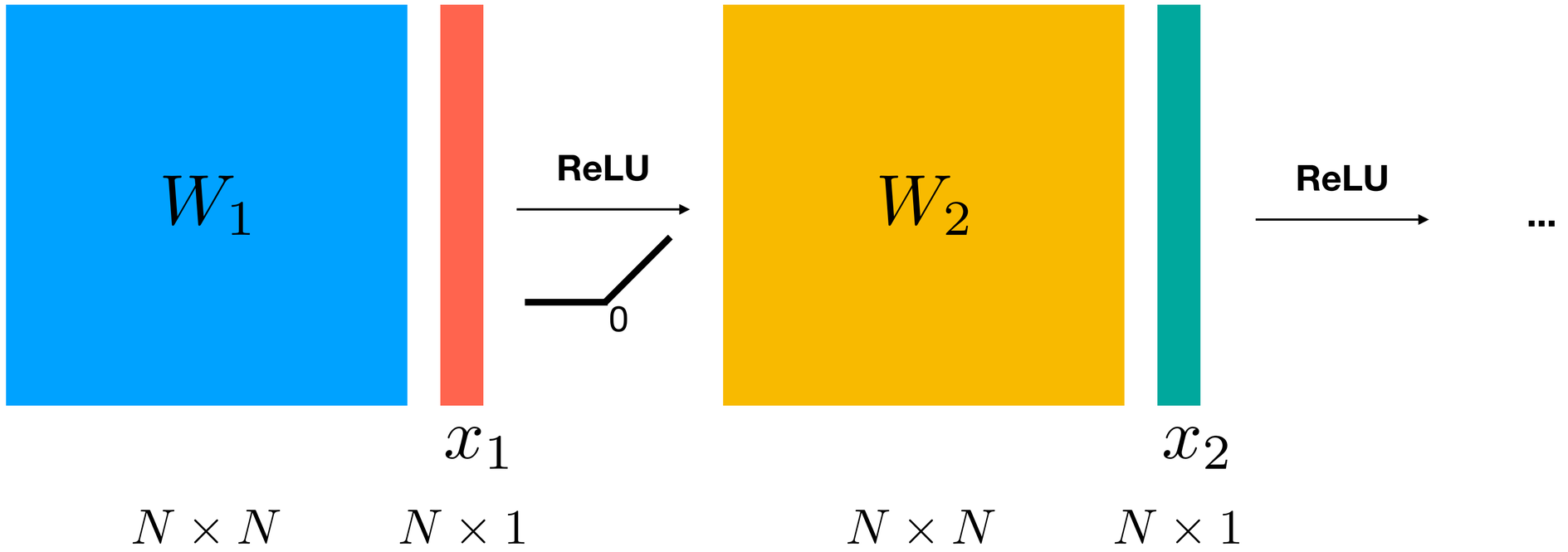
Neural Networks

Fully connected layers

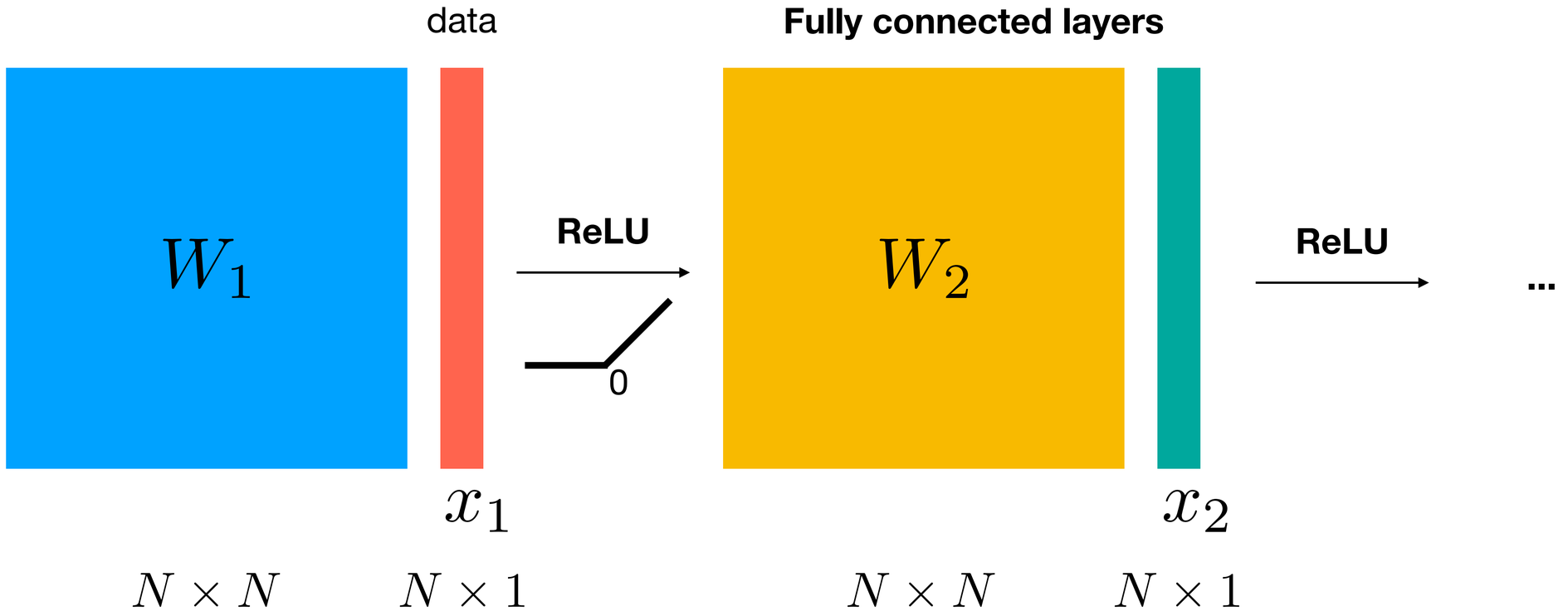


Neural Networks

Fully connected layers

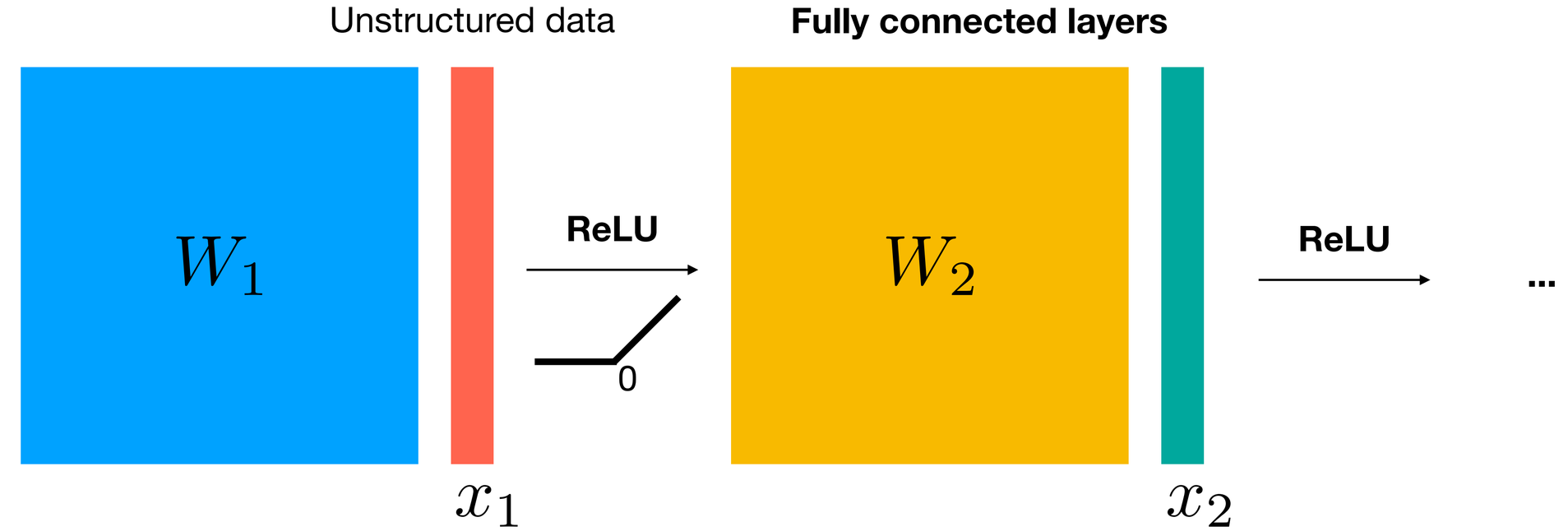


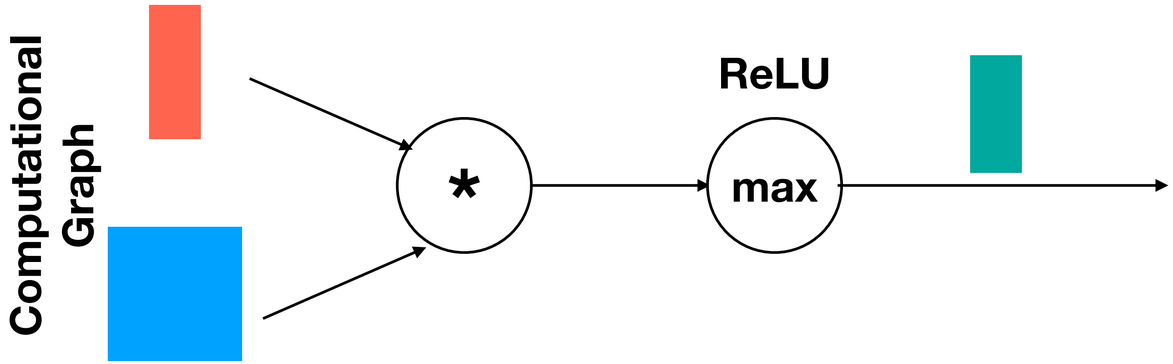
Neural Networks



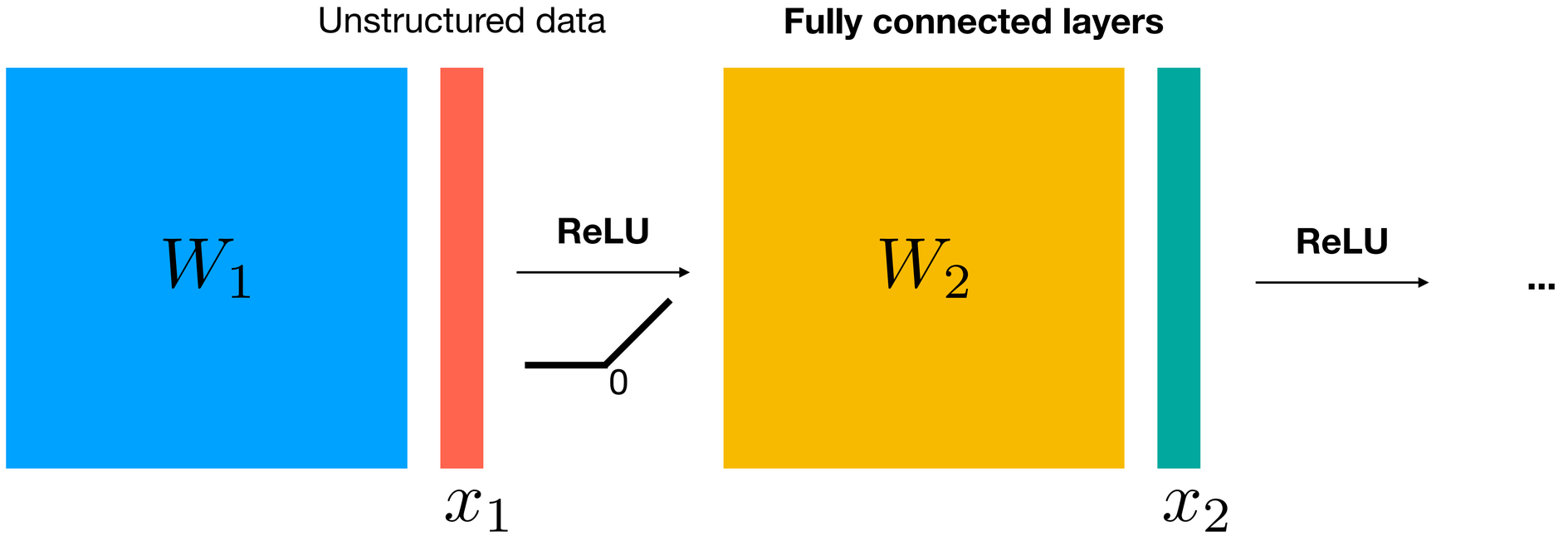
N represents number of pixels in an image

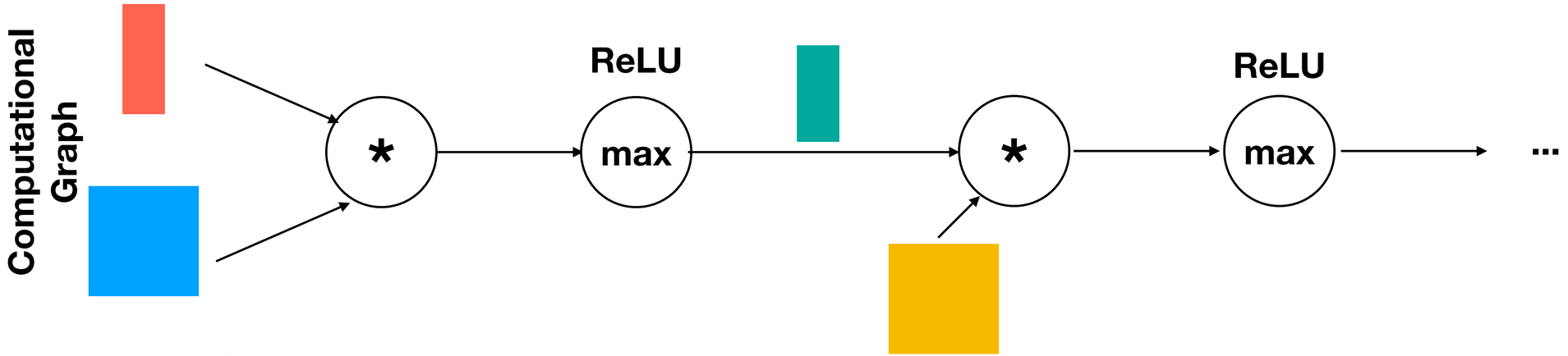
Neural Networks



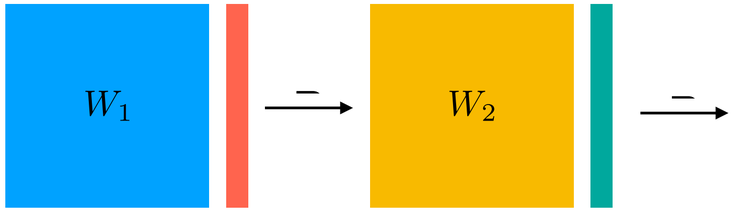


Neural Networks

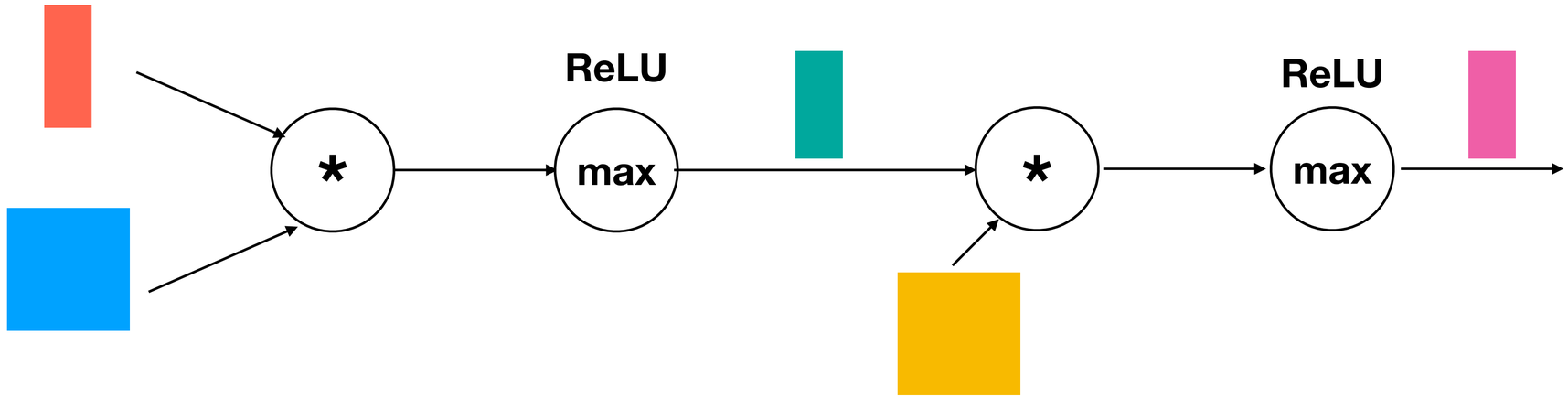




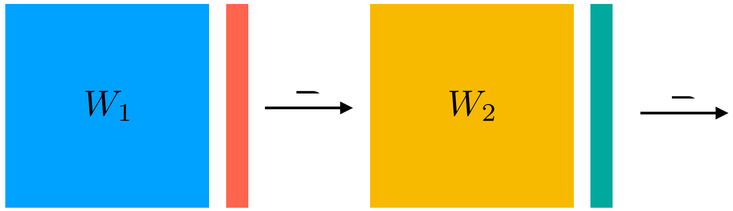
Two-layer model



Fully connected layers

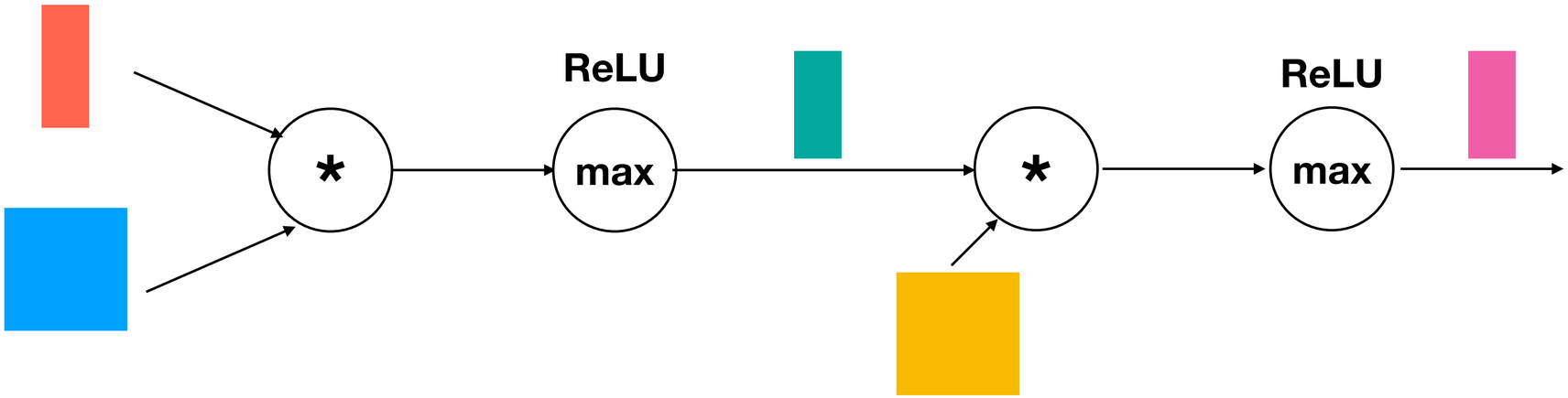


What can be a loss function ?



Two-layer model

Fully connected layers

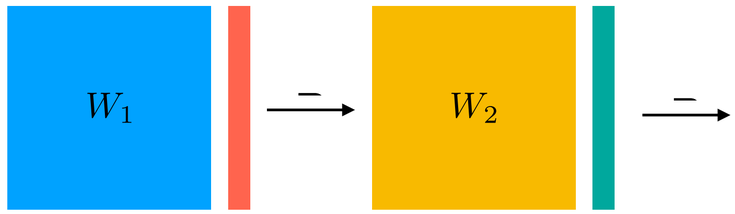


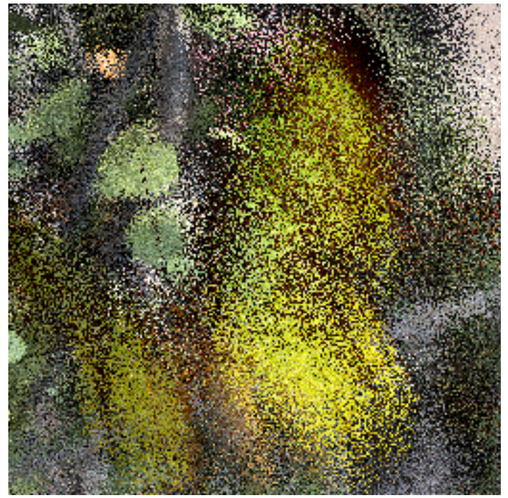
Reference

What can be a loss function ?

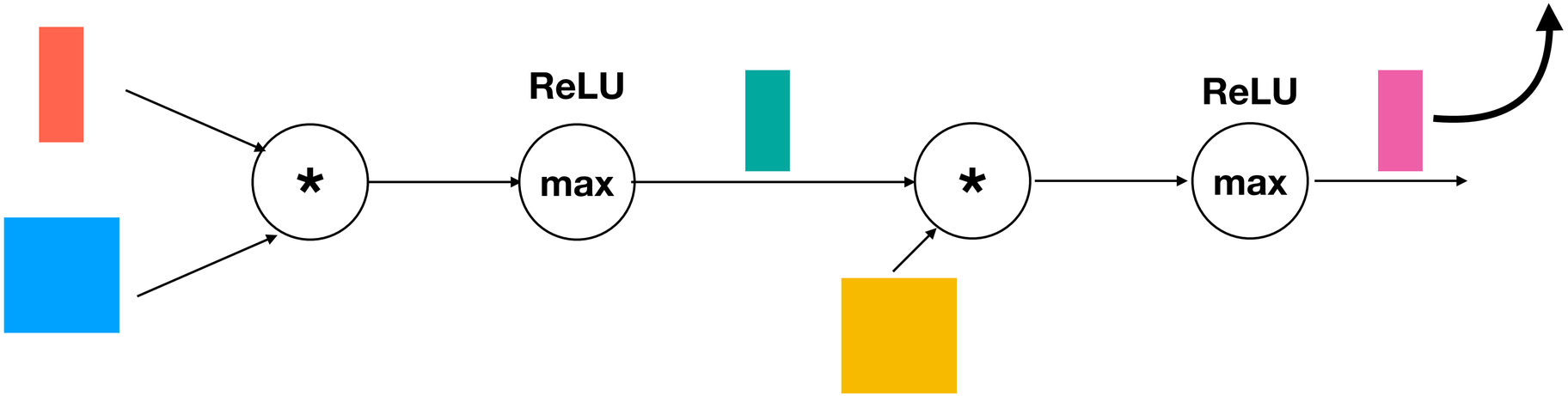
Two-layer model

Fully connected layers



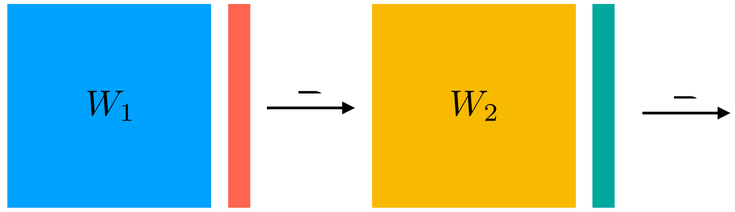


Reference

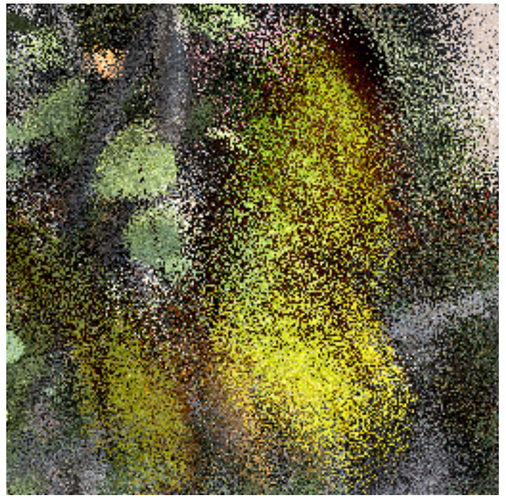


What can be a loss function ?

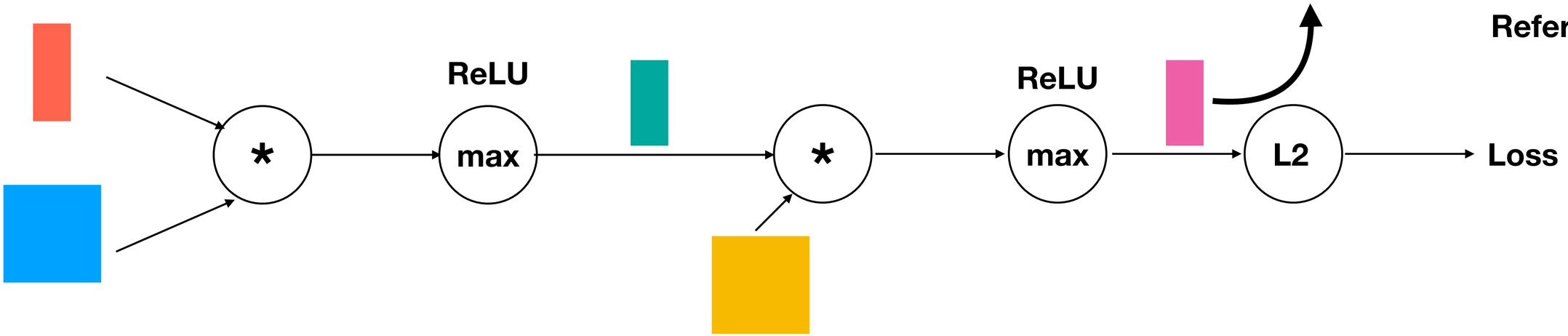
Two-layer model



Fully connected layers

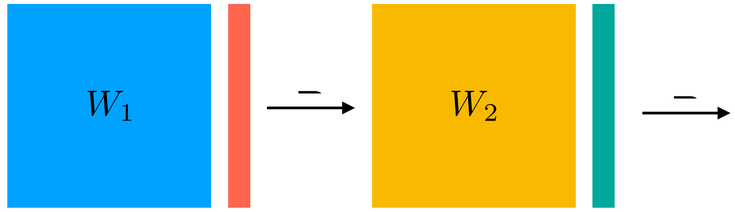


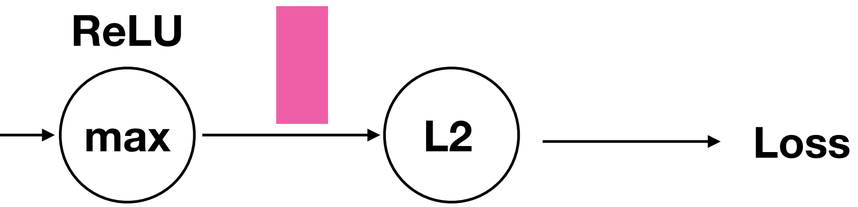
Reference

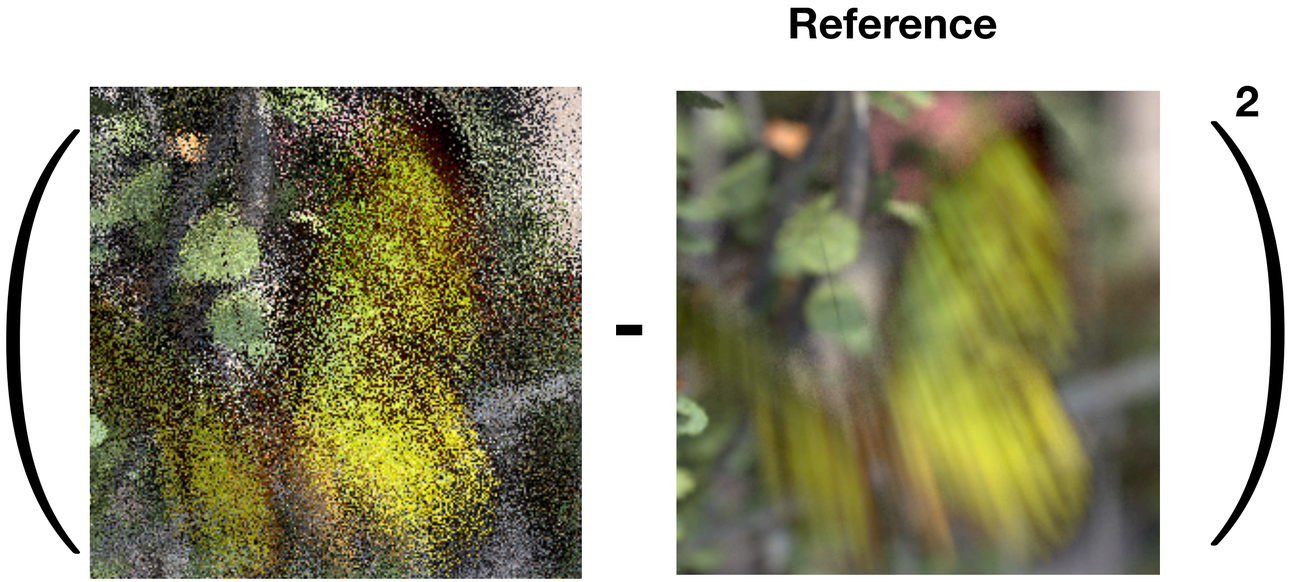


What can be a loss function ?

Two-layer model

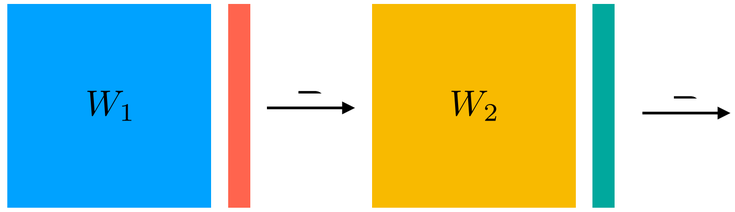


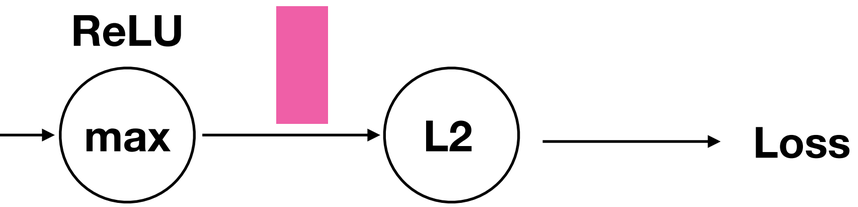


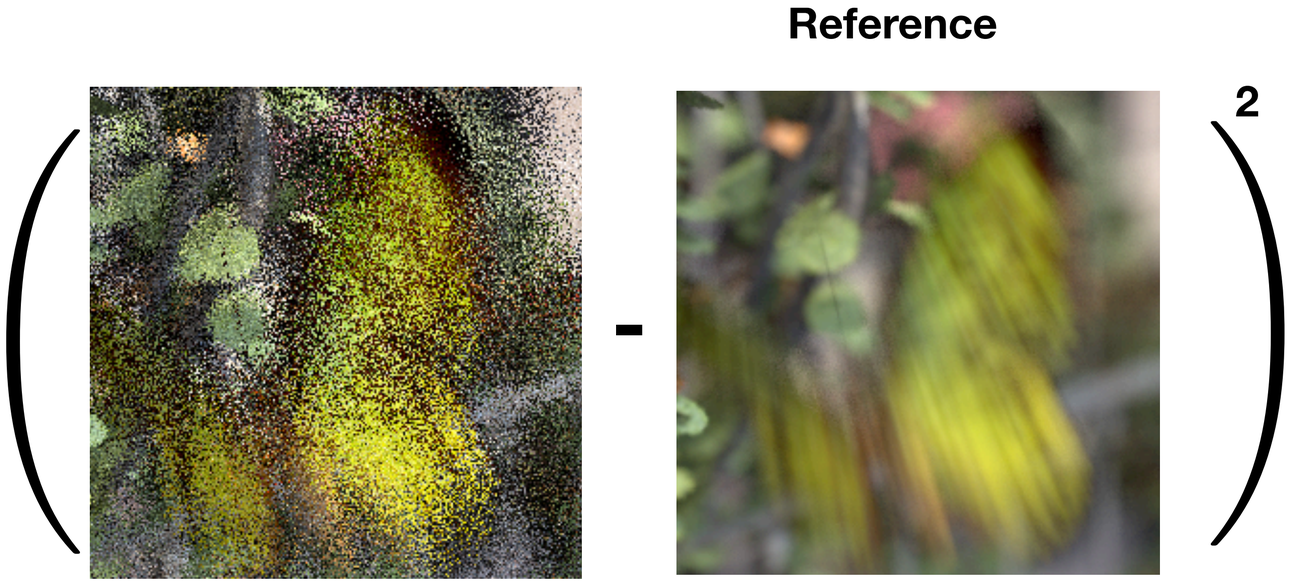


What can be a loss function ?

Two-layer model



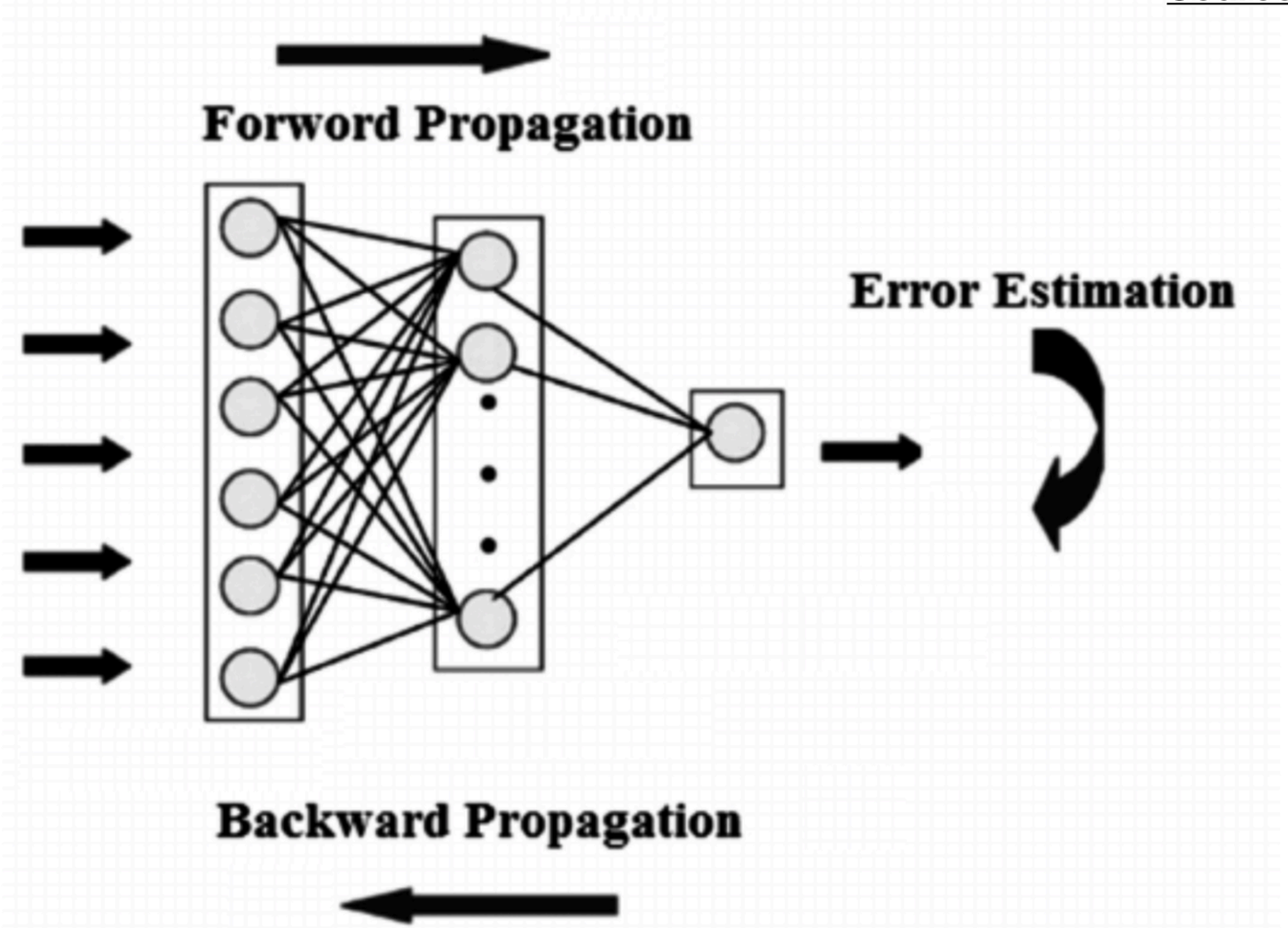




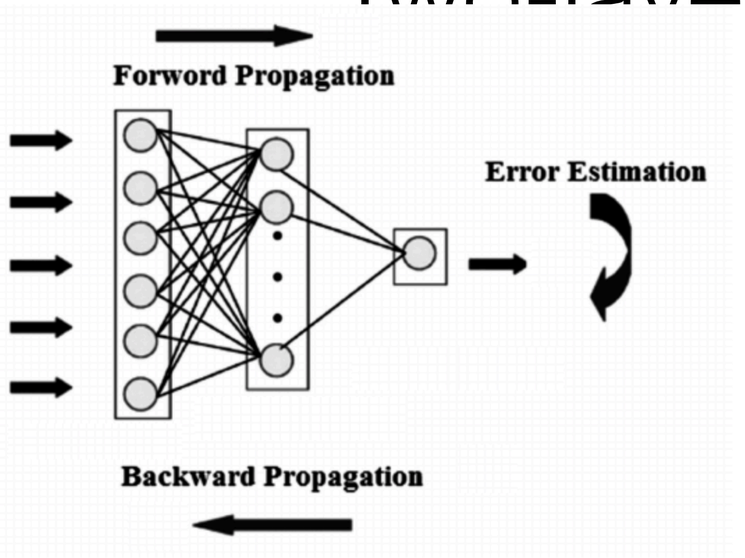
What can be a loss function ?

Two-layer model: Back propagation

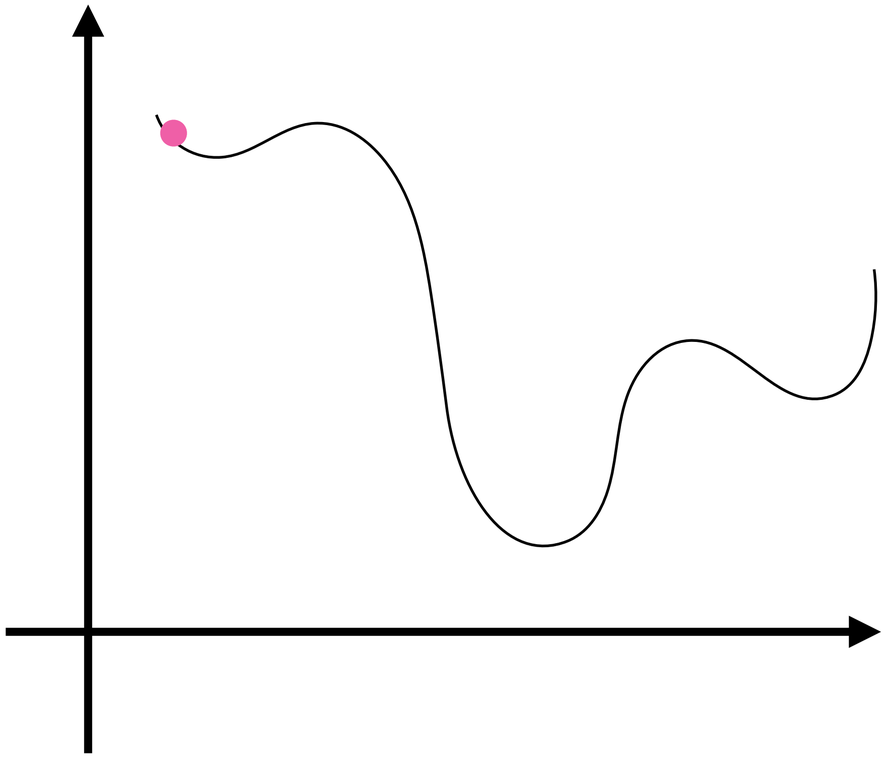
Source link



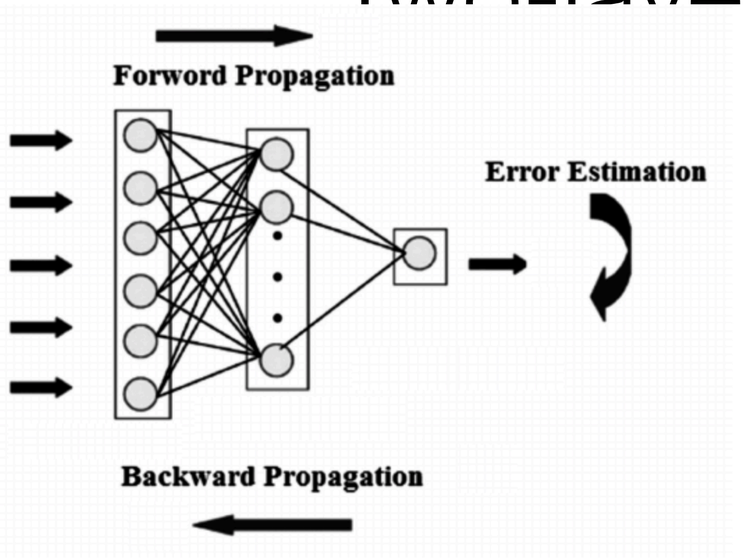
Two-layer model: Back propagation



Gradient Descent Algorithm
for back propagation

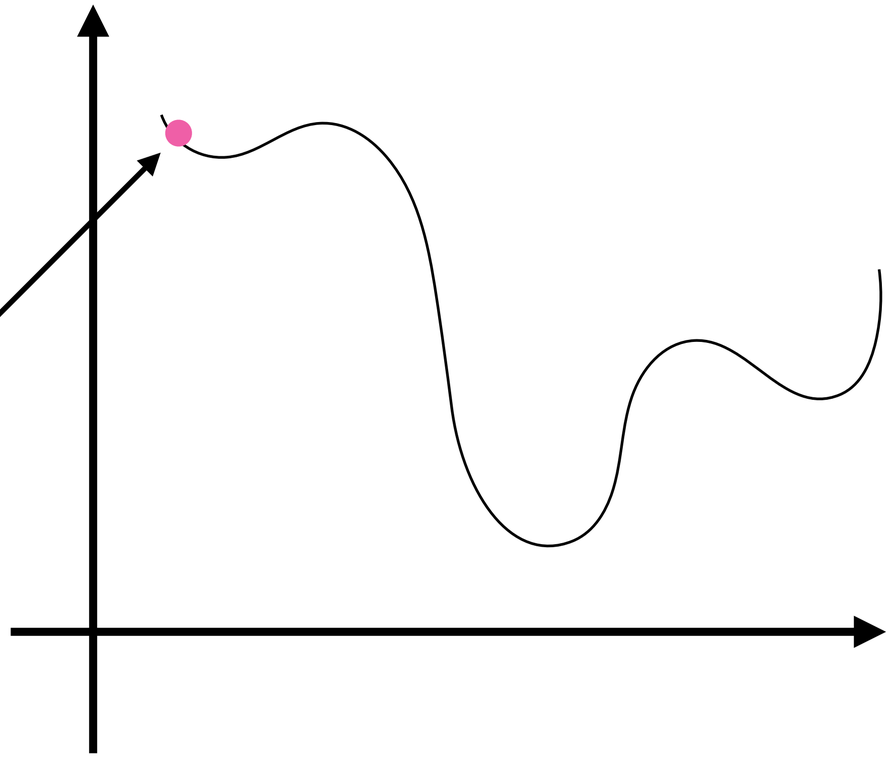


Two-layer model: Back propagation

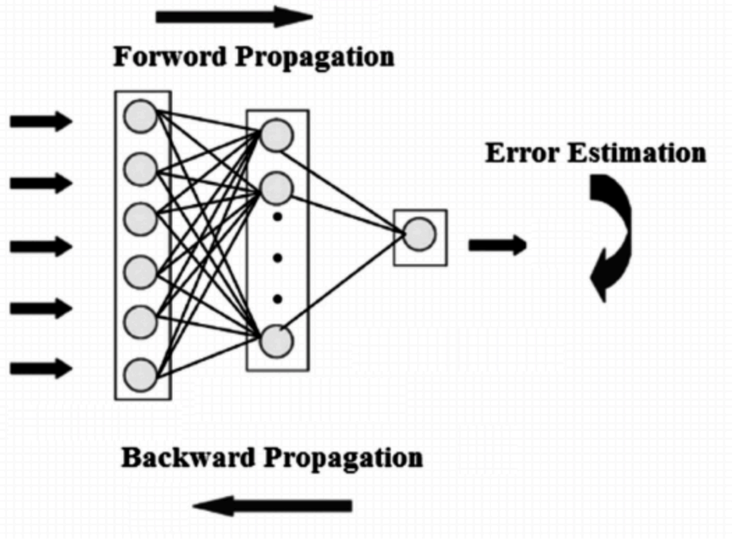


Gradient Descent Algorithm
for back propagation

Random initialization

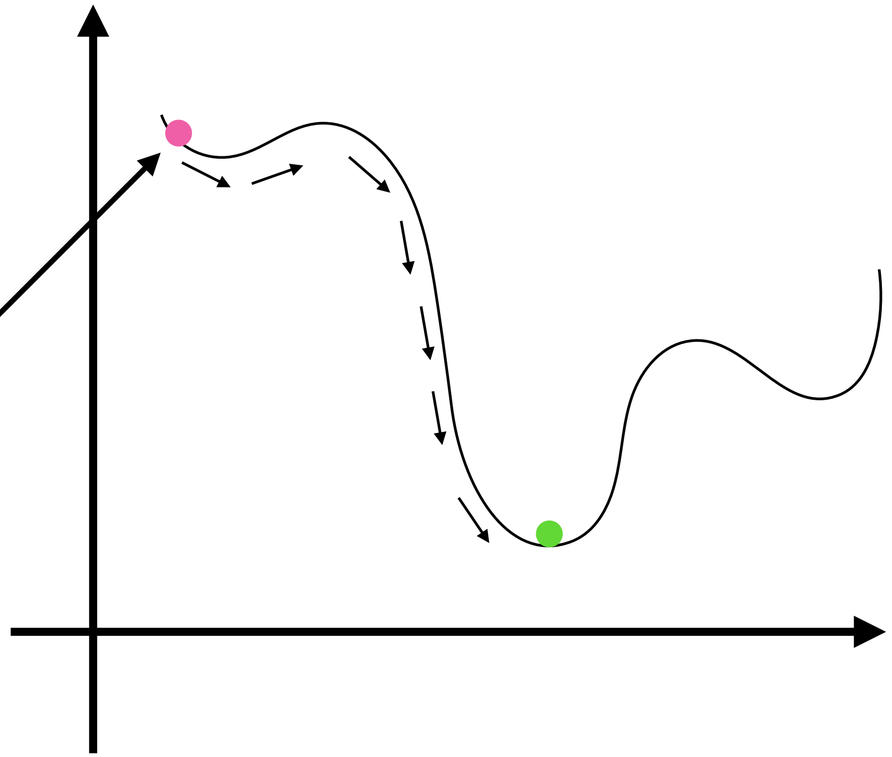


Two-layer model: Back propagation

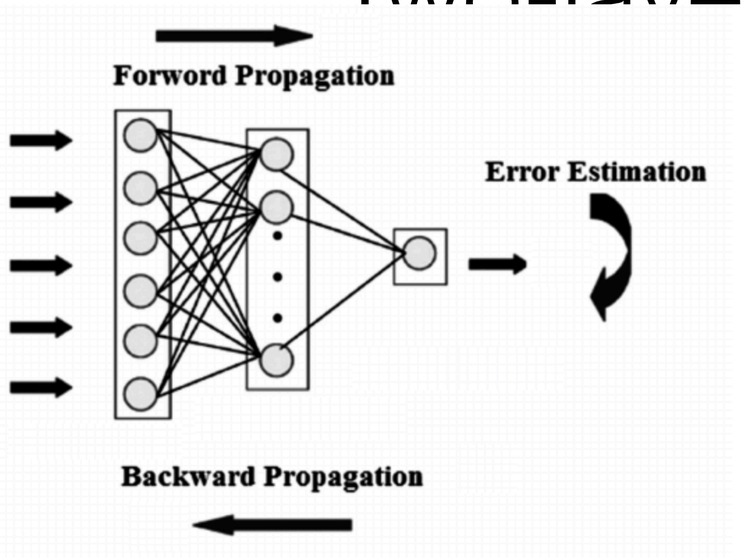


Gradient Descent Algorithm
for back propagation

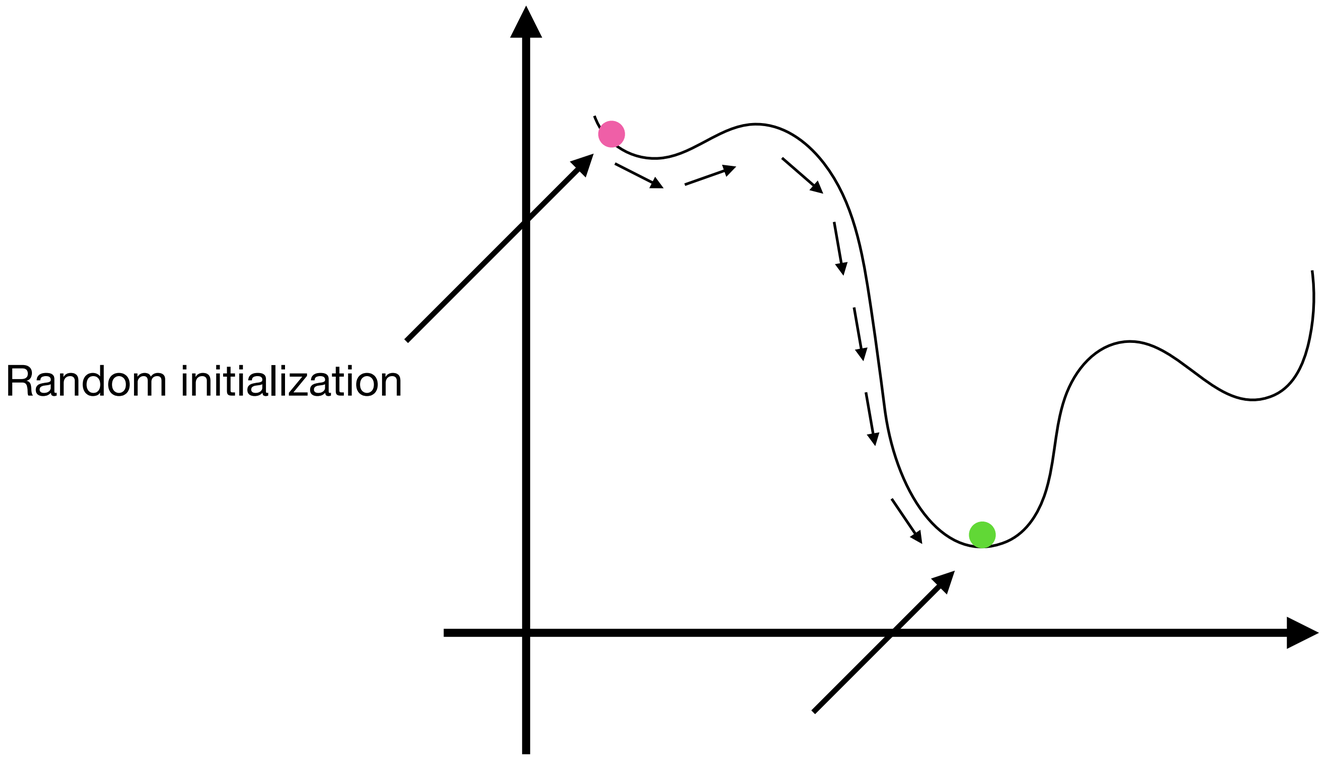
Random initialization



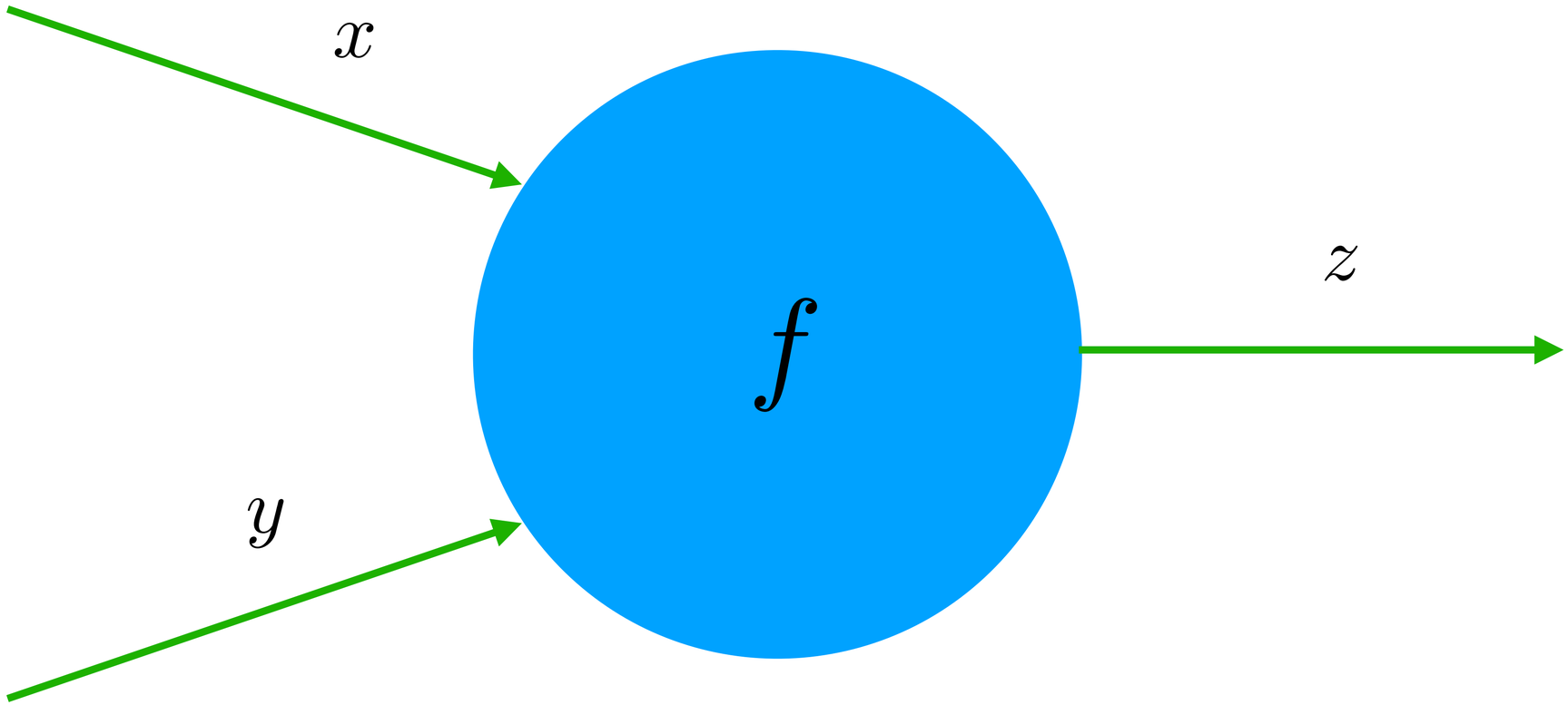
Two-layer model: Back propagation



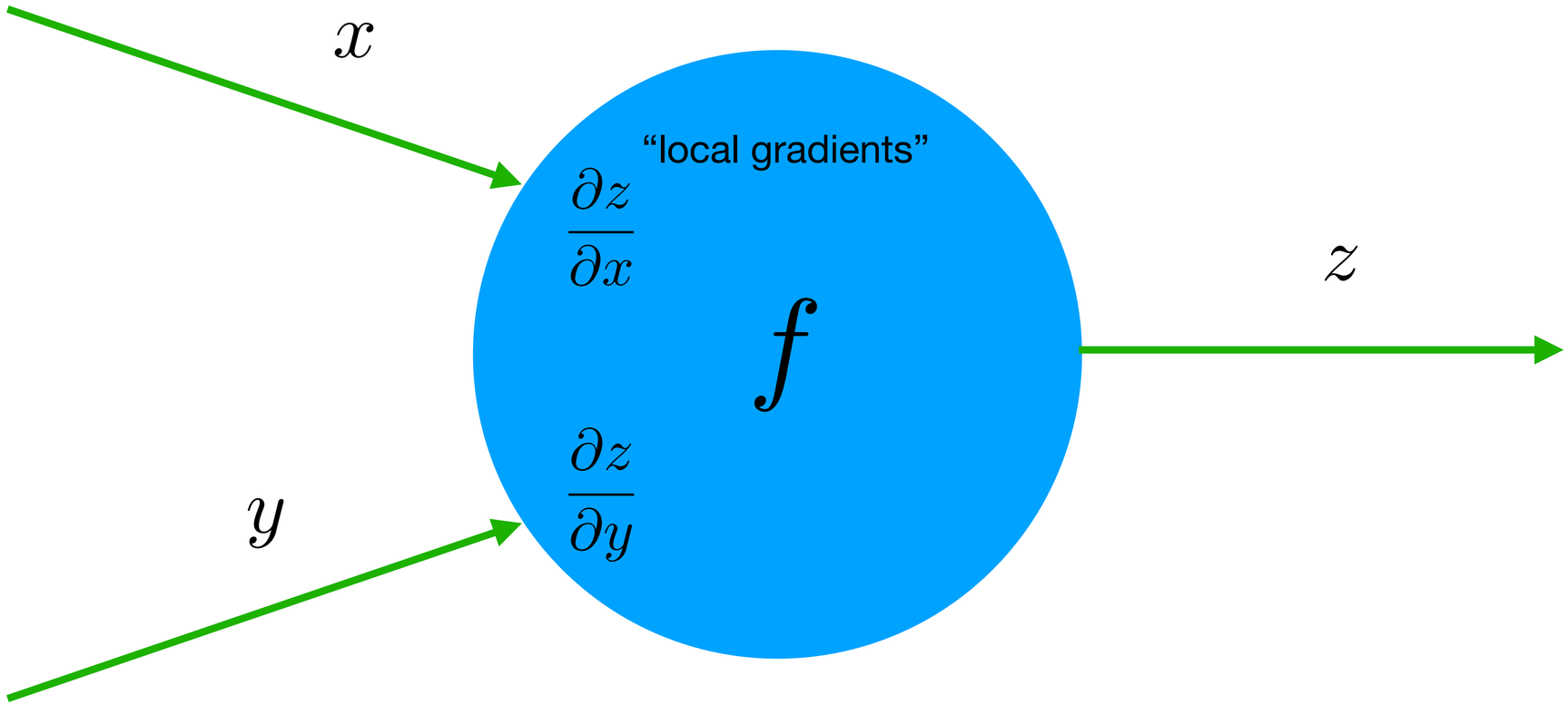
Gradient Descent Algorithm
for back propagation



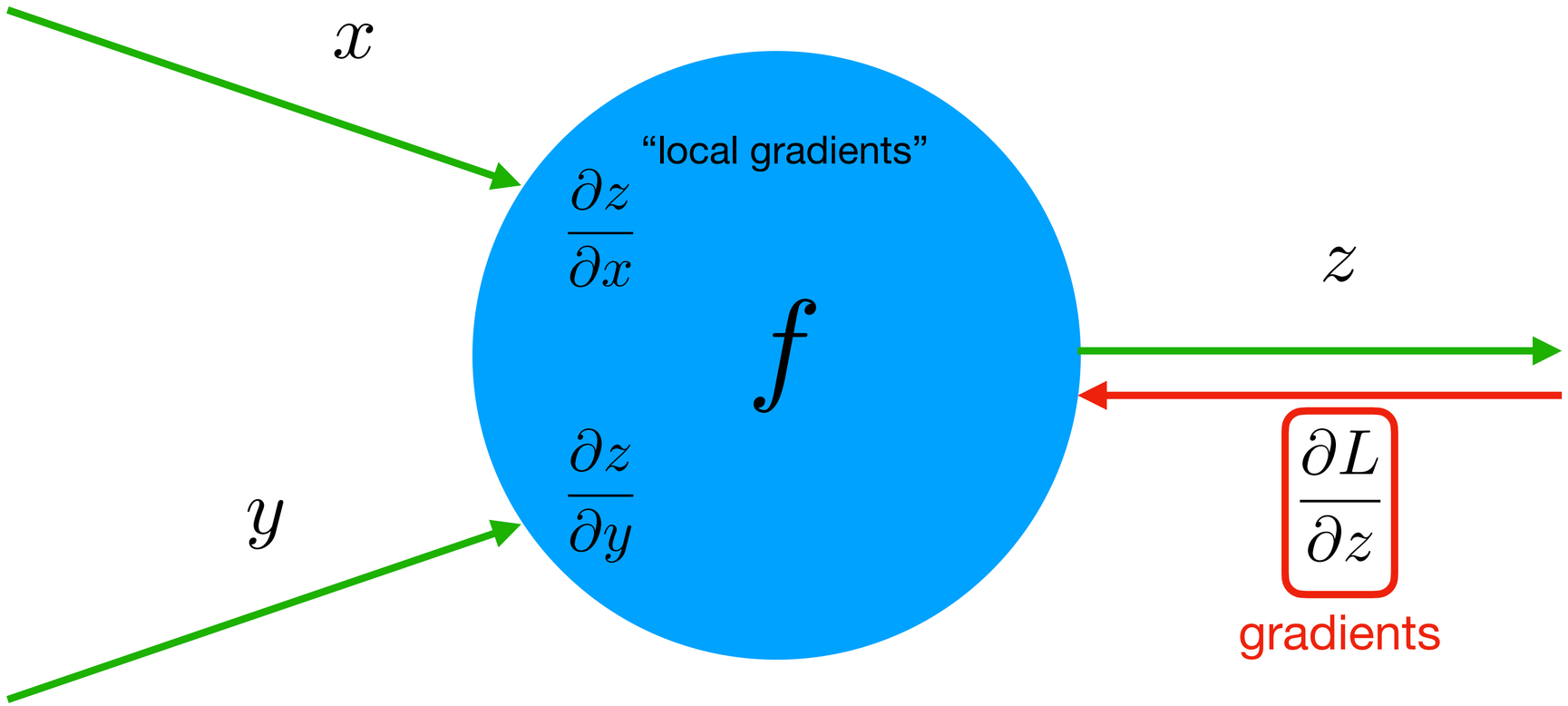
Back Propagation



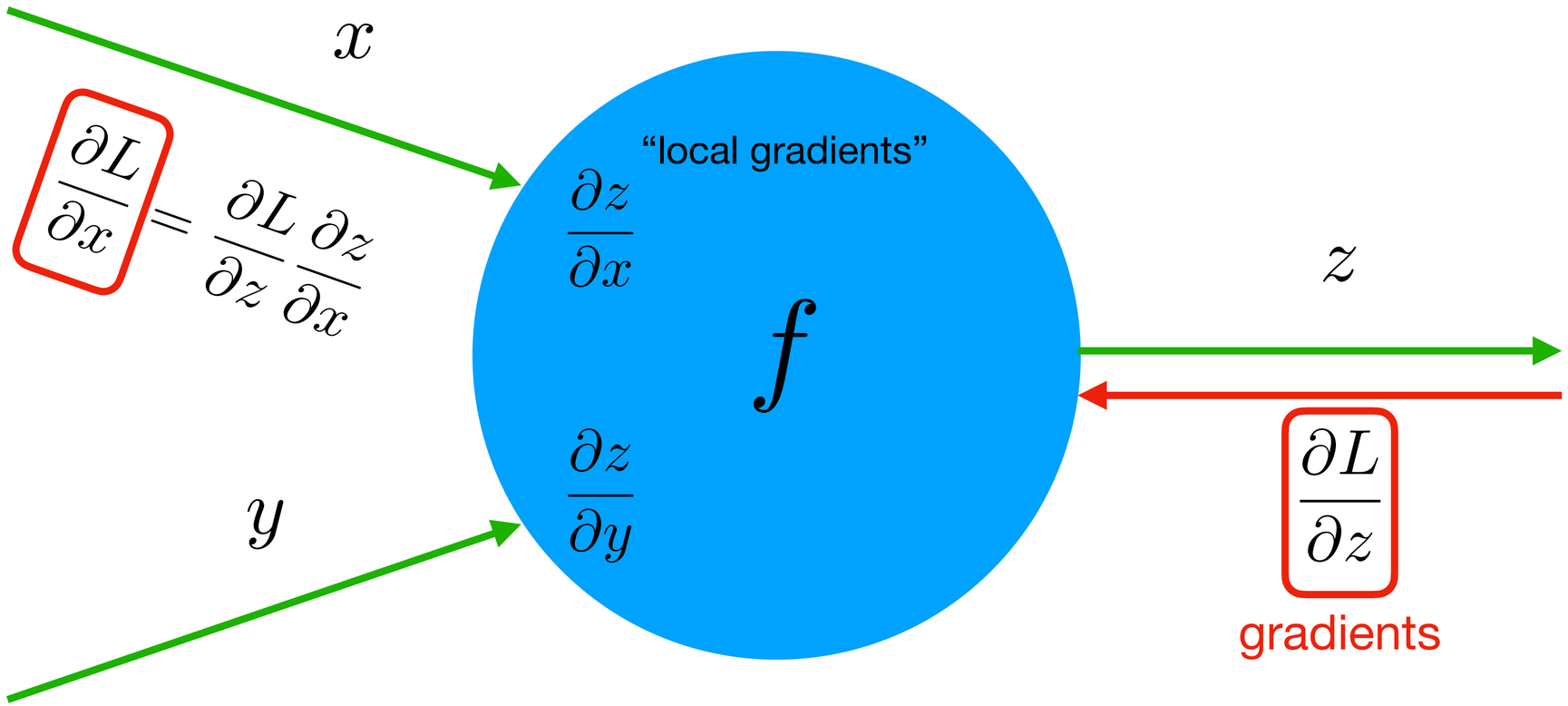
Back Propagation



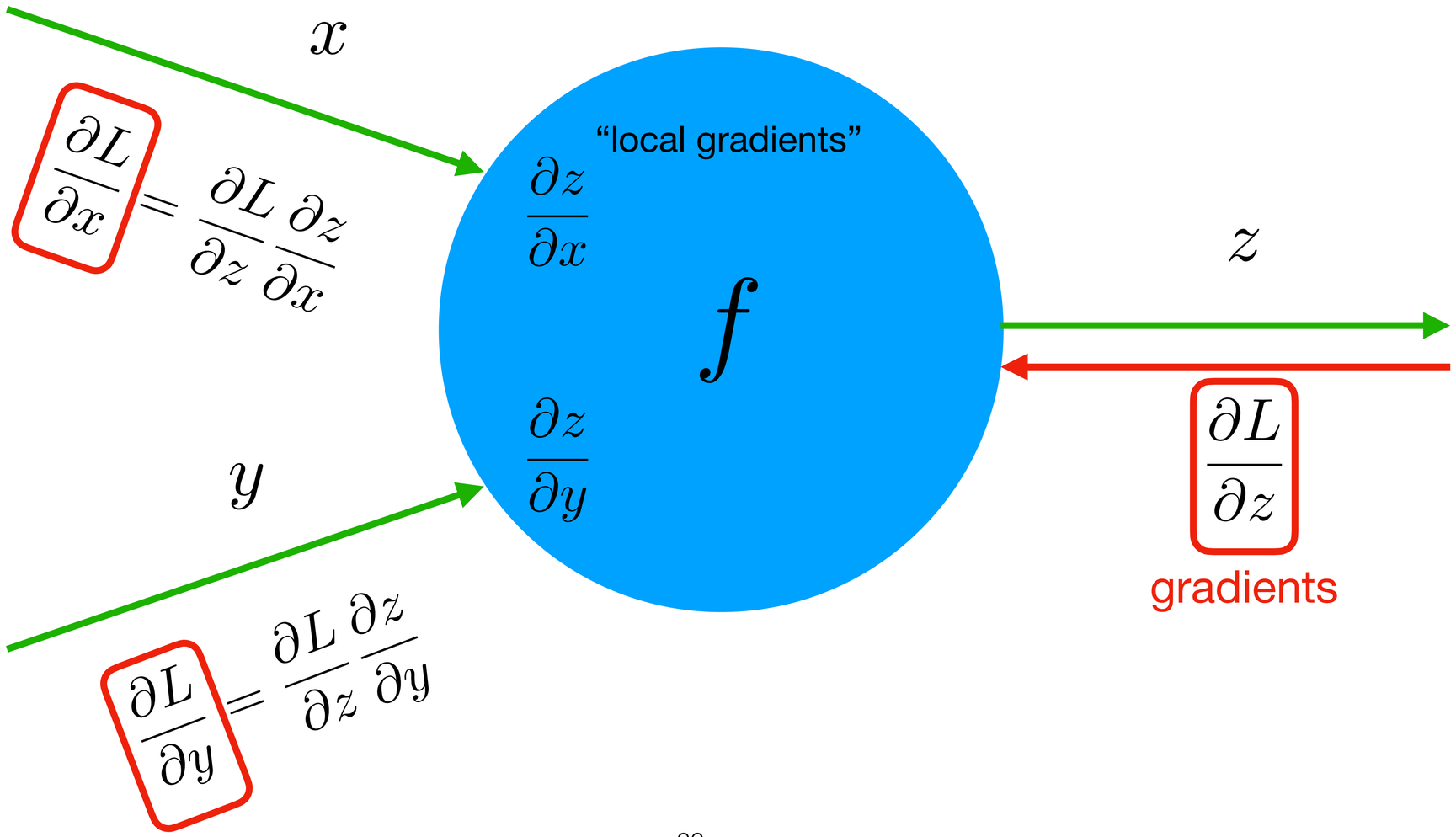
Back Propagation



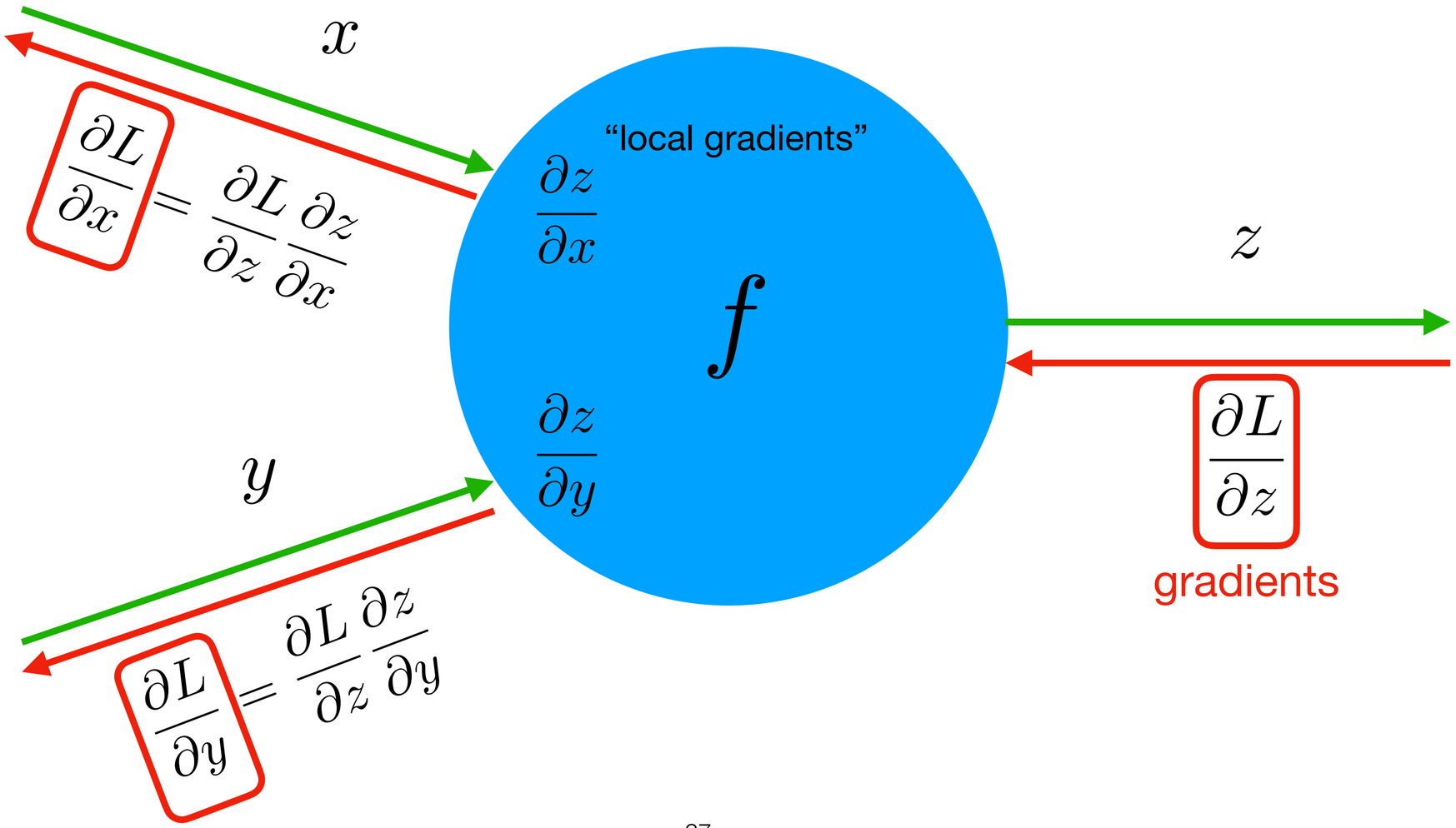
Back Propagation



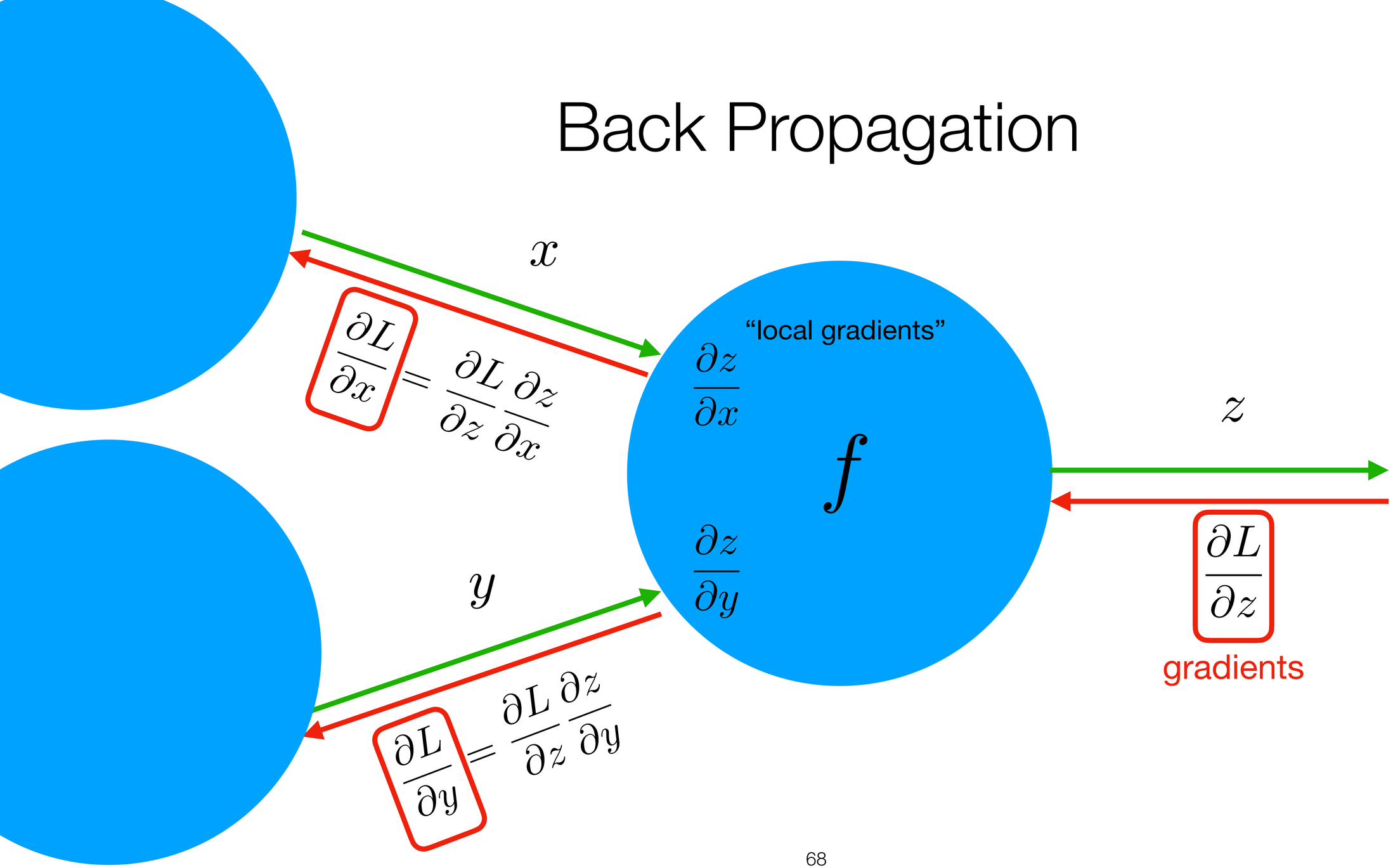
Back Propagation



Back Propagation



Back Propagation

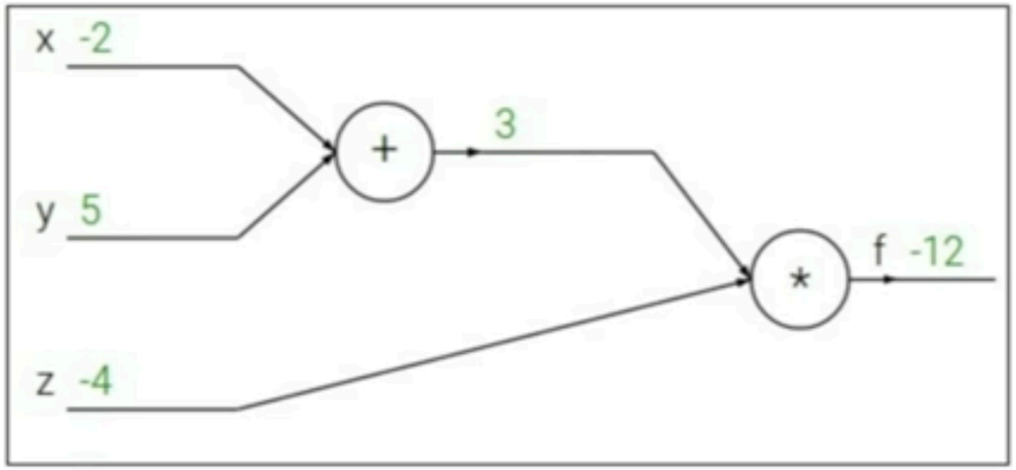


Back Propagation

Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



Slides courtesy: Stanford Online Course

Back Propagation

Backpropagation: a simple example

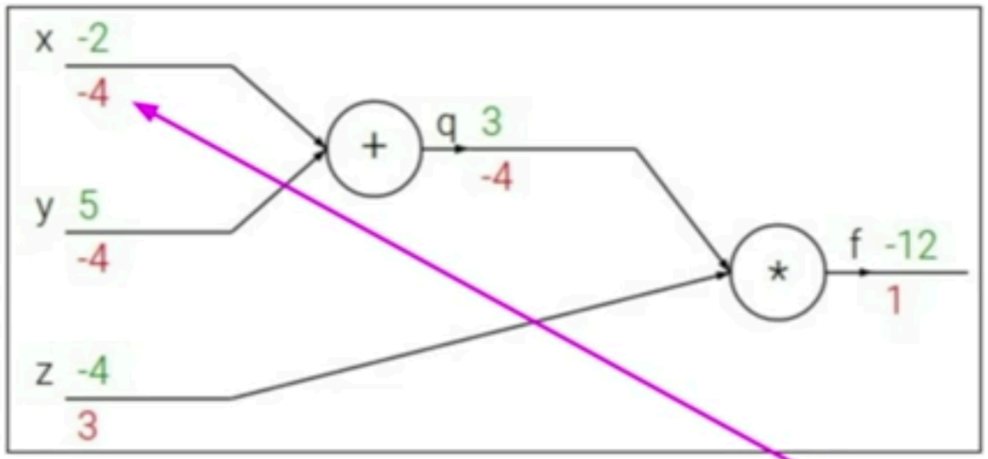
$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

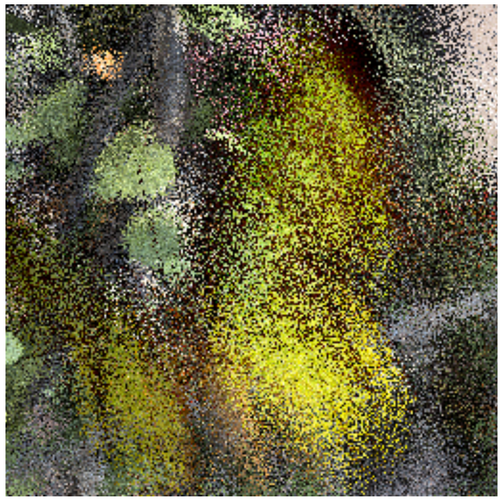
Slides courtesy: Stanford Online Course

Machine Learning for Filtering Monte Carlo Noise

Kalantari et al. [SIGGRAPH 2015]

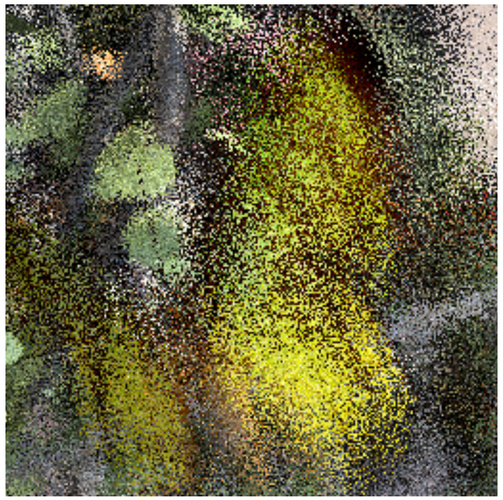
Reconstruction / Denoising

$$\hat{\mathbf{c}}_i = \frac{\sum_{j \in \mathcal{N}(i)} d_{i,j} \bar{\mathbf{c}}_j}{\sum_{j \in \mathcal{N}(i)} d_{i,j}} \quad , \quad \hat{\mathbf{c}} = \{\hat{c}_r, \hat{c}_g, \hat{c}_b\}$$



Reconstruction / Denoising

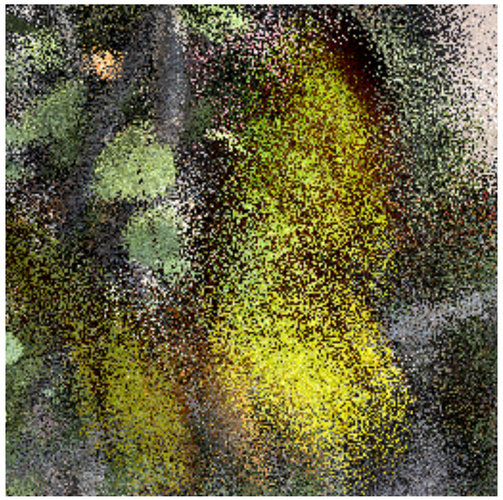
$$\hat{\mathbf{c}}_i = \frac{\sum_{j \in \mathcal{N}(i)} d_{i,j} \bar{\mathbf{c}}_j}{\sum_{j \in \mathcal{N}(i)} d_{i,j}} \quad , \quad \hat{\mathbf{c}} = \{\hat{c}_r, \hat{c}_g, \hat{c}_b\}$$



Reconstruction / Denoising

$$\hat{\mathbf{c}}_i = \frac{\sum_{j \in \mathcal{N}(i)} d_{i,j} \bar{\mathbf{c}}_j}{\sum_{j \in \mathcal{N}(i)} d_{i,j}} \quad , \quad \hat{\mathbf{c}} = \{\hat{c}_r, \hat{c}_g, \hat{c}_b\}$$

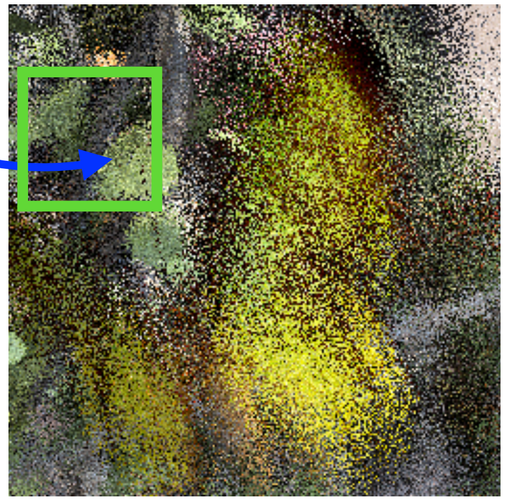
Pixel neighborhood



Reconstruction / Denoising

$$\hat{\mathbf{c}}_i = \frac{\sum_{j \in \mathcal{N}(i)} d_{i,j} \bar{\mathbf{c}}_j}{\sum_{j \in \mathcal{N}(i)} d_{i,j}}, \quad \hat{\mathbf{c}} = \{\hat{c}_r, \hat{c}_g, \hat{c}_b\}$$

Pixel neighborhood

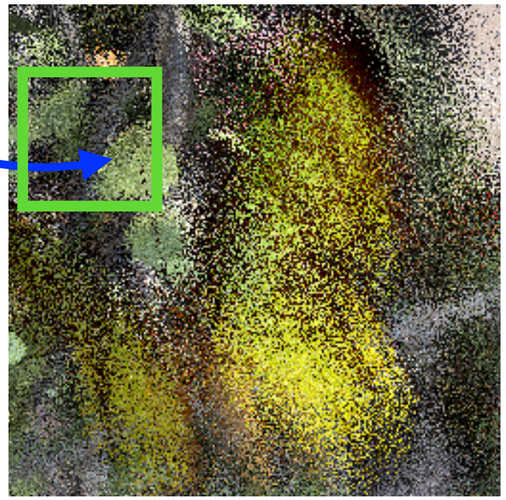


Reconstruction / Denoising

Filter weights

$$\hat{\mathbf{c}}_i = \frac{\sum_{j \in \mathcal{N}(i)} d_{i,j} \bar{\mathbf{c}}_j}{\sum_{j \in \mathcal{N}(i)} d_{i,j}}, \quad \hat{\mathbf{c}} = \{\hat{c}_r, \hat{c}_g, \hat{c}_b\}$$

Pixel neighborhood



Filter weights

Filter weights

$$\hat{\mathbf{c}}_i = \frac{\sum_{j \in \mathcal{N}(i)} d_{i,j} \bar{\mathbf{c}}_j}{\sum_{j \in \mathcal{N}(i)} d_{i,j}}$$

Pixel neighborhood

Filter weights

Filter weights

$$\hat{\mathbf{c}}_i = \frac{\sum_{j \in \mathcal{N}(i)} d_{i,j} \bar{\mathbf{c}}_j}{\sum_{j \in \mathcal{N}(i)} d_{i,j}}$$

Pixel neighborhood

For cross Bilateral filters:

Filter weights

Filter weights

$$\hat{\mathbf{c}}_i = \frac{\sum_{j \in \mathcal{N}(i)} d_{i,j} \bar{\mathbf{c}}_j}{\sum_{j \in \mathcal{N}(i)} d_{i,j}}$$

Pixel neighborhood

For cross Bilateral filters:

$$d_{i,j} = \exp\left[- \frac{\|\bar{\mathbf{p}}_i - \bar{\mathbf{p}}_j\|^2}{2\alpha_i^2} \right] \times \exp\left[- \frac{D(\bar{\mathbf{c}}_i, \bar{\mathbf{c}}_j)}{2\beta_i^2} \right] \times \prod_{k=1}^{K} \exp\left[- \frac{D_k(\bar{\mathbf{f}}_{i,k}, \bar{\mathbf{f}}_{j,k})}{2\gamma_{k,i}^2} \right],$$

Filter weights

Filter weights

$$\hat{\mathbf{c}}_i = \frac{\sum_{j \in \mathcal{N}(i)} d_{i,j} \bar{\mathbf{c}}_j}{\sum_{j \in \mathcal{N}(i)} d_{i,j}}$$

Pixel neighborhood

For cross Bilateral filters:

$$d_{i,j} = \exp\left[- \frac{\|\bar{\mathbf{p}}_i - \bar{\mathbf{p}}_j\|^2}{2\alpha_i^2} \right] \times \exp\left[- \frac{D(\bar{\mathbf{c}}_i, \bar{\mathbf{c}}_j)}{2\beta_i^2} \right] \times \prod_{k=1}^{K} \exp\left[- \frac{D_k(\bar{\mathbf{f}}_{i,k}, \bar{\mathbf{f}}_{j,k})}{2\gamma_{k,i}^2} \right],$$

Filter weights

Filter weights

$$\hat{\mathbf{c}}_i = \frac{\sum_{j \in \mathcal{N}(i)} d_{i,j} \bar{\mathbf{c}}_j}{\sum_{j \in \mathcal{N}(i)} d_{i,j}}$$

Pixel neighborhood

For cross Bilateral filters:

$$d_{i,j} = \exp\left[- \frac{\|\bar{\mathbf{p}}_i - \bar{\mathbf{p}}_j\|^2}{2\alpha_i^2} \right] \times \exp\left[- \frac{D(\bar{\mathbf{c}}_i, \bar{\mathbf{c}}_j)}{2\beta_i^2} \right] \times \prod_{k=1}^{K} \exp\left[- \frac{D_k(\bar{\mathbf{f}}_{i,k}, \bar{\mathbf{f}}_{j,k})}{2\gamma_{k,i}^2} \right],$$

Sen and Darabi [2012]

Filter weights

For cross Bilateral filters:

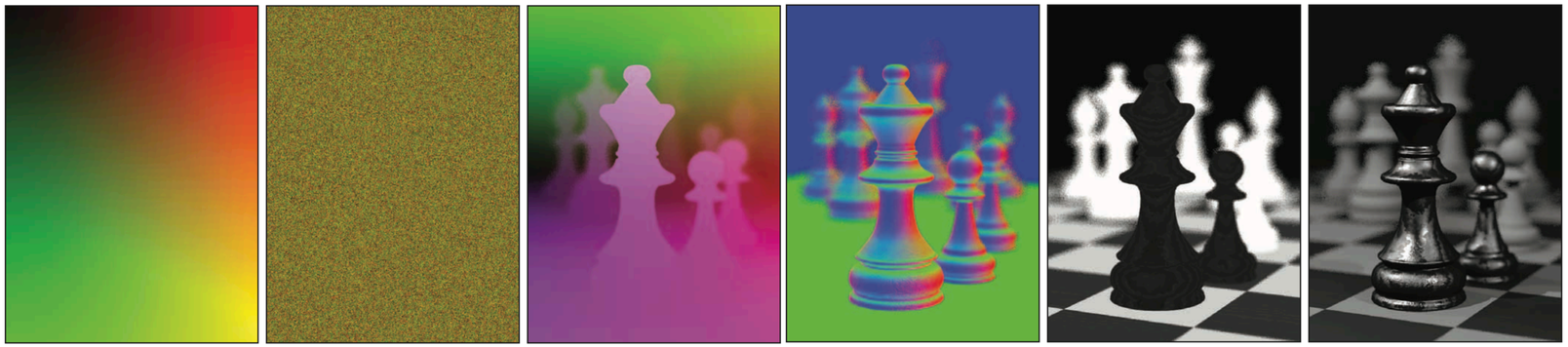
$$d_{i,j} = \exp\left[-\frac{\|\bar{\mathbf{p}}_i - \bar{\mathbf{p}}_j\|^2}{2\alpha_i^2}\right] \times \exp\left[-\frac{D(\bar{\mathbf{c}}_i, \bar{\mathbf{c}}_j)}{2\beta_i^2}\right] \times \prod_{k=1}^{K} \exp\left[-\frac{D_k(\bar{\mathbf{f}}_{i,k}, \bar{\mathbf{f}}_{j,k})}{2\gamma_{k,i}^2}\right],$$

Sen and Darabi [2012]

Filter weights

For cross Bilateral filters:

$$d_{i,j} = \exp\left[-\frac{\|\bar{\mathbf{p}}_i - \bar{\mathbf{p}}_j\|^2}{2\alpha_i^2}\right] \times \exp\left[-\frac{D(\bar{\mathbf{c}}_i, \bar{\mathbf{c}}_j)}{2\beta_i^2}\right] \times \prod_{k=1}^{K} \exp\left[-\frac{D_k(\bar{\mathbf{f}}_{i,k}, \bar{\mathbf{f}}_{j,k})}{2\gamma_{k,i}^2}\right],$$



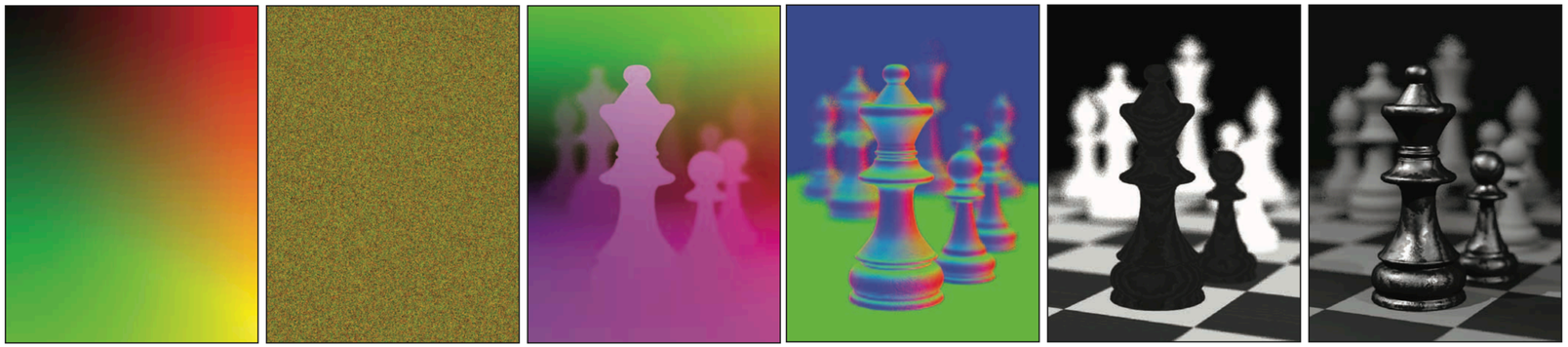
(a) Screen position **(b)** Random parameters **(c)** World space coords. **(d)** Surface normals **(e)** Texture value **(f)** Sample color

Filter weights

For cross Bilateral filters:

$$d_{i,j} = \exp\left[-\frac{\|\bar{\mathbf{p}}_i - \bar{\mathbf{p}}_j\|^2}{2\alpha_i^2}\right] \times \exp\left[-\frac{D(\bar{\mathbf{c}}_i, \bar{\mathbf{c}}_j)}{2\beta_i^2}\right] \times \prod_{k=1}^{K} \exp\left[-\frac{D_k(\bar{\mathbf{f}}_{i,k}, \bar{\mathbf{f}}_{j,k})}{2\gamma_{k,i}^2}\right],$$

Pixel screen coordinates



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

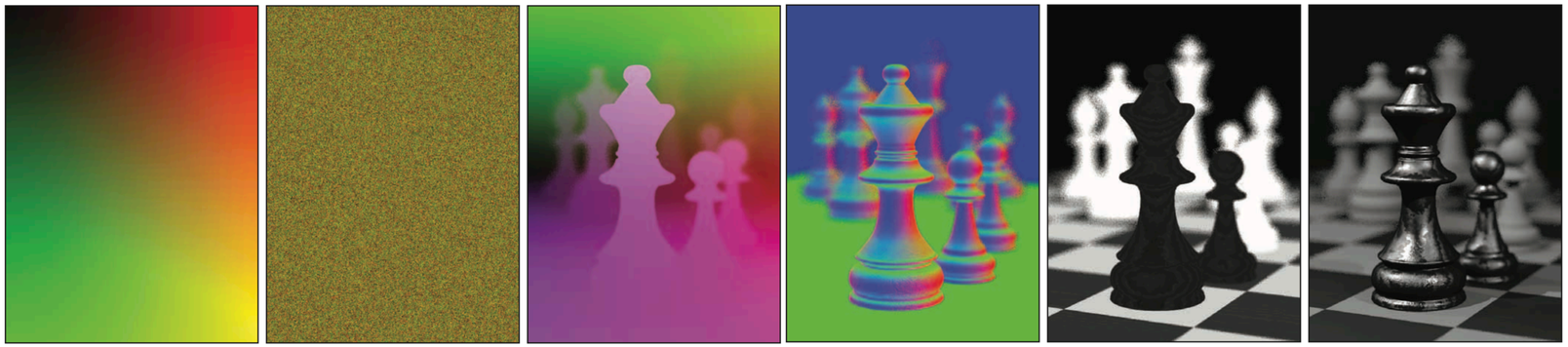
Filter weights

For cross Bilateral filters:

$$d_{i,j} = \exp\left[-\frac{\|\bar{\mathbf{p}}_i - \bar{\mathbf{p}}_j\|^2}{2\alpha_i^2}\right] \times \exp\left[-\frac{D(\bar{\mathbf{c}}_i, \bar{\mathbf{c}}_j)}{2\beta_i^2}\right] \times \prod_{k=1}^{K} \exp\left[-\frac{D_k(\bar{\mathbf{f}}_{i,k}, \bar{\mathbf{f}}_{j,k})}{2\gamma_{k,i}^2}\right],$$

Pixel screen coordinates

Mean sample color value



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

Filter weights

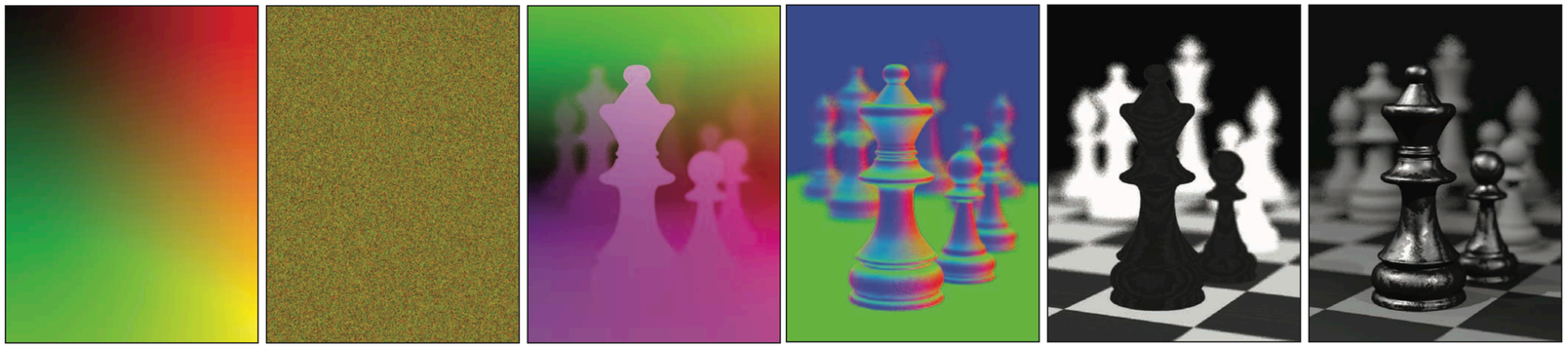
For cross Bilateral filters:

$$d_{i,j} = \exp\left[-\frac{\|\bar{\mathbf{p}}_i - \bar{\mathbf{p}}_j\|^2}{2\alpha_i^2}\right] \times \exp\left[-\frac{D(\bar{\mathbf{c}}_i, \bar{\mathbf{c}}_j)}{2\beta_i^2}\right] \times \prod_{k=1}^{K} \exp\left[-\frac{D_k(\bar{\mathbf{f}}_{i,k}, \bar{\mathbf{f}}_{j,k})}{2\gamma_{k,i}^2}\right],$$

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

Filter weights

For cross Bilateral filters:

$$d_{i,j} = \exp\left[-\frac{\|\bar{\mathbf{p}}_i - \bar{\mathbf{p}}_j\|^2}{2\alpha_i^2}\right] \times \exp\left[-\frac{D(\bar{\mathbf{c}}_i, \bar{\mathbf{c}}_j)}{2\beta_i^2}\right] \times \prod_{k=1}^{K} \exp\left[-\frac{D_k(\bar{\mathbf{f}}_{i,k}, \bar{\mathbf{f}}_{j,k})}{2\gamma_{k,i}^2}\right],$$

Pixel screen coordinates

Mean sample color value

Scene features

Filter weights

For cross Bilateral filters:

$$d_{i,j} = \exp\left[-\frac{\|\bar{\mathbf{p}}_i - \bar{\mathbf{p}}_j\|^2}{2\alpha_i^2}\right] \times \exp\left[-\frac{D(\bar{\mathbf{c}}_i, \bar{\mathbf{c}}_j)}{2\beta_i^2}\right] \times \prod_{k=1}^{K} \exp\left[-\frac{D_k(\bar{\mathbf{f}}_{i,k}, \bar{\mathbf{f}}_{j,k})}{2\gamma_{k,i}^2}\right],$$

Pixel screen coordinates

Mean sample color value

Scene features

What are the **optimal** parameters ?

Neural Network Approach

- Feed-forward Neural network
- Best part: We can learn weights in a training phase
- Back propagation: Important for training weights
- For Back propagation, the Loss function should be differentiable and
- all the intermediate functionals should be differentiable.

One Hidden-layer model

Relative Mean Square Error:

$$E_i = \frac{n}{2} \sum_{q \in \{r,g,b\}} \frac{\left(\hat{c}_{i,q} - c_{i,q}\right)^2}{c_{i,q}^2 + \varepsilon}$$

One Hidden-layer model

Relative Mean Square Error:

$$E_i = \frac{n}{2} \sum_{q \in \{r,g,b\}} \frac{\left(\hat{c}_{i,q} - c_{i,q}\right)^2}{c_{i,q}^2 + \varepsilon}$$

$$\frac{\partial E_i}{\partial w_{t,s}^l} = \sum_{m=1}^{M} \left[\sum_{q \in \{r,g,b\}} \left[\frac{\partial E_{i,q}}{\partial \hat{c}_{i,q}} \frac{\partial \hat{c}_{i,q}}{\partial \theta_{m,i}} \right] \frac{\partial \theta_{m,i}}{\partial w_{t,s}^l} \right]$$

$$\frac{\partial E_i}{\partial \hat{c}_{i,q}} = ???$$

One Hidden-layer model

Relative Mean Square Error:

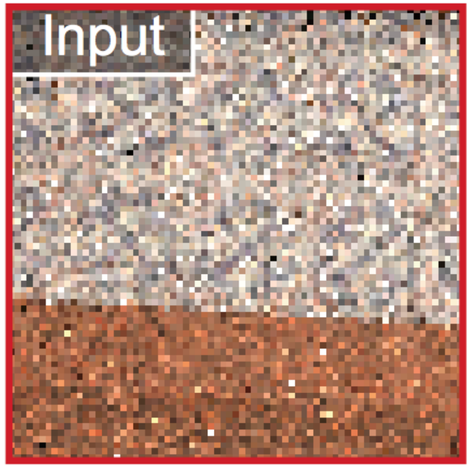
$$E_i = \frac{n}{2} \sum_{q \in \{r,g,b\}} \frac{\left(\hat{c}_{i,q} - c_{i,q}\right)^2}{c_{i,q}^2 + \varepsilon}$$

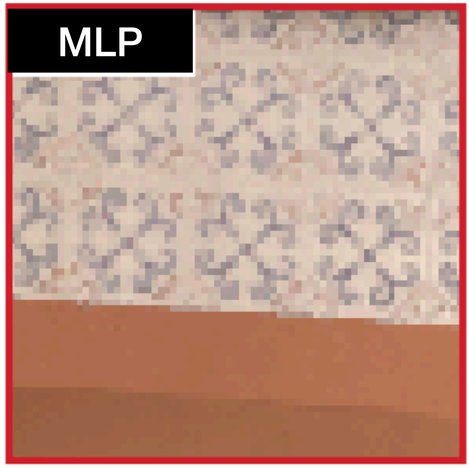
$$\frac{\partial E_i}{\partial w_{t,s}^l} = \sum_{m=1}^{M} \left[\sum_{q \in \{r,g,b\}} \left[\frac{\partial E_{i,q}}{\partial \hat{c}_{i,q}} \frac{\partial \hat{c}_{i,q}}{\partial \theta_{m,i}} \right] \frac{\partial \theta_{m,i}}{\partial w_{t,s}^l} \right]$$

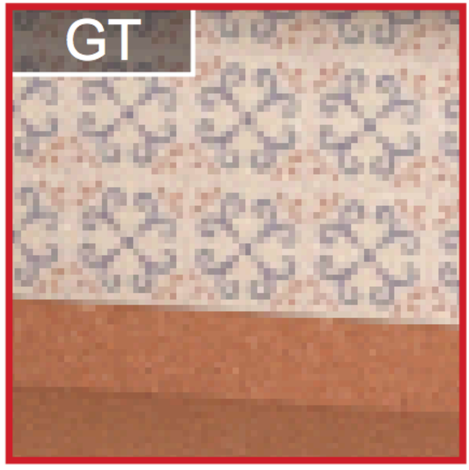
$$\frac{\partial E_i}{\partial \hat{c}_{i,q}} = n \frac{\hat{c}_{i,q} - c_{i,q}}{c_{i,q}^2 + \epsilon}$$

Results

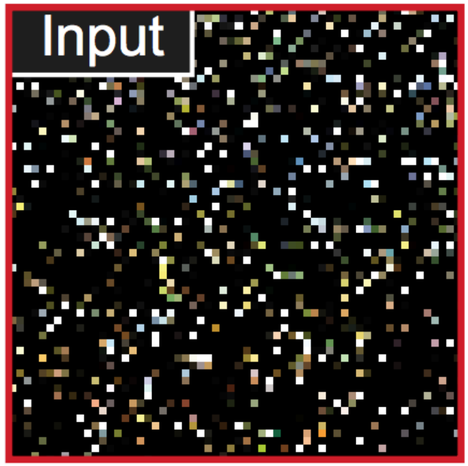

4spp

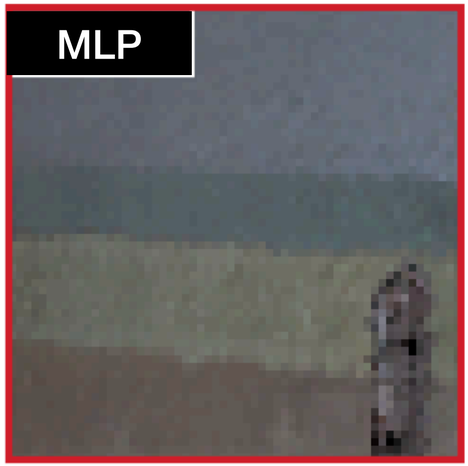

Input


MLP


GT


4spp


Input


MLP


GT

Introduction to CNNs

Introduction to CNNs

Kernel Predicting Denoising

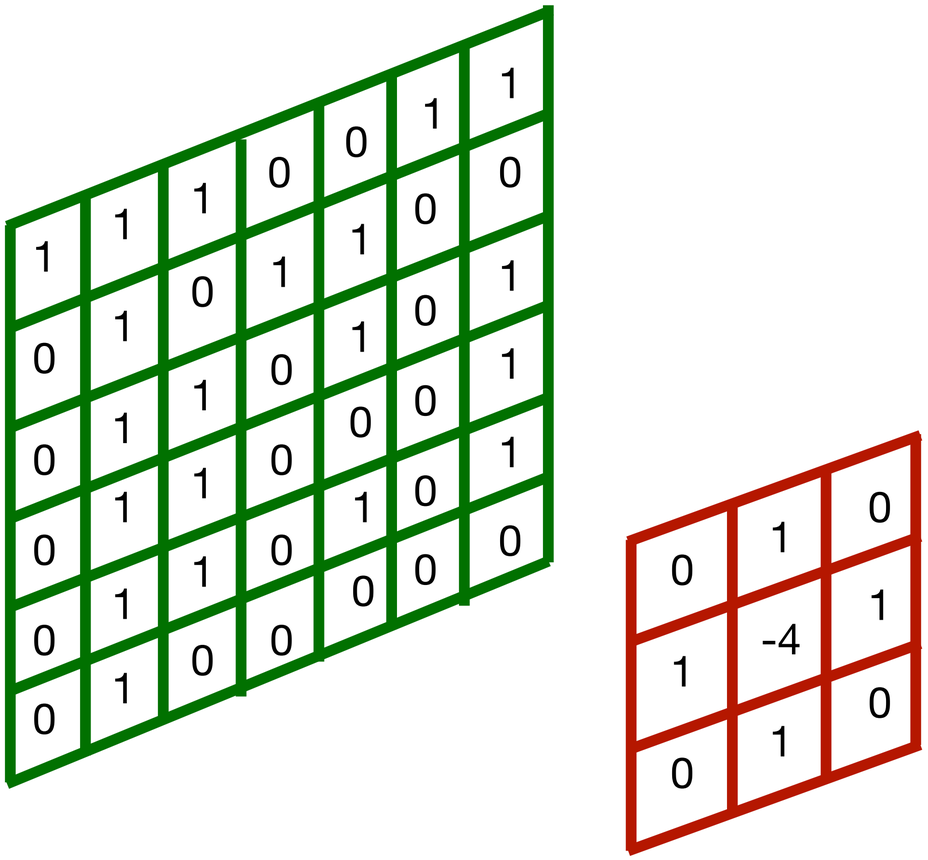
Introduction to CNNs

Kernel Predicting Denoising

Neural Importance Sampling

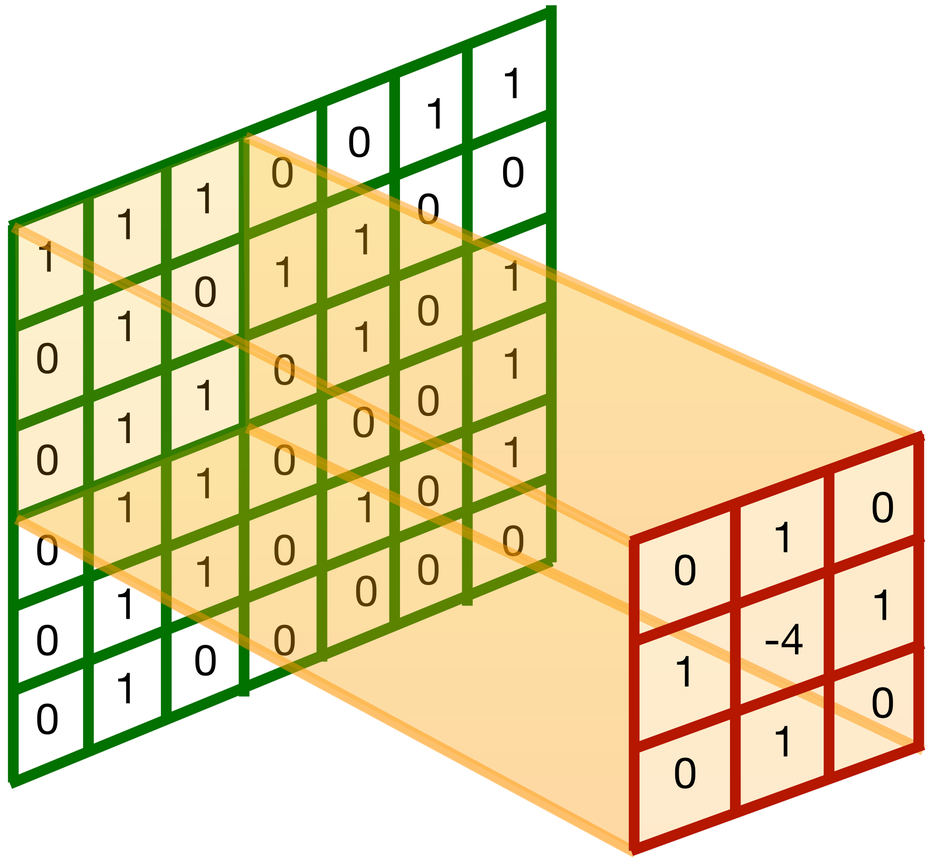
Introduction to CNNs

Convolution



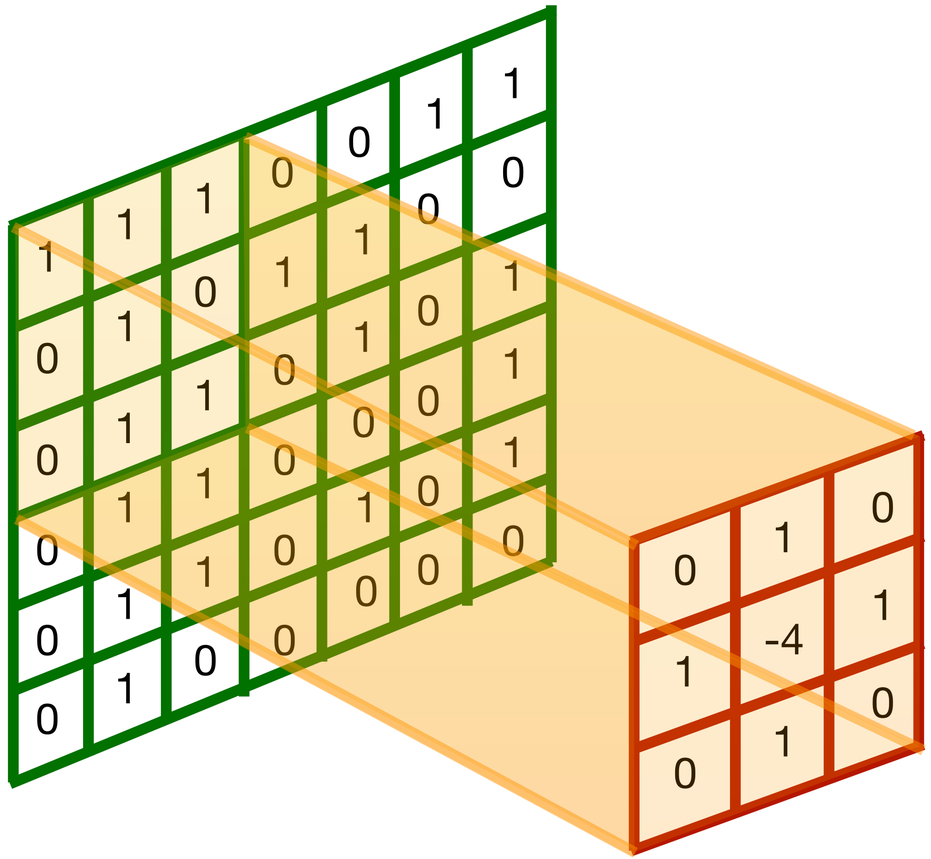
No zero padding

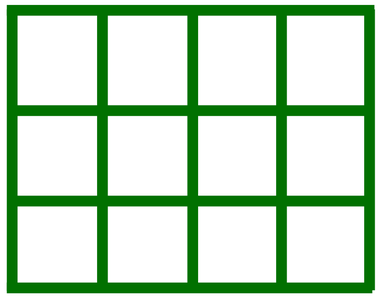
Convolution



No zero padding

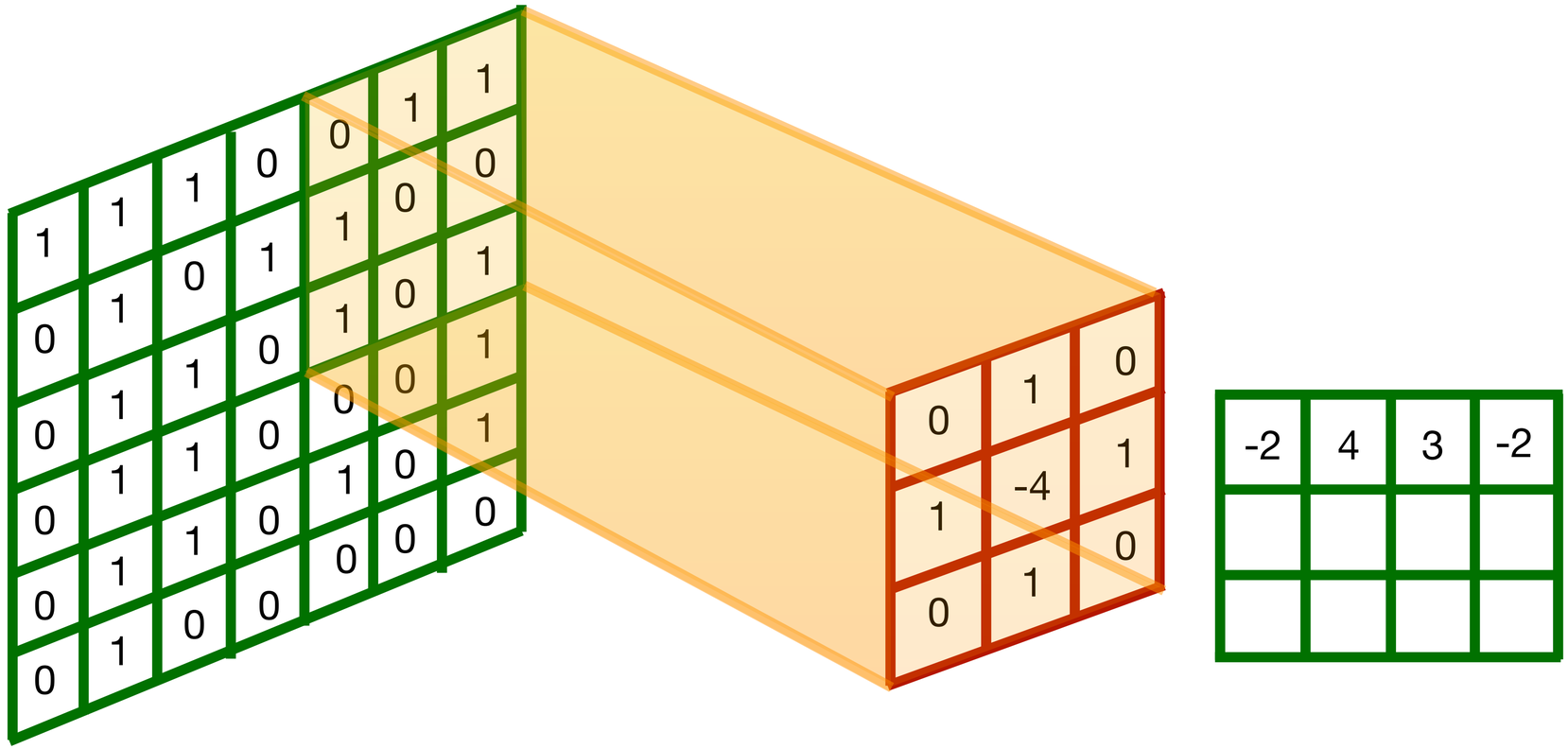
Stride-1 Convolution





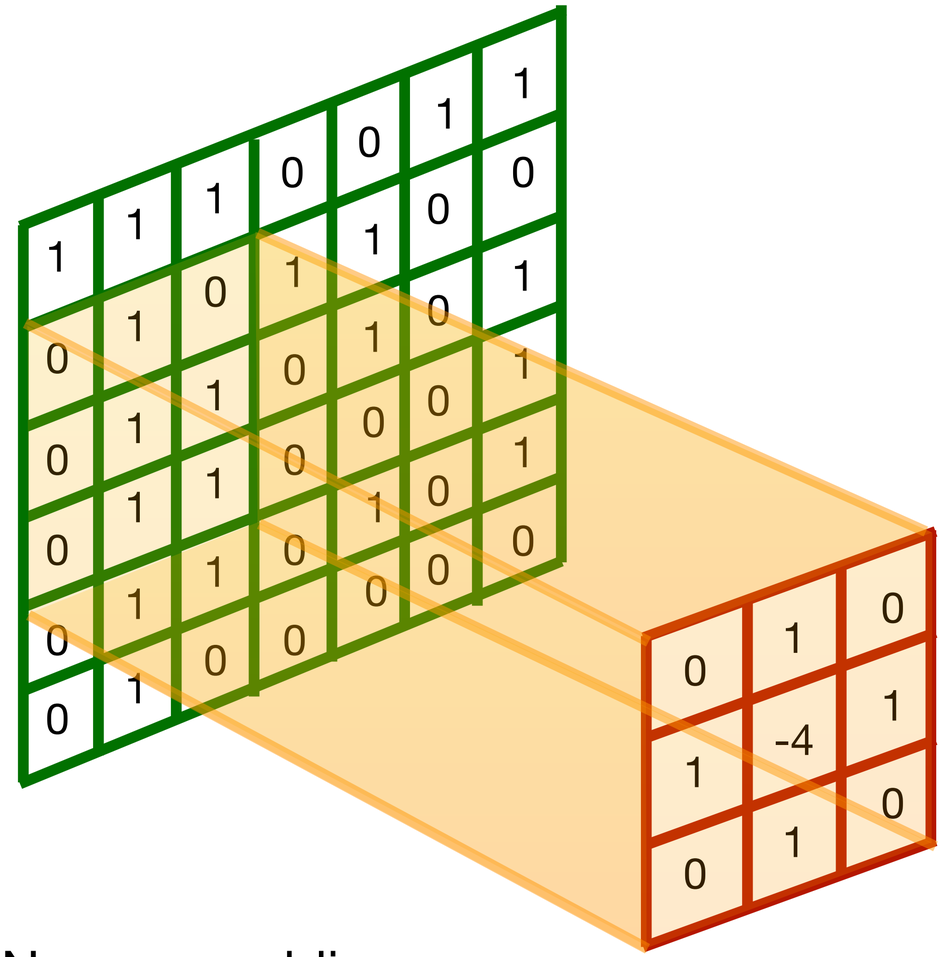
No zero padding

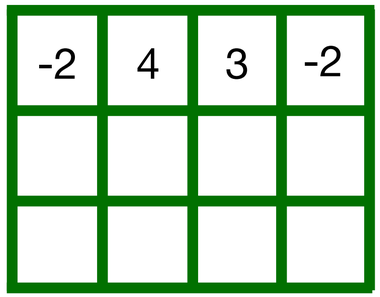
Stride-1 Convolution



No zero padding

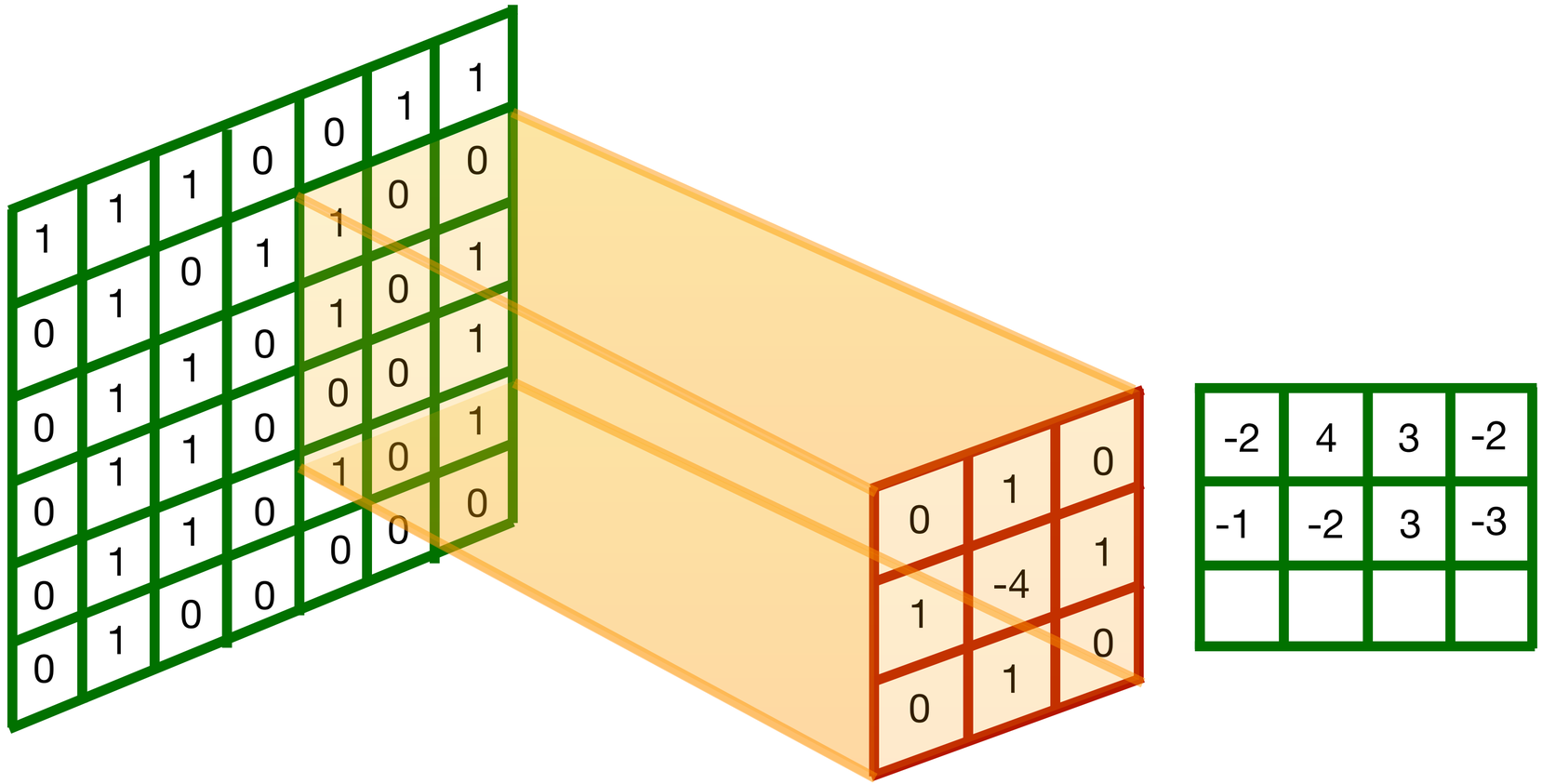
Stride-1 Convolution





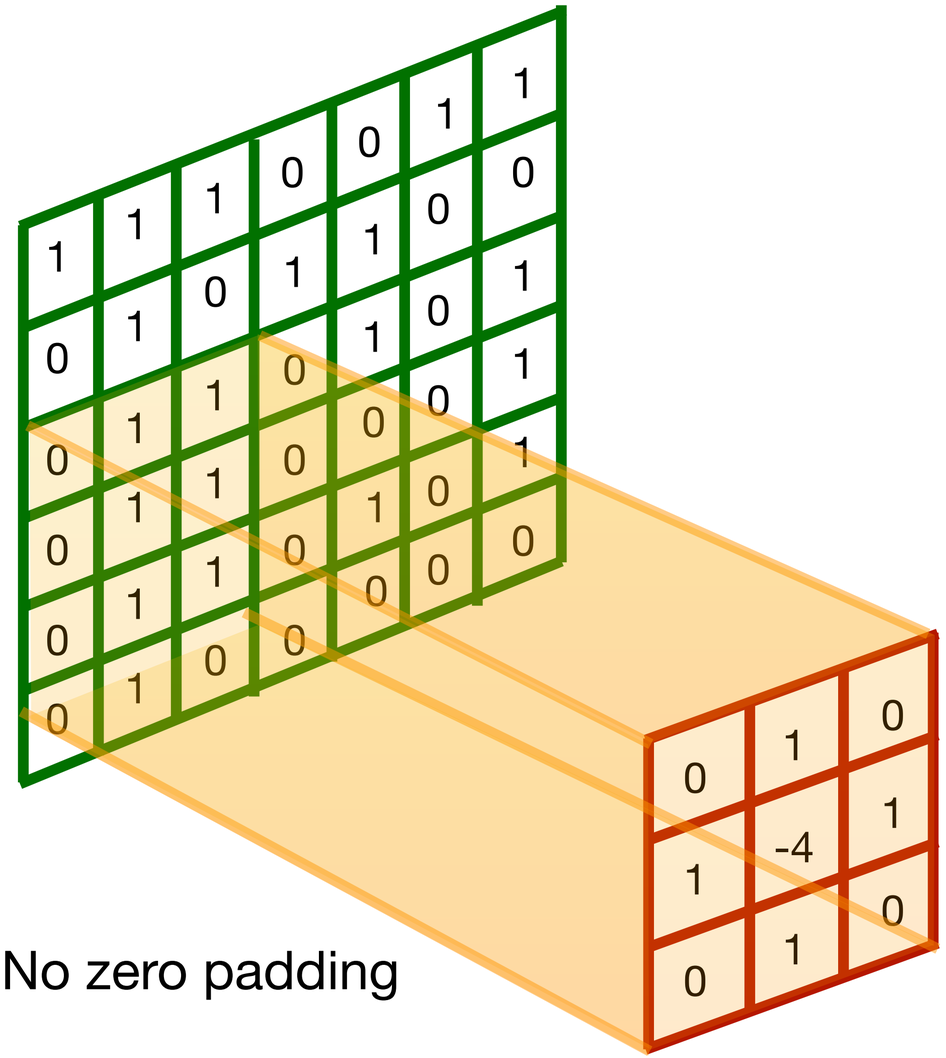
No zero padding

Stride-1 Convolution



No zero padding

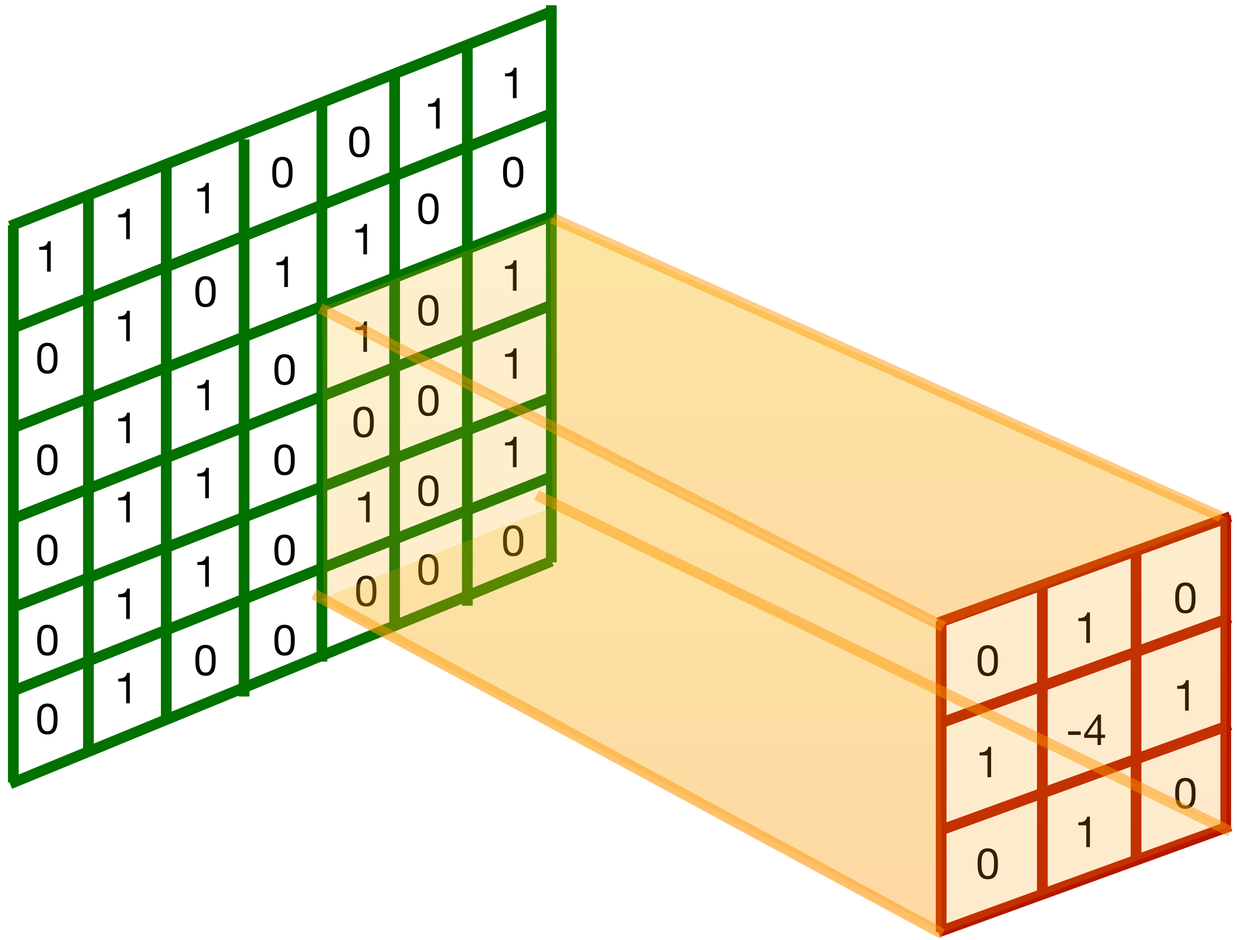
Stride-1 Convolution



-2	4	3	-2
-1	-2	3	-3

0

Stride-1 Convolution

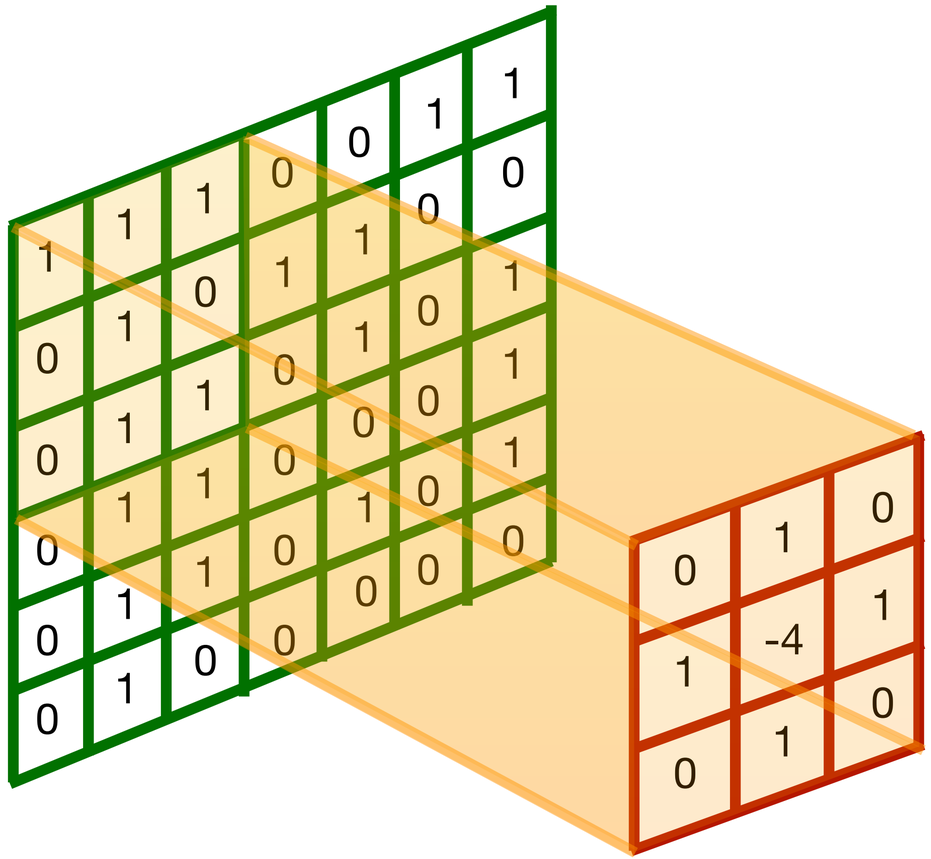


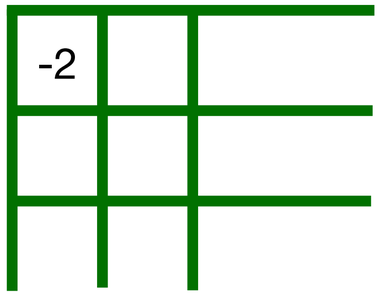
-2	4	3	-2
-1	-2	3	-3
-1	-2	1	1

No zero padding

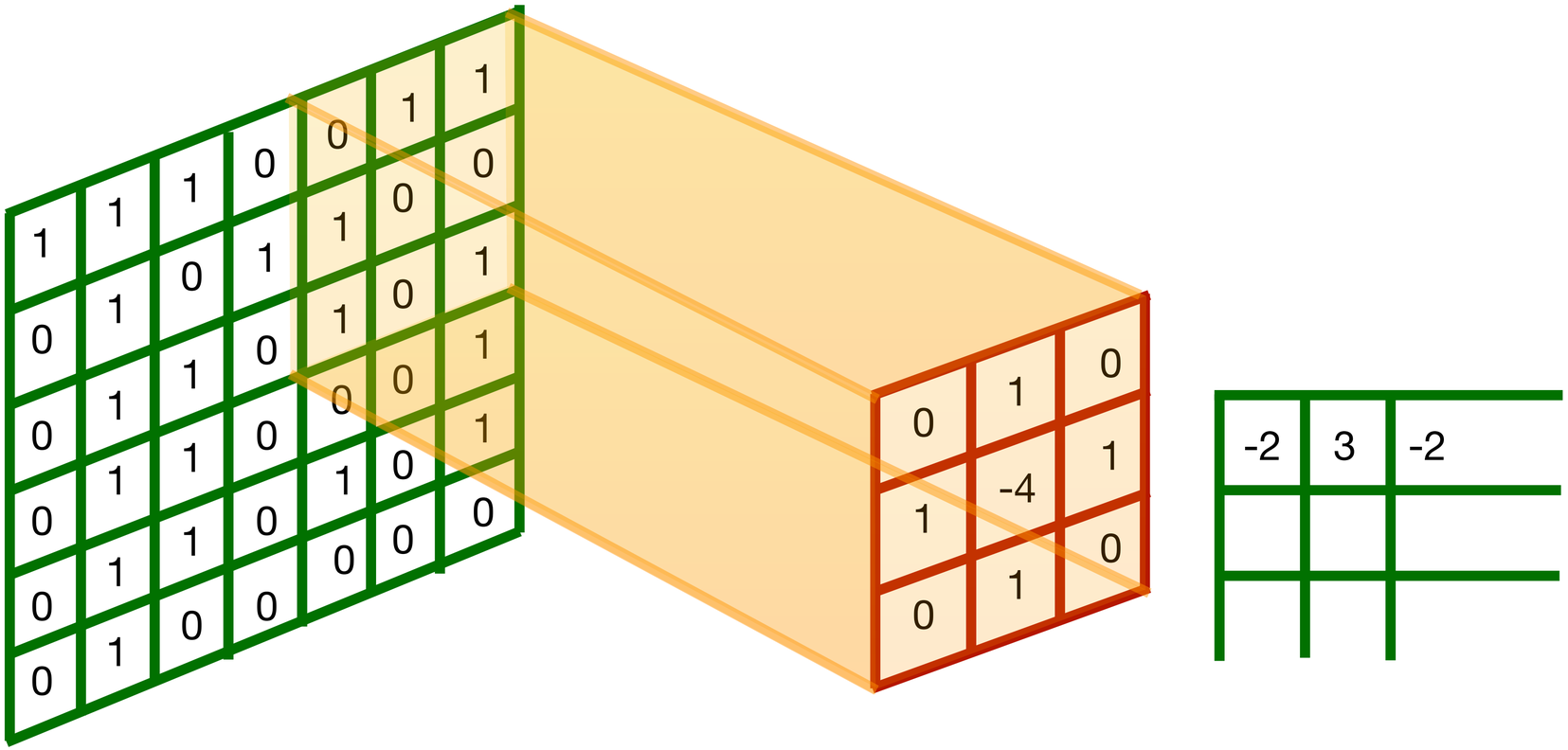
0

Stride-2 Convolution



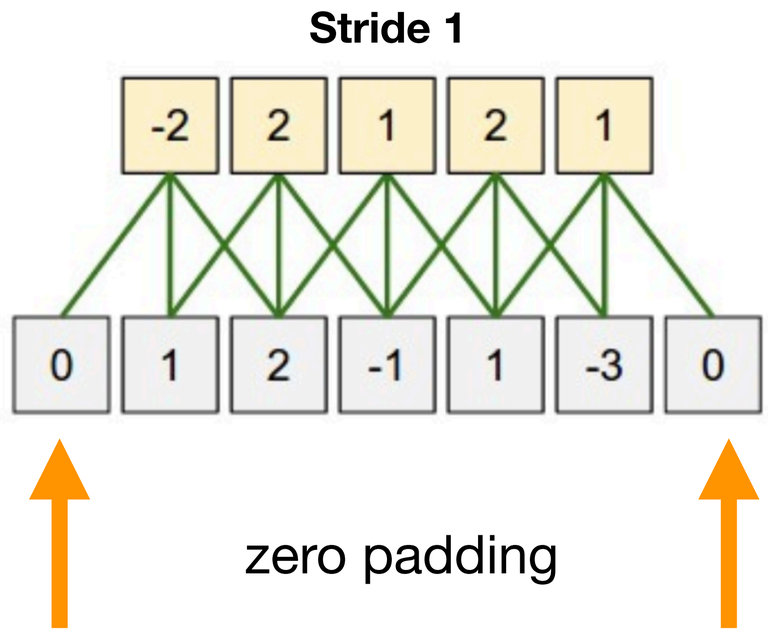


Stride-2 Convolution



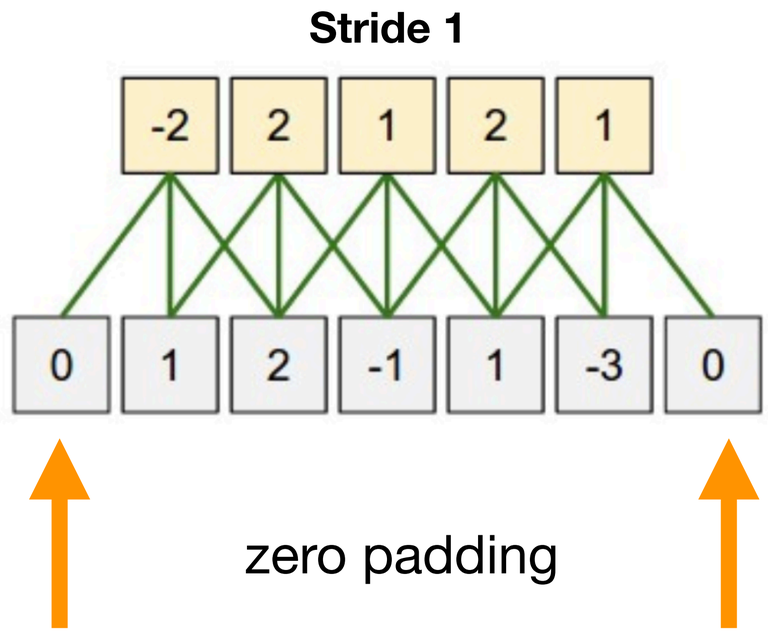
Zero Padding and Strides

1D image to illustrate the strides and zero padding



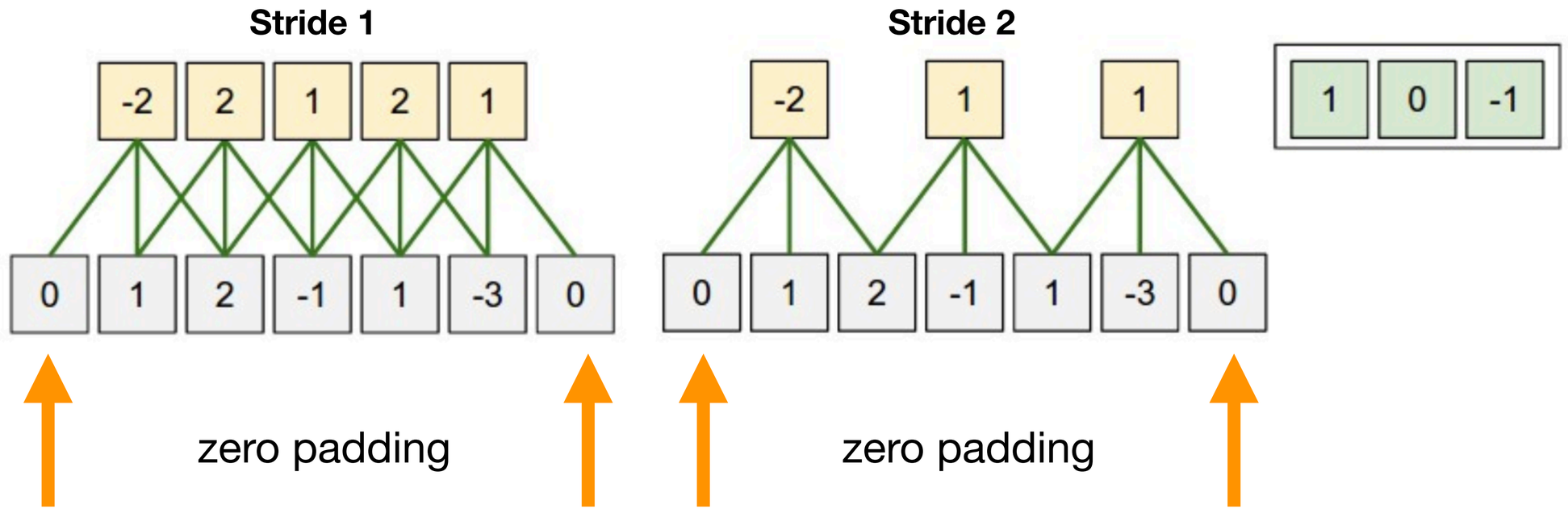
Zero Padding and Strides

1D image to illustrate the strides and zero padding

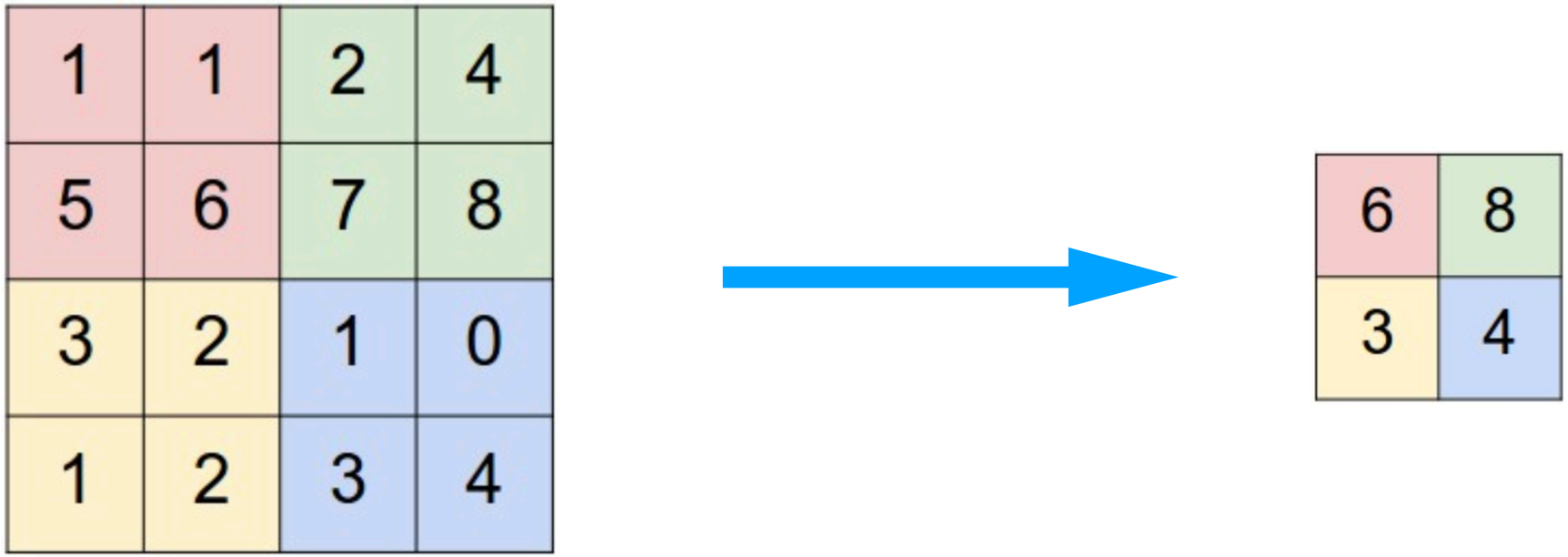


Strides

1D image to illustrate the strides and zero padding



Max Pooling / Down Sampling



Overview on Convolutional Neural Networks

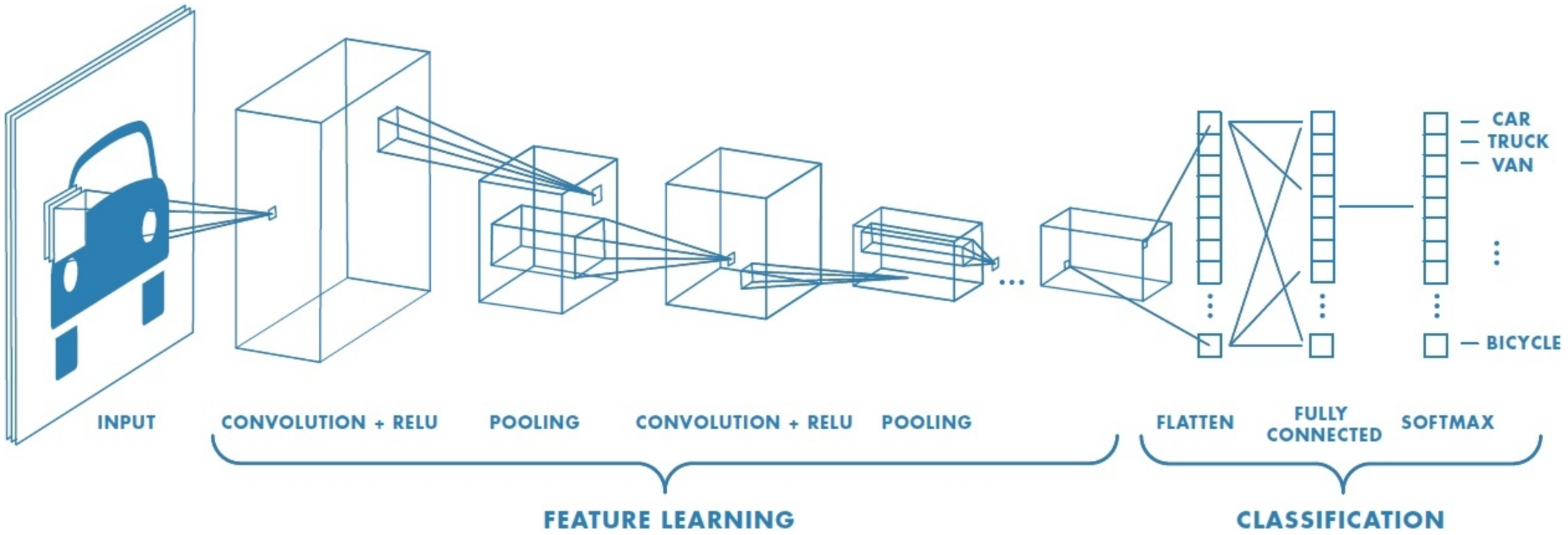
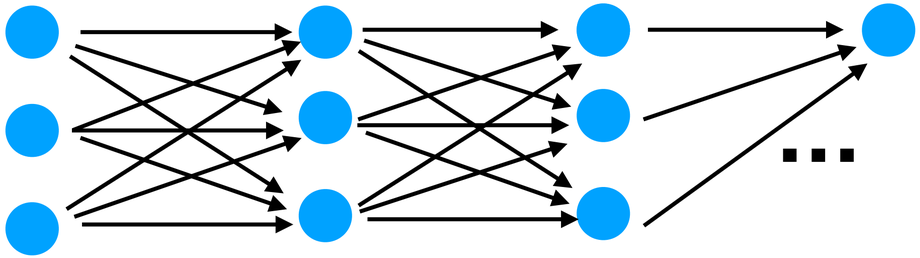


Image Courtesy: Mathworks (online tutorial)

Multi-layer Perceptron vs. CNNs

Multi-layer Perceptron vs. CNNs

Multi-layer perceptron

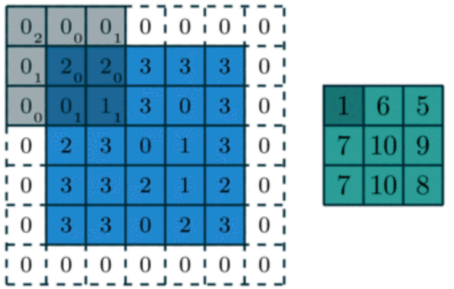


All nodes are fully connected in all layers

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

Introduction to CNNs

Kernel-Predicting Denoising

Kernel-Predicting Networks for Denoising Monte-Carlo Renderings

Bako et al. [2017]

Limitations of MLP based Denoiser

Kernel was pre-selected to be joint bilateral filter

Limitations of MLP based Denoiser

Kernel was pre-selected to be joint bilateral filter

- Unable to explicitly capture all details

Limitations of MLP based Denoiser

Kernel was pre-selected to be joint bilateral filter

- Unable to explicitly capture all details
- lacked flexibility to handle wide range of MC noise in production scenes

Limitations of MLP based Denoiser

Kernel was pre-selected to be joint bilateral filter

- Unable to explicitly capture all details
- lacked flexibility to handle wide range of MC noise in production scenes

Fixed

Limitations of MLP based Denoiser

Kernel was pre-selected to be joint bilateral filter

- Unable to explicitly capture all details
- lacked flexibility to handle wide range of MC noise in production scenes

Fixed

- can cause unstable weights causing bright ringing and color artifacts

Limitations of MLP based Denoiser

Kernel was pre-selected to be joint bilateral filter

- Unable to explicitly capture all details
- lacked flexibility to handle wide range of MC noise in production scenes

Fixed

- can cause unstable weights causing bright ringing and color artifacts

Too many parameters to optimize

Requirements

The function must be flexible to capture complex relationship between input data and reference colors over wide range of scenarios.

Choice of loss function is crucial. Should capture perceptual aspects of the scene.

To avoid overfitting, large dataset required

Using a Vanilla CNN

Denoising a raw, noisy color buffer causes overblurring

Using a Vanilla CNN

Denoising a raw, noisy color buffer causes overblurring

- difficulty in distinguishing scene details and MC noise

Using a Vanilla CNN

Denoising a raw, noisy color buffer causes overblurring

- difficulty in distinguishing scene details and MC noise

High dynamic range

Using a Vanilla CNN

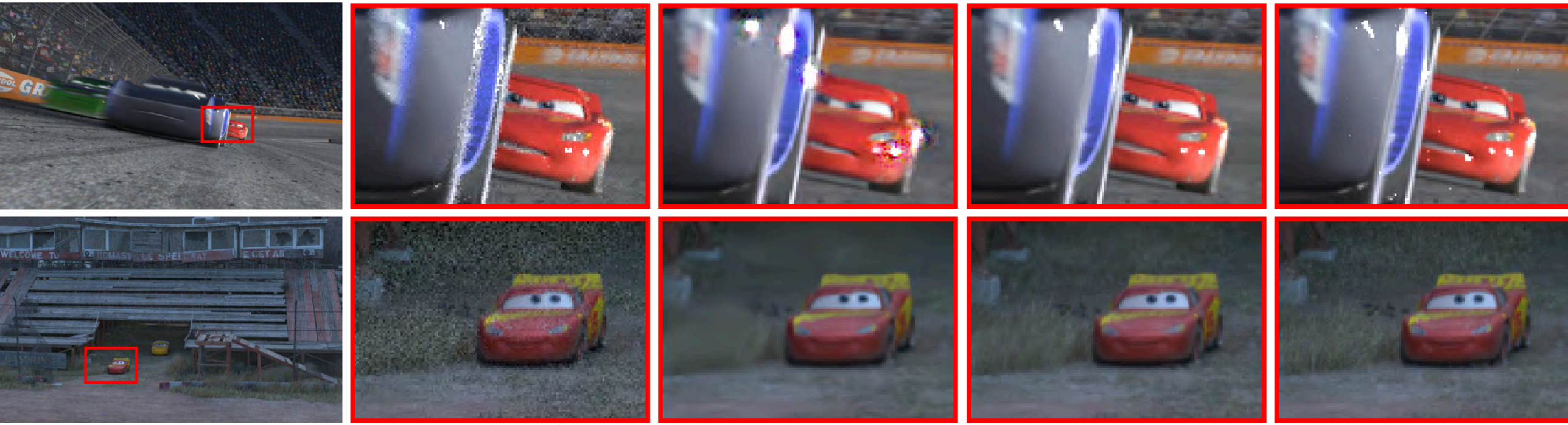
Denoising a raw, noisy color buffer causes overblurring

- difficulty in distinguishing scene details and MC noise

High dynamic range

- can cause unstable weights causing bright ringing and color artifacts

Vanilla CNN



Ours | Input (32 spp) | Vanilla CNN | Ours | Ref. (1K spp)

Denoising Model

$$\widehat{\boldsymbol{\theta}}_p = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, \ell(\overline{\mathbf{c}}_p, g(\mathbf{X}_p; \boldsymbol{\theta}))$$

Denoised function with parameters

Reference image

$$\widehat{\mathbf{c}}_p = g(\mathbf{X}_p; \widehat{\boldsymbol{\theta}}_p)$$

Denoised value

$$\ell(\overline{\mathbf{c}}, \widehat{\mathbf{c}})$$

Loss function

Computational Model

$$\widehat{\boldsymbol{\theta}}_p = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{q \in \mathcal{N}(p)} \left(\mathbf{c}_q - \boldsymbol{\theta}^\top \phi(\mathbf{x}_q)\right)^2 \omega(\mathbf{x}_p, \mathbf{x}_q)$$

Neighborhood

$$\widehat{\mathbf{c}}_p = g(\mathbf{X}_p; \widehat{\boldsymbol{\theta}}_p)$$

Denoised value

$$\phi : \mathbb{R}^{3+\bar{D}} \rightarrow \mathbb{R}^{\bar{M}}$$

$\omega(\mathbf{x}_p, \mathbf{x}_q)$ Kernel weights

$$\widehat{\mathbf{c}}_p = \widehat{\boldsymbol{\theta}}_p^\top \phi(\mathbf{x}_p)$$

Final denoised value

Direct Prediction Network

Direct prediction convolution network: outputs denoised image

$$\widehat{\mathbf{c}}_p = g_{\text{direct}}(\mathbf{X}_p; \boldsymbol{\theta}) = \mathbf{z}_p^L$$

Direct Prediction Network

Direct prediction convolution network: outputs denoised image

$$\widehat{\mathbf{c}}_p = g_{\text{direct}}(\mathbf{X}_p; \boldsymbol{\theta}) = \mathbf{z}_p^L$$

Issues:

The constrained nature and complexity of the problem makes optimization difficult.

The magnitude and variance of stochastic gradients computed during training can be large, which slows convergence of training loss.

Kernel Prediction Network

Kernel prediction convolution network: outputs learned kernel weights

$$w_{pq} = \frac{\exp([\mathbf{z}_p^L]_q)}{\sum_{q' \in \mathcal{N}(p)} \exp([\mathbf{z}_p^L]_{q'})}$$

$$0 \leq w_{pq} \leq 1$$

Softmax activation to enforce weights within range

Denoised color values:

$$\widehat{\mathbf{c}}_p = g_{\text{weighted}}(\mathbf{X}_p; \boldsymbol{\theta}) = \sum_{q \in \mathcal{N}(p)} \mathbf{c}_q w_{pq}$$

Kernel Prediction Network

$$w_{pq} = \frac{\exp([\mathbf{z}_p^L]_q)}{\sum_{q' \in \mathcal{N}(p)} \exp([\mathbf{z}_p^L]_{q'})}$$

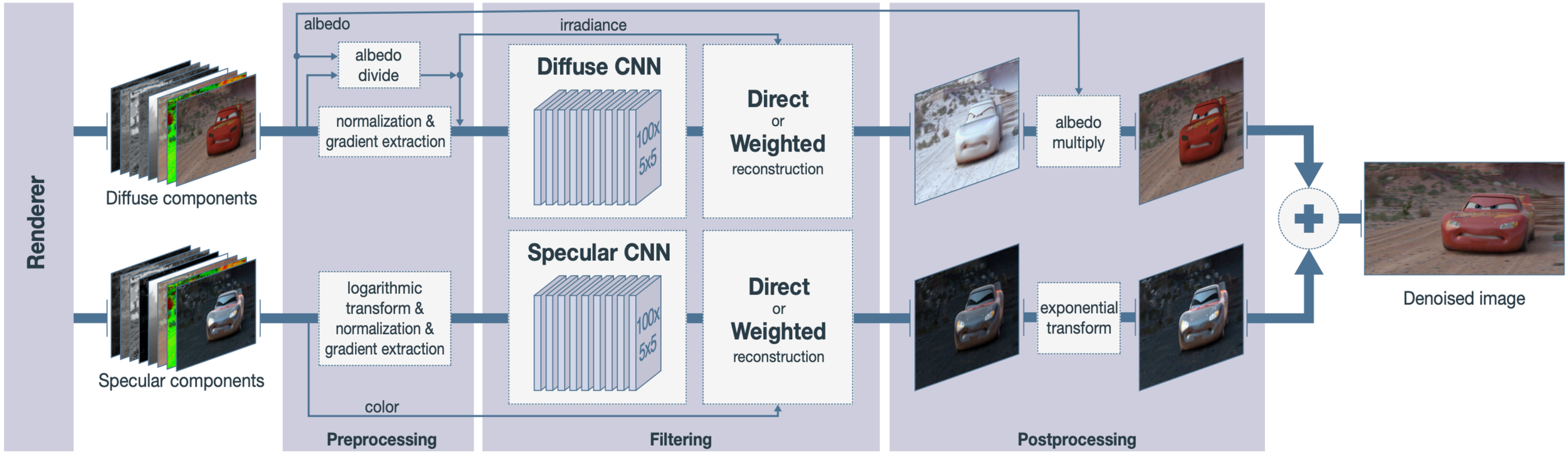
$$0 \leq w_{pq} \leq 1$$

$$\widehat{\mathbf{c}}_p = g_{\text{weighted}}(\mathbf{X}_p; \boldsymbol{\theta}) = \sum_{q \in \mathcal{N}(p)} \mathbf{c}_q w_{pq}$$

Final color estimate always lies within the convex hull of the respective neighborhood (avoid color shifts).

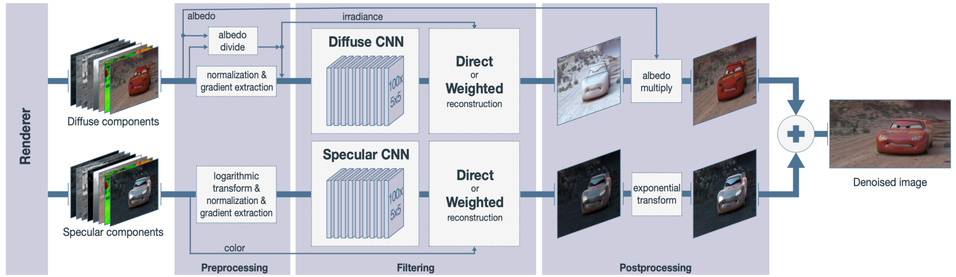
Ensures well-behaved gradients of the error w.r.t the kernel weights

Proposed Architecture



Diffuse/Specular components

Each component is denoised separately

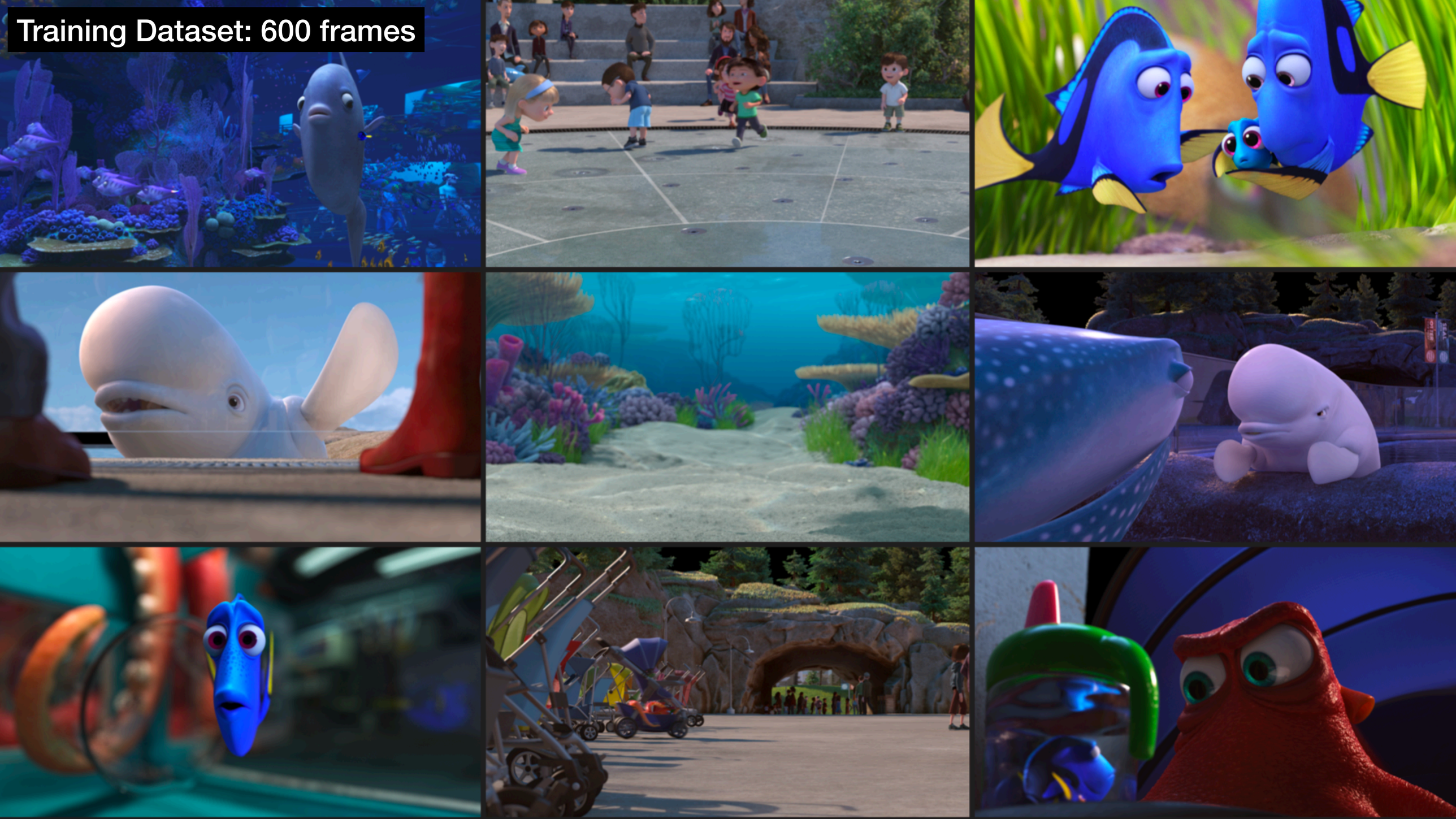


Diffuse components are well-behaved and typically has small ranges

- albedo is factored out to allow large range kernels $\tilde{\mathbf{c}}_{\text{diffuse}} = \mathbf{c}_{\text{diffuse}} \oslash (\mathbf{f}_{\text{albedo}} + \epsilon)$

Specular components are challenging due to high dynamic ranges: uses logarithmic transform

$$\tilde{\mathbf{c}}_{\text{specular}} = \log(1 + \mathbf{c}_{\text{specular}})$$


Training Dataset: 600 frames

Training

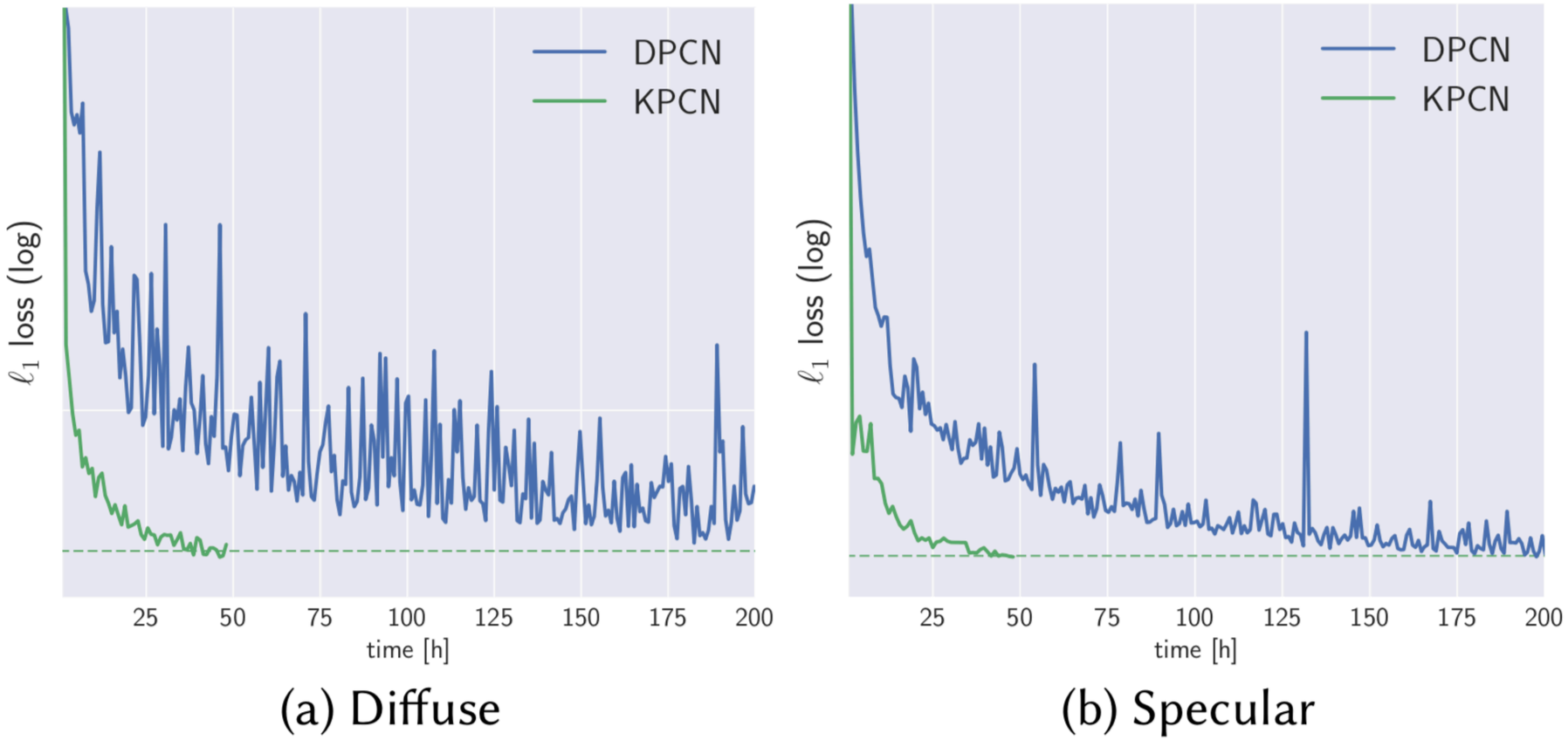
8-hidden layers used with 100 kernels of 5x5 in each layer for each network

For KPCN (kernel-predicting network), output kernel size used = 21

Weights for 128 spp and 32 spp networks were initialized using Xavier method

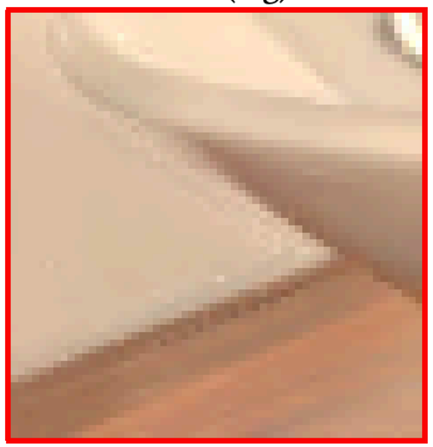
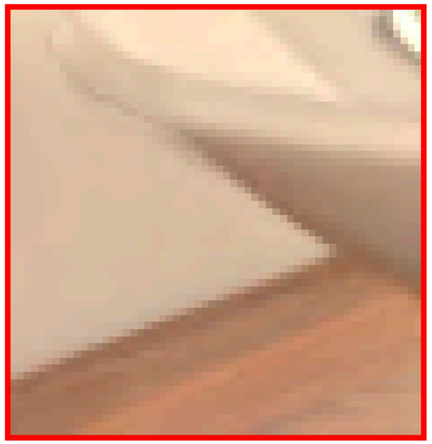
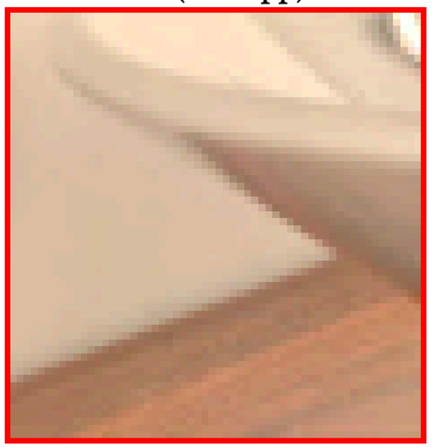
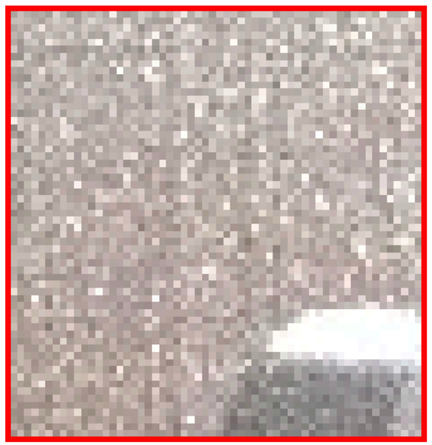
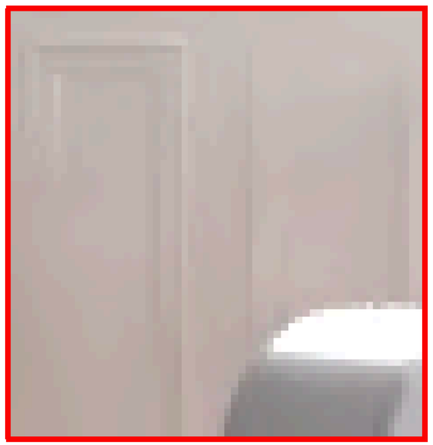
Diffuse and specular components were independently trained with L1 loss metric

Learning rate of DPCN vs. KPCN

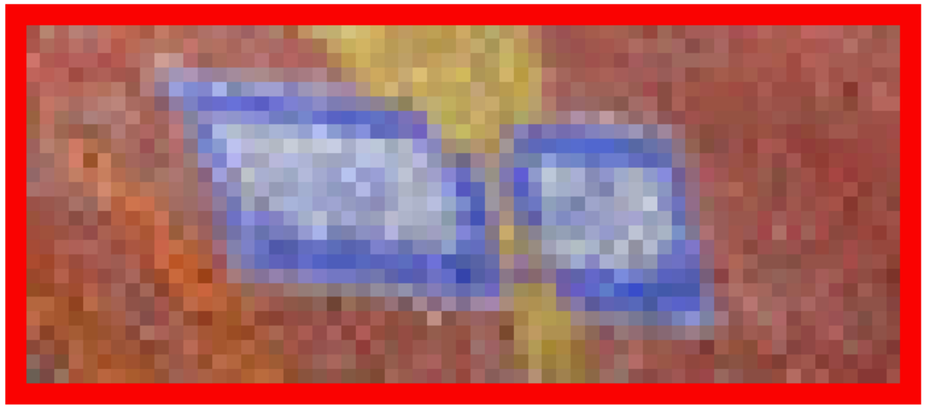


On Cars 3 dataset, KPCN converges 5-6x faster

Results

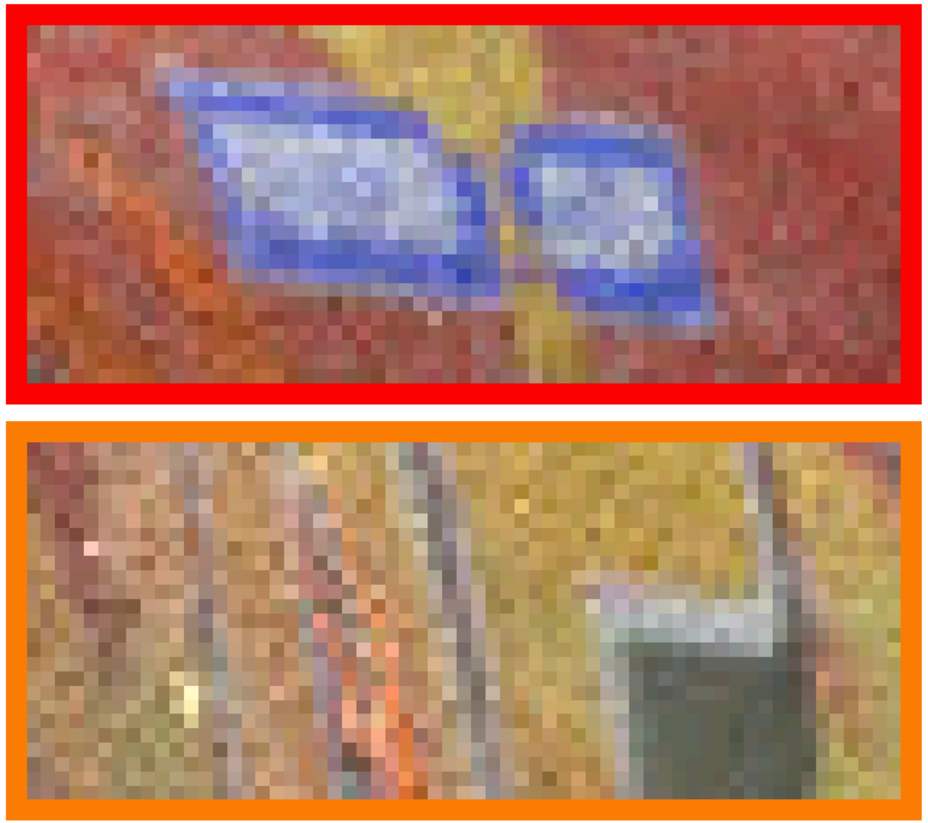
Ours	Input (128 spp)	NFOR (log)	Ours	Ref. (32K spp)
				
relative ℓ_2 $1-$ SSIM	29.15e-3 0.562	0.90e-3 0.019	0.69e-3 0.017	
				
relative ℓ_2 $1-$ SSIM	38.57e-3 0.552	1.12e-3 0.025	0.92e-3 0.024	

Ablation study



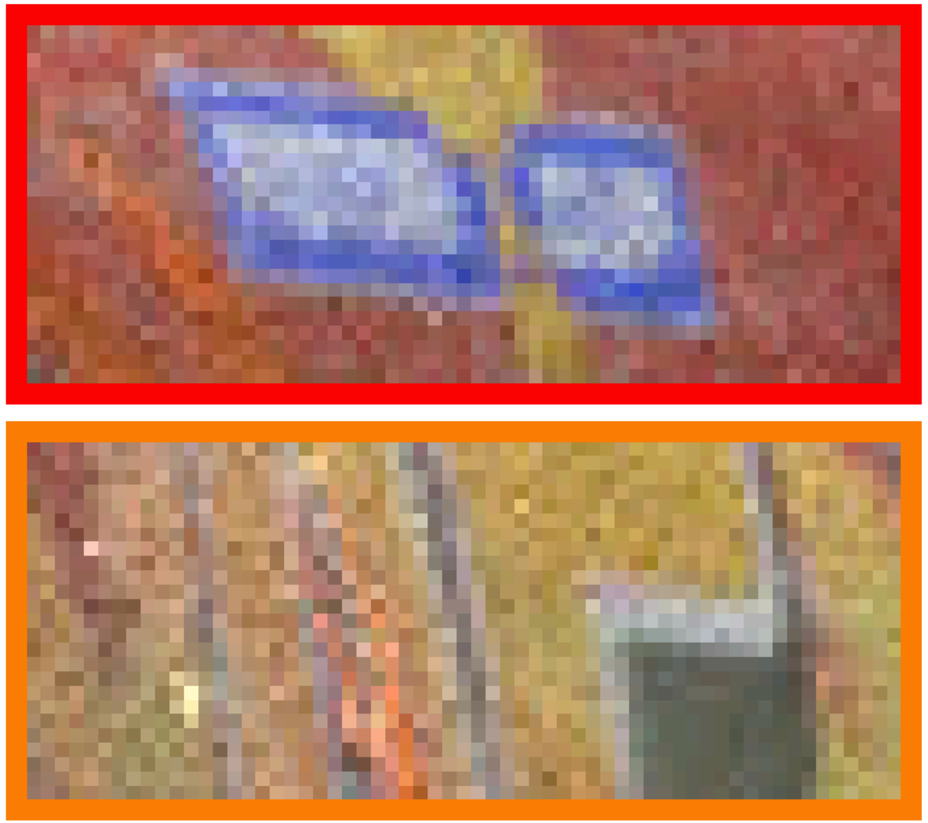
Input (32 spp)

Ablation study



Input (32 spp)

Ablation study



Input (32 spp)

w/o Decomposition,
w/o Albedo divide

Ablation study

w/o Decomposition, w/o Albedo divide

w/ Decomposition, w/o Albedo divide

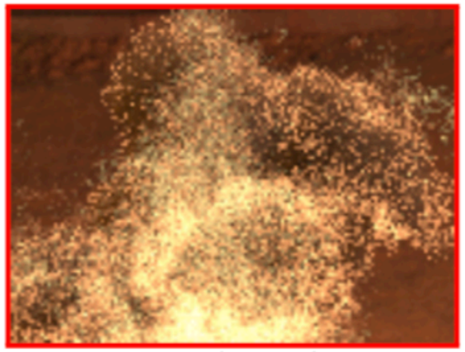
w/o Decomposition, w/ Albedo divide

w/ Decomposition, w/ Albedo divide

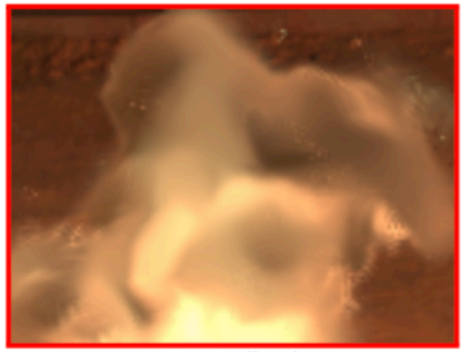
Ref. (2K spp)

Ablation study

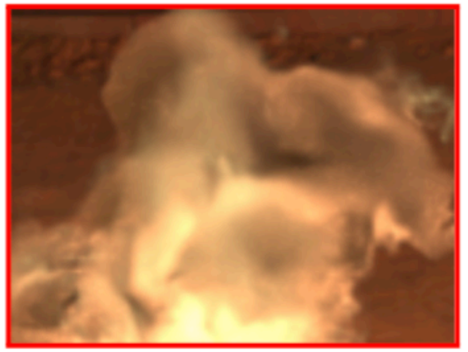
Ours



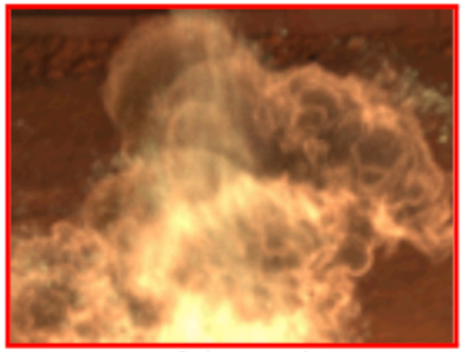
Input (32 spp)



NFOR (log)



Ours



Ref. (8K spp)

Ours

Input (32 spp)

Ours

Ref. (1K spp)

Also works on Piper short movie frames

Interactive Reconstruction of Monte Carlo Sequences

Chaitanya et al. [2017]

Toy Story (Pixar)

Pacific Rim (ILM)

Crisis 3 (Crytek)

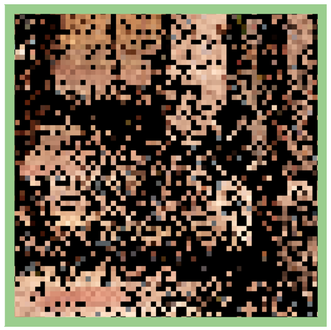
Halo 3 (Bungie)

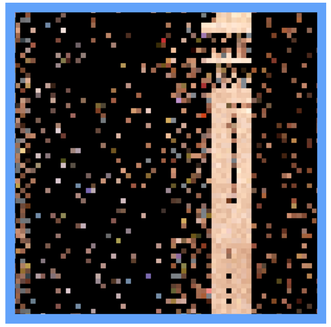
Motivation: Interactive Reconstruction

Limited to a few rays per pixel @ 1080p @ 30Hz

Never enough to reconstruct an image

Deep learning approach for interactive graphics



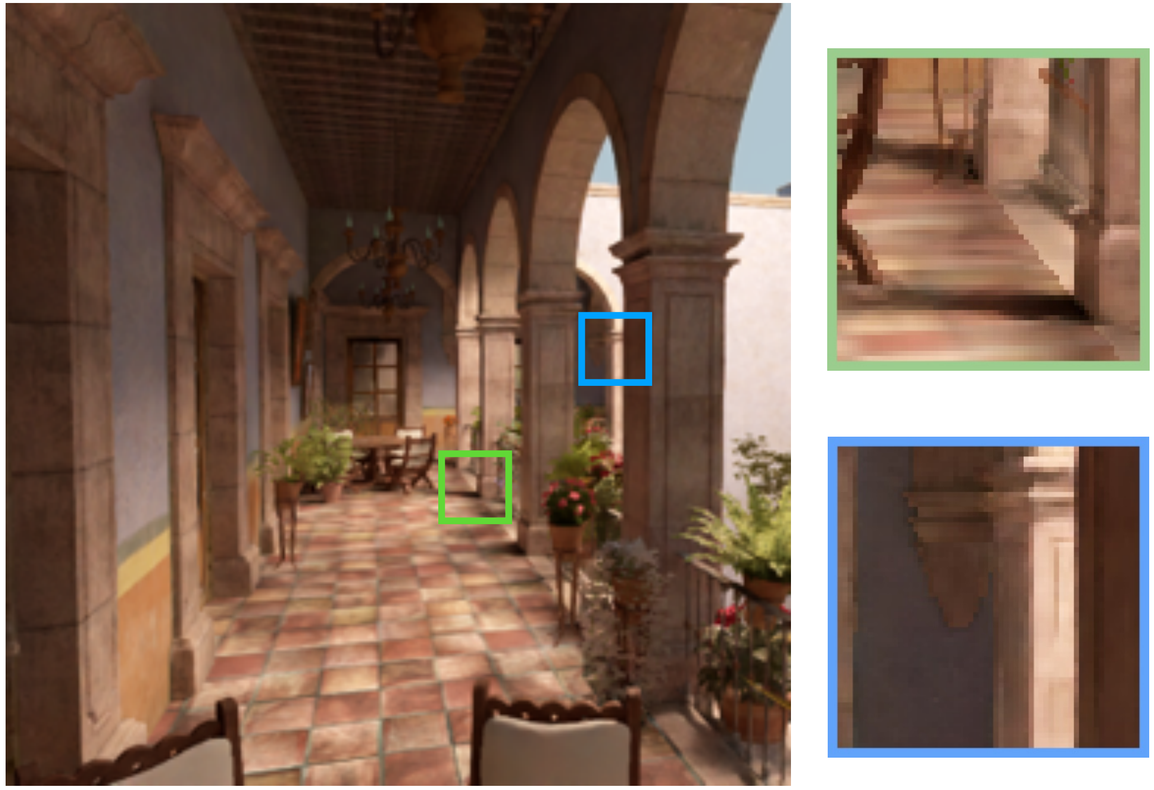


Motivation: Interactive Reconstruction

Limited to a few rays per pixel @ 1080p @ 30Hz

Never enough to reconstruct an image

Deep learning approach for interactive graphics



Problem Statement

Handle generic effects:

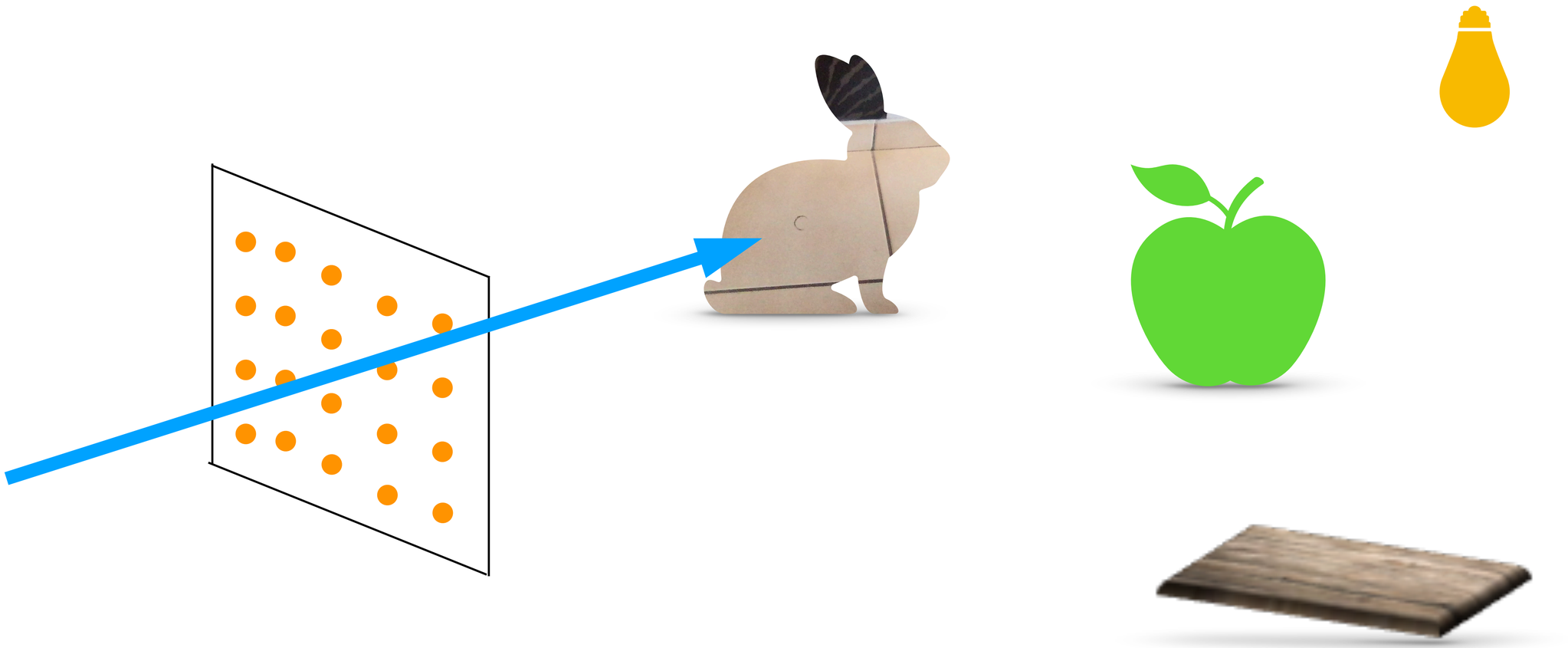
- Soft shadows

- Diffuse and specular reflections

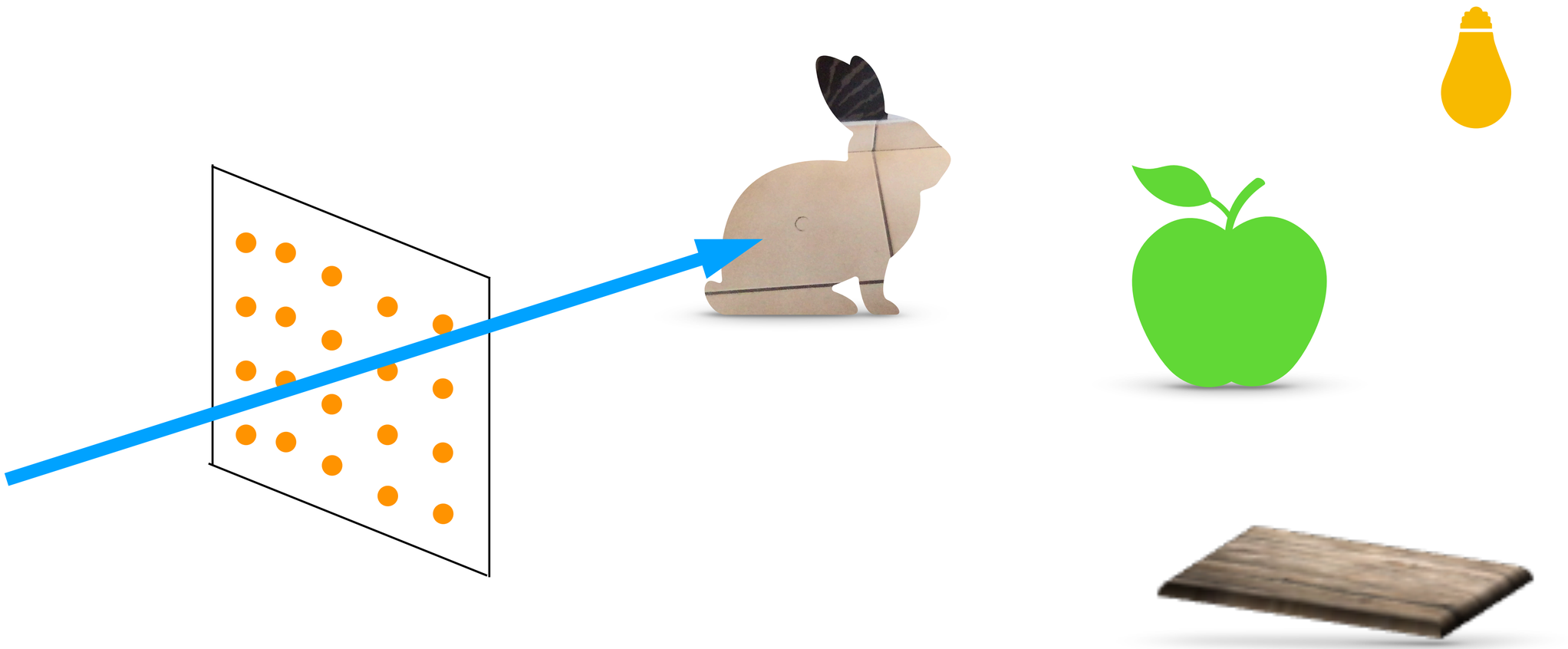
- Global illumination (one-bounce)

- No Motion blur or depth of field

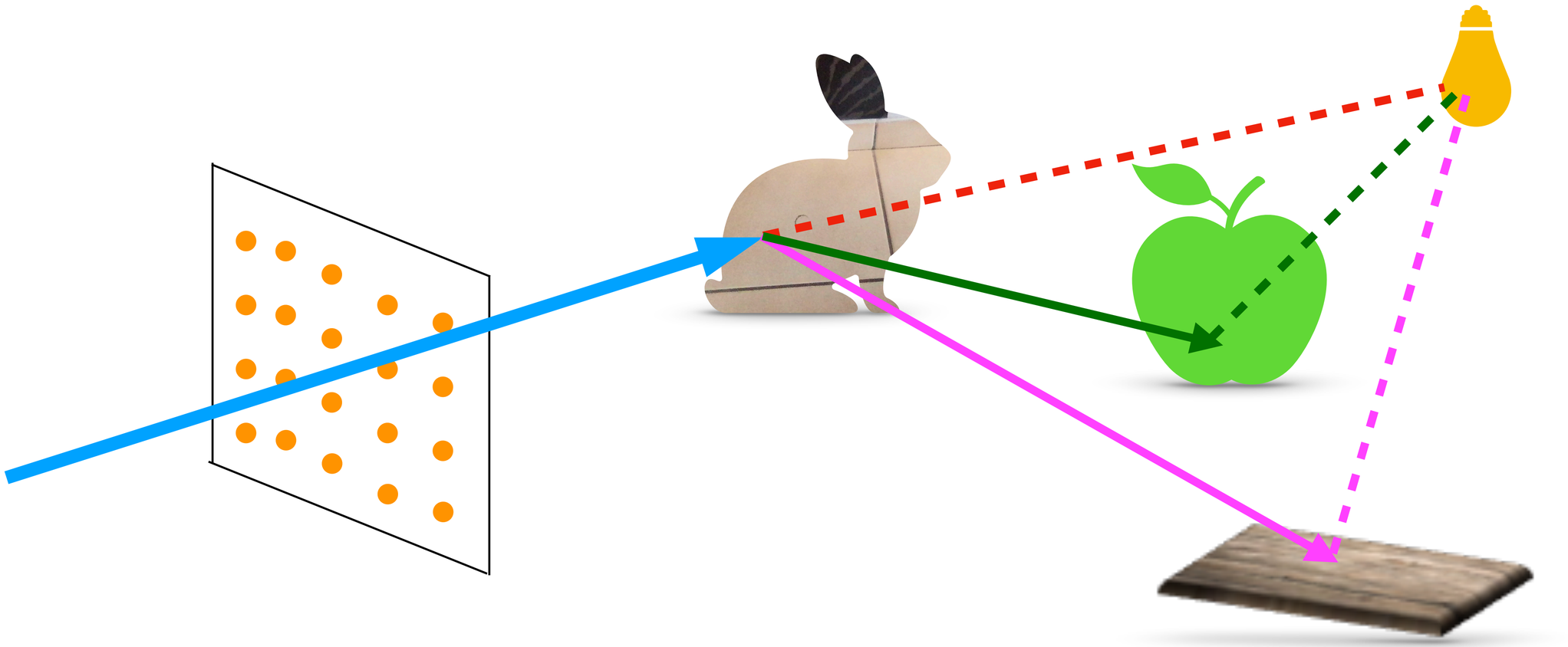
System setup: Path tracing



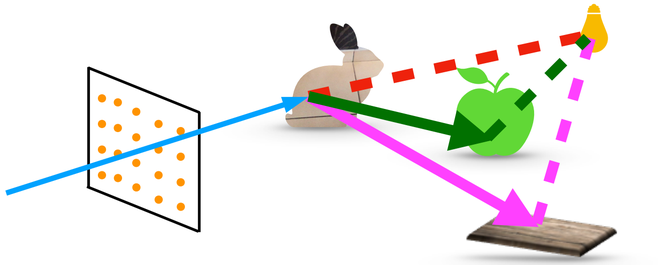
System setup: Path tracing



System setup: Path tracing



System setup: Path tracing



Rasterize primary hits in G-buffers

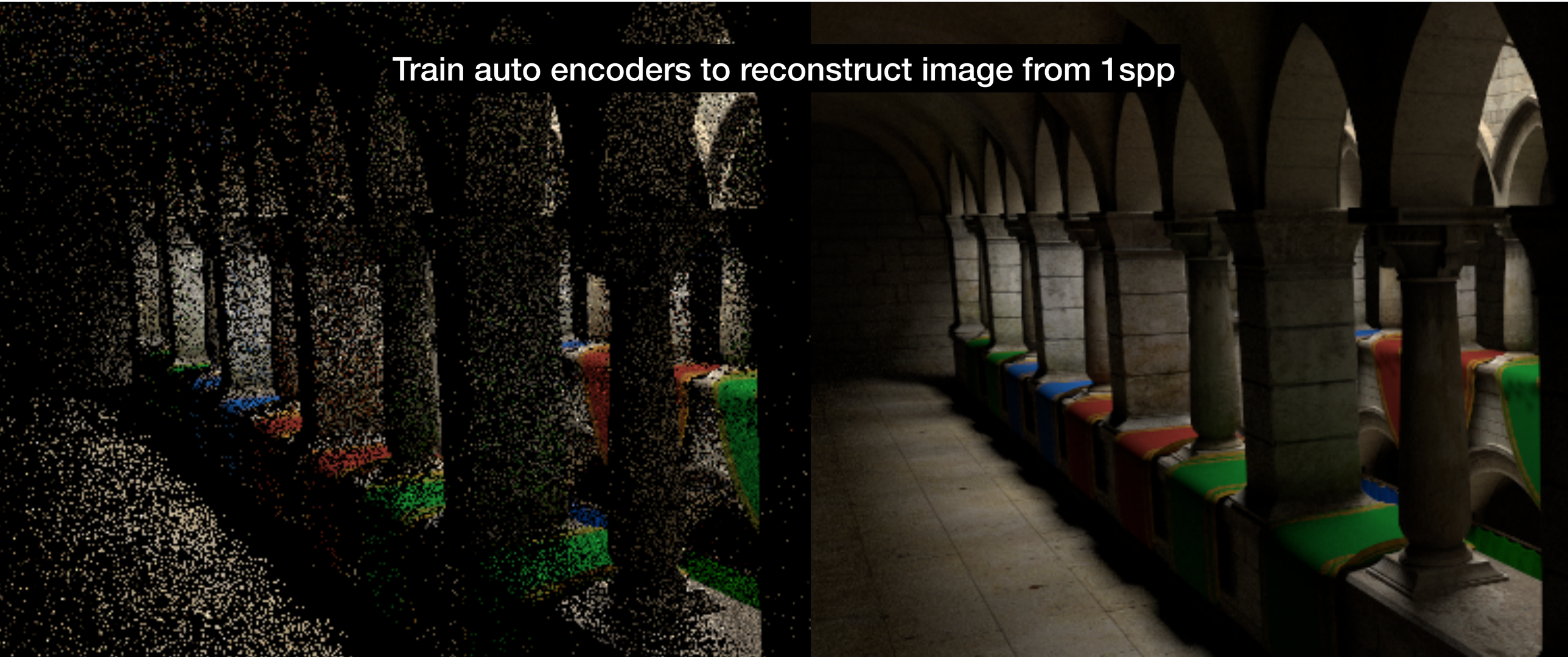
Path-tracing from the primary paths

- 1 ray for direct shadows

- 2 rays for indirect (sample + connect)

1 direct + 1 indirect path (spp)

Denoising Autoencoder (DAE)



Recurrent Autoencoder

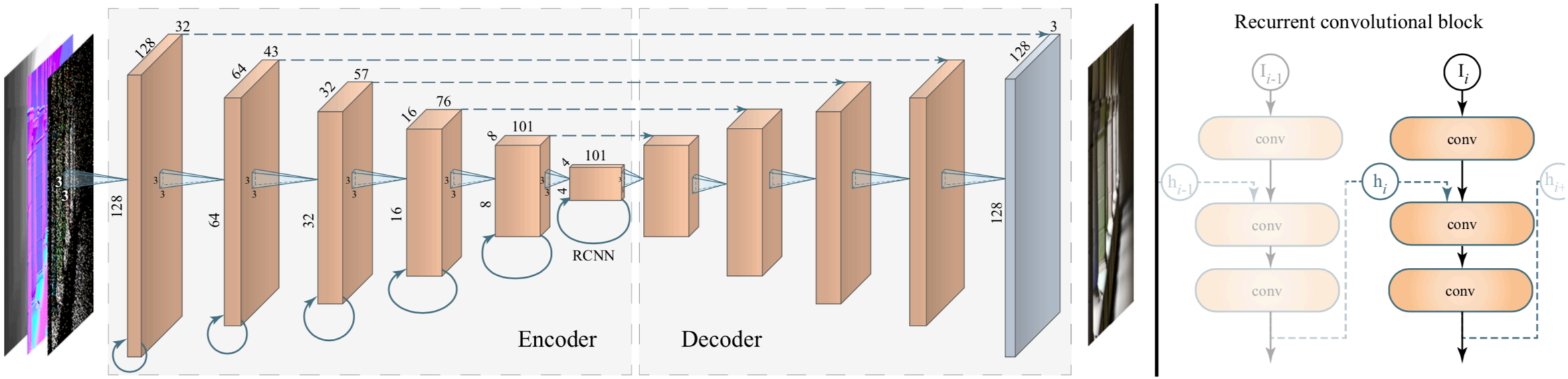


Fig. 2. Architecture of our recurrent autoencoder. The input is 7 scalar values per pixel (noisy RGB, normal vector, depth, roughness). Each encoder stage has a convolution and 2×2 max pooling. A decoder stage applies a 2×2 nearest neighbor upsampling, concatenates the per-pixel feature maps from a skip connection (the spatial resolutions agree), and applies two sets of convolution and pooling. All convolutions have a 3×3-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.

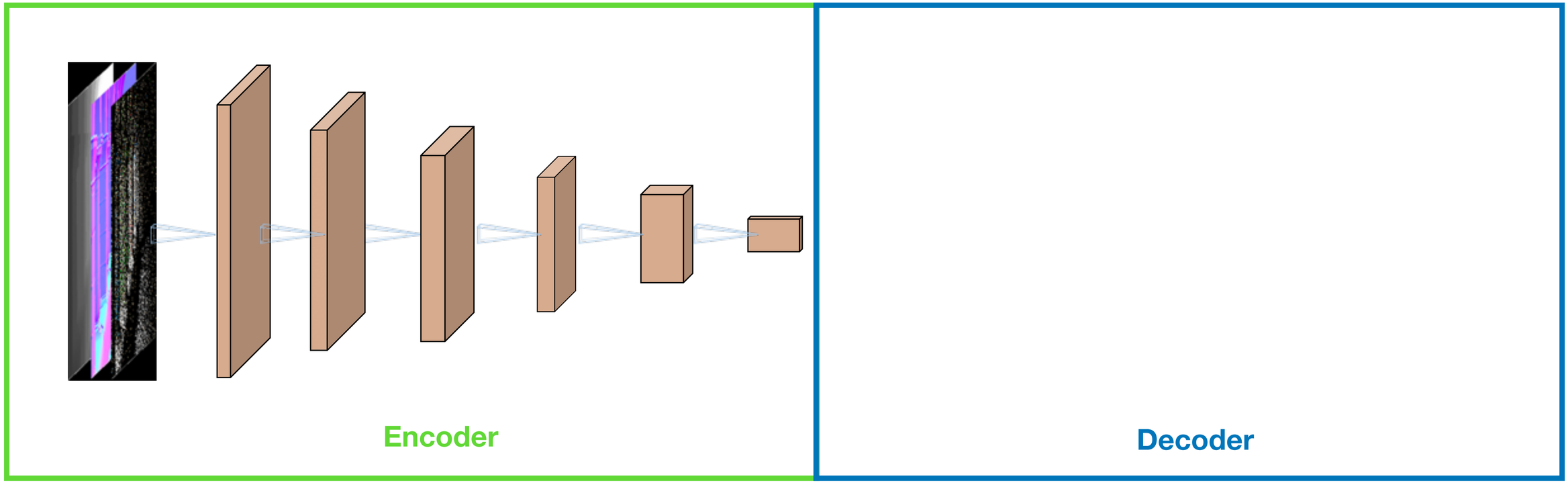
[Chaitanya et al. 2017]

Recurrent Neural Networks

Encoder and decoder stages for dimensionality reduction

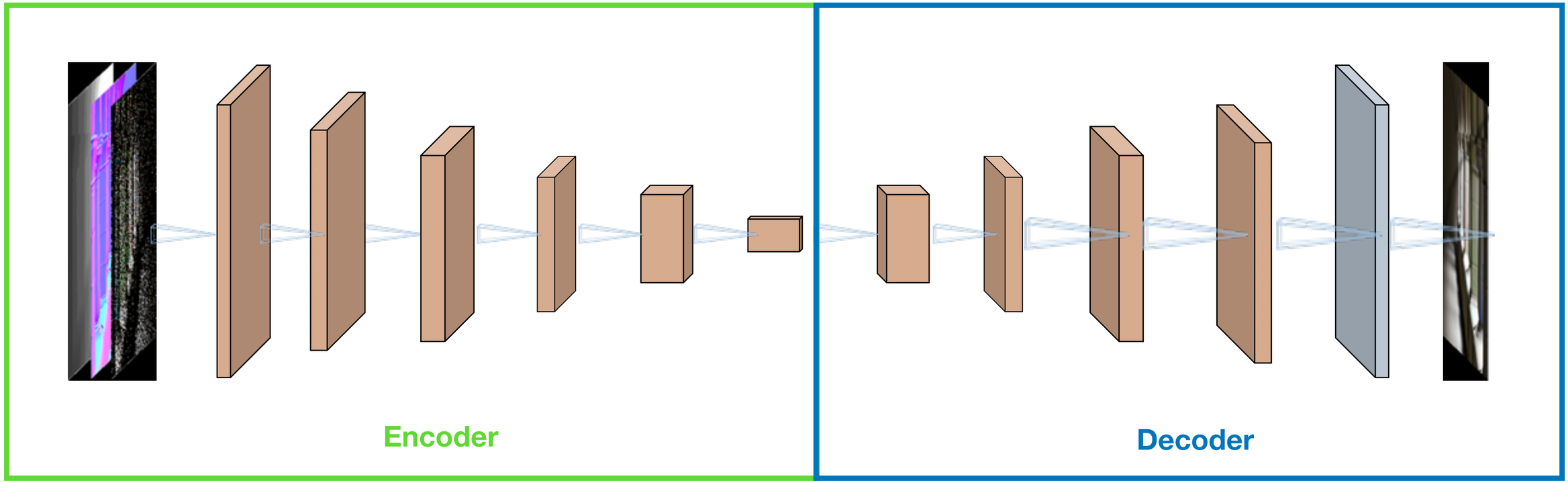
Recurrent Neural Networks

Encoder and decoder stages for dimensionality reduction



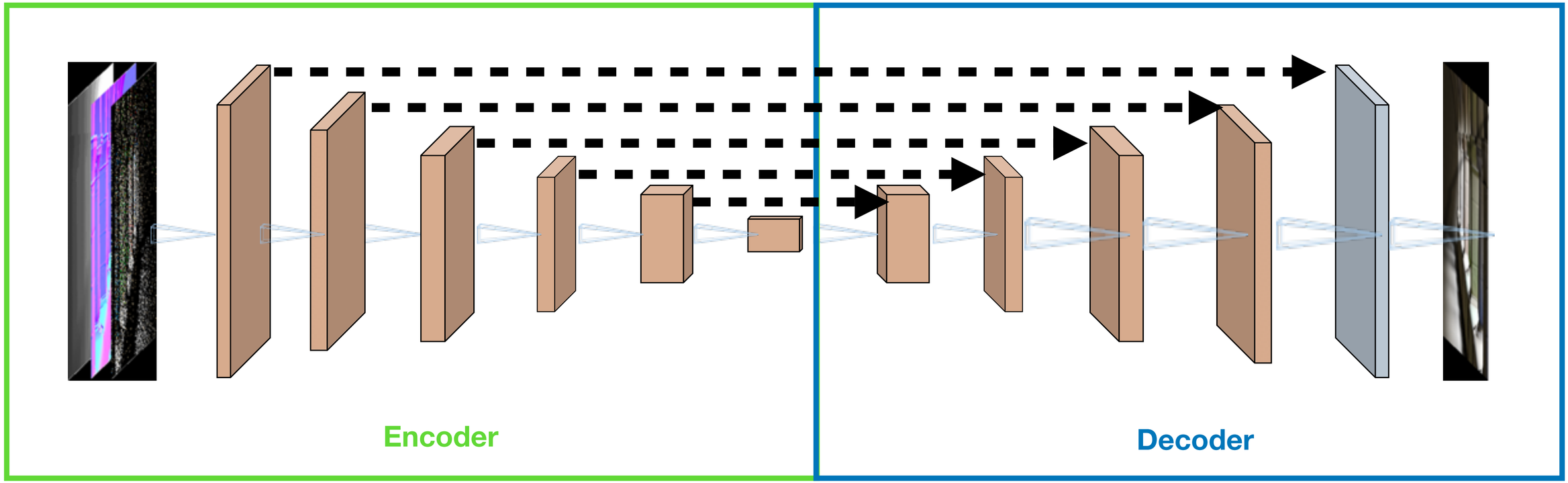
Recurrent Neural Networks

Encoder and decoder stages for dimensionality reduction



Recurrent Neural Networks

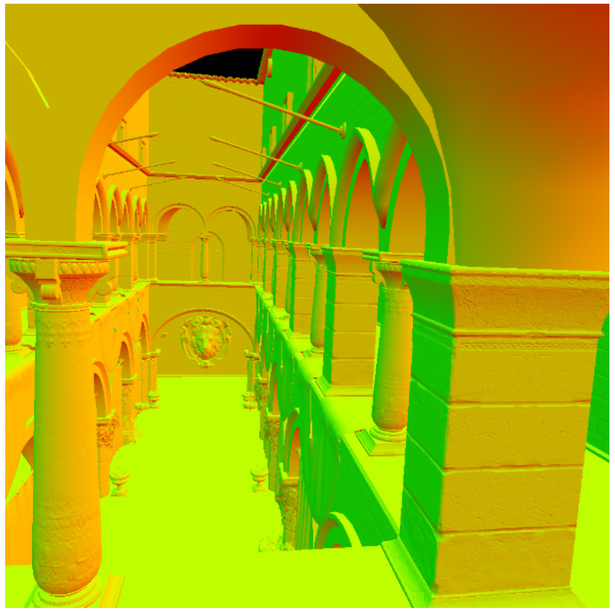
Encoder and decoder stages for dimensionality reduction



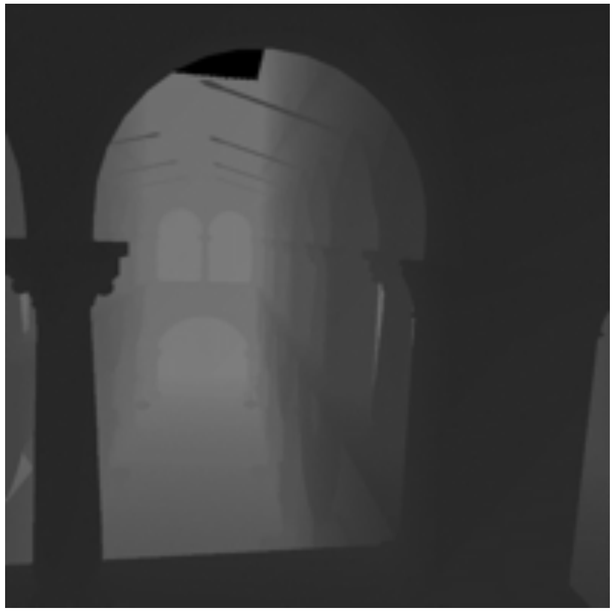
Skip connections to reintroduce lost information

Auxillary Features

Untextured color



View space normals



Linearize depth

Training sequences

SponzaDiffuse

SponzaGlossy

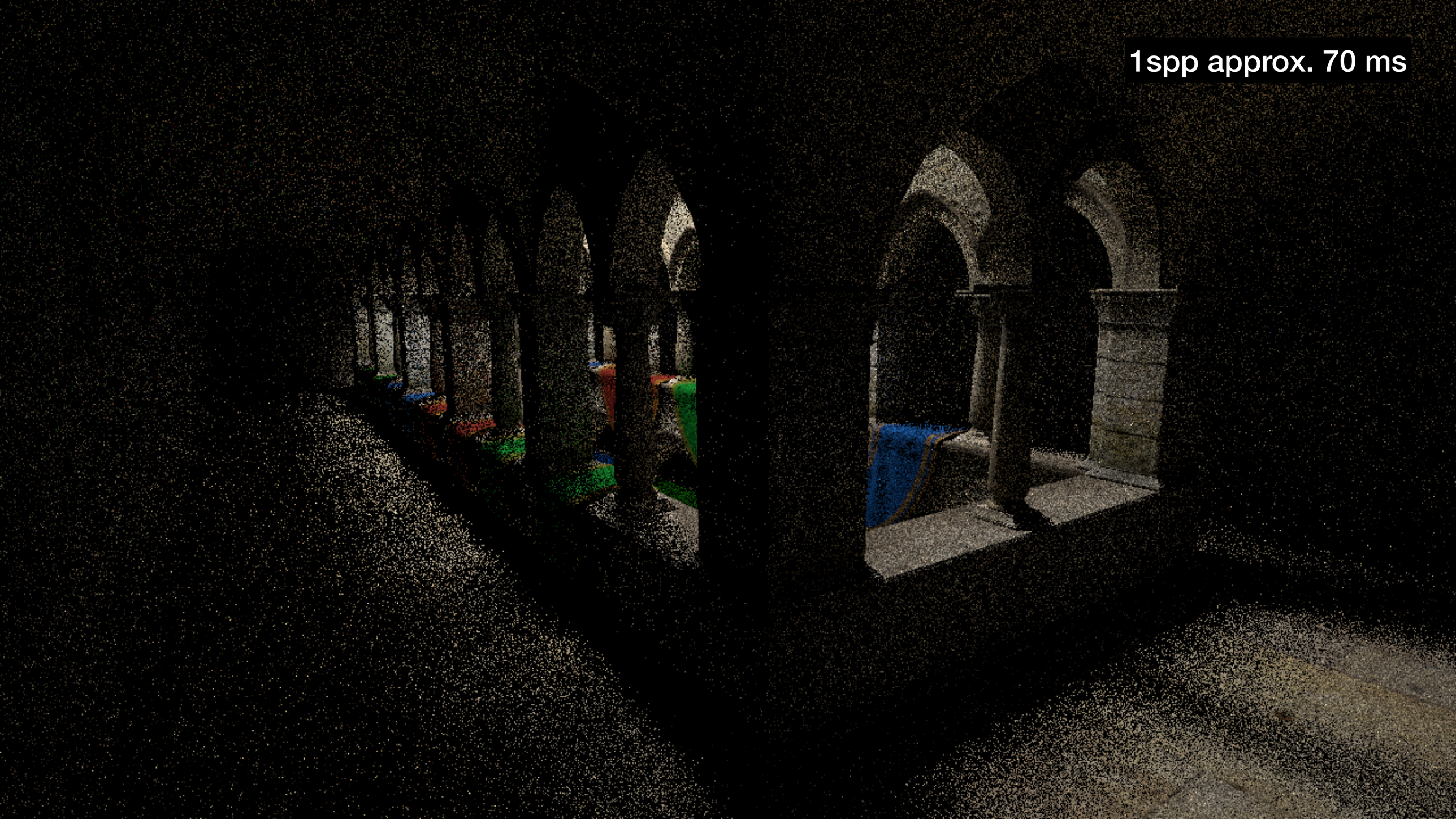
Classroom

Training sequences

SponzaDiffuse

SponzaGlossy

Classroom

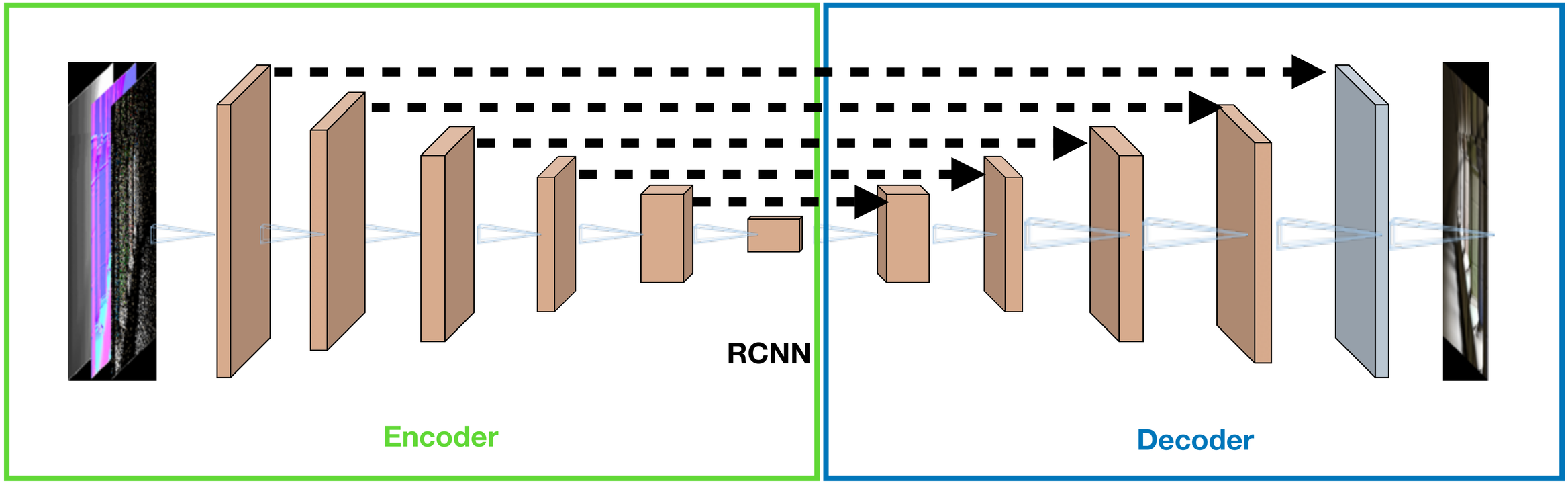

1spp approx. 70 ms


DAE 1spp
approx. 70 ms + approx. 60 ms


Reference 1024 spp
approx. 240 ms

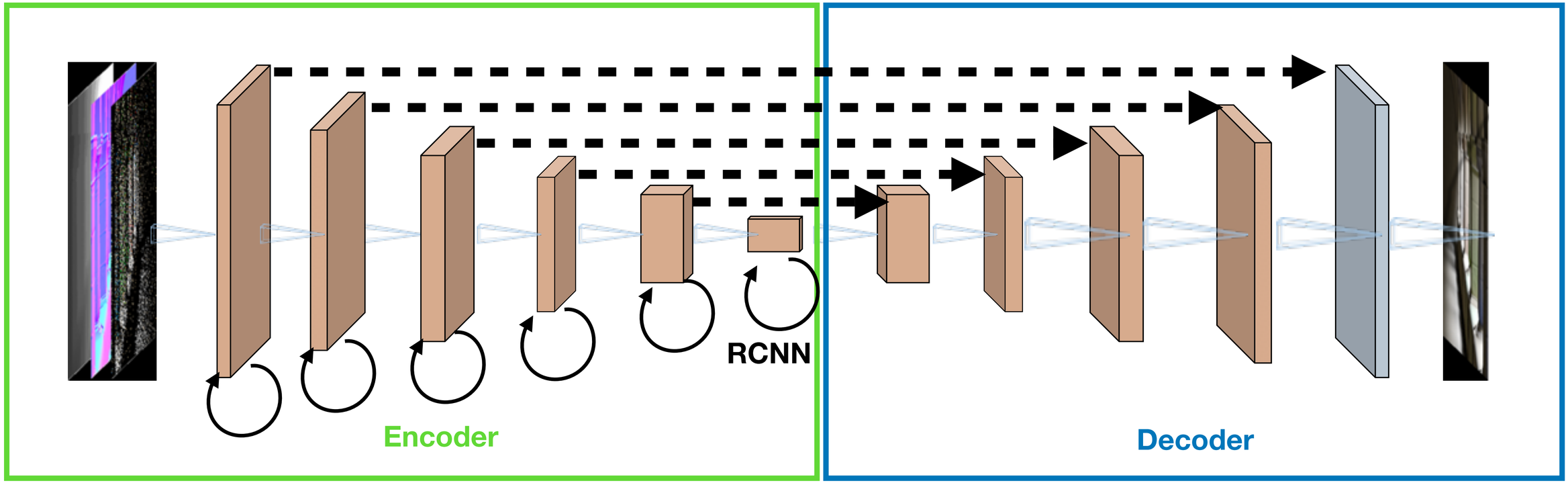
Recurrent Denoising Autoencoder

Feedback loops to retain important information after every encoding stage

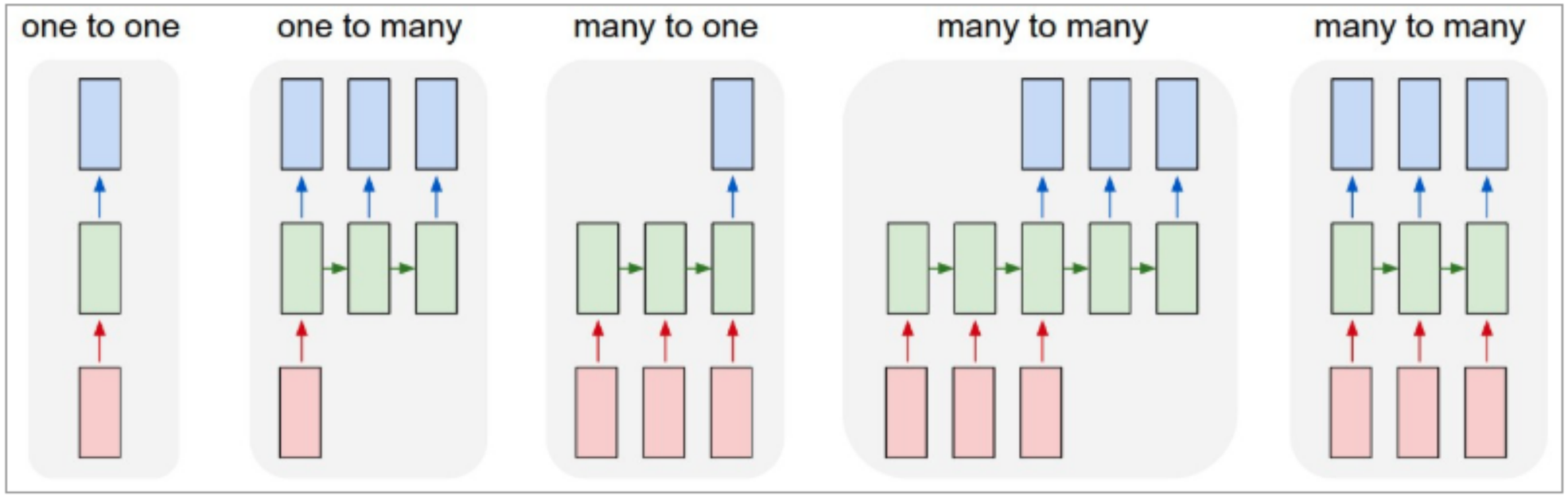


Recurrent Denoising Autoencoder

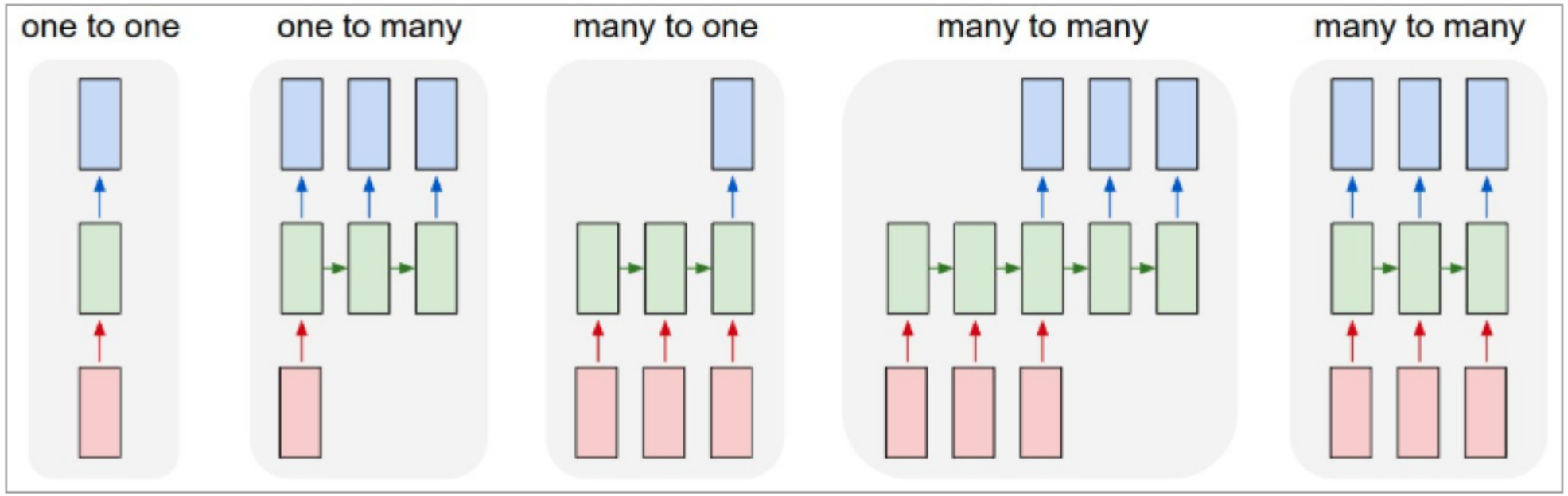
Feedback loops to retain important information after every encoding stage



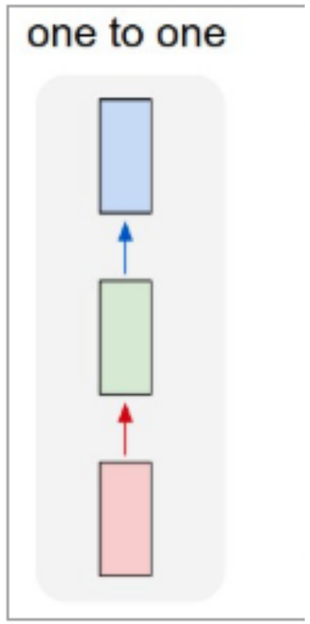
Recurrent Neural Networks



Recurrent Neural Networks

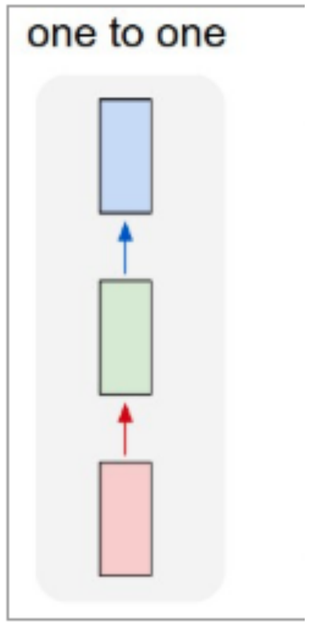


Recurrent Neural Networks



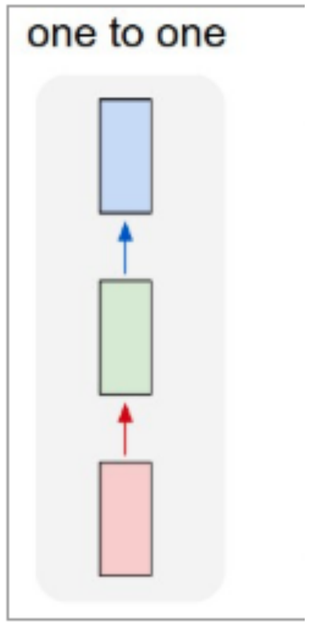
Recurrent Neural Networks

CNNs,
fixed input,
fixed output



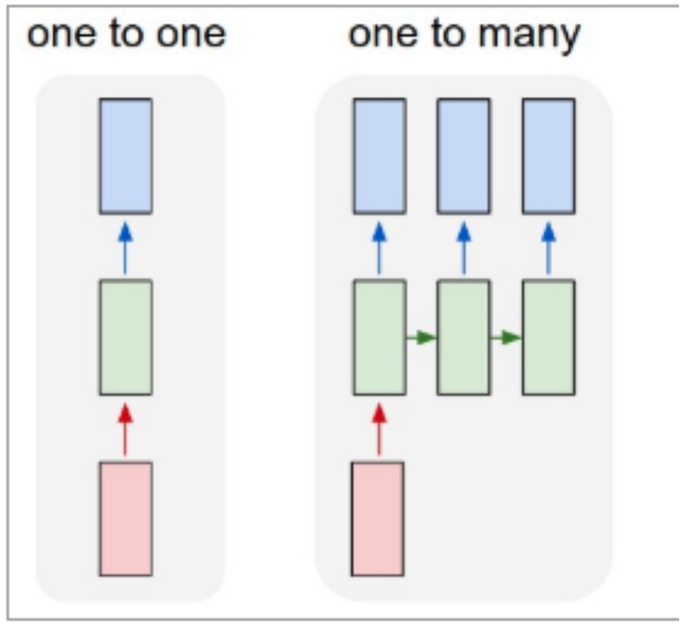
Recurrent Neural Networks

CNNs,
fixed input,
fixed output



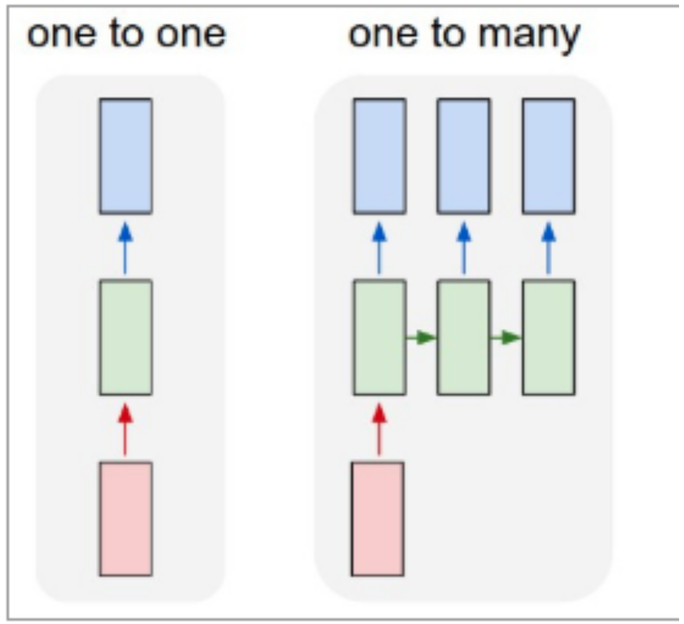
Recurrent Neural Networks

CNNs,
fixed input,
fixed output



Recurrent Neural Networks

CNNs,
fixed input,
fixed output



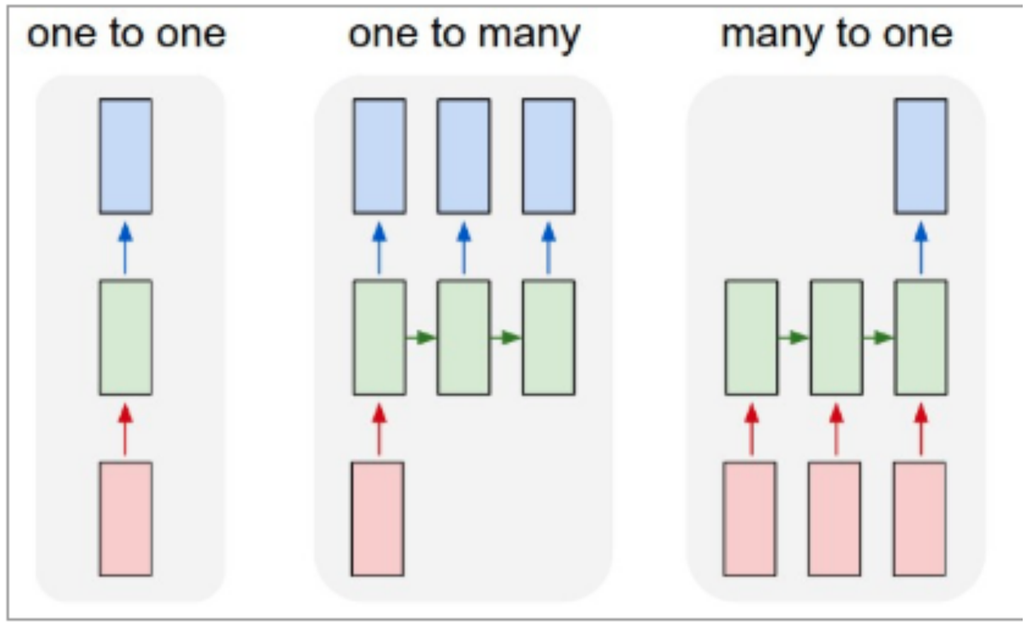
Sequence output

e.g., image captioning takes an image as input and outputs a sentence of words

Recurrent Neural Networks

CNNs,
fixed input,
fixed output

Sequence input



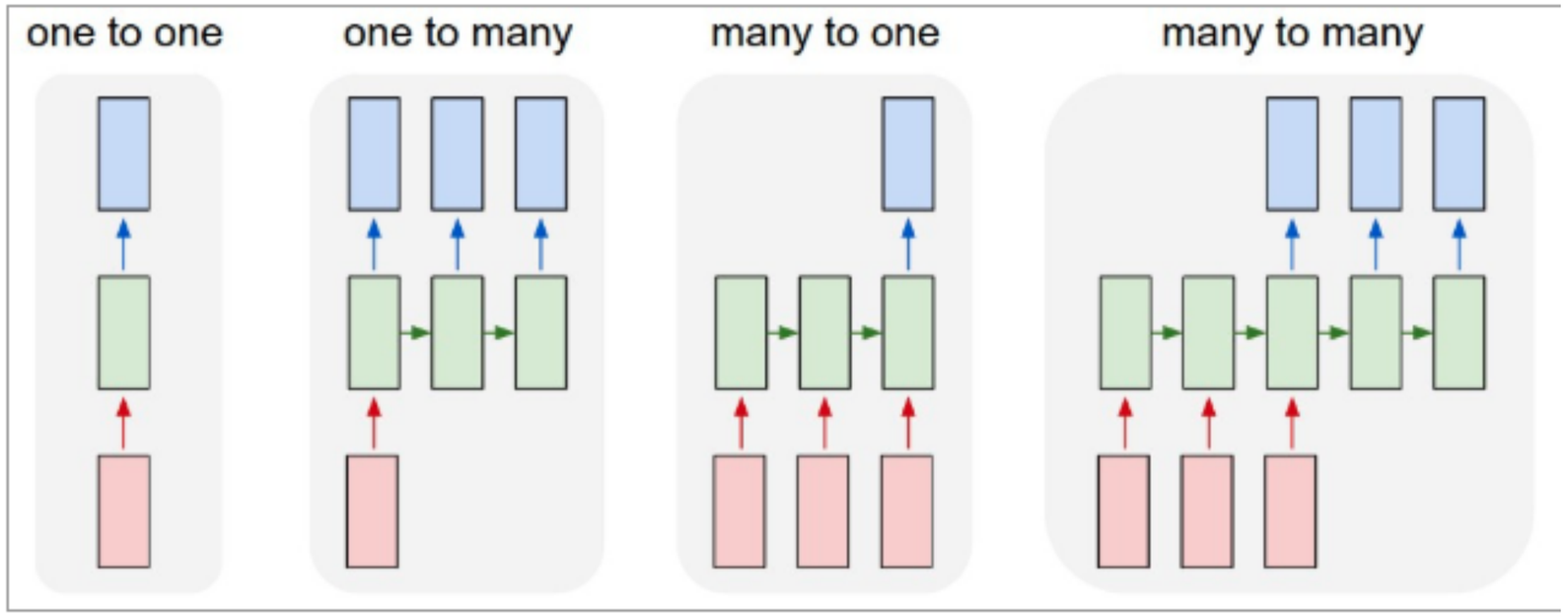
Sequence output

e.g., to know the sentiments of a sentence

Recurrent Neural Networks

CNNs,
fixed input,
fixed output

Sequence input



Sequence output

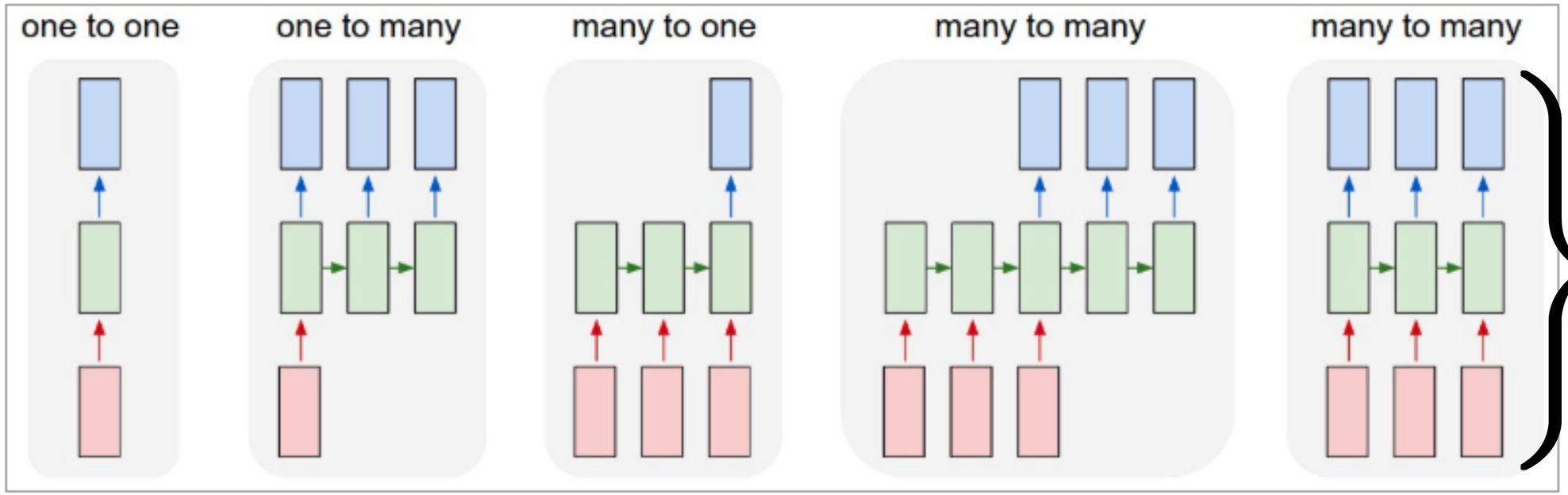
Sequence input,
Sequence output.
e.g. Machine translation

Recurrent Neural Networks

CNNs,
fixed input,
fixed output

Sequence input

Synced sequence
Input & output



e.g., video classification where we want to label each frame

Sequence output

Sequence input,
Sequence output.
e.g. Machine translation

Training

Input is a sequence of 7 frames

128x128 random image crop per sequence

Play the sequence forward/backward

Each frame advance the camera or random seed

Loss Functions

Spatial Loss to emphasize more the dark regions

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

Loss Functions

Spatial Loss to emphasize more the dark regions

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

Temporal loss

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

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

$$L_g = \frac{1}{N}\sum_i^N |\nabla P_i - \nabla T_i|$$

Loss Functions

Spatial Loss to emphasize more the dark regions

$$L_s = \frac{1}{N}\sum_i^N |P_i - T_i|$$

Temporal loss

$$L_t = \frac{1}{N}\sum_i^N \left(\left| \frac{\partial P_i}{\partial t} - \frac{\partial T_i}{\partial t} \right| \right)$$

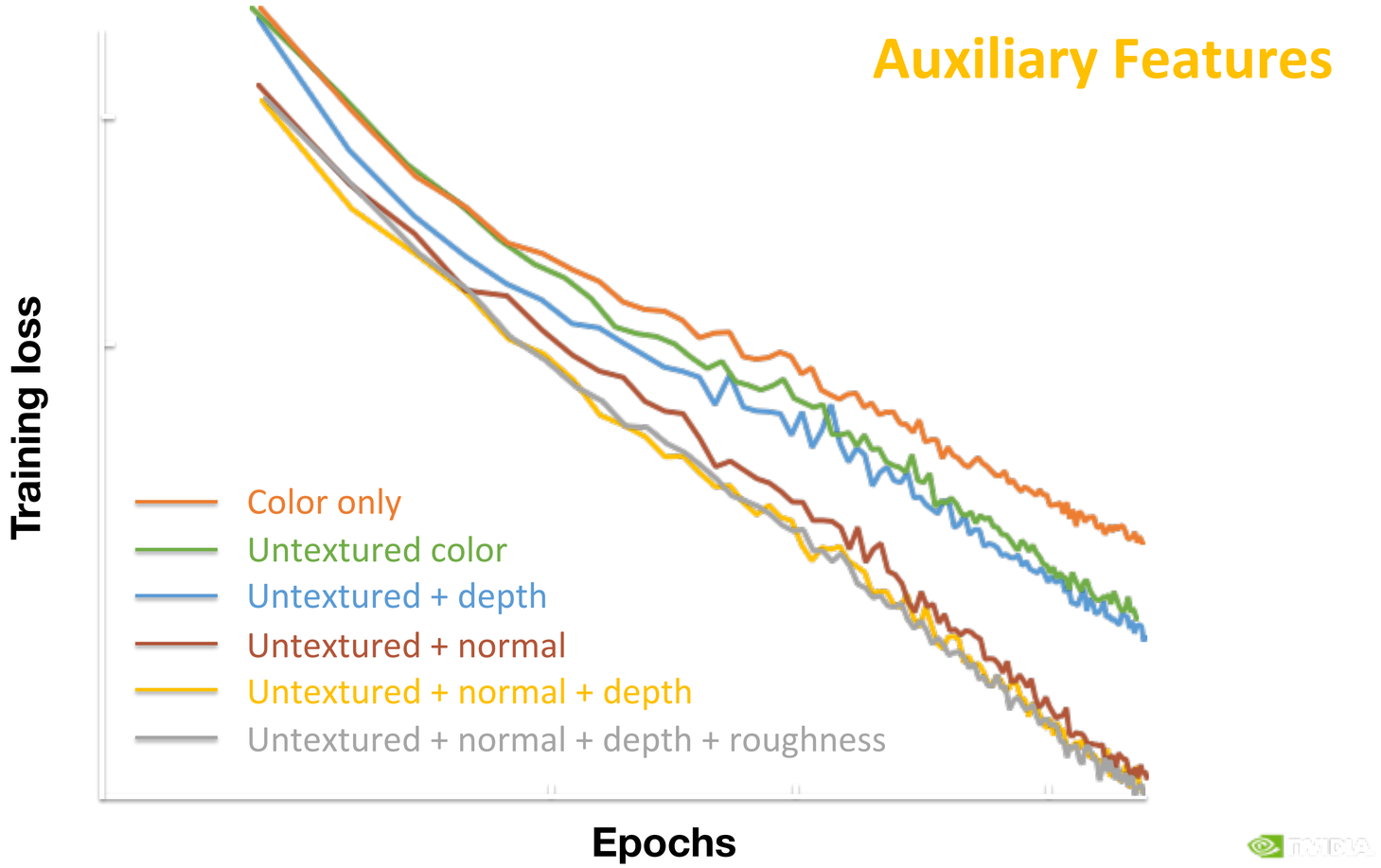
High frequency error norm loss for stable edges

$$L_g = \frac{1}{N}\sum_i^N |\nabla P_i - \nabla T_i|$$

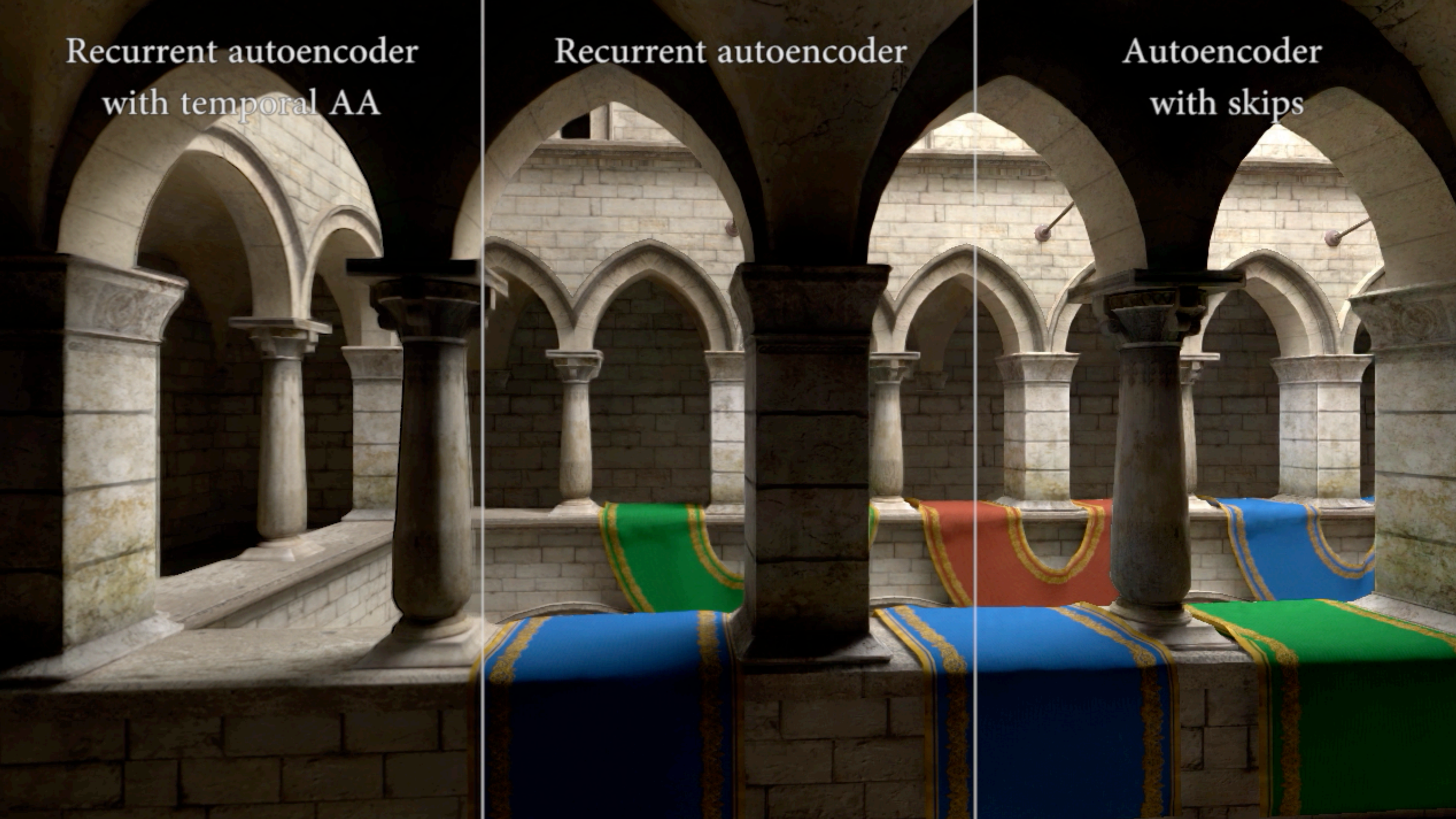
Final Loss is a weighted averaged of above losses

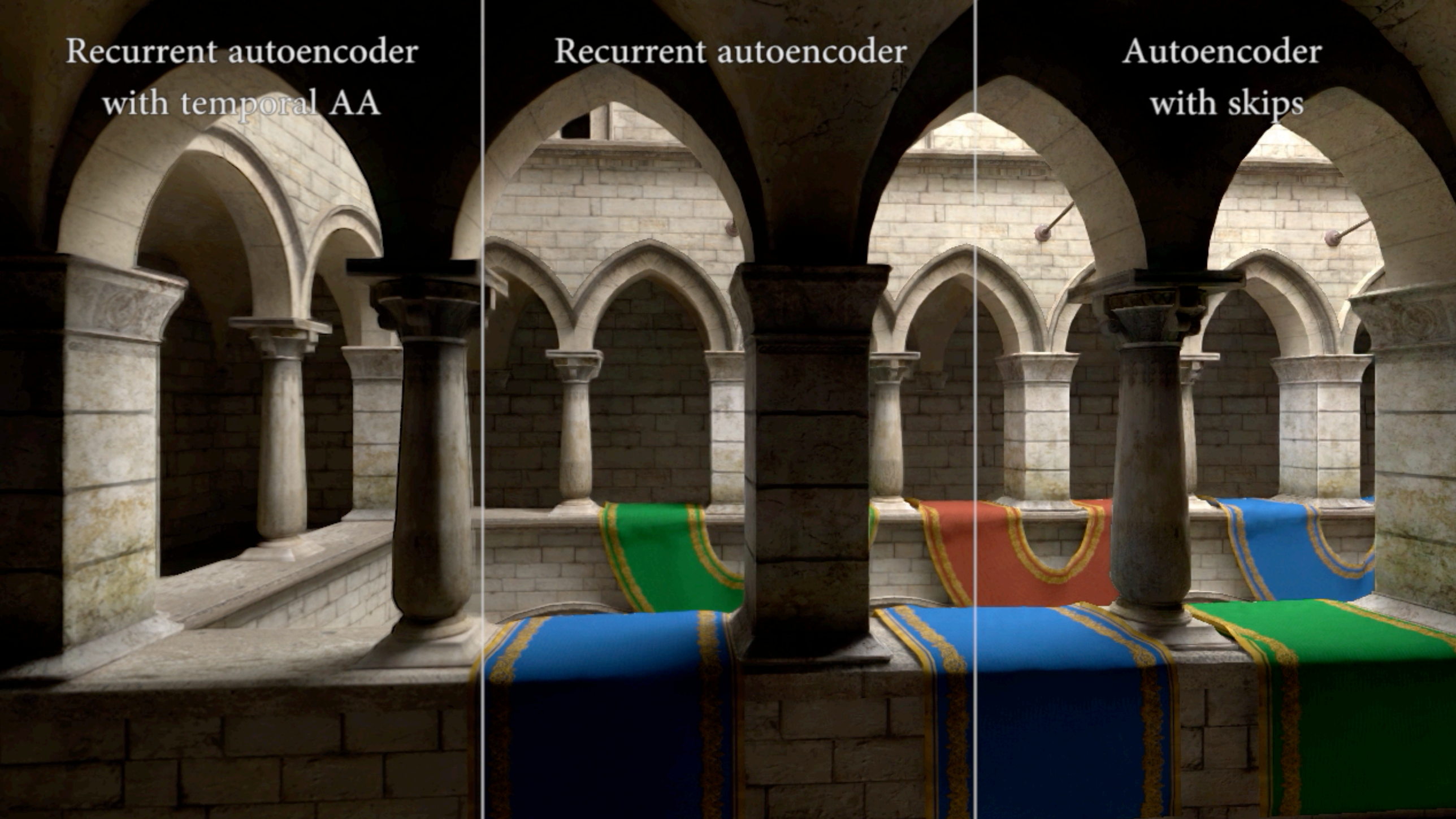
$$L = w_s L_s + w_g L_g + w_t L_t$$

Training Loss depends on Auxiliary Features



Temporal Stability


Recurrent autoencoder
with temporal AA
Recurrent autoencoder
Autoencoder
with skips


Recurrent autoencoder
with temporal AA
Recurrent autoencoder
Autoencoder
with skips


1 sample/pixel input


1 sample/pixel input


Recurrent autoencoder
1 sample/pixel input


Recurrent autoencoder
1 sample/pixel input

Introduction to CNNs

Introduction to CNNs

Kernel Predicting Denoising

Introduction to CNNs

Kernel Predicting Denoising

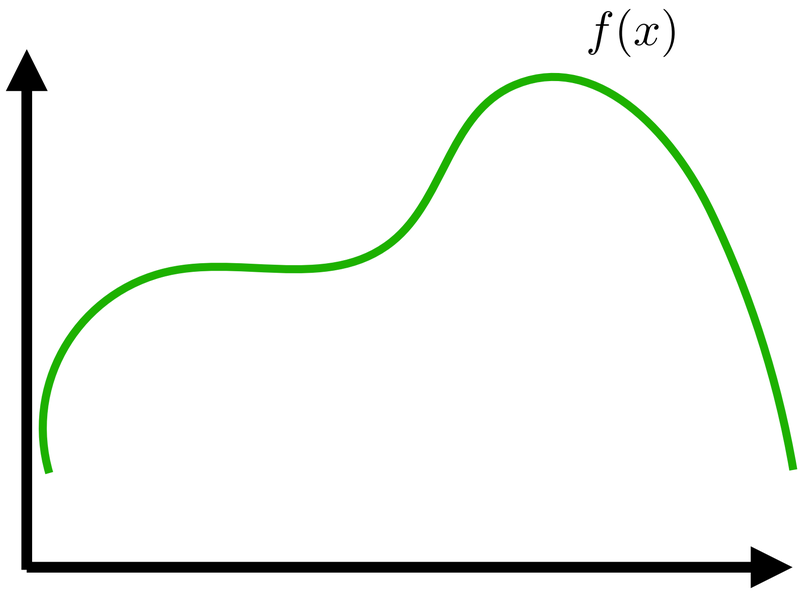
Neural Importance Sampling

Normalizing Flows: Importance Sampling

Importance Sampling

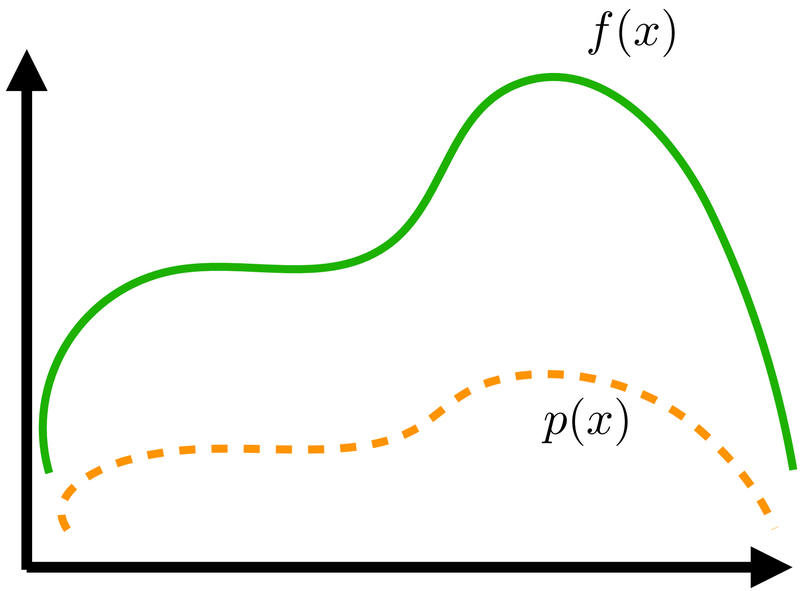
$$I_N = \frac{1}{N} \sum_{k=1}^{N} \frac{f(x)}{p(x)}$$

Importance Sampling



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

Importance Sampling



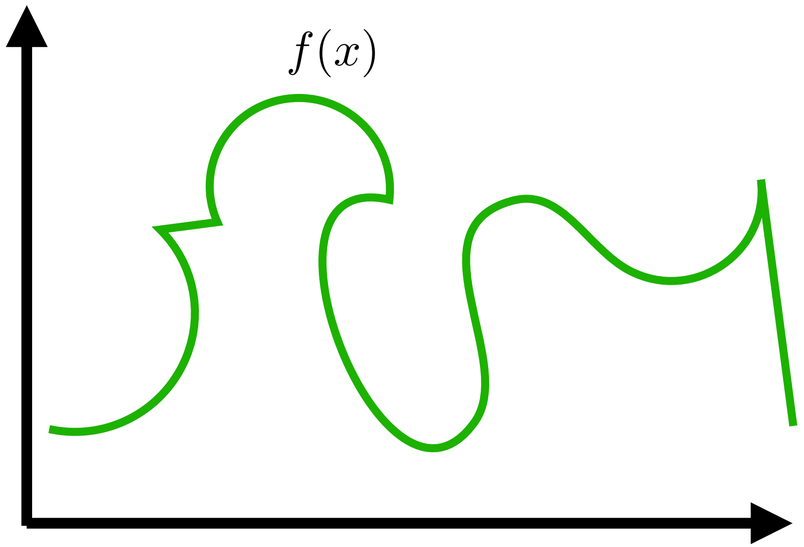
$$I_N = \frac{1}{N} \sum_{k=1}^{N} \frac{f(x)}{p(x)}$$

Importance Sampling

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

$$p(x) = ???$$

Importance Sampling



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

$$p(x) = ???$$

Normalizing Flows

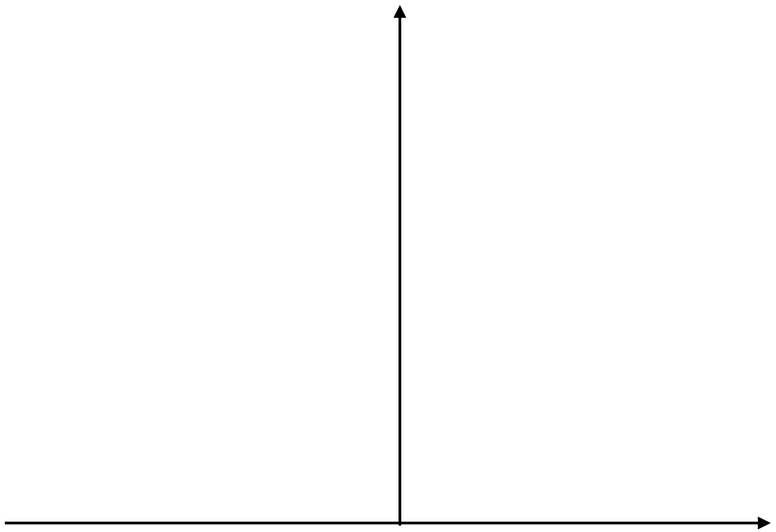
Normalizing Flows

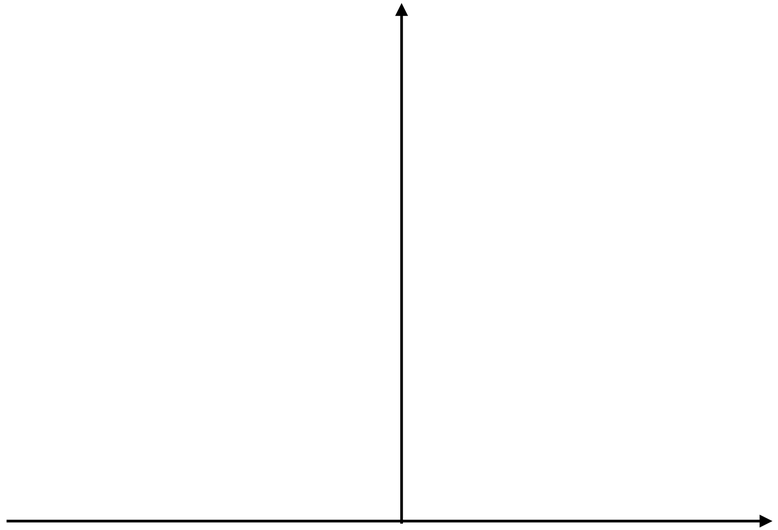
Technique used in Machine learning to build complex probability distributions by transforming simple ones

Used in the context of generative modeling

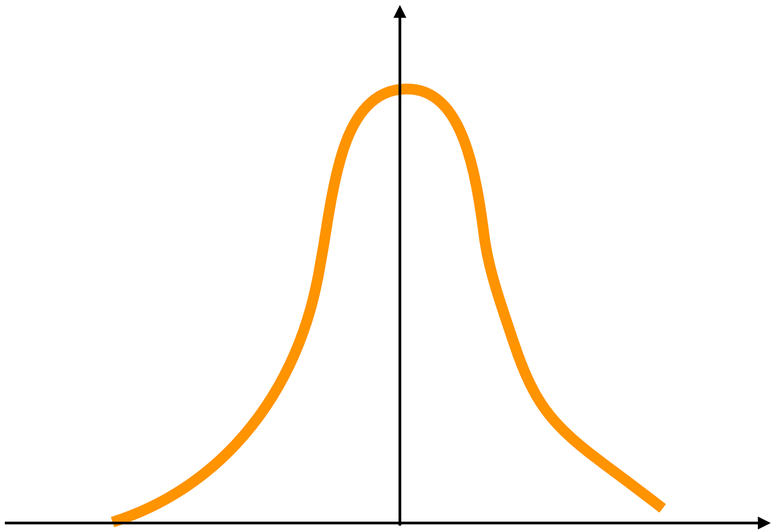
Generative modeling: learning without any target (unsupervised)

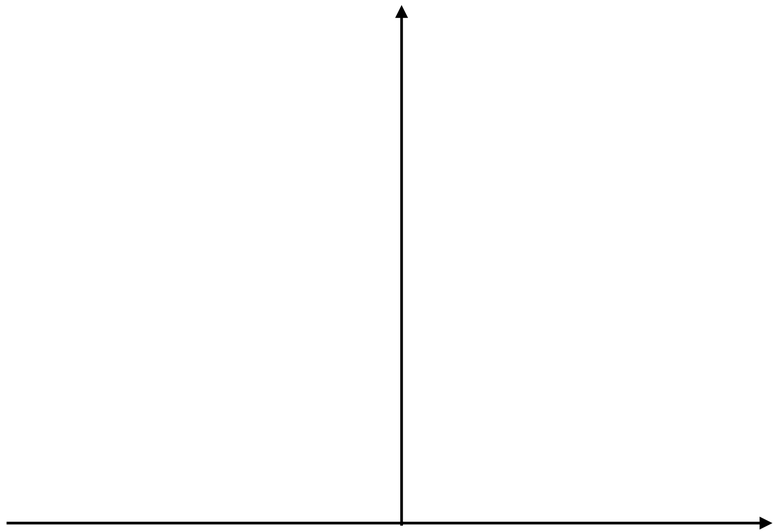
Complex Probability distributions from simple ones



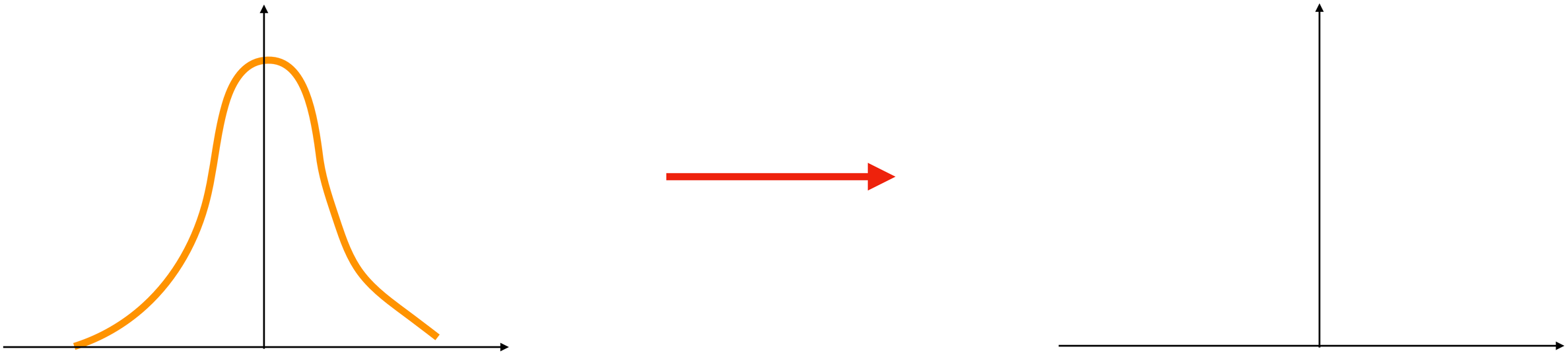


Complex Probability distributions from simple ones

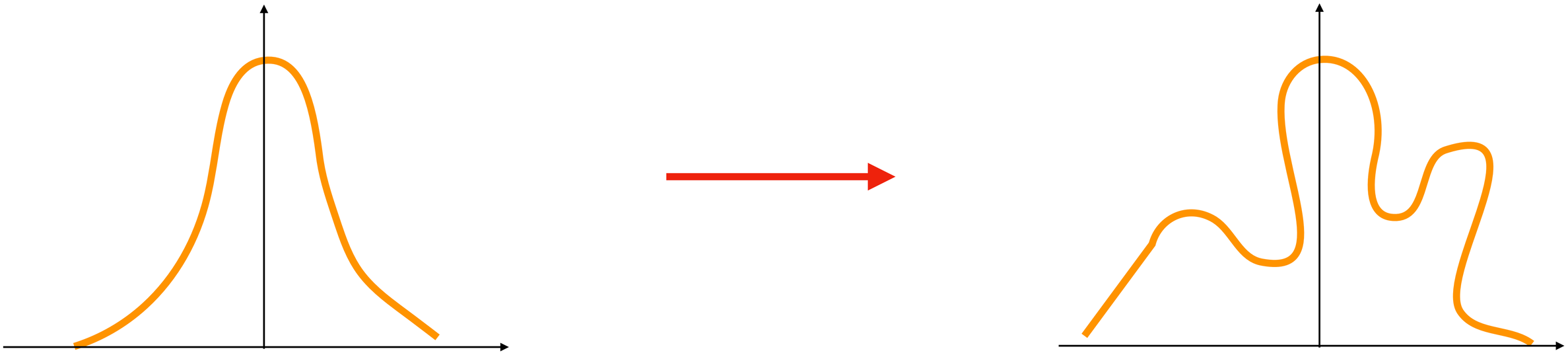




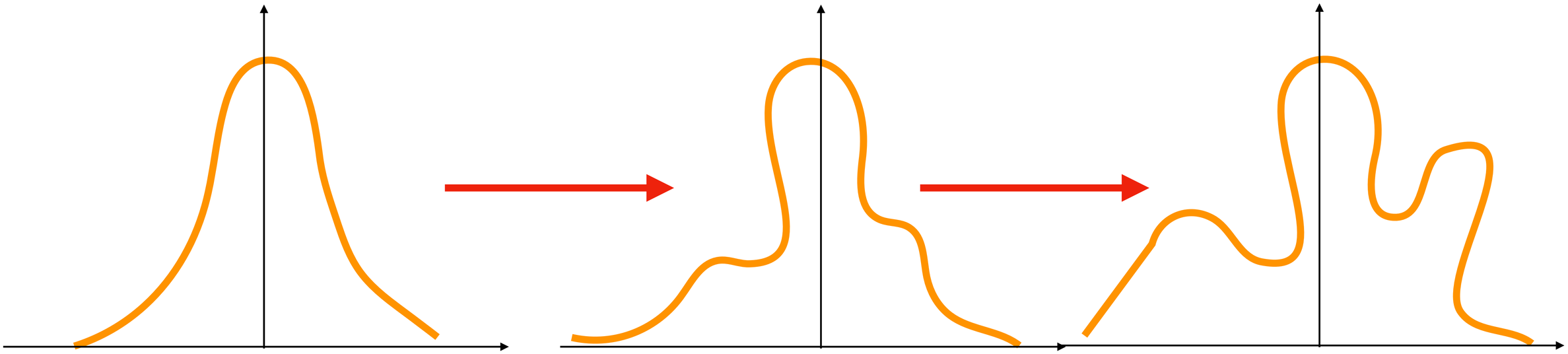
Complex Probability distributions from simple ones



Complex Probability distributions from simple ones



Complex Probability distributions from simple ones



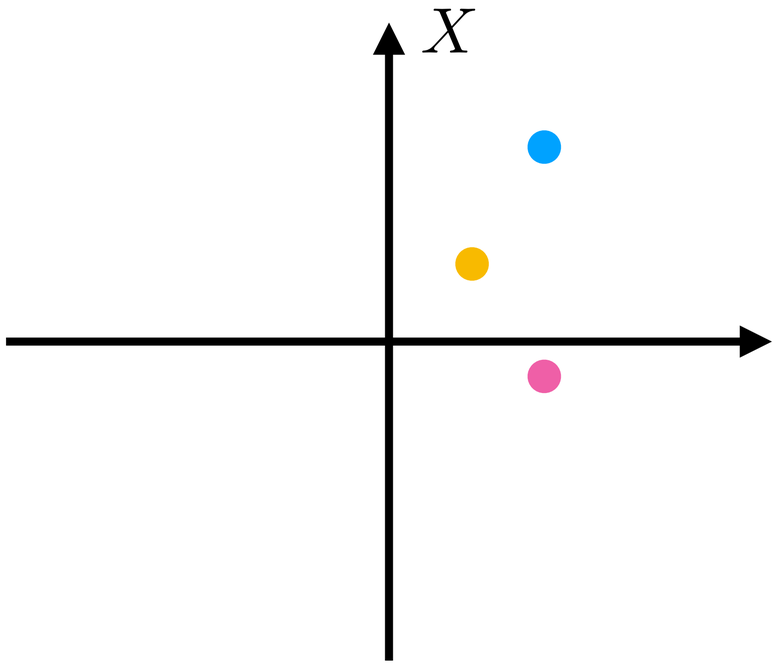
Normalizing Flows:
Basic mathematical framework

$$z \sim p_\theta(z)$$ Given a continuous variable with a distribution

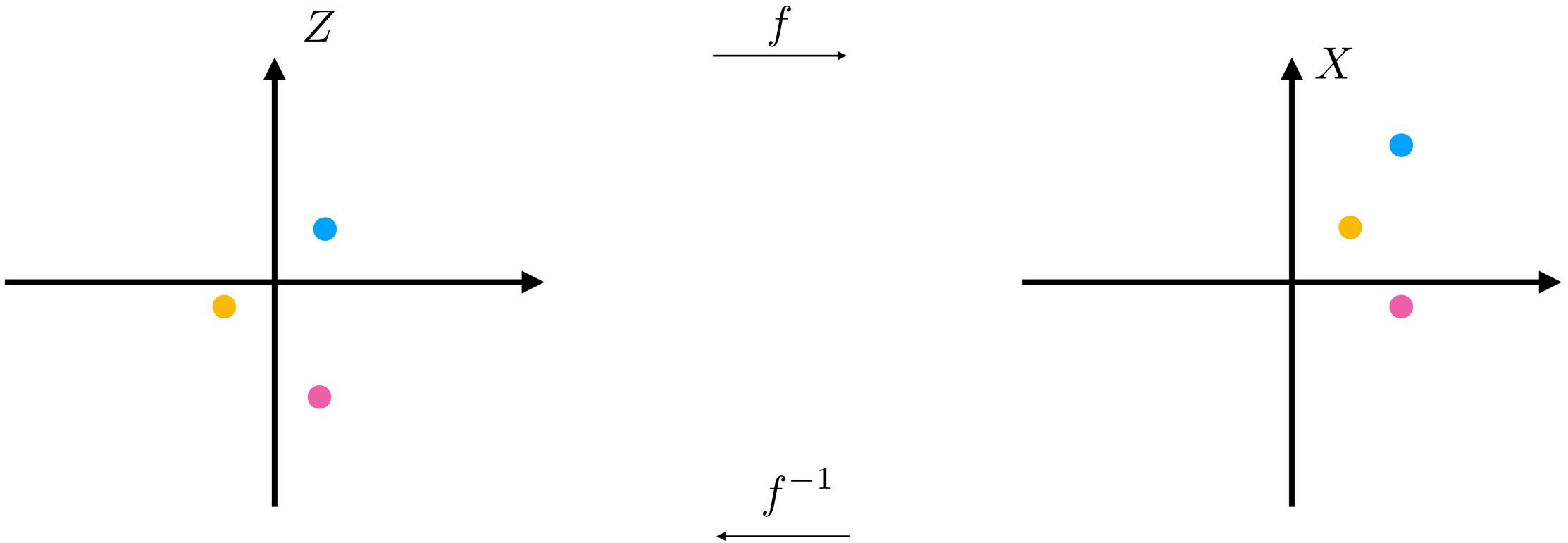
$$x = f_\theta(z) = f_k \circ \circ \circ f_2 \circ f_1(z)$$ New distribution obtained

$$\text{each } f_i \text{ is invertible (bijective)}$$

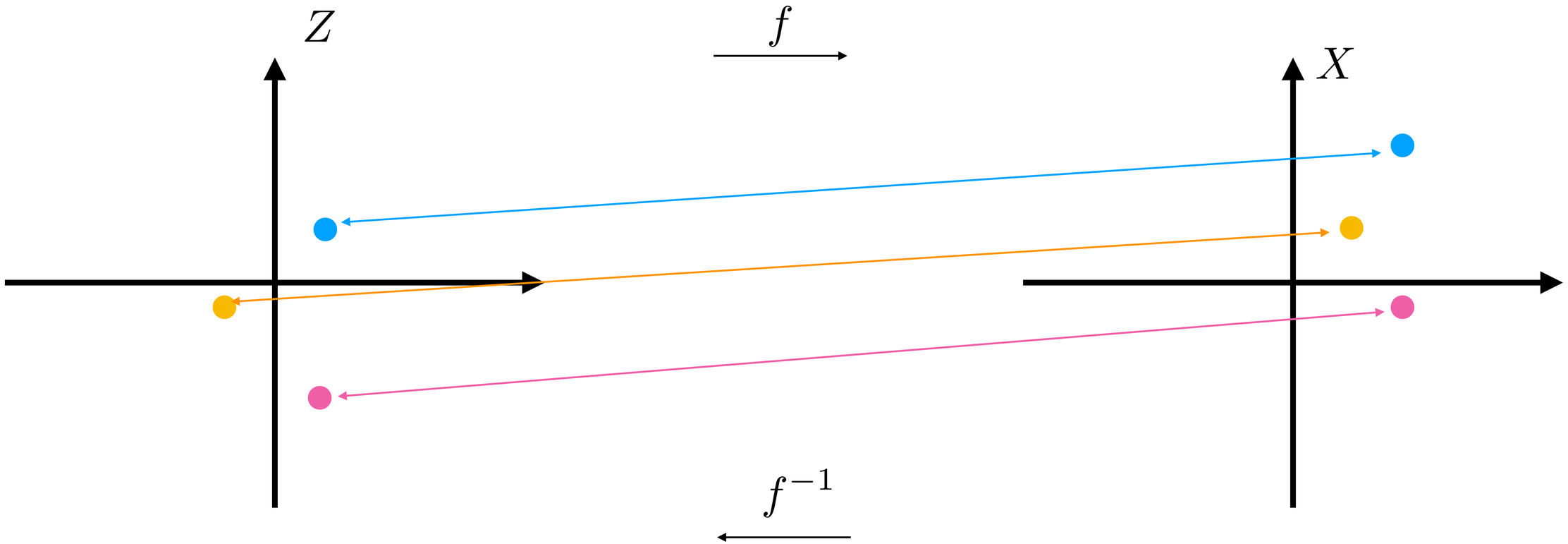
Distributions



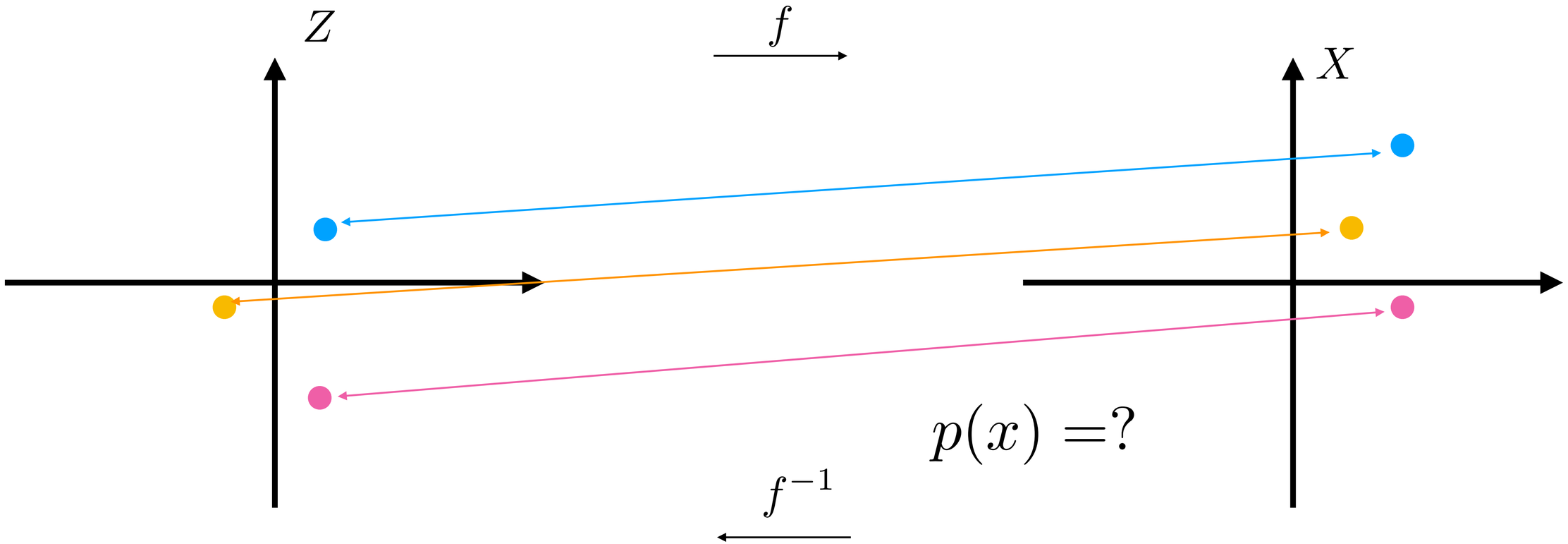
Distributions



Distributions



Distributions



Distributions

$$z \sim p_\theta(z)$$

Given a continuous variable with a distribution

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

$$\text{each } f_i \text{ is invertible (bijective)}$$

$$p(x) = p(f^{-1}(x))$$

Distributions

$z \sim p_\theta(z)$ Given a continuous variable with a distribution

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

$$\text{each } f_i \text{ is invertible (bijective)}$$

$$p(x) \neq p(f^{-1}(x))$$

Change of Variables

$$f : Z \to X, f \text{ is invertible}$$
$$p(z) \text{ defined over } z \in Z$$

Change of Variables

$$f : Z \to X, f \text{ is invertible}$$
$$p(z) \text{ defined over } z \in Z$$

Change of variable formula says that:

Change of Variables

$$f : Z \to X, f \text{ is invertible}$$
$$p(z) \text{ defined over } z \in Z$$

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 $z \in Z$

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

$$p(x) = p(z) \left| det \left(\frac{\partial z}{\partial x} \right) \right|$$

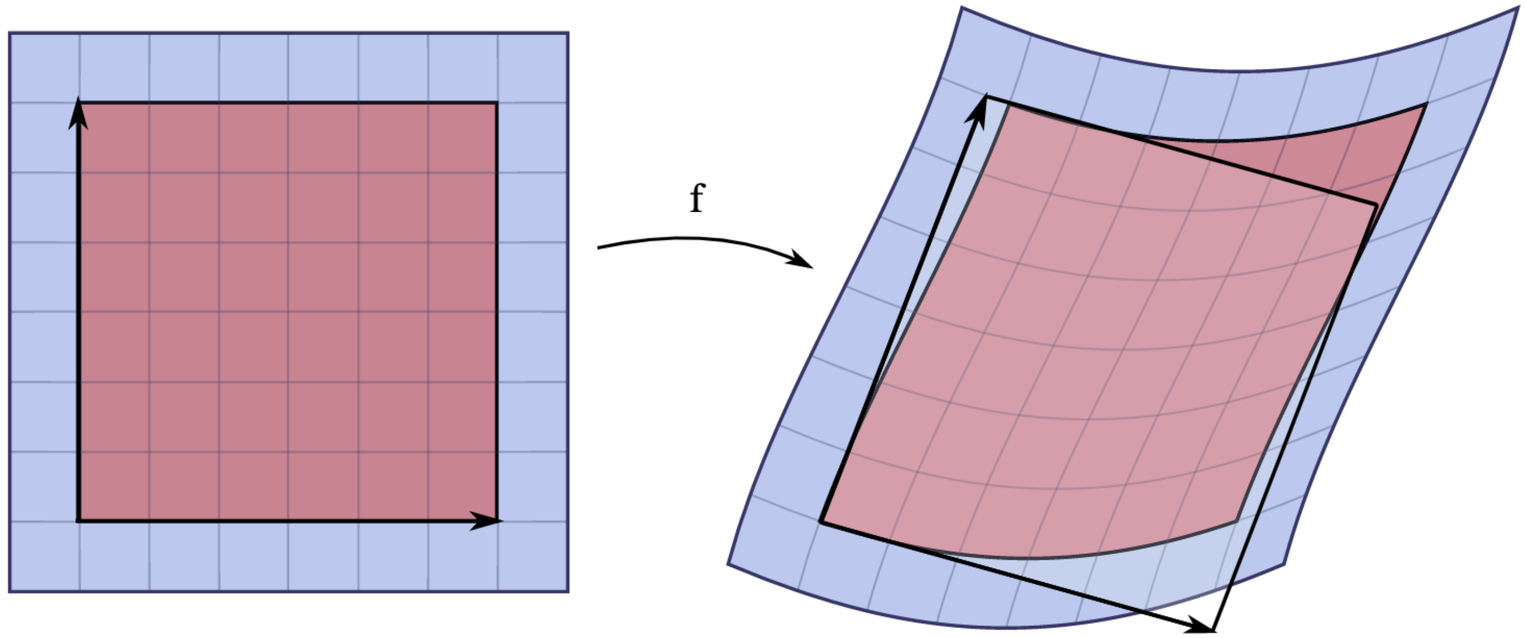
Jacobian Matrix

$$\mathbf{f} : \mathbb{R}^n \rightarrow \mathbb{R}^m$$

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial \mathbf{f}}{\partial x_1} & \cdots & \dfrac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial x_1} & \cdots & \dfrac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Jacobian Matrix

$$\mathbf{f} : \mathbb{R}^2 \rightarrow \mathbb{R}^2$$



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

Jacobian Matrix

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

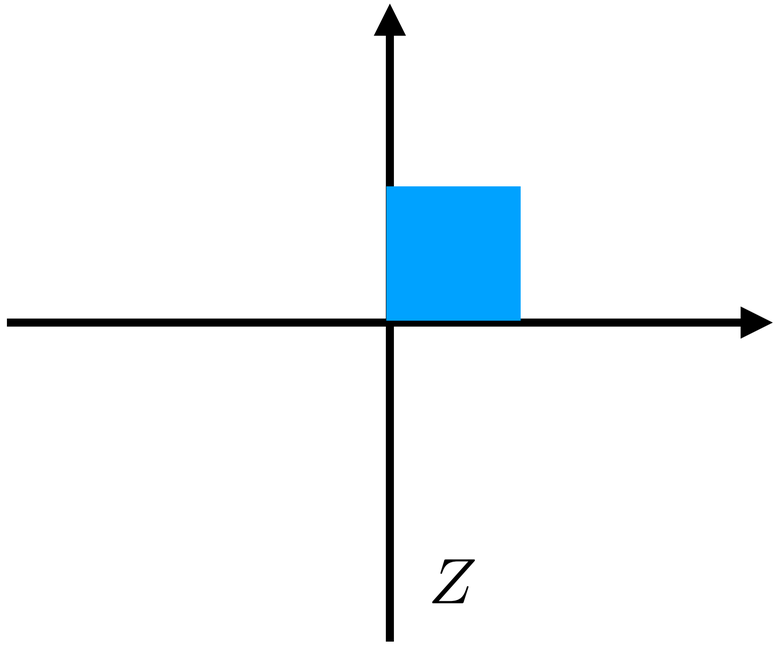
$p(z)$ defined over $z \in Z$

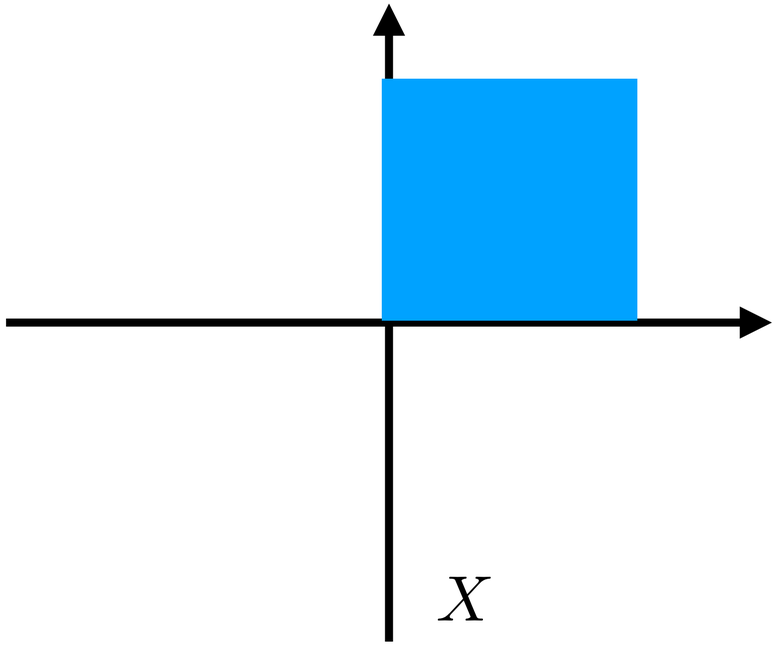
$$p(x) = p(f^{-1}(x)) \left| det \left(\frac{\partial f^{-1}(x)}{\partial x} \right) \right|$$

$$p(x) = p(z) \left| det \left(\frac{\partial z}{\partial x} \right) \right|$$

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

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

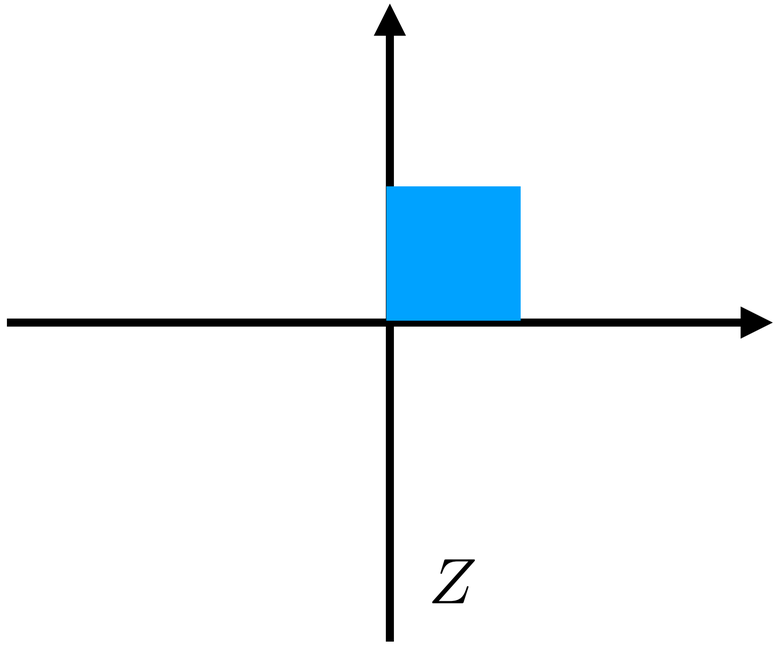


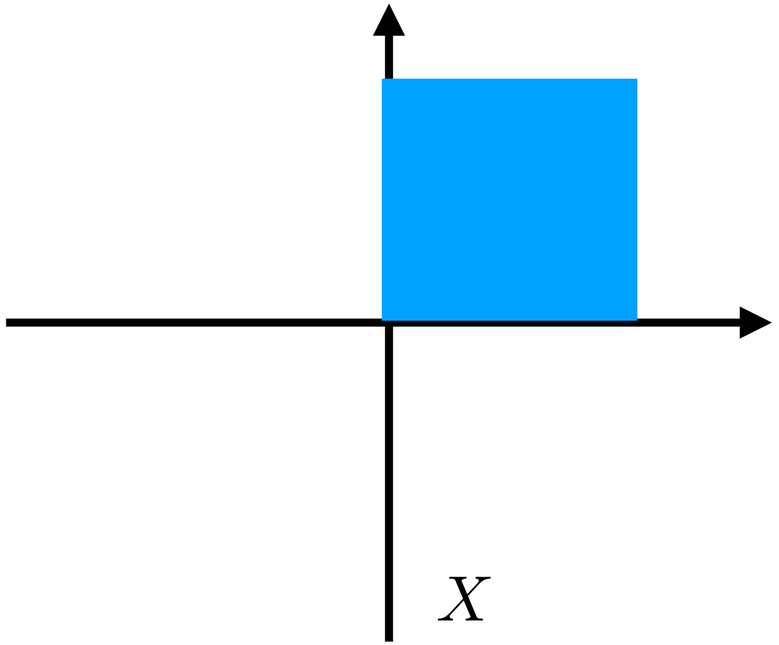


$f : Z \to X, f$ is invertible

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

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

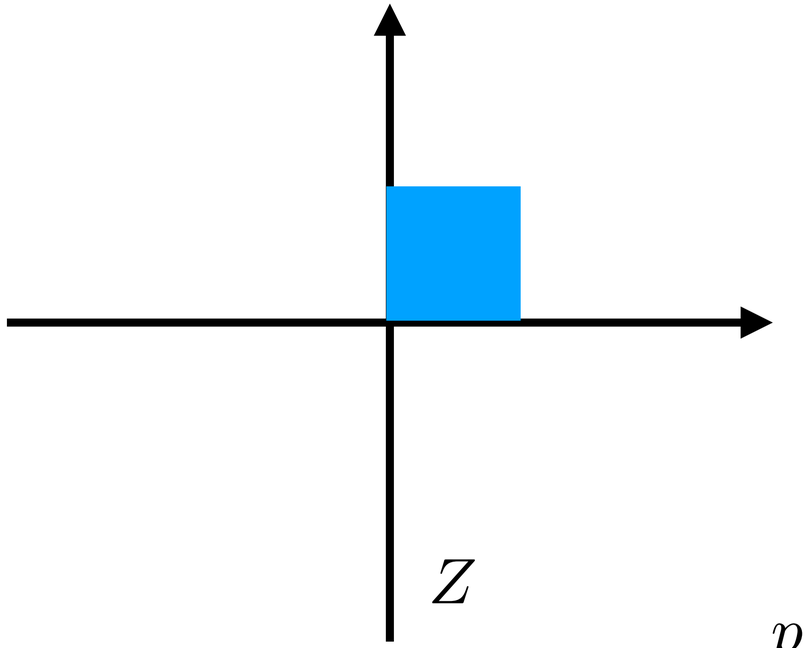




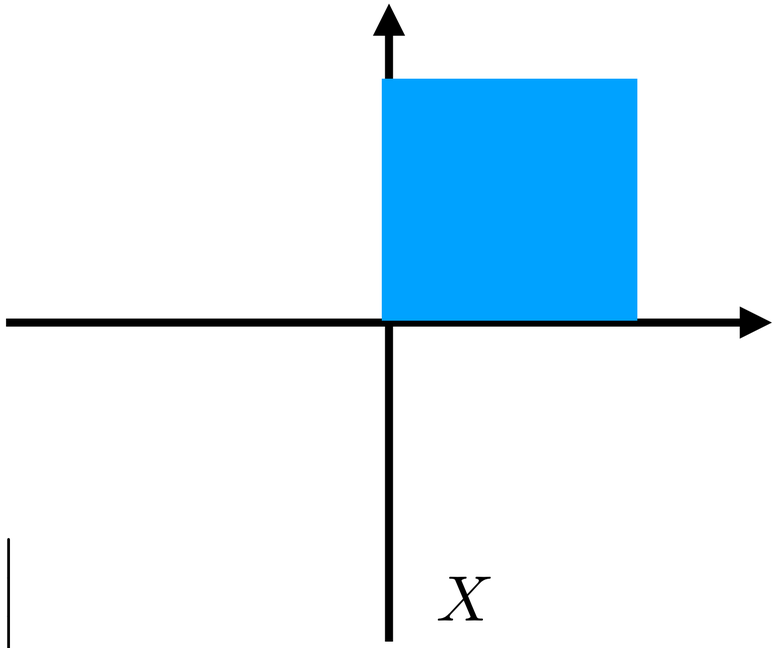
$f : Z \to X, f$ is invertible

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

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



$$p(x) = p(z) \quad \left|\frac{1}{det\left(\frac{\partial x}{\partial z}\right)}\right|$$



Maximize Log-likelihood

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

$p(z)$ 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

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

$p(z)$ 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

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

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

$$\mathbf{J} = \begin{bmatrix} \ell_{1,1} & & & & 0 \\ \ell_{2,1} & \ell_{2,2} & & & \\ \ell_{3,1} & \ell_{3,2} & \ddots & & \\ \vdots & \vdots & \ddots & \ddots & \\ \ell_{n,1} & \ell_{n,2} & \cdots & \ell_{n,n-1} & \ell_{n,n} \end{bmatrix}$$

Jacobian: Lower Triangular Matrix

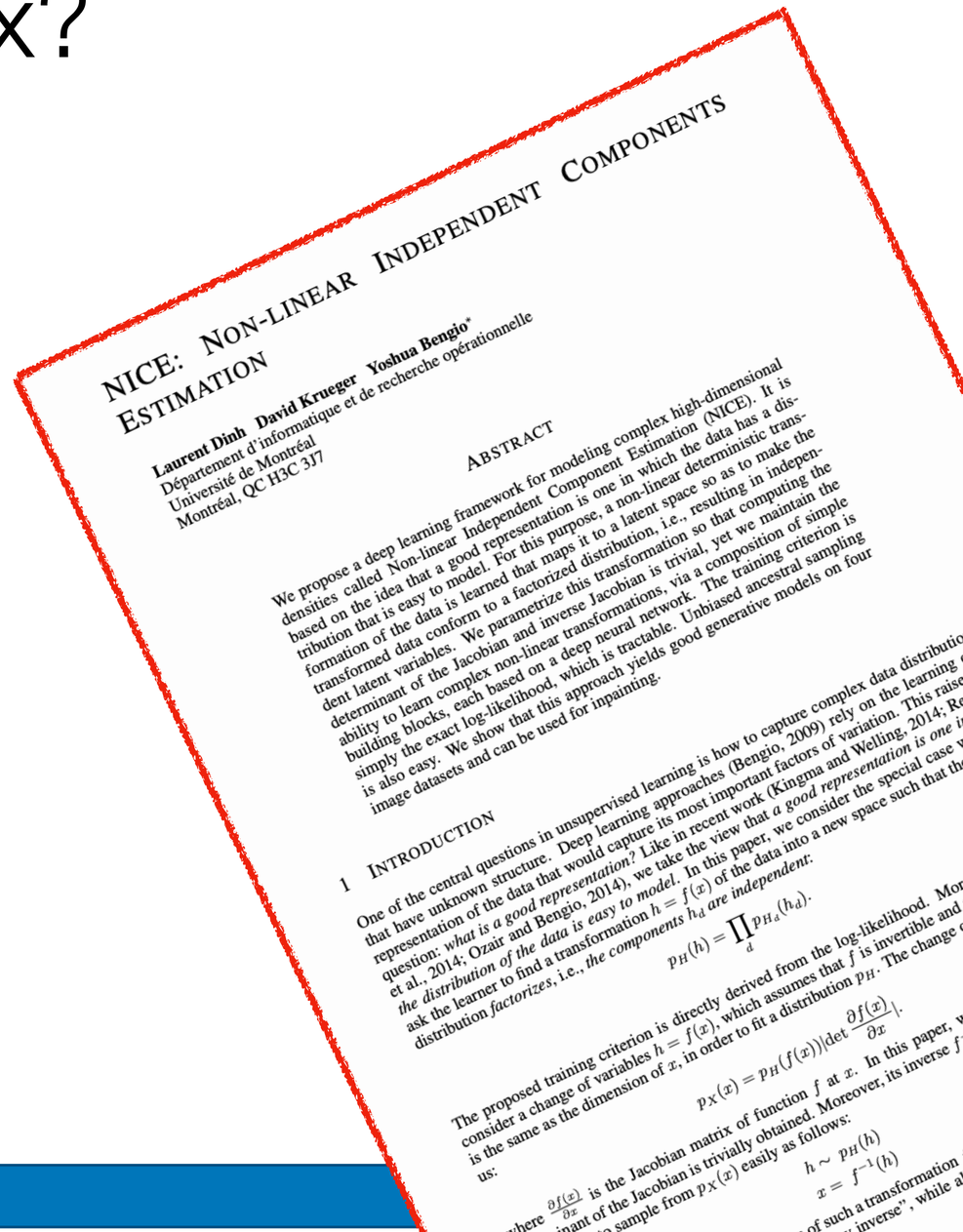
$f : Z \to X, f$ is invertible

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

$$\mathbf{J} = \begin{bmatrix} \ell_{1,1} & & & & 0 \\ \ell_{2,1} & \ell_{2,2} & & & \\ \ell_{3,1} & \ell_{3,2} & \ddots & & \\ \vdots & \vdots & \ddots & \ddots & \\ \ell_{n,1} & \ell_{n,2} & \cdots & \ell_{n,n-1} & \ell_{n,n} \end{bmatrix}$$

How to ensure lower-triangular Jacobian matrix?

$$z \in \mathbb{R}^D$$



NICE: NON-LINEAR INDEPENDENT COMPONENTS ESTIMATION

Laurent Dinh **David Krueger** **Yoshua Bengio***
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

ABSTRACT

We propose a deep learning framework for modeling complex high-dimensional densities called Non-linear Independent Component Estimation (NICE). It is based on the idea that a good representation is one in which the data has a distribution that is easy to model. For this purpose, a non-linear deterministic transformation of the data is learned that maps it to a latent space so as to make the transformed data conform to a factorized distribution, i.e., resulting in independent latent variables. We parametrize this transformation so that computing the determinant of the Jacobian and inverse Jacobian is trivial, yet we maintain the ability to learn complex non-linear transformations, via a composition of simple building blocks, each based on a deep neural network. The training criterion is simply the exact log-likelihood, which is tractable. Unbiased ancestral sampling is also easy. We show that this approach yields good generative models on four image datasets and can be used for inpainting.

1 INTRODUCTION

One of the central questions in unsupervised learning is how to capture complex data distributio
that have unknown structure. Deep learning approaches (Bengio, 2009) rely on the learning
representation of the data that would capture its most important factors of variation. This raise
question: *what is a good representation?* Like in recent work (Kingma and Welling, 2014; Re
et al., 2014; Ozair and Bengio, 2014), we take the view that *a good representation is one i*
the distribution of the data is easy to model. In this paper, we consider the special case w
ask the learner to find a transformation $h = f(x)$ of the data into a new space such that th
distribution *factorizes*, i.e., *the components* h_d *are independent*.

$$p_H(h) = \prod_d p_{H_d}(h_d).$$

The proposed training criterion is directly derived from the log-likelihood. Mo
consider a change of variables $h = f(x)$, which assumes that f is invertible and
is the same as the dimension of x, in order to fit a distribution p_H. The change
us:

$$p_X(x) = p_H(f(x))|\det \frac{\partial f(x)}{\partial x}|.$$

here $\frac{\partial f(x)}{\partial x}$ is the Jacobian matrix of function f at x. In this paper,
ninant of the Jacobian is trivially obtained. Moreover, its inverse f
o sample from $p_X(x)$ easily as follows:

$$h \sim p_H(h)$$
$$x = f^{-1}(h)$$

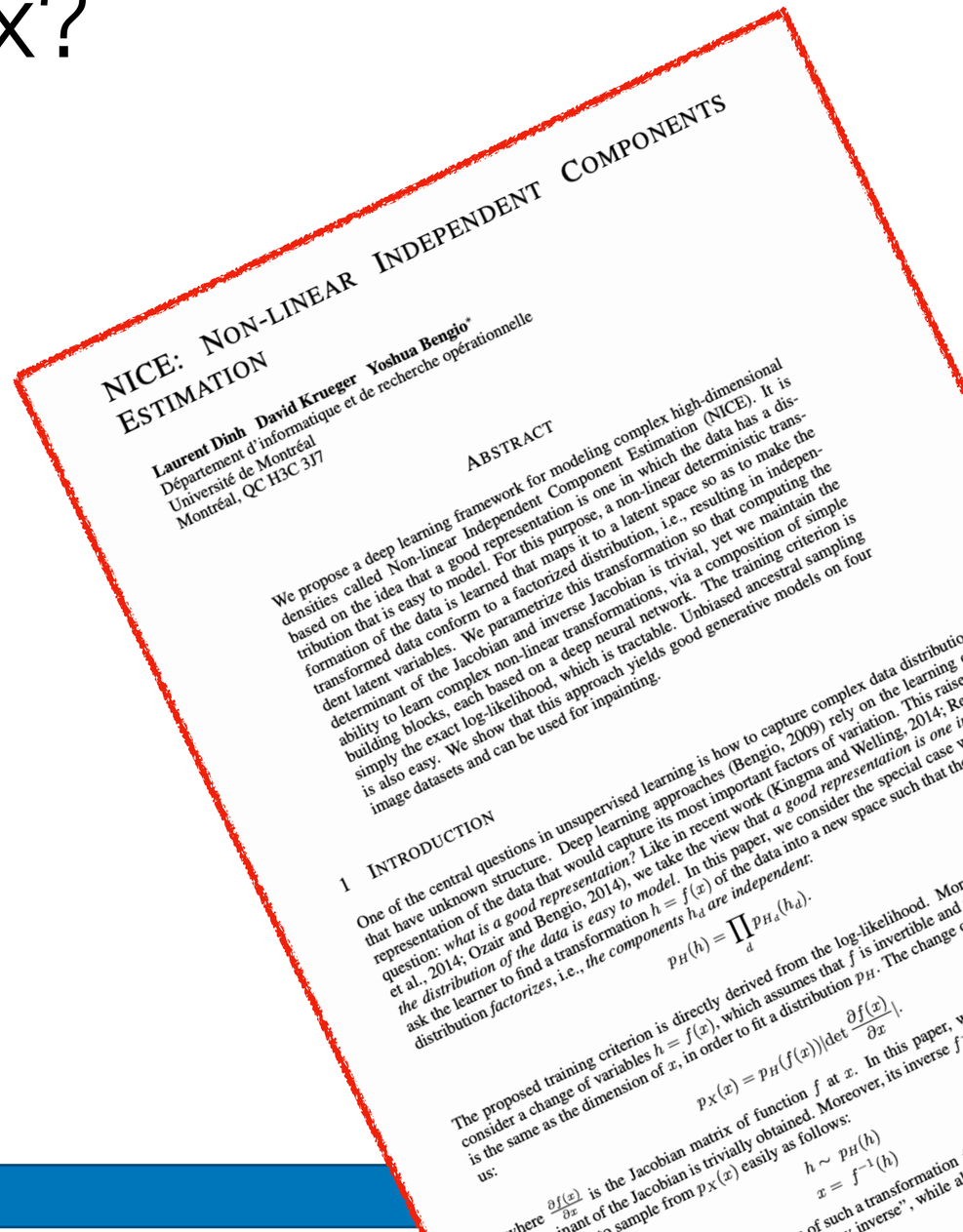
of such a transformation
inverse", while a

How to ensure lower-triangular Jacobian matrix?

$$z \in \mathbb{R}^D$$

$$z_{1:d}$$

$$z_{d+1:D}$$



NICE: NON-LINEAR INDEPENDENT COMPONENTS ESTIMATION

Laurent Dinh **David Krueger** **Yoshua Bengio***
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

ABSTRACT

We propose a deep learning framework for modeling complex high-dimensional densities called Non-linear Independent Component Estimation (NICE). It is based on the idea that a good representation is one in which the data has a distribution that is easy to model. For this purpose, a non-linear deterministic transformation of the data is learned that maps it to a latent space so as to make the transformed data conform to a factorized distribution, i.e., resulting in independent latent variables. We parametrize this transformation so that computing the determinant of the Jacobian and inverse Jacobian is trivial, yet we maintain the ability to learn complex non-linear transformations, via a composition of simple building blocks, each based on a deep neural network. The training criterion is simply the exact log-likelihood, which is tractable. Unbiased ancestral sampling is also easy. We show that this approach yields good generative models on four image datasets and can be used for inpainting.

1 INTRODUCTION

One of the central questions in unsupervised learning is how to capture complex data distributio
that have unknown structure. Deep learning approaches (Bengio, 2009) rely on the learning
representation of the data that would capture its most important factors of variation. This raise
question: *what is a good representation?* Like in recent work (Kingma and Welling, 2014; Re
et al., 2014; Ozair and Bengio, 2014), we take the view that *a good representation is one i*
the distribution of the data is easy to model. In this paper, we consider the special case w
ask the learner to find a transformation $h = f(x)$ of the data into a new space such that th
distribution *factorizes*, i.e., *the components* h_d *are independent*.

$$p_H(h) = \prod_d p_{H_d}(h_d).$$

The proposed training criterion is directly derived from the log-likelihood. Mo
consider a change of variables $h = f(x)$, which assumes that f is invertible and
is the same as the dimension of x, in order to fit a distribution p_H. The change o
us:

$$p_X(x) = p_H(f(x))|\det \frac{\partial f(x)}{\partial x}|.$$

here $\frac{\partial f(x)}{\partial x}$ is the Jacobian matrix of function f at x. In this paper, w
ignant of the Jacobian is trivially obtained. Moreover, its inverse f
o sample from $p_X(x)$ easily as follows:

$$h \sim p_H(h)$$
$$x = f^{-1}(h)$$

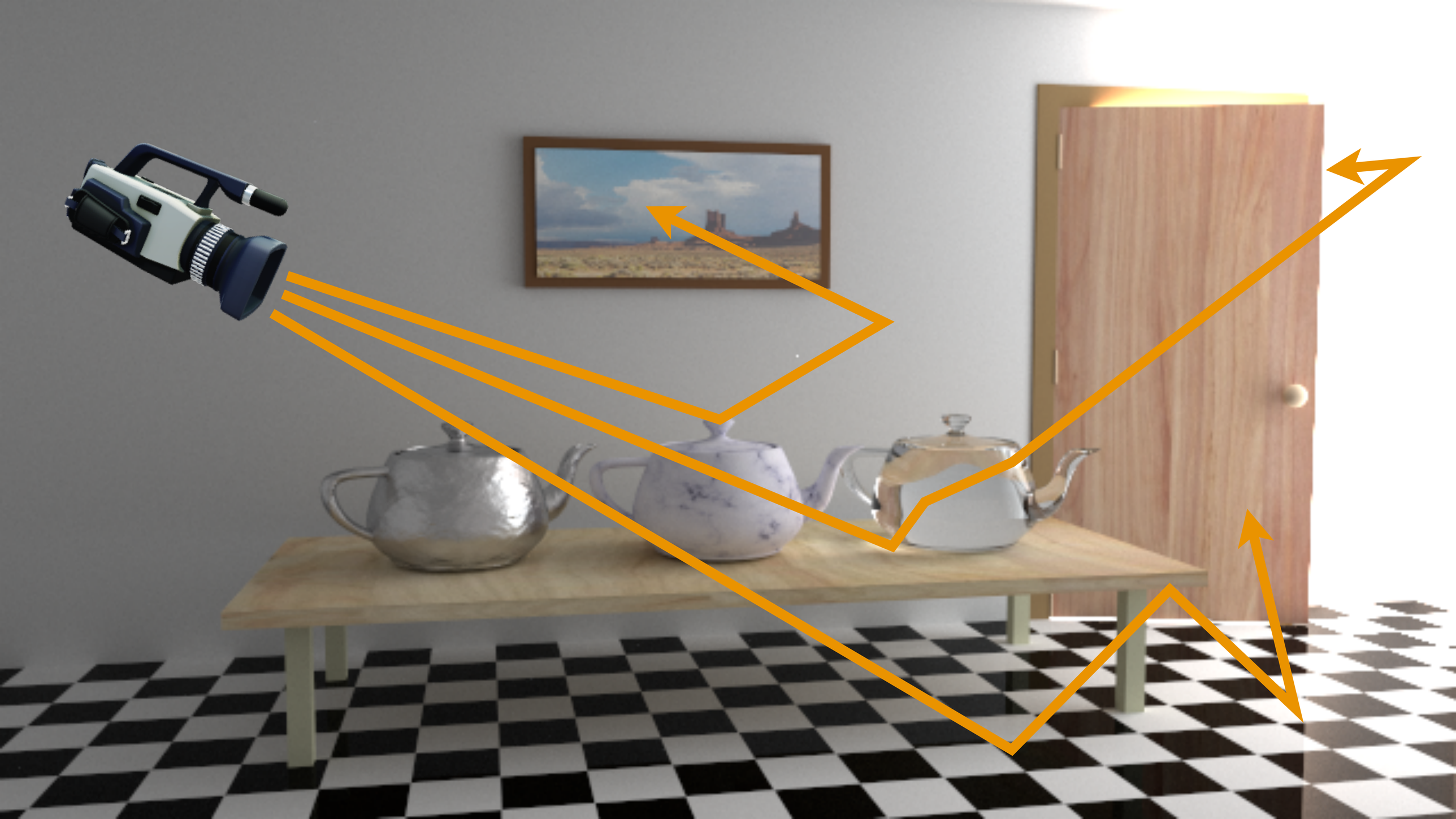
of such a transformation
inverse", while a

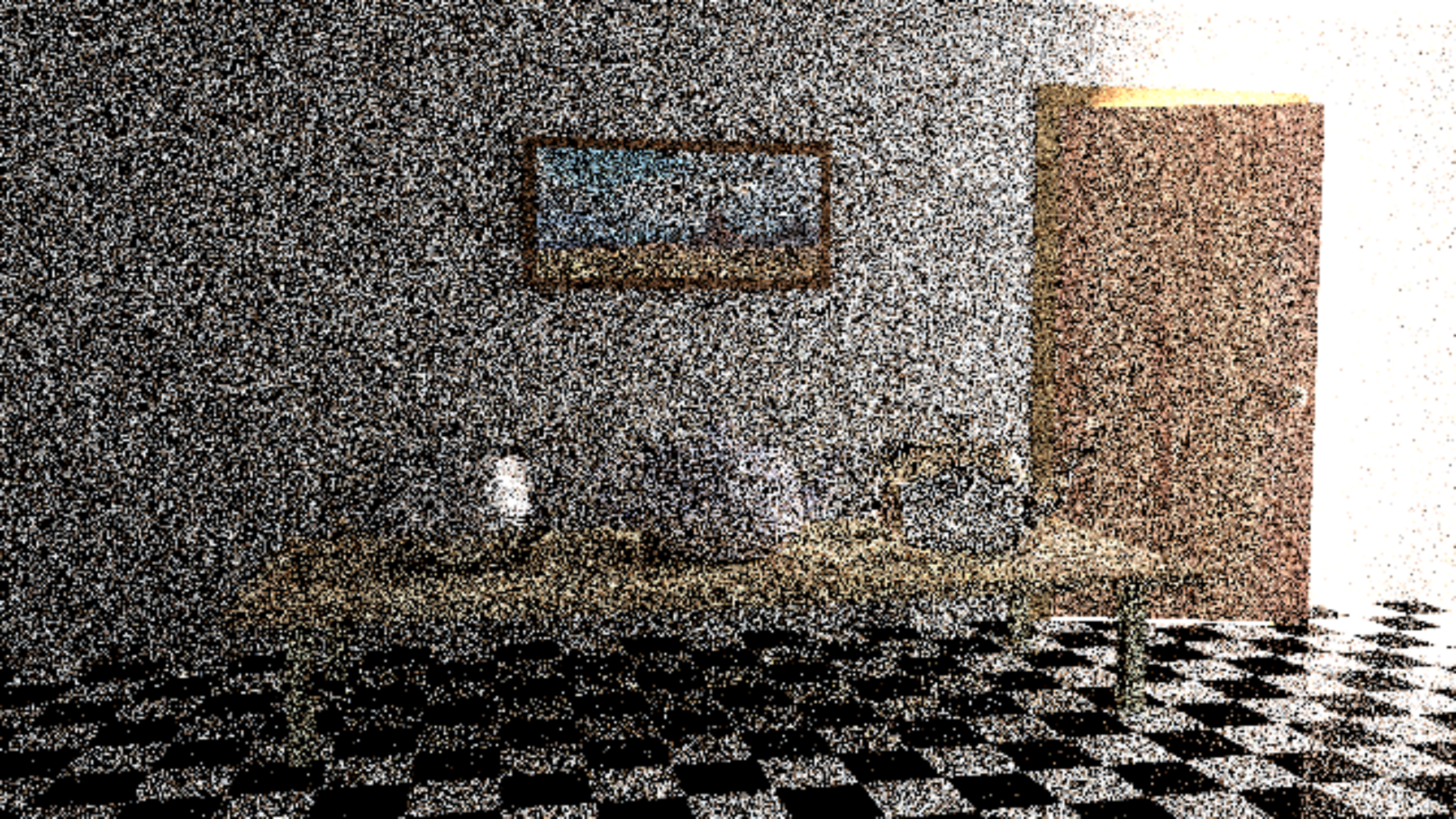
Neural Importance Sampling

Thomas Müller
Brian McWilliams
Fabrice Rousselle
Markus Gross
Jan Novák

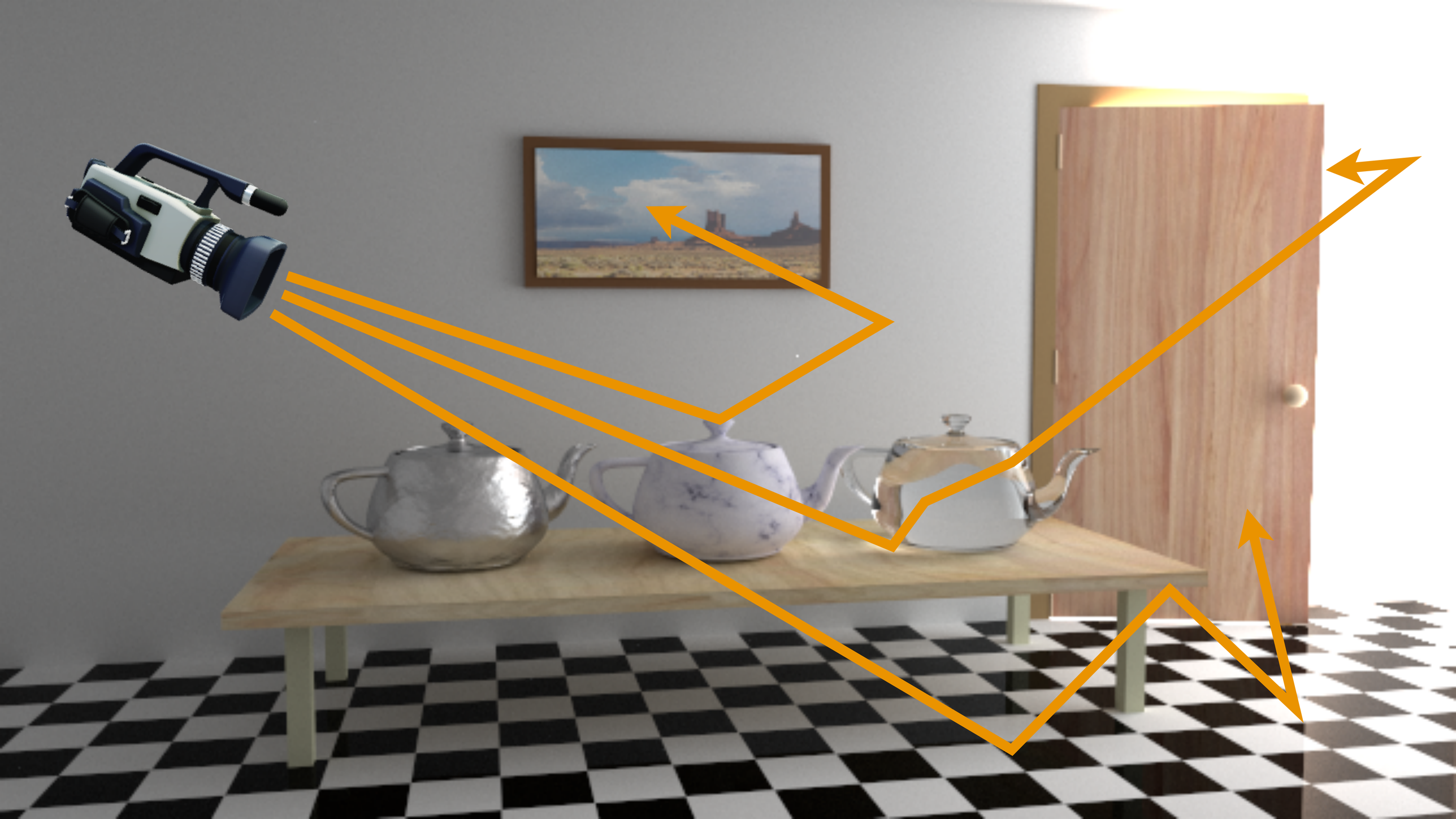
Affiliation: NVIDIA

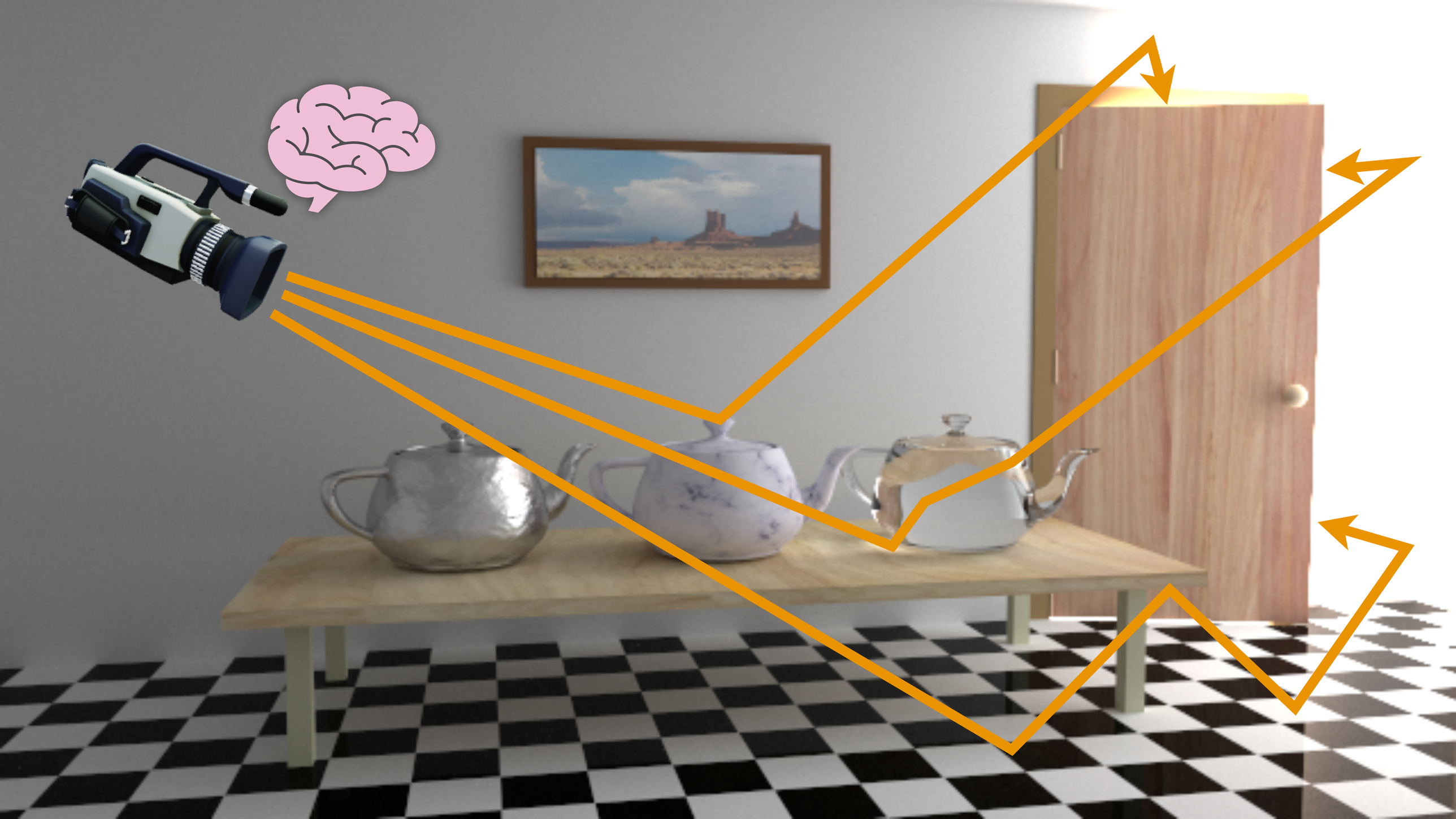
Work done while at: ETH zürich, Disney Research, cgl computer graphics laboratory

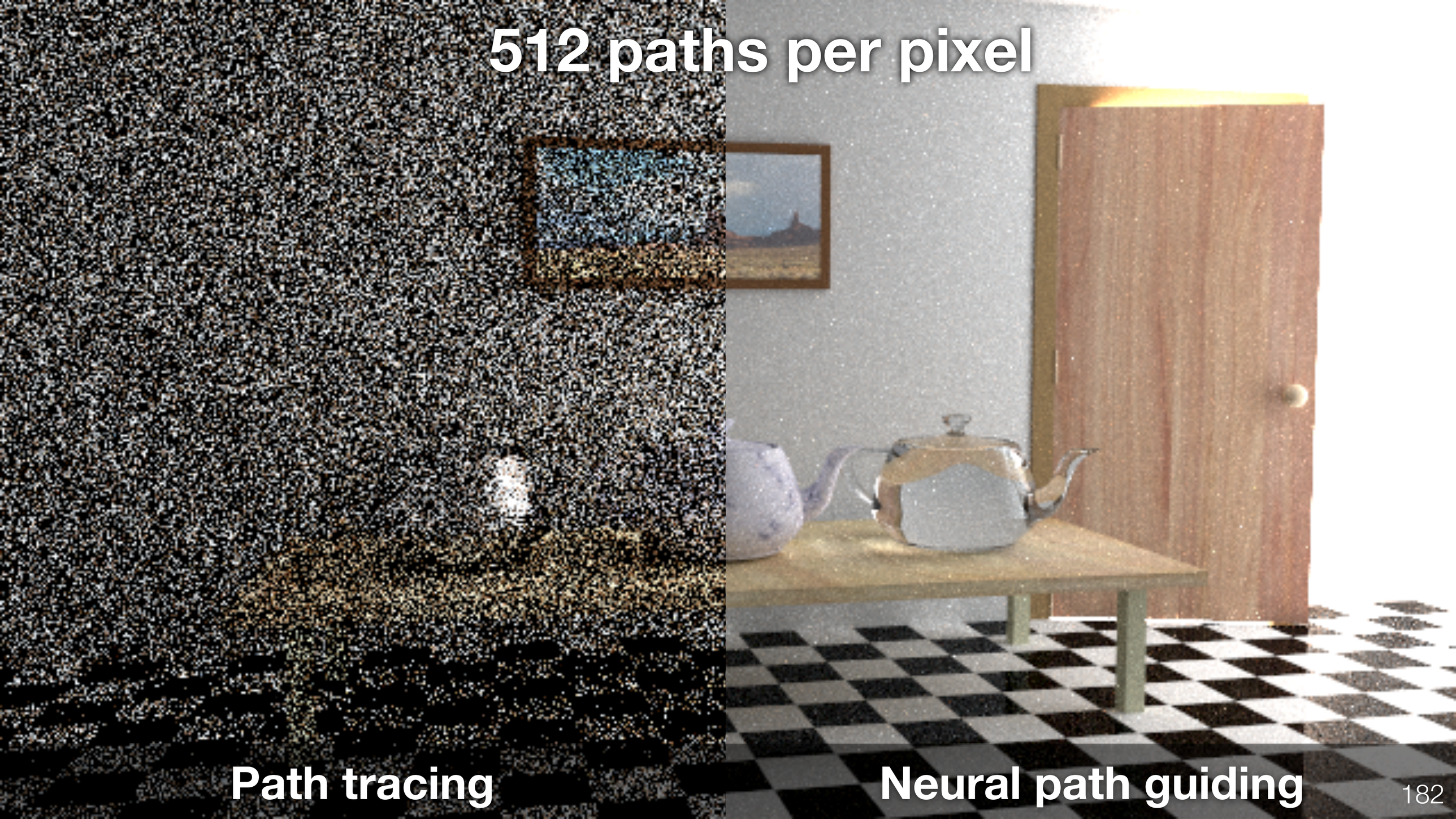




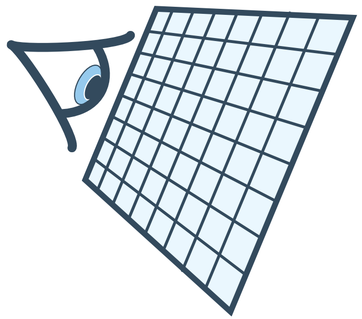

Render time: sometimes >100 cpu-hours



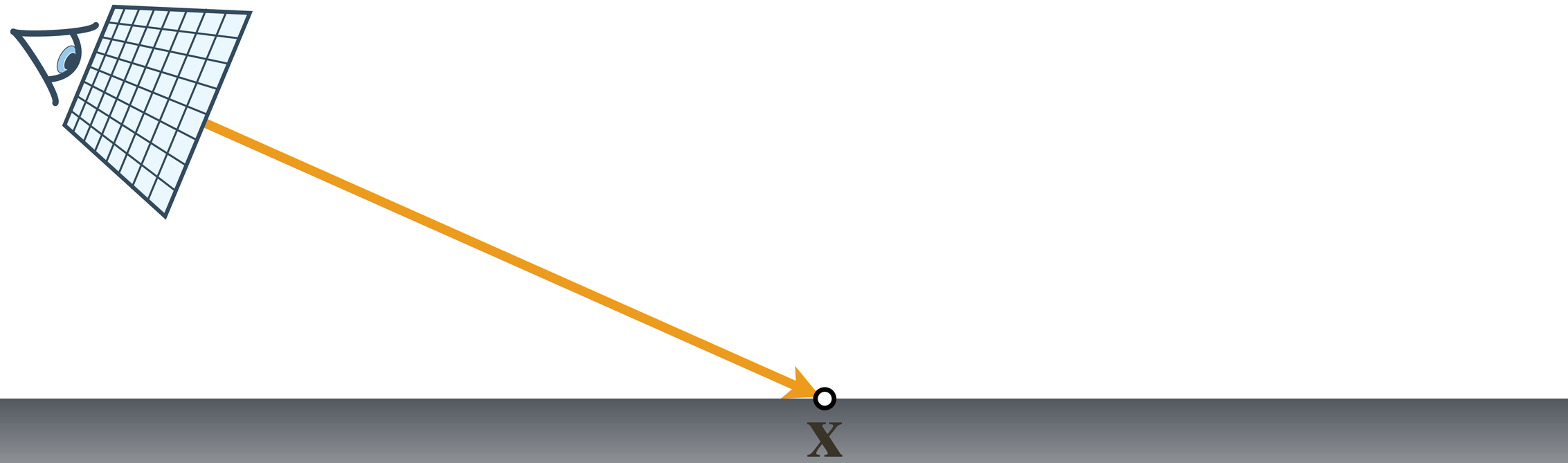



512 paths per pixel
Path tracing
Neural path guiding

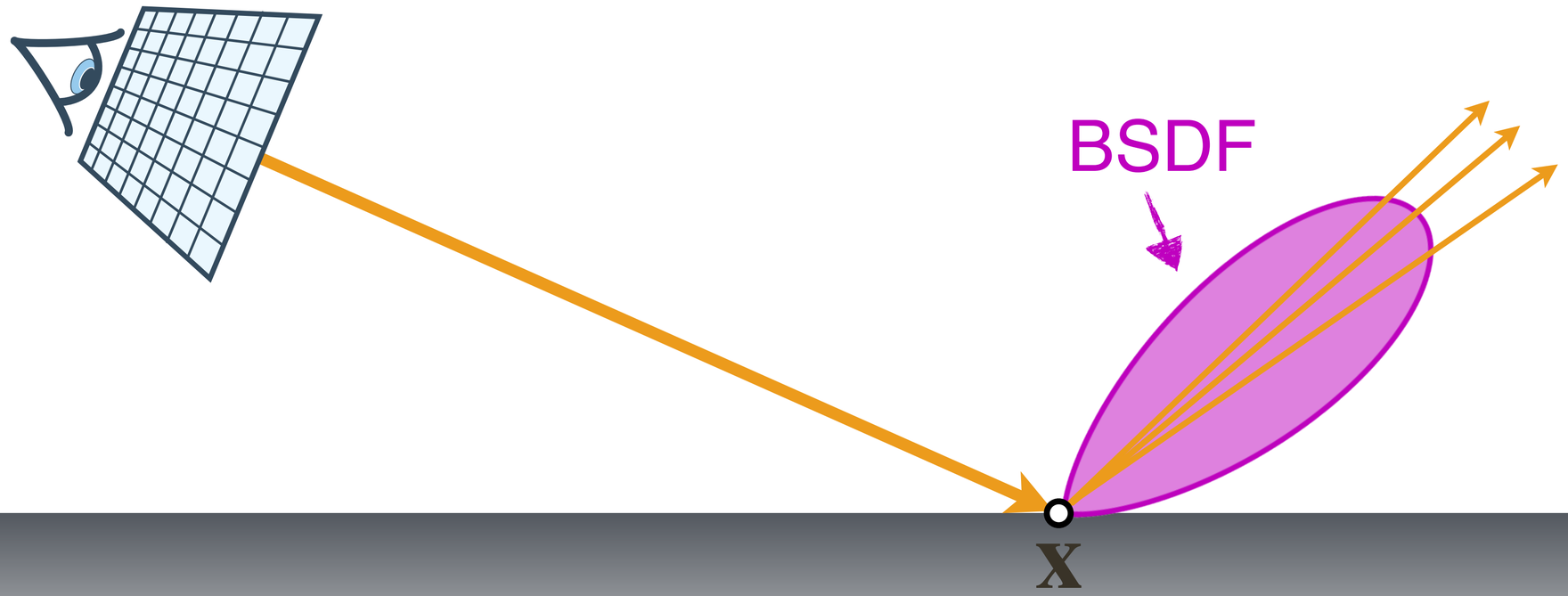
Path tracing: BSDF sampling



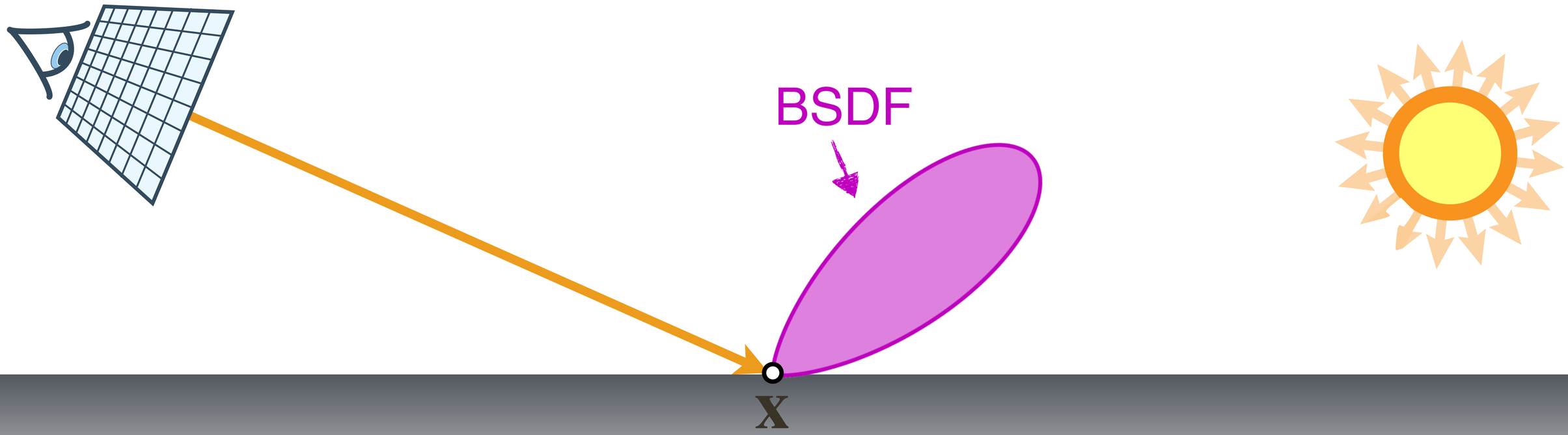
Path tracing: BSDF sampling



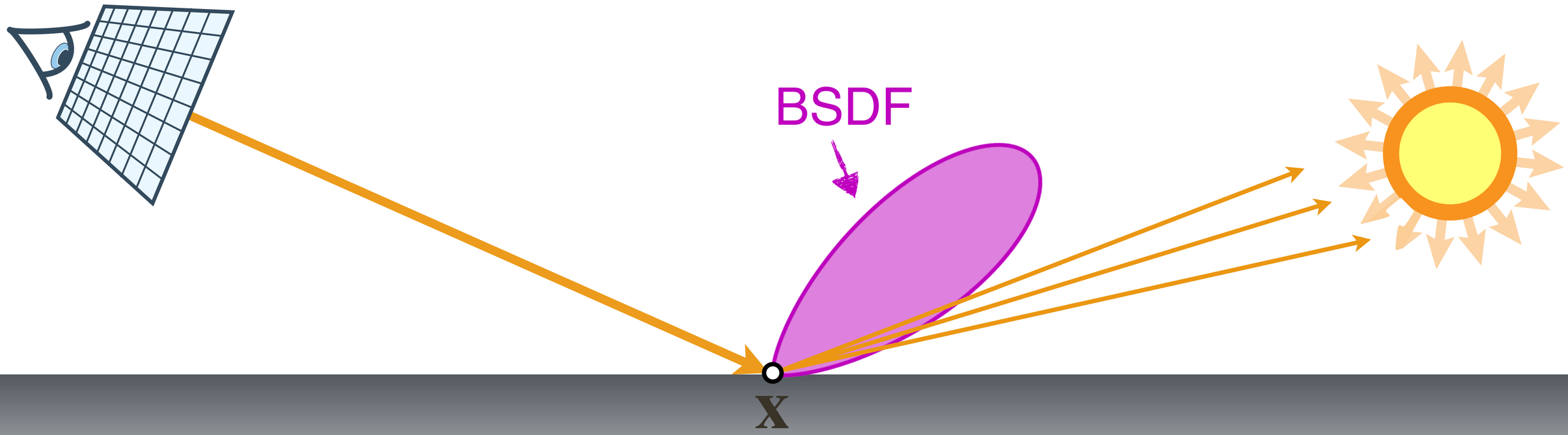
Path tracing: BSDF sampling



Path tracing: direct-illumination sampling

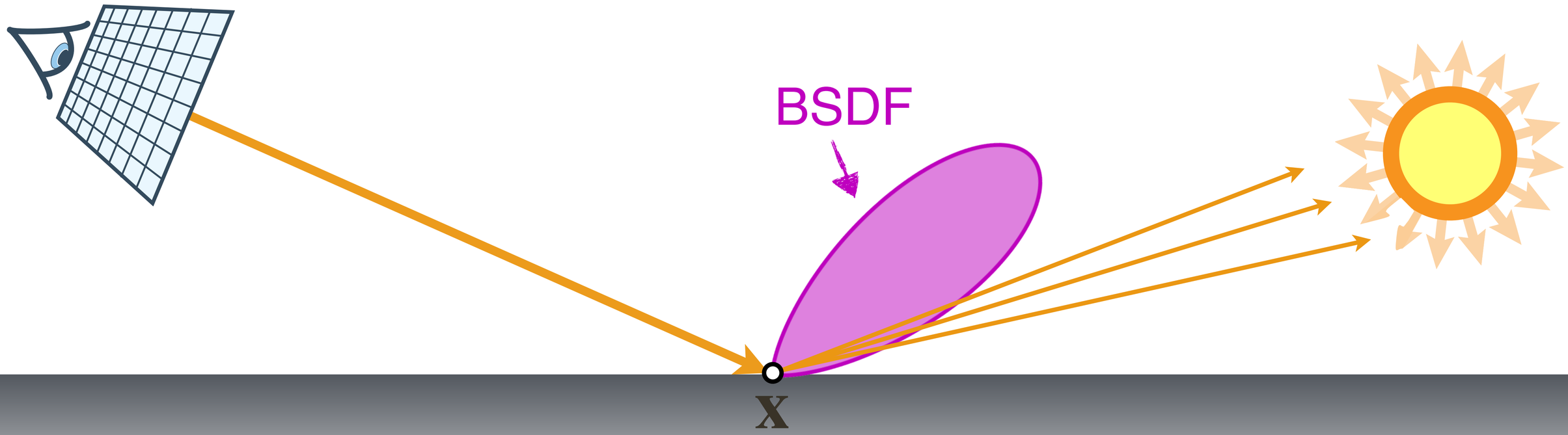


Path tracing: direct-illumination sampling

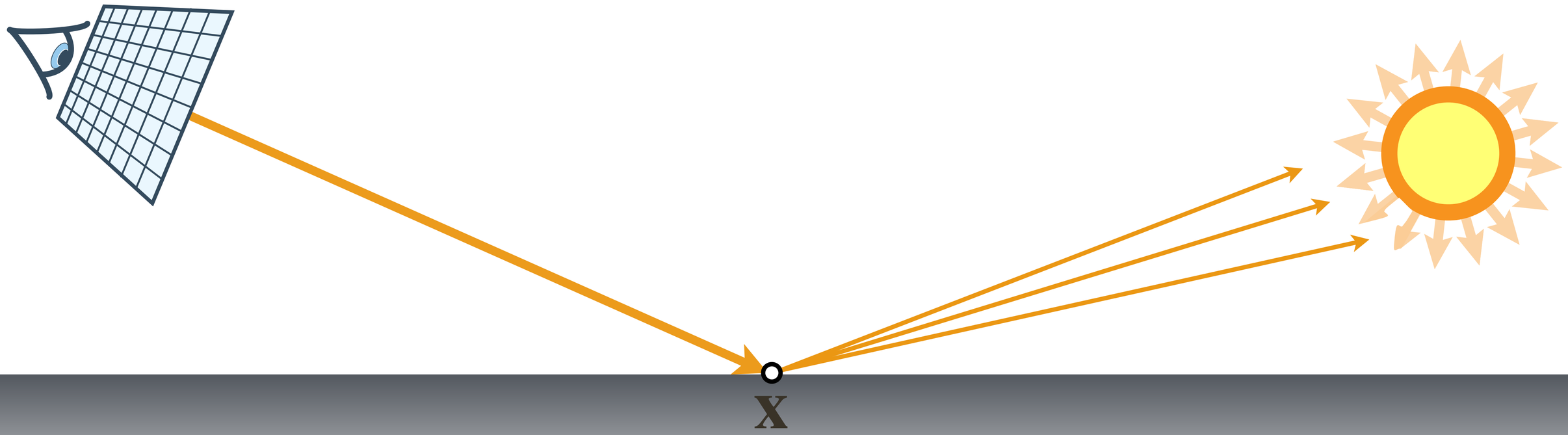


Path tracing: direct-illumination sampling

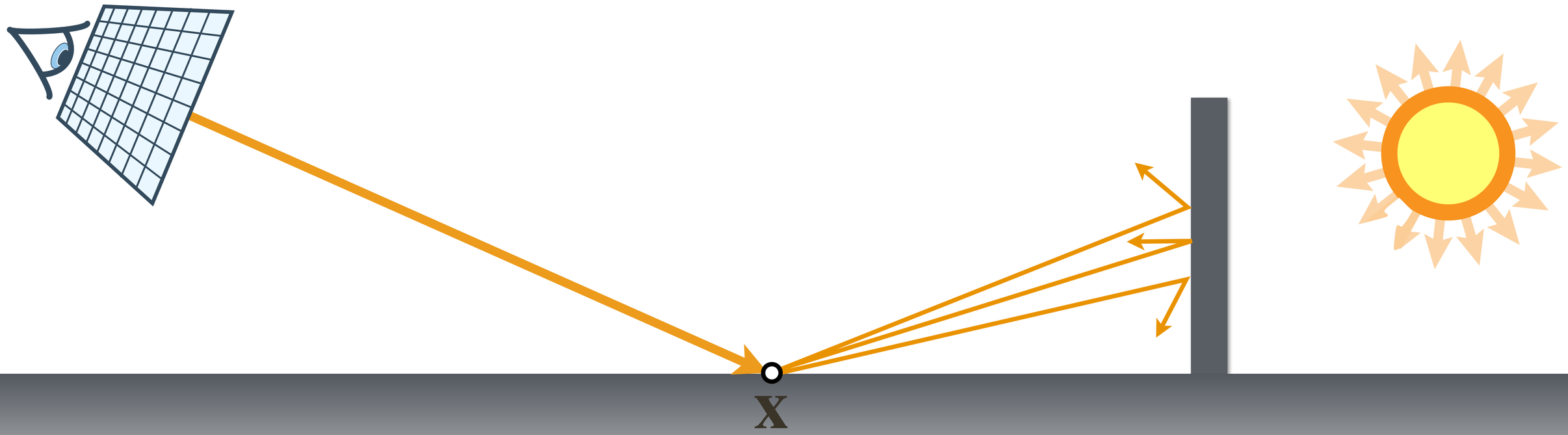
Multiple Importance Sampling
[Veach and Guibas 1995]



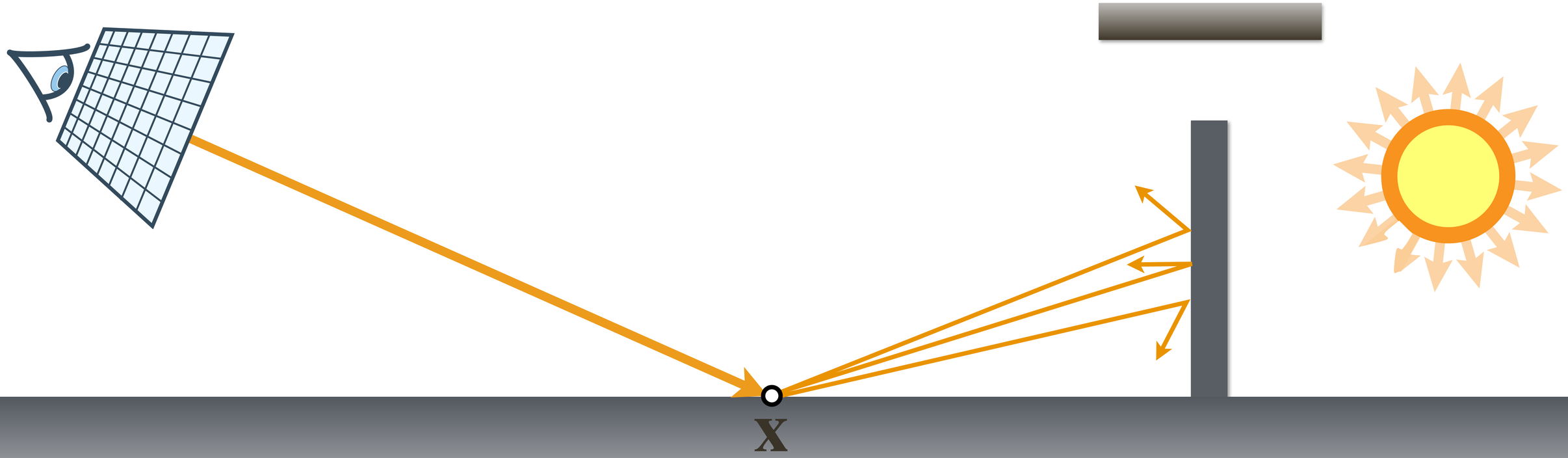
Where is path guiding useful?



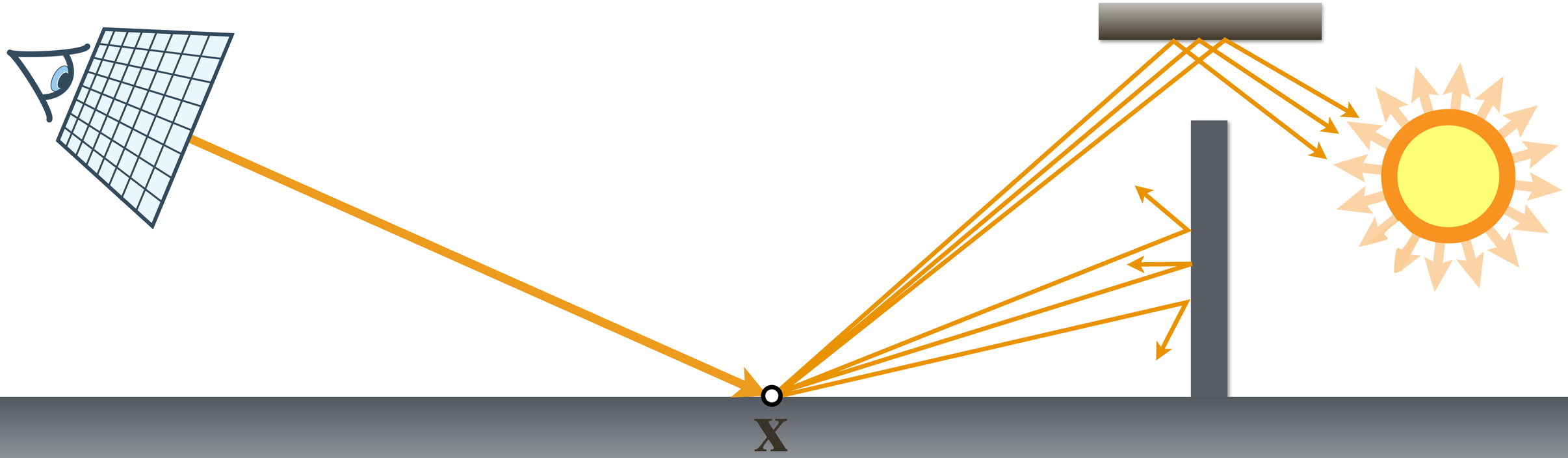
Where is path guiding useful?



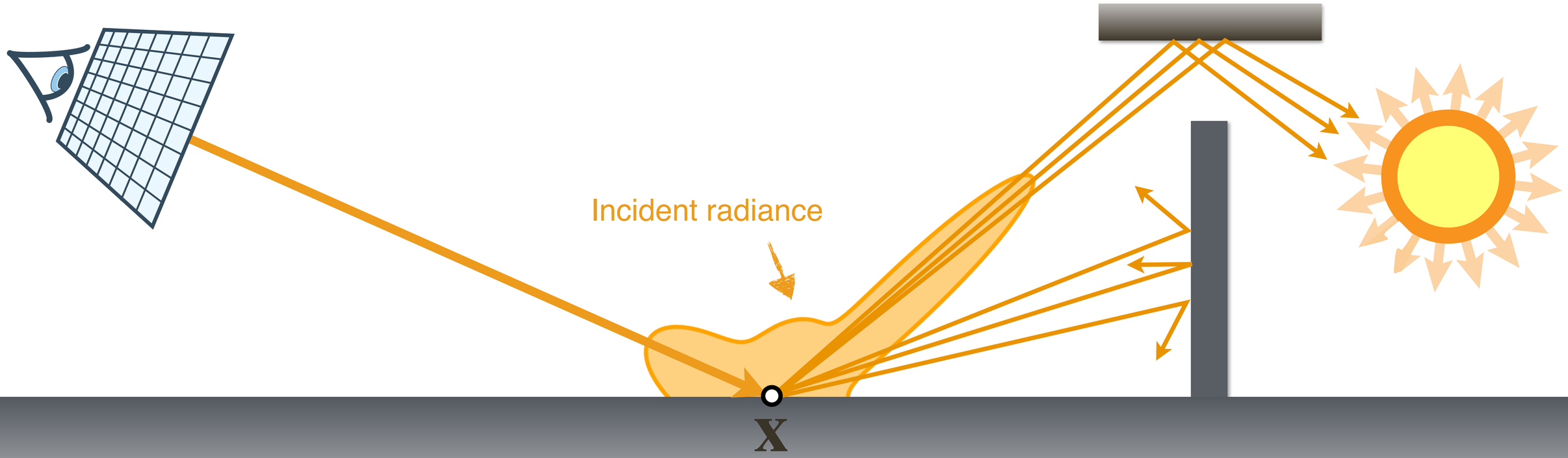
Where is path guiding useful?



Where is path guiding useful?

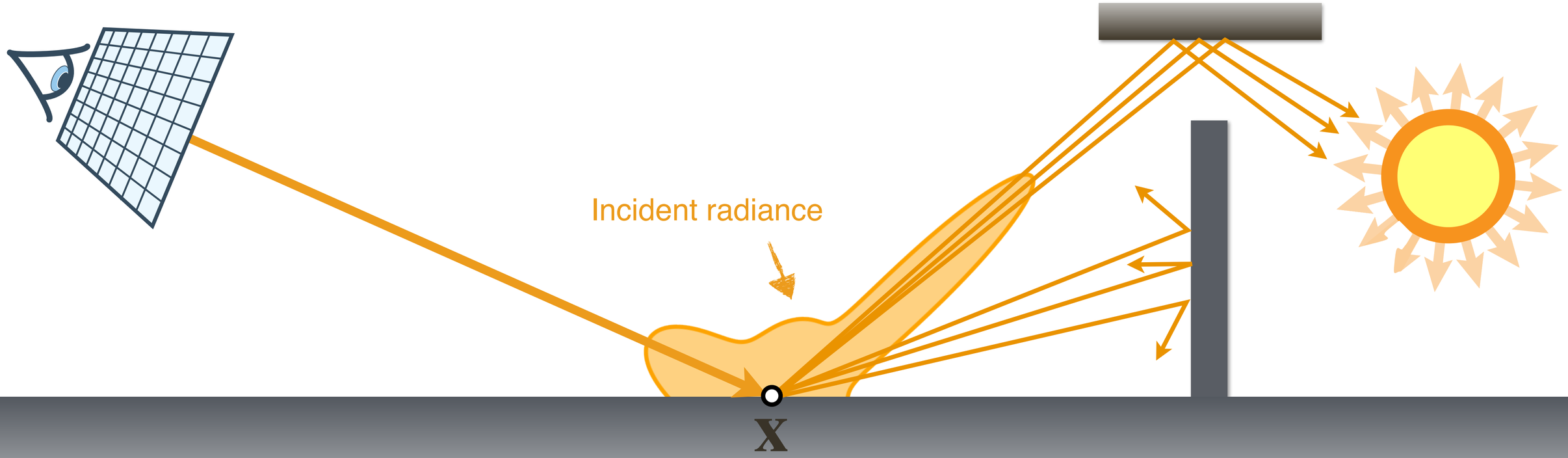


Where is path guiding useful?

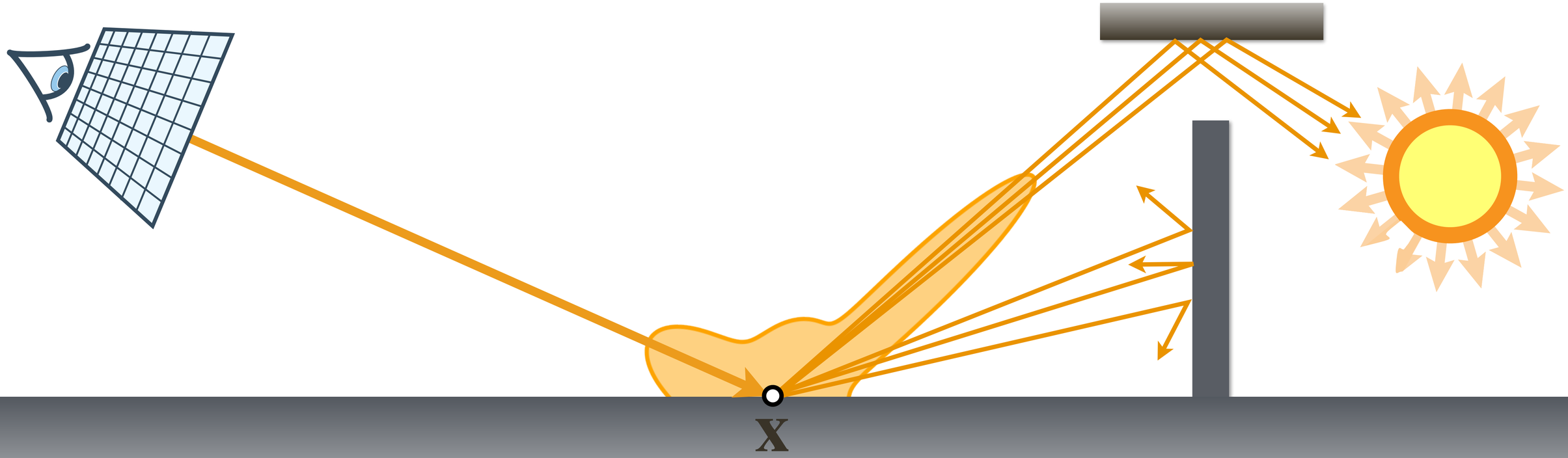


Where is path guiding useful?

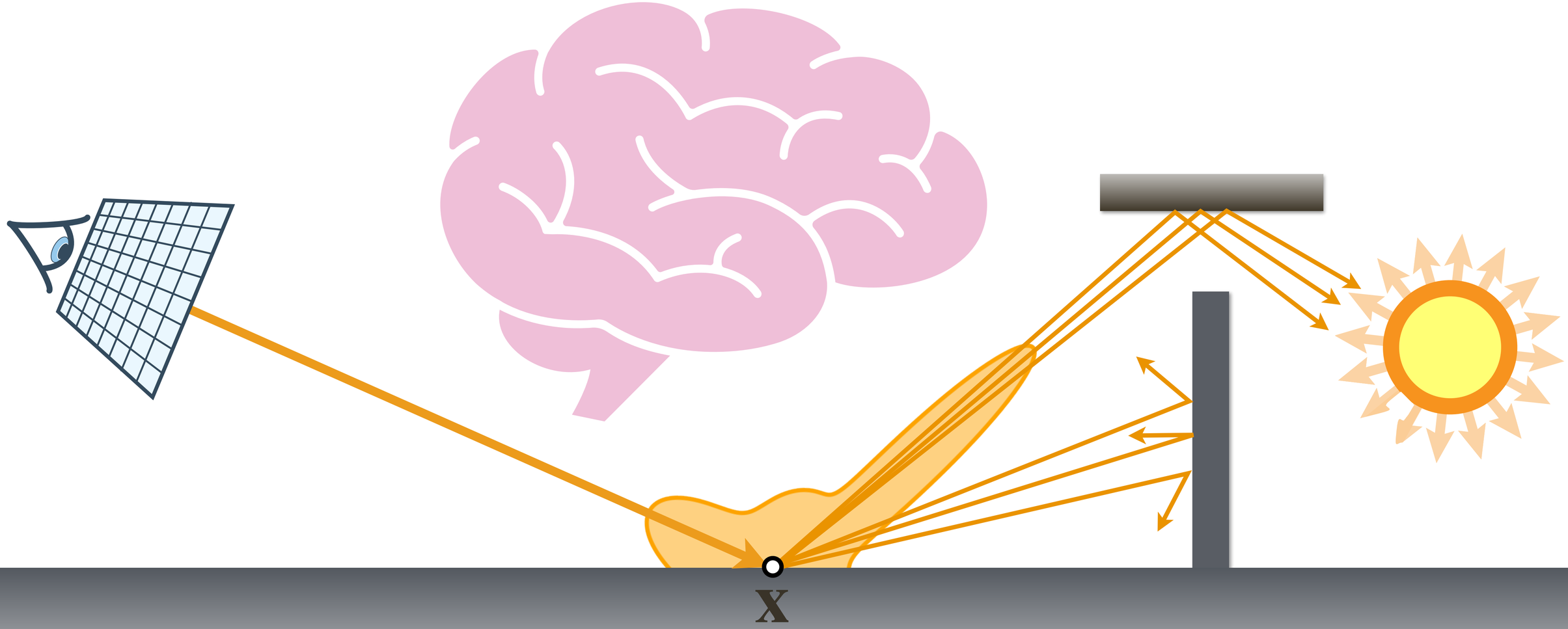
Goal: Sample proportional to incident radiance.



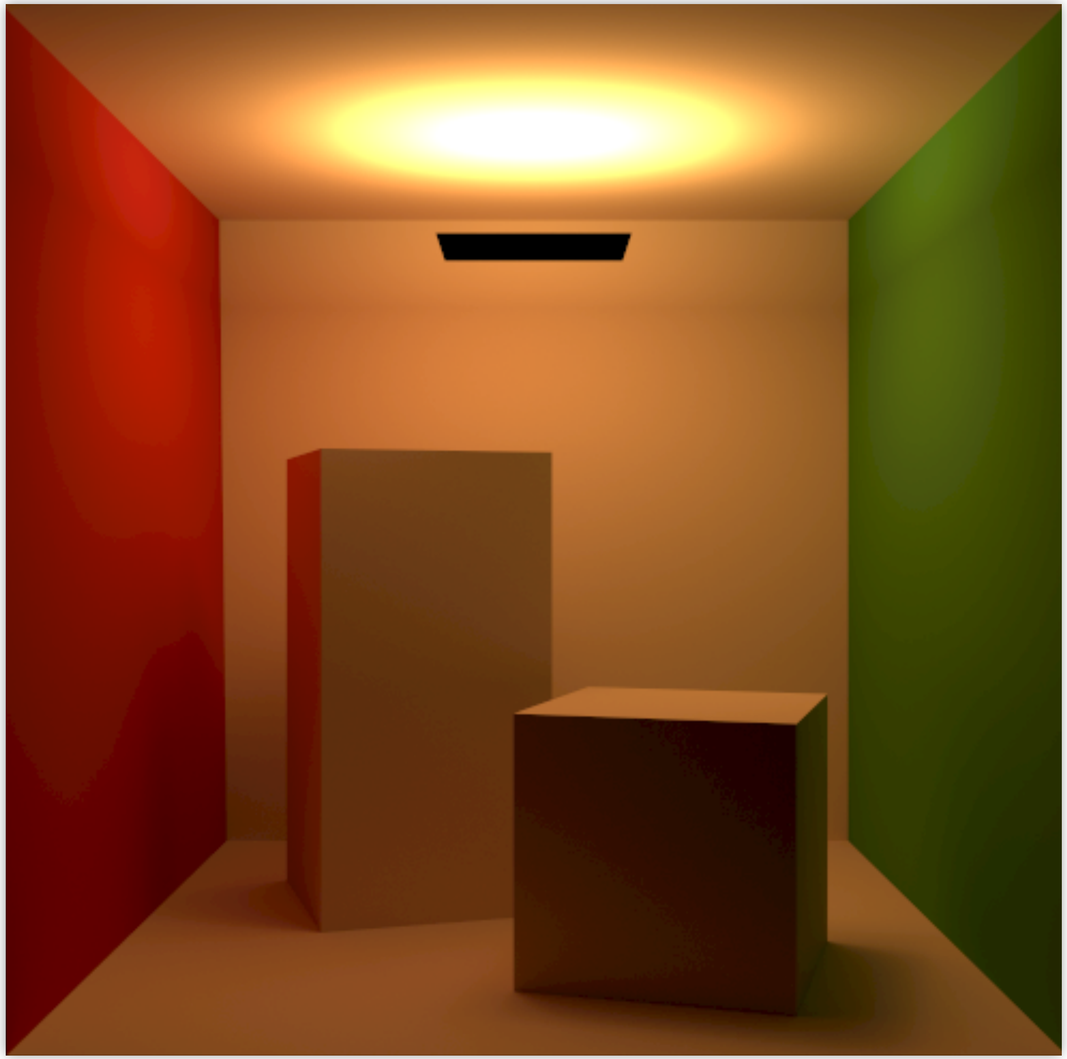
Where is path guiding useful?



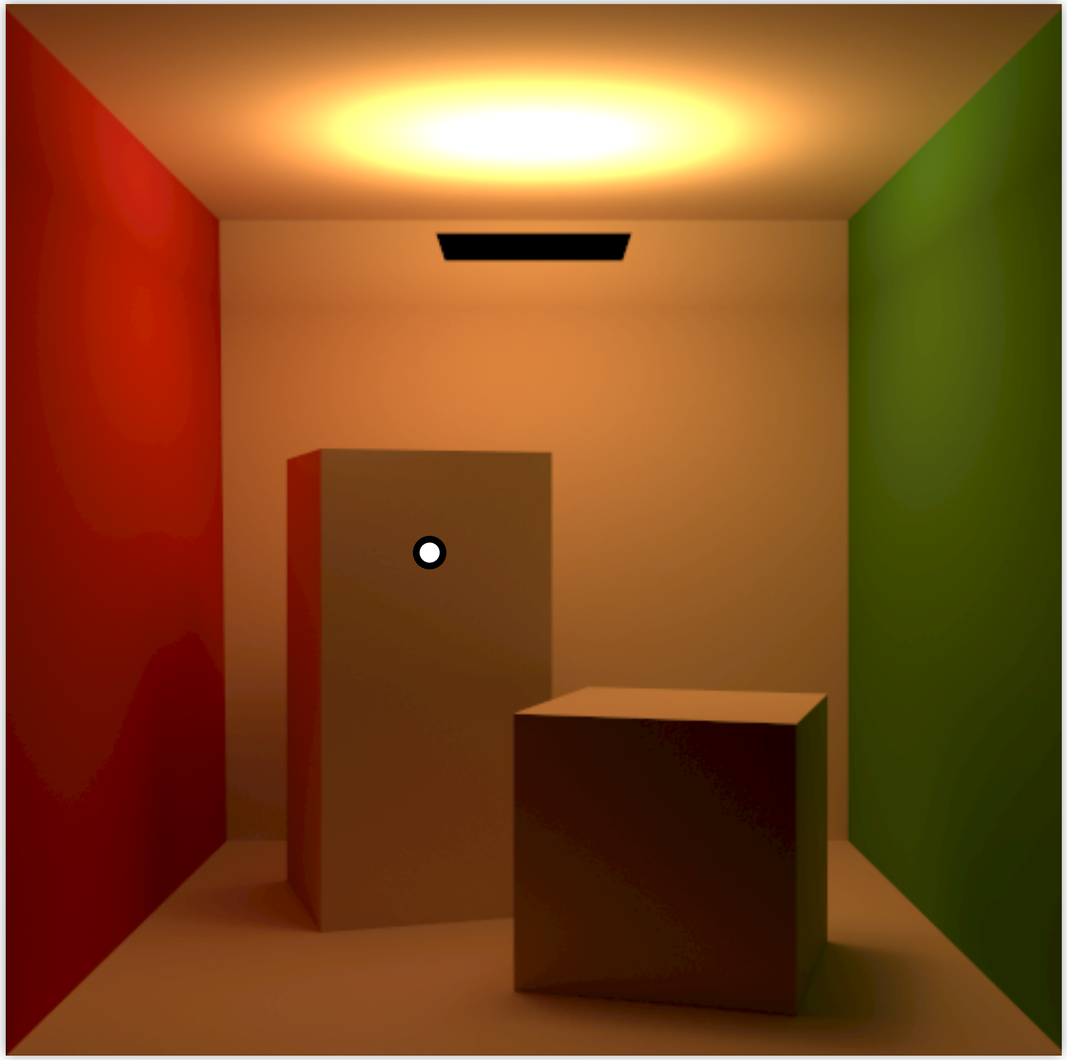
Where is path guiding useful?



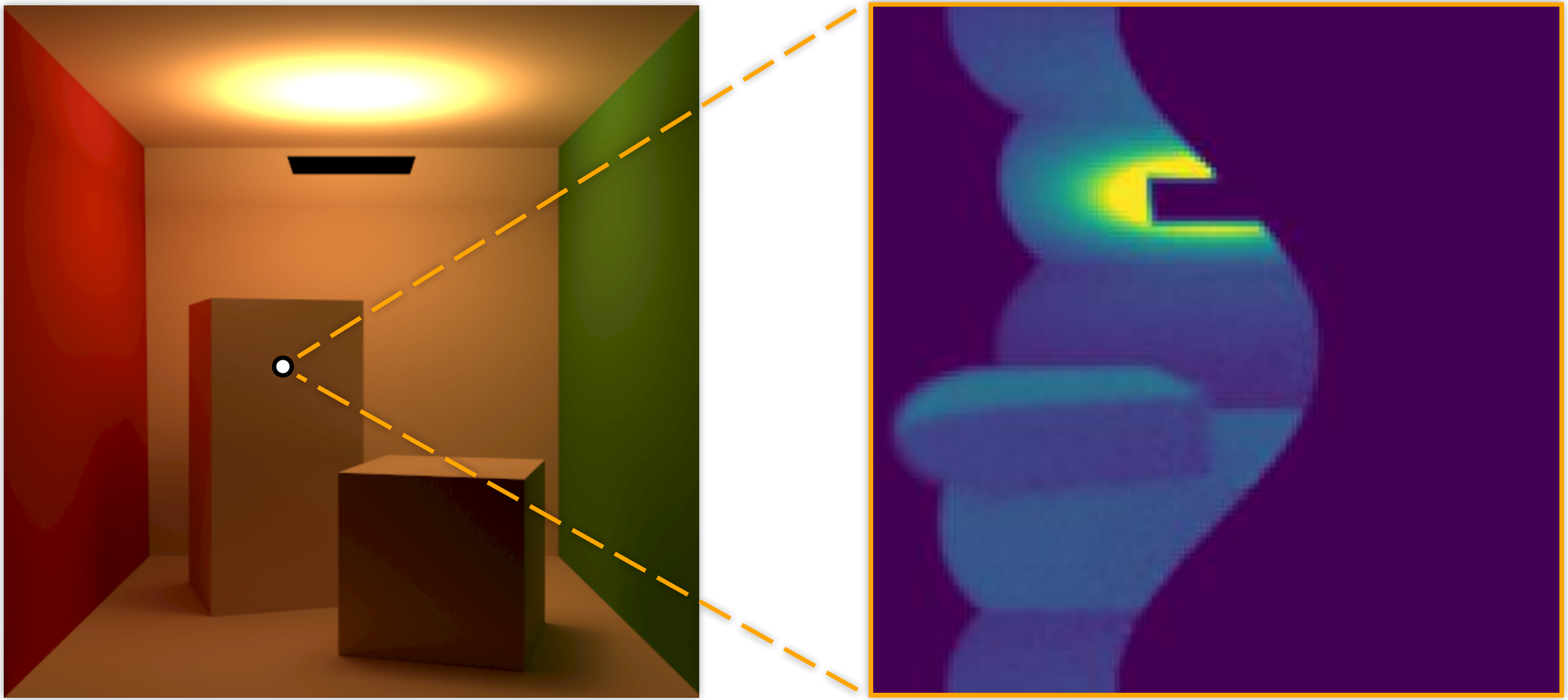
Learning incident radiance in a Cornell box



Learning incident radiance in a Cornell box

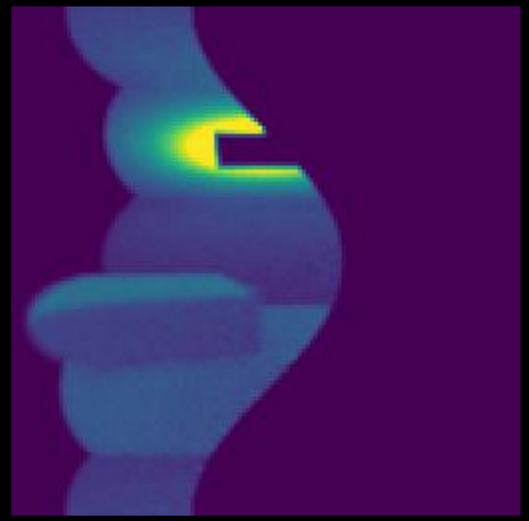


Learning incident radiance in a Cornell box



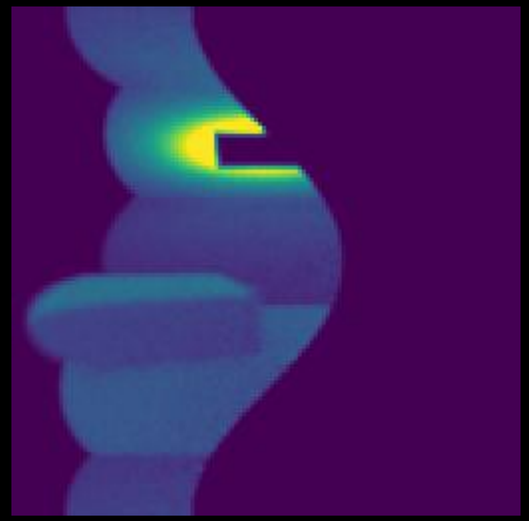
Neural networks as function approximators

Reference

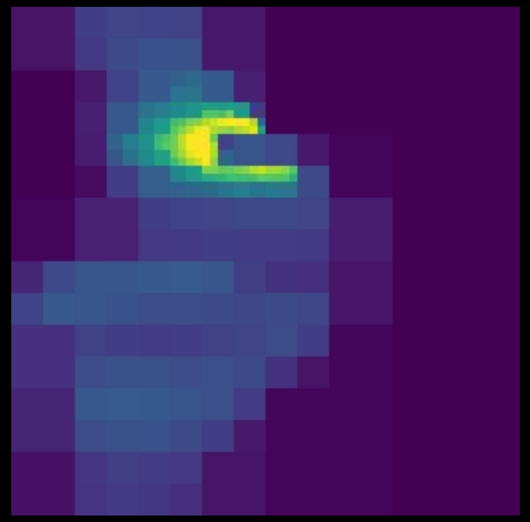


Neural networks as function approximators

Reference

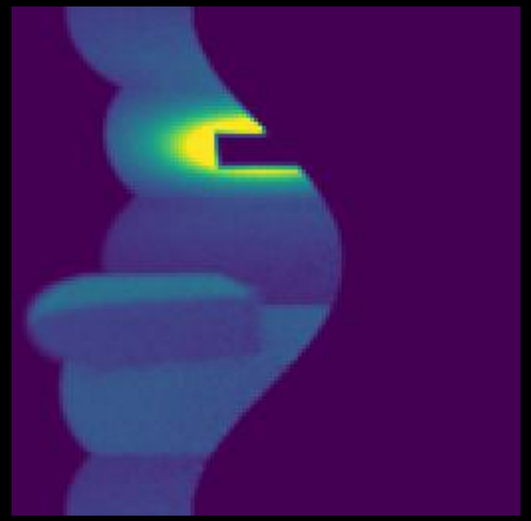


SD-tree [Müller et al. 2017]

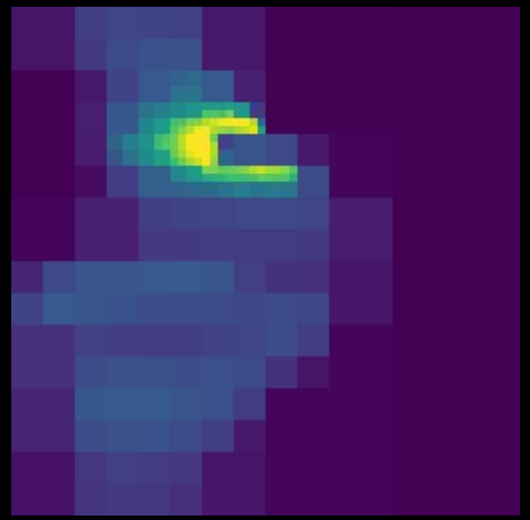


Neural networks as function approximators

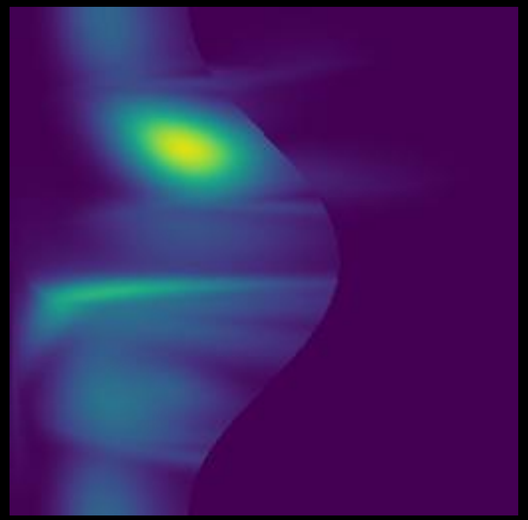
Reference



SD-tree [Müller et al. 2017]

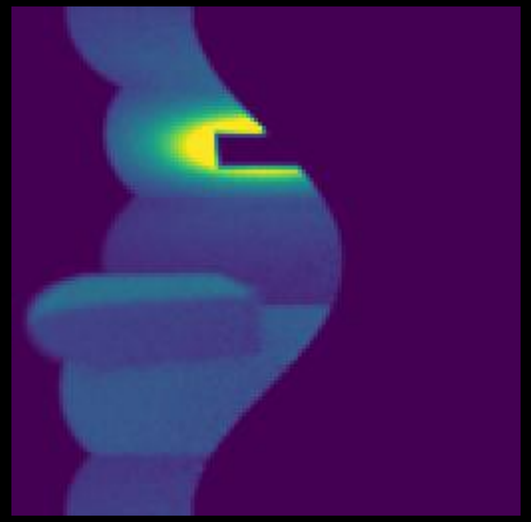


GMM [Vorba et al. 2014]

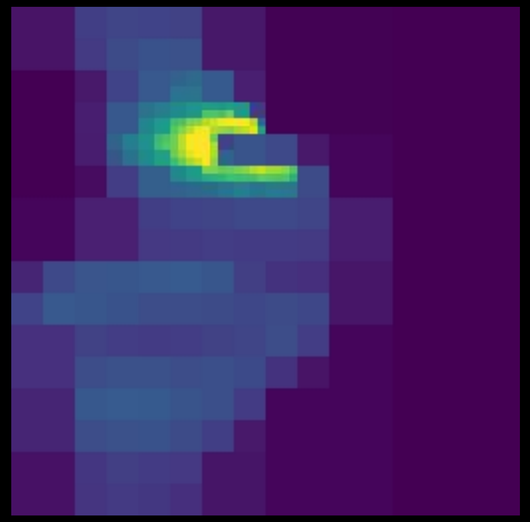


Neural networks as function approximators

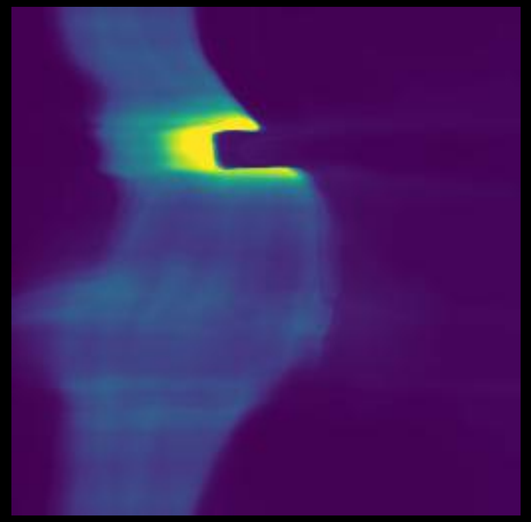
Reference



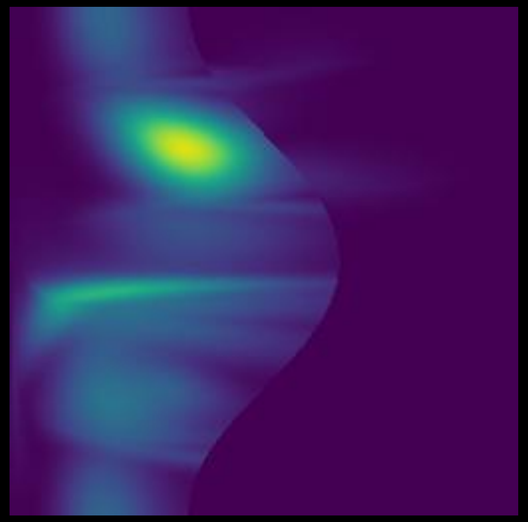
SD-tree [Müller et al. 2017]

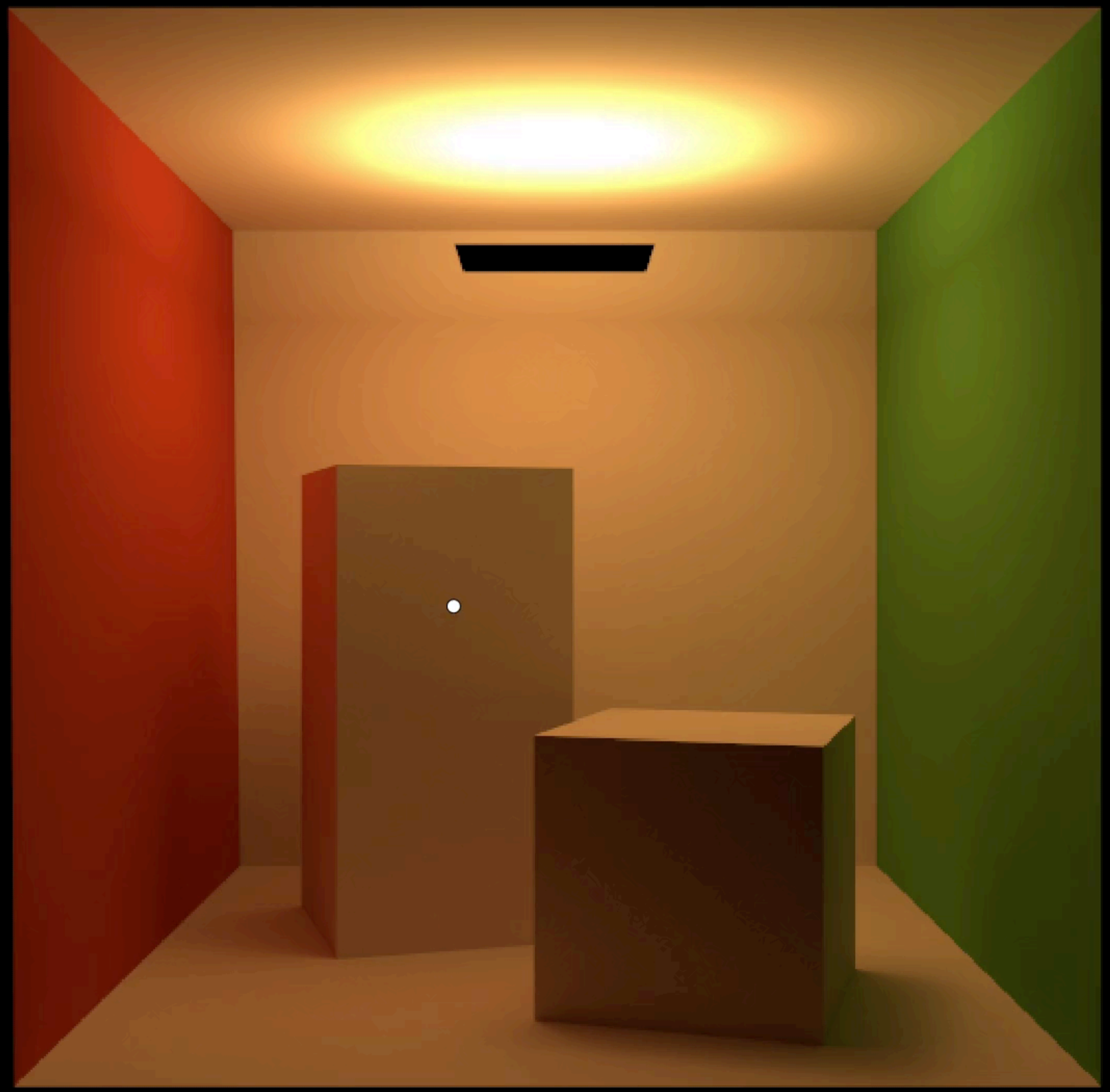


Neural Network



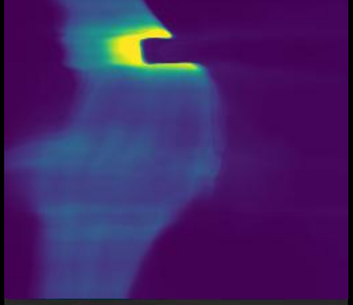
GMM [Vorba et al. 2014]



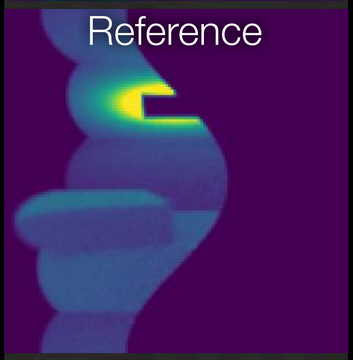


Directional distribution

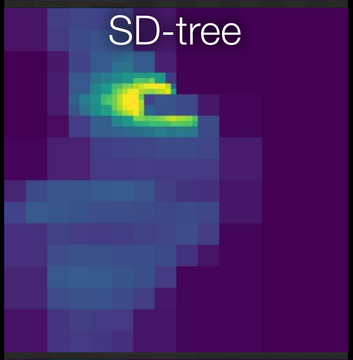
Neural network



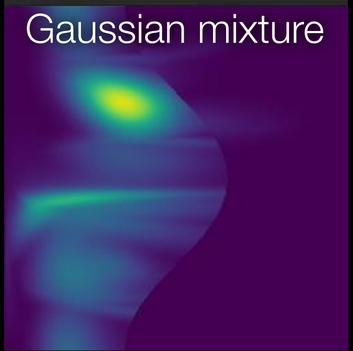
Reference

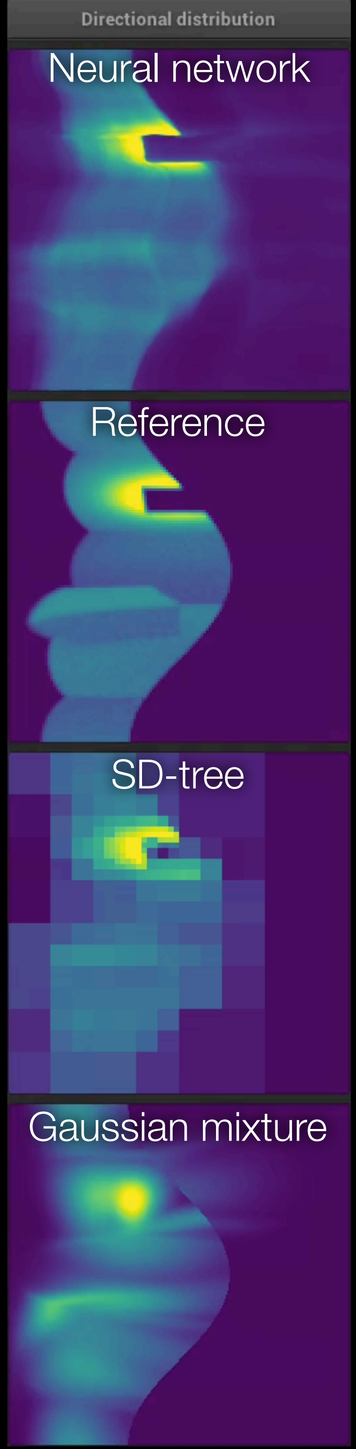


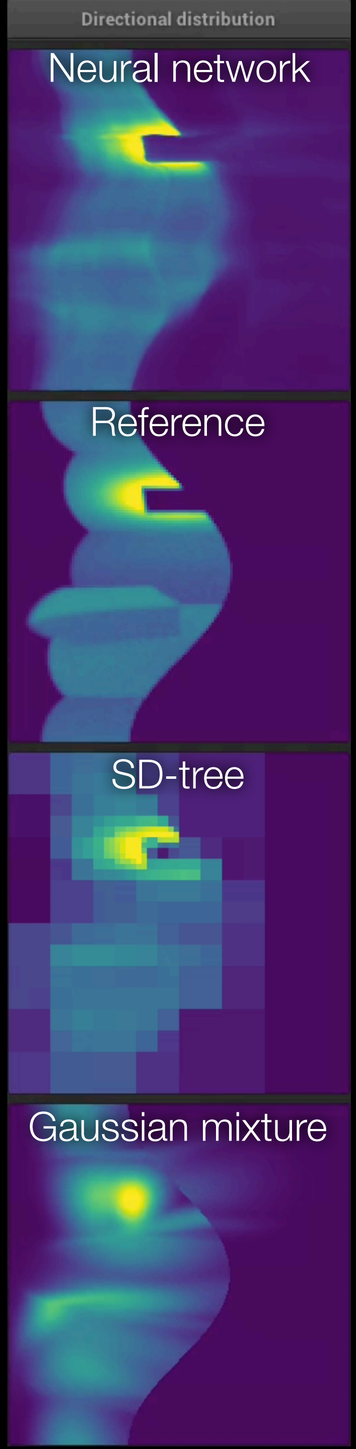
SD-tree



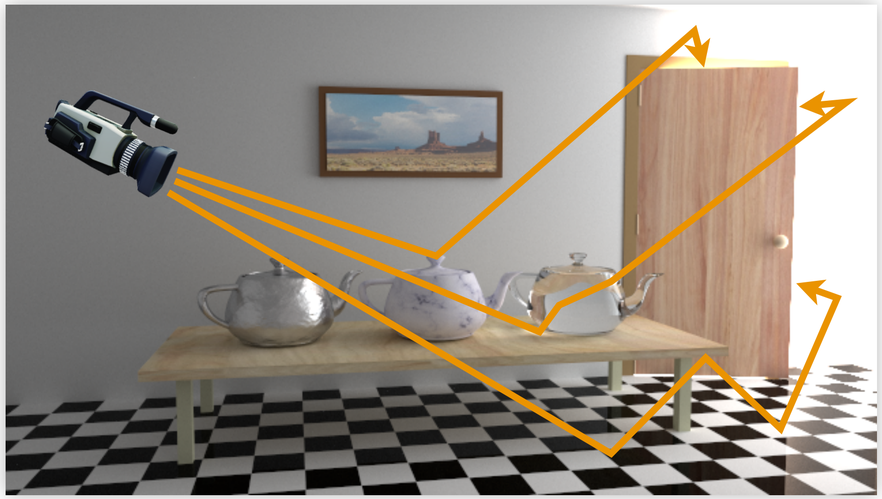
Gaussian mixture



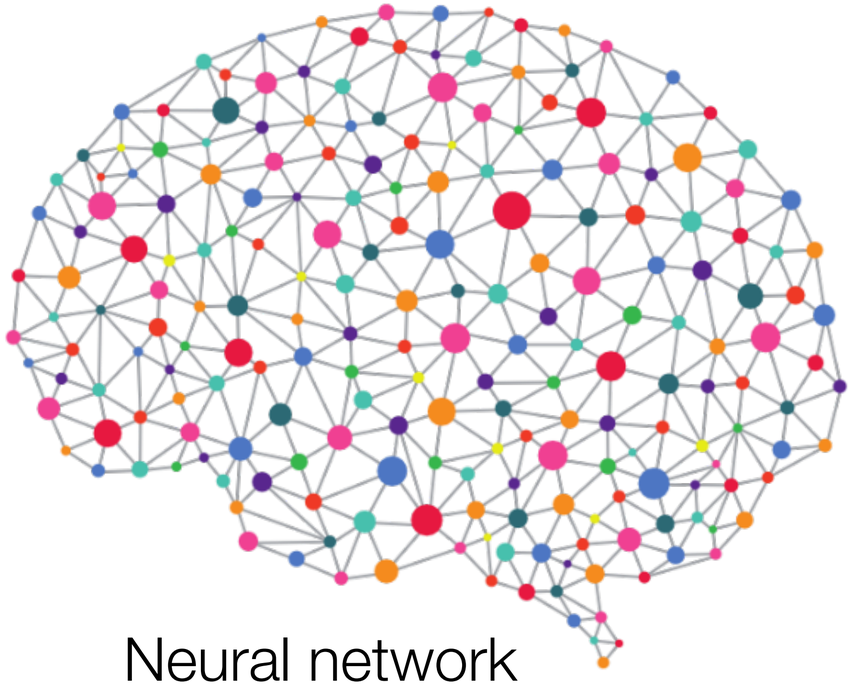

Directional distribution
Neural network
Reference
SD-tree
Gaussian mixture


Directional distribution
Neural network
Reference
SD-tree
Gaussian mixture

Neural path guiding overview

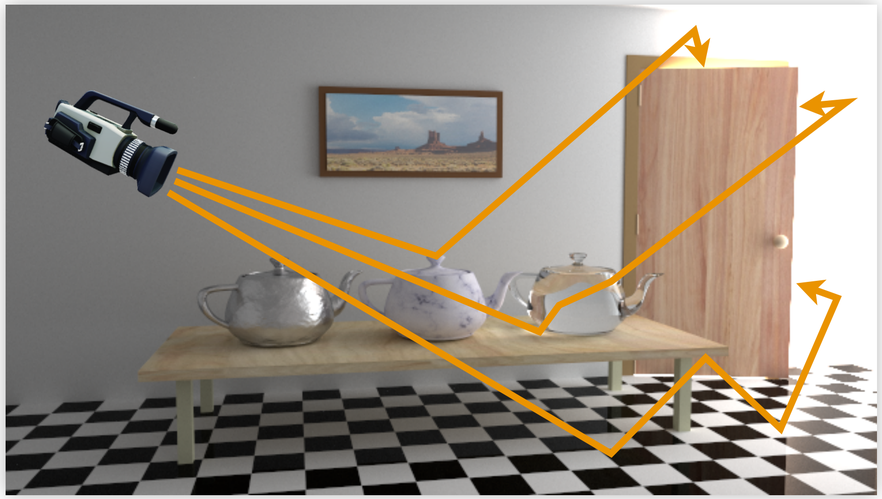


Path tracer



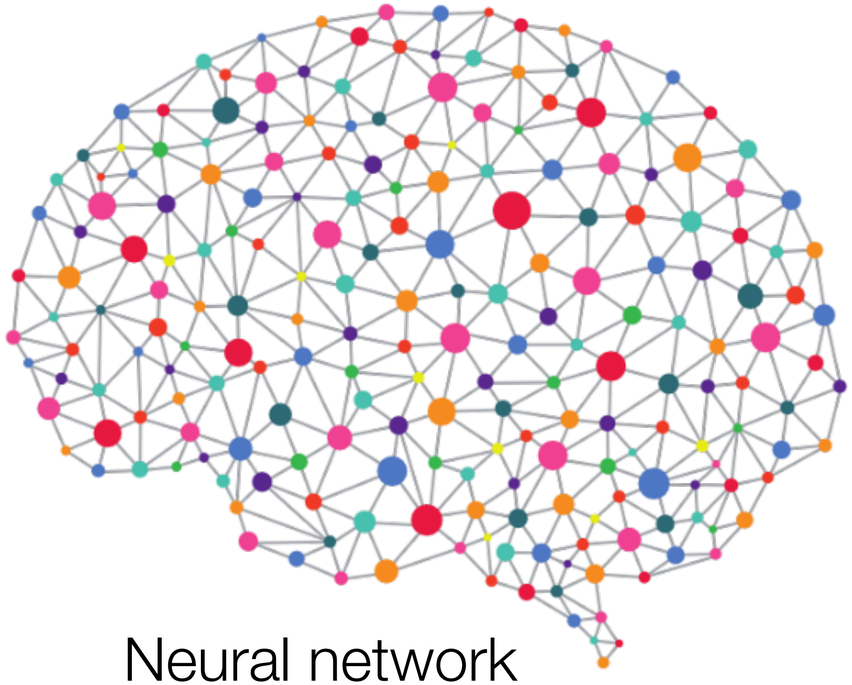
Neural network

Neural path guiding overview



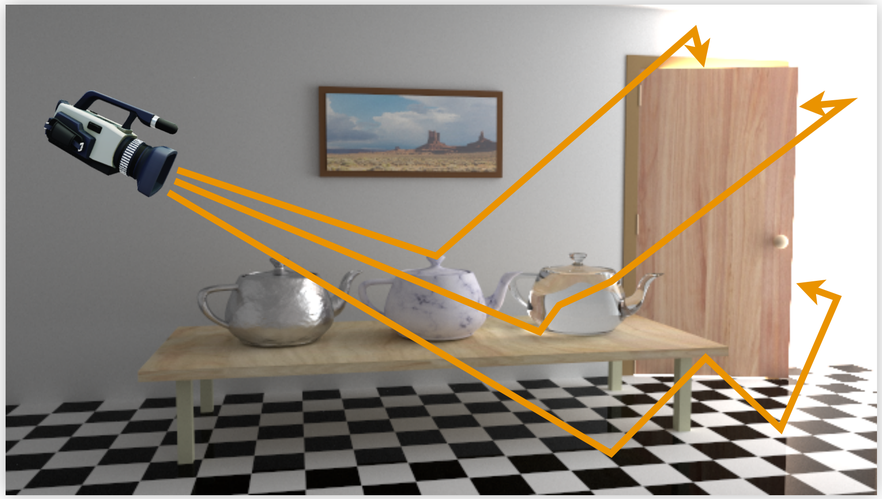
Path tracer

Sample



Neural network

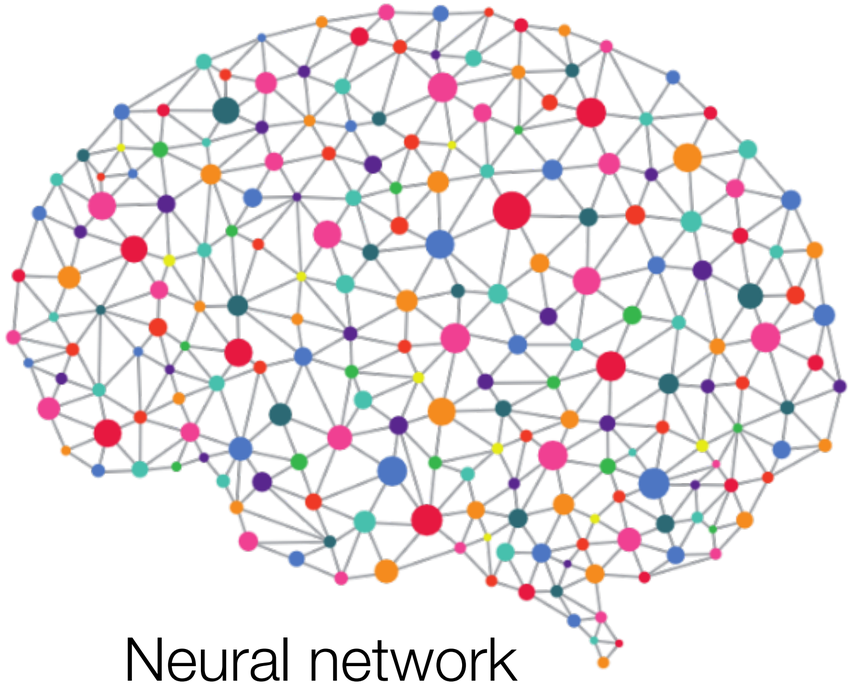
Neural path guiding overview



Path tracer

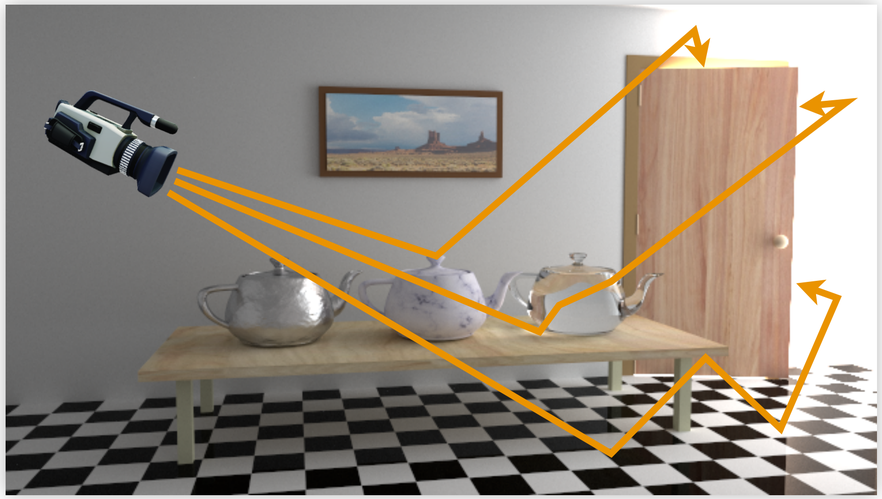
Sample

Optimize



Neural network

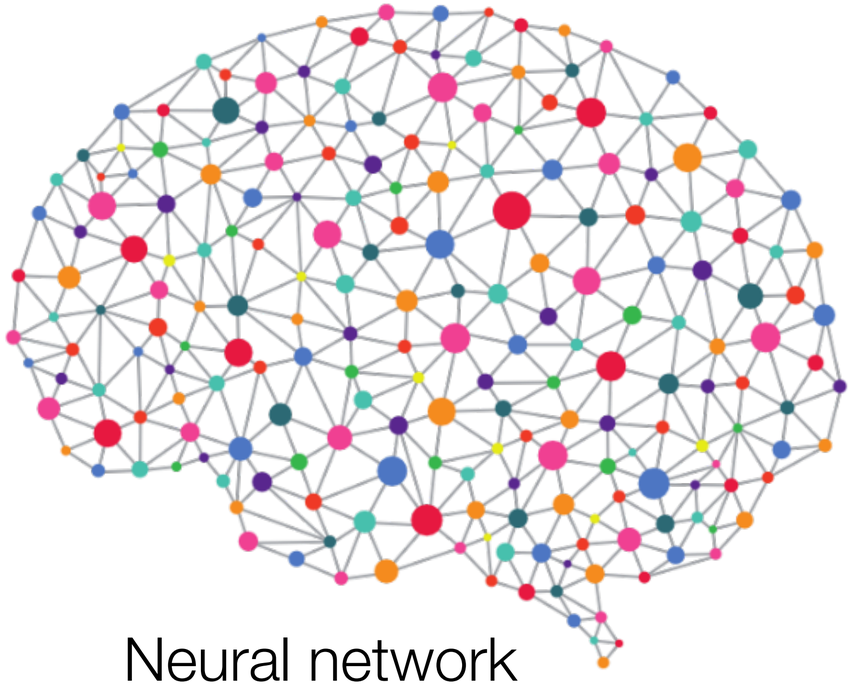
Neural path guiding overview



Path tracer

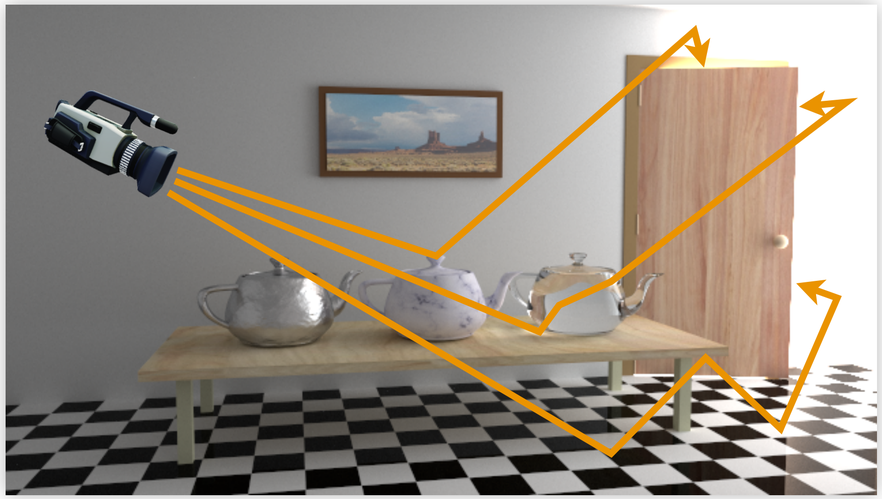
Sample

Optimize



Neural network

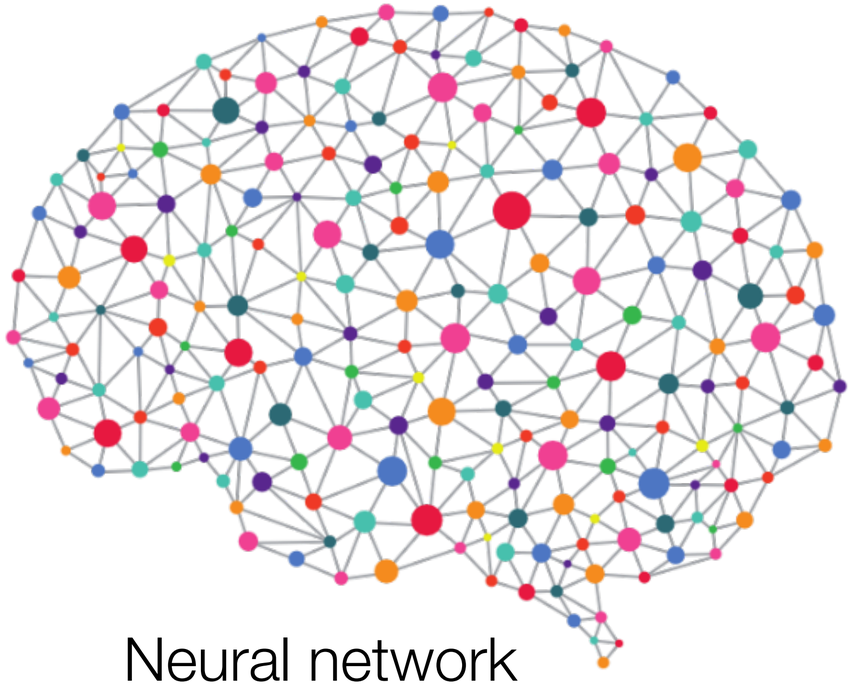
Neural path guiding overview



Path tracer

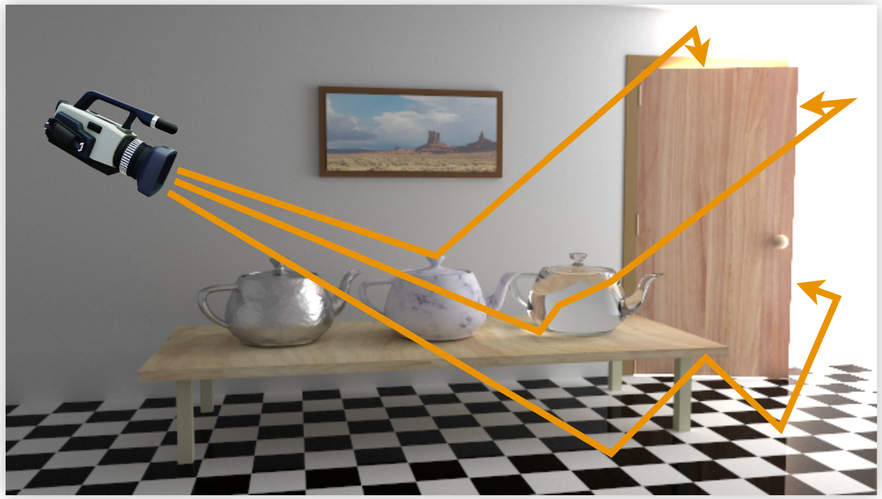
Sample

Optimize



Neural network

Neural path guiding overview

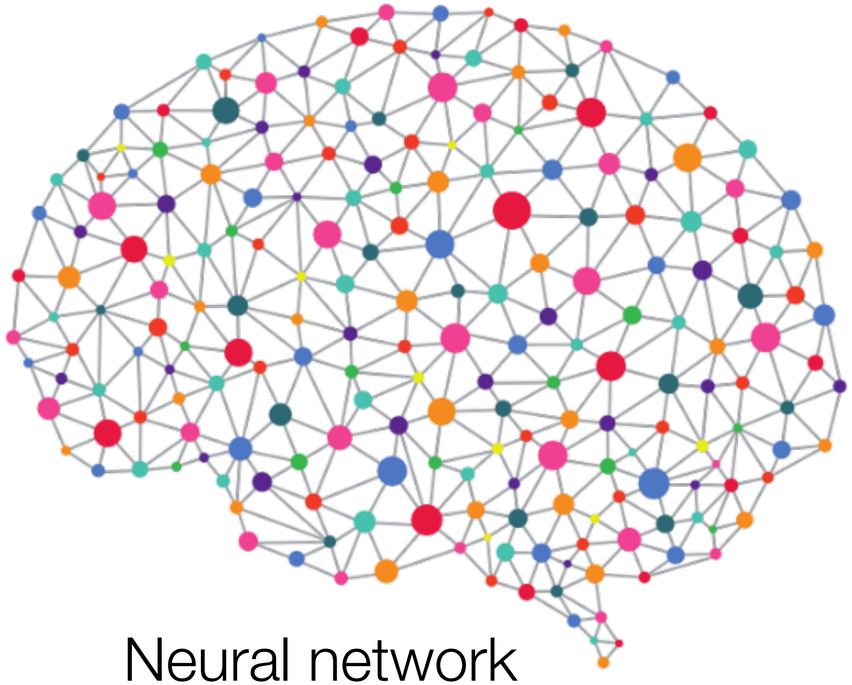


Path tracer

Sample

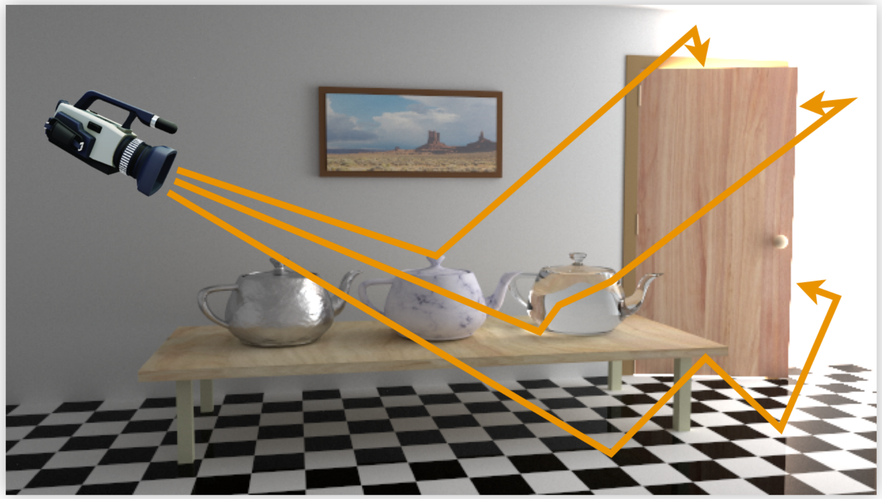
Feedback loop

Optimize



Neural network

Neural path guiding overview



Path tracer

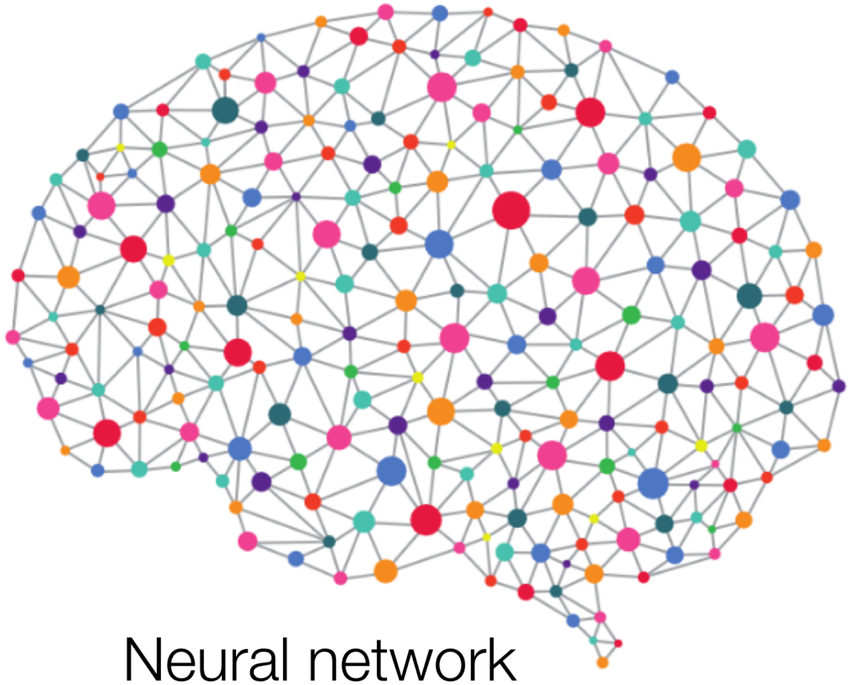
How?

Sample

Feedback loop

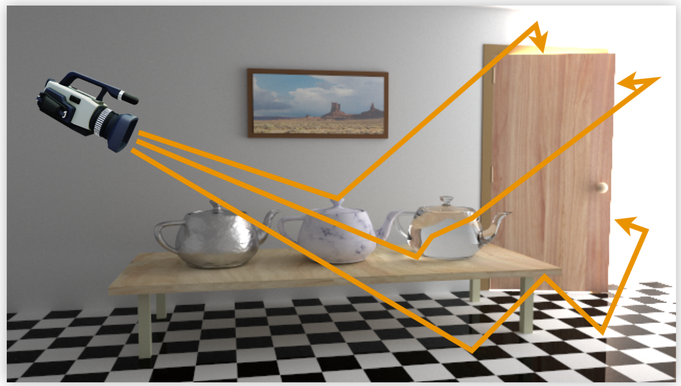
Optimize

How?



Neural network

How to draw samples?

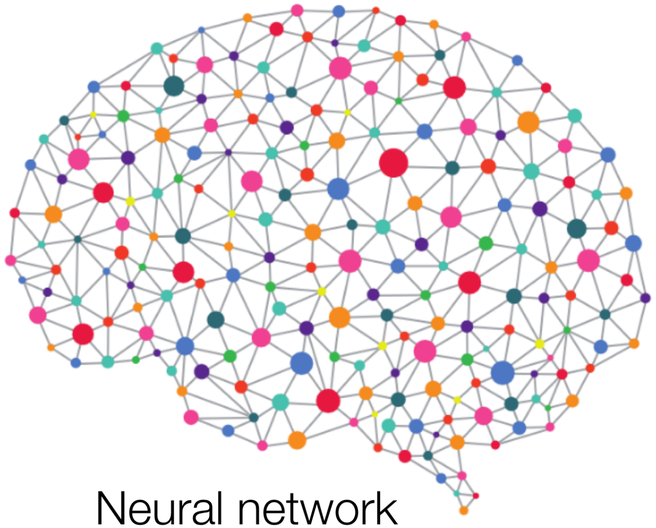


Path tracer

Sample

Feedback loop

Optimize



Neural network

Goal: warp random numbers to good distribution with NN

Goal: warp random numbers to good distribution with NN

Goal: warp random numbers to good distribution with NN

Goal: warp random numbers to good distribution with NN

Goal: warp random numbers to good distribution with NN

Goal: warp random numbers to good distribution with NN

Monte Carlo estimator

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

Goal: warp random numbers to good distribution with NN

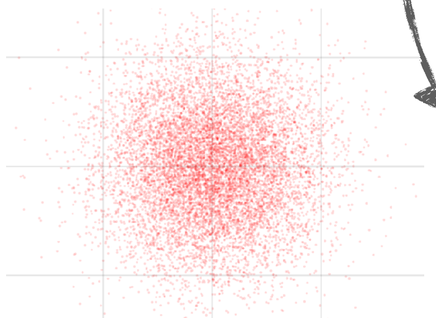
Monte Carlo estimator

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

Need p in closed form!

Goal: warp random numbers to good distribution with NN

Random number

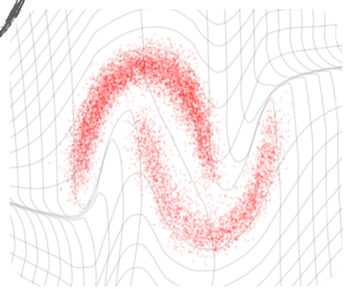


[Dinh et al. 2016]

z

x

Sample



[Dinh et al. 2016]

Monte Carlo estimator

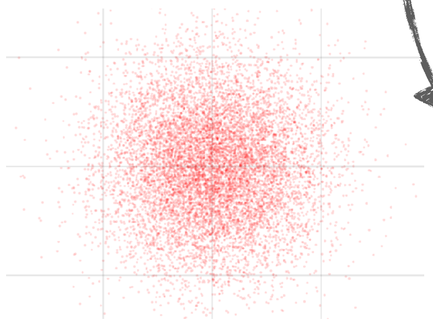
$$F \approx \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)}$$

Need p in closed form!

Addressed by "normalizing flows"

Parameterizing a bijection allows using the change-of-variable formula

Random number



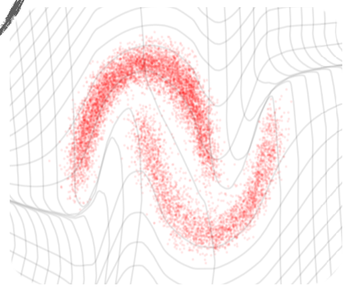
[Dinh et al. 2016]

z → $m(z;\phi)$ →

Parametric bijective function

x

Sample



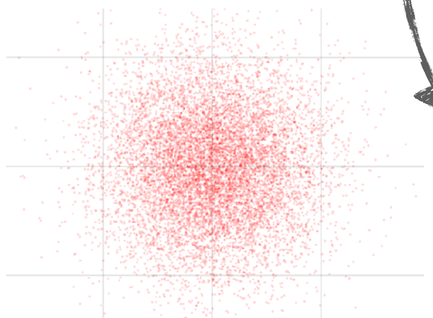
[Dinh et al. 2016]

Parameterizing a bijection allows using the change-of-variable formula

Random number

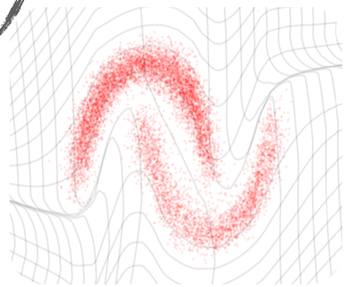
Parametric bijective function

Sample



[Dinh et al. 2016]

$z \rightarrow m(z;\phi) \rightarrow x$



[Dinh et al. 2016]

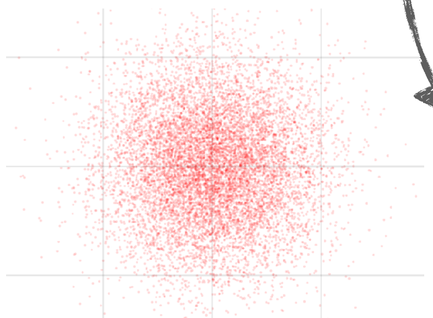
$$p(x) = p(z) \cdot \left| det \left(\frac{\partial m(z)}{\partial z^T} \right) \right|^{-1}$$

Parameterizing a bijection allows using the change-of-variable formula

Random number

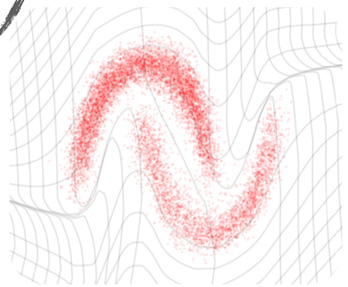
Parametric bijective function

Sample



[Dinh et al. 2016]

$z \rightarrow m(z;\phi) \rightarrow x$



[Dinh et al. 2016]

$$p(x) = p(z) \cdot \left| det \left(\frac{\partial m(z)}{\partial z^T} \right) \right|^{-1}$$

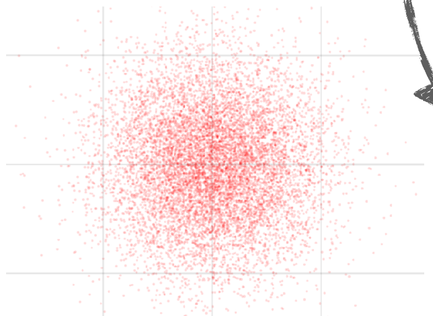
Our choice, e.g. Gaussian

Parameterizing a bijection allows using the change-of-variable formula

Random number

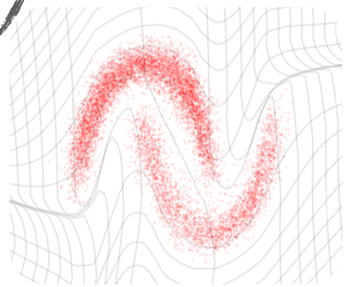
Parametric bijective function

Sample



[Dinh et al. 2016]

$z \rightarrow m(z;\phi) \rightarrow x$



[Dinh et al. 2016]

$$p(x) = p(z) \cdot \left| det \left(\frac{\partial m(z)}{\partial z^T} \right) \right|^{-1}$$

Our choice, e.g. Gaussian

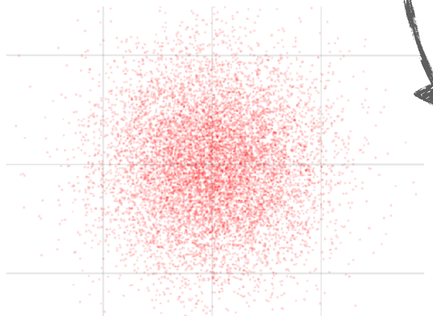
Squishing/stretching by m

Parameterizing a bijection allows using the change-of-variable formula

Random number

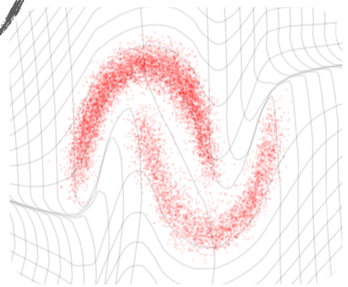
Parametric bijective function

Sample



[Dinh et al. 2016]

$z \rightarrow m(z;\phi) \rightarrow x$



[Dinh et al. 2016]

$$p(x) = p(z) \cdot \left| det \left(\frac{\partial m(z)}{\partial z^T} \right) \right|^{-1}$$

Our choice, e.g. Gaussian

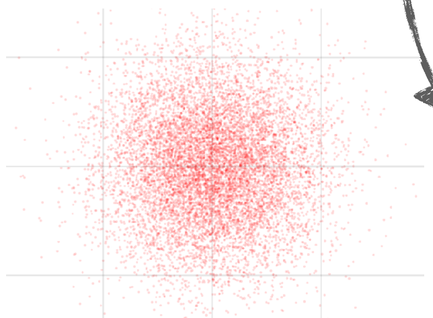
Squishing/stretching by m

A chain of simple bijections can model complicated functions

Parametric bijective functions
(normalizing flow)

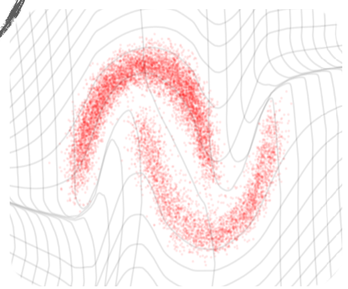
Random number

Sample



[Dinh et al. 2016]

$$z \rightarrow m_1 \circ m_2 \circ \ldots \circ m_L \rightarrow x$$



[Dinh et al. 2016]

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

Our choice, e.g. Gaussian

Squishing/stretching by m

Affine bijections

NICE [Dinh et al. 2014]

$$m(z; t) = t + z$$

translation

Affine bijections

RealNVP [Dinh et al. 2016]

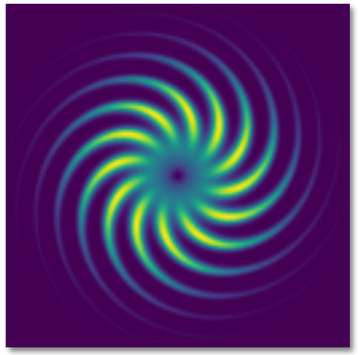
$$m(z; t, s) = t + z \cdot s$$

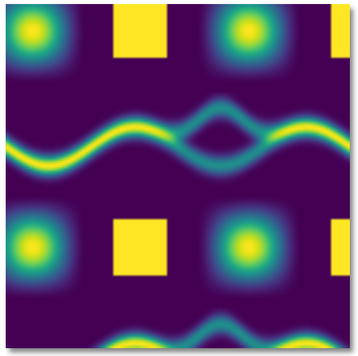
translation & scale

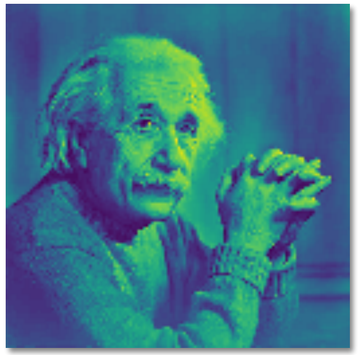
Affine bijections (RealNVP) are not good enough

Affine bijections (RealNVP) are not good enough

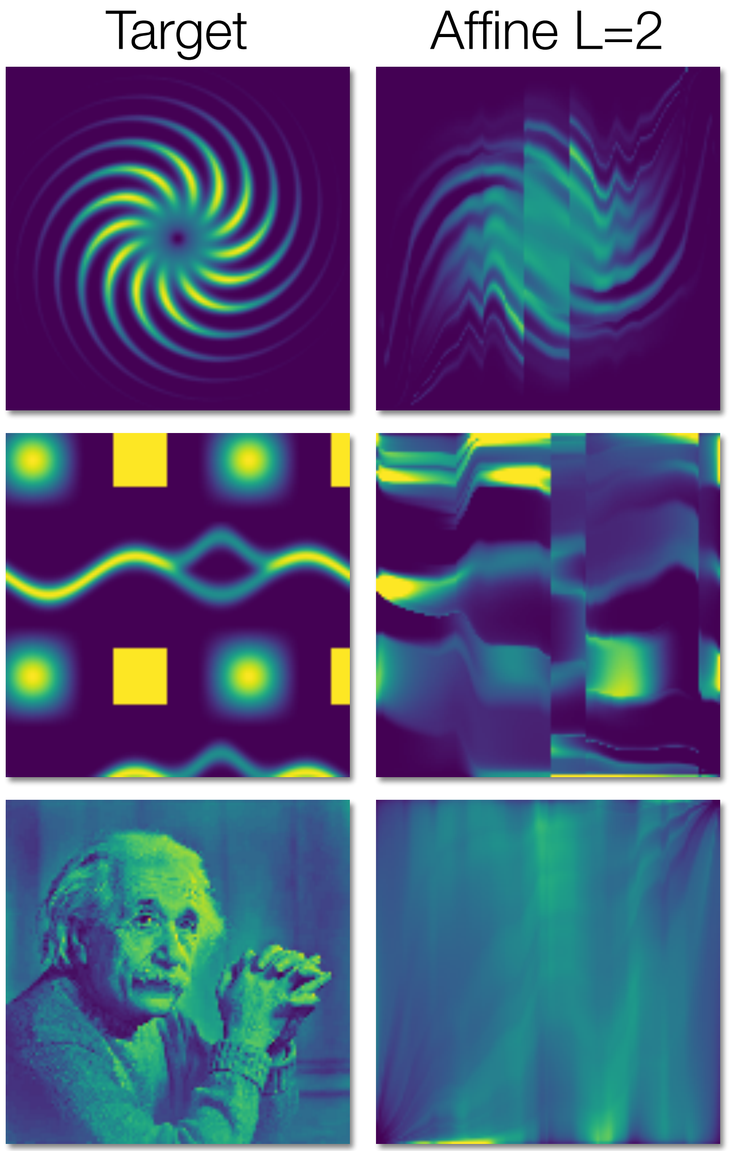
Target



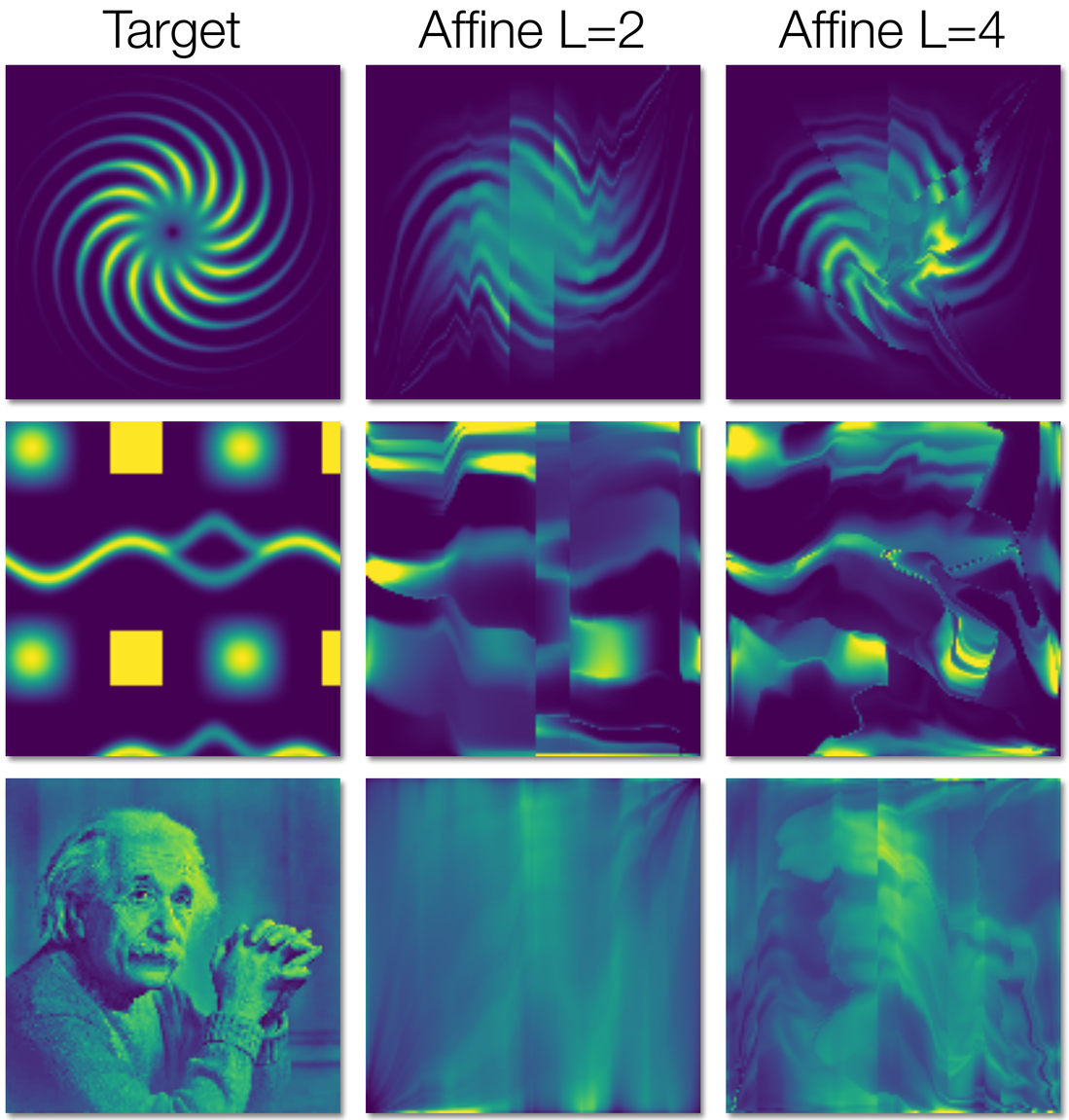




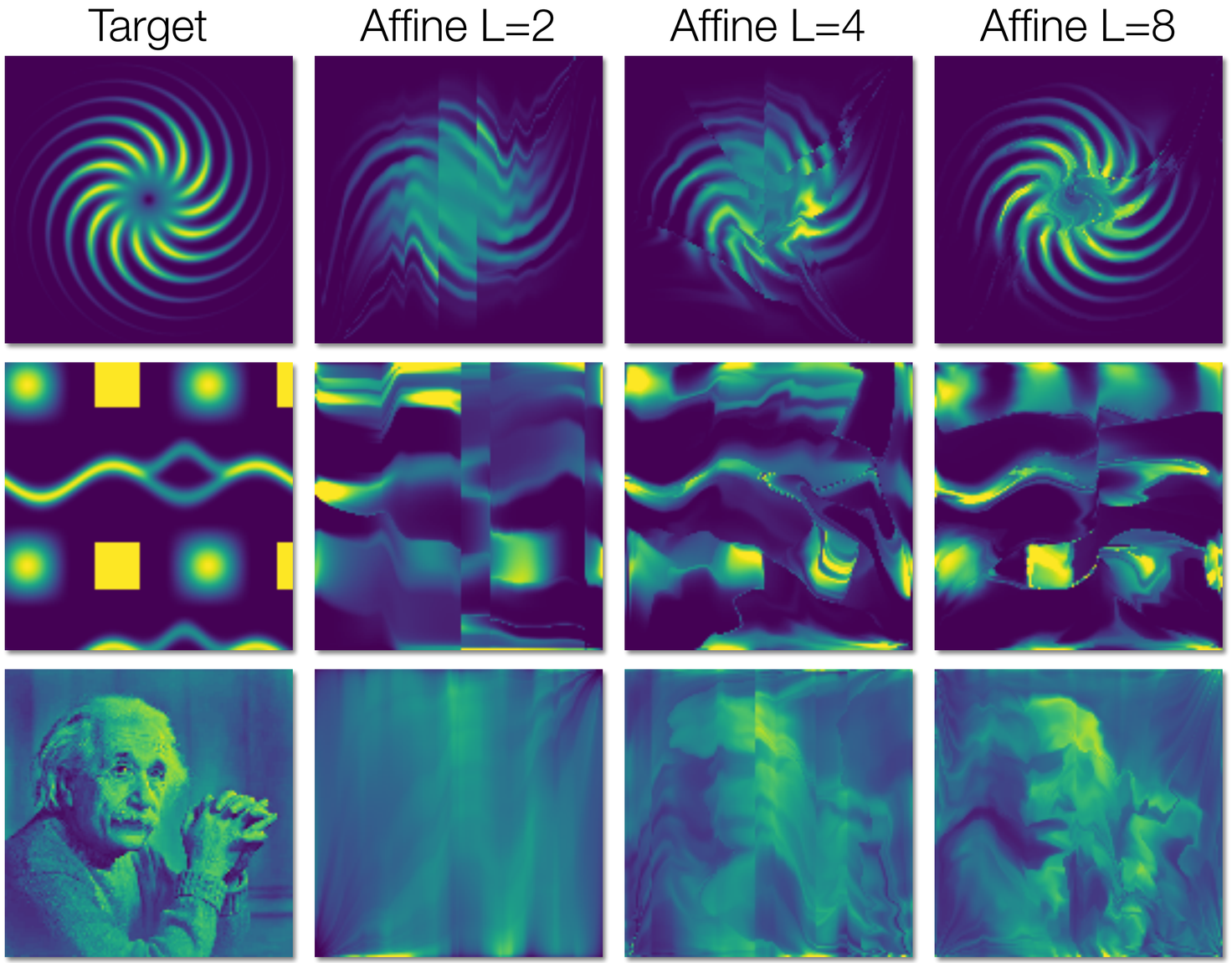
Affine bijections (RealNVP) are not good enough



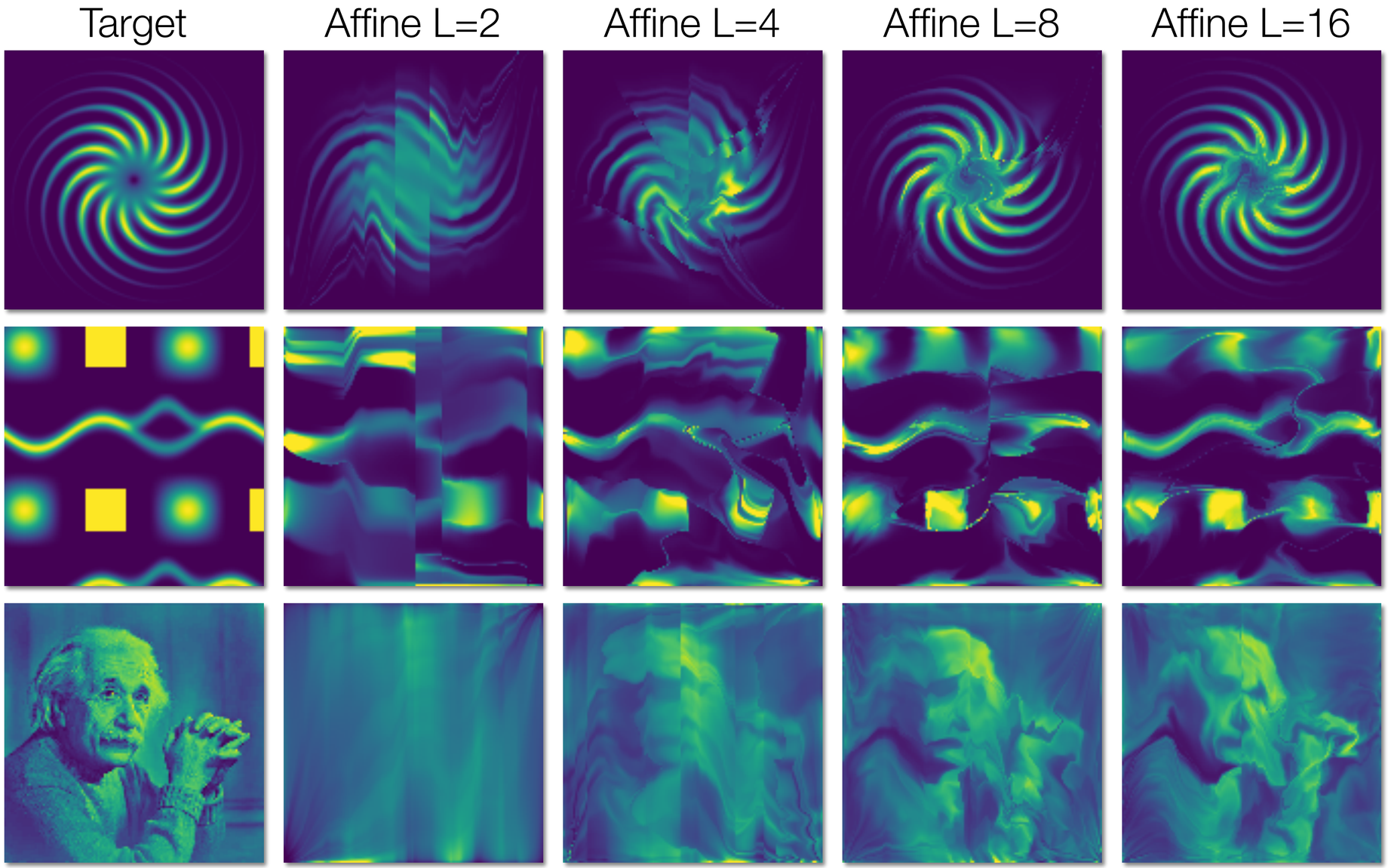
Affine bijections (RealNVP) are not good enough



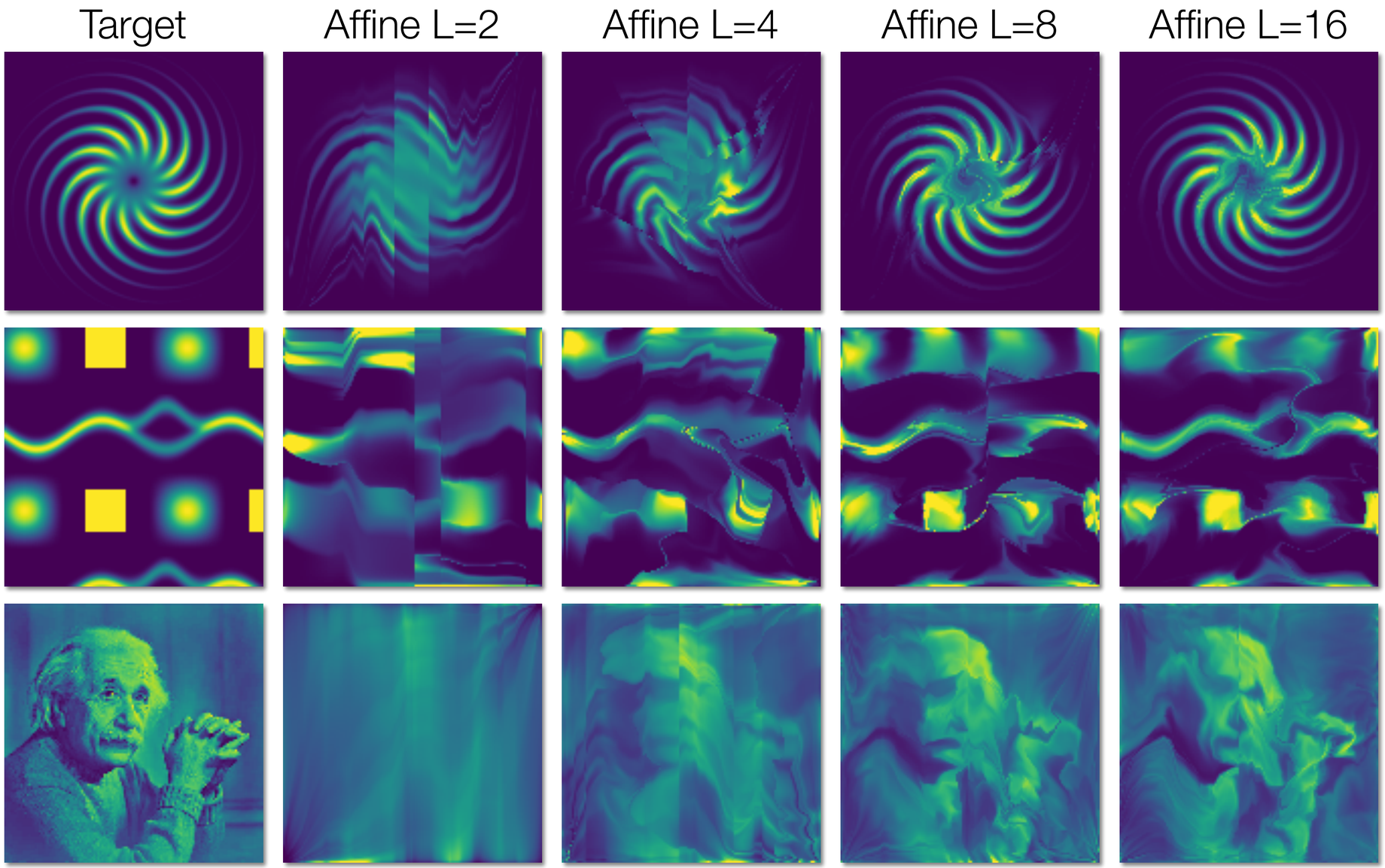
Affine bijections (RealNVP) are not good enough



Affine bijections (RealNVP) are not good enough

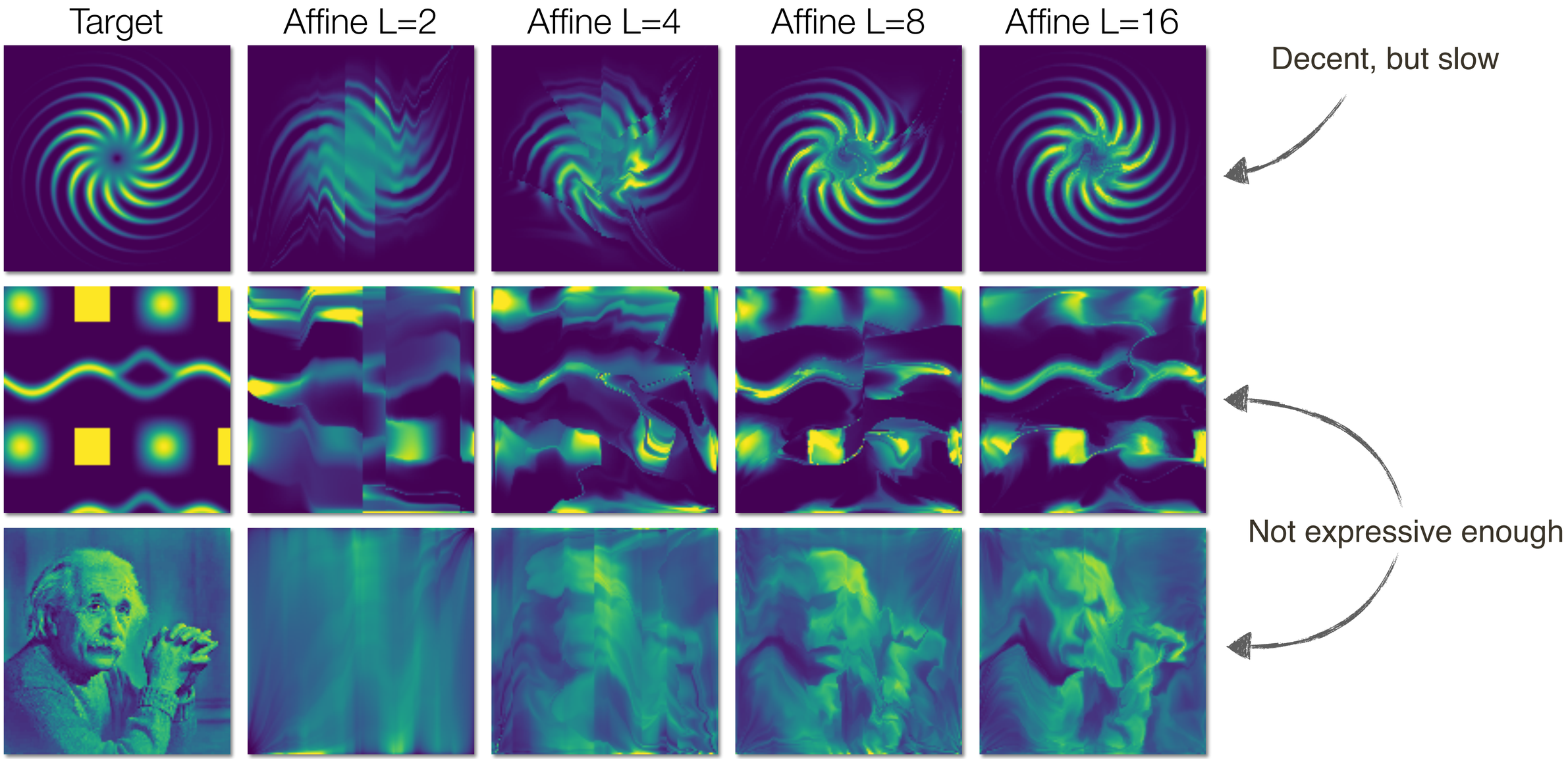


Affine bijections (RealNVP) are not good enough

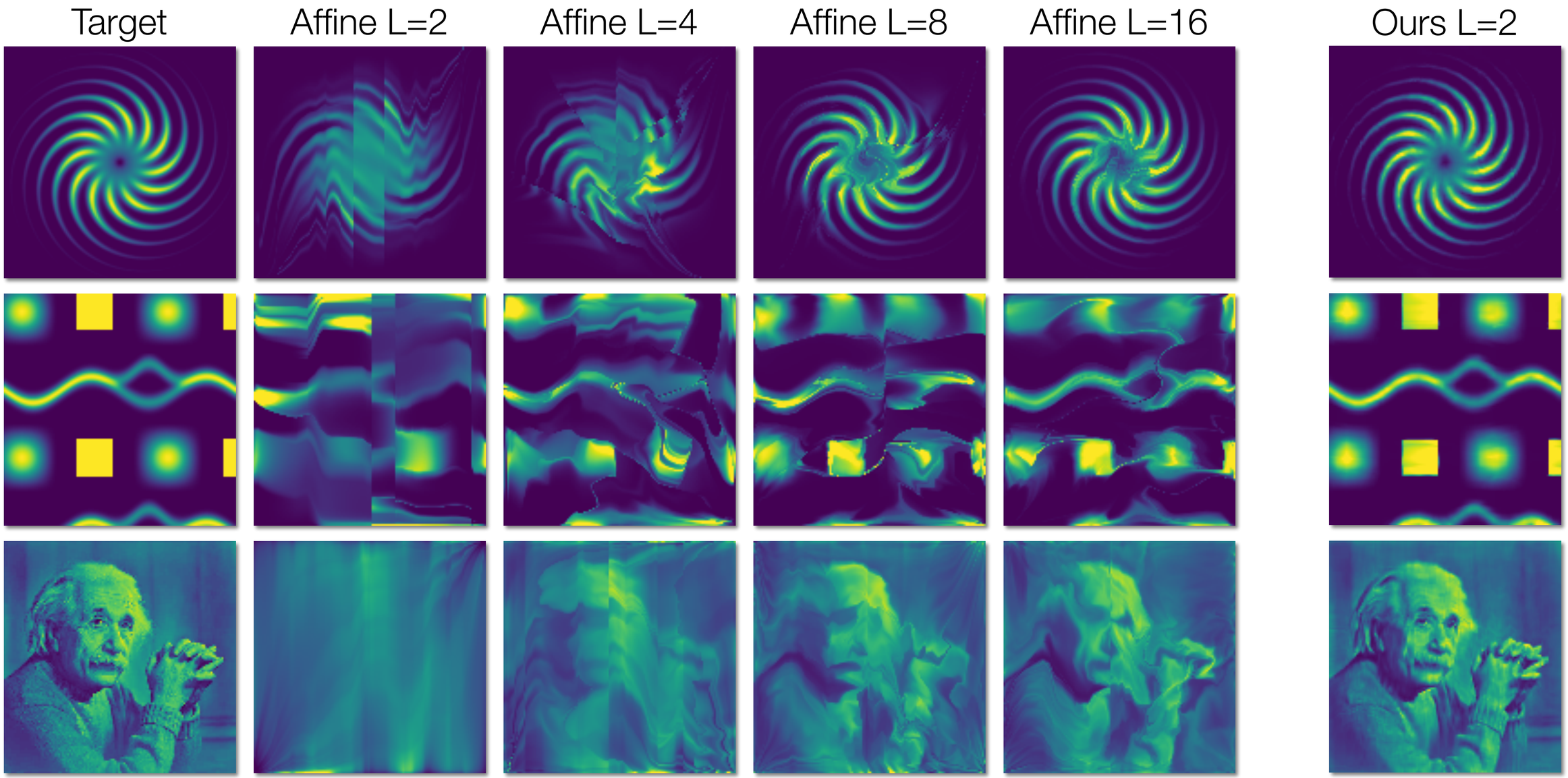


Decent, but slow

Affine bijections (RealNVP) are not good enough



Affine bijections (RealNVP) are not good enough



Piecewise-polynomial bijections: definition

Piecewise-polynomial bijections: definition

$$z \sim \mathcal{U}(0,1)$$

Piecewise-polynomial bijections: definition

$$z \sim \mathcal{U}(0,1) \qquad m : [0,1] \to [0,1]$$

Piecewise-polynomial bijections: definition

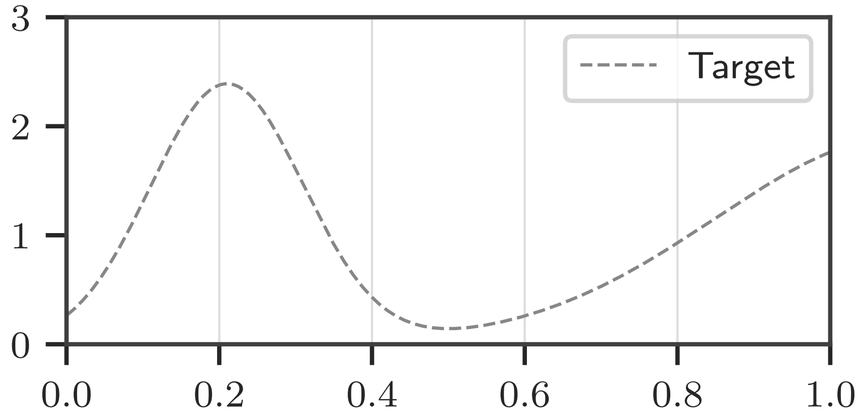
$$z \sim \mathcal{U}(0,1) \qquad m : [0,1] \to [0,1]$$

$$p(x) = dm^{-1}/dx$$

Piecewise-polynomial bijections: definition

$$z \sim \mathcal{U}(0,1) \qquad m : [0,1] \to [0,1]$$

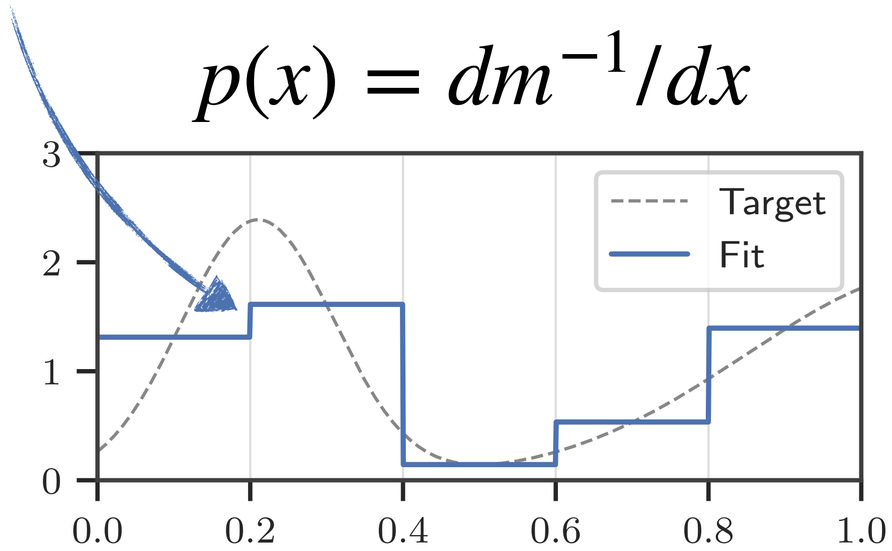
$$p(x) = dm^{-1}/dx$$



Piecewise-polynomial bijections: definition

$$z \sim \mathcal{U}(0,1) \qquad m : [0,1] \to [0,1]$$

Predicted by
Neural Network



Piecewise-polynomial bijections: definition

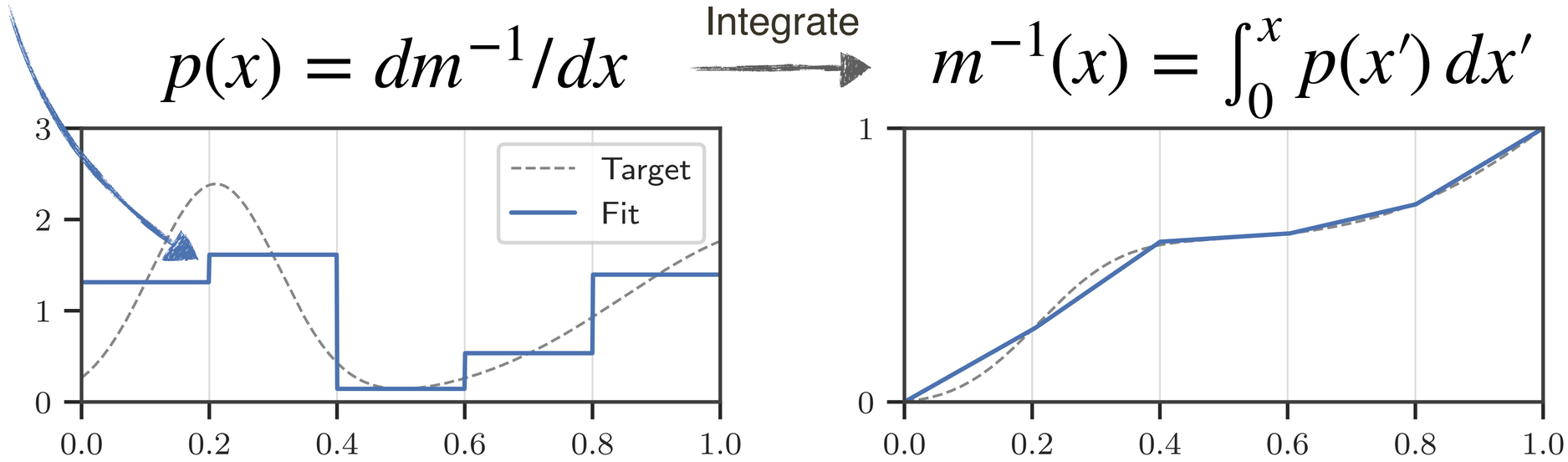
$$z \sim \mathcal{U}(0,1) \qquad m : [0,1] \to [0,1]$$

Predicted by Neural Network

$$p(x) = dm^{-1}/dx$$

Integrate

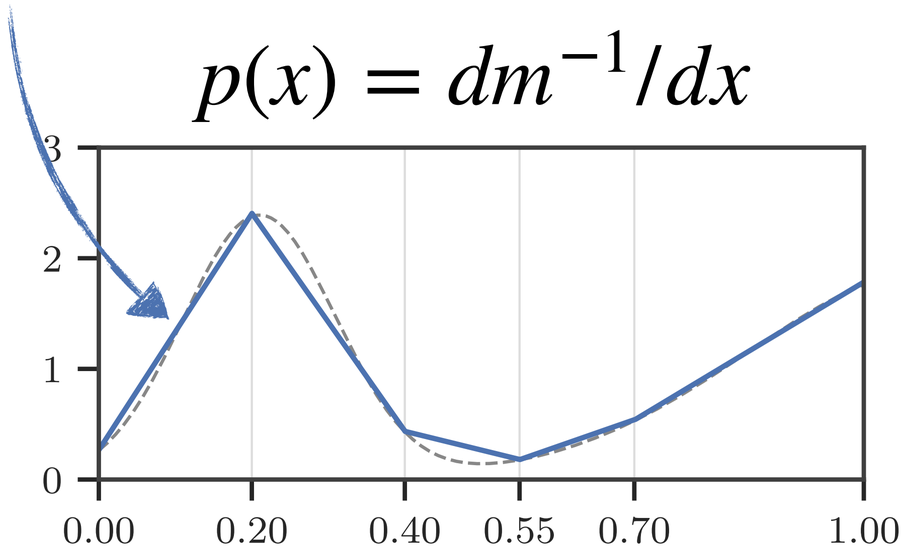
$$m^{-1}(x) = \int_0^x p(x')\,dx'$$



Piecewise-polynomial bijections: definition

$$z \sim \mathcal{U}(0,1) \qquad m : [0,1] \to [0,1]$$

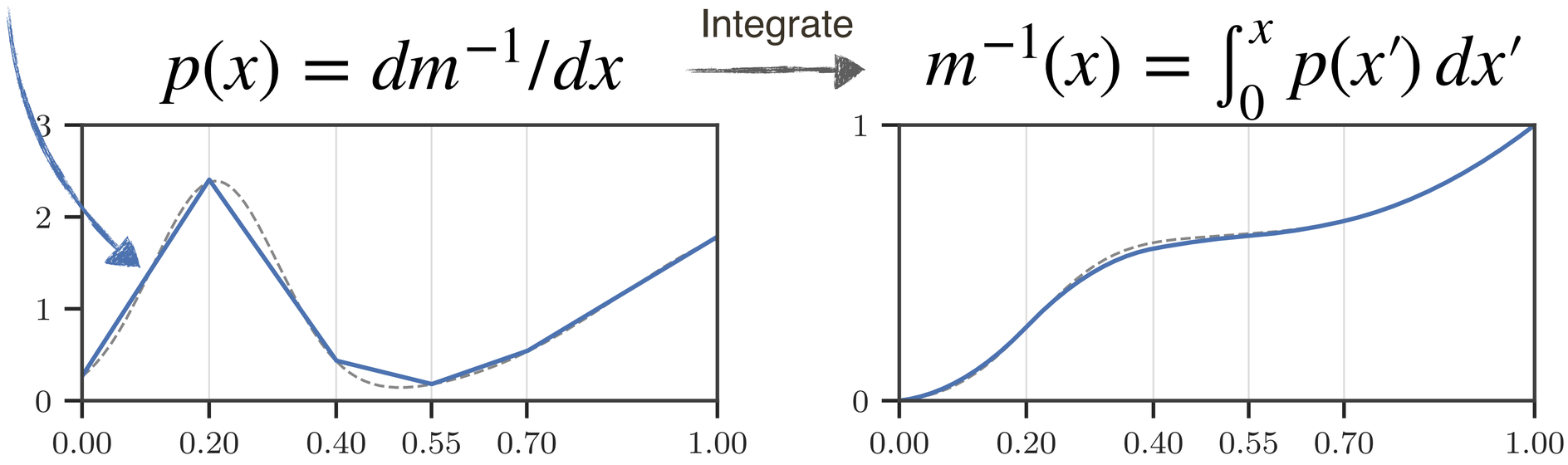
Predicted by
Neural Network



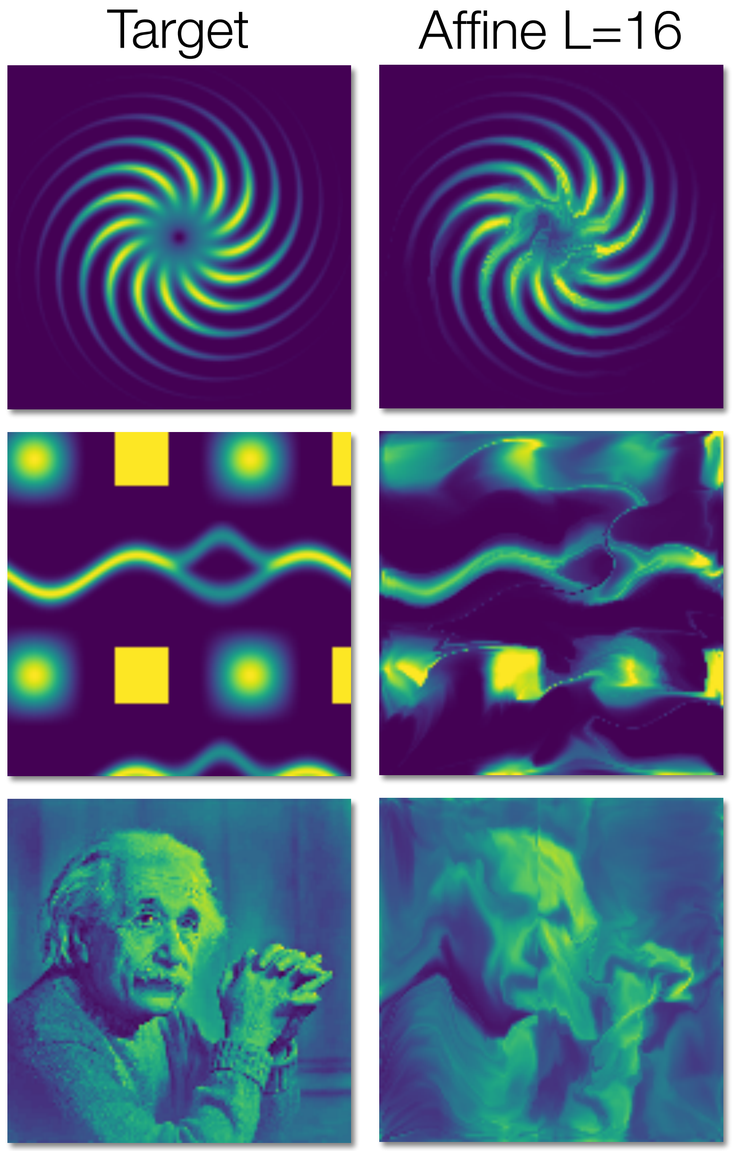
Piecewise-polynomial bijections: definition

$$z \sim \mathcal{U}(0,1) \qquad m : [0,1] \to [0,1]$$

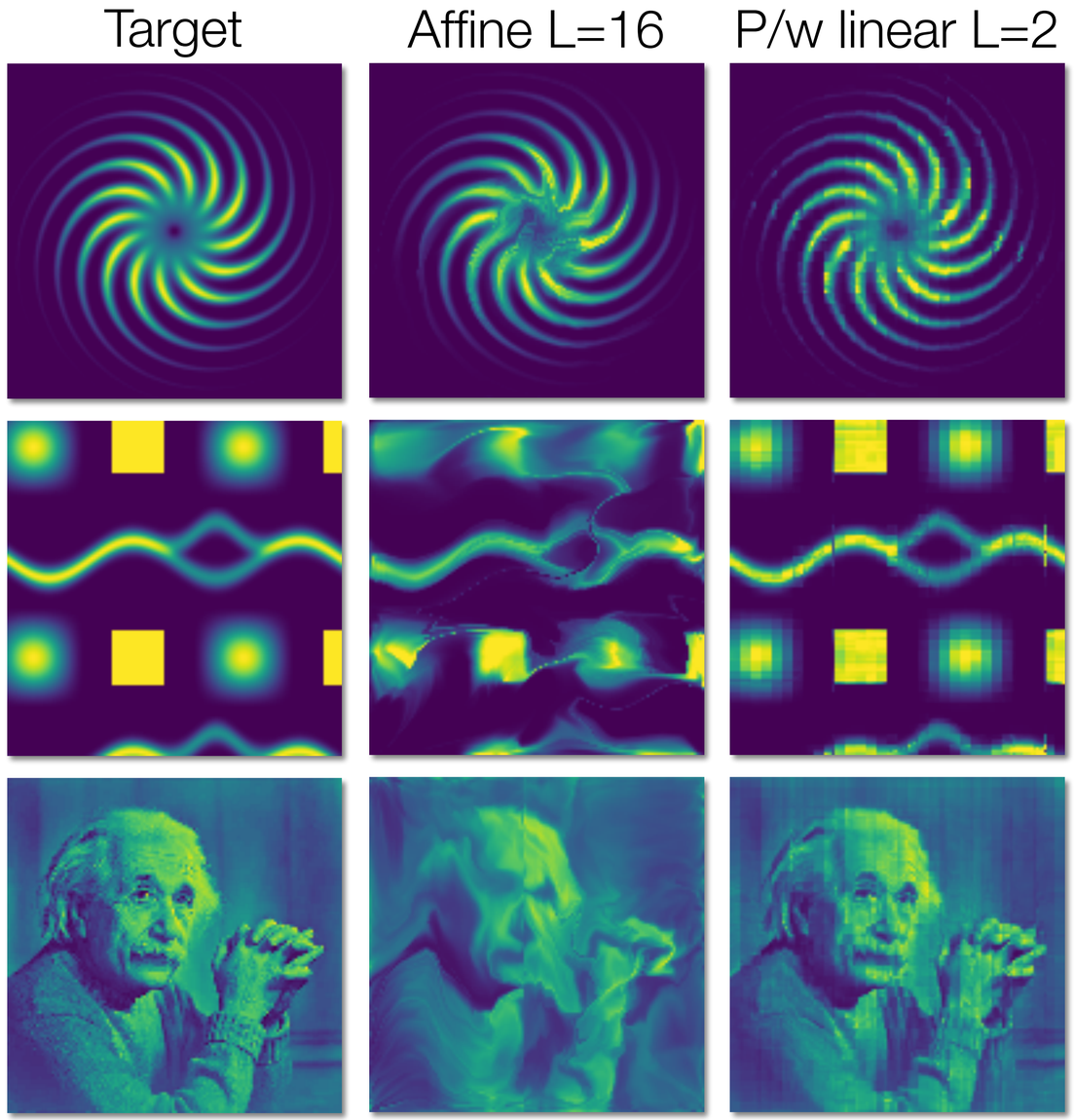
Predicted by Neural Network



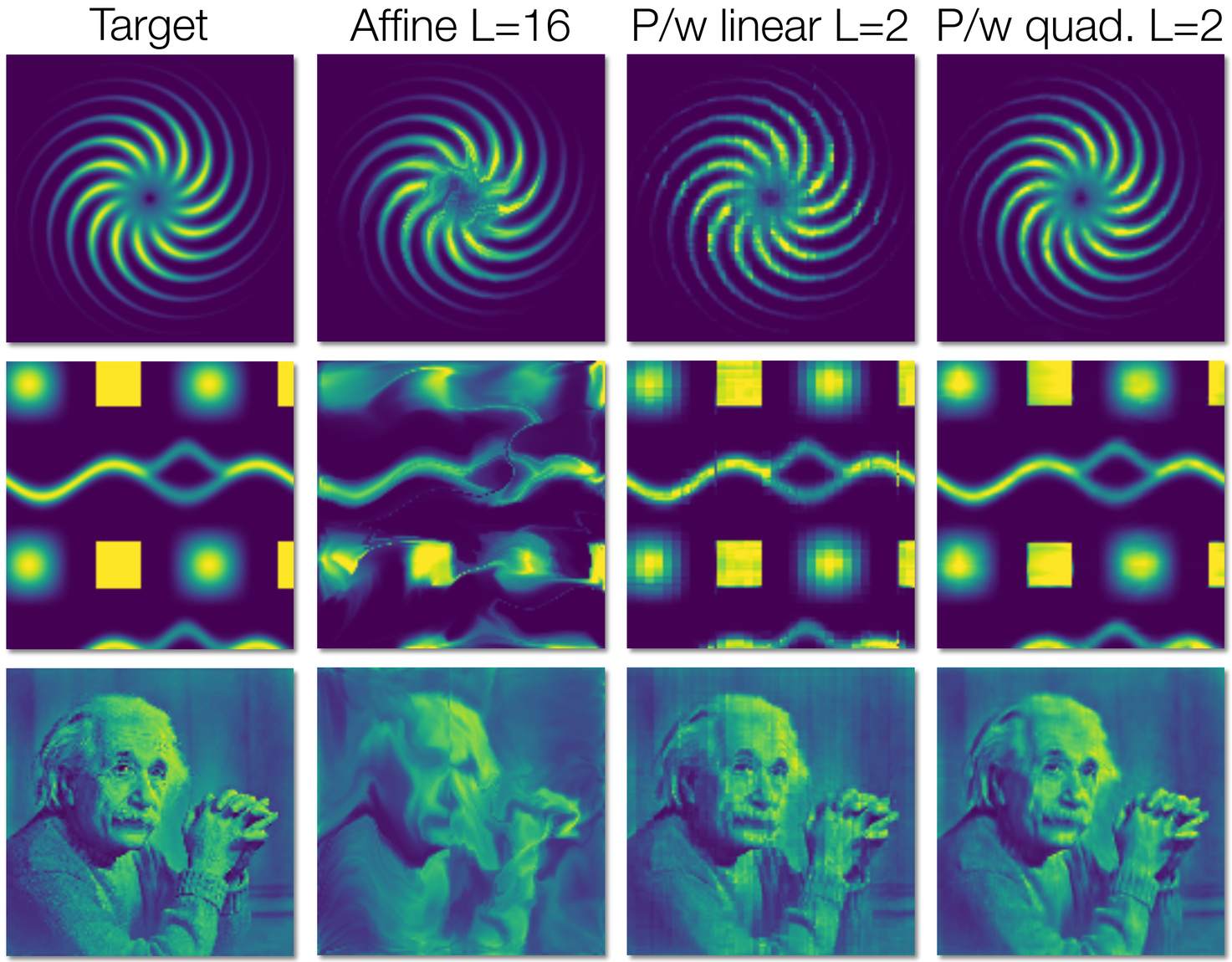
Piecewise-polynomial bijections perform better than affine ones



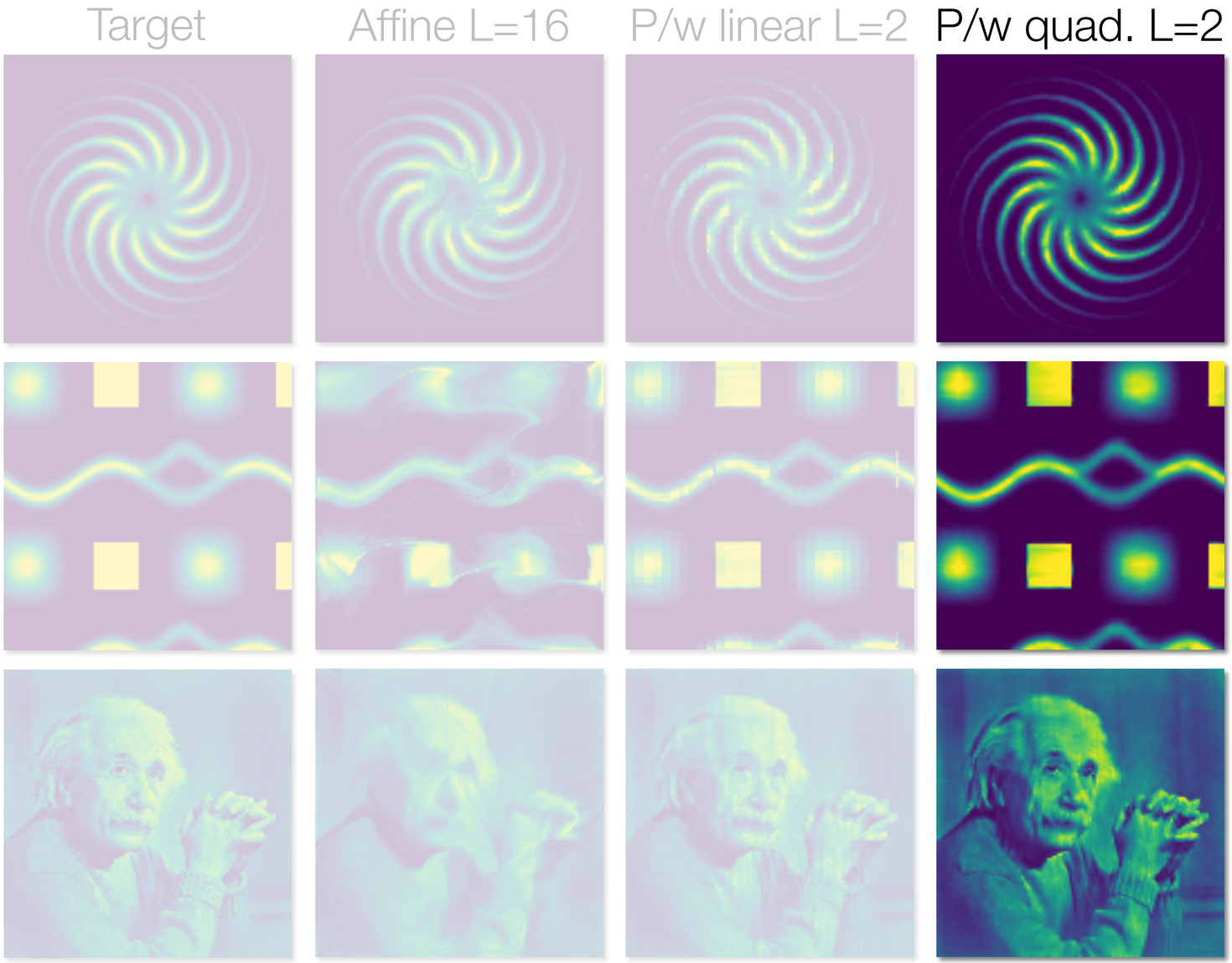
Piecewise-polynomial bijections perform better than affine ones



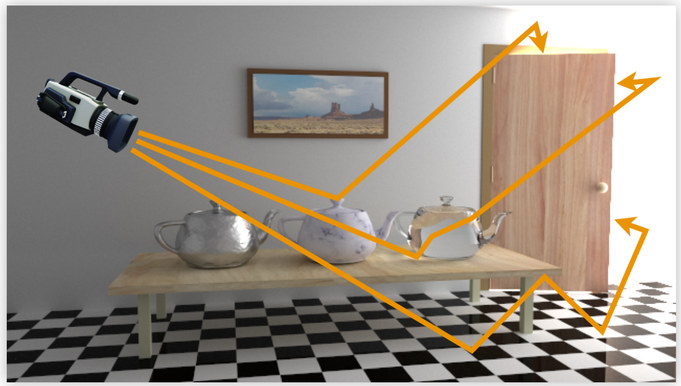
Piecewise-polynomial bijections perform better than affine ones



Piecewise-polynomial bijections perform better than affine ones



How to optimize?

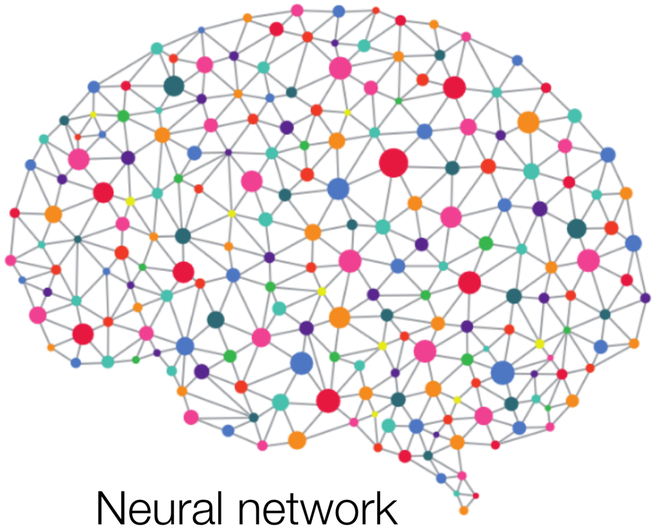


Path tracer

Sample

Feedback loop

Optimize

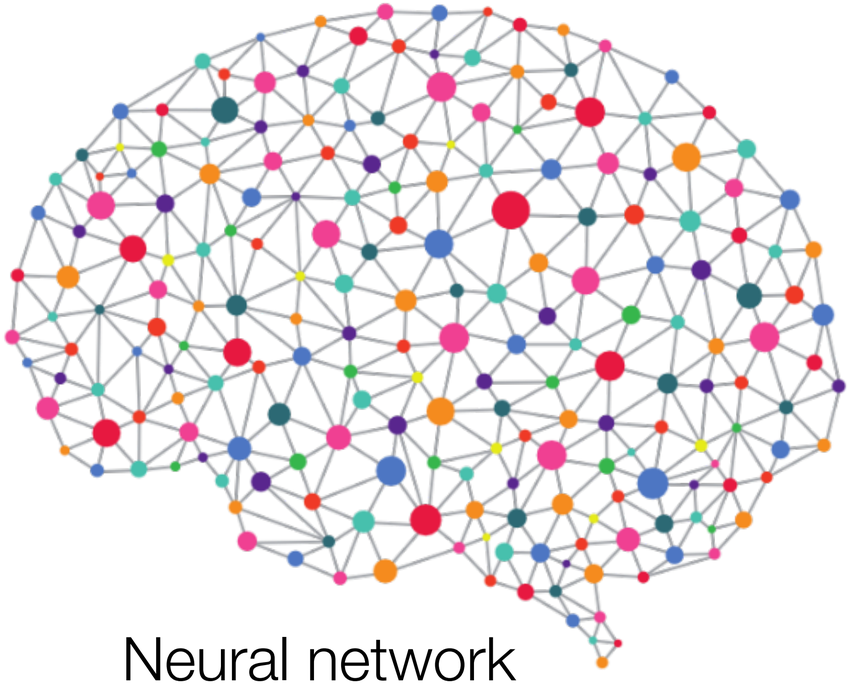


Neural network

Training with data from the correct distribution is simple

Training data

Optimize



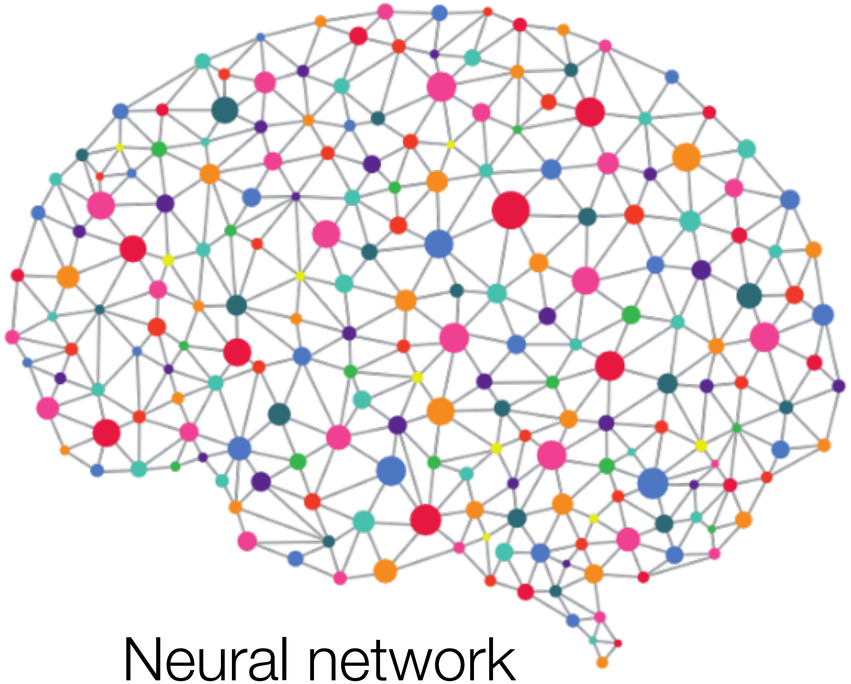
Neural network

Training with data from the correct distribution is simple

Training data

Desired distribution

Optimize



Neural network

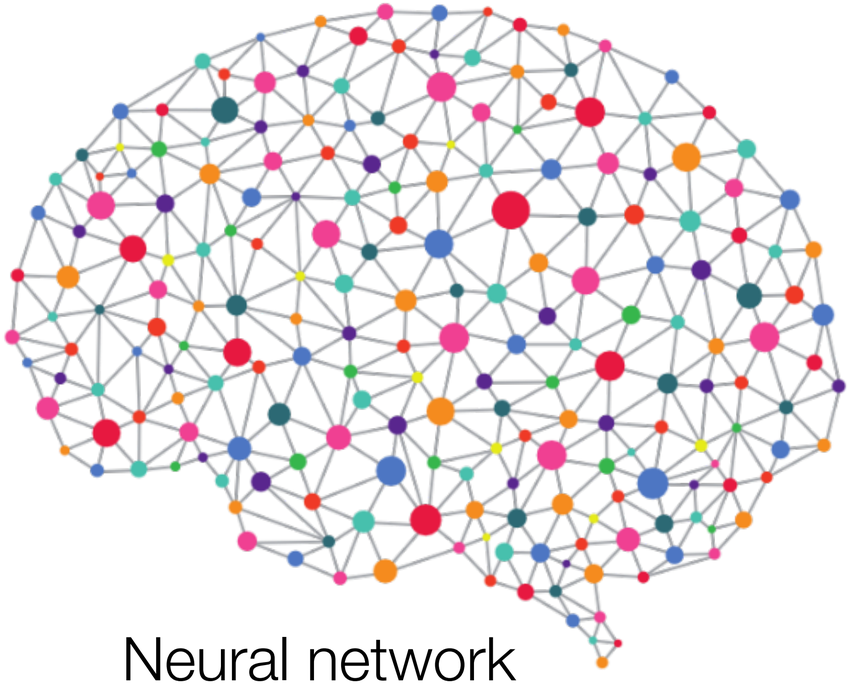
Training with data from the correct distribution is simple

Training data

Desired distribution

Optimize

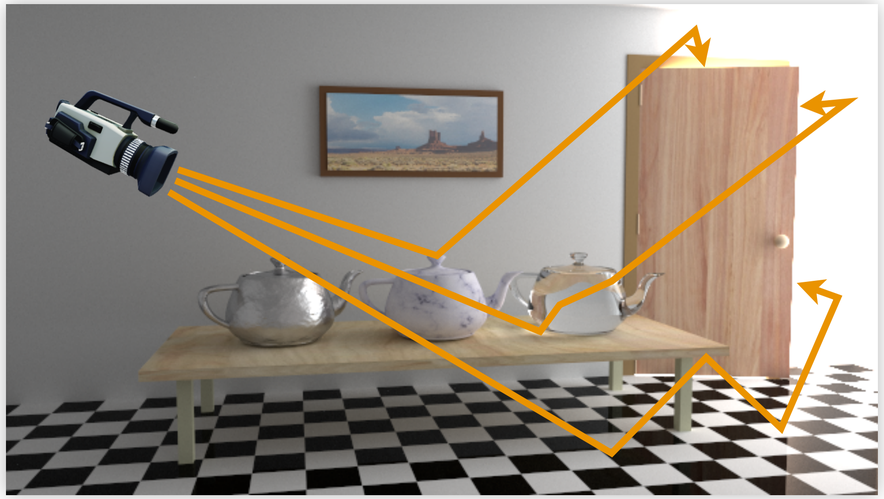
$$\nabla_\theta \log p(x;\theta)$$



Neural network

Training from Monte Carlo samples requires careful weighting

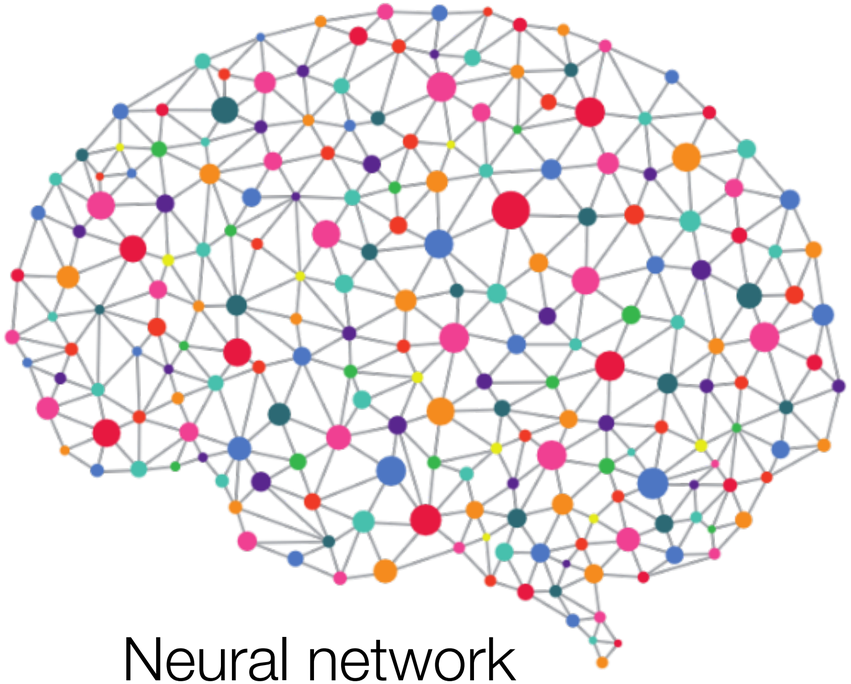
Arbitrarily distributed



Training data

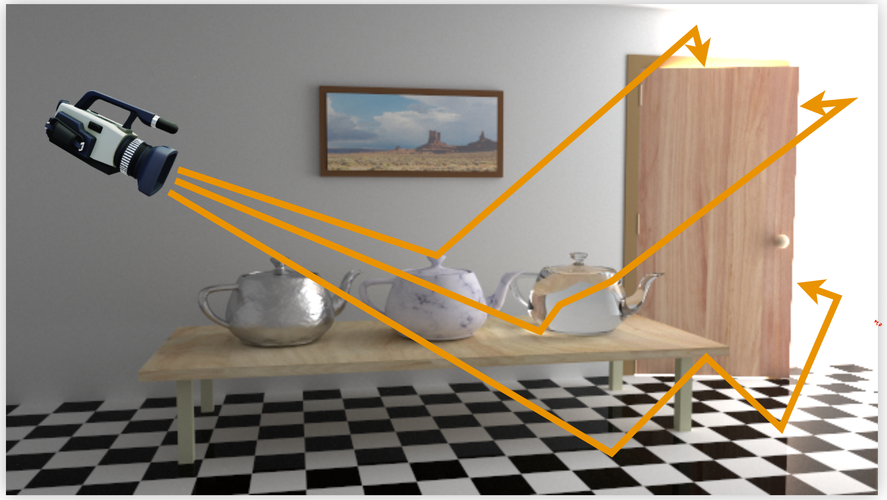
Optimize

$$\nabla_\theta \log p(x;\theta)$$



Neural network

Training from Monte Carlo samples requires careful weighting

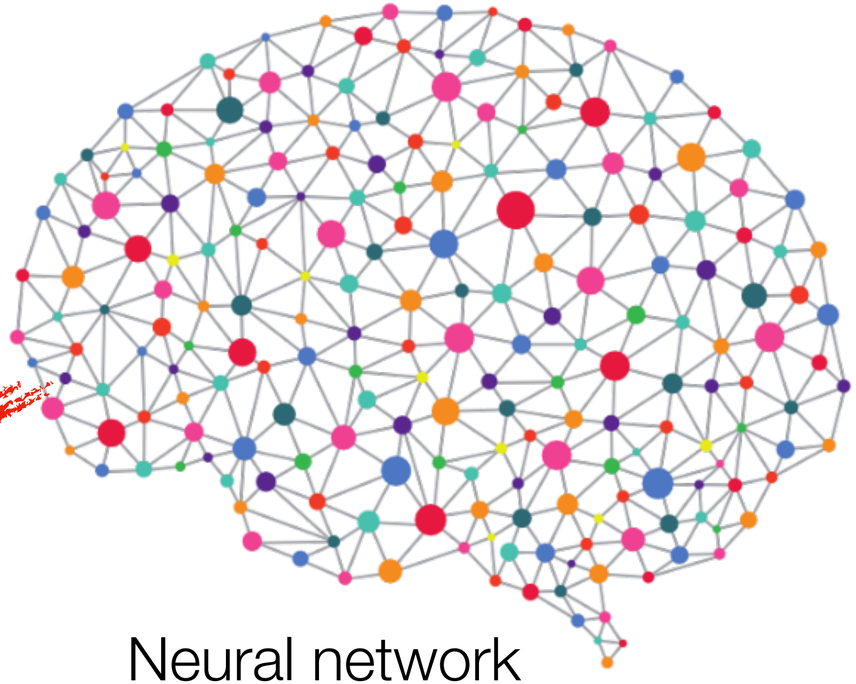


Training data

Arbitrarily distributed

Optimize

$\nabla_\theta \ln p(x;\theta)$



Neural network

Training with data from the correct distribution is simple

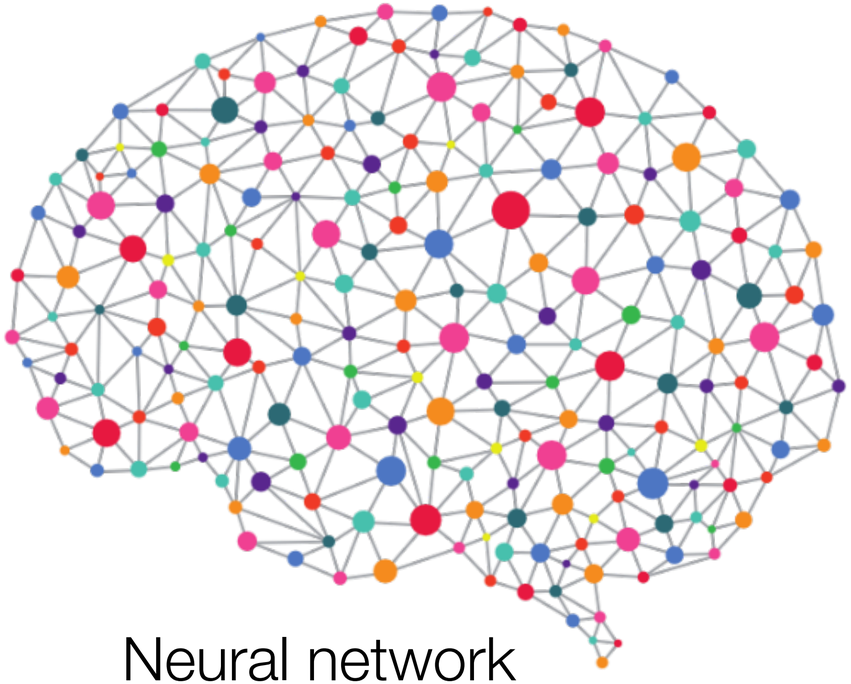
Training data

Desired distribution

Optimize

$$\nabla_\theta \log p(x;\theta)$$

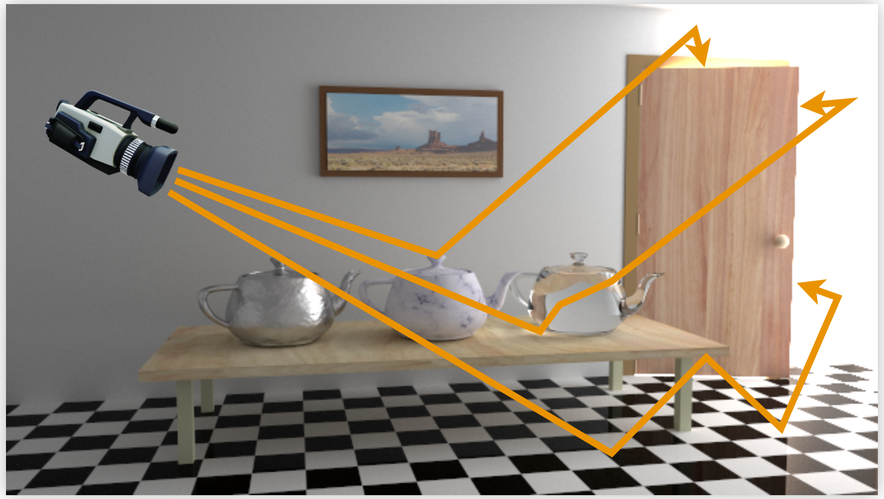
min KL-divergence



Neural network

Training from Monte Carlo samples requires careful weighting

$p(x;\theta)$ distributed

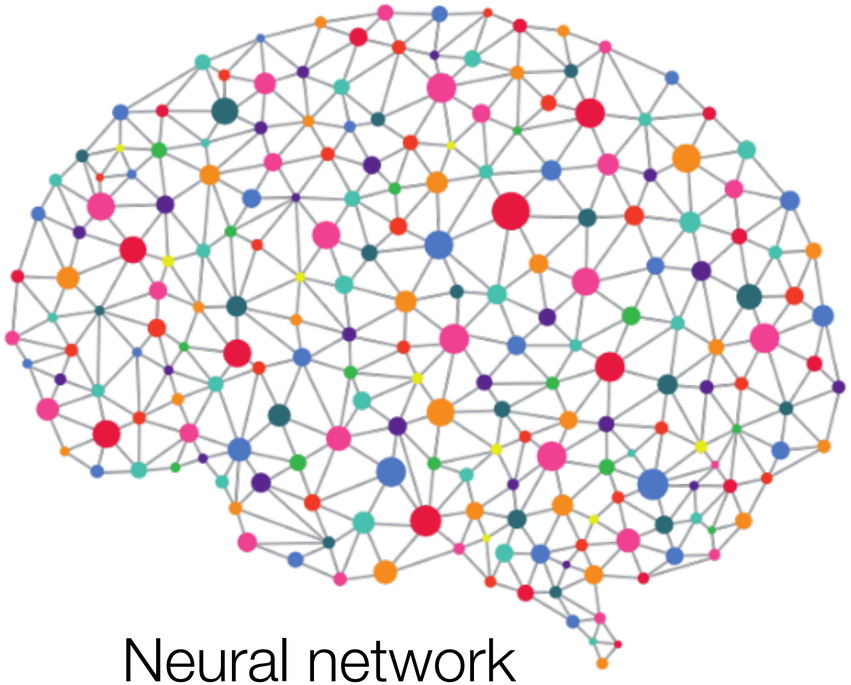


Training data

Optimize

$$\frac{f(x)}{p(x;\theta)}\nabla_\theta \log p(x;\theta)$$

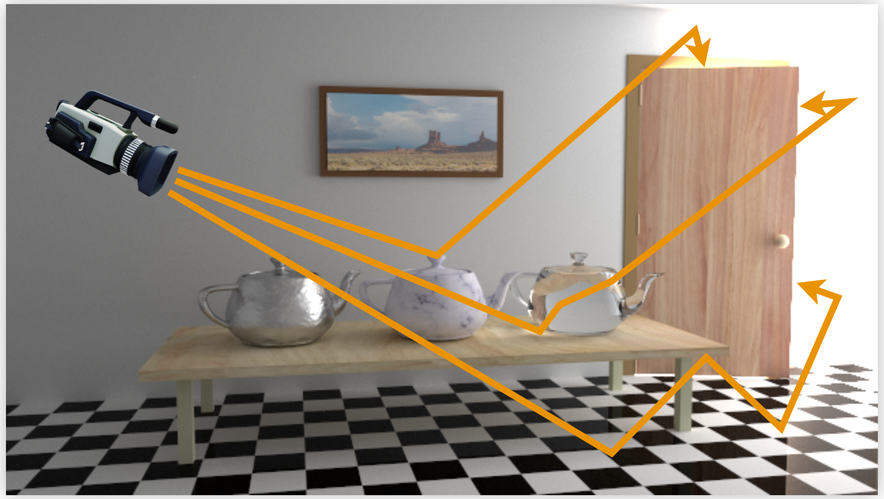
min KL-divergence



Neural network

Training from Monte Carlo samples requires careful weighting

$p(x;\theta)$ distributed

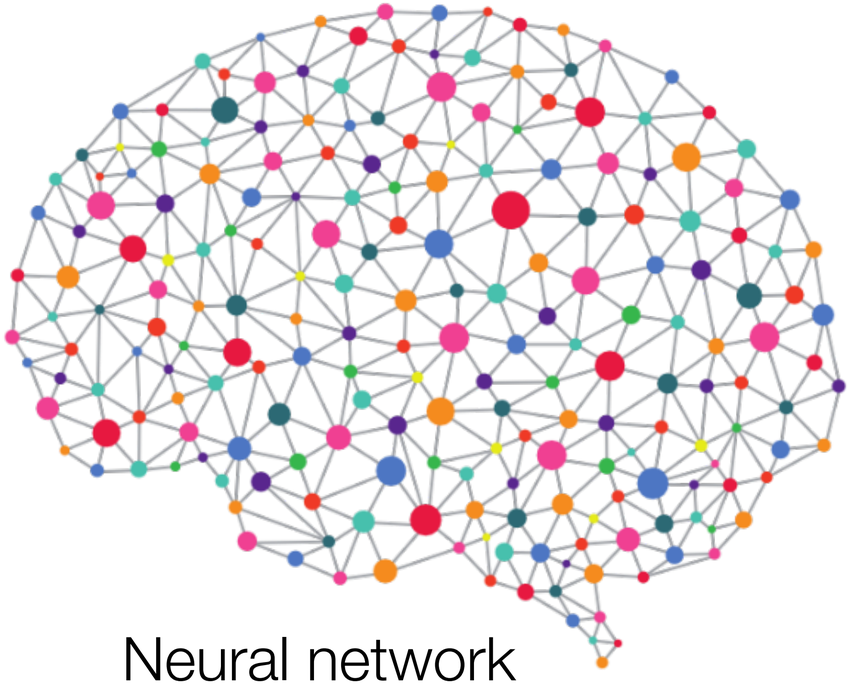


Training data

Optimize

$$\frac{f(x)^2}{p(x;\theta)^2}\nabla_\theta \log p(x;\theta)$$

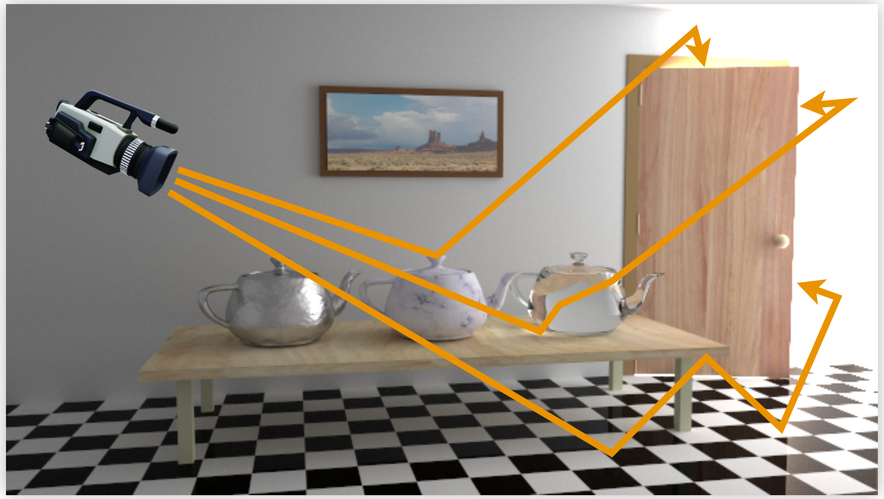
min variance



Neural network

Training from Monte Carlo samples requires careful weighting

$p(x;\theta)$ distributed

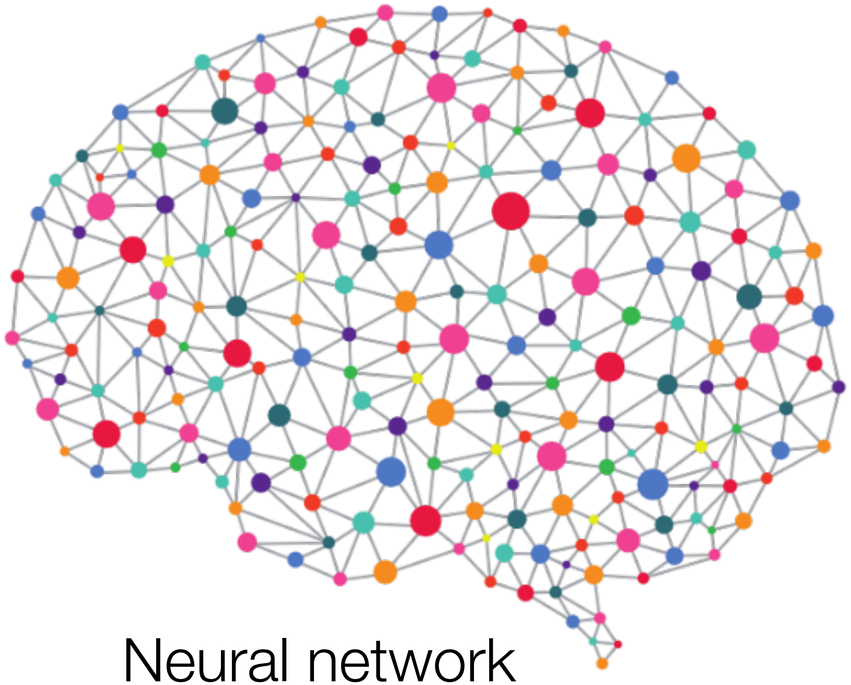


Training data

Optimize

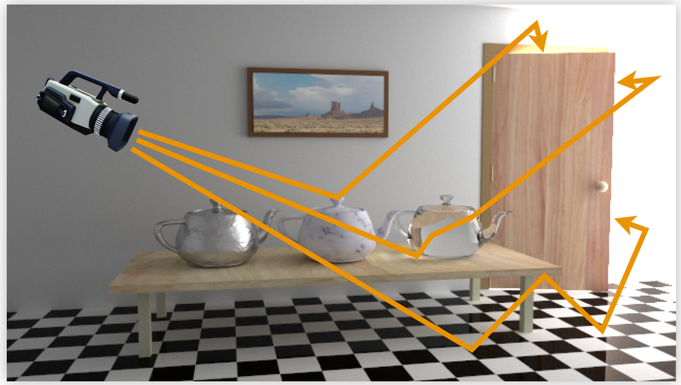
$$\frac{f(x)^2}{p(x;\theta)^2}\nabla_\theta \log p(x;\theta)$$

min χ^2-divergence



Neural network

Putting it together...

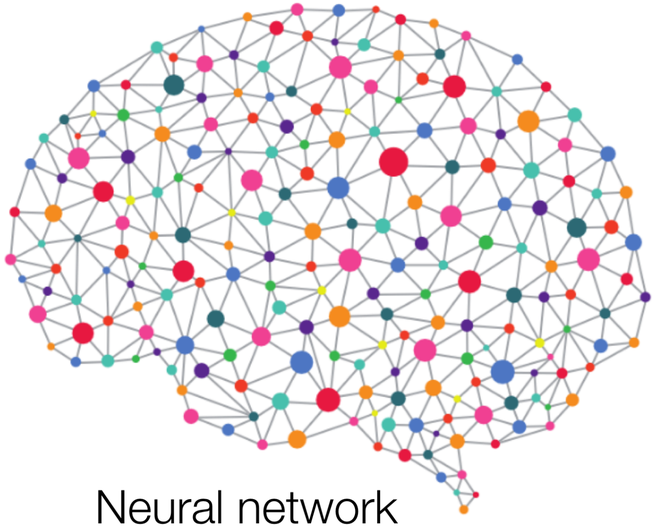


Path tracer

Sample

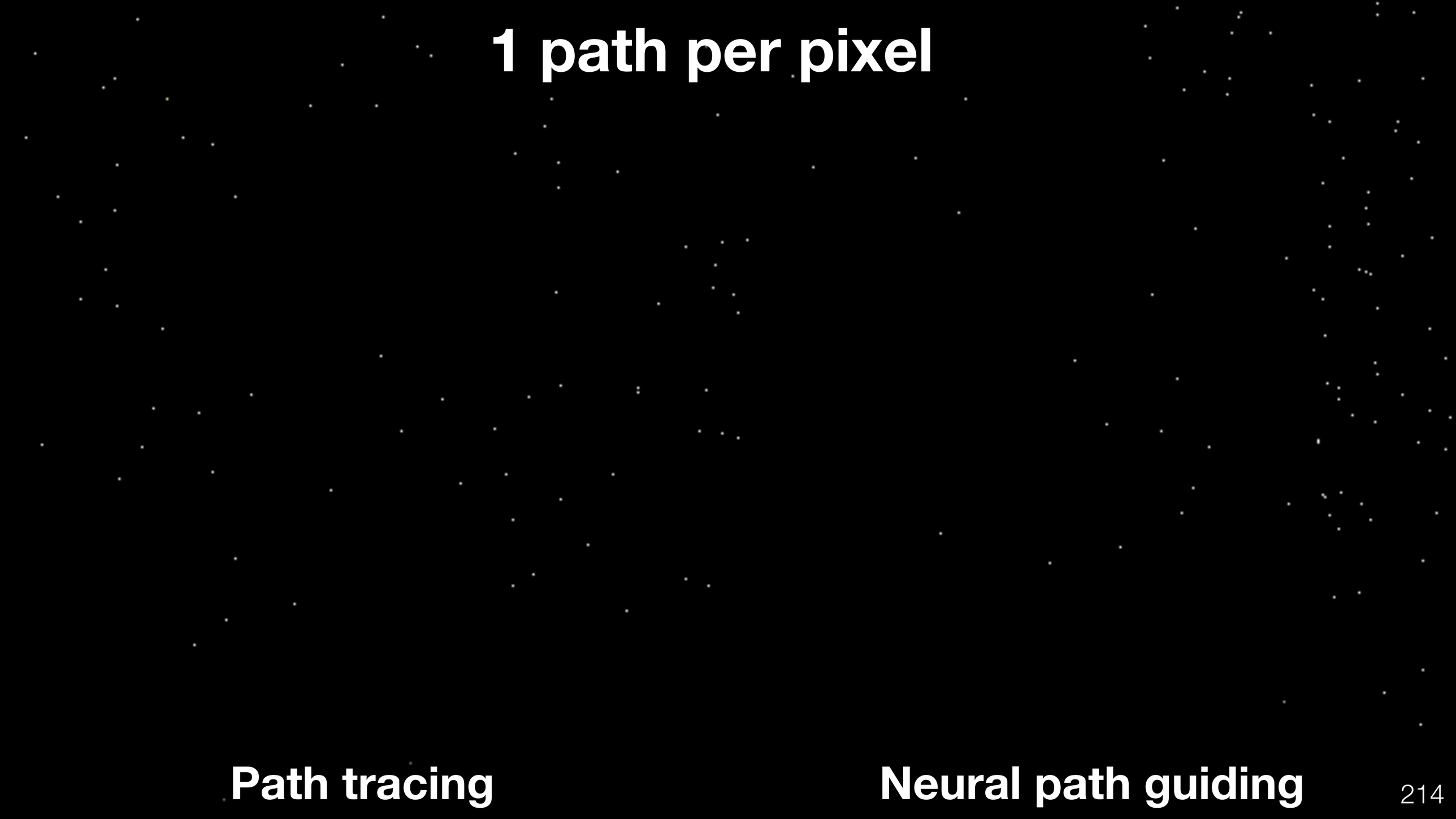
Feedback loop

Optimize



Neural network

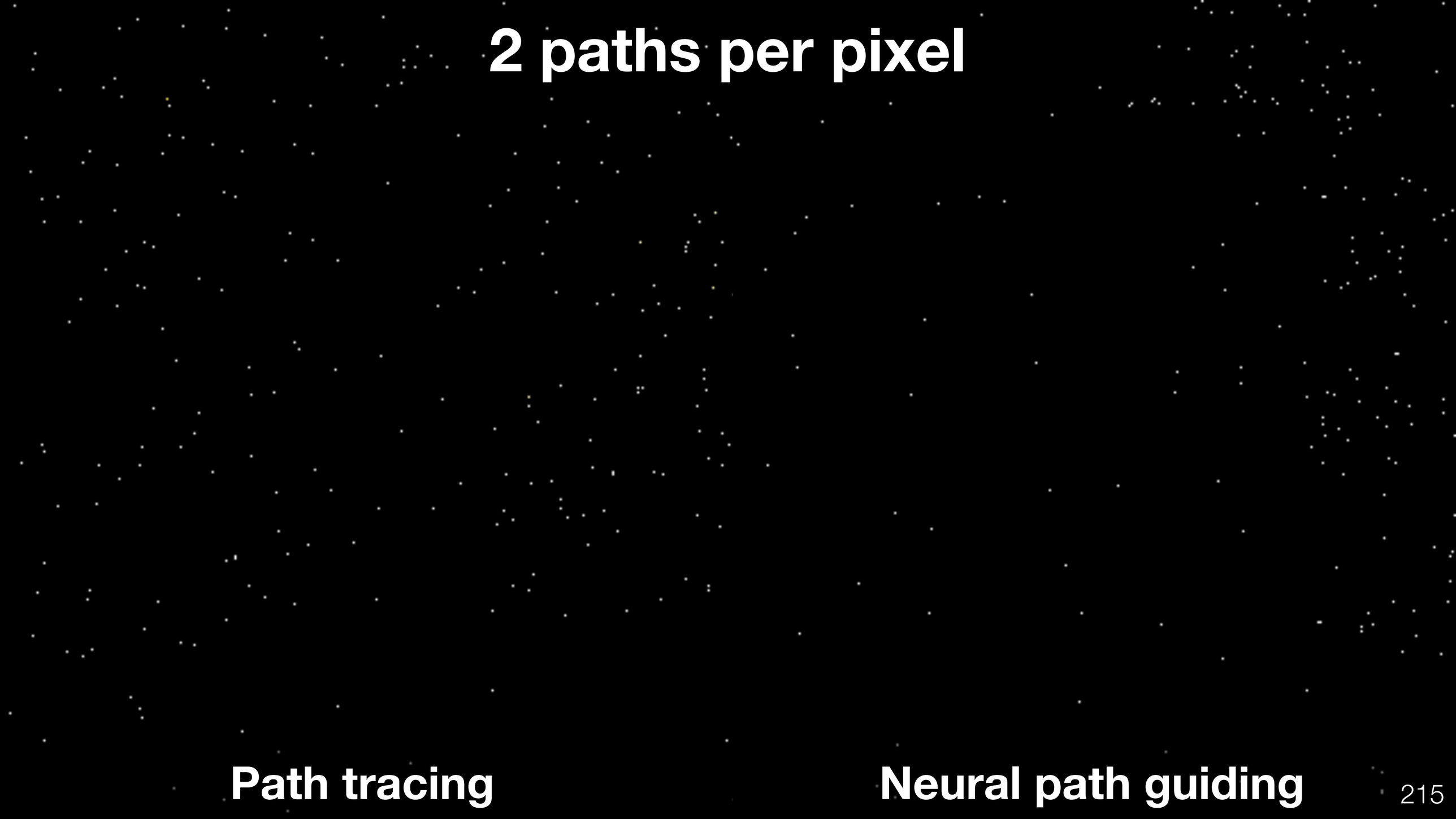
1 path per pixel



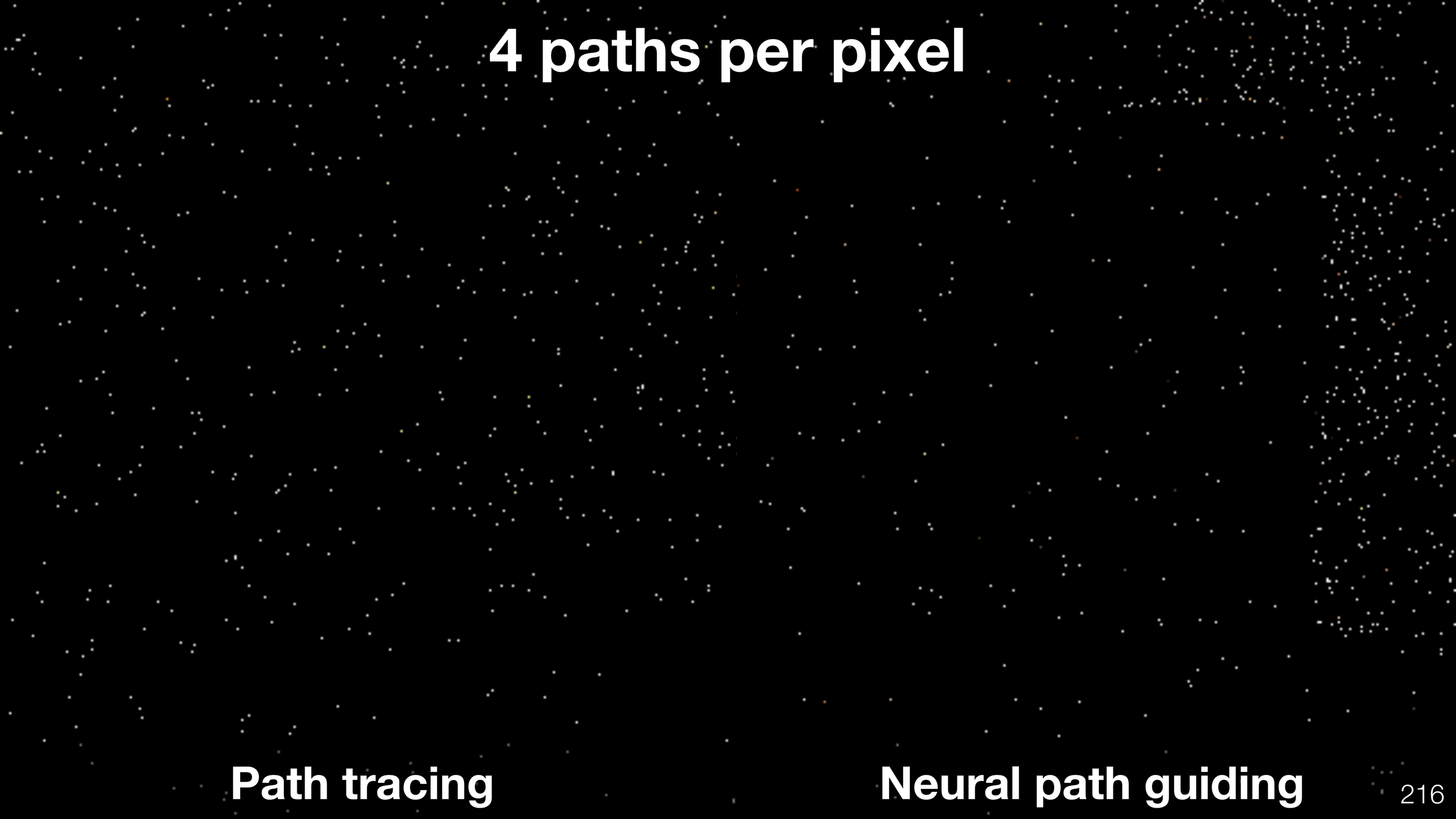
Path tracing

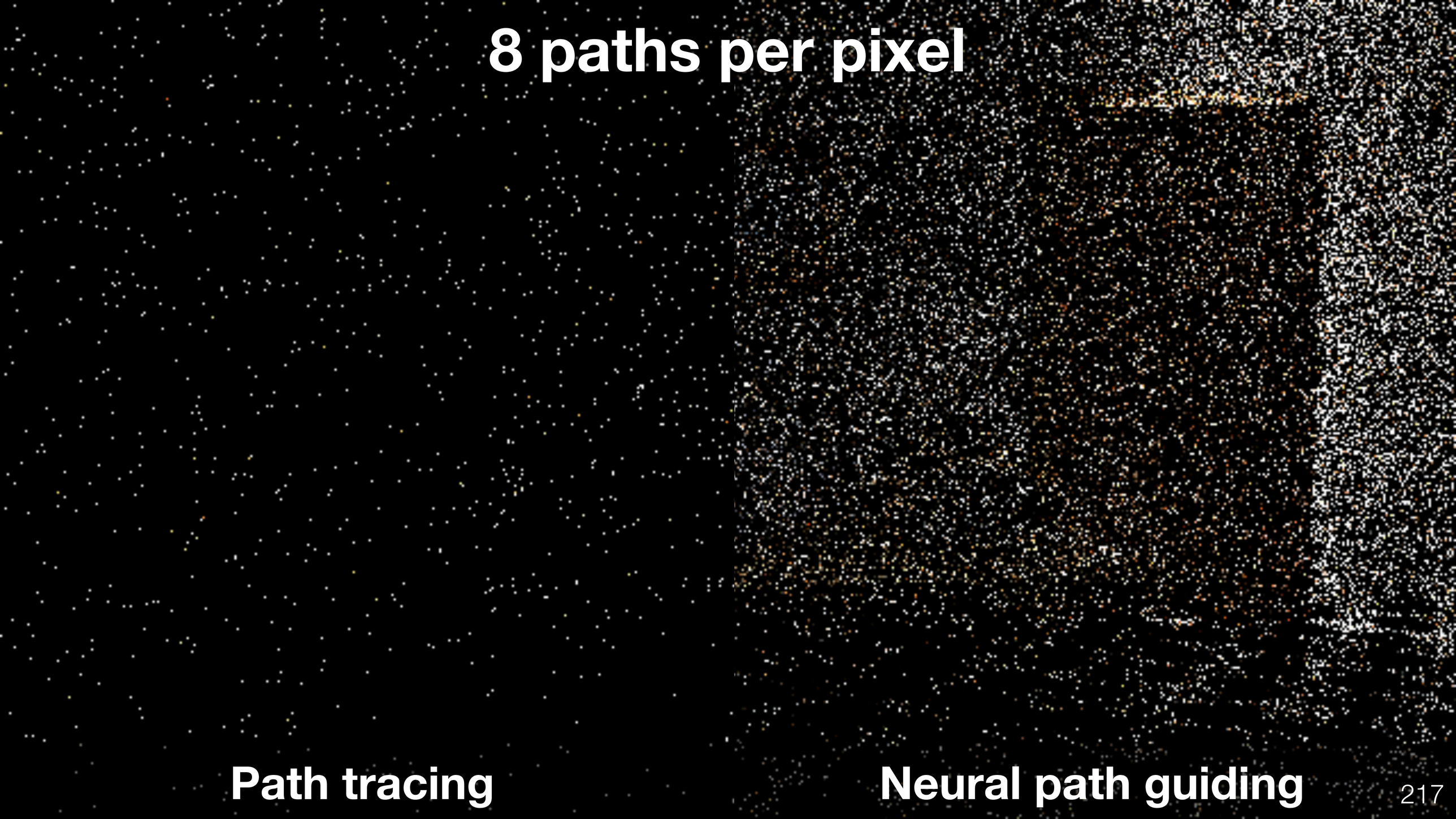
Neural path guiding

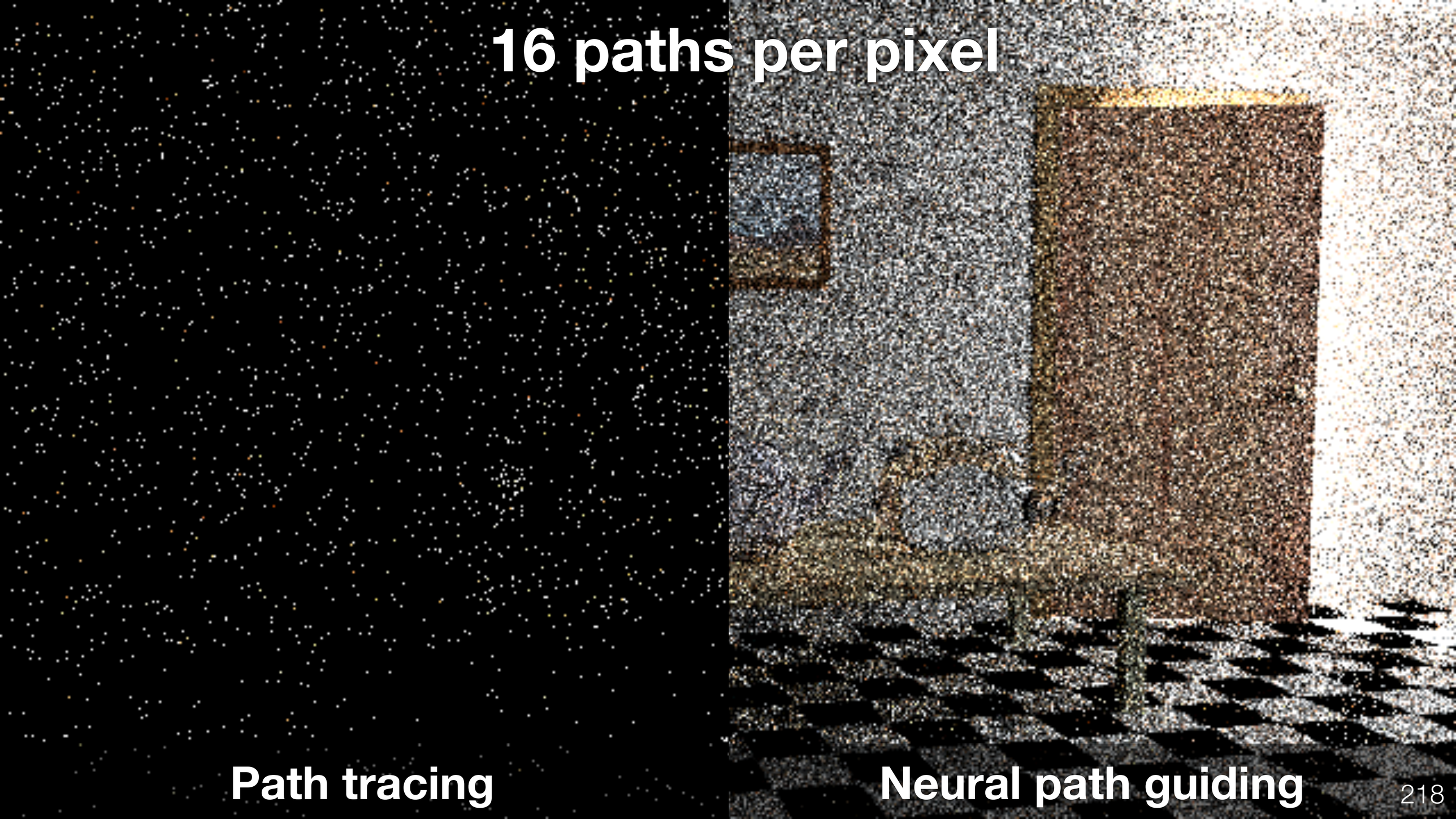
2 paths per pixel

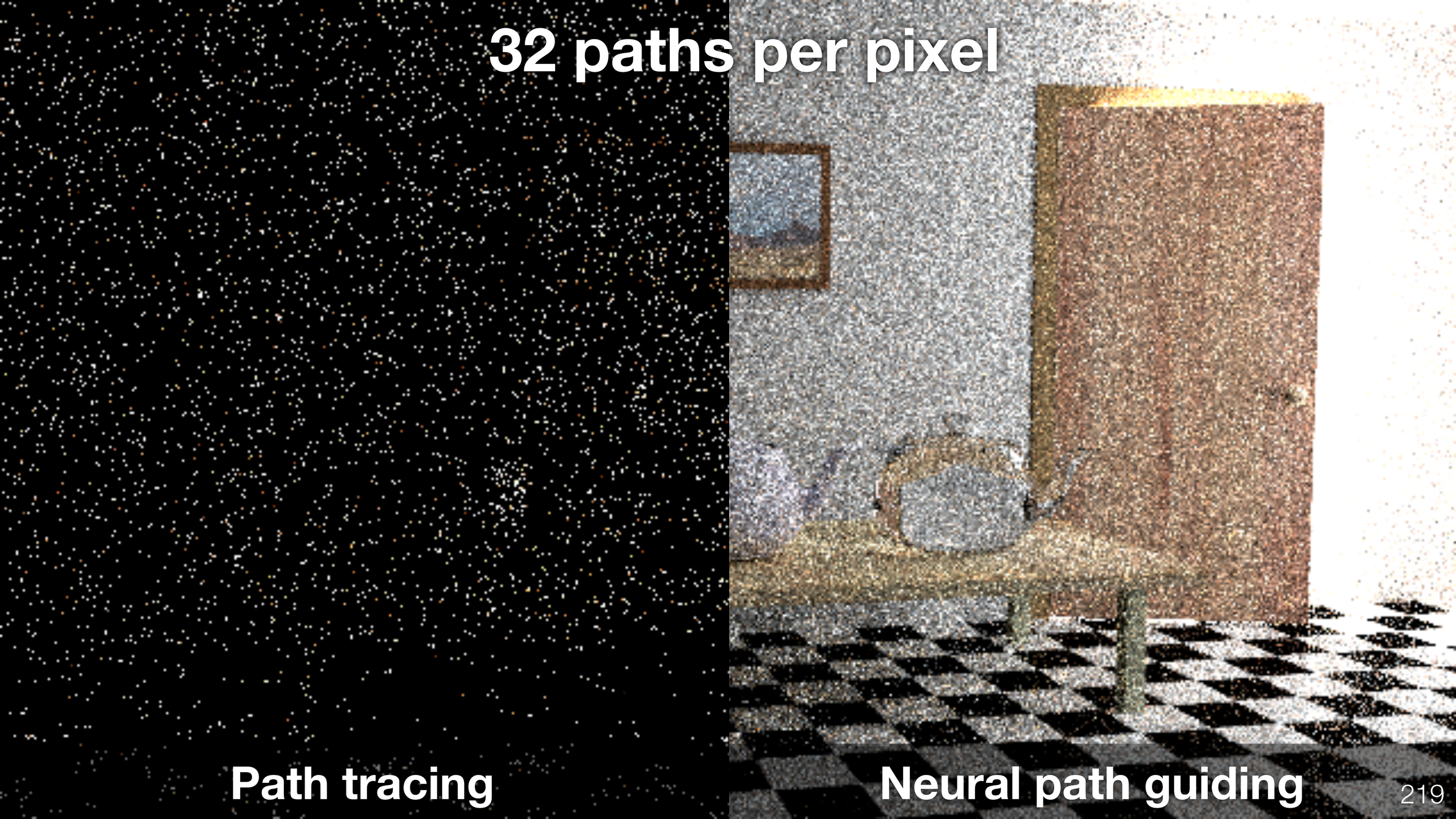


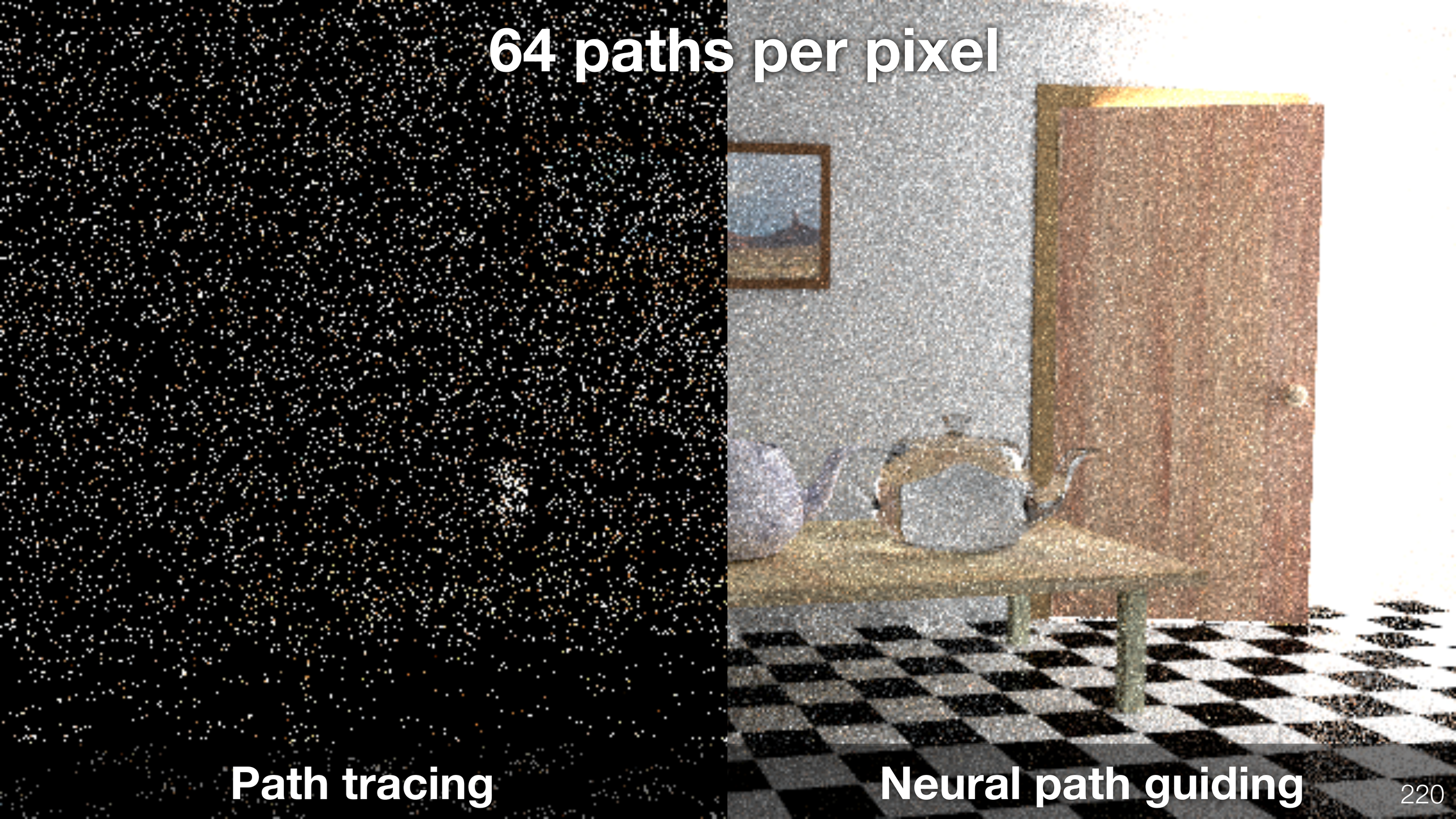
Path tracing **Neural path guiding**

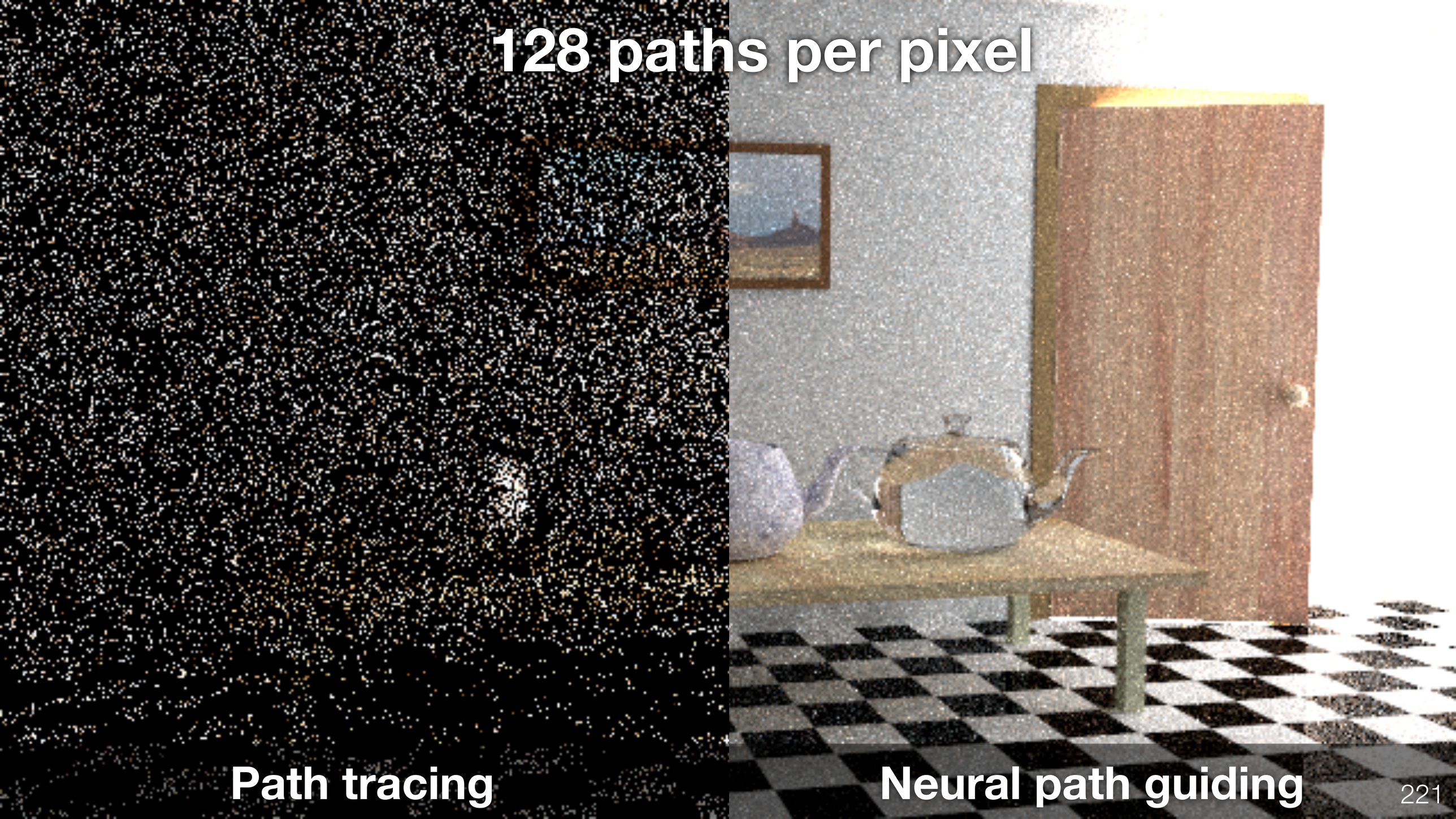

4 paths per pixel
Path tracing
Neural path guiding

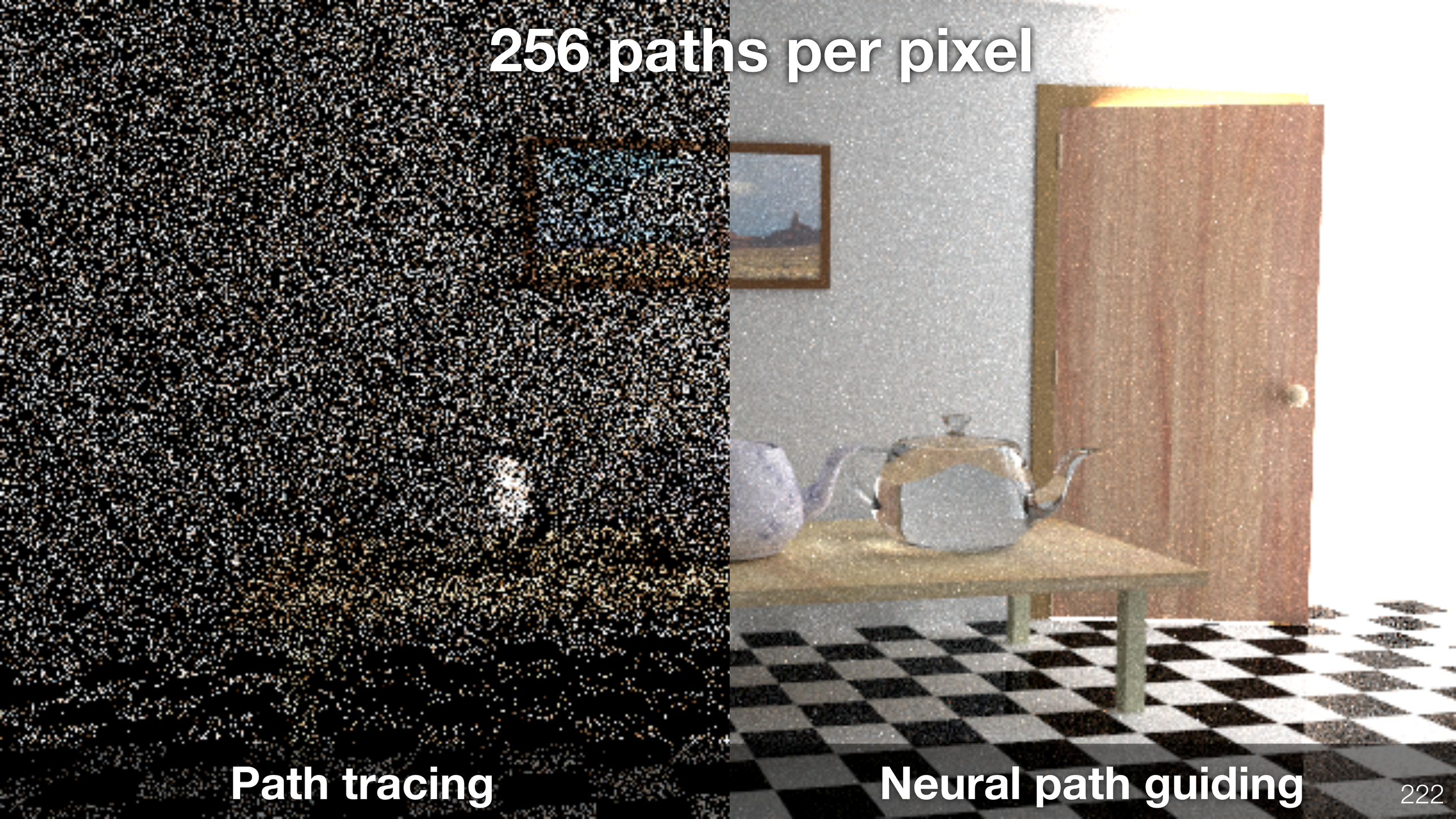

8 paths per pixel
Path tracing
Neural path guiding

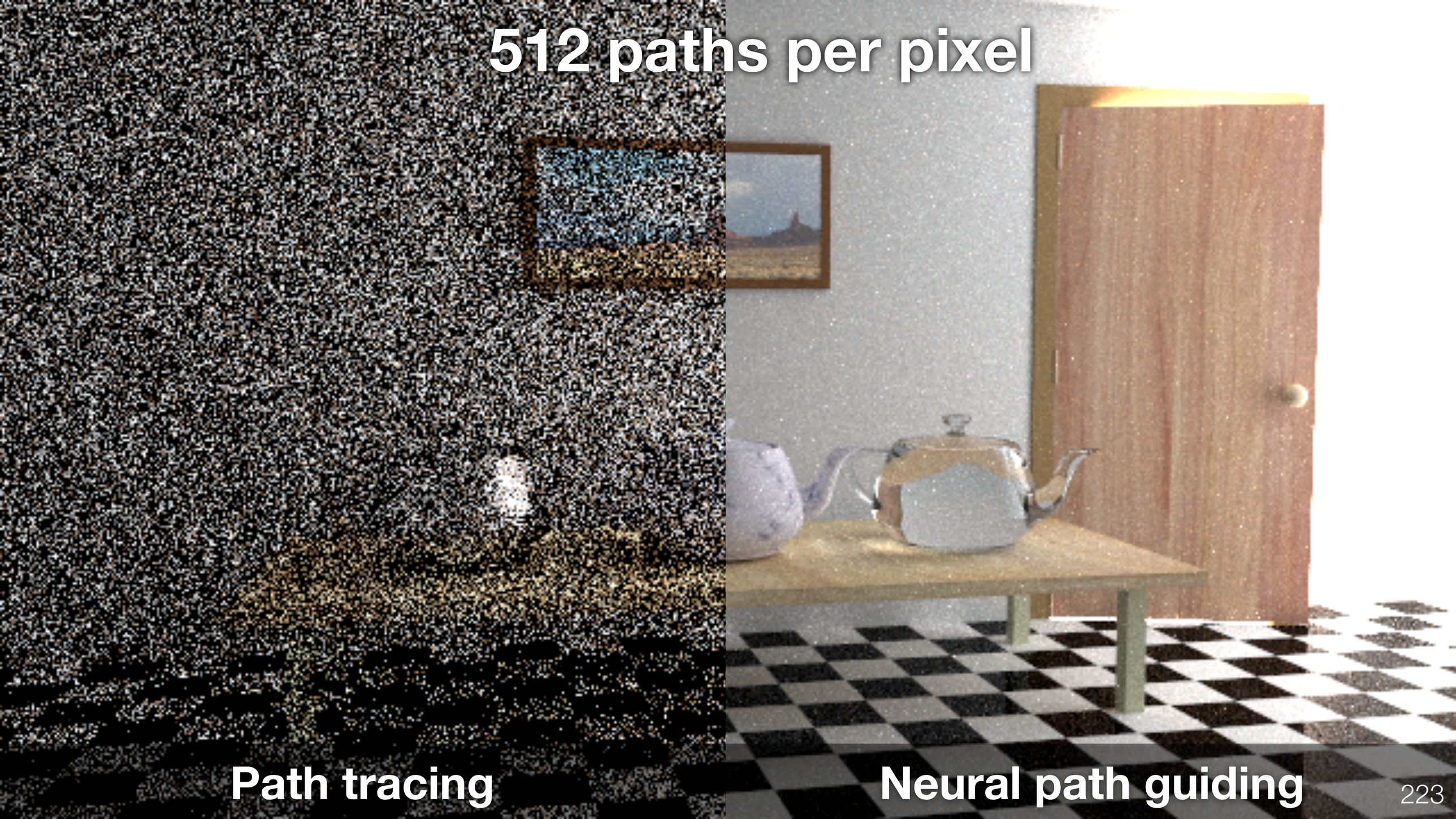

16 paths per pixel
Path tracing
Neural path guiding


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

Conclusion

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

Acknowledgments

Thanks to Chaitanya et al. and Muller et al. for making their slides publicly available.