

Fast Blue-Noise Generation via Unsupervised Learning

Daniele Giunchi

Alejandro Sztrajman

Anthony Steed

University College London

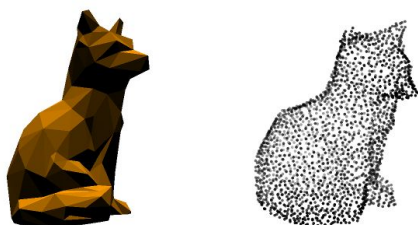
BLUE NOISE

Computer Graphics

Dithering for Images



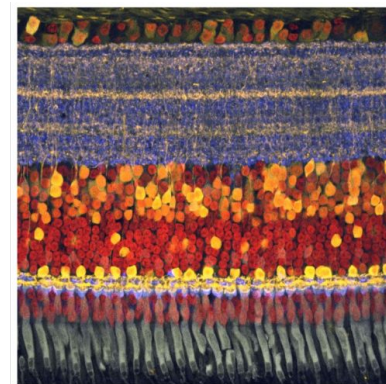
Mesh Sampling



Sampling for Raytracing



Biological models



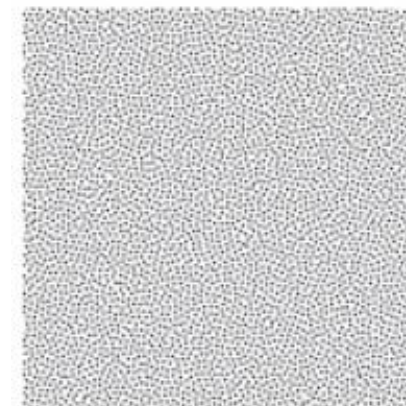
Physical Processes



Audio signals

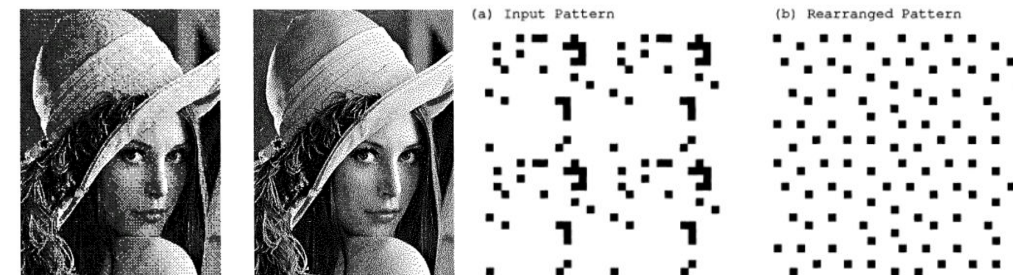


Low-discrepancy sequences

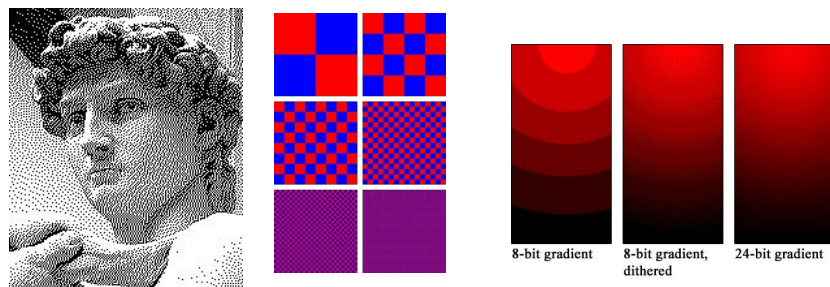


Dithering

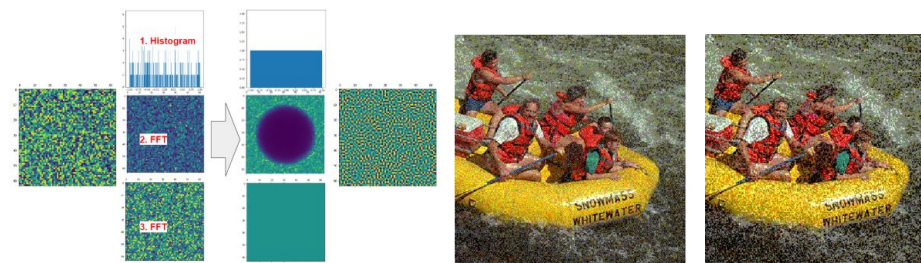
Dither is applying **intentionally noise** to a signal, to minimize distortion during the quantization process. In CG it gives also the illusion of color depth, and helps avoid color banding.



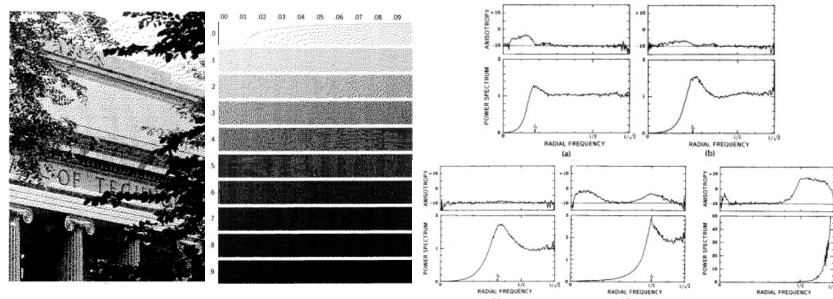
[Ulichney R.A. 1993] Void-and-cluster method for dither array generation



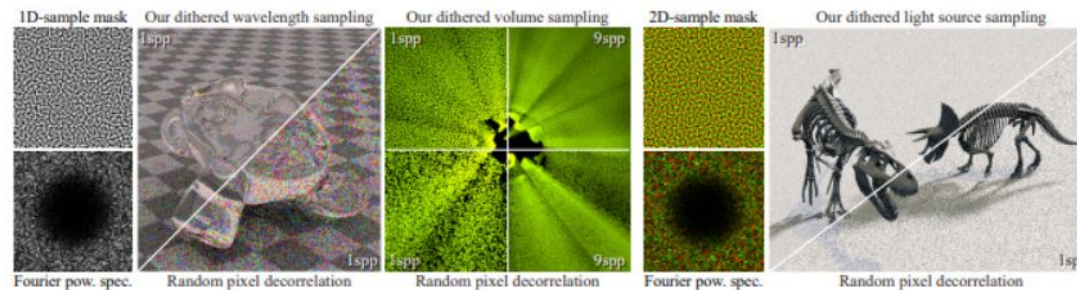
Images taken Courtesy of Wikipedia



[Wronski B. 2020] Optimizing blue noise dithering: backpropagation through Fourier transform and sorting.



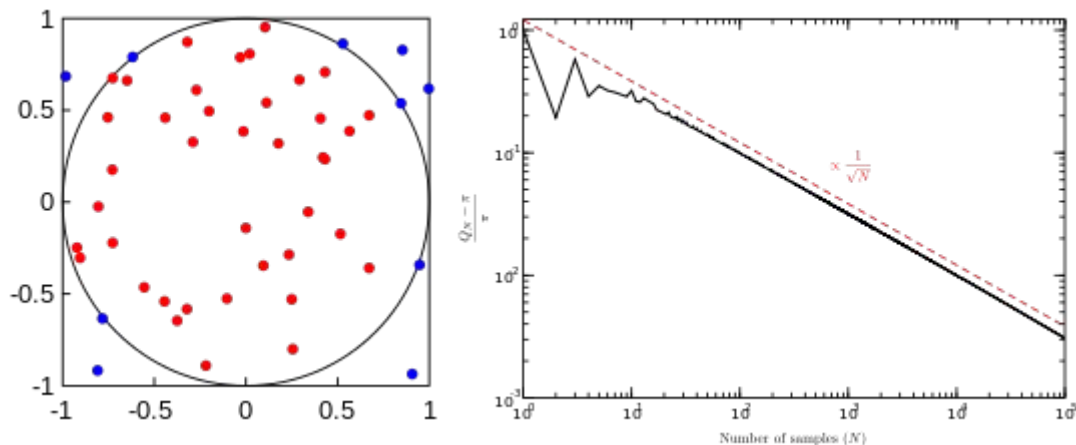
[Ulichney R.A. 1988] Dithering with blue noise.



[Georgiev I et al. 2016] Blue-noise dithered sampling

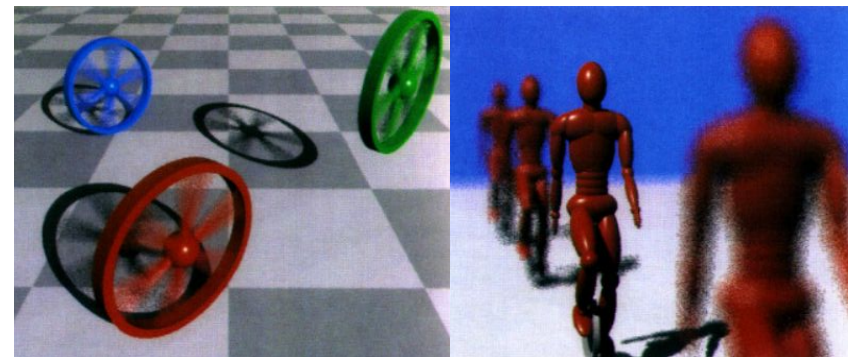
Monte Carlo Integration

MCI is an integration technique that evaluate an integral by randomly **sampling** the domain of the function to integrate. There are different ways to sample and such method is use with higher dimensional integrals.

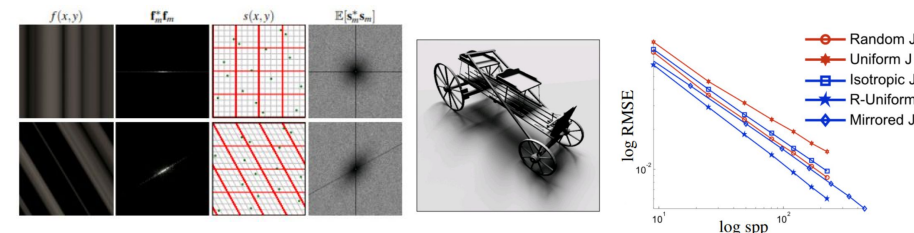


Images taken Courtesy of Wikipedia

Montecarlo sampling



[Mithcell P.D. 1991] Spectrally Optimal Sampling for Distribution Ray Tracing. Computer Graphics

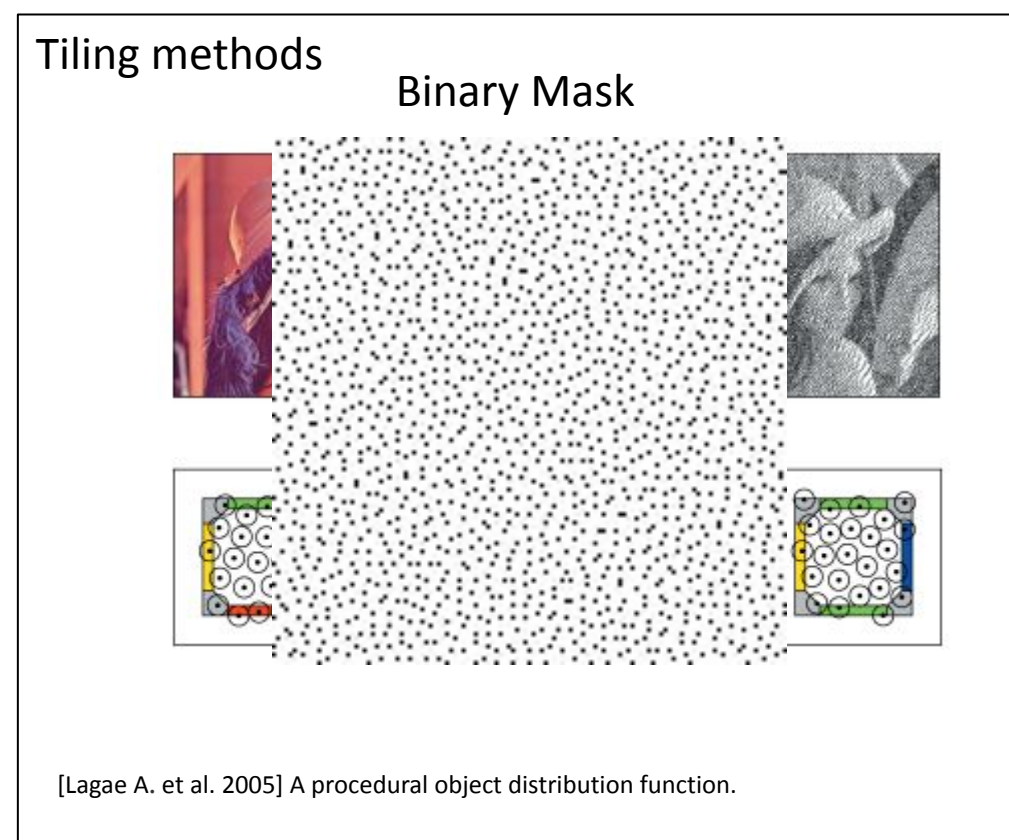
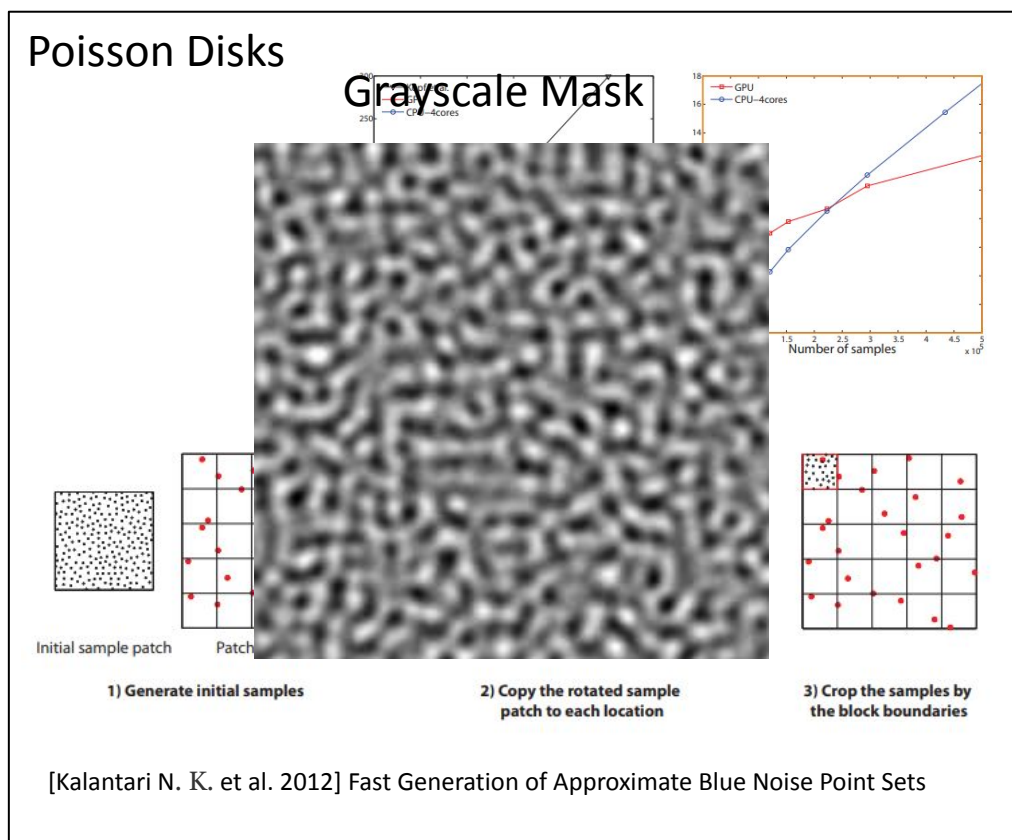


[Singh G. et al. 2014] Analysis of sample correlations for Monte Carlo rendering

Motivation

Current Methods suffer one of these two issues:

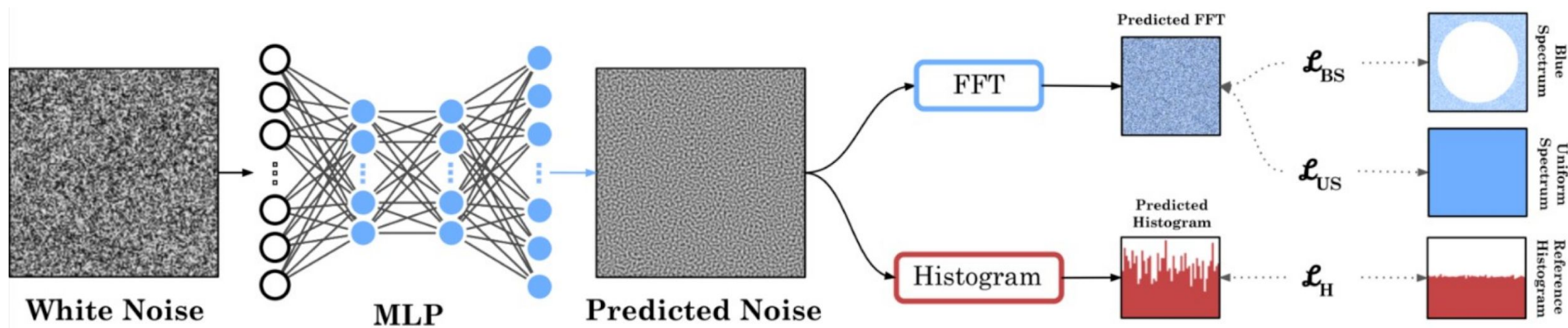
- Slow computation but high accuracy
- Fast computation but low accuracy or presence of artifacts



Our Approach

- Create a **Neural Network** that can compute fast the blue noise, acting a blue noise generator
- Define an input from which to start: **White Noise** (uniform random generator)
- Define a Loss that is the joint contribution of 3 different losses
- Test on 2 typical computer graphics applications
Dithering and Monte Carlo Integration

Architecture & Loss Function



- Shallow architecture
- Generated output is modelled by 3 different losses modelled from [Wronski 2020] analysis on blue noise:
 - Blue Spectrum loss
 - Uniform Spectrum loss
 - Histogram loss

LOSS in detail

Blue Spectrum Loss:

$$\mathcal{L}_{\text{BS}} = \sum_{ij} \hat{\phi}_{ij} \left[\max \left(0, \frac{\omega_{\text{cutoff}} - r_{ij}^2}{\omega_{\text{cutoff}}} \right) \right]^2$$

Uniform Spectrum Loss:

$$\mathcal{L}_{\text{US}} = \sum_{ij} \left[\nabla^2 \hat{\phi}_{ij} \right]^2 + \nabla_i \hat{\phi}_{ij} + \nabla_j \hat{\phi}_{ij}$$

Histogram Loss:

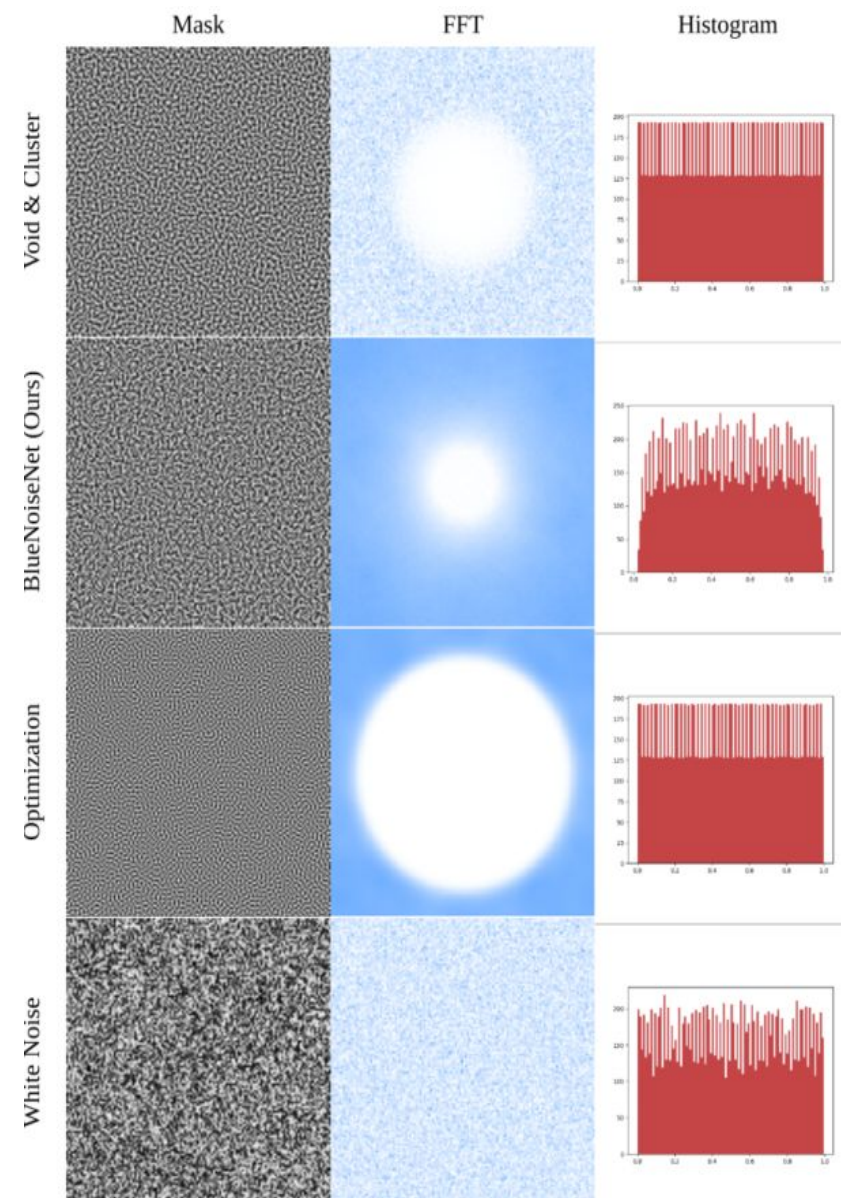
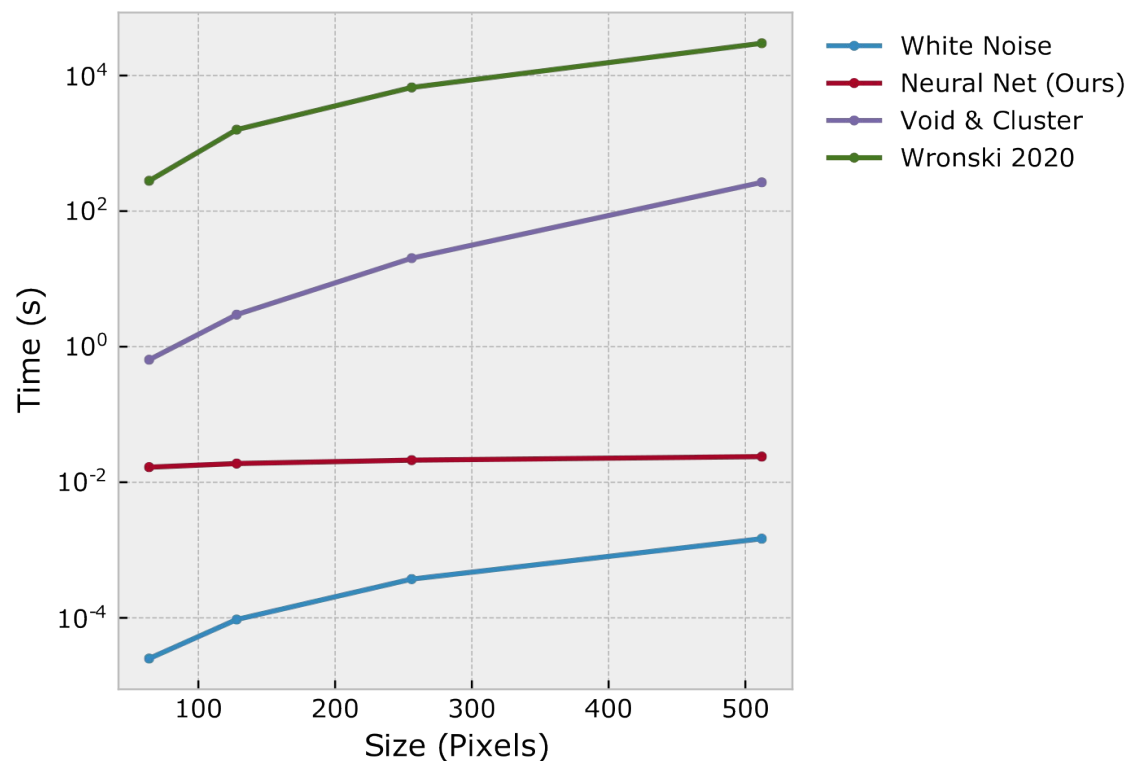
$$\mathcal{L}_{\text{H}} = \sum_k [\text{Histogram}(\hat{s})_k - \text{RefH}_k]^2$$

TRAINING

- Training is unsupervised:
 - White noise is self-generated
 - Losses works until accuracy reaches the plateau
- We opted 128x128 images but different sizes can be used
- Epoch: 12
- Number of generated samples is 32000

RESULTS

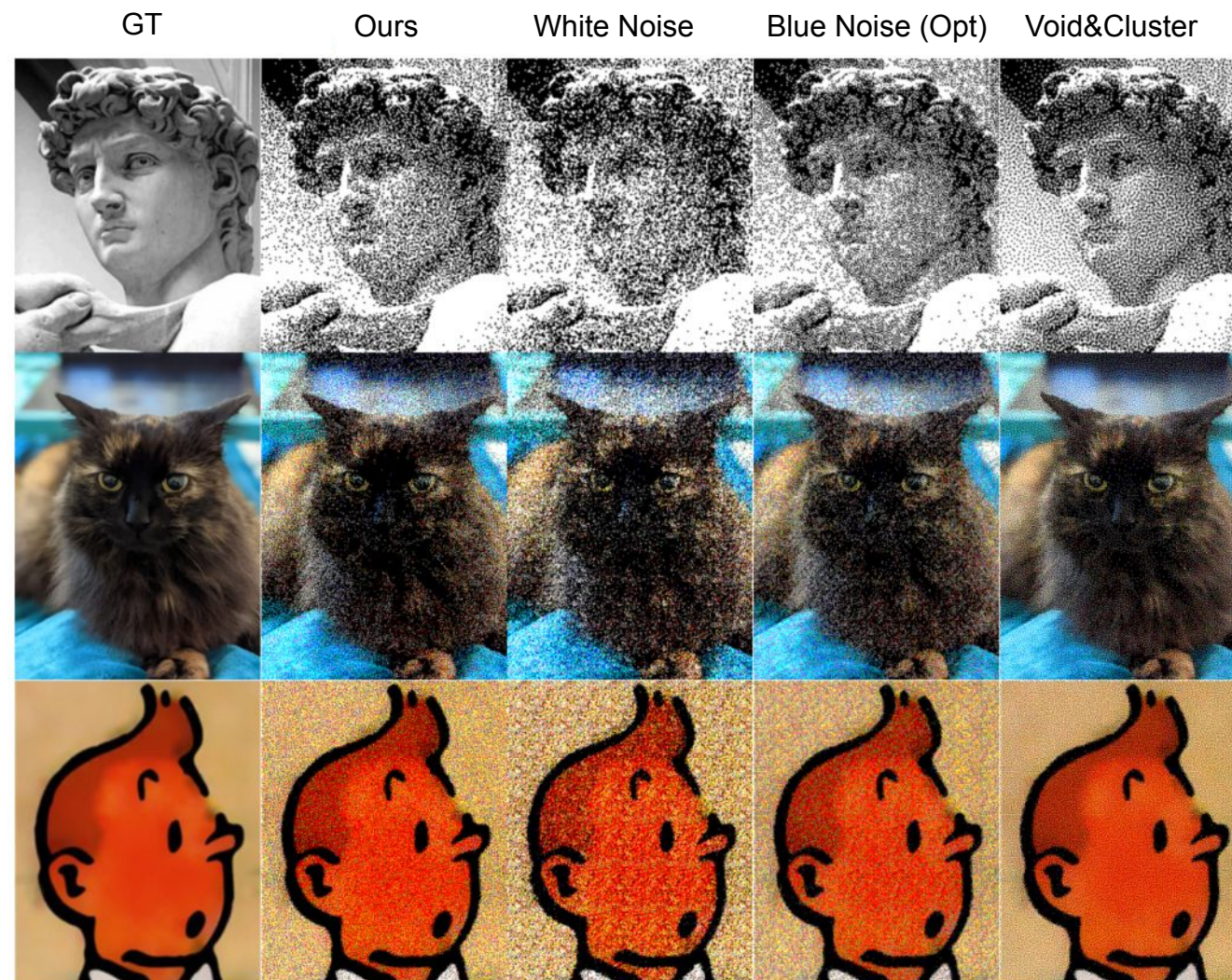
- Compare with Void and Cluster, original Wronski, and White Noise
- Applied to two different problem in CG: dithering and Montecarlo sampling for raytracing.



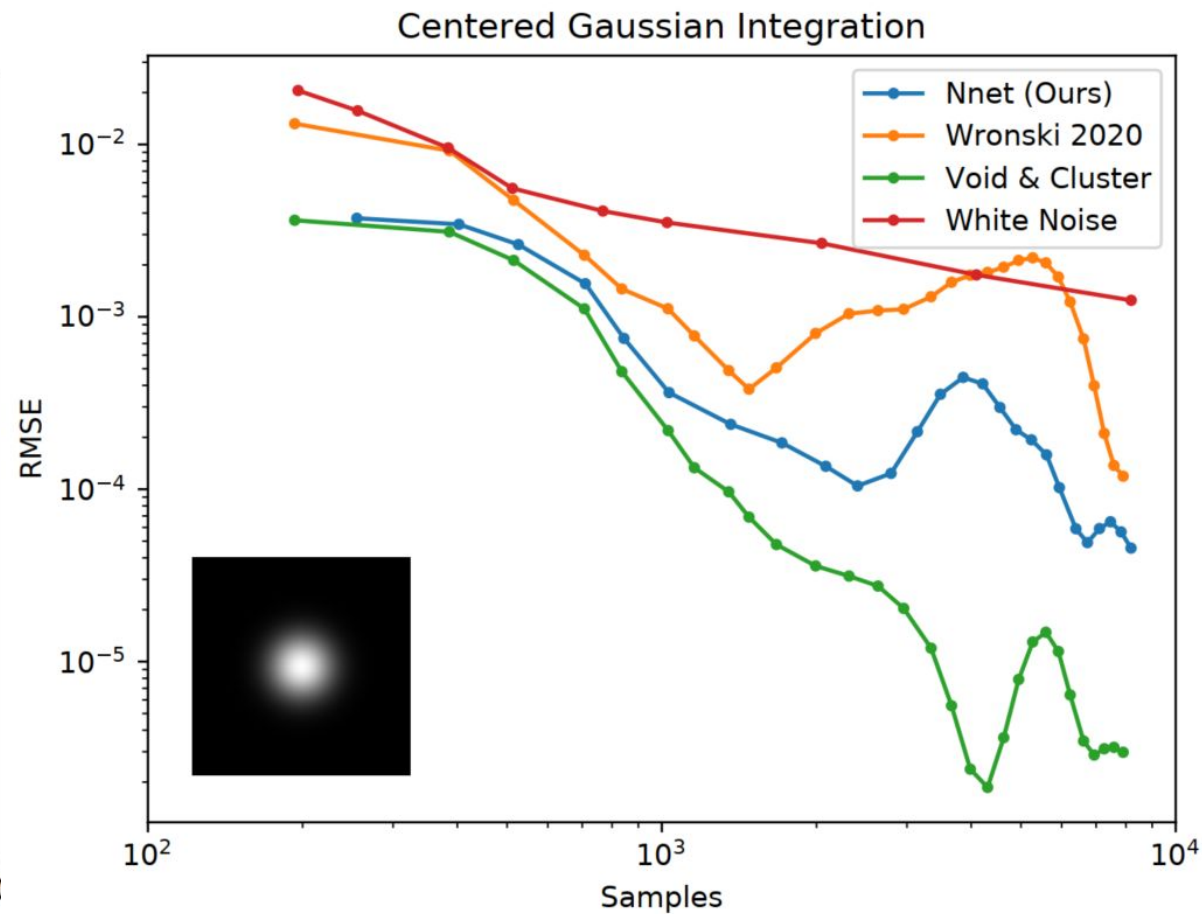
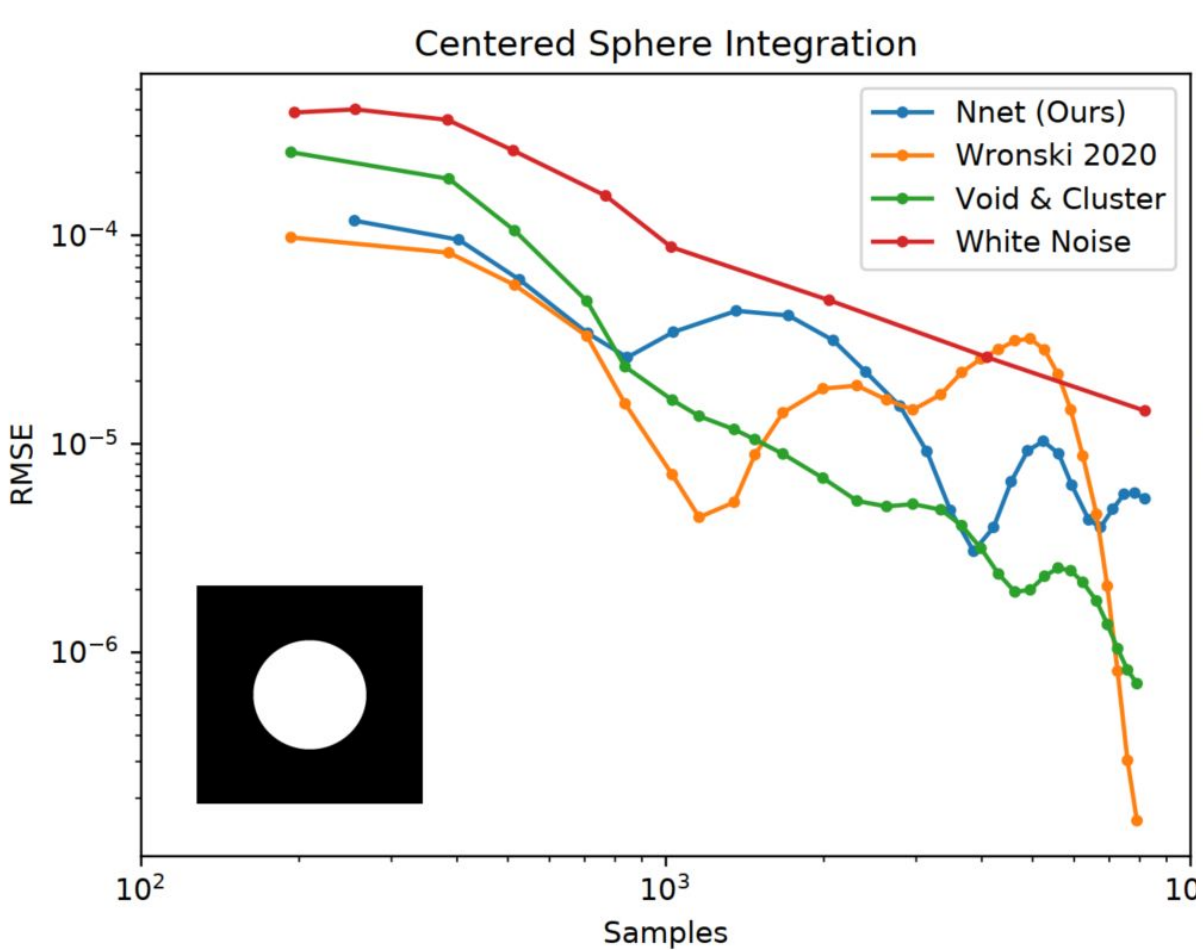
DISCUSSIONS - DITHERING

- Less accurate but faster than void and cluster
- Equal than Wronski but faster
- Naturally better than White Noise

Method	RMSE
White Noise	0.069 ± 0.002
Neural Network (Ours)	0.046 ± 0.001
Wronski 2020	0.046 ± 0.001
Void & Cluster	0.029 ± 0.001



DISCUSSION – MC INTEGRATION



Future Work

Improvements

- Leveraging Void&Cluster method that is the current state of the art for accuracy in Blue Noise generation
- Improve Loss function embedding directly application such as dithering.
- Generative Adversarial Network for BN Generation

Follow ups on Generalisation

- Improve dimensionality
- Generation of a BN independent by resolution
- Extend to different noise colors

Conclusion

- Analyzed advantages and disadvantages of current Blue Noise algorithms
- Proposed a neural network that try to mitigate the issues on accuracy of fast methods and time consumption of accurate methods
- Tested in two of the most relevant applications in computer graphics achieving improvement of velocity and a reasonable accuracy if compared with the most precise (but slow) methods.

THANK YOU

 <https://github.com/dgiunchi/bluenoise.git>

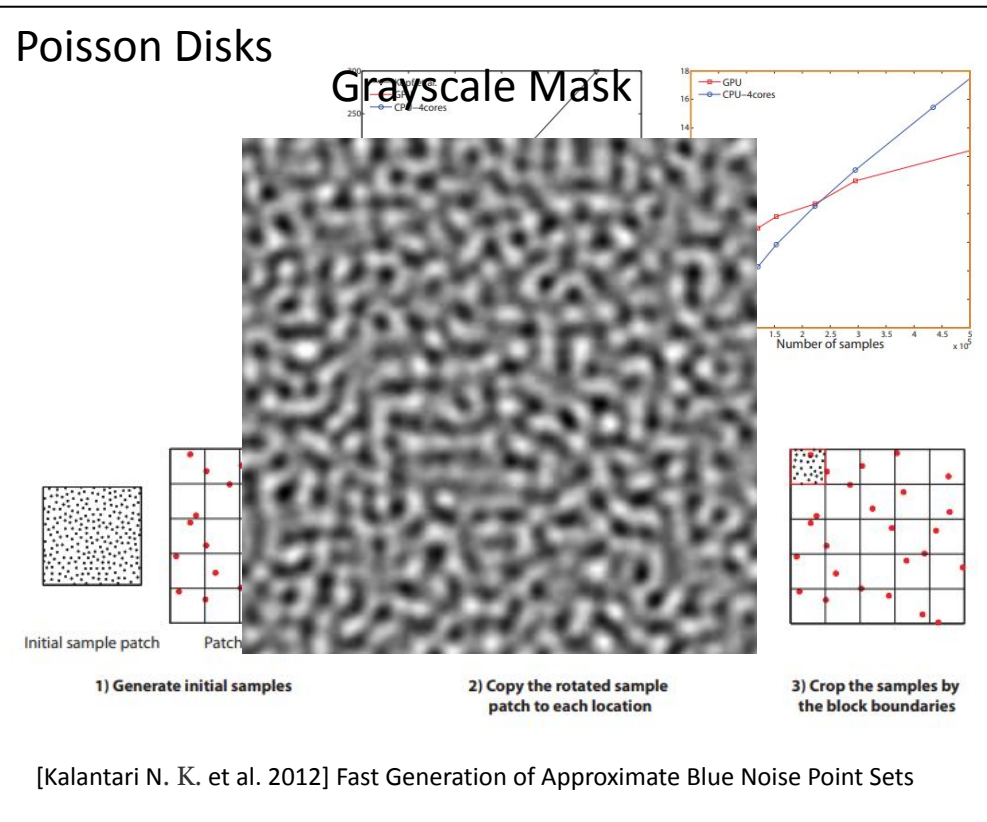
Motivation

Current Methods suffer one of these two issues:

- Slow computation but high accuracy
- Fast computation but low accuracy or presence of artifacts

Poisson Disks

Grayscale Mask

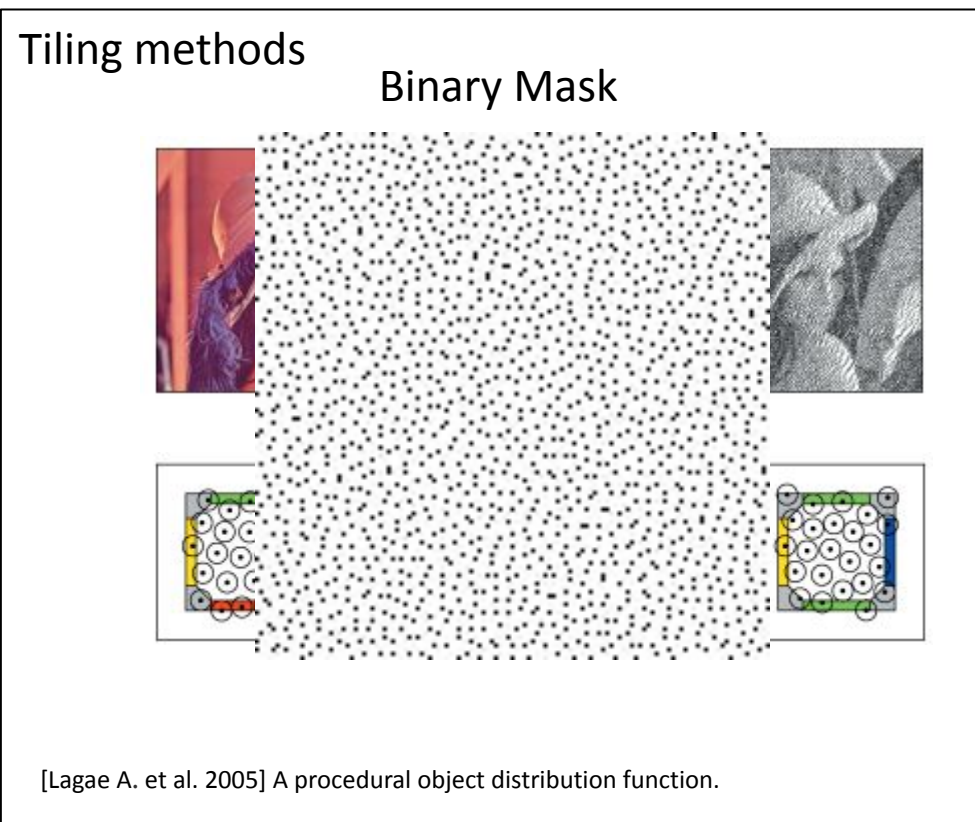


1) Generate initial samples 2) Copy the rotated sample patch to each location 3) Crop the samples by the block boundaries

[Kalantari N. K. et al. 2012] Fast Generation of Approximate Blue Noise Point Sets

Tiling methods

Binary Mask



[Lagae A. et al. 2005] A procedural object distribution function.