Correlated error distribution

Corentin Salaun

This lecture

- Image space noise correlation using sampler
- Rendering setup
- Sample optimization before rendering
- Scene aware optimization

Reminders

Correlated sampling & image perception

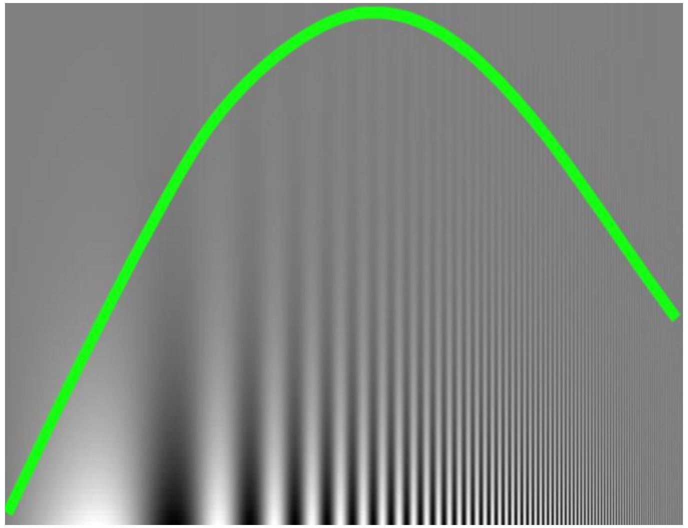
Correlated sampling

- Sample correlation can be used to reduce Monte Carlo error per pixel
  - Use smoothness of the function to distribute samples
  - Uniformly distribute Monte Carlo samples
  - Generally based on error upper bound (KH-inequality)

- Correlation rely on knowledge/assumption about the integration problem
  - Use the random process only for sub-part of the problem

Perception based error

- Mathematical error (MSE, RelMSE) consider all pixel independently
  - Measure the quality of the per pixel estimation
- Image perception is more important
  - All pixel seen at the same time
  - They are not independent
- We are more sensitive to some frequency than other
  - High frequency are naturally filterer by the eye



Motivation

Perceptual error distribution

Image dithering

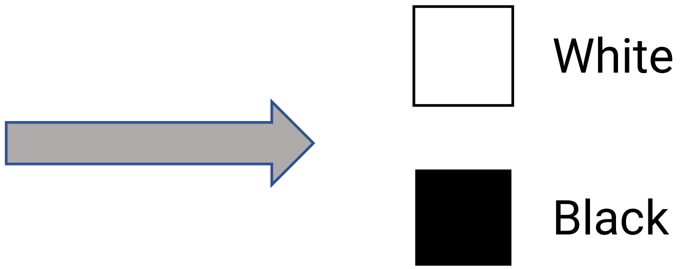


Image dithering

White if (input > dither)

Black else

Image dithering

Threshold

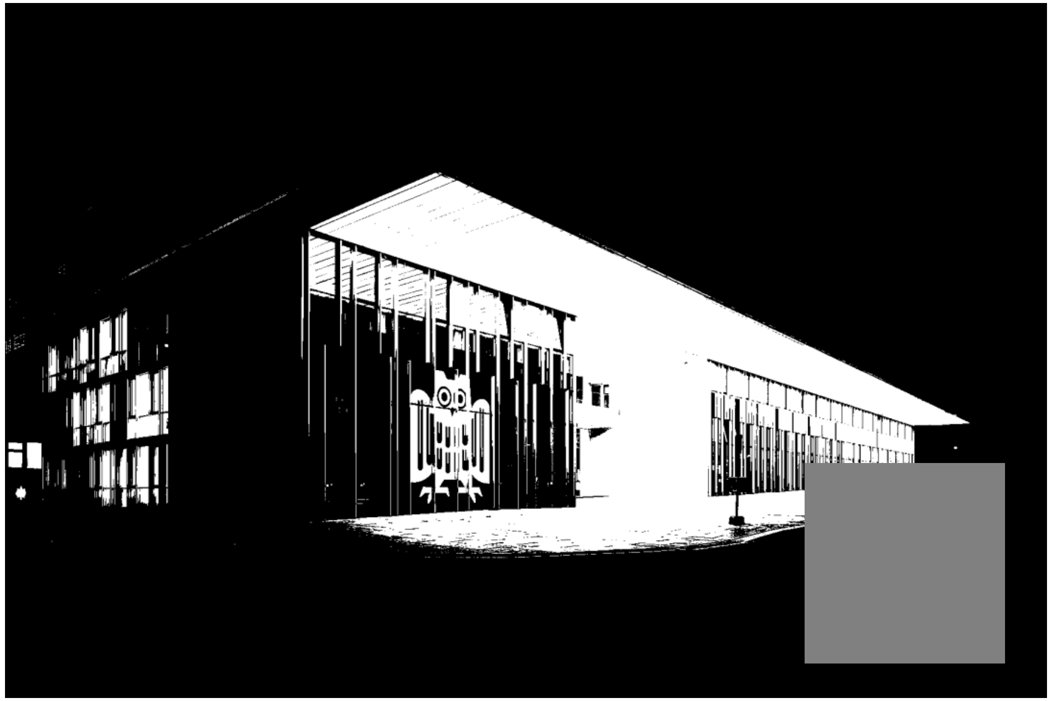


Image dithering

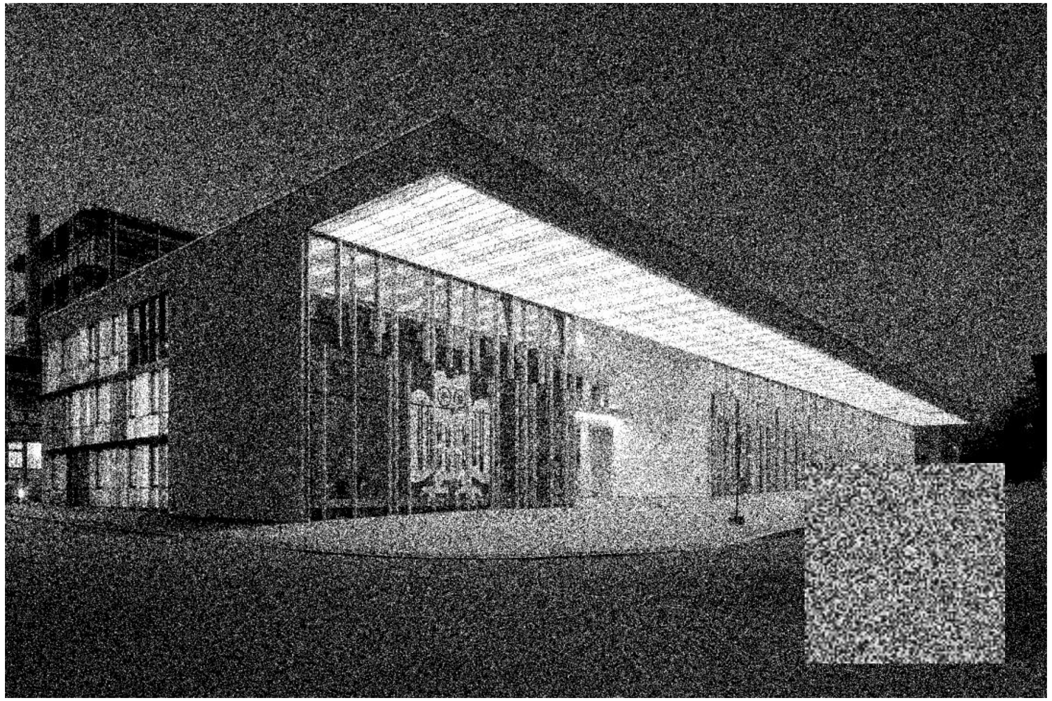


Image dithering

Threshold

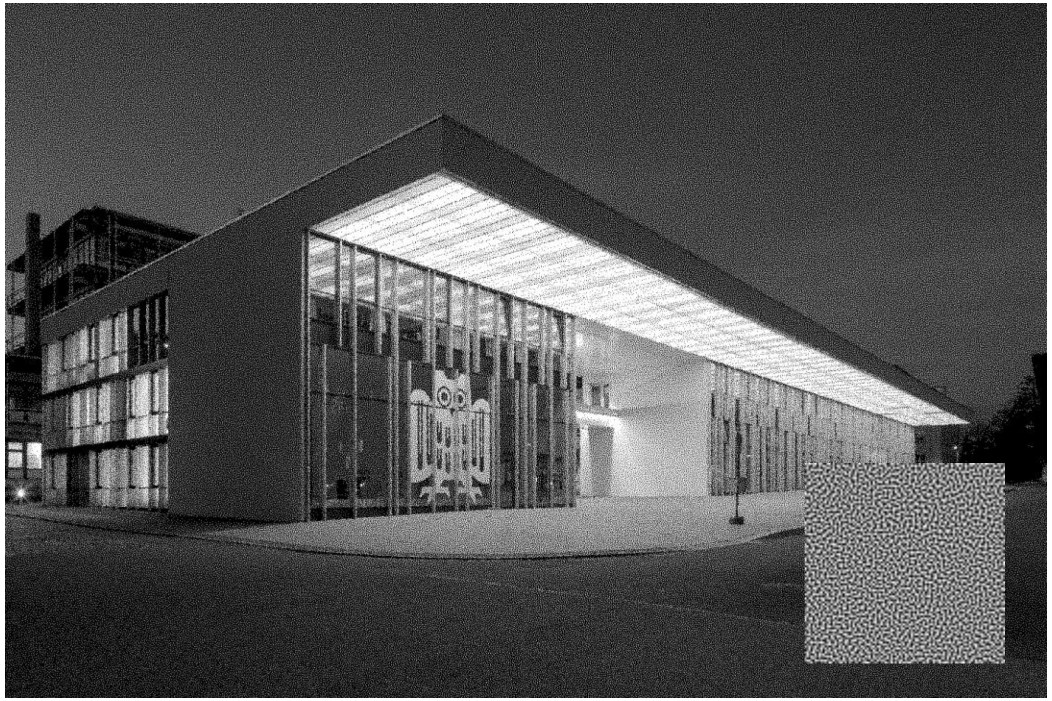
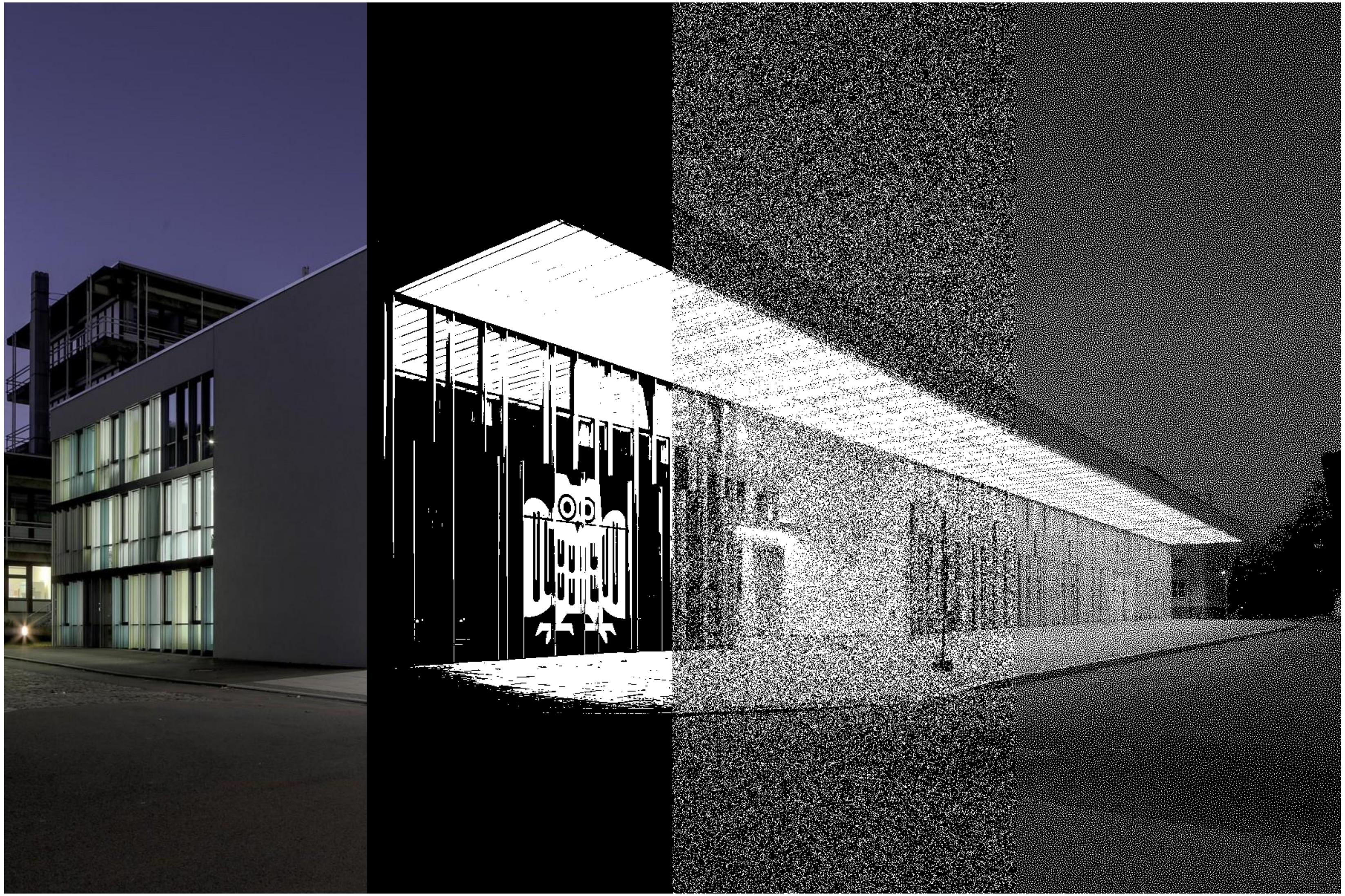
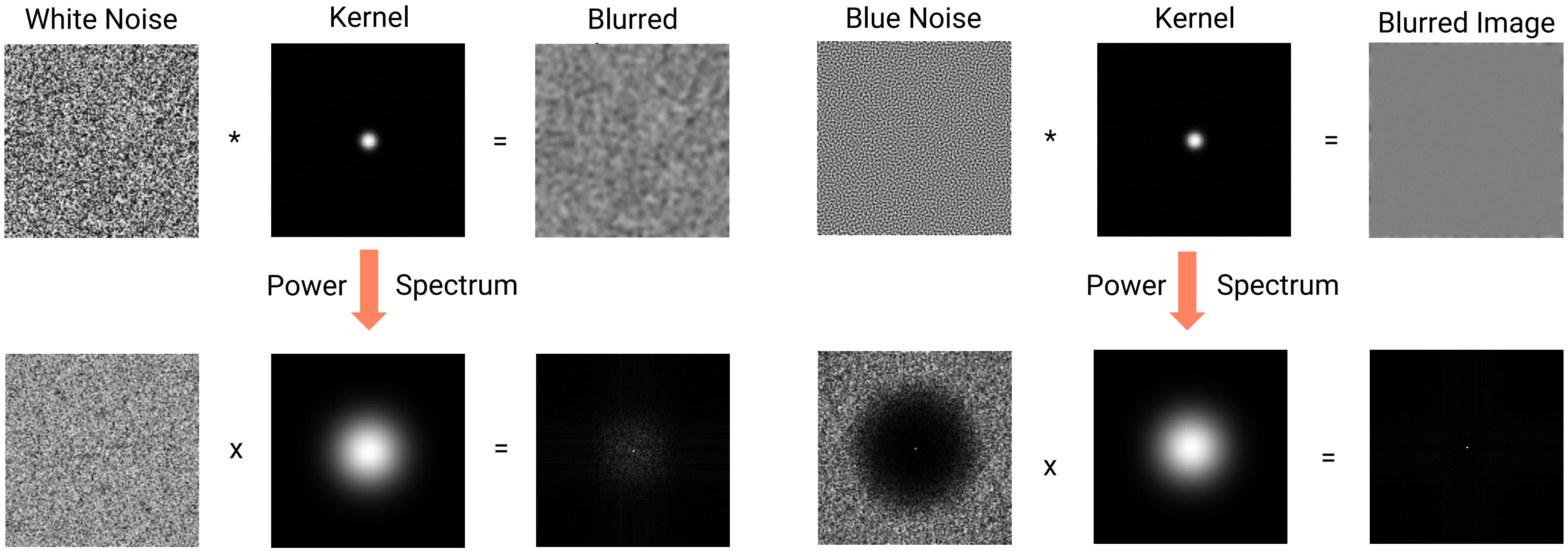


Image dithering



Noise filtering

- Denoising BN image produce less error
- Our eye works as a low pass filter



Better perceptually and numerically!

Dithering for rendering

Consider a rendering where all pixel are simulated independently

- A pixel value only depend on the sample of that pixel
  - Works for forward rendering (pathtracing, direct lightning, …)
  - Doesn’t work for backward, mixed rendering (BDPT, … as light traced samples contribute to all pixel)

- Close pixel render similar function (smooth scene)
- Samples can be interpreted as sequence
  - Generalize to good sample sequence (Rank1, PMJ02, Sobol, …)

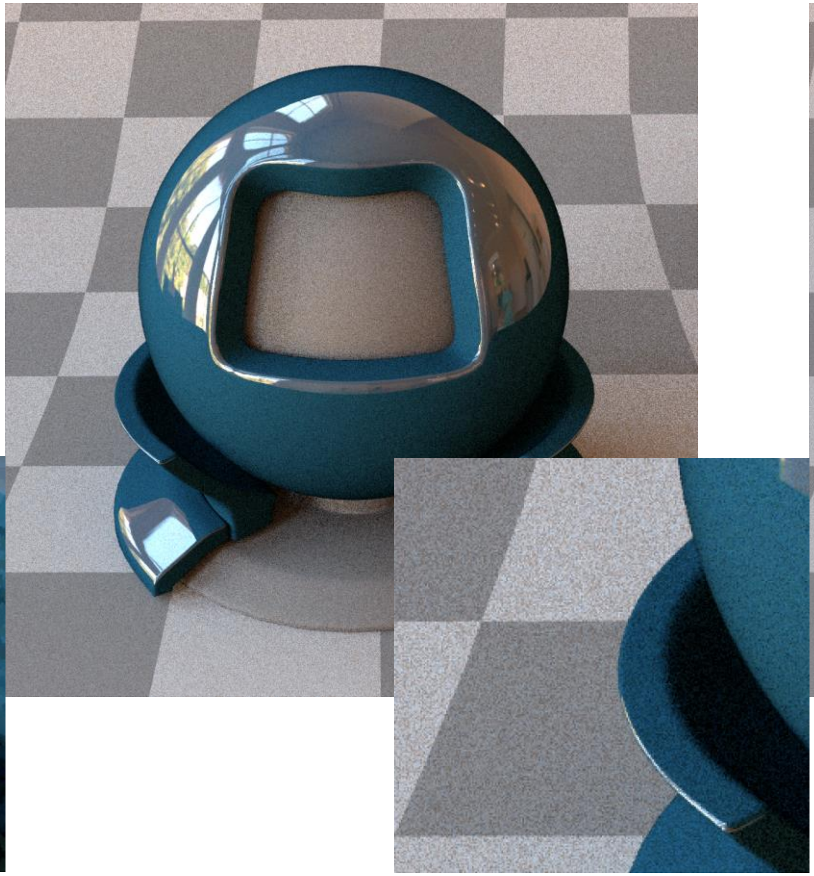
- Is it possible to optimize the sample distribution to improve rendering ?
  - For mean squared error ? No
  - For perception ? Yes

Dithering for rendering

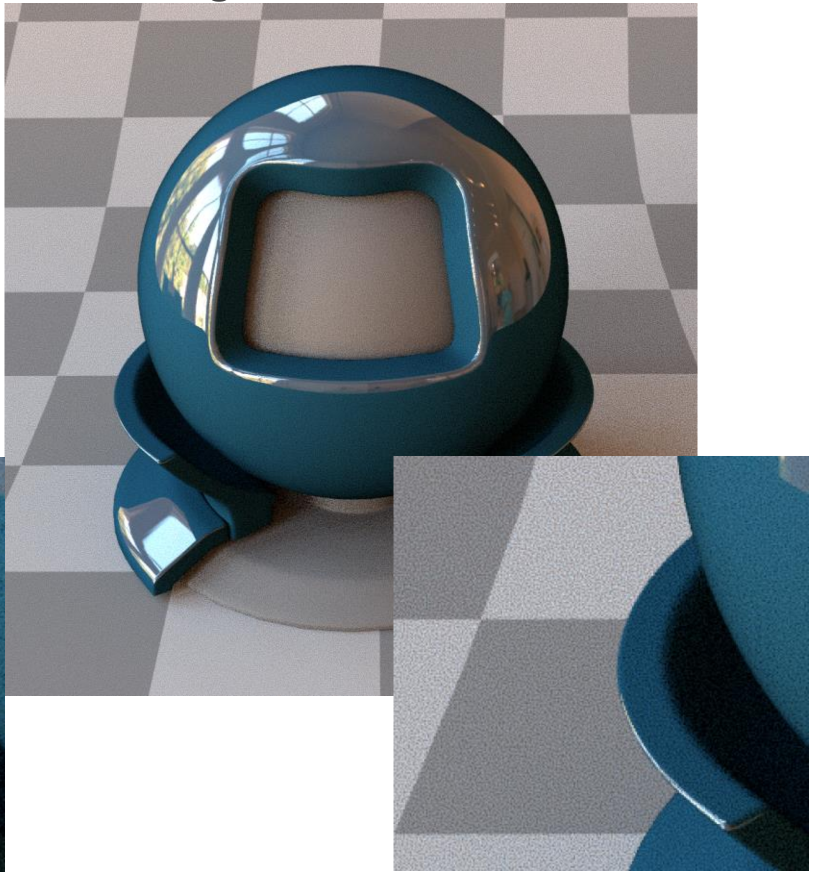
- Can we correlate pixel integration on image plane ?

Positive correlation

No correlation



Negative correlation



Dithering for rendering

What is the desired correlation ?

- Negative correlation is the best correlation
  - It’s based on HVS sensitivity to low frequency
  - High frequency/negative correlation are “blurred” by human eye

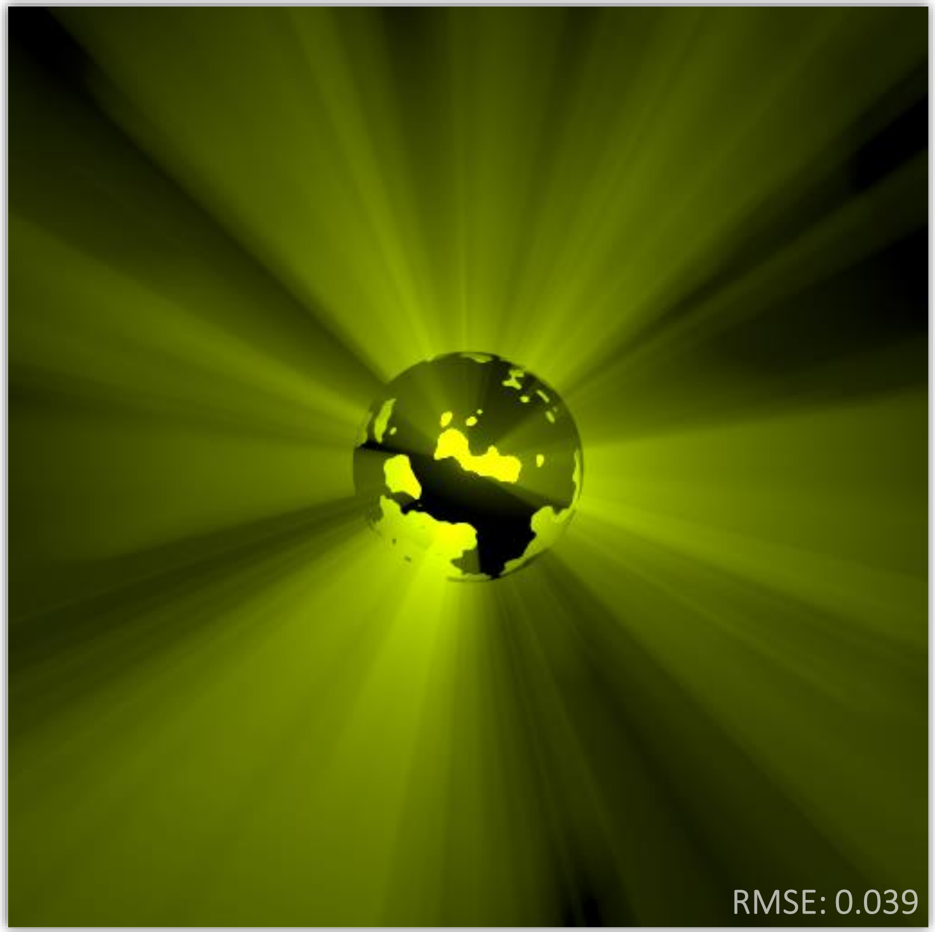
- Positive correlation create artifacts
  - Less noise but some color splat

- Uniform correlation over the image plane
  - Having multiple correlation or quality make them visually unpleasant
  - This looks unnatural

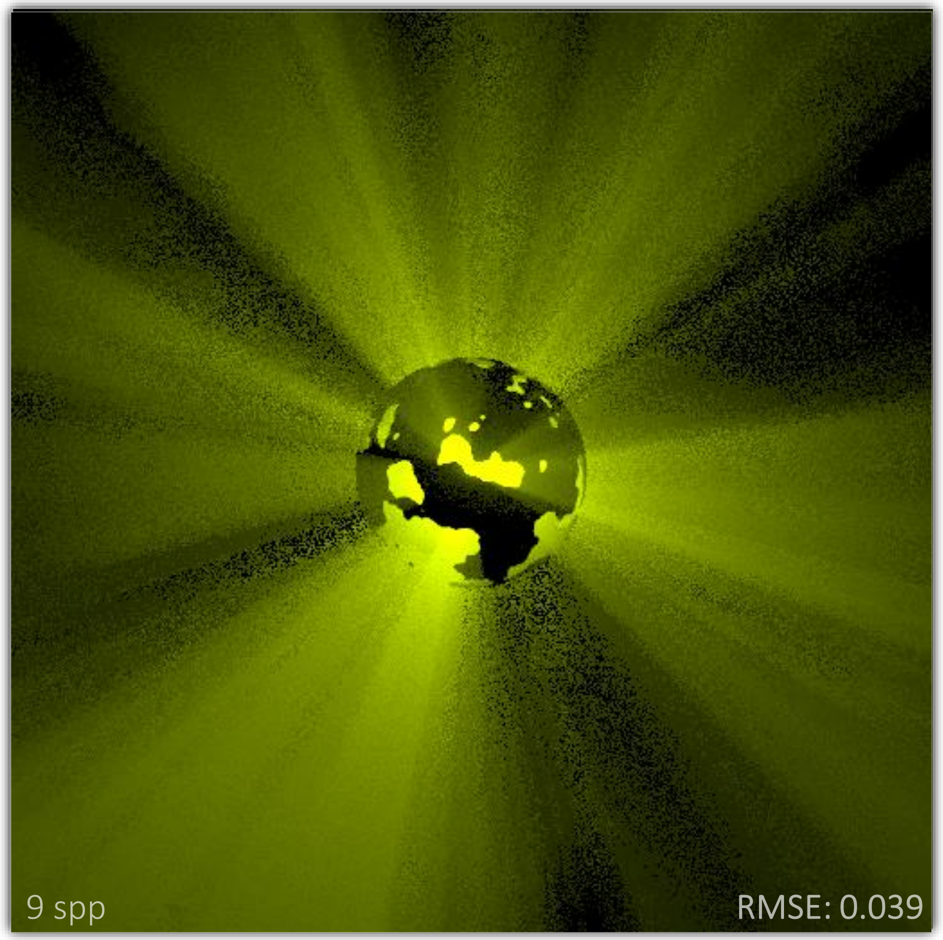
Blue-noise Dithered Sampling

[Georgiev & Fajardo 2016]

Blue noise error distribution



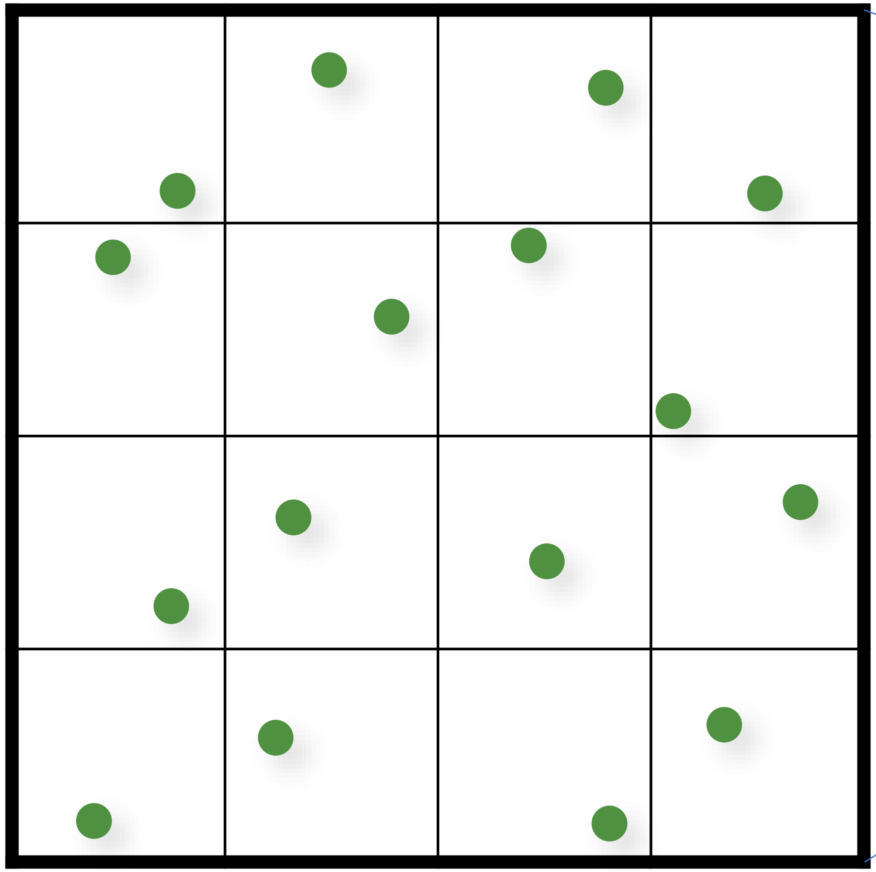
white-noise pixel error distribution

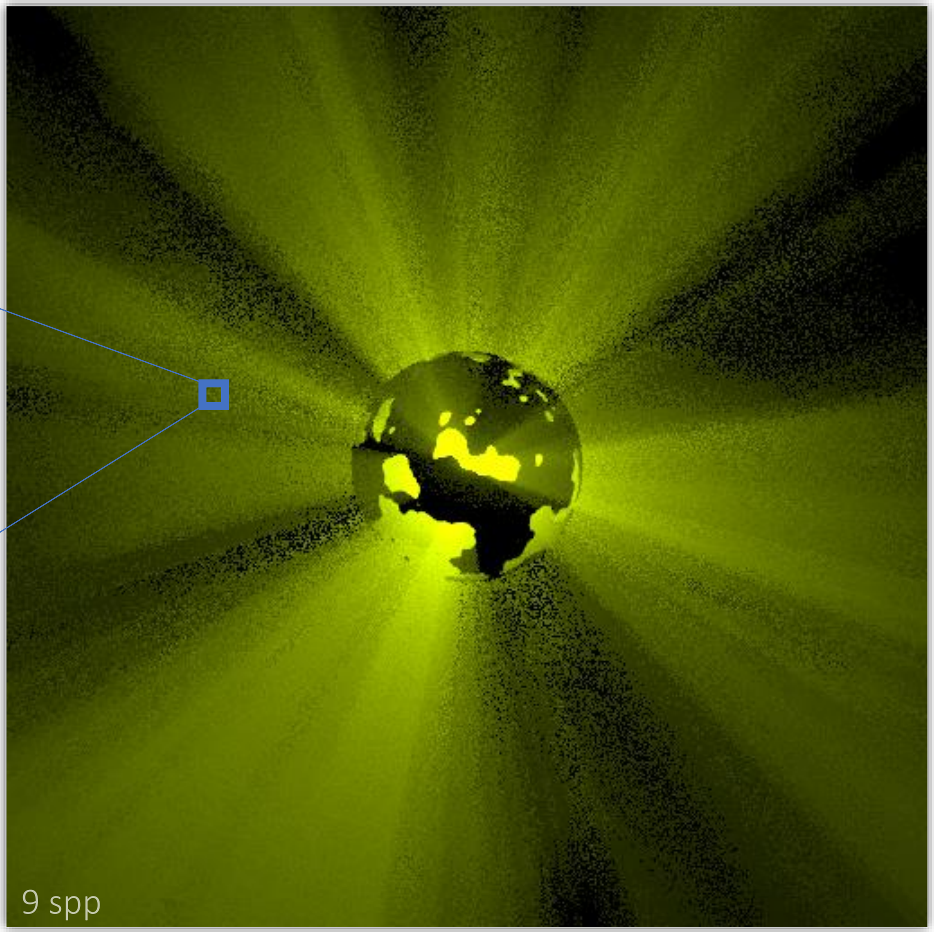


blue-noise pixel error distribution

Blue noise error distribution

Stratified sampling





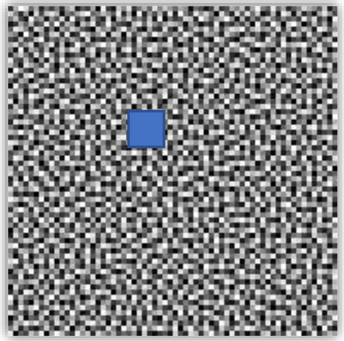
Objectif

- Having similar samples will result in close rendering value
  - Function is smooth

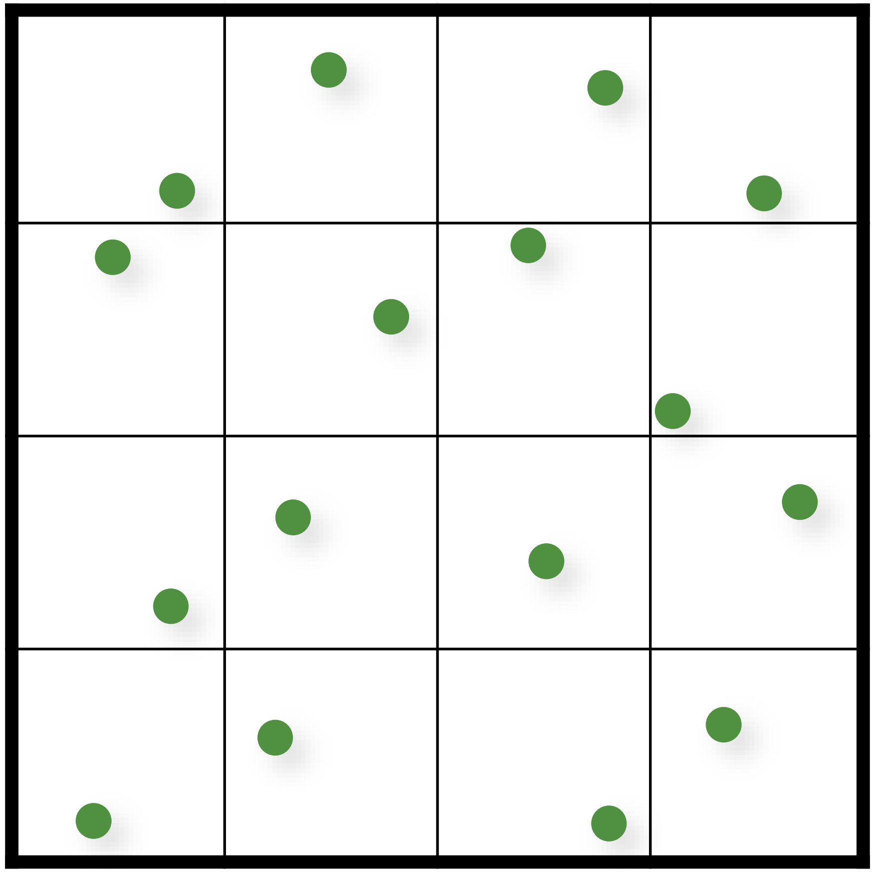
- To get as much different value we want as much different samples as possible
  - While keeping good per pixel quality
  - We expect a correlation between sample distribution and estimate distribution

Sampling with dither masks

1. Create sampling pattern
2. For each pixel
   a. Look-up value from dither mask (tiled over the image)
   b. Use value to offset sample pattern

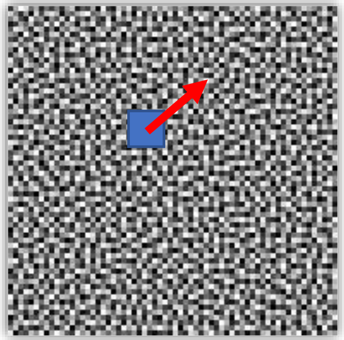


Stratified sampling

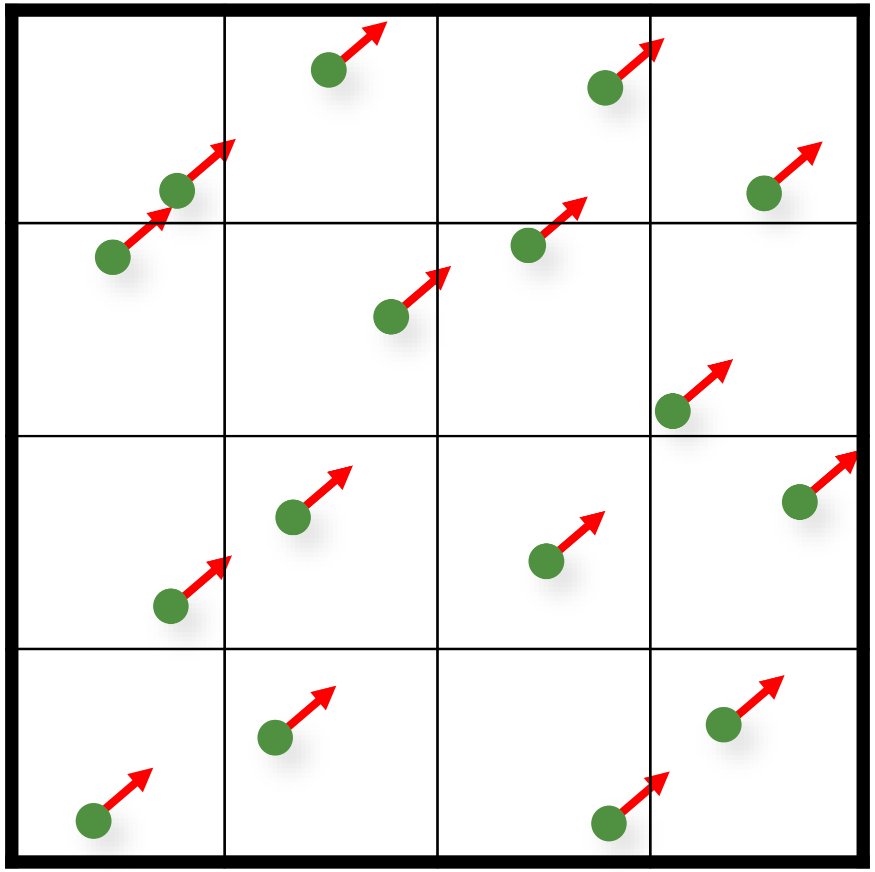


Sampling with dither masks

1. Create sampling pattern
2. For each pixel
   a. Look-up value from dither mask (tiled over the image)
   b. Use value to offset sample pattern

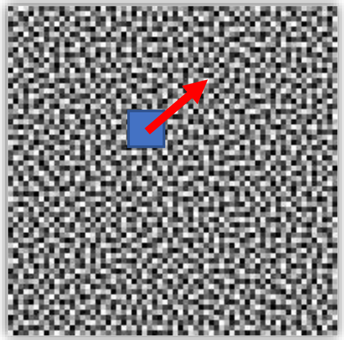


Stratified sampling

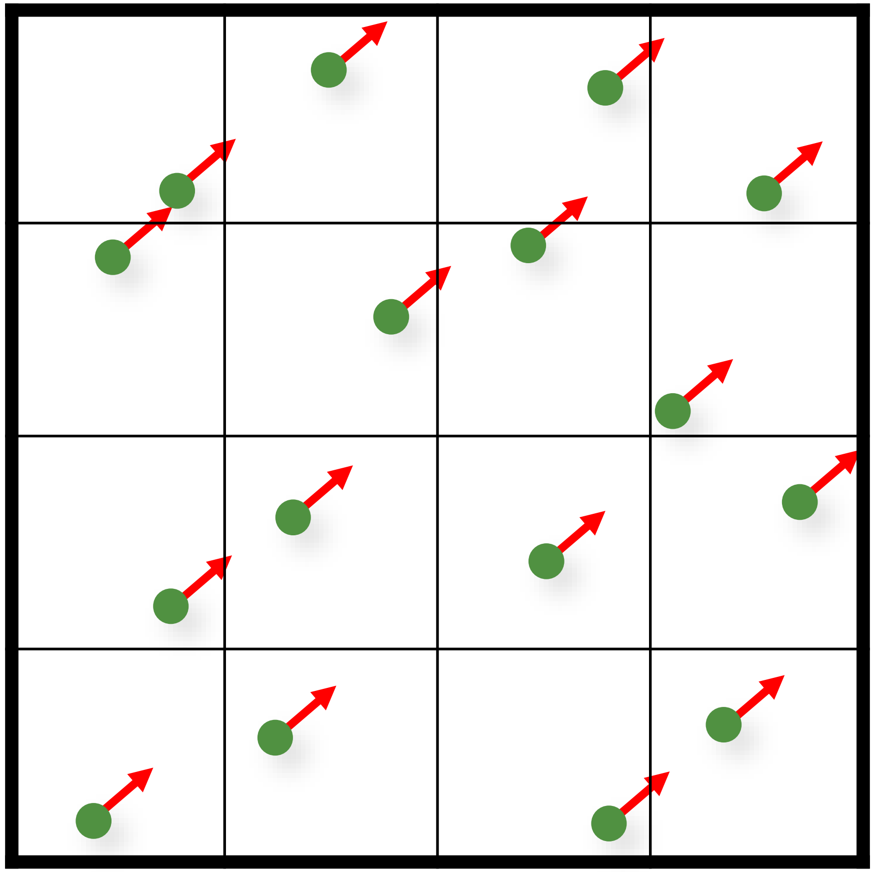


Sampling with dither masks

1. Create sampling pattern
2. For each pixel
   a. Look-up value from dither mask (tiled over the image)
   b. Use value to offset sample pattern

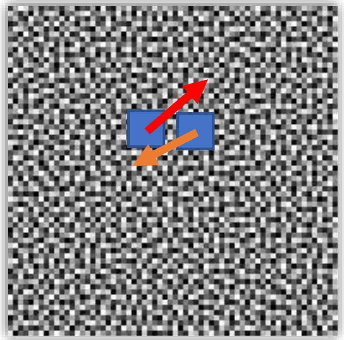


Stratified sampling

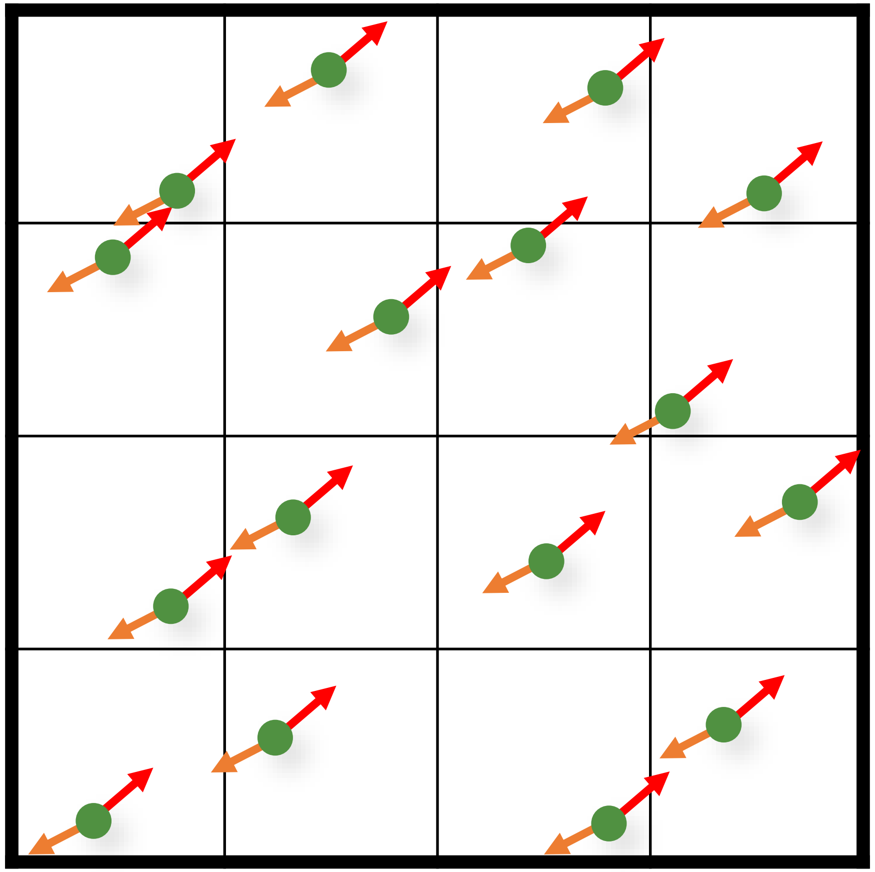


Sampling with dither masks

1. Create sampling pattern
2. For each pixel
   a. Look-up value from dither mask (tiled over the image)
   b. Use value to offset sample pattern



Stratified sampling



Sampling with dither masks

- Dithered sampling ensure similar offset will result in similar sample-set
  - Similar samples will produce similar rendering if the functions are similar
  - Shift can be added directly inside the renderer
  - Dither mask is a new input to the renderer

- Negative pixel correlation
  - Shift map should be as different as possible
  - Shift map should be “unbiased”
  - It’s difficult to optimize it using SGD based method

Simulated annealing

- Optimize shift map by swapping pixels
  - Initialize with random shift map
  - Try swapping pixel to improve some energy
  - Iterate

- The dither mask can be optimize on a small scale and tile over the image plan
  - Need a toroidal optimization
  - Can create some visual artifact as the noise pattern repeat

Simulated annealing

Pseudo code

$I \leftarrow \mathrm{RandomInitialization}()$
for $k \in \{0, k_{max}\}$ **do**
  $p_x, p_y \leftarrow \mathrm{RandomPixel}()$
  $e \leftarrow \boxed{E(I)}$
  $I_{new} \leftarrow \mathrm{swap}(I, p_x, p_y)$
  $e_{new} \leftarrow \boxed{E(I_{new})}$
  if $e < e_{new}$ and $exp(\frac{e - e_{new}}{c_k}) < \mathrm{rand}(0,1)$ **then**
    $I_{new} \leftarrow \mathrm{swap}(I_{new}, p_x, p_y)$
  end if $I \leftarrow I_{new}$
end for
Return I

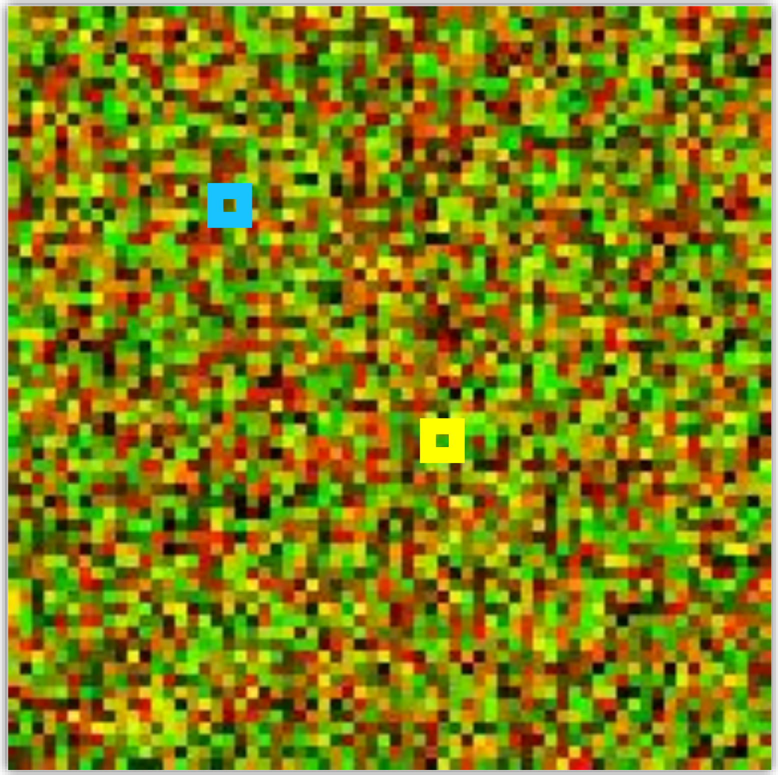
Energy term

$$E(I) = \underbrace{\sum_{p \neq q}}_{\text{pixels}} E(p,q) = \sum_{p \neq q} \underbrace{e^{-\frac{\overbrace{\|i_p - i_q\|^2}^{\text{pixel distance}}}{\sigma_i^2}}}_{\text{image-space Gaussian}} \cdot \underbrace{e^{-\frac{\overbrace{\|s_p - s_q\|^{d/2}}^{\text{sample distance}}}{\sigma_s^2}}}_{\text{sample-space Gaussian}}$$

Simulated annealing

Pseudo code

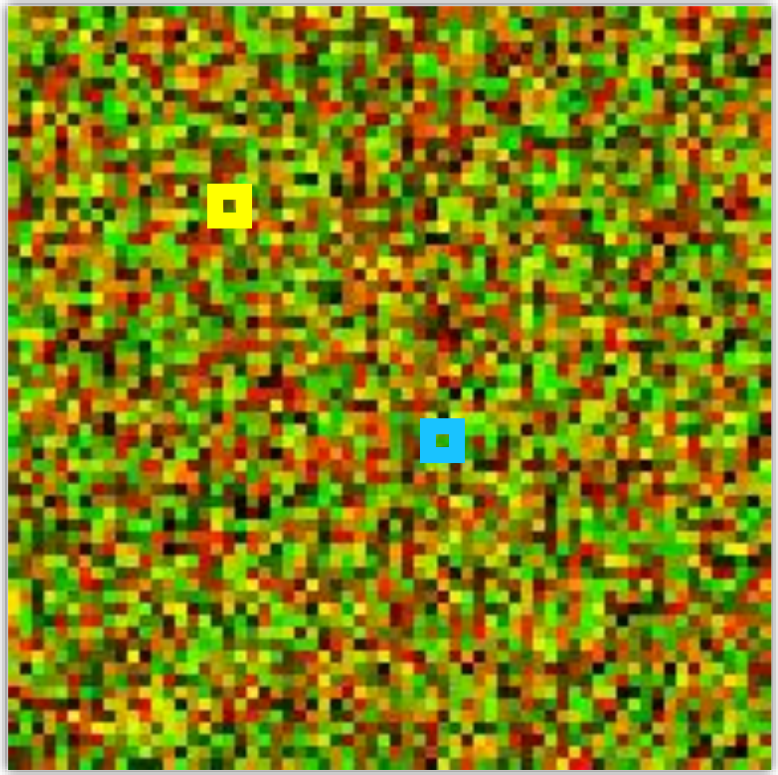
$I \leftarrow$ RandomInitialization()
for $k \in \{0, k_{max}\}$ **do**
  $p_x, p_y \leftarrow$RandomPixel()
  $e \leftarrow E(I)$
  $I_{new} \leftarrow \mathrm{swap}(I, p_x, p_y)$
  $e_{new} \leftarrow E(I_{new})$
  if $e < e_{new}$ and $exp(\frac{e - e_{new}}{c_k})$ $<$rand$(0, 1)$ **then**
    $I_{new} \leftarrow \mathrm{swap}(I_{new}, p_x, p_y)$
  end if $I \leftarrow I_{new}$
end for
Return I



Simulated annealing

Pseudo code

$I \leftarrow \mathrm{RandomInitialization}()$
for $k \in \{0, k_{max}\}$ **do**
  $p_x, p_y \leftarrow \mathrm{RandomPixel}()$
  $e \leftarrow E(I)$
  $I_{new} \leftarrow \mathrm{swap}(I, p_x, p_y)$
  $e_{new} \leftarrow E(I_{new})$
  if $e < e_{new}$ and $exp(\frac{e - e_{new}}{c_k}) < \mathrm{rand}(0,1)$ **then**
    $I_{new} \leftarrow \mathrm{swap}(I_{new}, p_x, p_y)$
  end if $I \leftarrow I_{new}$
end for
Return I



Live demo

Simulated annealing

The sum over every pixel can be simplified

- Spatial gaussian support can be restricted to few pixel
- It is possible to optimize different region of the tile at the same time
- Quality will depend on the approximation of the gaussian

Energy term

$$E(I) = \underbrace{\sum_{p \neq q}}_{\text{pixels}} E(p,q) = \sum_{p \neq q} \underbrace{e^{-\frac{\|i_p - i_q\|^2}{\sigma_i^2}}}_{\substack{\text{image-space} \\ \text{Gaussian}}} \cdot \underbrace{e^{-\frac{\|s_p - s_q\|^{d/2}}{\sigma_s^2}}}_{\substack{\text{sample-space} \\ \text{Gaussian}}}$$

Simulated annealing

Simulated annealing is an importance optimization algorithm

- Works on discreet set (Set of pixel)
- Energy function requires only a point-wise evaluation (no derivative)
- Can converge to the optimal solution under some conditions

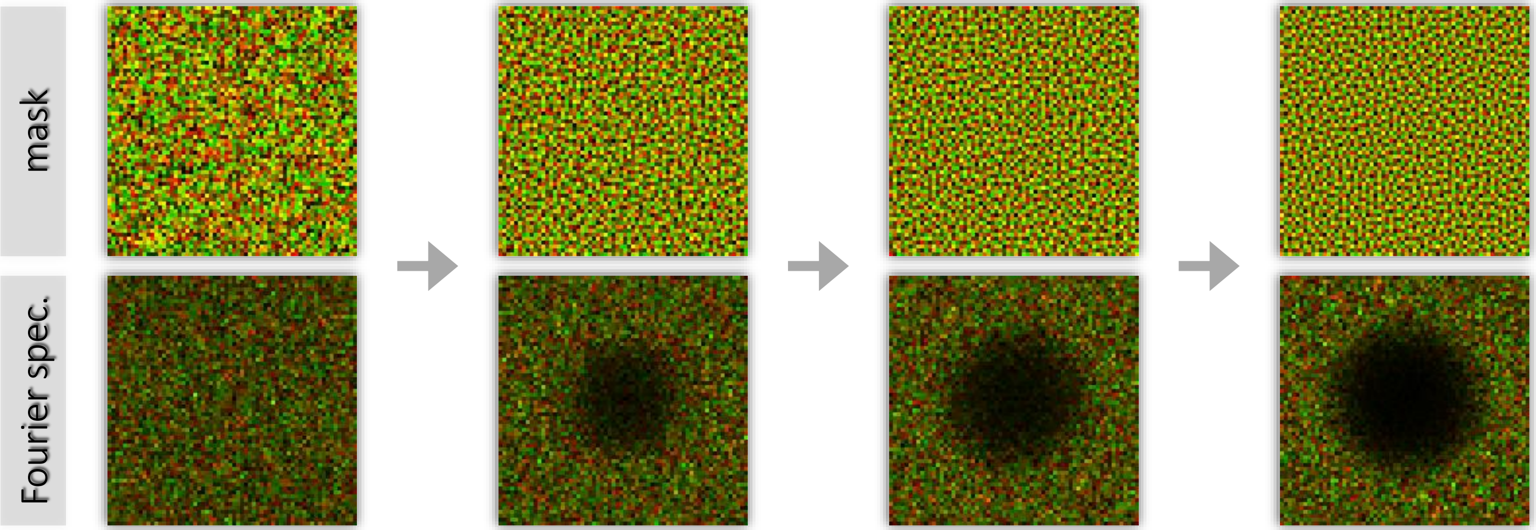
- Slow optimization in general
- Difficult to parallelize on a massive scale

Dither mask construction

1. Generate random dither mask

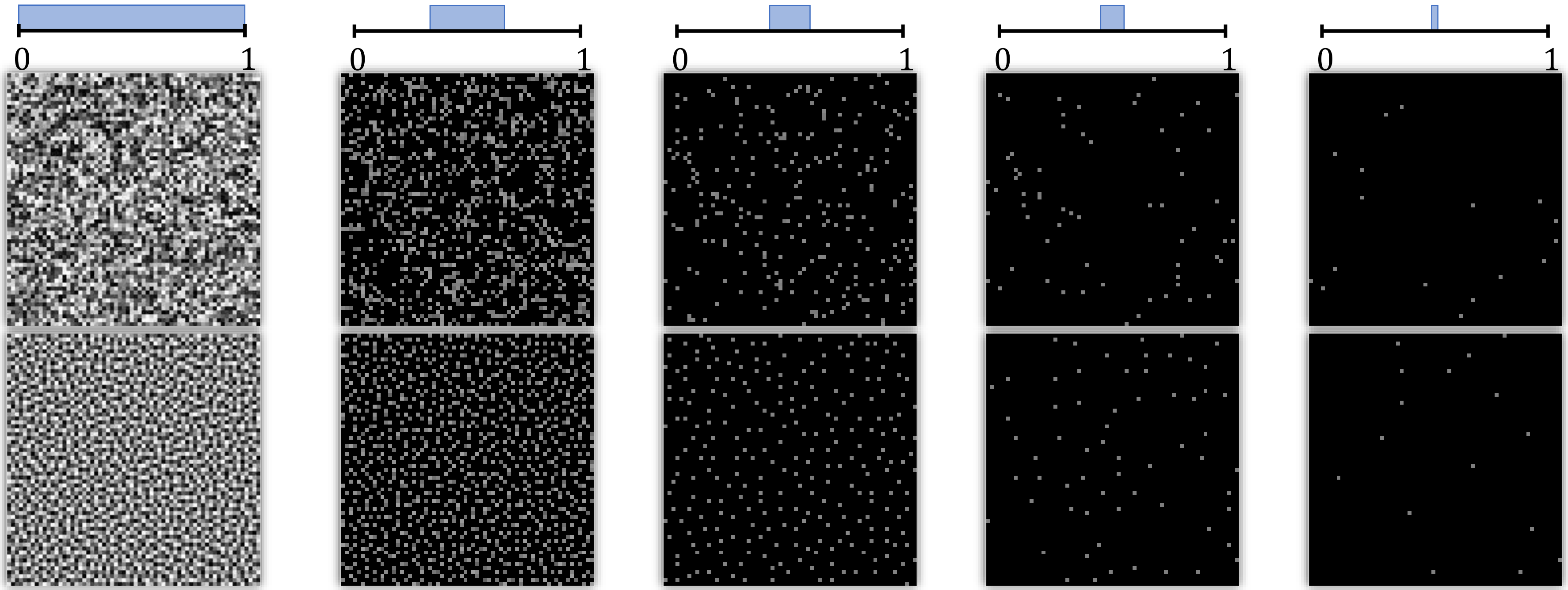
2. Simulated annealing by random pixel swapping

$$E(M) = \underbrace{\sum_{p \neq q}}_{\text{pixels}} E(p,q) = \sum_{p \neq q} \underbrace{e^{-\frac{\overbrace{\|i_p - i_q\|}^{\text{pixel distance}}{}^{2}}{\sigma_i^2}}}_{\text{image-space Gaussian}} \cdot \underbrace{e^{-\frac{\overbrace{\|s_p - s_q\|}^{\text{sample distance}}{}^{d/2}}{\sigma_s^2}}}_{\text{sample-space Gaussian}}$$

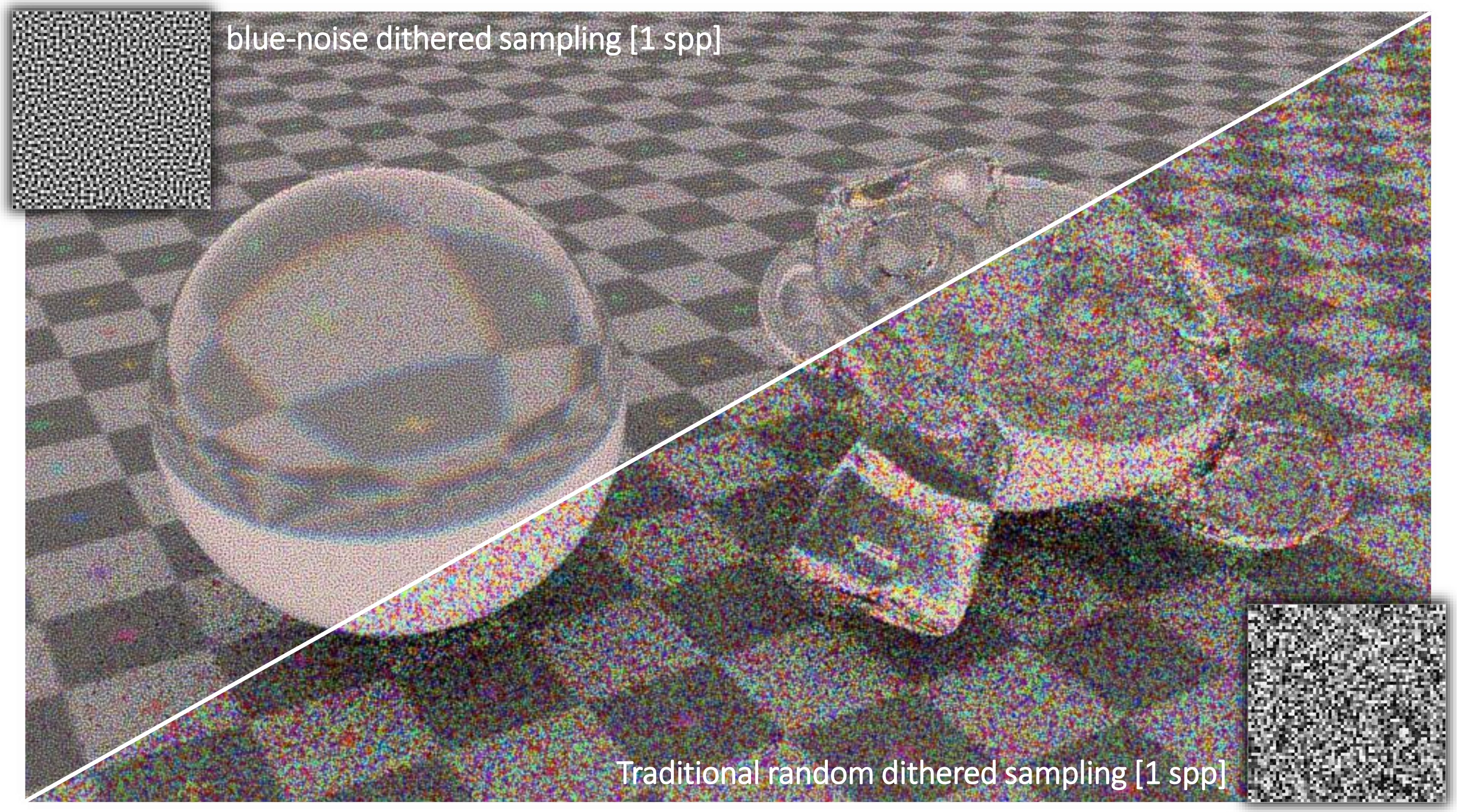


Sampling with dither masks

1. Create sampling pattern
2. For each pixel
   a. Look-up value from dither mask (tiled over the image)
   b. Use value to offset sample pattern (surface, direct sampling, …)

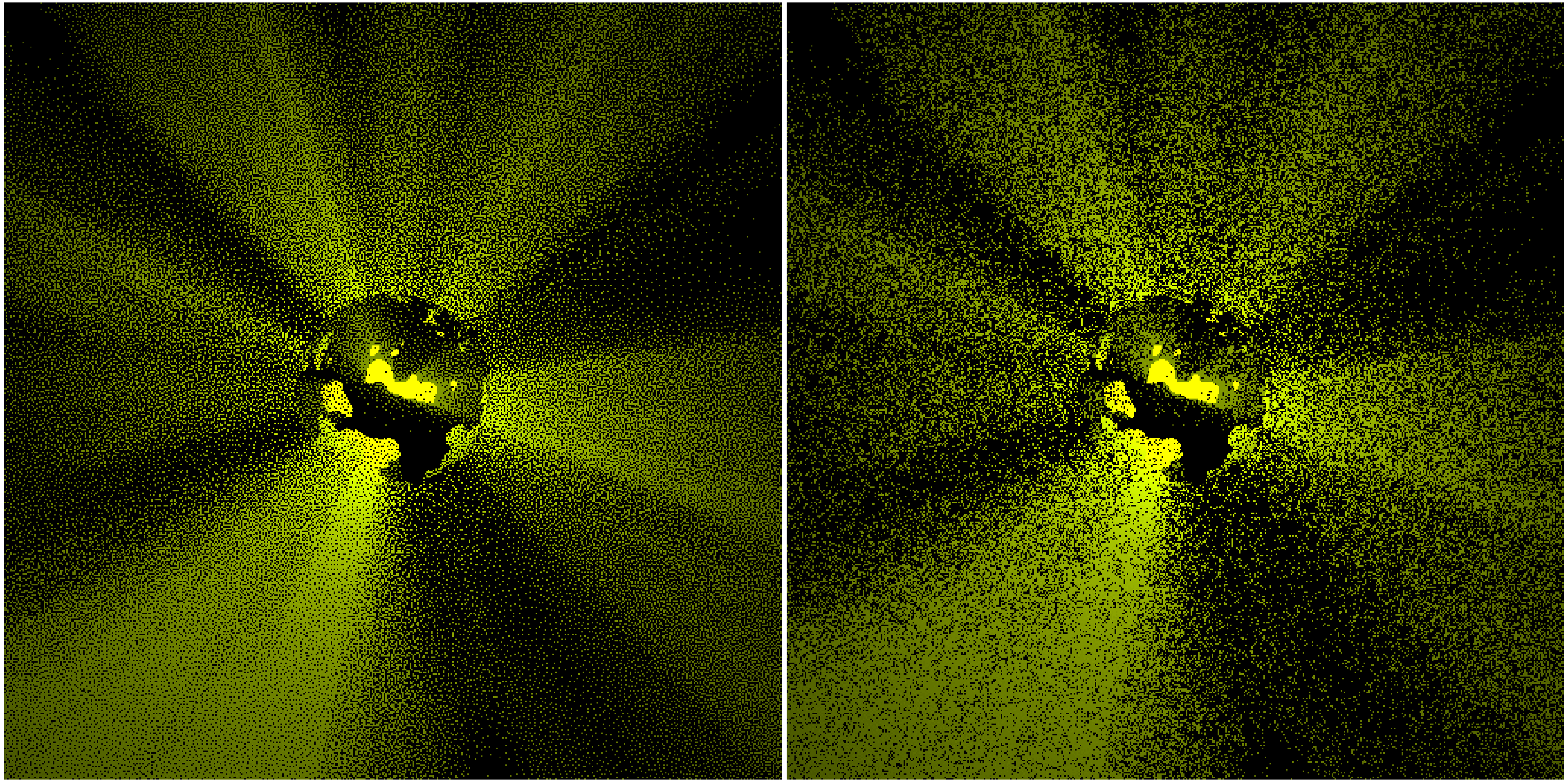


Sampling with dither masks



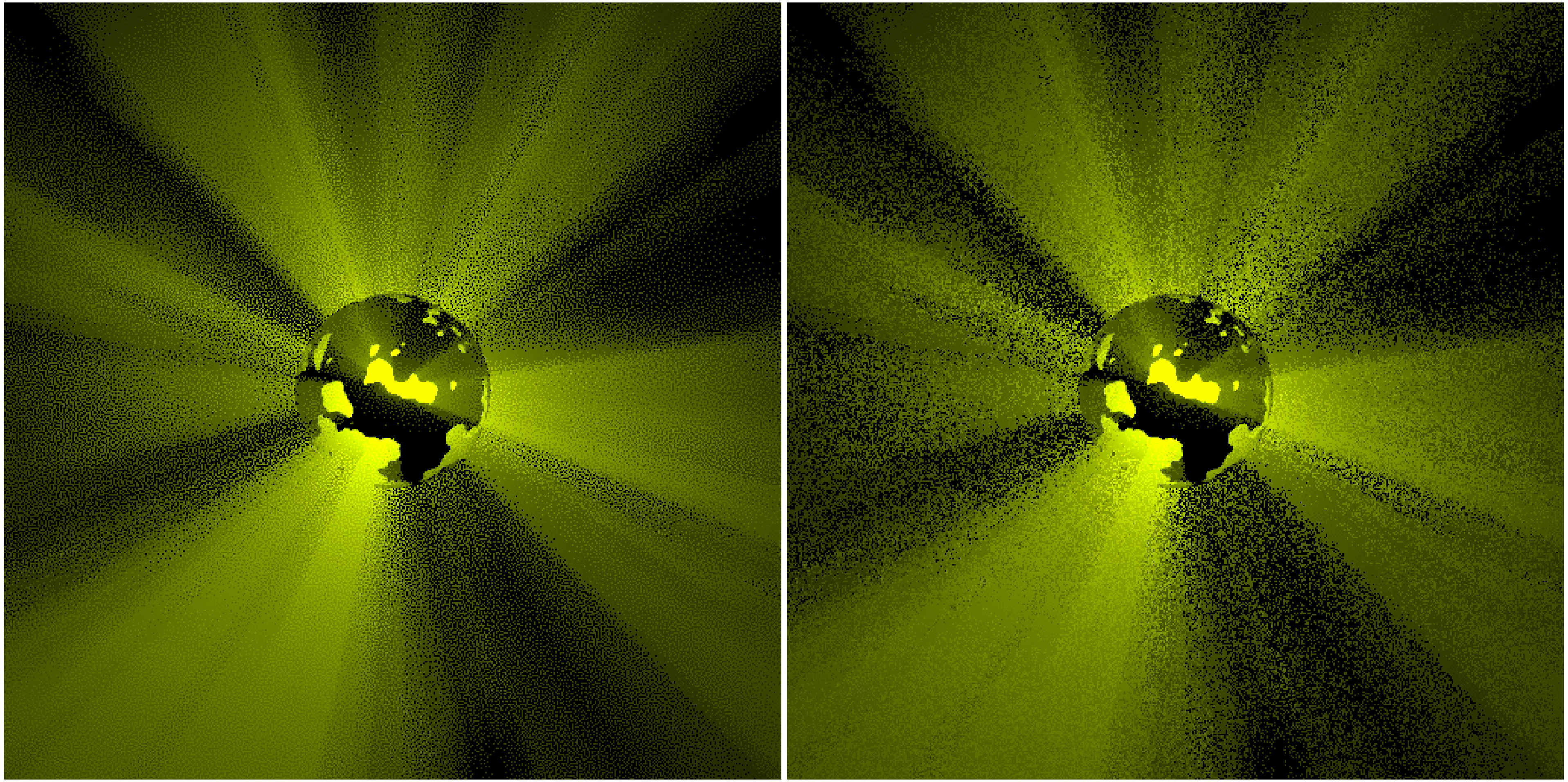
Sampling with dither masks

1 sample per pixel



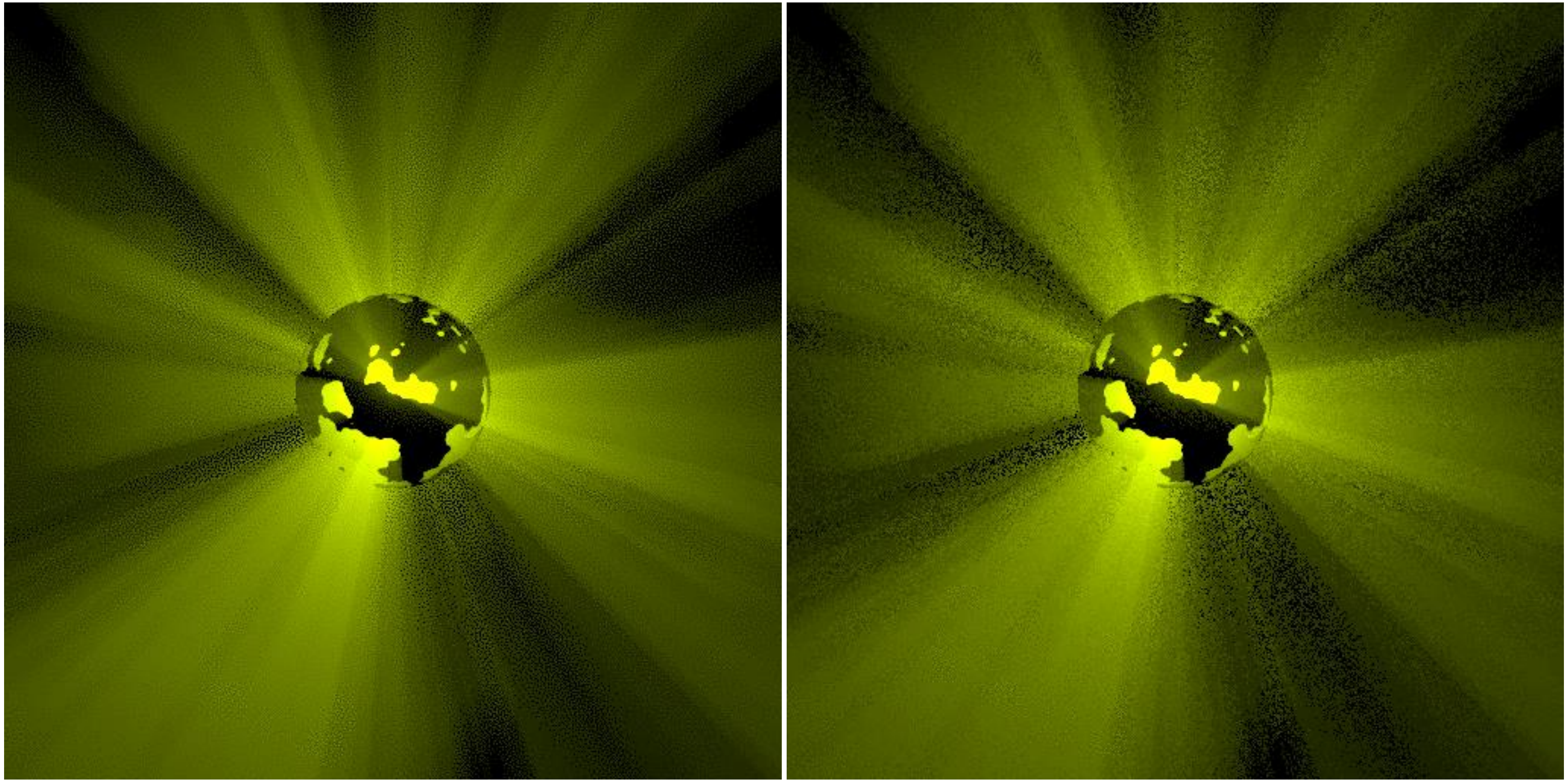
Sampling with dither masks

4 sample per pixel



Sampling with dither masks

9 sample per pixel



Sampling with dither masks

- Connection between halftoning and MC sampling
- Blue correlation > white decorrelation
- Simple, fast method
  - Showed 1D and 2D sampling
- Limitations
  - Improvement only when pixel integrals are correlated
  - Occasional mask tiling artifacts
  - Higher dimensions more difficult
  - Remapping of the samples limit quality

A LOW-DISCREPANCY SAMPLER THAT DISTRIBUTES MONTE CARLO ERRORS AS A BLUE NOISE IN SCREEN SPACE

[Heitz & al 2019]

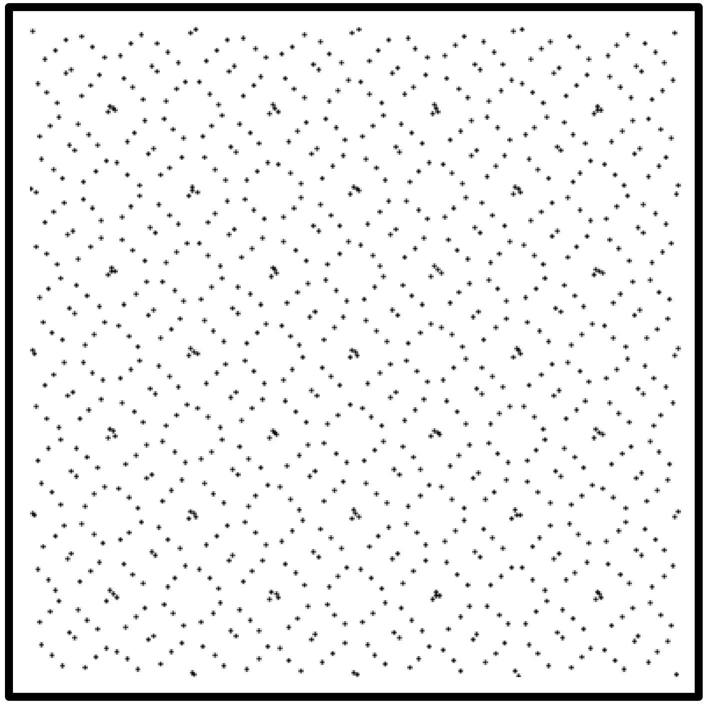
Objectives

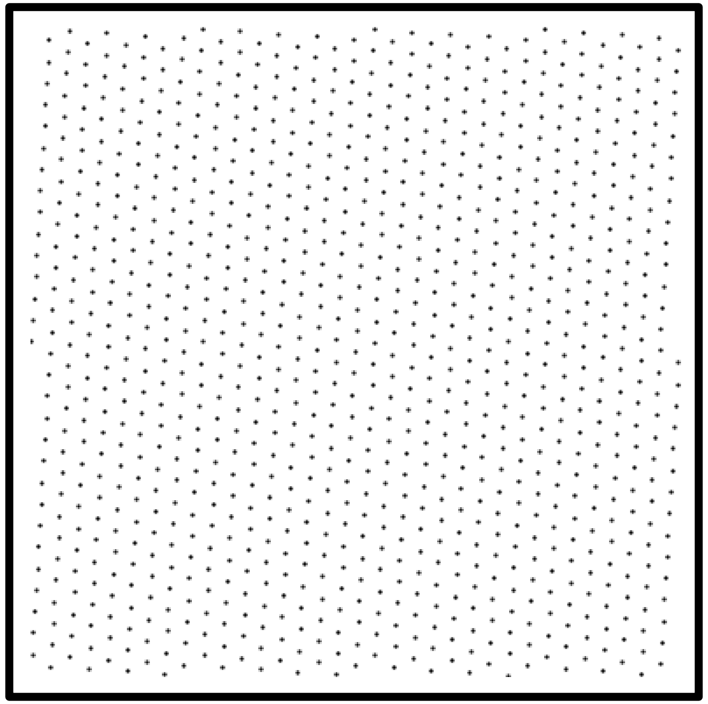
- BNDS limitations
  - Tradeoff between per-pixel sampling quality and error distribution
  - Can be worst than uncorrelated
  - Rely on correlation between sample difference and rendering difference

- Improvements
  - Directly optimize for rendering purpose
  - Ensure worse case falls back to uncorrelated

Sample stratification

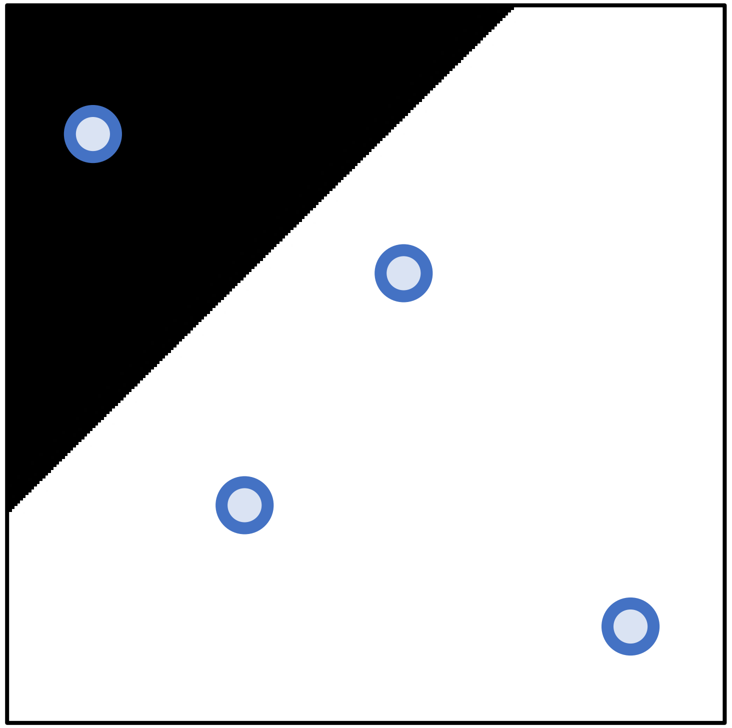
- Use Owen's Scrambling or Rank1 lattice
  - Use a Scrambling Key instead of a shift vector
  - Ensure good integration at each pixel

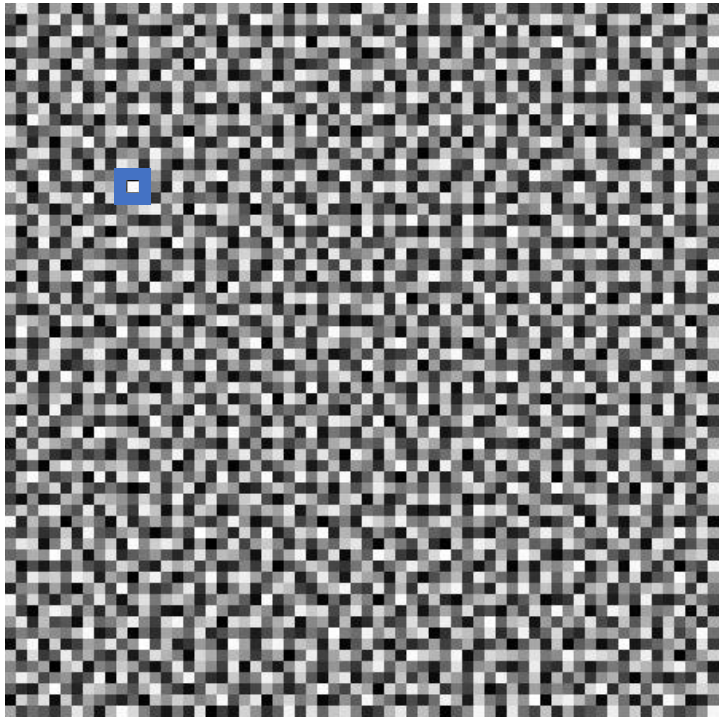




Change the optimization space

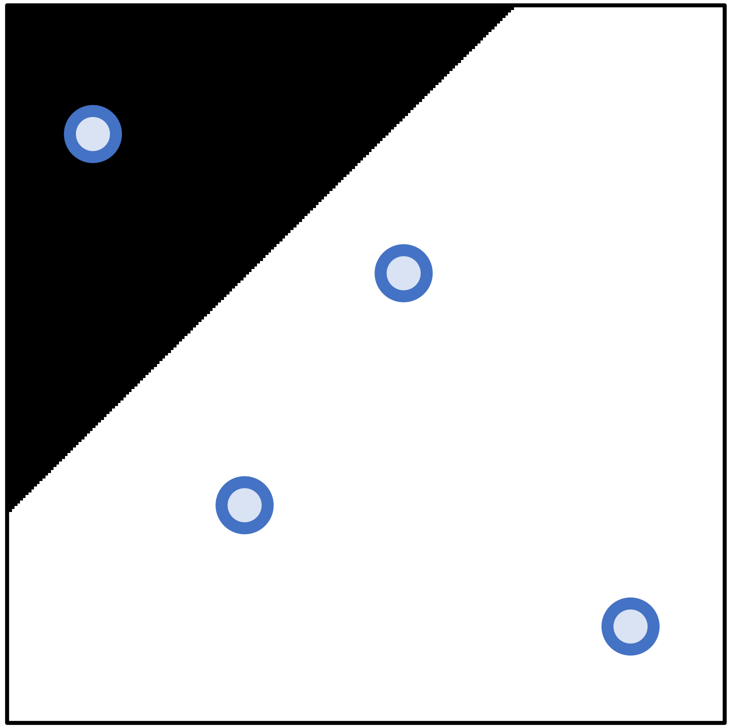
- Change from **sample space** optimization to **result space optimization**
  - Assume every pixel render the same integrand
  - But all pixel use a different XOR keys or seed

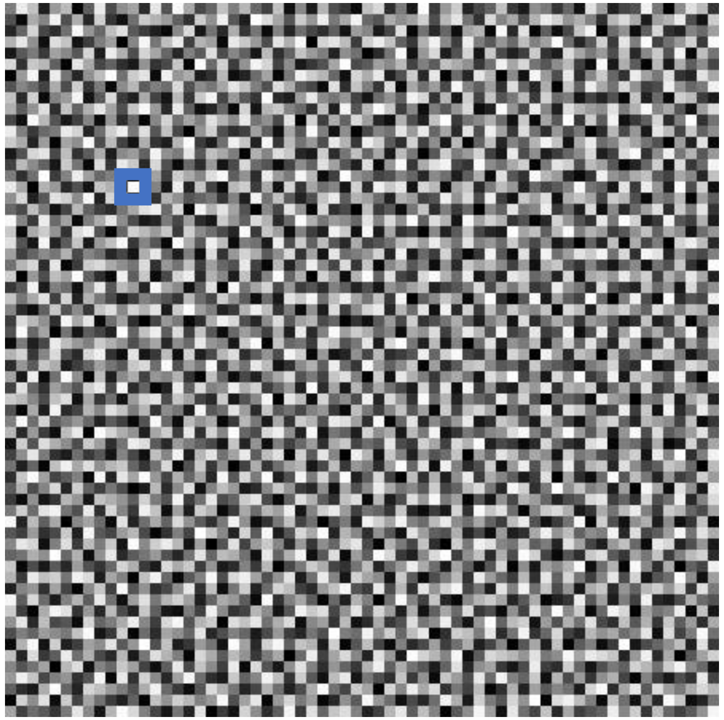




Change the optimization space

- Change from **sample space** optimization to **result space optimization**
  - Assume every pixel render the same integrand
  - But all pixel use a different XOR keys or seed





Change the optimization space

- Change from **sample space** optimization to **result space optimization**
  - Assume every pixel render the same integrand
  - But all pixel use a different sample set

$$E(M) = \underbrace{\sum_{p \neq q}}_{\text{pixels}} E(p,q) = \sum_{p \neq q} \underbrace{e^{-\frac{\overbrace{\|i_p - i_q\|^2}^{\text{pixel distance}}}{\sigma_i^2}}}_{\text{image-space Gaussian}} \cdot \underbrace{e^{-\frac{\overbrace{\|s_p - s_q\|^{d/2}}^{\text{sample distance}}}{\sigma_s^2}}}_{\text{sample-space Gaussian}}$$

Change the optimization space

- Change from **sample space** optimization to **result space optimization**
  - Assume every pixel render the same integrand
  - But all pixel use a different sample set

$$E(M) = \underbrace{\sum_{p \neq q}}_{\text{pixels}} E(p,q) = \sum_{p \neq q} \underbrace{e^{-\frac{\overbrace{\|i_p - i_q\|^2}^{\text{pixel distance}}}{\sigma_i^2}}}_{\text{image-space Gaussian}} \cdot \underbrace{\|\varepsilon_p - \varepsilon_q\|^2}_{\text{Rendering error}}$$

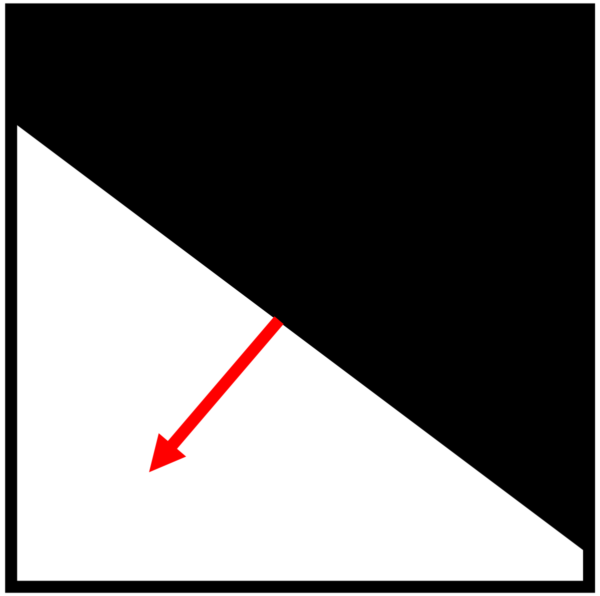
Optimization setup

- Optimize the distribution of XOR key
  - Define a class of integrands
  - **Oriented Heavisides** define by **θ** and **d**
  - **D** dimensional integrands

$$E(M) = \underbrace{\sum_{p \neq q}}_{\text{pixels}} E(p,q) = \sum_{p \neq q} \underbrace{e^{-\frac{\overbrace{\|i_p - i_q\|^2}^{\text{pixel distance}}}{\sigma_i^2}}}_{\text{image-space Gaussian}} \cdot \underbrace{\|\varepsilon_p - \varepsilon_q\|^2}_{\text{Rendering error}}$$

ε_p is a vector of **signed** integration error $(f(x_p) - F)$ on a large set of integrand (typically 65 536 randomized integrands)

The optimization is also done using simulated annealing



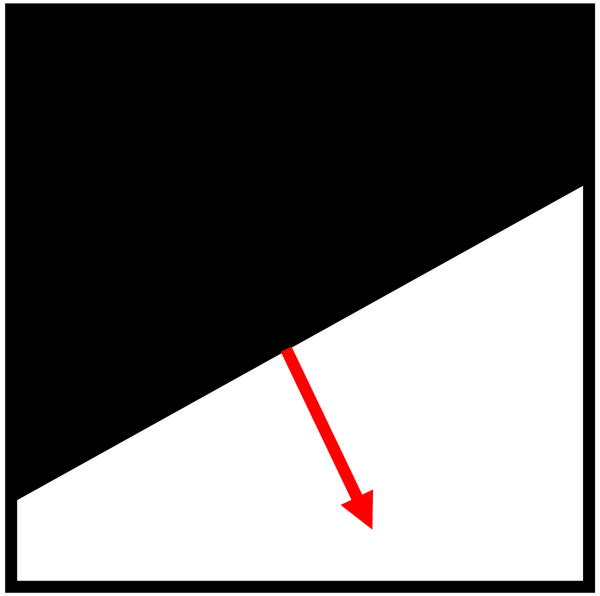
Optimization setup

- Optimize the distribution of XOR key
  - Define a class of integrands
  - **Oriented Heavisides** define by **θ** and **d**
  - **D** dimensional integrands

$$E(M) = \underbrace{\sum_{p \neq q}}_{\text{pixels}} E(p,q) = \sum_{p \neq q} \underbrace{e^{-\frac{\overbrace{\|i_p - i_q\|^2}^{\text{pixel distance}}}{\sigma_i^2}}}_{\text{image-space Gaussian}} \cdot \underbrace{\|\varepsilon_p - \varepsilon_q\|^2}_{\text{Rendering error}}$$

ε_p is a vector of **signed** integration error $(f(x_p) - F)$ on a large set of integrand (typically 65 536 randomized integrands)

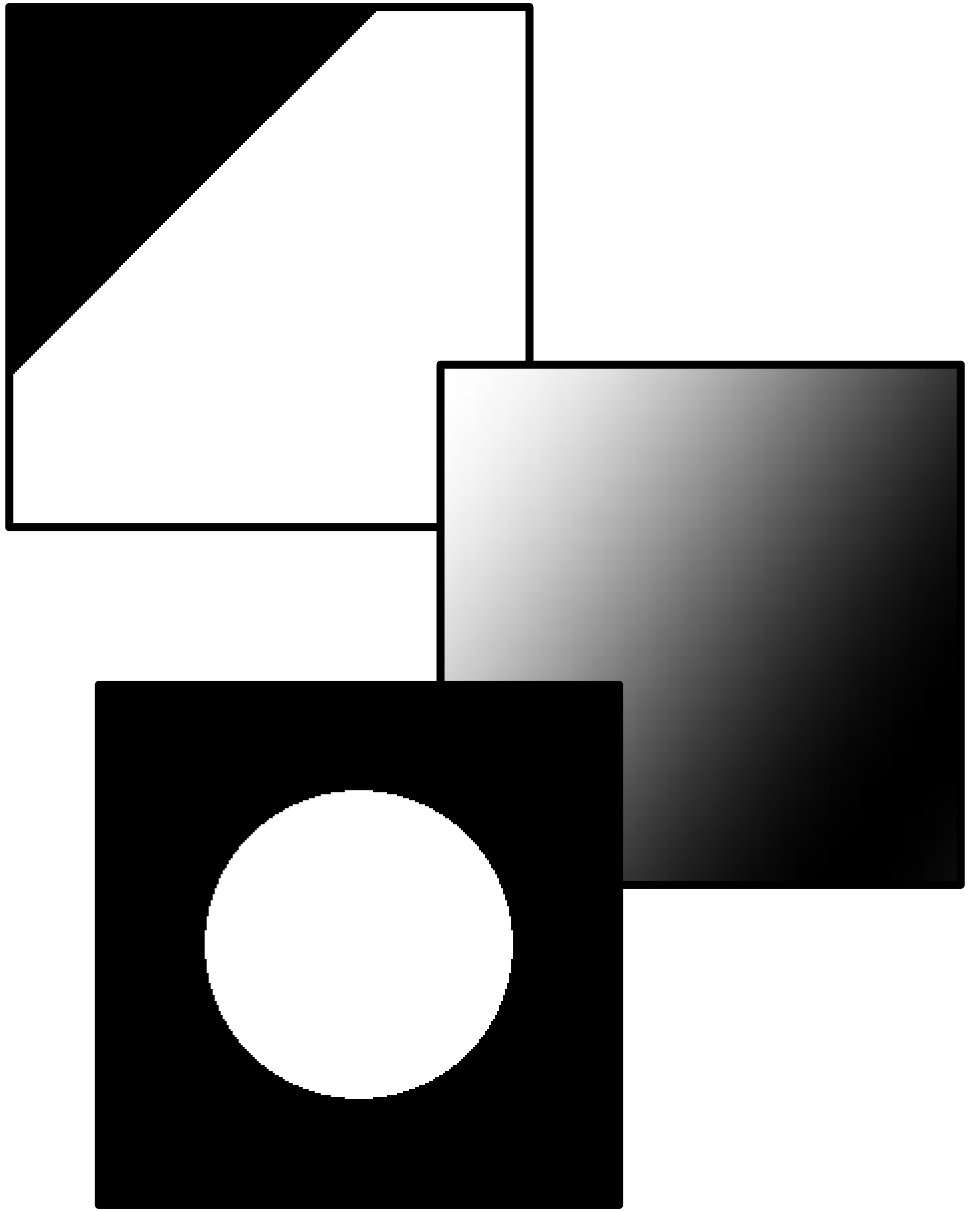
The optimization is also done using simulated annealing



Optimization setup

- Optimize the distribution of XOR key
  - Define a class of integrands
  - **Oriented Heavisides** define by **θ** and **d**
  - **D** dimensional integrands

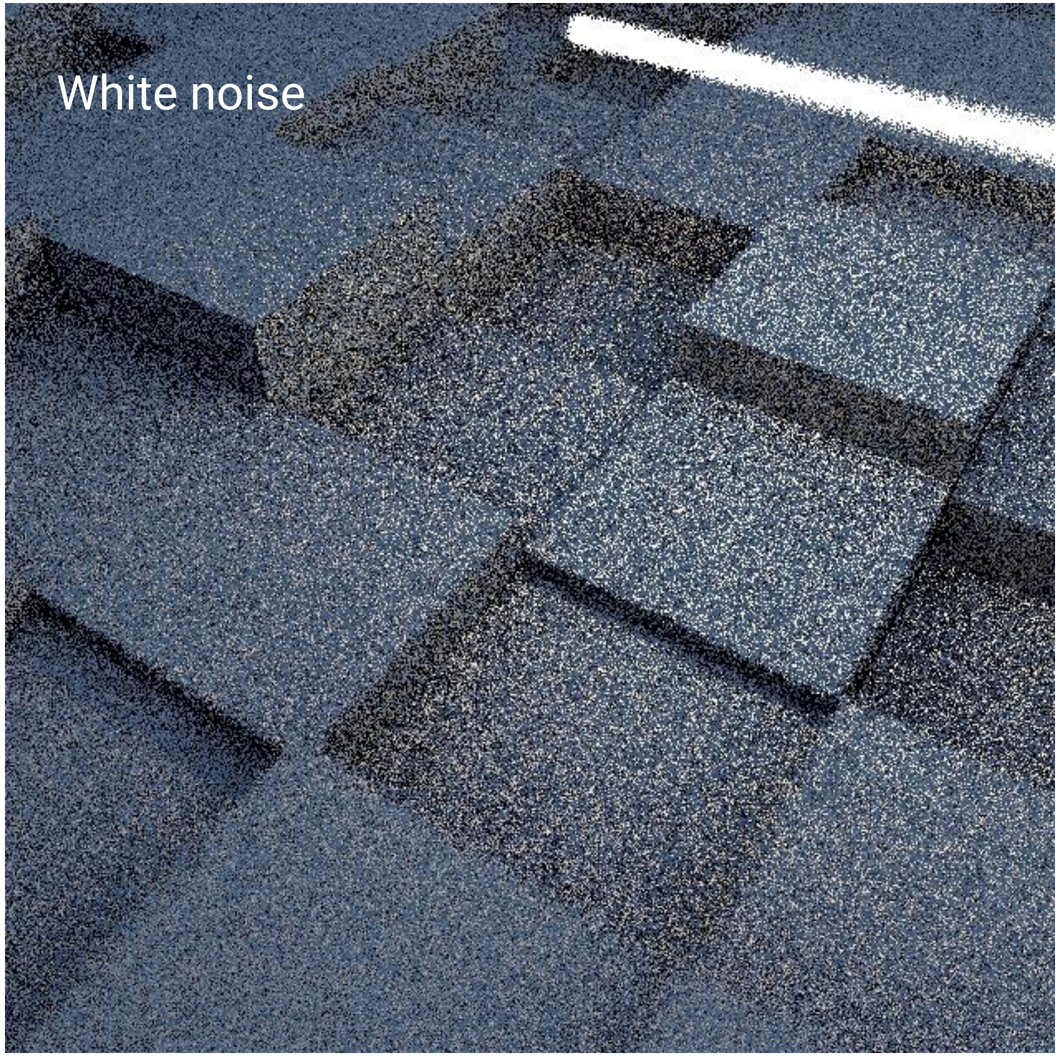
- Robust to other class of integrand
  - Varying orientation
  - Varying smoothness
  - Varying shape

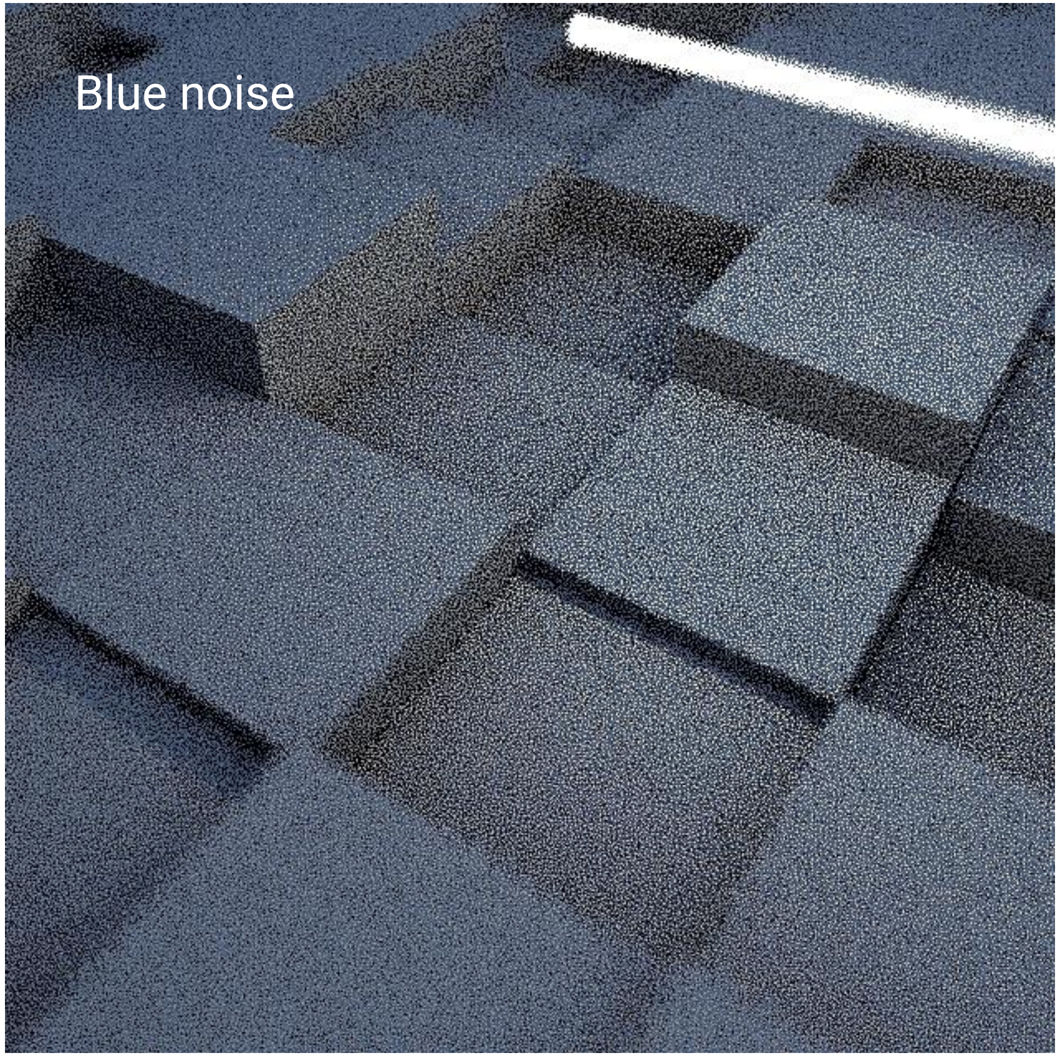


Optimization setup

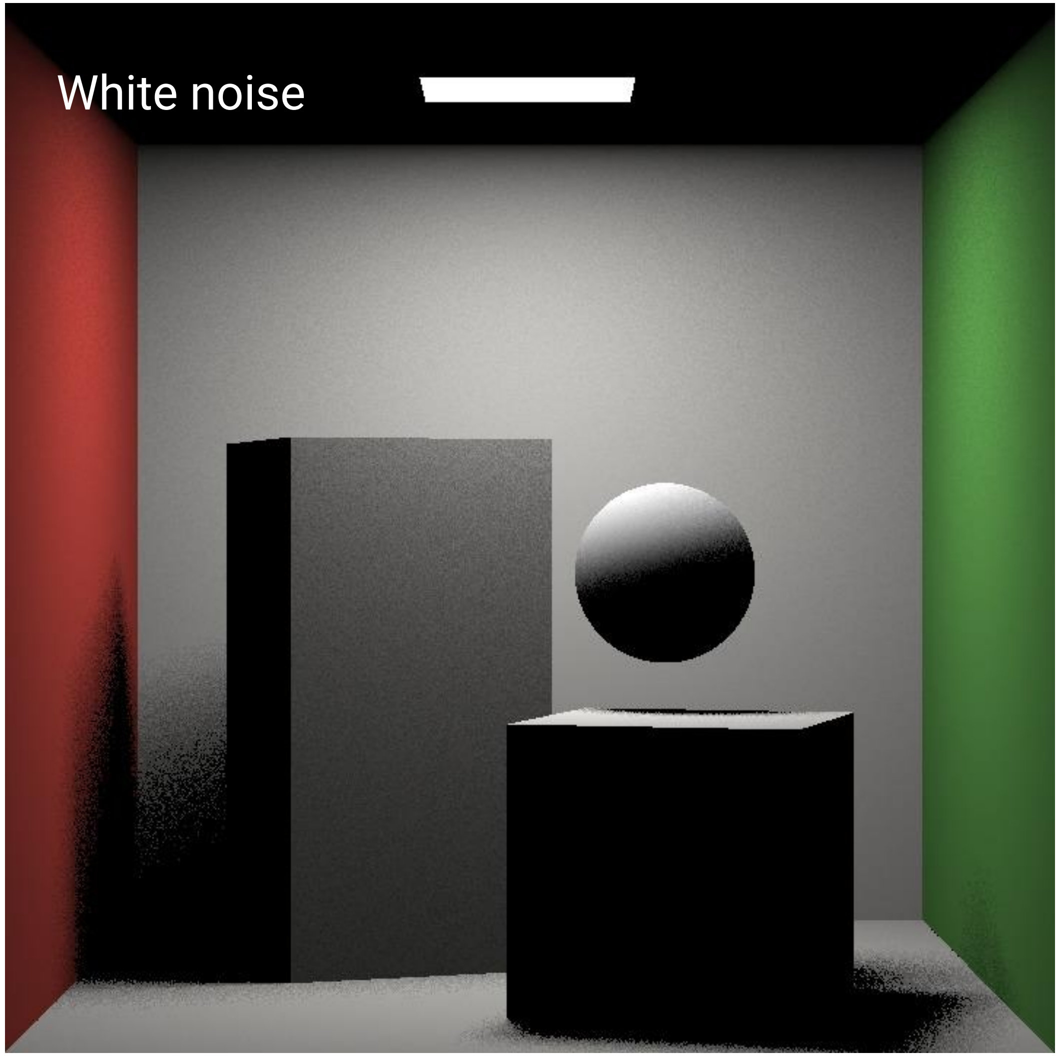
- Rely on simple function
  - Step function and linear function show properties from rendering (visibility)
  - Act as a form of discrepancy measure
  - Quality of the result depend on the choice of the function class
- Error vector can be simplified
  - Store the distance norm between to sample set (1 value per pair)
- Works with progressive sampling
  - It is possible to optimize the progressive sample set distribution
  - Still limited by the dimensionality

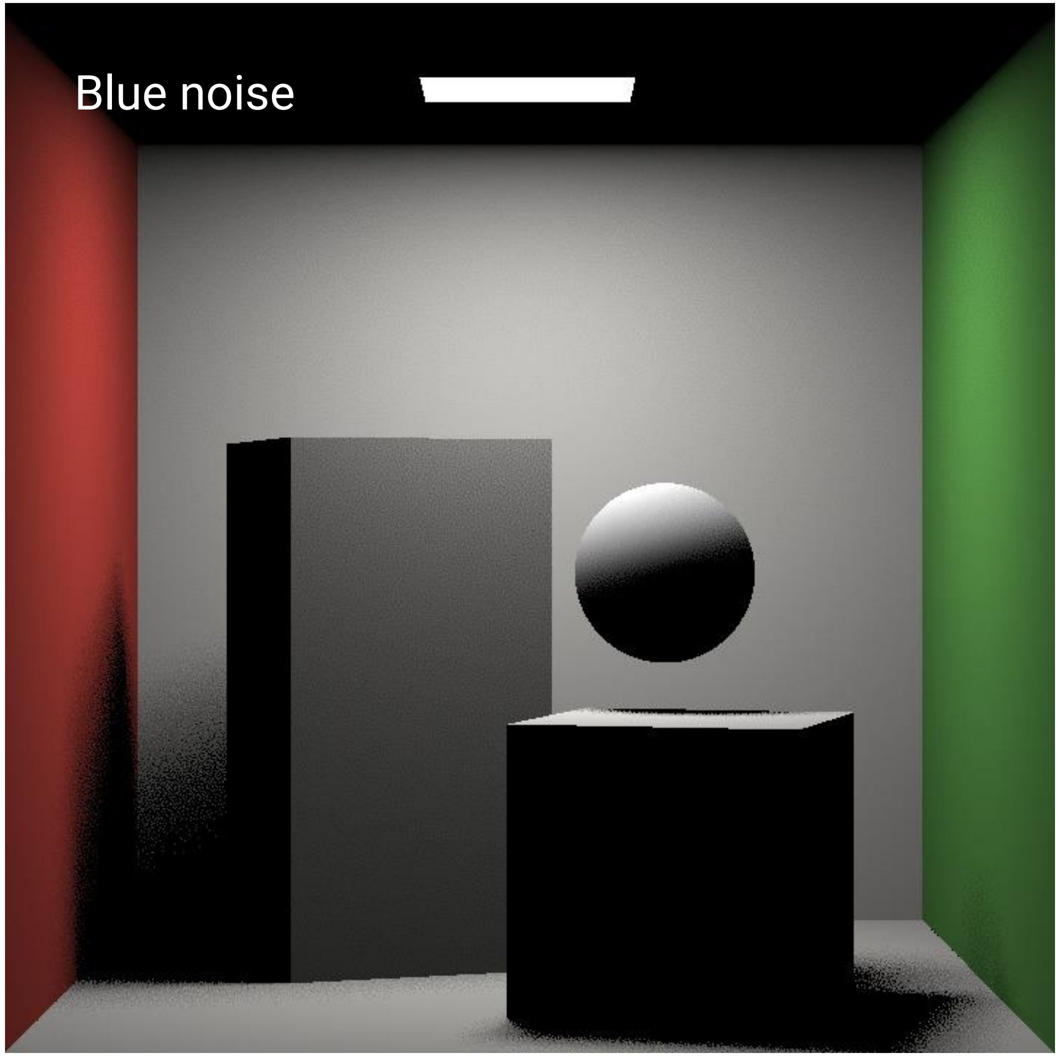
Results



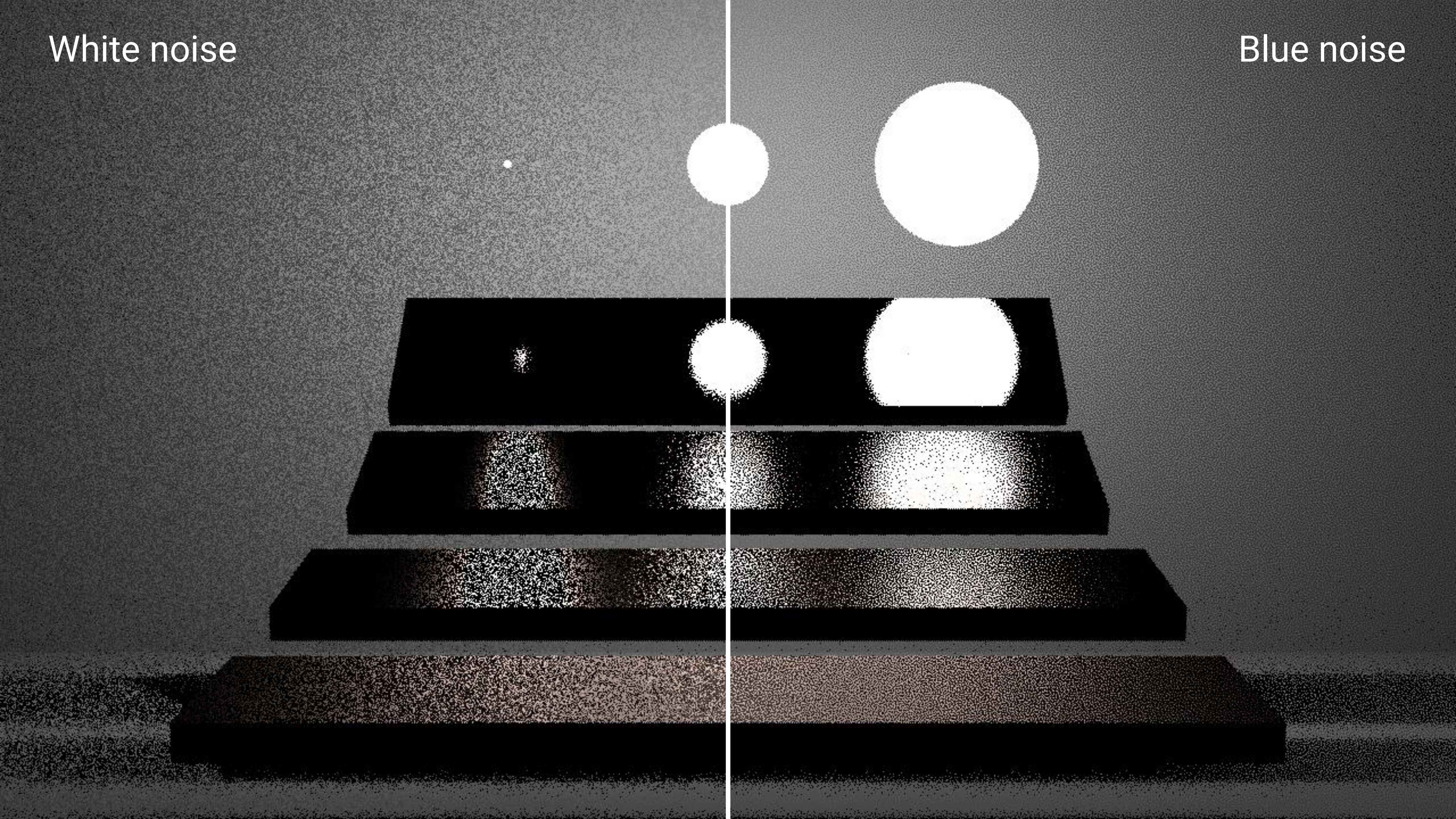


Results






White noise
Blue noise


White noise
Blue noise

Results

- **Share** limitations with BNDS !
  - Can't handle **high-dimensional** integrands
  - Not robust on **complex integrands**

- Still **you should use this method** rather than BNDS
  - Can **only do better**, not worse

Distributing Monte Carlo Errors as a Blue Noise in Screen Space by Permuting Pixel Seeds Between Frames

[Heitz & al 2019]

Objectives

- Previous works precompute a shift map or sample set distribution
  - Not adaptive to the rendering
  - Need one optimization per sample count/dimension

- Scene adaptive sample optimization at runtime
  - Use existing sampler
  - Can achieve higher quality
  - Work at arbitrary sample count/dimensionality
  - No precomputation

Redefining the rendering process

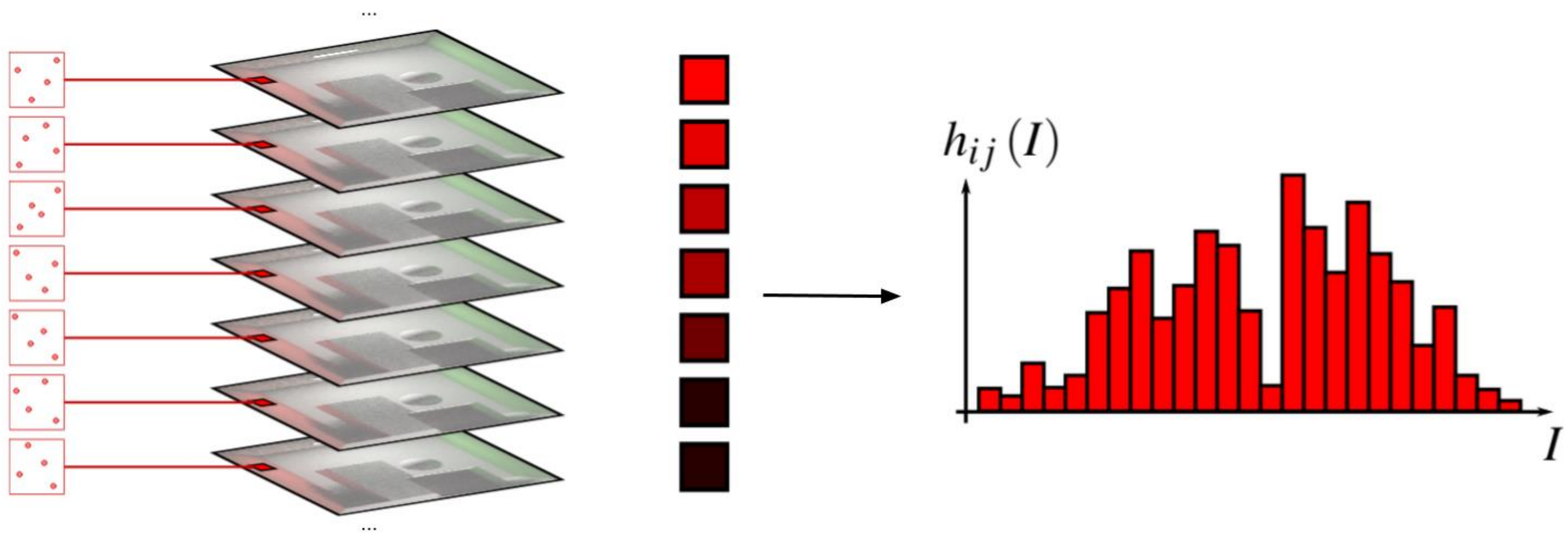
- Integration space is often high dimensional an not smooth
  - Modify the integration process to reduce dimensionality
  - Operate on a smooth space

- Change from Sample space to estimate space
  - Construct the histogram of estimate and sample it
  - The histogram is a discreet representation of the PDF of estimates
  - Histogram is a 1D space independently to the sampling space

$$\int_{\Omega} f(x)dx = \int_0^1 H^{-1}(u)du$$

Histogram of estimate

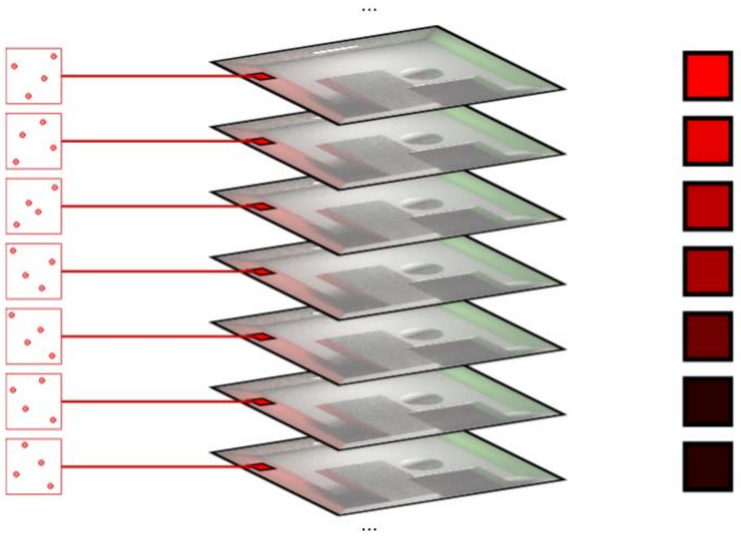
- How to construct the histogram of estimates ?
  - For each pixel, render the image with a large number of independent rendering
  - Rendering can be done with multiple samples
  - It's possible to use Low discrepancy sampler
  - Each rendering can be determined by the seed of the random generator



Histogram of estimate

- Histogram naturally create a smooth space
  - Sample set are ordered by rendering value (monotone function)
  - Estimate the continuous estimate distribution

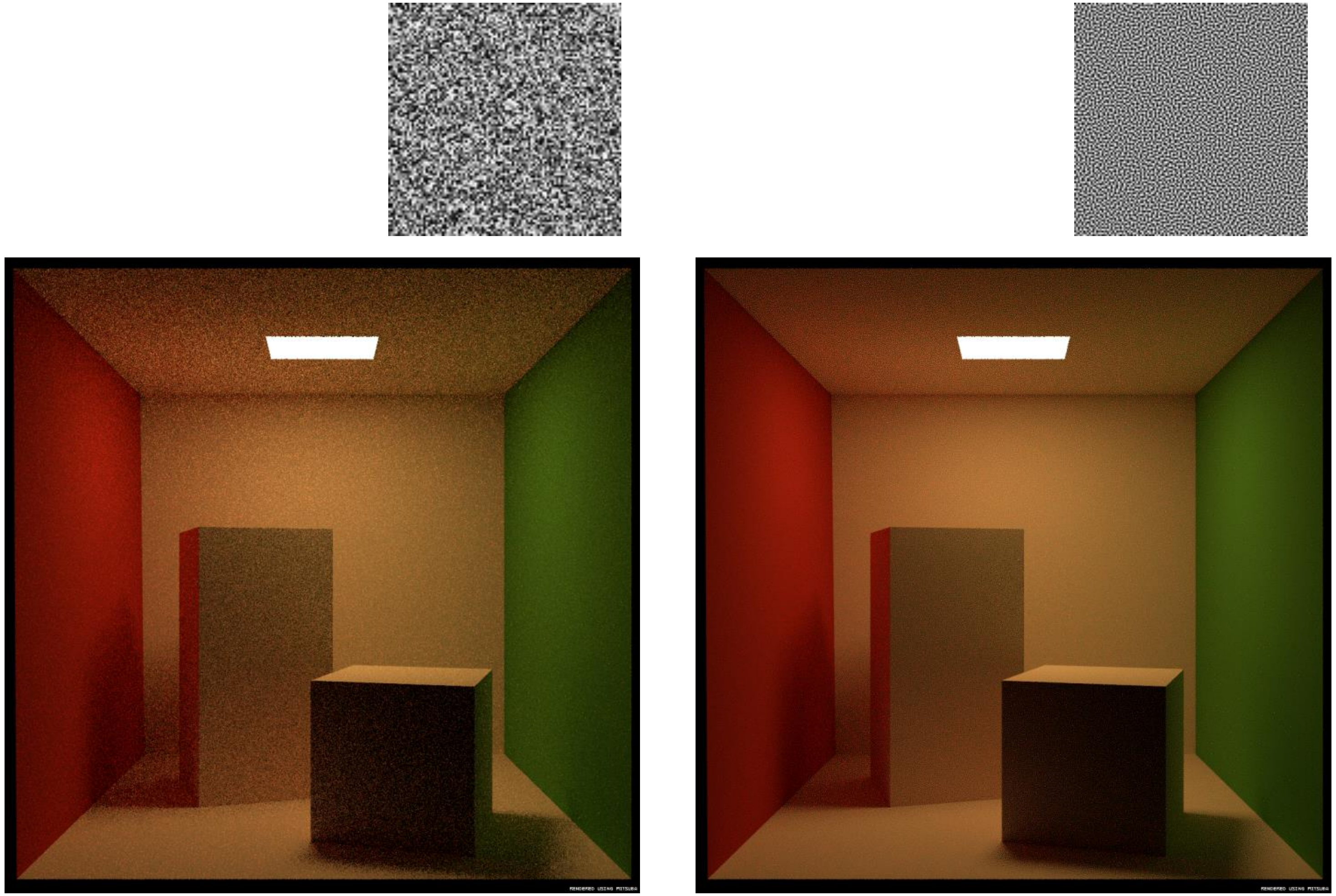
- Simple to construct
  - Rendering with varying sample set
  - Sorting operation based on luminance
  - Once sorted just need to store the seed



Histogram sampling

- Histogram are staircase function
  - It is easily invertible

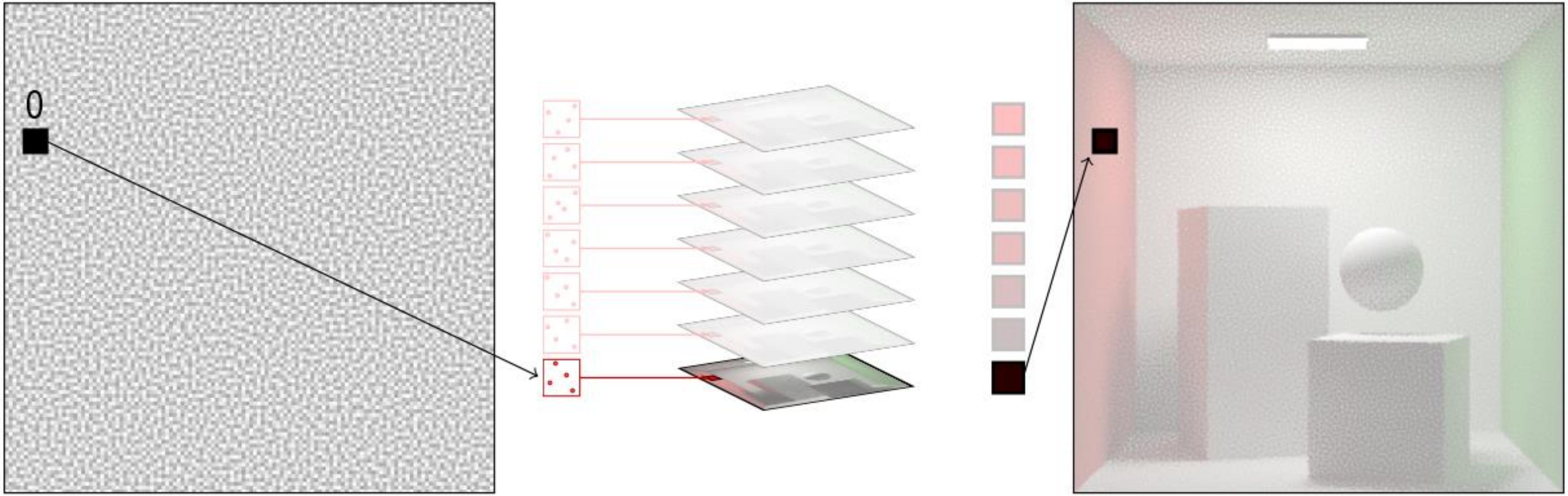
$\int_0^1 H^{-1}(u)du \approx H^{-1}(\xi)$ with $\xi \sim U(0,1)$



ξ can be generated with :
- Random uncorrelated generator
- A dither mask

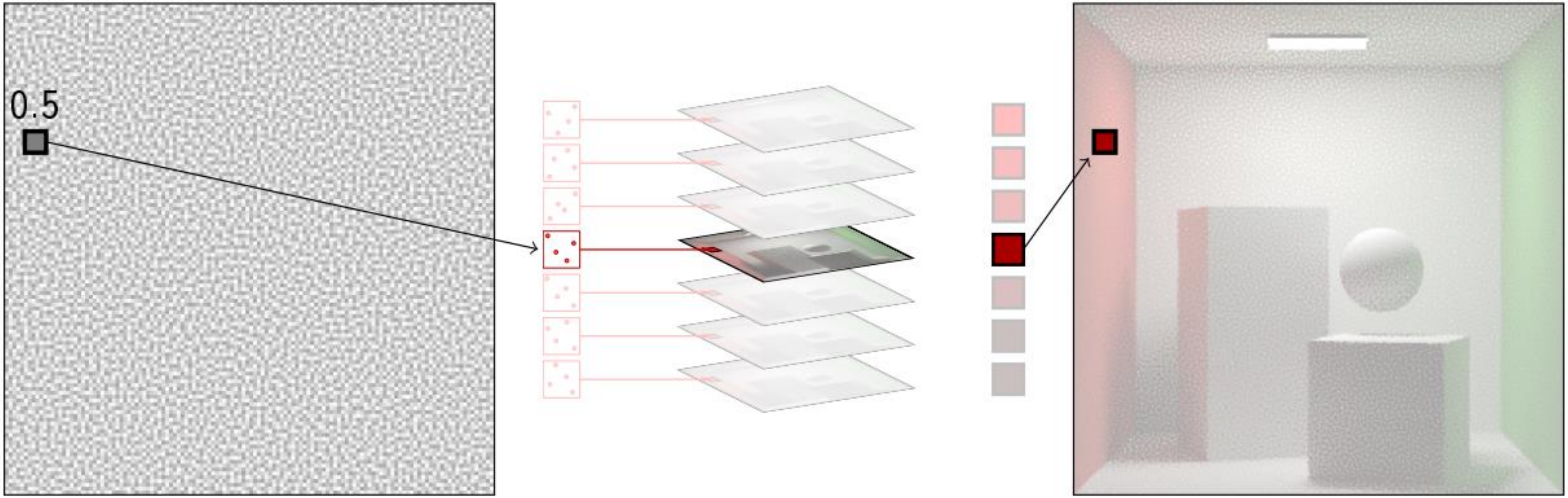
Histogram sampling

- Use the dithering mask value to select the corresponding seed
  - List of estimate values sorted based on luminance



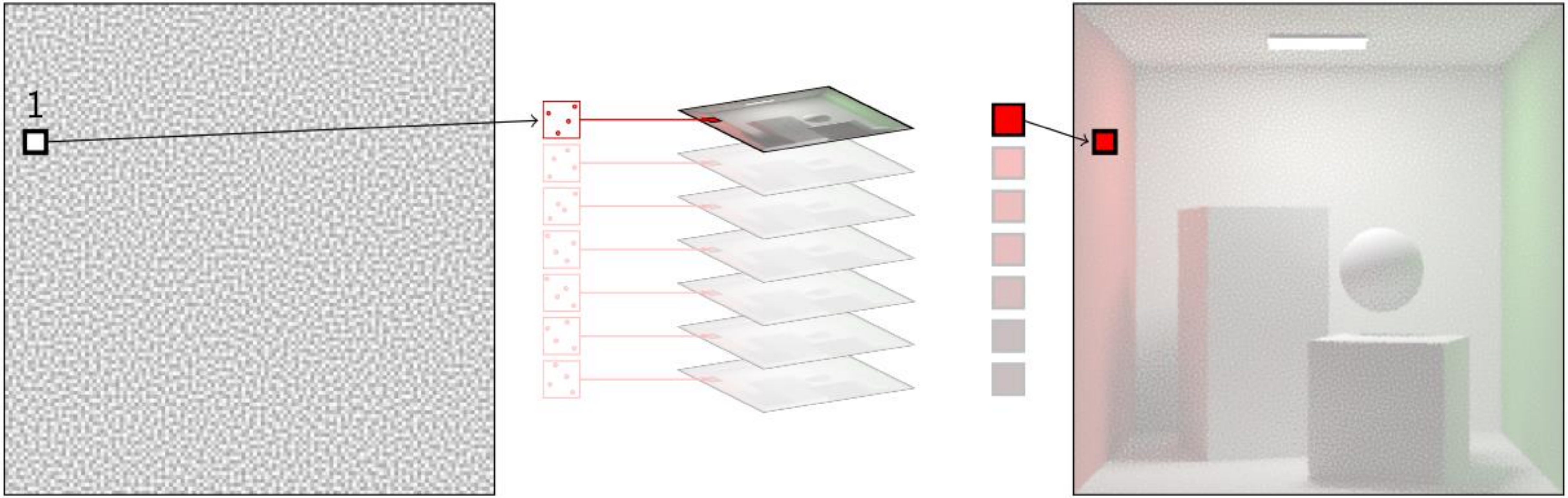
Histogram sampling

- Use the dithering mask value to select the corresponding seed
  - List of estimate values sorted based on luminance



Histogram sampling

- Use the dithering mask value to select the corresponding seed
  - List of estimate values sorted based on luminance

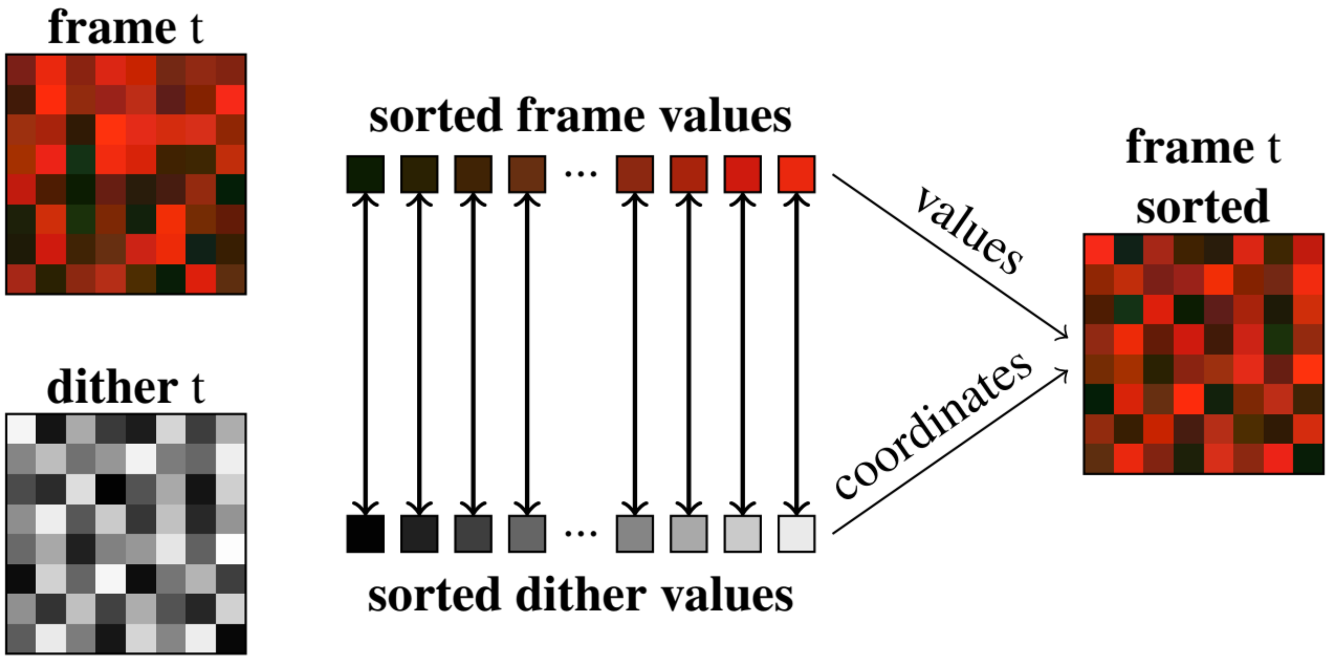


Histogram sampling

- In practice computing the histogram mean multiple rendering
  - Simply average them
- Choosing 1 sample over 4 smartly can produce lower visual error
  - Trade off pixel quality and noise distribution
- Work in high dimensional cases
- Works for all multiple samples per pixel
- Can use Low discrepancy sampler

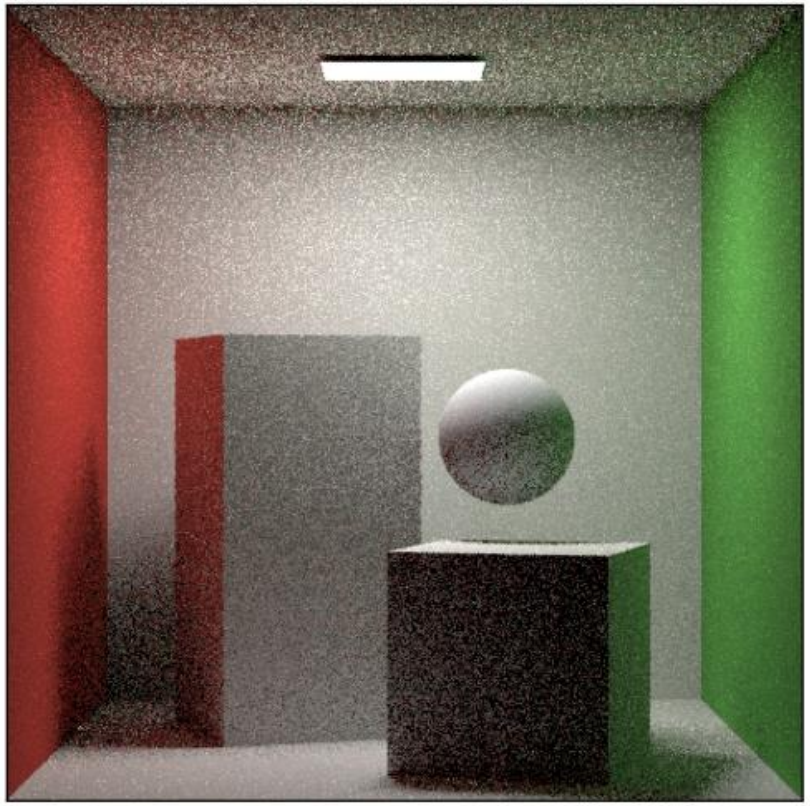
Practical algorithm

- Construct the histogram locally
  - Split the image space in small blocs
  - Use optimal transport to map frame values and dither mask
- Can be combined in a temporal algorithm to avoid re-rendering
- Use the powerful properties of sorting
  - Sorting naturally create a correlated and smooth space



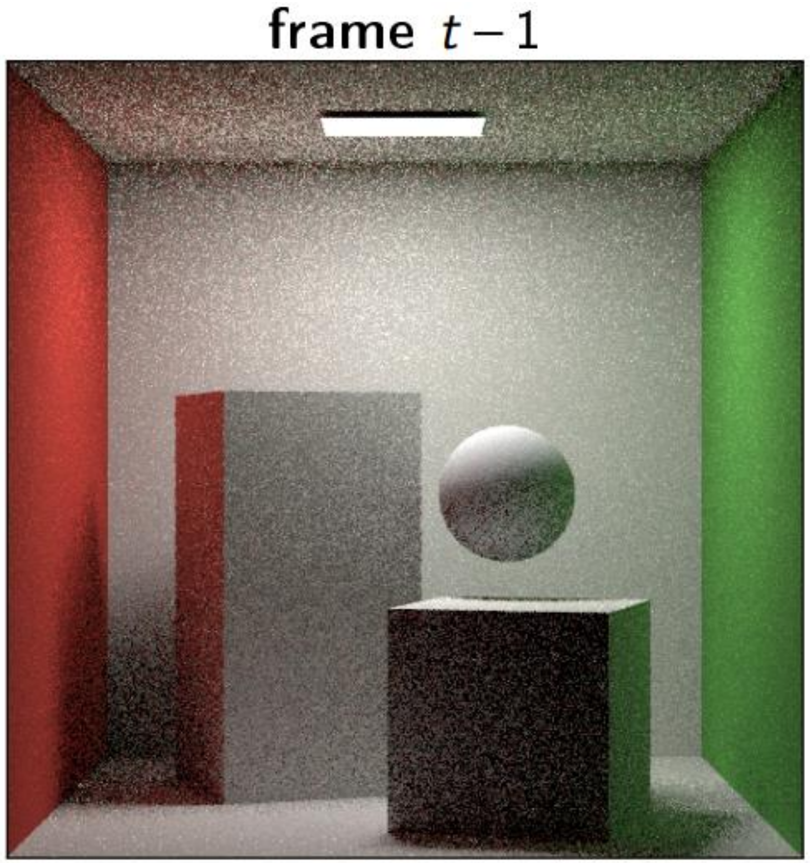
Temporal algorithm

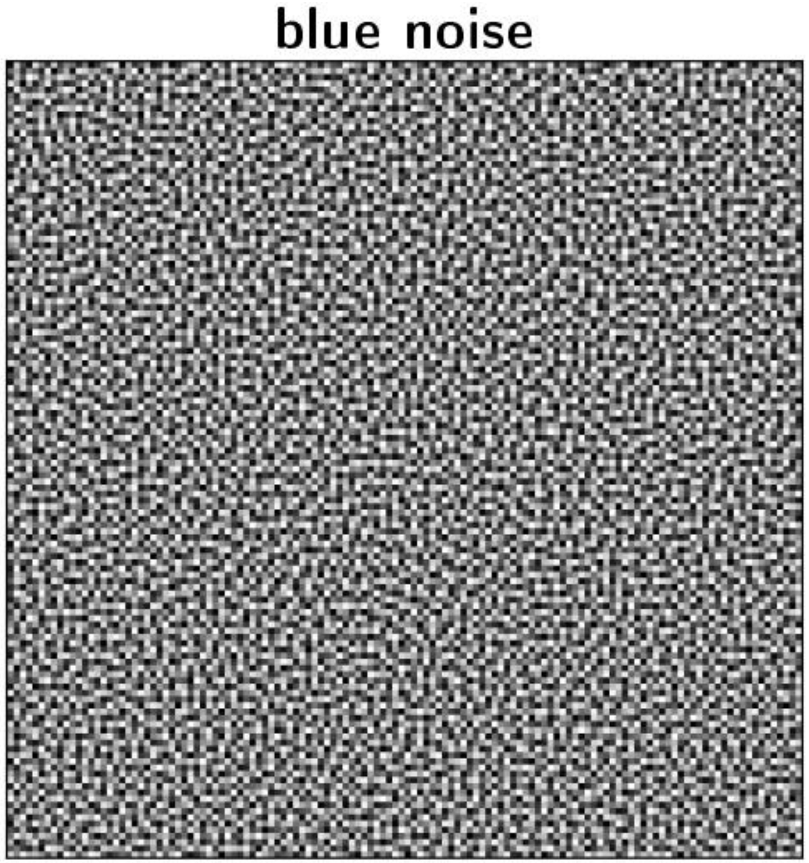
frame $t-1$



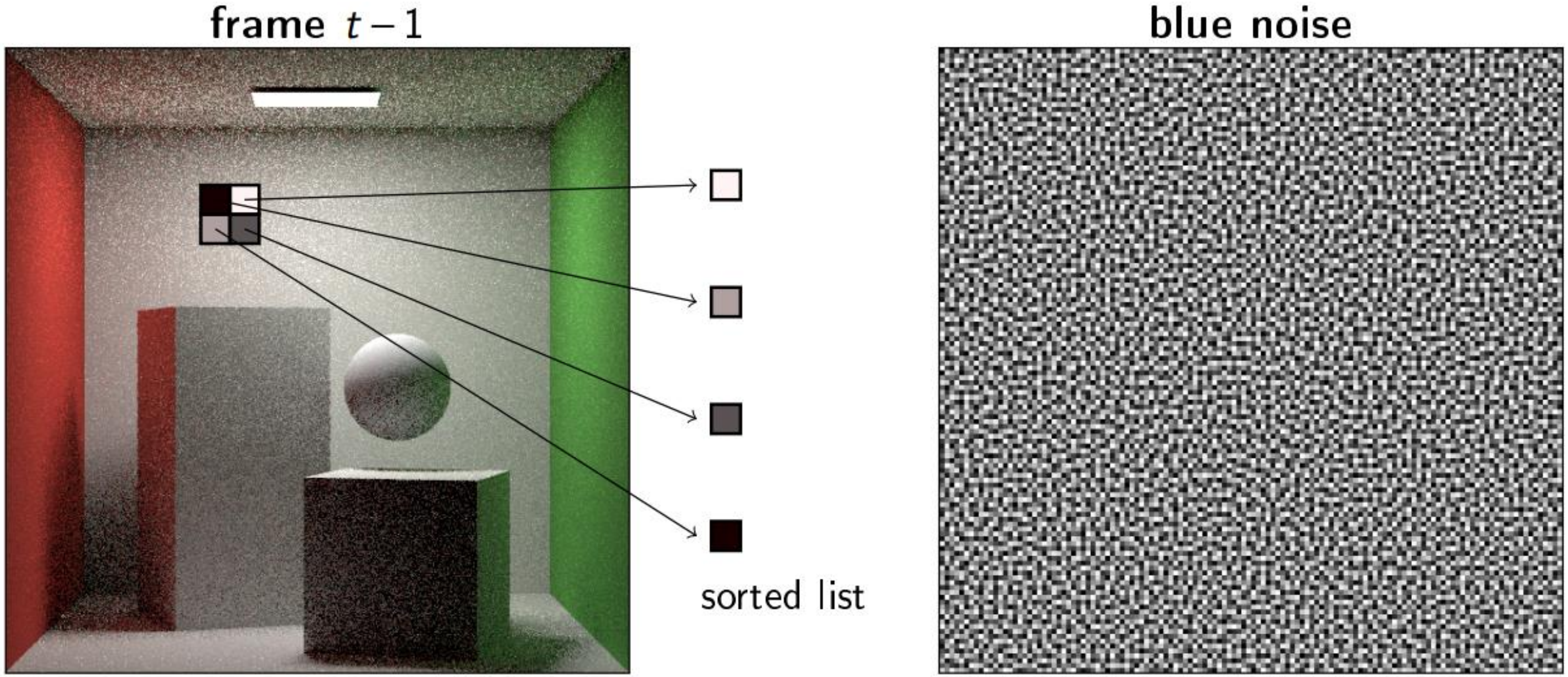
frame t

Temporal algorithm

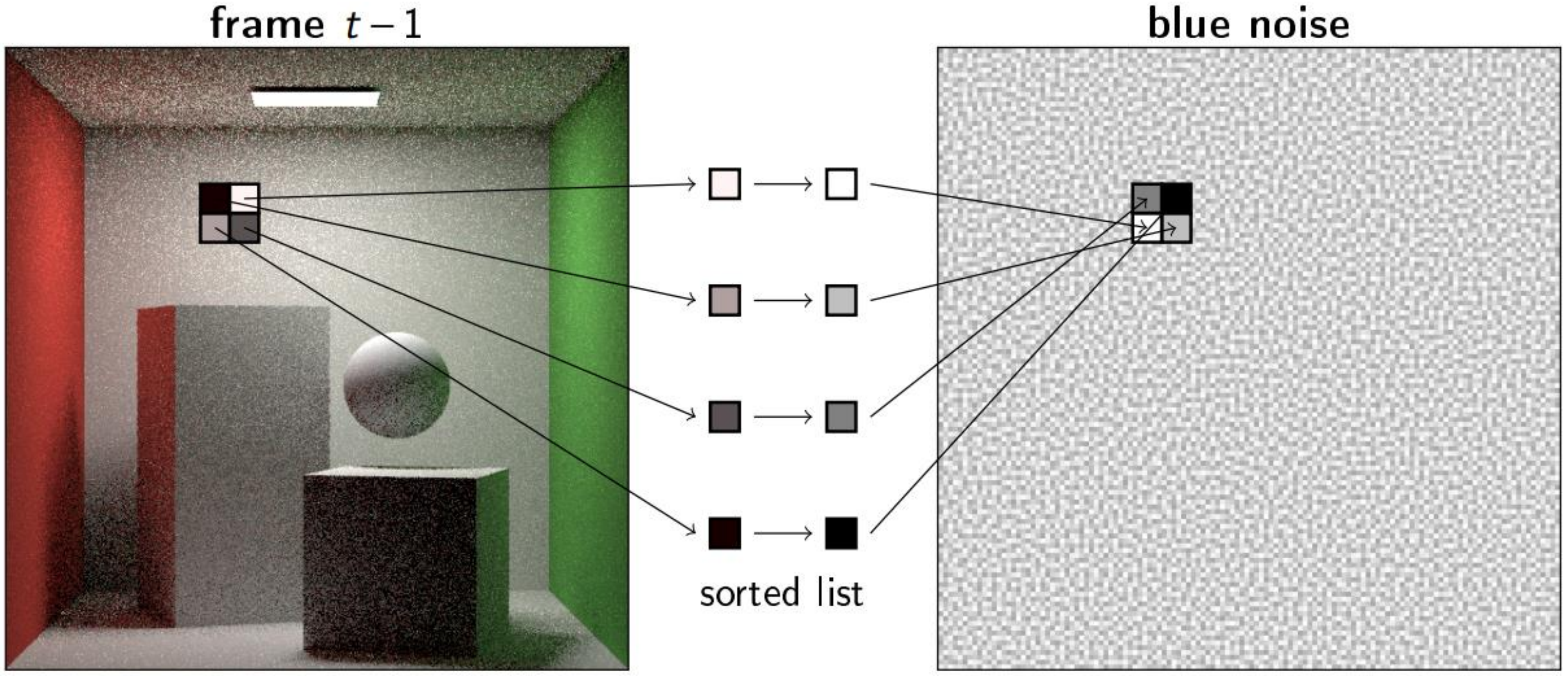




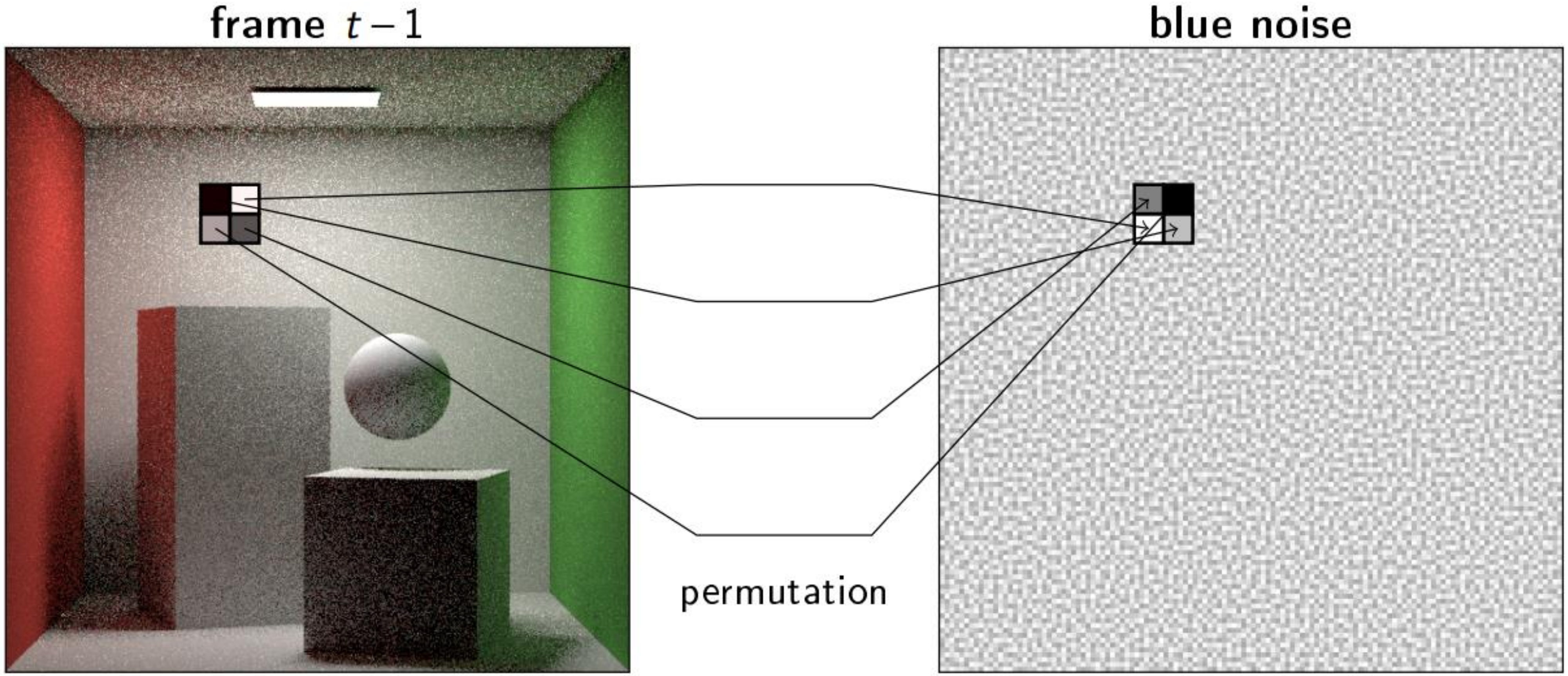
Temporal algorithm



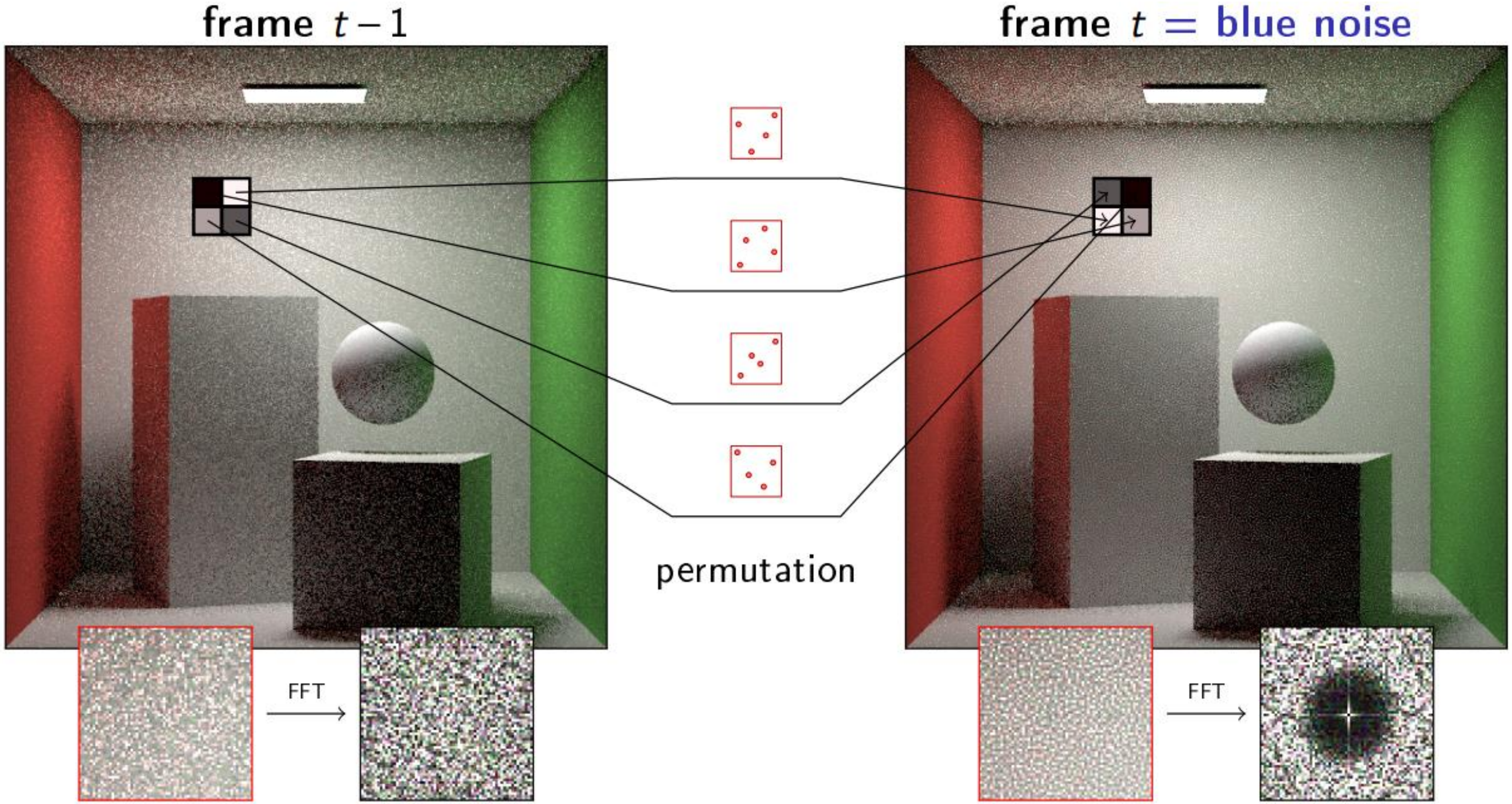
Temporal algorithm



Temporal algorithm



Temporal algorithm

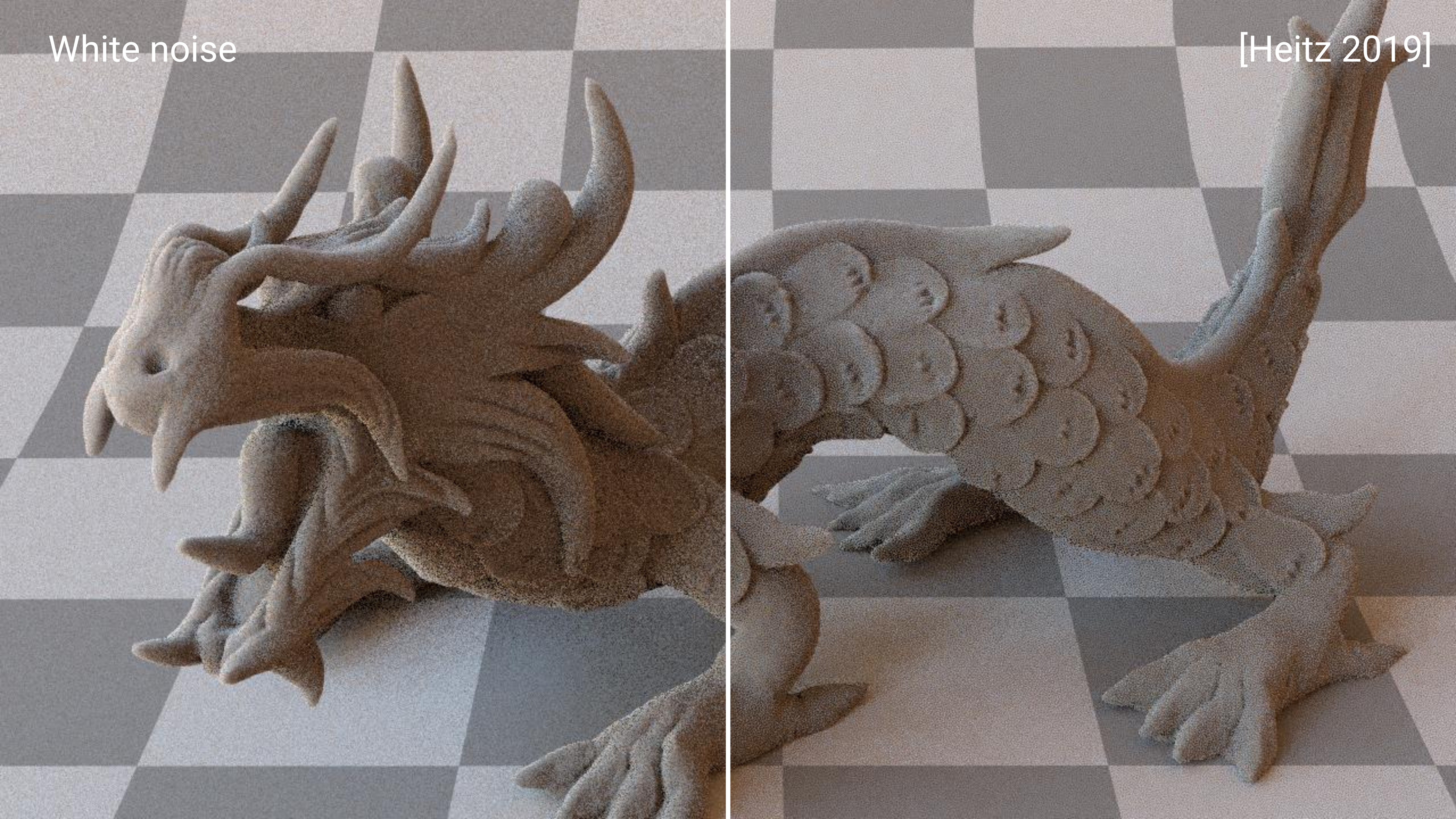


Pseudocode

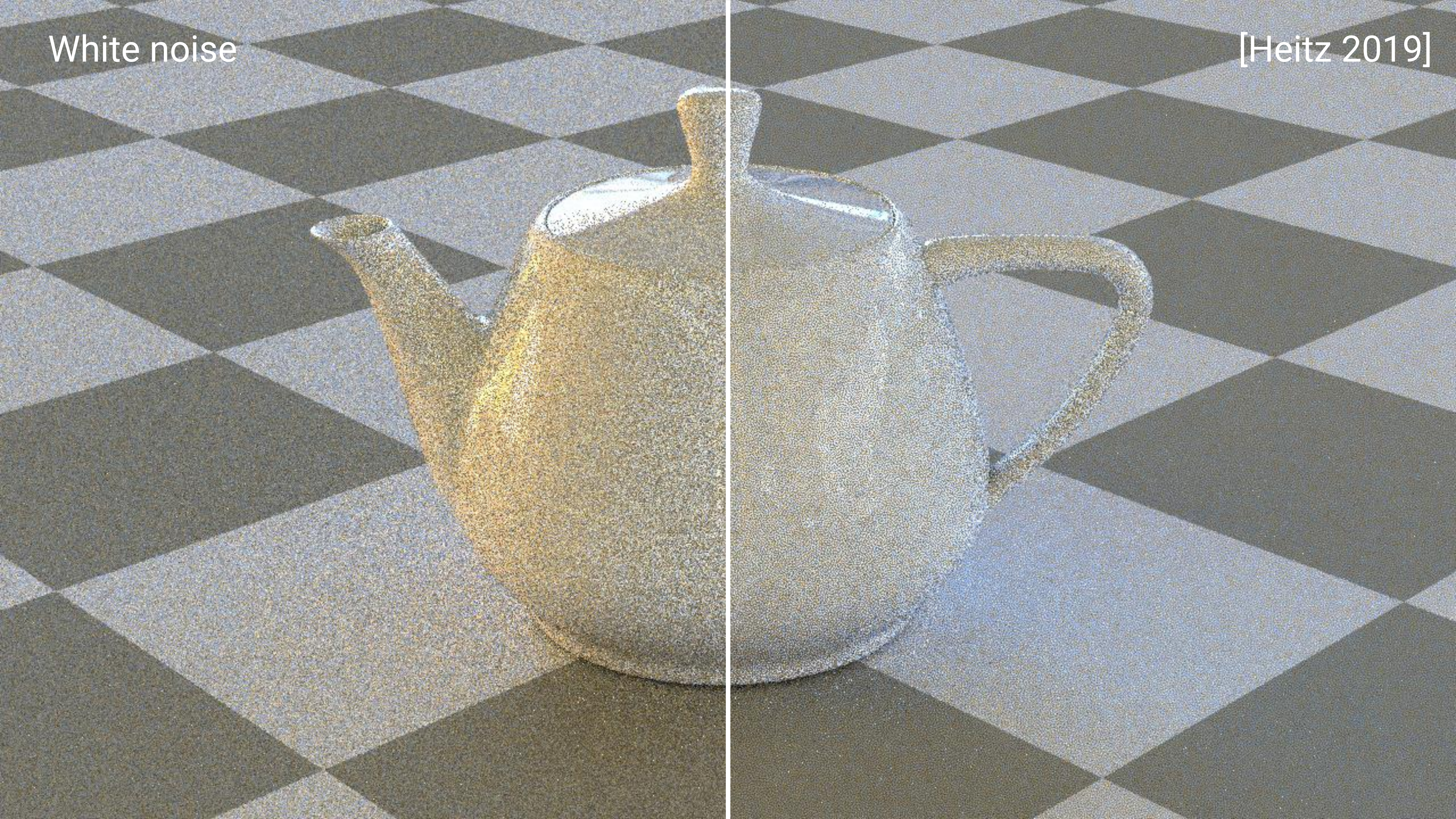
Per bloc :

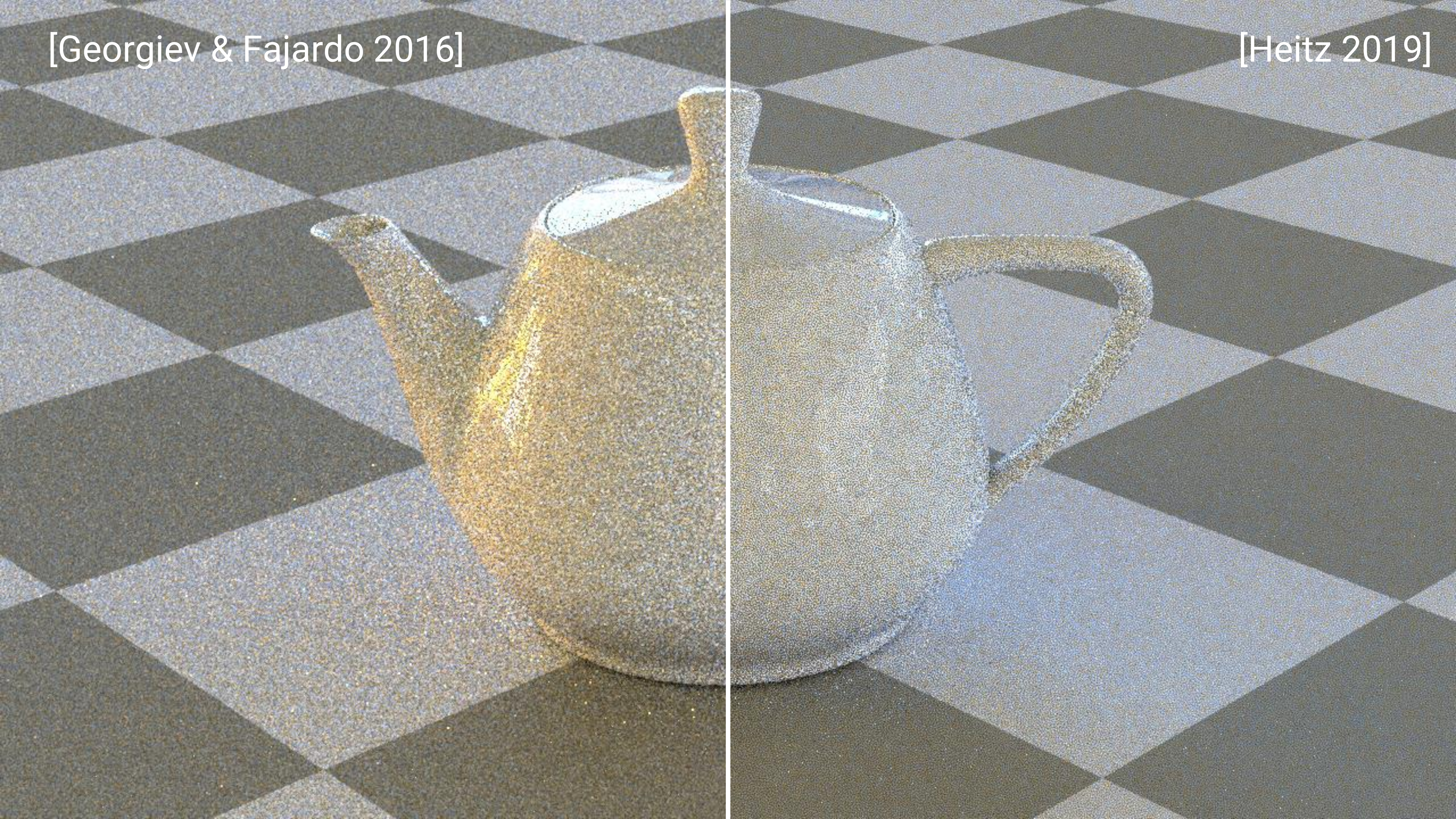
- Sort the dither mask and rendering values
- Assign in order the seed that produced the i-th value to the position of the i-th dithering value
- Render the next frame

Require: I Current rendering, S Current seed, D dither mask
  $S_{t+1} \leftarrow S$
  for each 4×4 bloc b **do**
    $I_b \leftarrow \mathrm{sort}(I[b])$
    $D_b \leftarrow \mathrm{sort}(D[b])$
    for i in $[0..16]$ **do**
      $S_{t+1}(D_b(i)) \leftarrow S(I_b(i))$
    end for
  end for
  Return S_{t+1}


White noise
[Heitz 2019]


[Georgiev & Fajardo 2016]
[Heitz 2019]


White noise
[Heitz 2019]


[Georgiev & Fajardo 2016]
[Heitz 2019]

Results

- Really high quality result in best case
- Scene specific sample distribution
- Works for complex rendering setup
- Low overhead
- Need to render every frames with the same seeds
- Break on edges/texture with significantly worse visual quality
    - Can be improved with some segmentation map but increase cost significantly

Scalable Multi-Class Sampling via Filtered Sliced Optimal Transport

[Salaun & al 2022]

Objectives

- Previous works used arbitrary energy minimization
  - Need provably good energy is needed
  - Lack of theoretical understanding

- This work propose
  - Theorical framework explaining why blue noise error distribution is a good choice
  - General energy term derived from Monte Carlo error
  - An optimization strategy for arbitrary perception metric

Error distribution as sample optimization

- For single pixel it's possible to measure sampling quality with discrepancy
  - Use error upper bound based on sampling quality and function variation
  - This equation is not differentiable and slow to evaluate

$$|I - \frac{1}{n}\sum_{i=1}^{n} f(x_i)| \leq V(f) \cdot D$$

- It is possible to define a similar bound using a differentiable metric (Wasserstein distance)
  - Based on Lipschitz inequality
  - This can be used to optimize pointset [Paulin et al. 2020]

$$\left|\int f(x)\,dx - \frac{1}{n}\sum_{i=1}^{n} f(x^i)\right| \leq \mathrm{Lip}(f)\,.\,W_1(X, 1_\Omega).$$

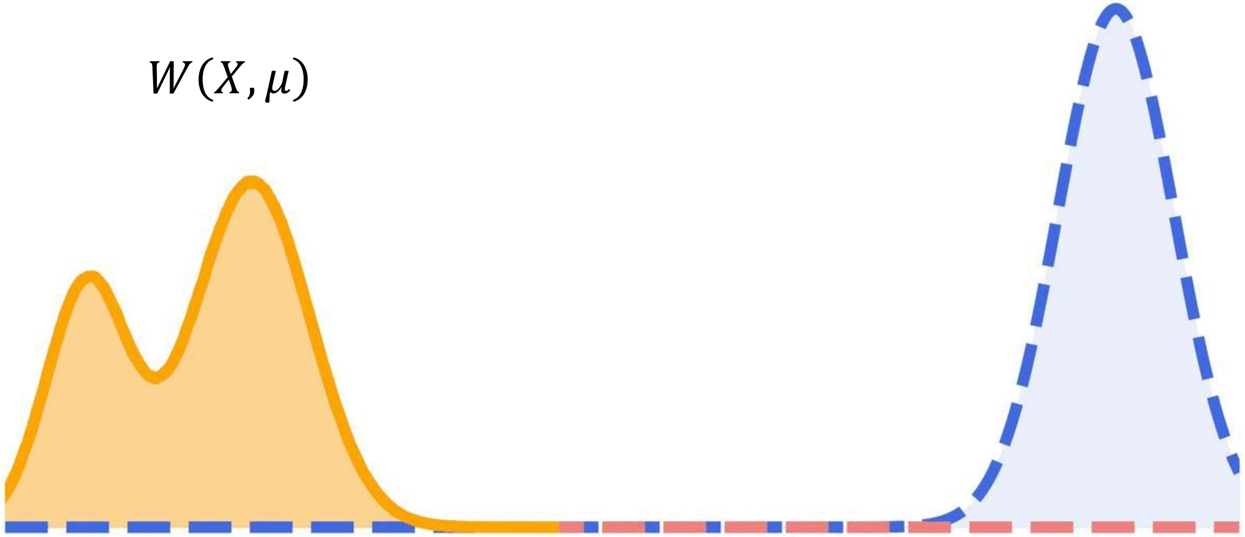
Wasserstein distance

The Wasserstein distance is define by the optimal transport plan minimizing the cost of moving the distribution X to the target distribution μ

- The cost is the total mass displaced per unit distance
- Thinking about 2 piles of sand : Where each sand grain should go to minimize displacement cost ?

- Costly to evaluate even with discreet distribution

$W(X, \mu)$

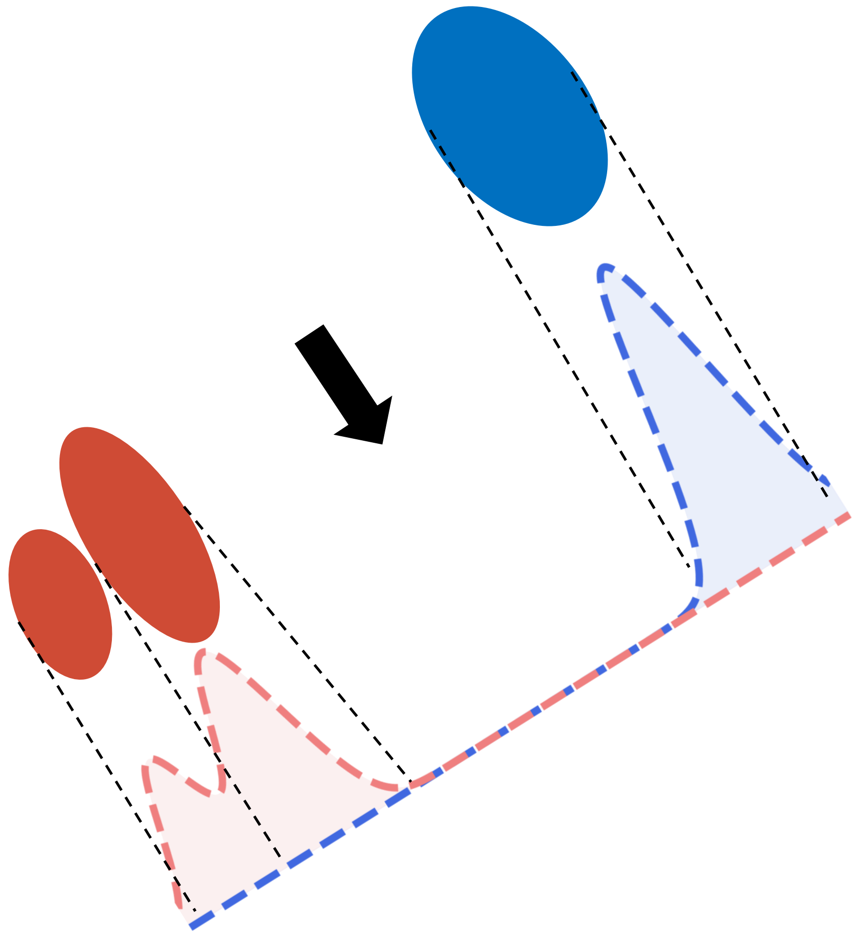


Sliced Wasserstein distance

- Full dimensional Wasserstein distance is often to costly to be used
- Instead we can use an upper bound using the sliced version

$$SW(X, U) = \int_{S^{d-1}} W(X^\theta, U^\theta) d\theta$$

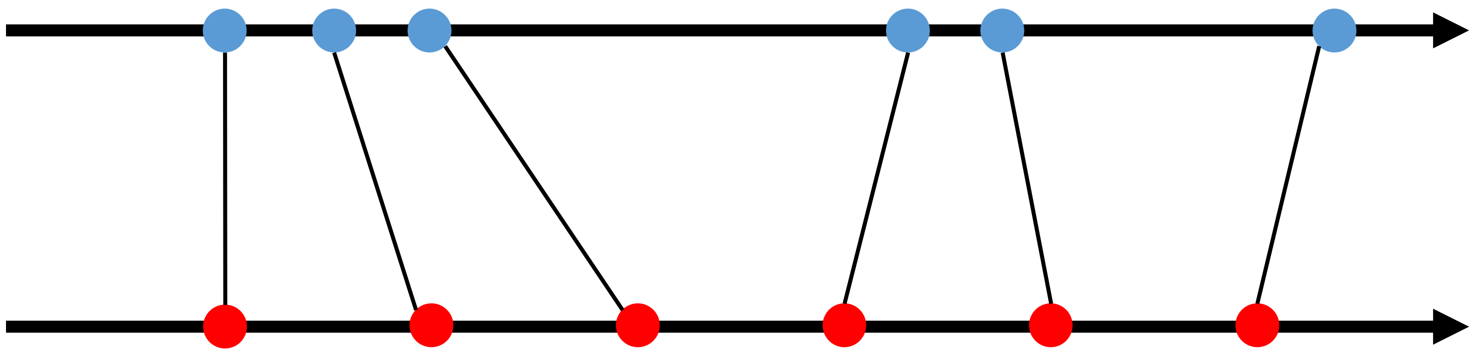
- This integral over all direction can be solved using Monte Carlo estimation



Sliced Wasserstein distance

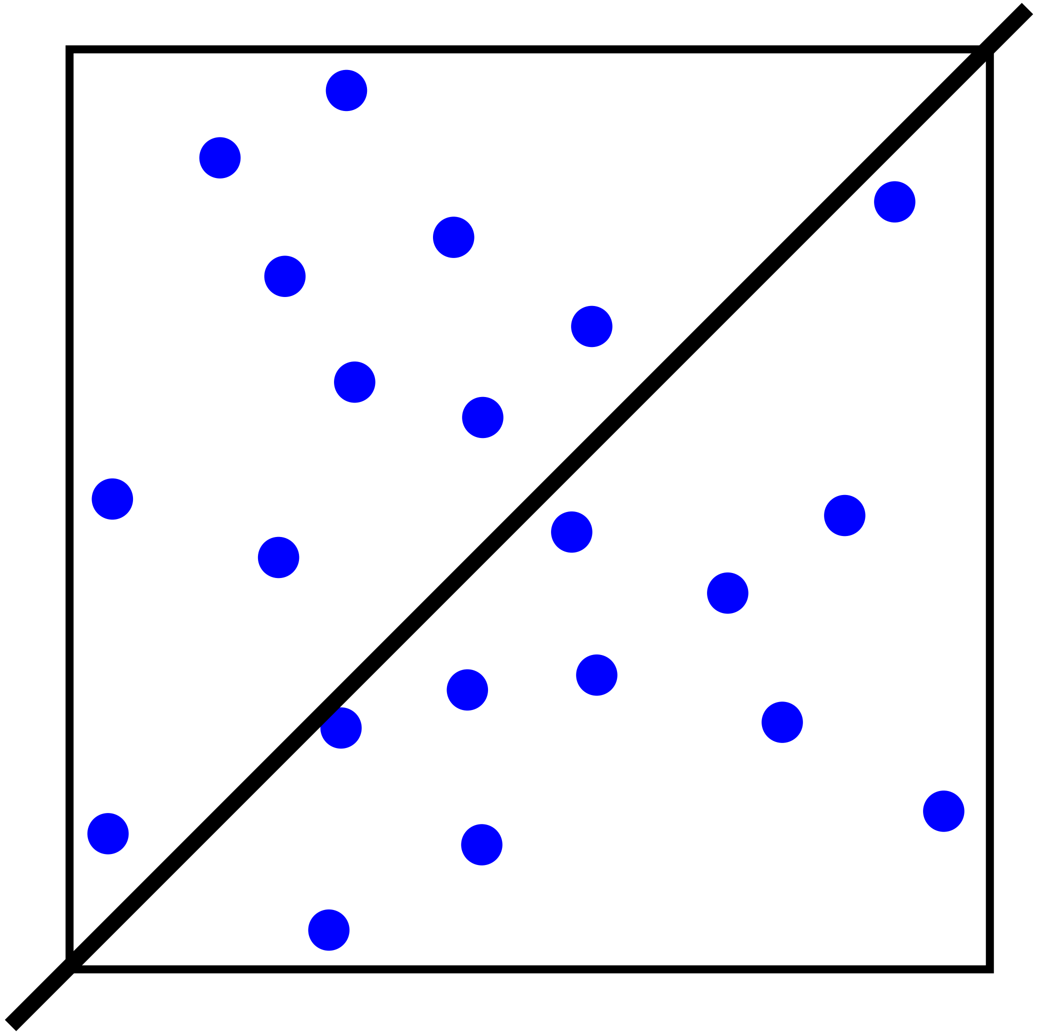
- For each slice we need to solve the optimal transport plan
  - 1D optimal transport between 2 discreet distribution has a simple solution
  - Also work with discretization in semi discreet case

Solving the 1D optimal transport only require sorting points



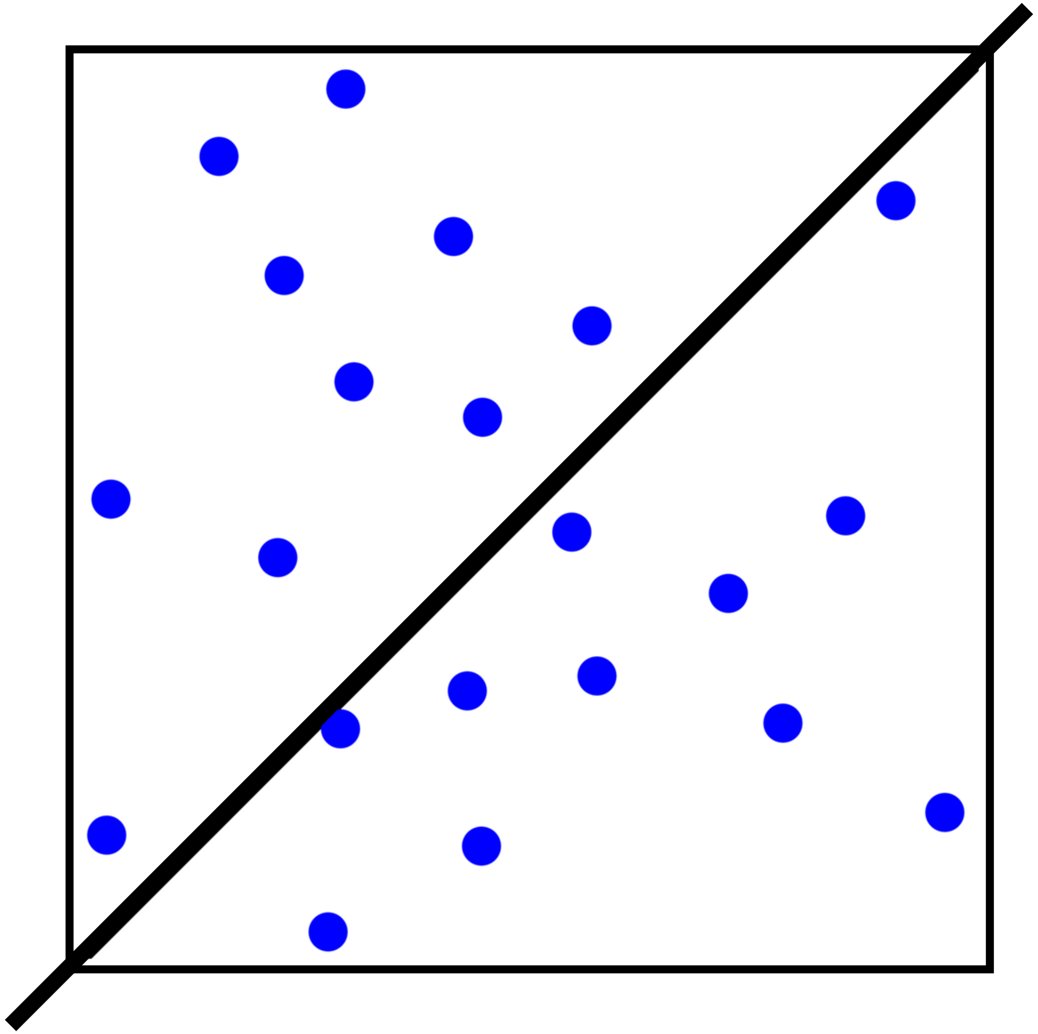
Sliced Wasserstein distance derivatives

- It is possible to compute the derivatives of the Sliced Wasserstein distance
  - For all sliced, the derivative is the vector for each point to the associated target



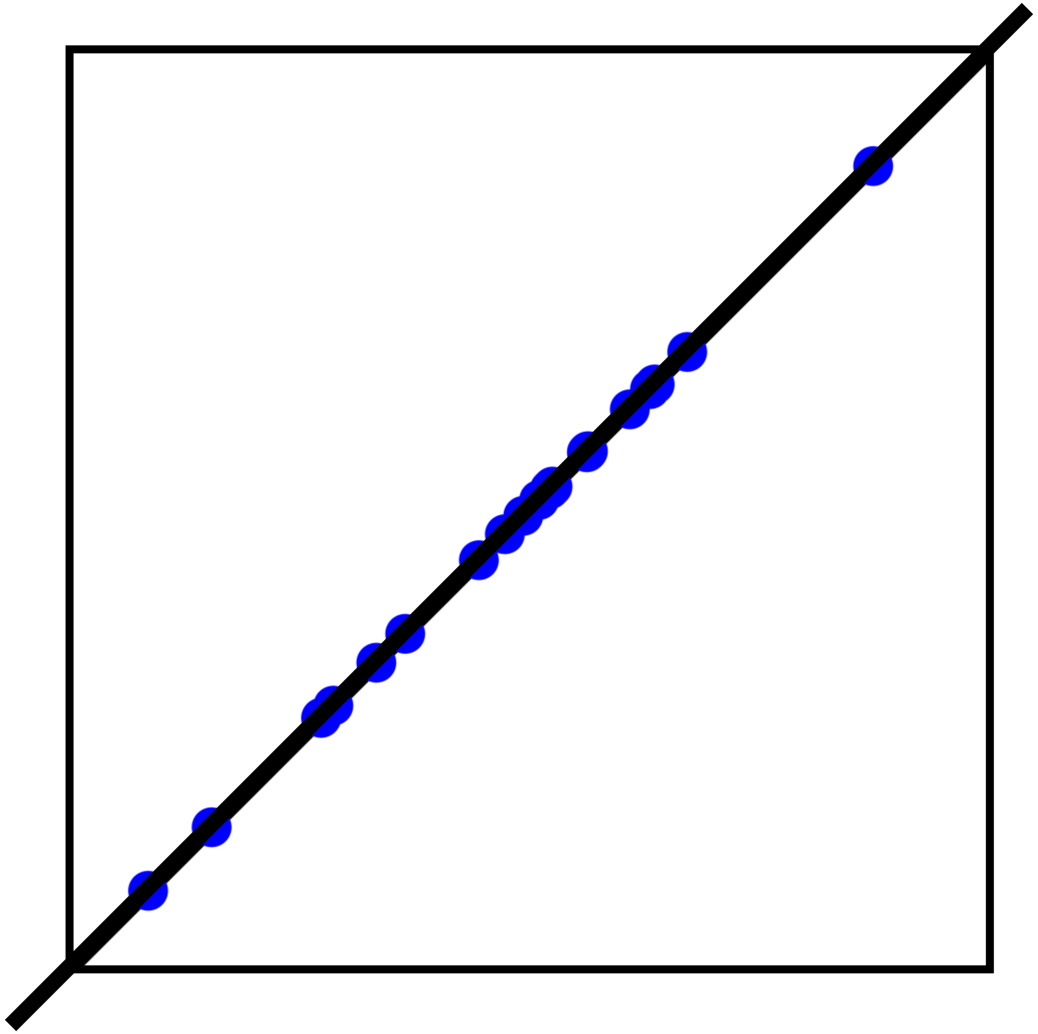
Sliced Wasserstein distance derivatives

- It is possible to compute the derivatives of the Sliced Wasserstein distance
  - For all sliced, the derivative is the vector for each point to the associated target



Sliced Wasserstein distance derivatives

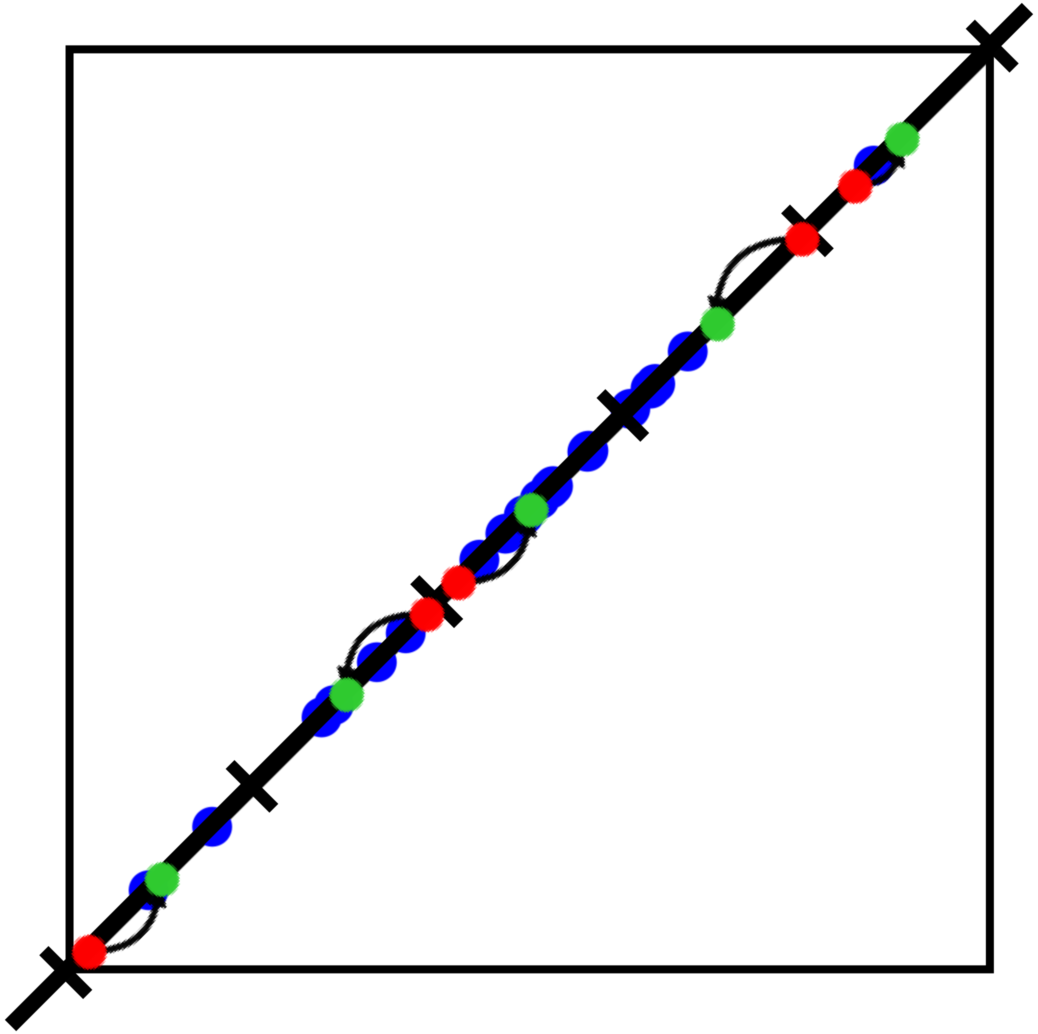
- It is possible to compute the derivatives of the Sliced Wasserstein distance
  - For all sliced, the derivative is the vector for each point to the associated target



Sliced Wasserstein distance derivatives

- It is possible to compute the derivatives of the Sliced Wasserstein distance
  - For all sliced, the derivative is the vector for each point to the associated target

- Finally use a simple Gradient based optimization to optimize the pointset
  - The gradient are noisy but will get smoothed over iterations



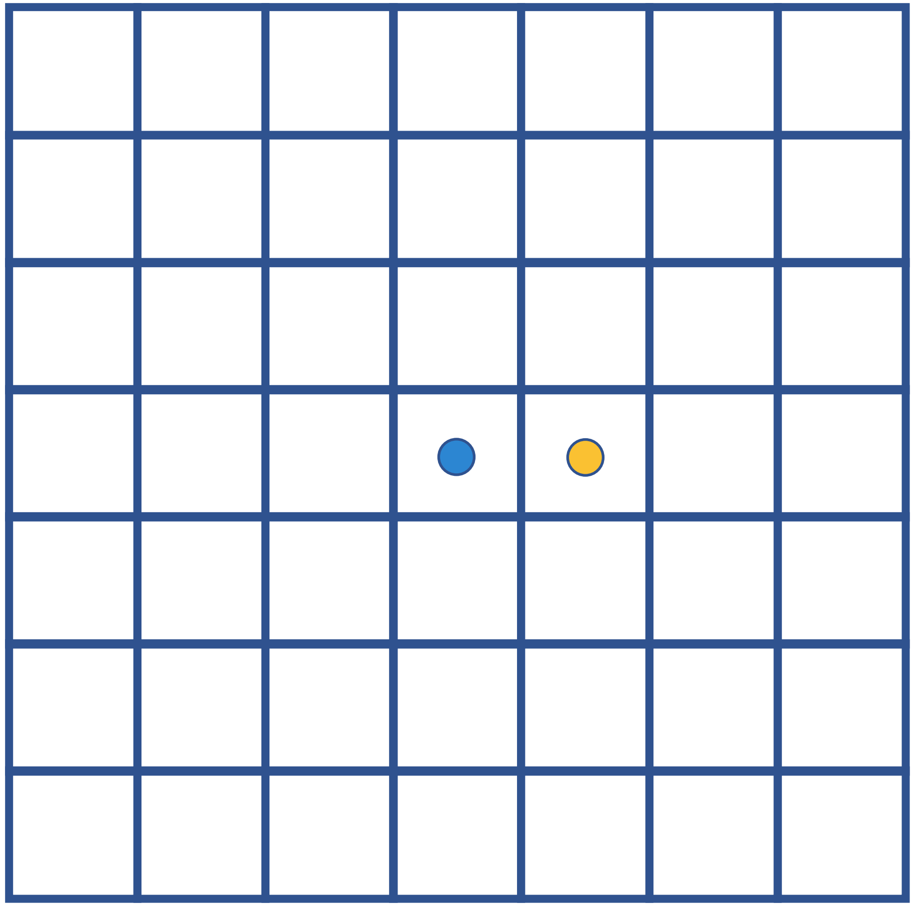
Sliced optimal transport sampling algorithm

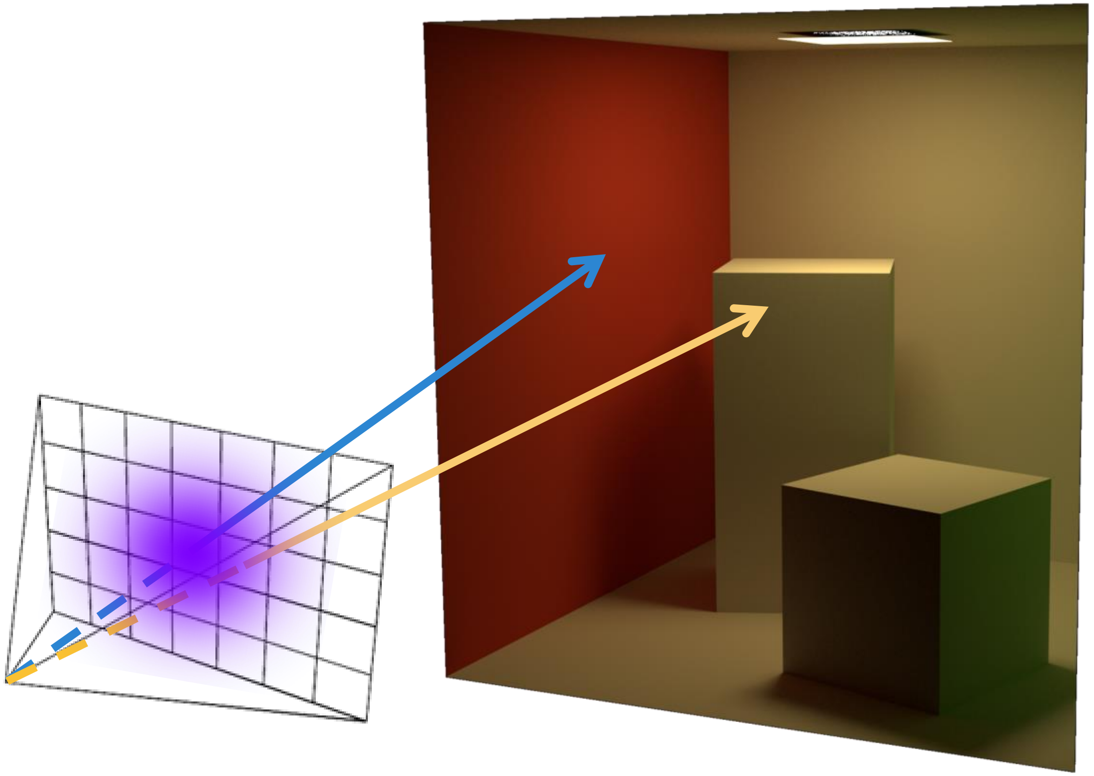
- Randomly regenerate U at each step to avoid overfit to random points
- It's possible to use advanced gradient descent
- Possible to average multiple projection at each step and parallelize it
- Relatively fast convergence but still an optimization task

Example

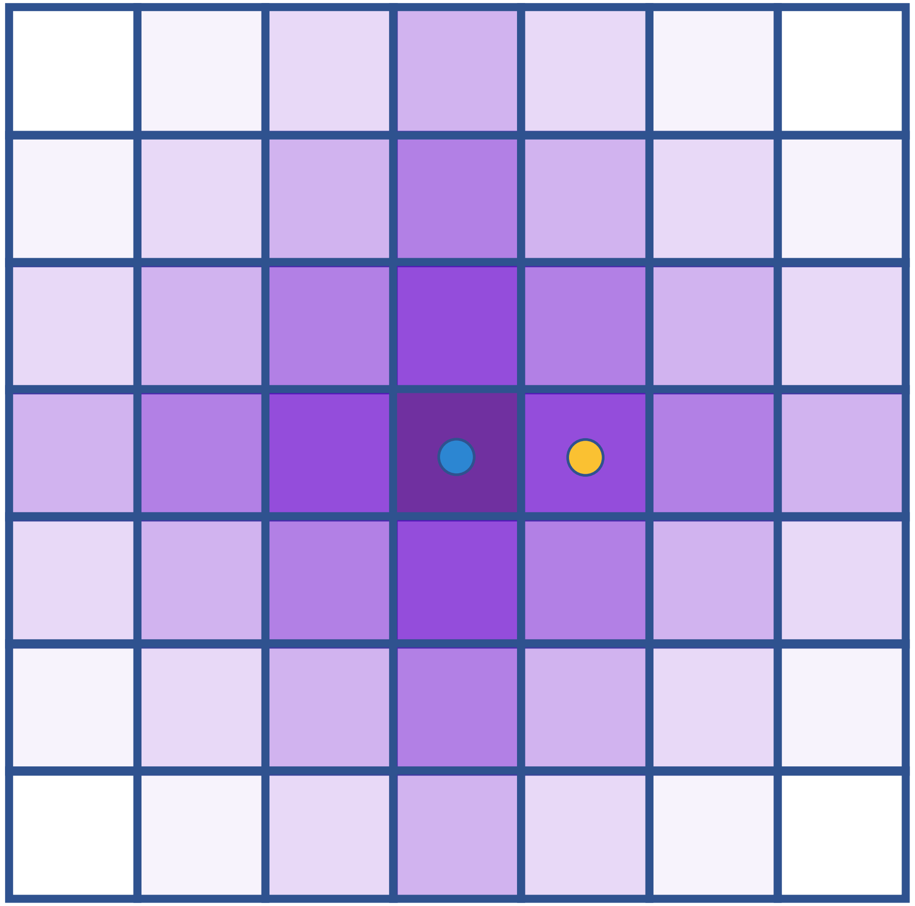
$$
\begin{array}{l}
X \leftarrow \mathrm{rand}(N, D) \\
\textbf{for } i \in \{0, i_{max}\} \textbf{ do} \\
\quad \theta \leftarrow \mathrm{rand\_direction}(D) \\
\quad U \leftarrow \mathrm{rand}(N, D) \\
\quad X^\theta \leftarrow X \cdot \theta \\
\quad U^\theta \leftarrow U \cdot \theta \\
\quad X \leftarrow X + \lambda(U^\theta - X^\theta)\theta \\
\textbf{end for} \\
\textbf{Return } X
\end{array}
$$

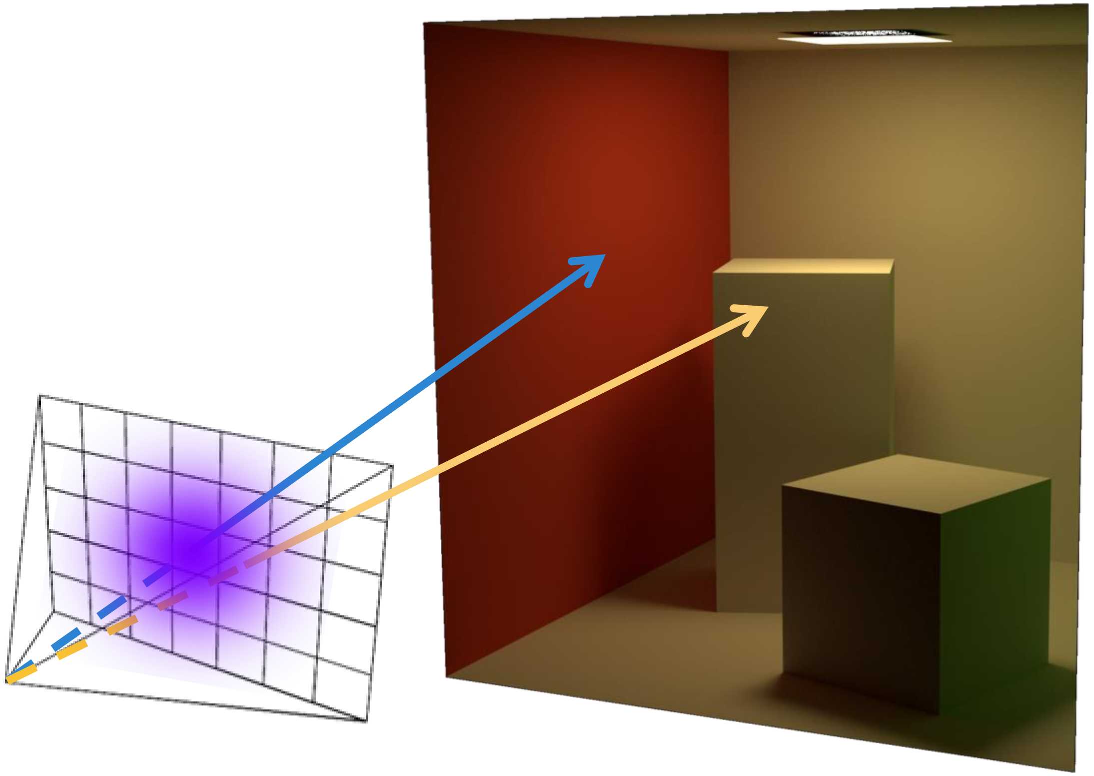
Multi class rendering





Multi class rendering

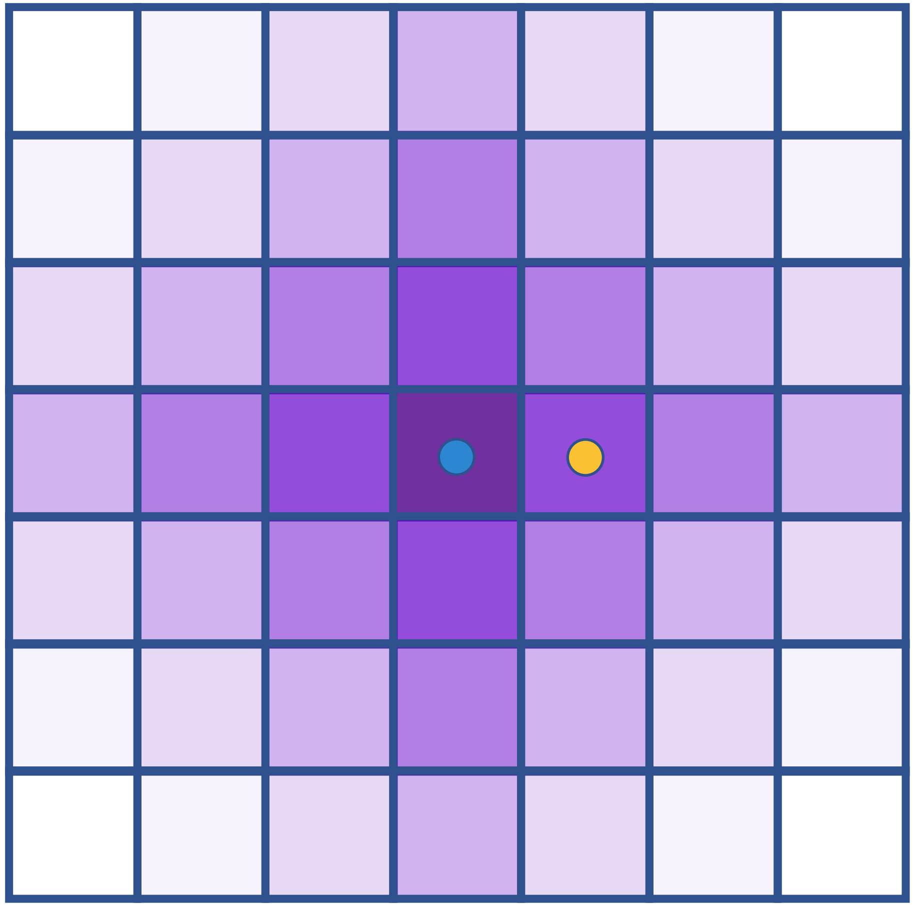


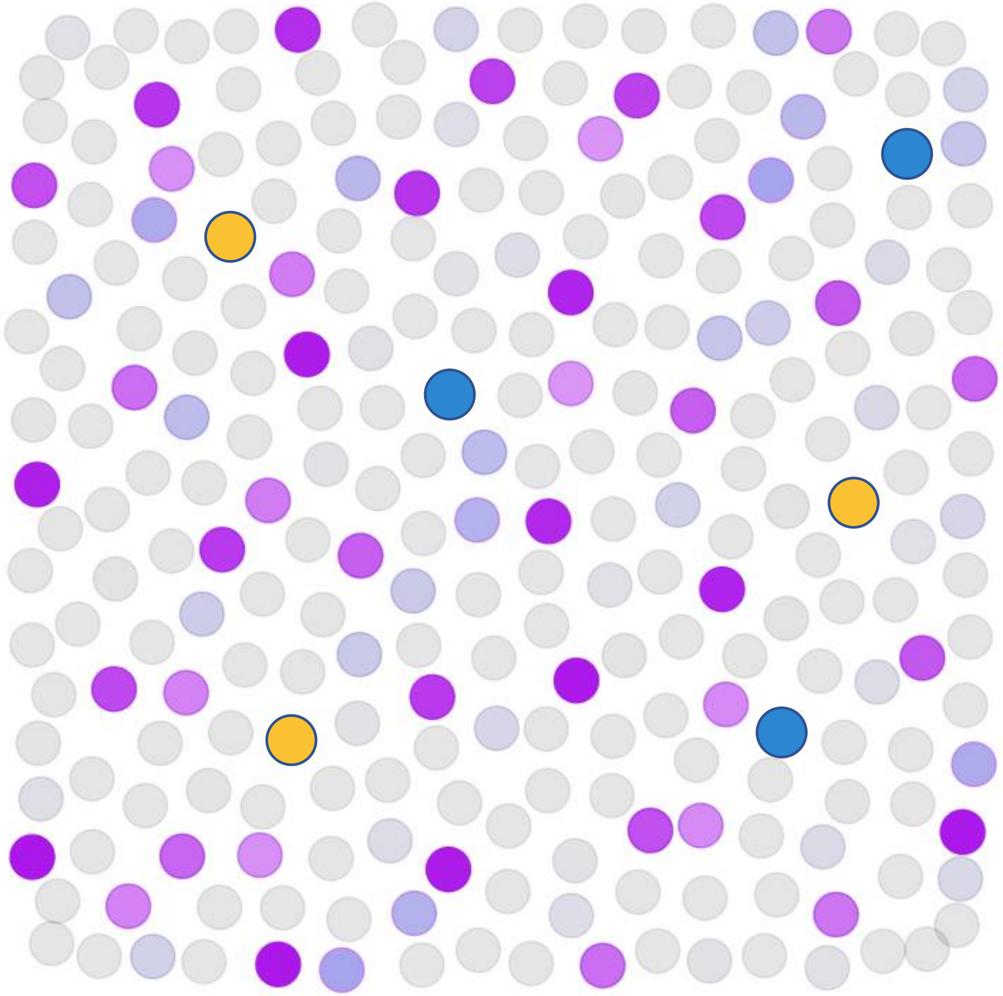


Multi class rendering

- Our eye see pixel together
  - The samples of multiple pixel are seen together
  - Need a commune optimization
  - One sample contribute to multiple pixel integration (into our eye)
    - Some have more contribution than others
- Contrary to previous work we consider pixel integration not independent
  - Samples from multiple pixel can be seen as a single sampling sequence
  - Lot of conflicting objectives

Multi class rendering





Error distribution as sample optimization

- It is possible to define a Multi-class error bound
  - Sum of weighted contribution of neighboring pixels
  - All samples of the image contributes to every other pixel
- It make sample contribution non binary
  - Samples may contribute to the integration with varying factors
  - We introduce Filtering into the Wasserstein distance
  - End up to a barycentric optimization

$$Q_w(X) = \frac{1}{N} \sum_{x_i \in X} \underbrace{w(x_i)}_{\text{image-space Gaussian}} f(x_i) \approx I_w$$

$$\varepsilon_w(X) = |Q_w(X) - I_w|$$

$$\sum_p \varepsilon_{w_p}(X) \leq L_f \sum_p B_1(X, w_p, U) = Loss$$

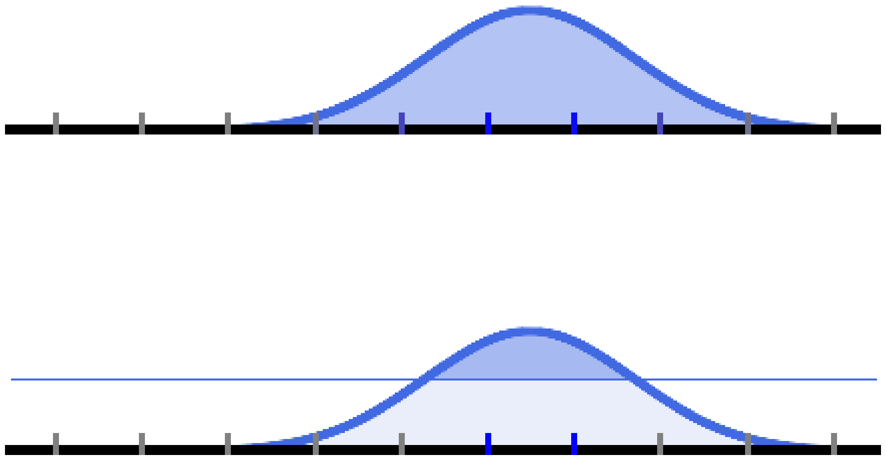
$$B_1(X, w_p, U) = \int_{\mathbb{R}} W(X_{w>z}, U_{w>z}) dz$$

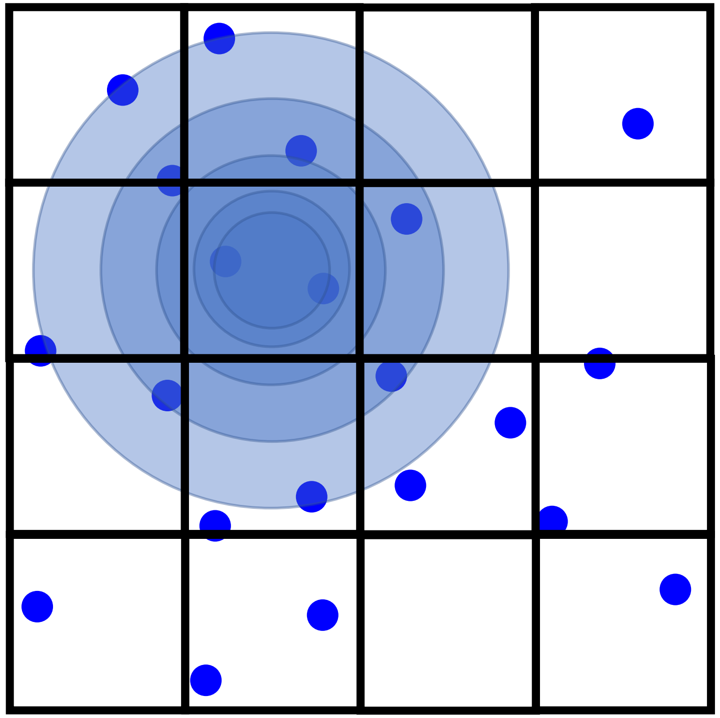
Filtered Wasserstein distance

Single optimization for multiple objectives

$$B_1(X, w_p, U) = \int_{\mathbb{R}} W(X_{w>z}, U_{w>z})dz$$

Decompose a complex optimization into many small problems



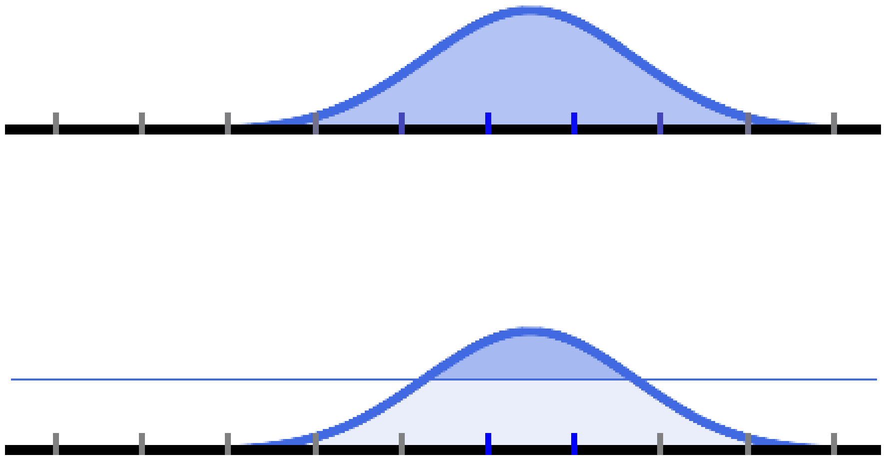


Filtered Wasserstein distance

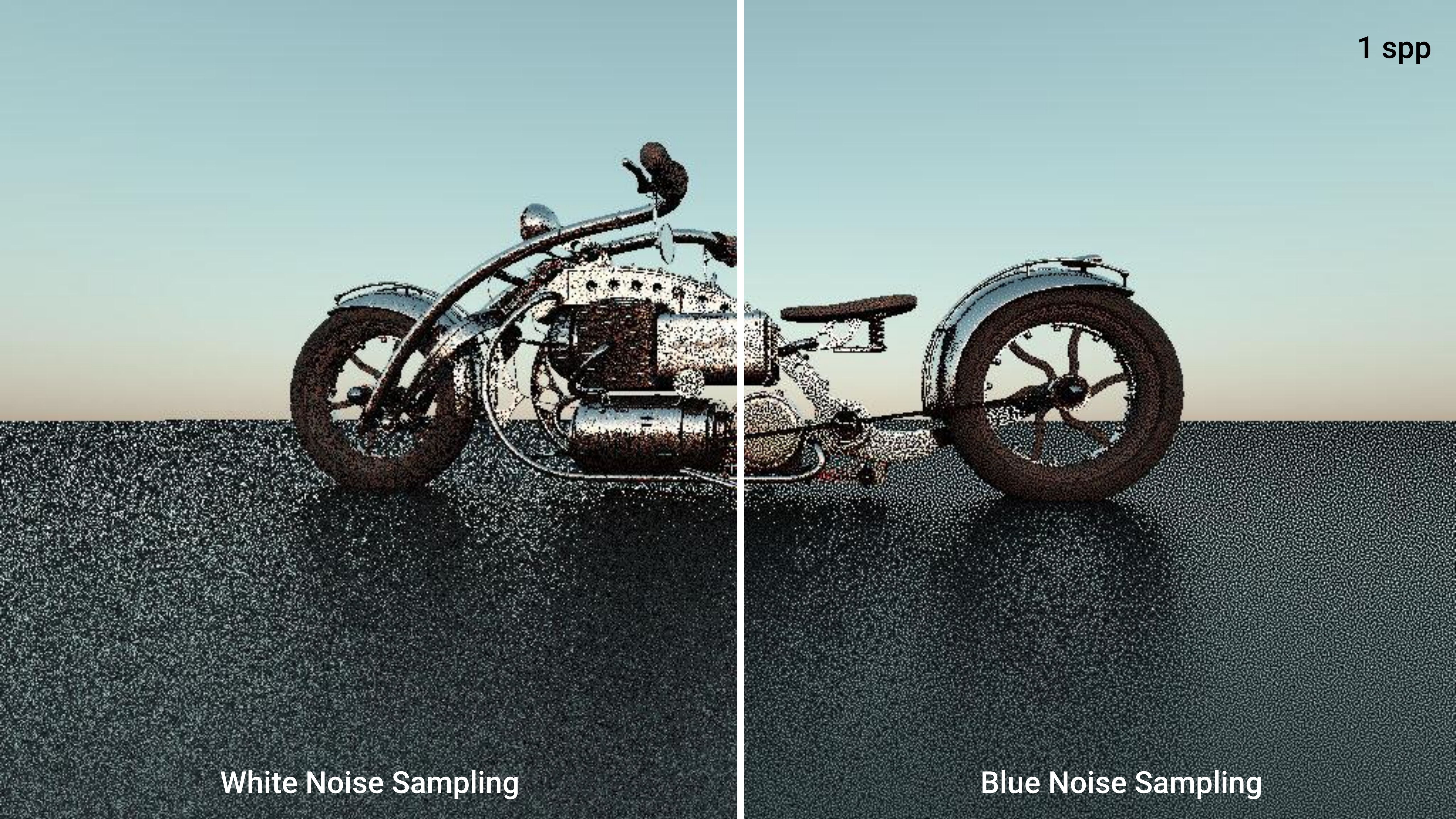
Single optimization for multiple objectives

$$B_1(X, w_p, U) = \int_{\mathbb{R}} W(X_{w>z}, U_{w>z})dz$$

Decompose a complex optimization into many small problems



$X \leftarrow \mathrm{rand}(N, D)$
for $i \in \{0, i_{max}\}$ **do**
  $w_i \leftarrow \mathrm{rand\_Kernel}()$
  $z \leftarrow \mathrm{rand}(0, 1)$
  $X_{w_i>z} \leftarrow \mathrm{sample\_selection}(X, w_i, z)$
  $\theta \leftarrow \mathrm{rand\_direction}(D)$
  $U_{w_i>z} \leftarrow \mathrm{rand}(N_{w_i>z}, D)$
  $X^{\theta}_{w_i>z} \leftarrow X_{w_i>z} \cdot \theta$
  $U^{\theta}_{w_i>z} \leftarrow U_{w_i>z} \cdot \theta$
  $X_{w_i>z} \leftarrow X_{w_i>z} + \lambda(U^{\theta}_{w_i>z} - X^{\theta}_{w_i>z})\theta$
end for
Return X


1 spp
White Noise Sampling
Blue Noise Sampling

Results

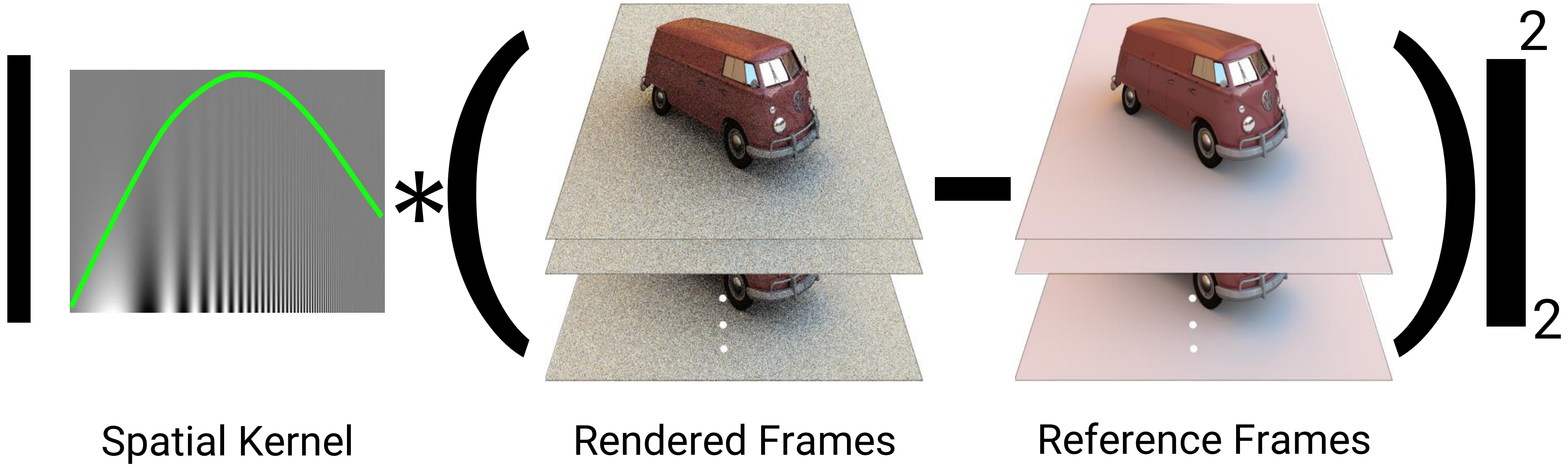
- This is the first gradient based optimization that produce blue noise error distribution
- The blue noise property is a consequence of the minimization of the L1 error bound
  - It mean BN is the expected property because of the perceptual kernel we choose
  - An other kernel could result in other correlation

- The method can be extended to more complex perception models
  - We used the same gaussian as other work for comparison

- Optimization is slow even using a SGD based optimizer
  - The reason is the important cost of computing Wasserstein distance
  - Need to average lot of Wasserstein distance per step to get noise reduction

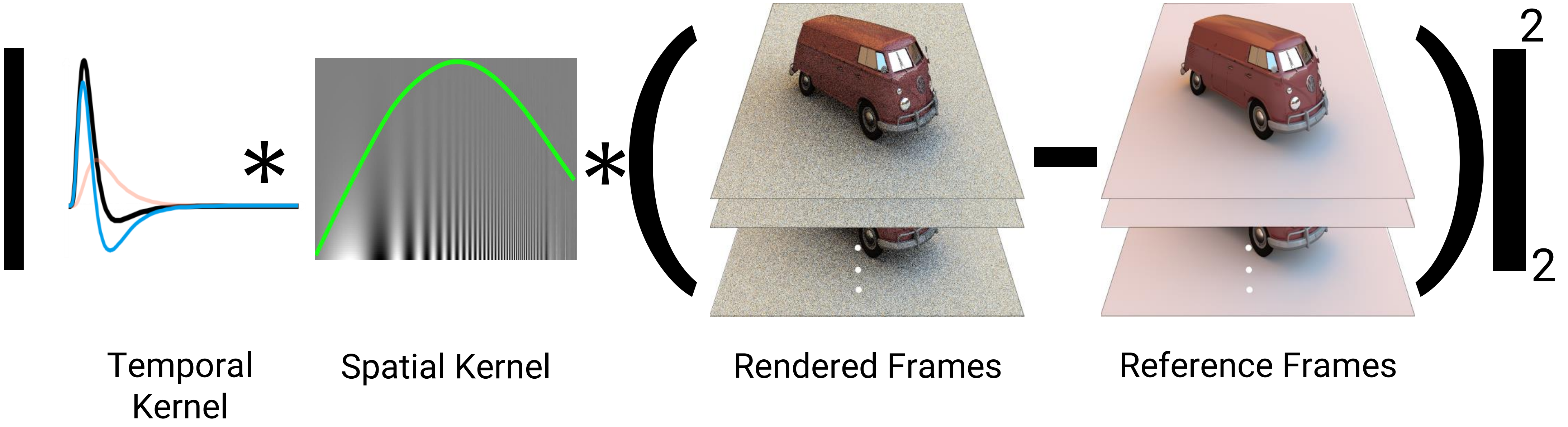
Temporal extension

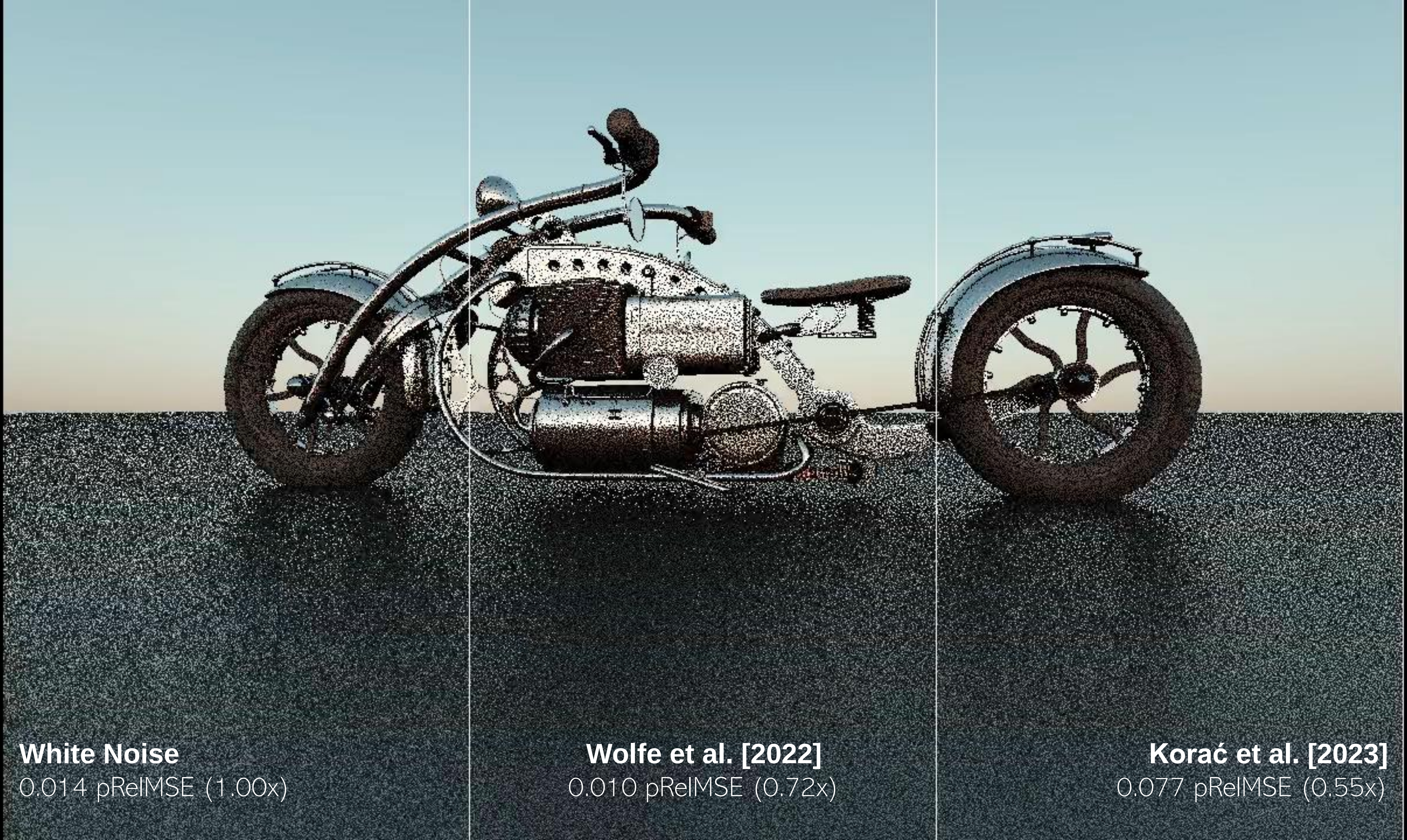
It's possible to model the perception in space and time



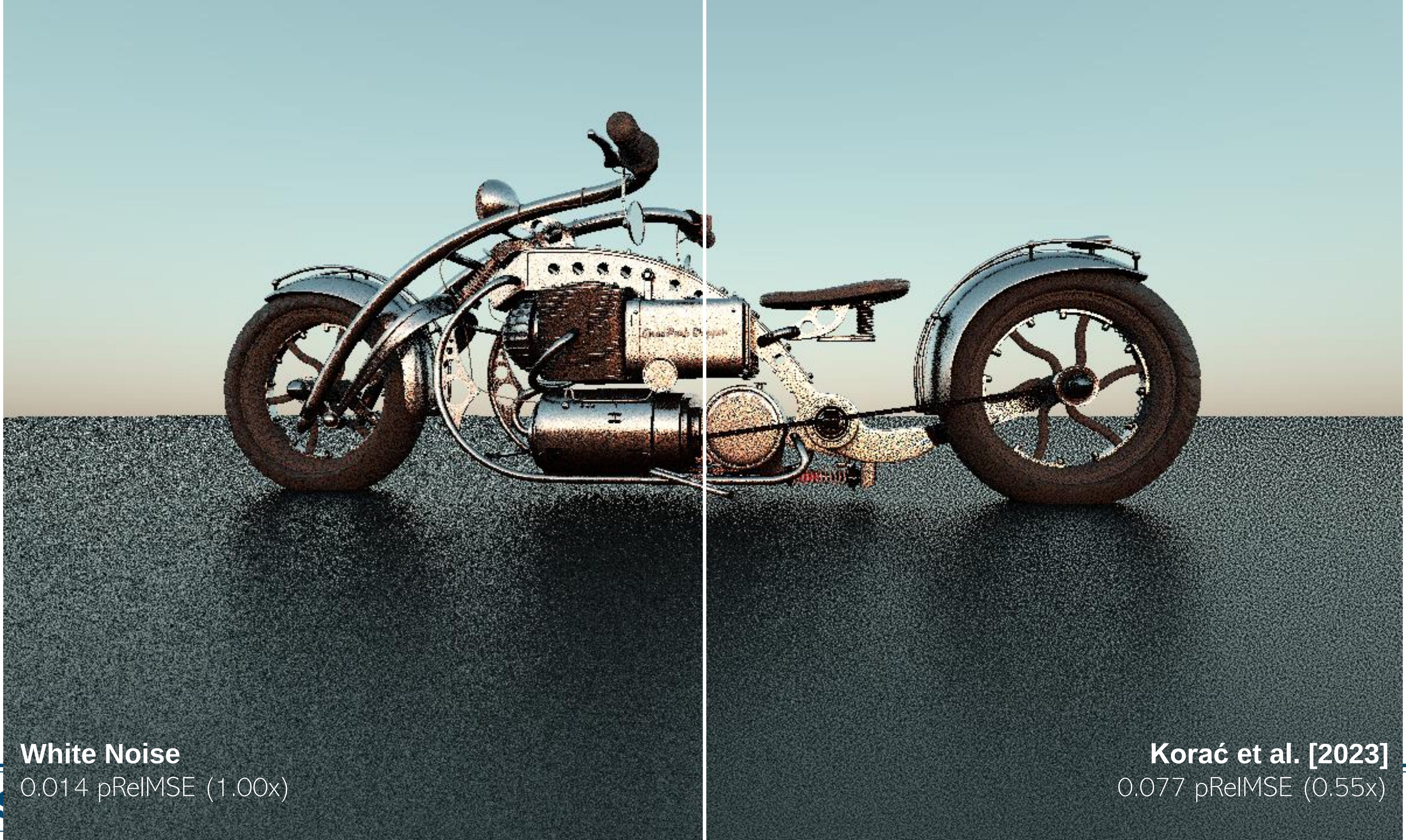
Temporal extension

It’s possible to model the perception in space and time





White Noise
0.014 pRelMSE (1.00x)
Wolfe et al. [2022]
0.010 pRelMSE (0.72x)
Korać et al. [2023]
0.077 pRelMSE (0.55x)


White Noise
0.014 pRelMSE (1.00x)
Korać et al. [2023]
0.077 pRelMSE (0.55x)


White Noise
0.0065 pRelMSE (1.00x)
Wolfe et al. [2022]
0.0043 pRelMSE (0.66x)
Korać et al. [2023]
0.0031 pRelMSE (0.48x)


White Noise
0.0065 pRelMSE (1.00x)
Korać et al. [2023]
0.0031 pRelMSE (0.48x)

Temporal extension

- Extension to temporal domain result in Blue noise error distribution
  - Blue noise property is only visible when frames are average or see in an animation
  - Each frame individually looks like uncorrelated noise
  - Result from the energy that aim only for temporal optimization not single frames

- Could be improve with reprojection and scene dependent information
  - All the samples works for multiple scene
  - No reprojection is done
  - Lot of improvement is possible

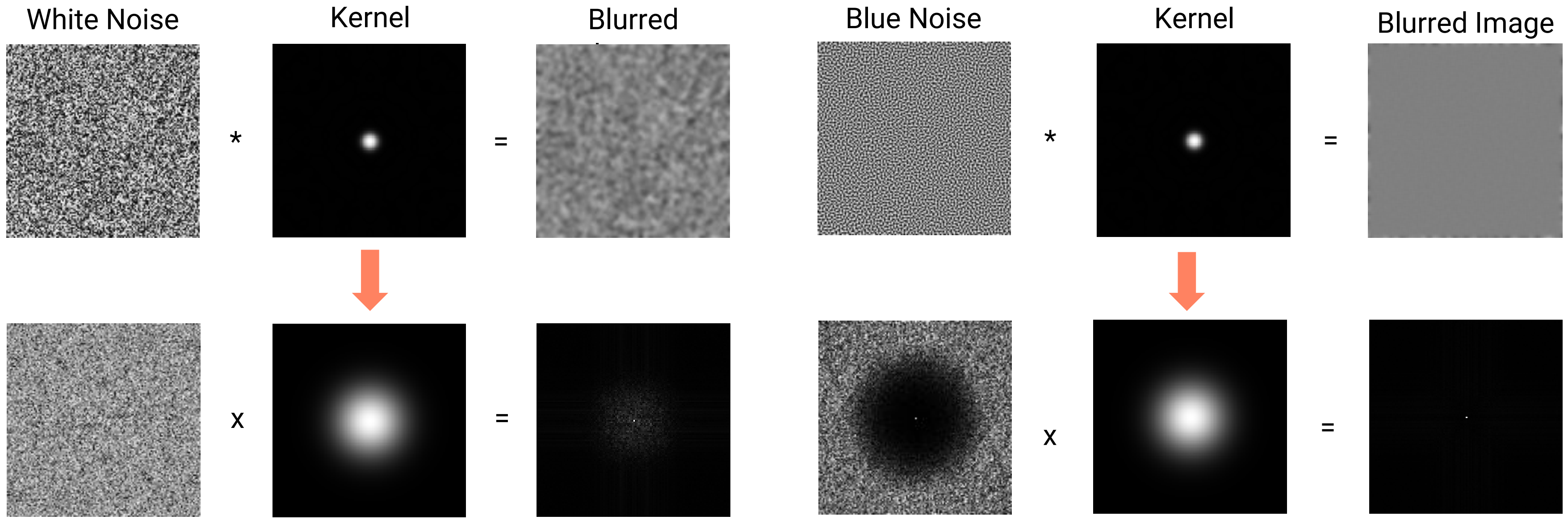
Blue noise error and denoising

Blue noise error and denoising

- All previous method focus on image perception as raw rendering
  - Noise perception only work on noisy rendering
- Denoising also benefit from high frequency noise
  - Simple blur follow exactly the same equations
  - Generally analytic filtering works
  - Neural based denoising require specific attention

Analytic denoising

- Denoising is often composted of low pass filters
  - Even when using bilateral filters
- More generally blue noise error distribution create a good sampling distribution for groups of pixel
  - More information locally



Neural based denoising

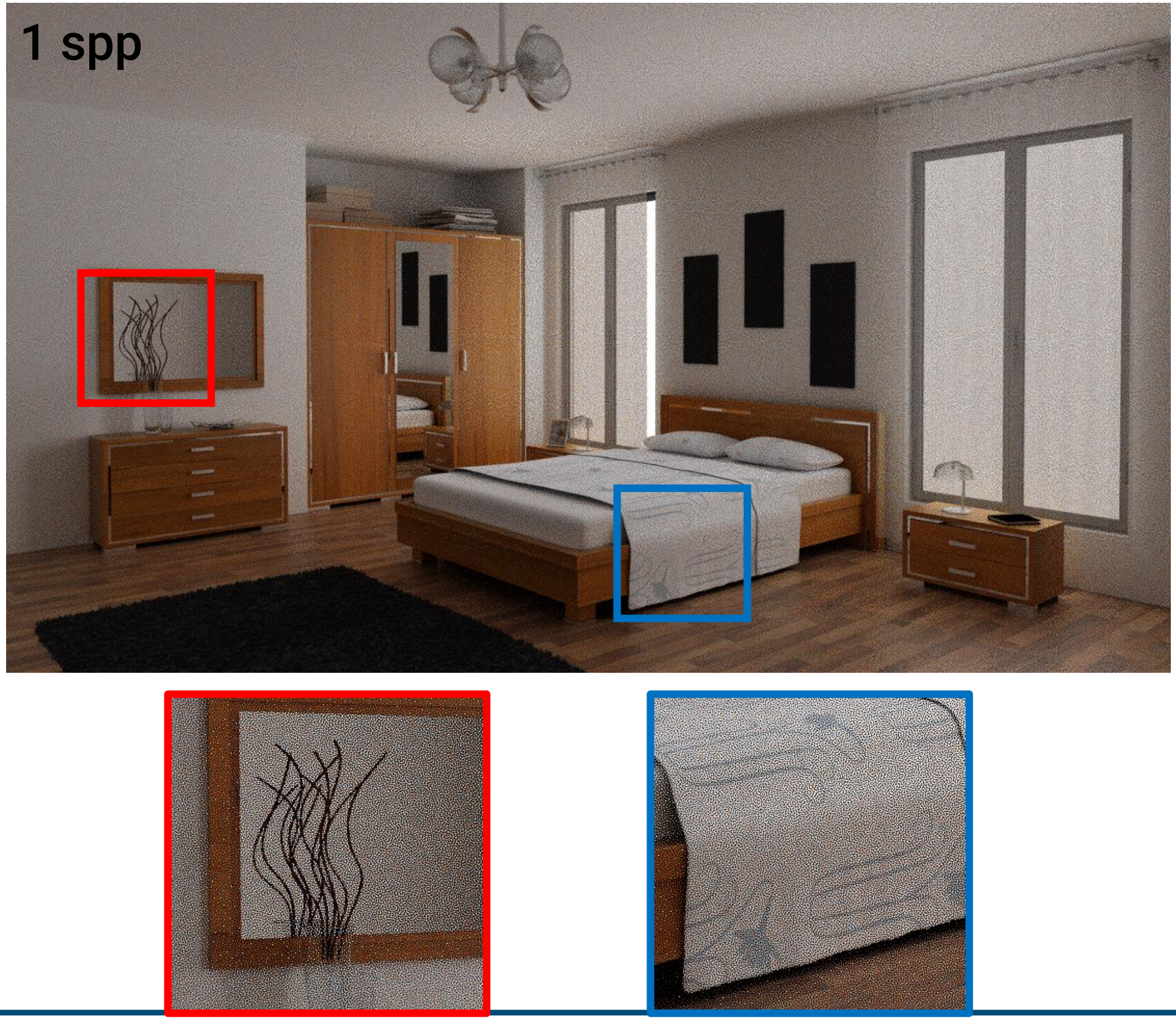
- Neural based denoiser I now the state of the art
  - High efficiency and order of magnitude higher quality
  - Can be guided with G-buffers

- Blue noise distribution works with Neural based denoiser
  - If the network is trained on this type of noise
  - Reduce randomness and better guide information to the denoiser
  - Particularly true with kernel prediction network

- Denoiser train on Blue noise achieve high error is denoising uncorrelated noise
  - Need to ensure Blue noise distribution is actually achieved

White Noise

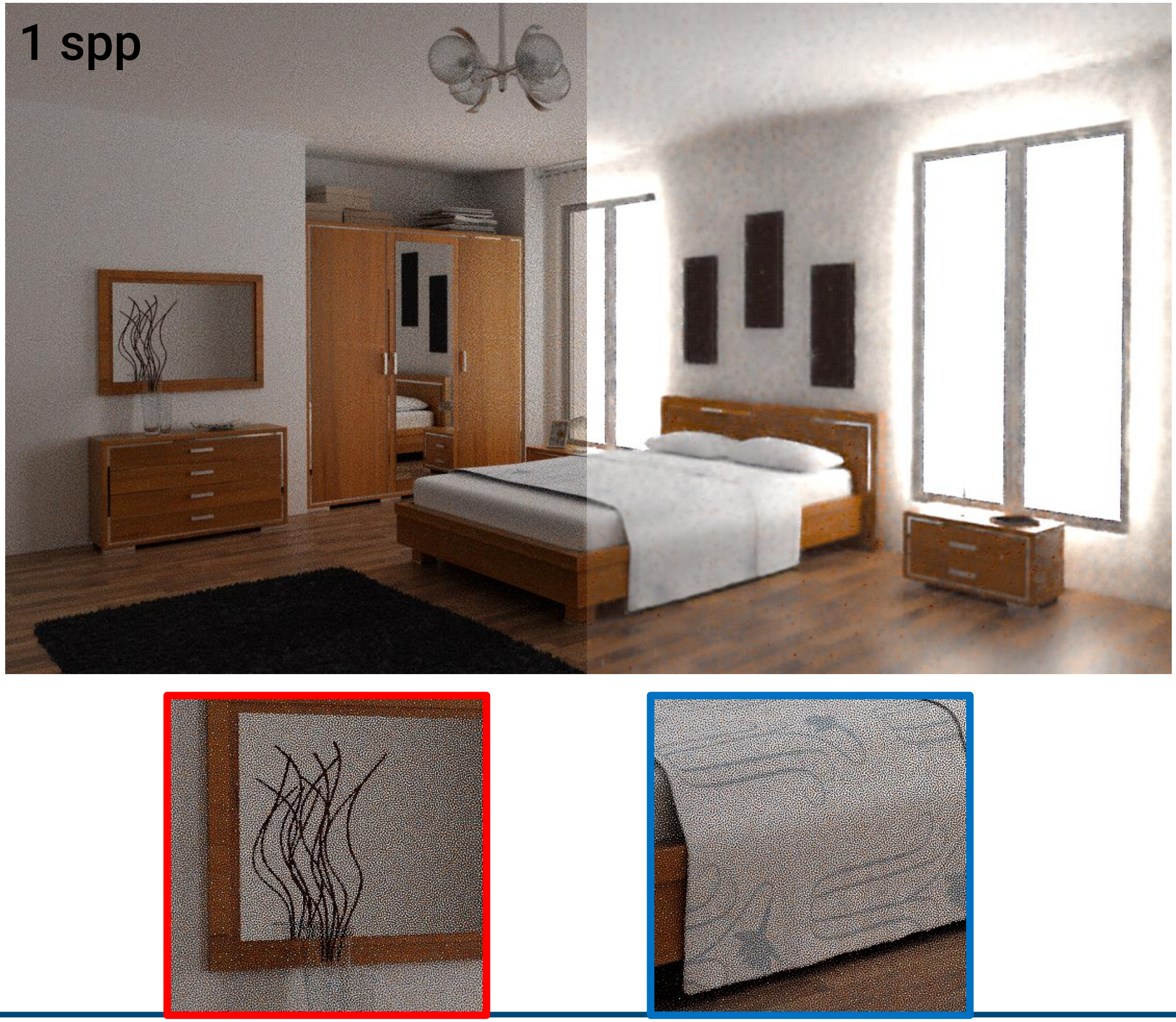
Blue Noise



*using a simple Unet

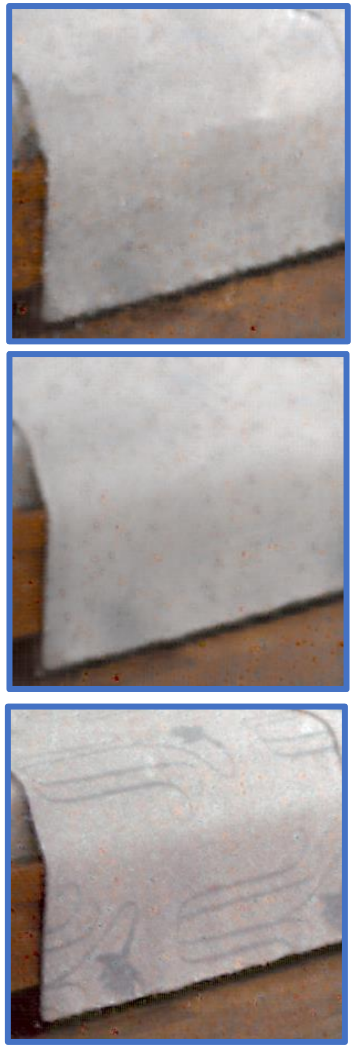
Noisy Input | Denoised Image*

Noisy Input | Denoised Image*



White Noise sampling

Blue Noise sampling



White Noise denoiser*

Blue Noise denoiser*

*simple Unet

White Noise | Blue Noise

Noisy Input | Denoised Image | Noisy Input | Denoised Image

Blue noise error and denoising

- Blue noise error distribution improve rendering
  - For real time and offline rendering
  - Adapt sampling for denoising
- Each problem need tailored algorithm
  - There is no perfect algorithm
  - Match strengths and weakness with your problem
- Low Blue noise quality can be still better than no correlation at all

Summary

What have we learned today?

Summary

- Blue noise error distribution is an important axis of improvement
  - Simply include the perception into the sampling process
  - Good coordination with denoising

- Require few conditions
  - Smooth integrand in screen space
  - Low dimension for apriori method
  - Temporal stability for a posteriori