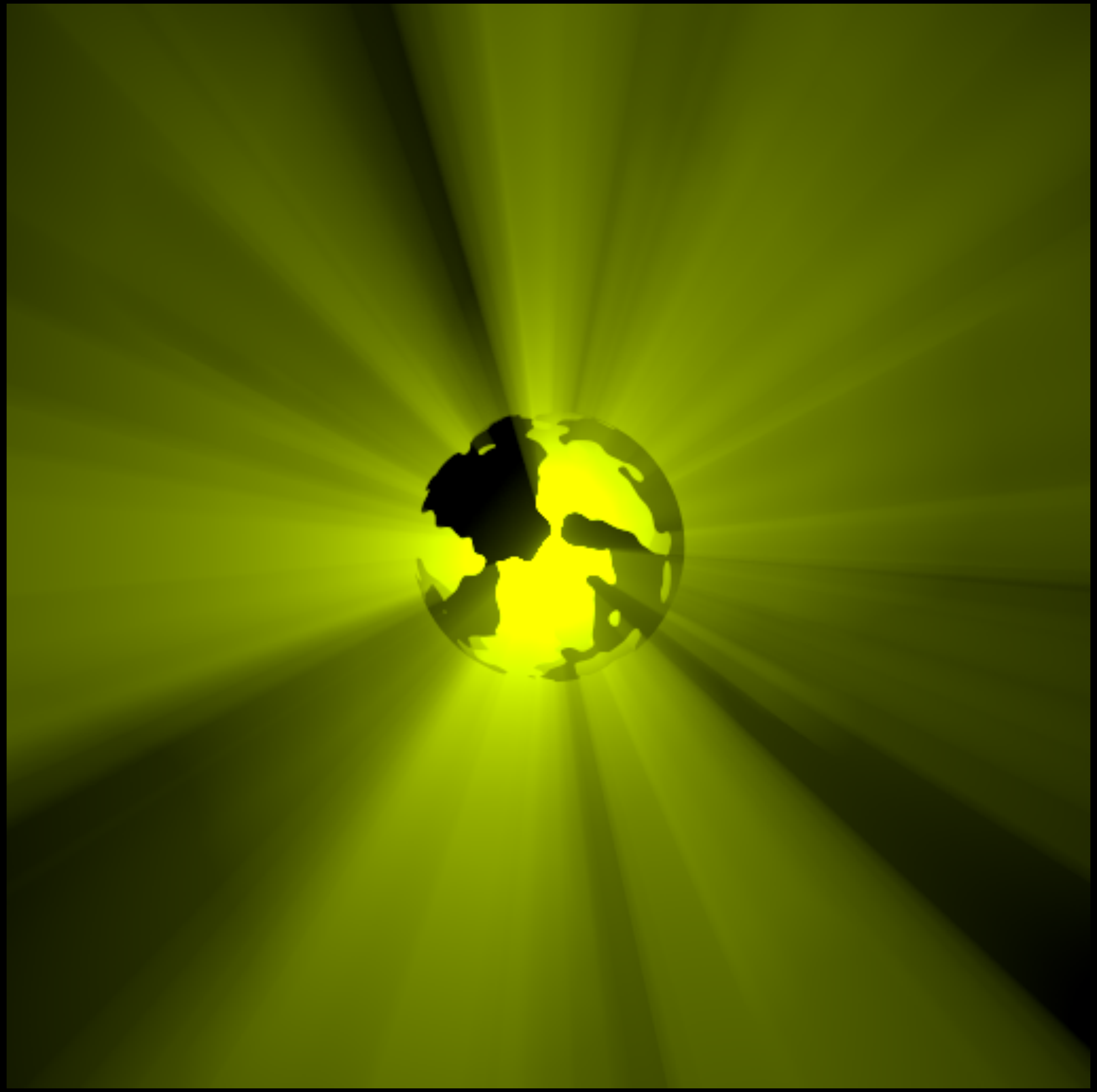
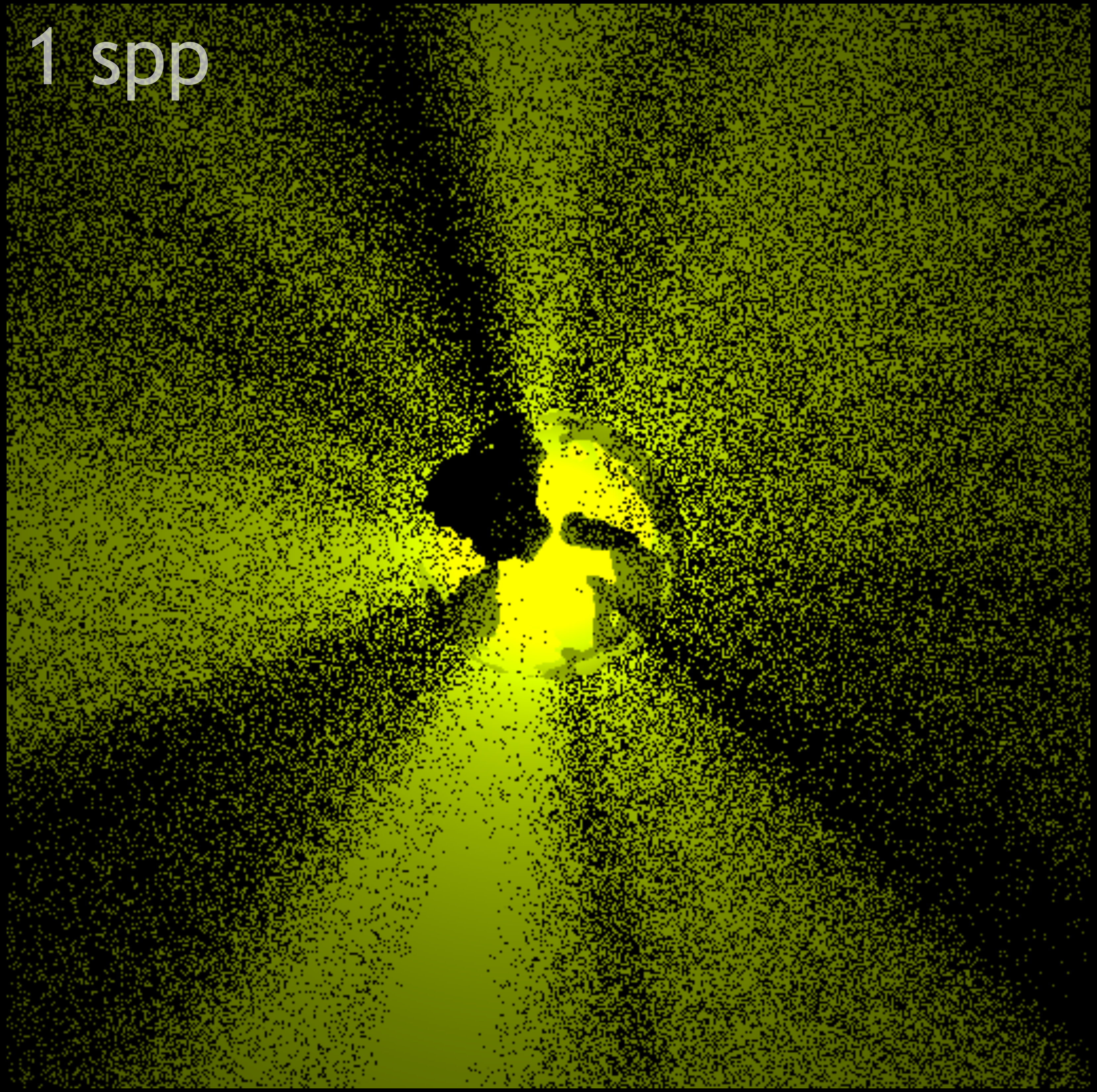
Blue-noise pixel error dithering

Iliyan Georgiev
Autodesk

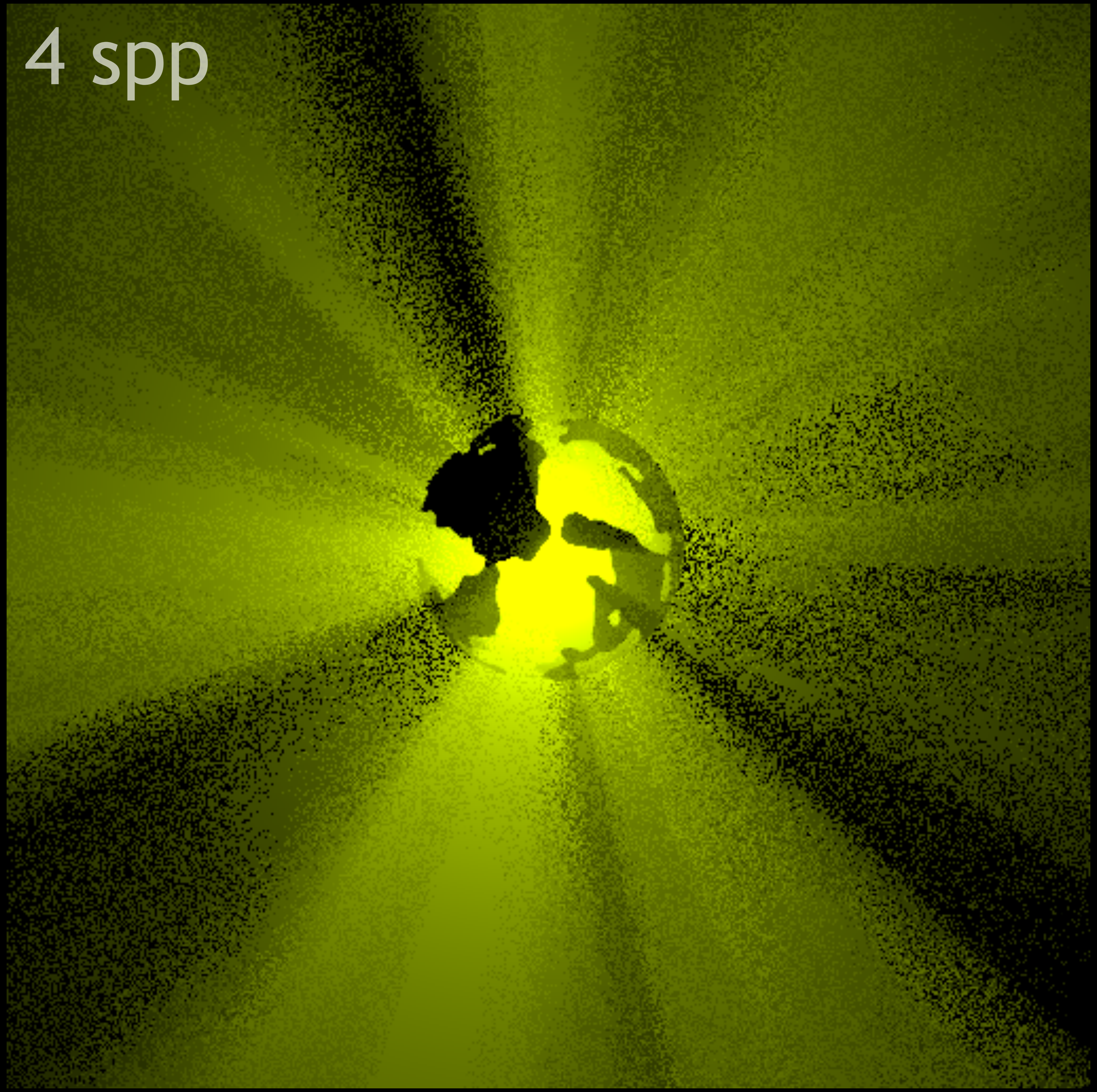
Motivation



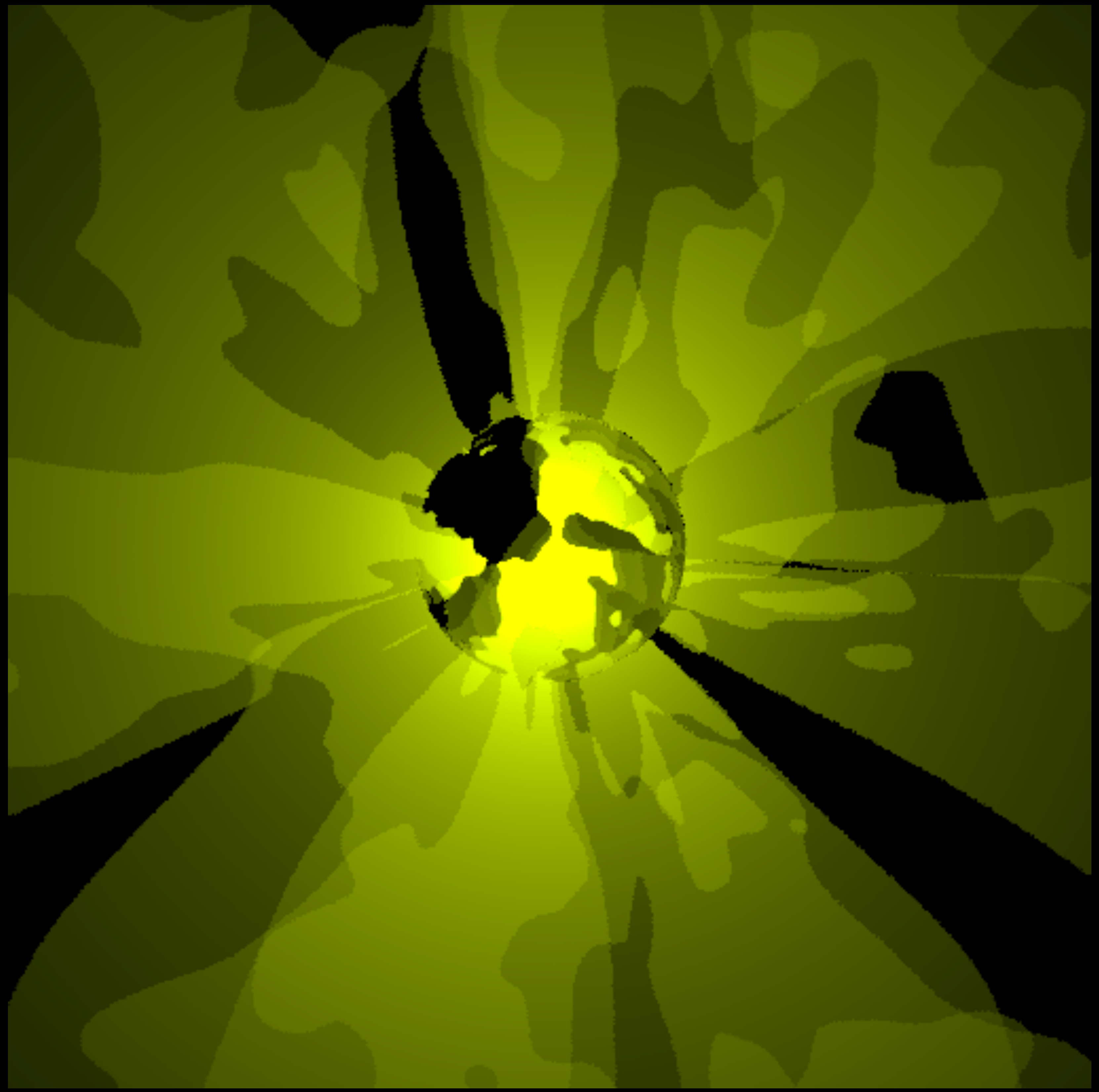
Motivation

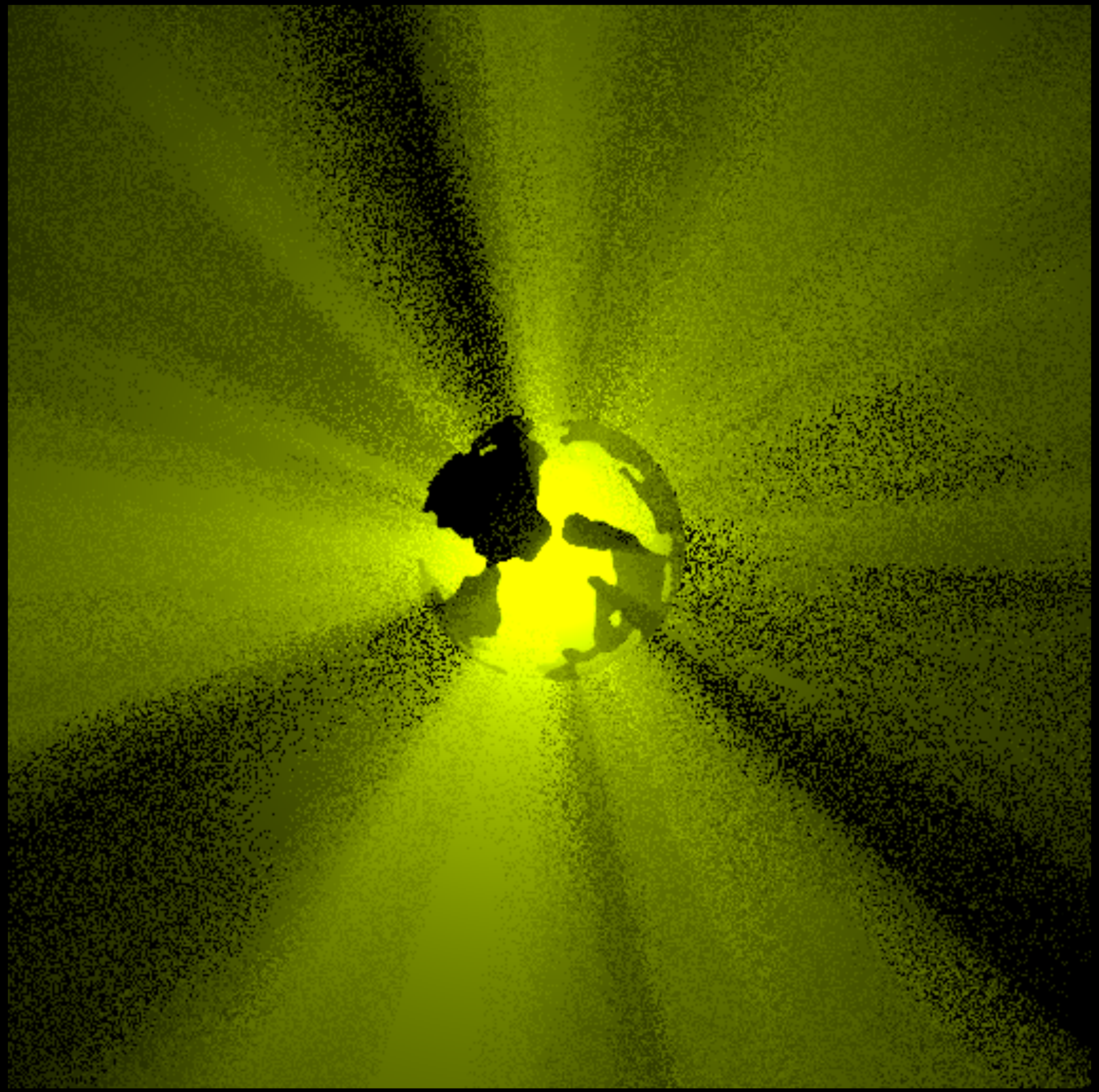


Motivation



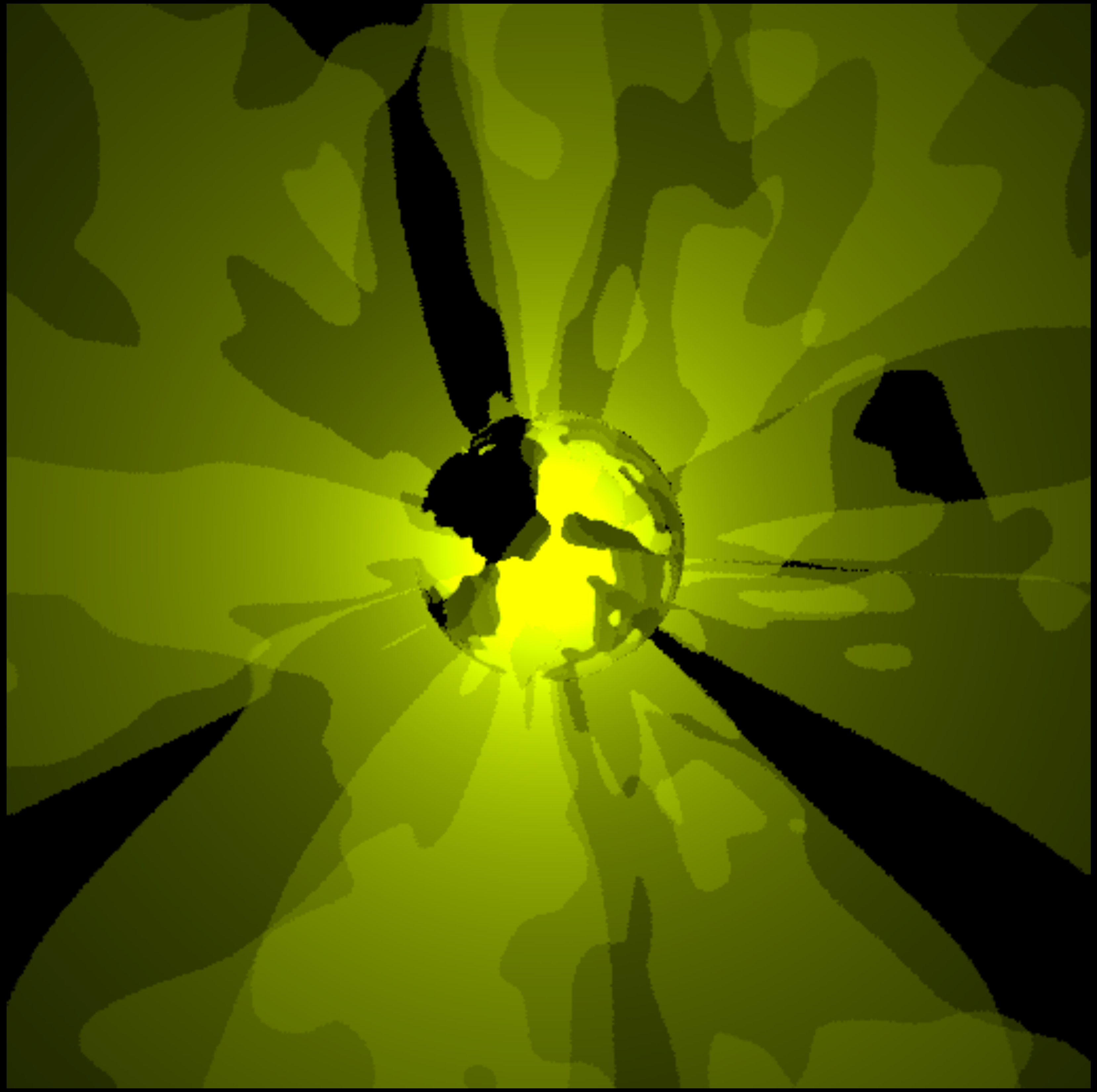
Motivation


correlated sampling


uncorrelated sampling

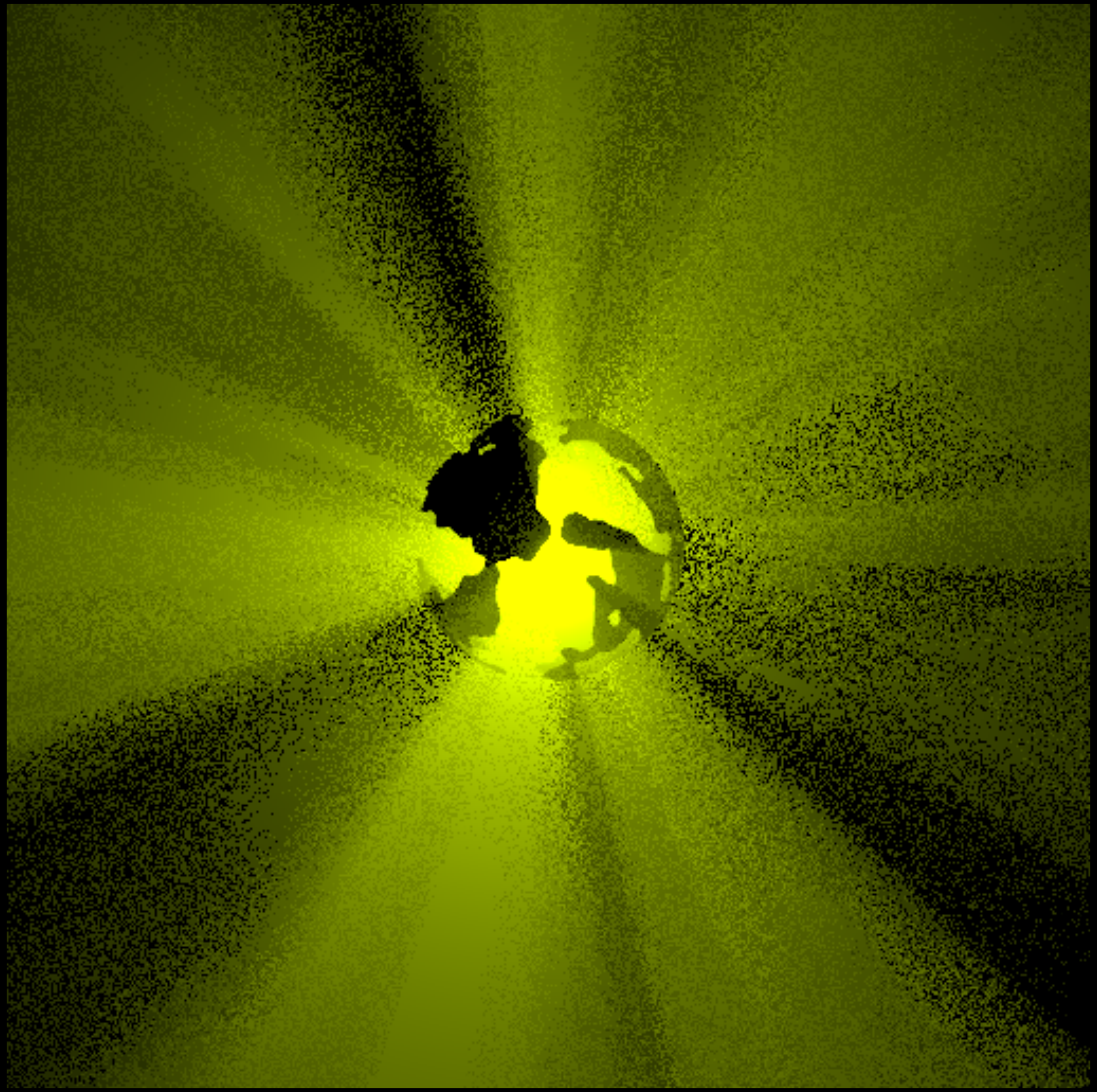
?

Motivation

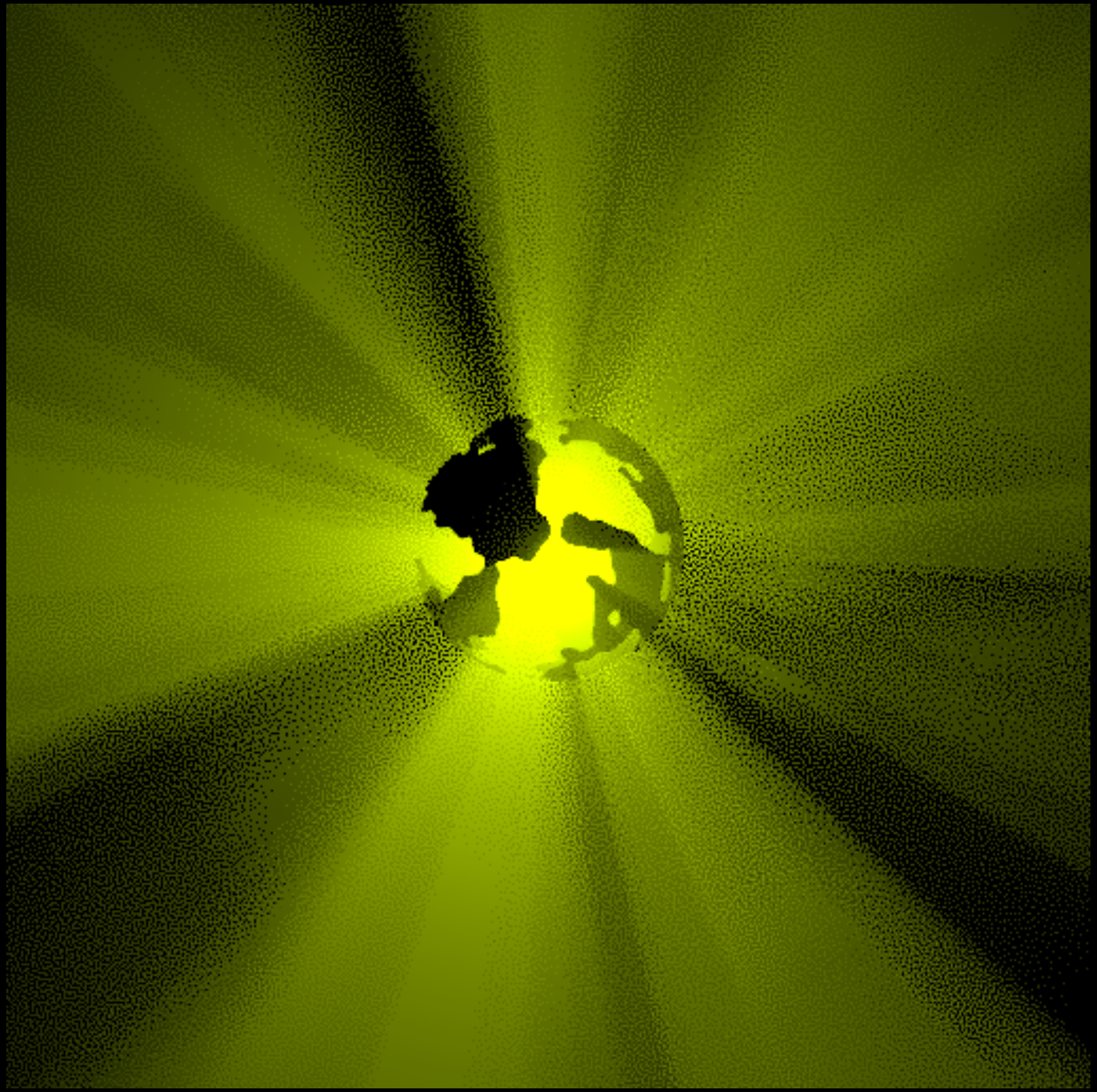


correlated sampling

positively



uncorrelated sampling

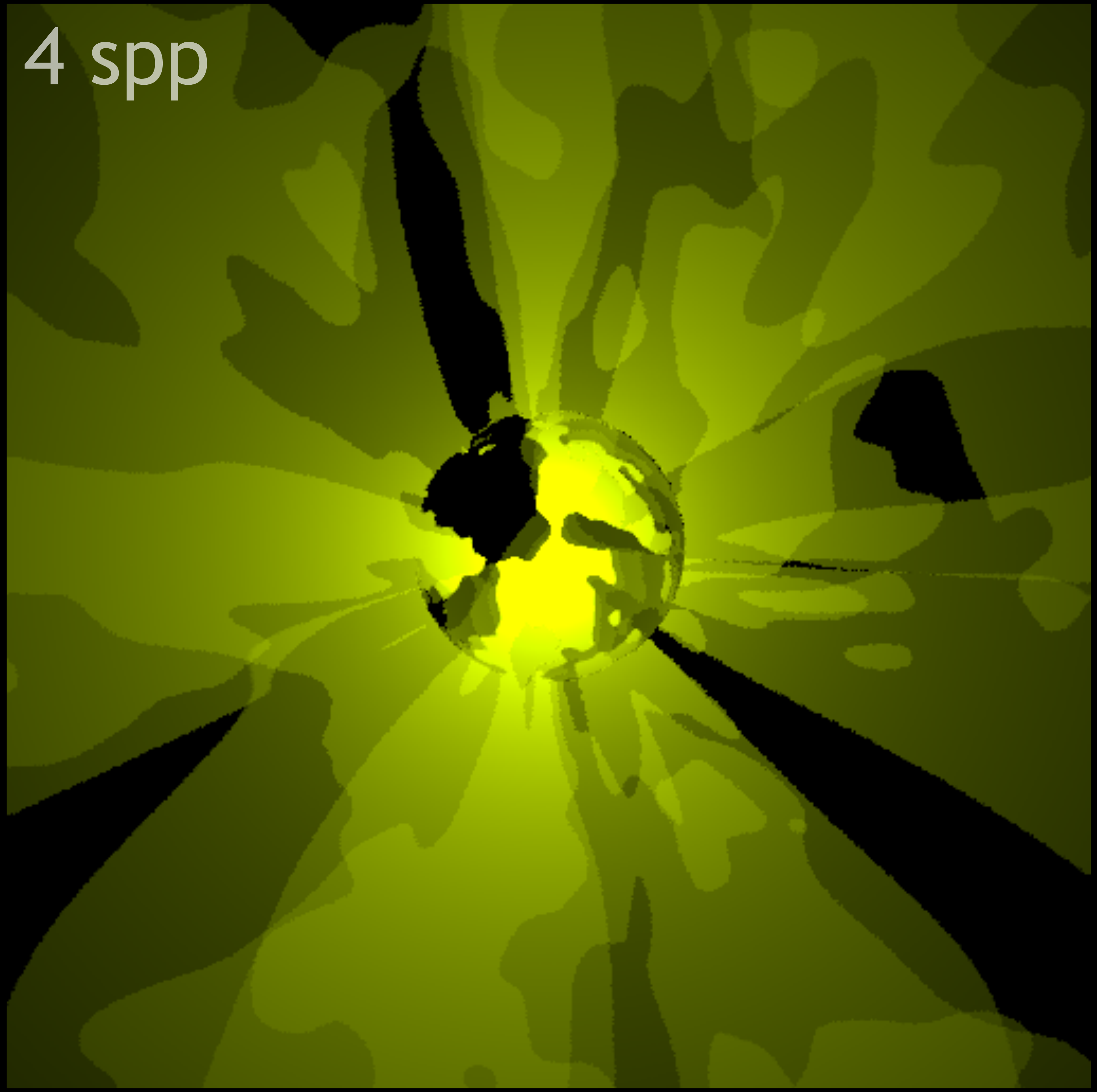


correlated sampling

negatively

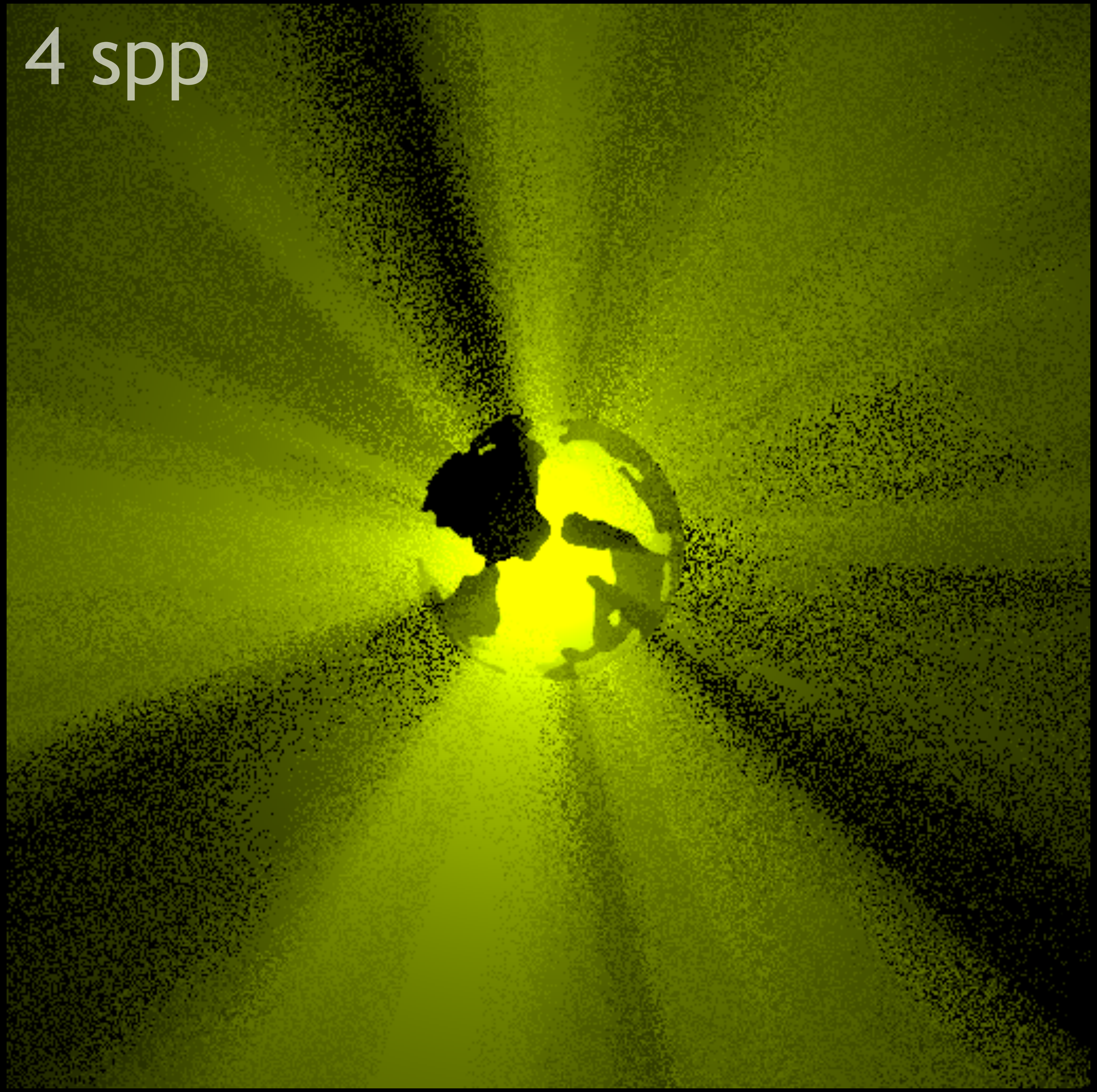
Motivation

equal error

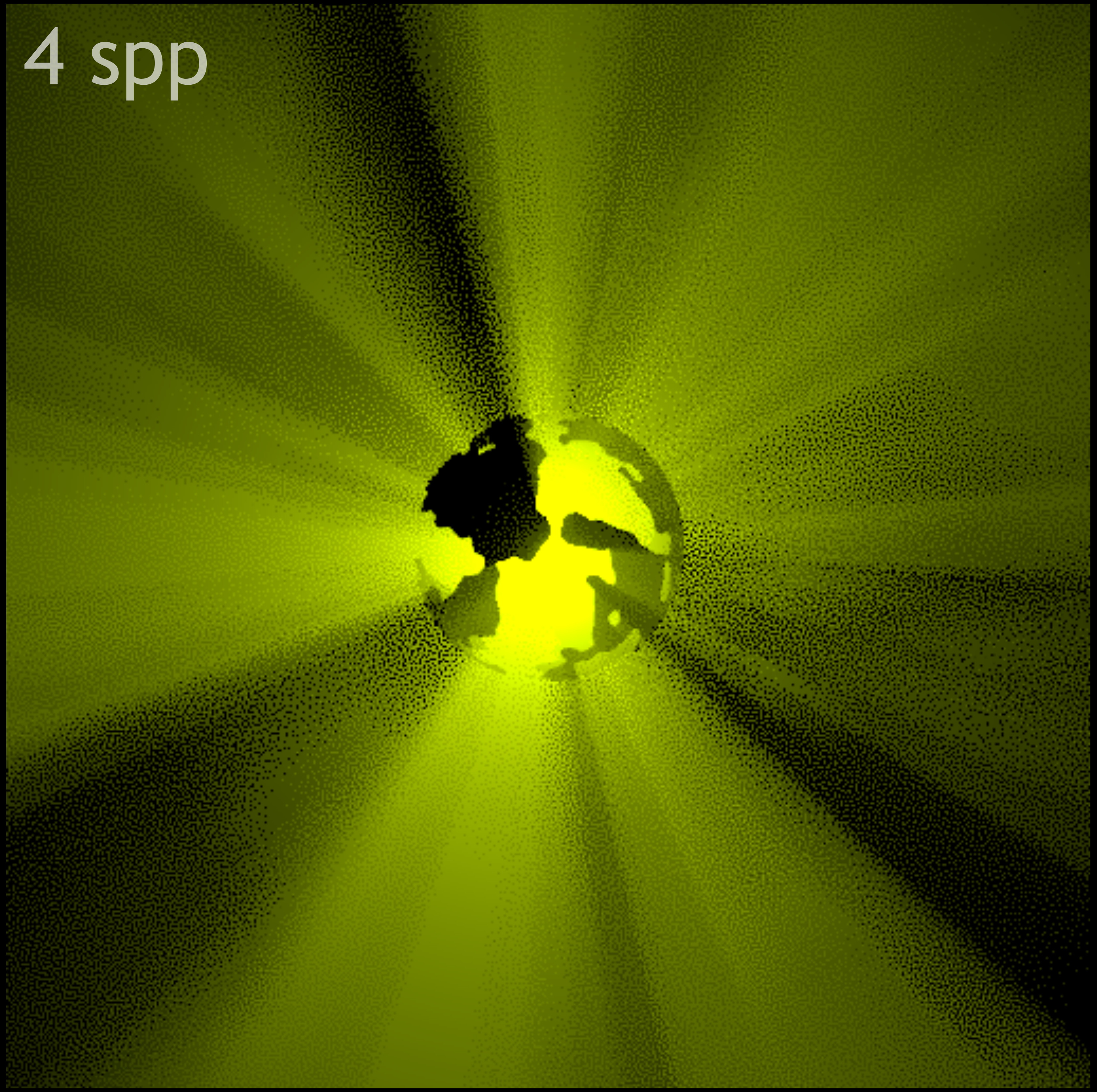


correlated sampling

positively



uncorrelated sampling

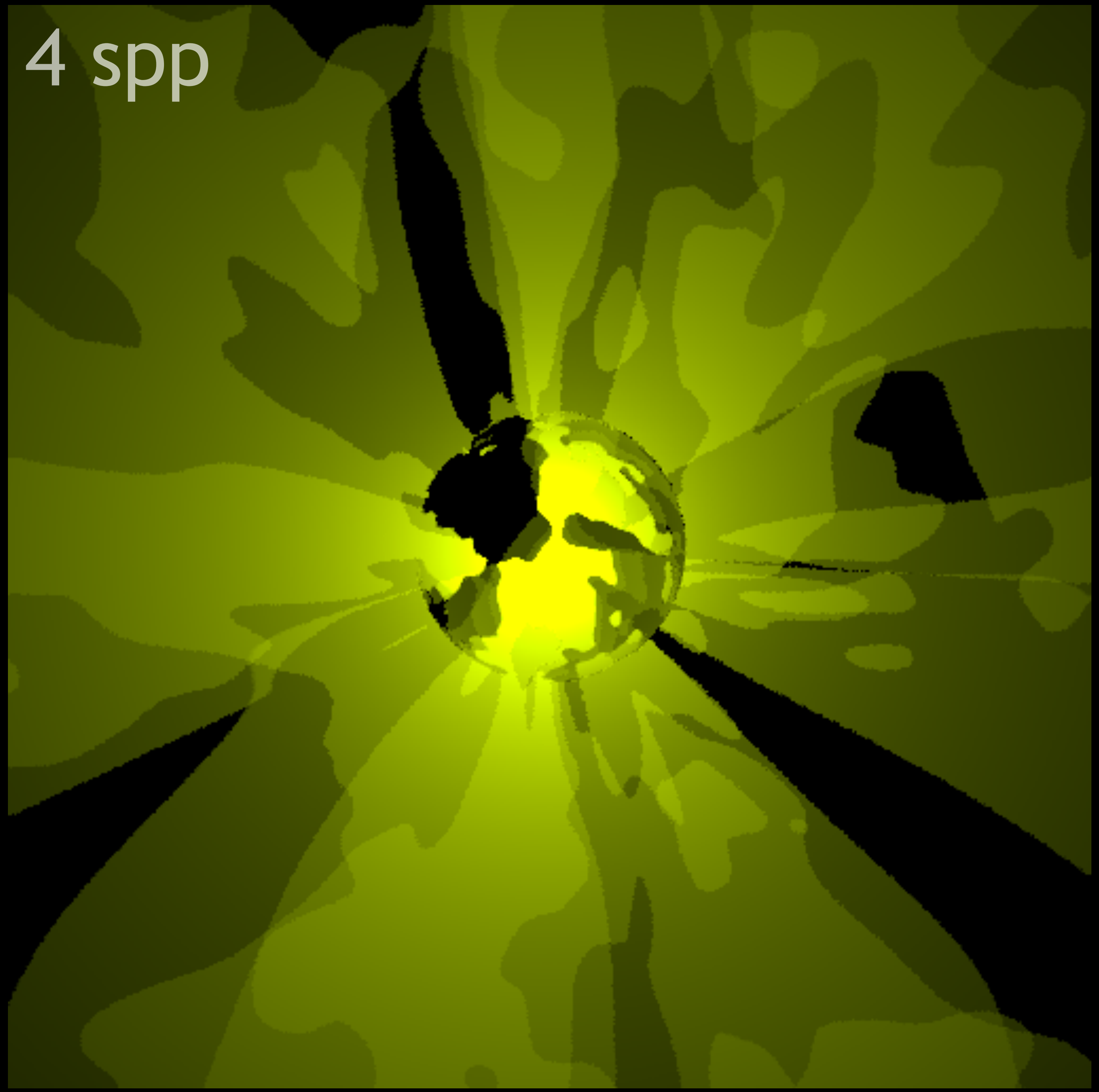


correlated sampling

negatively

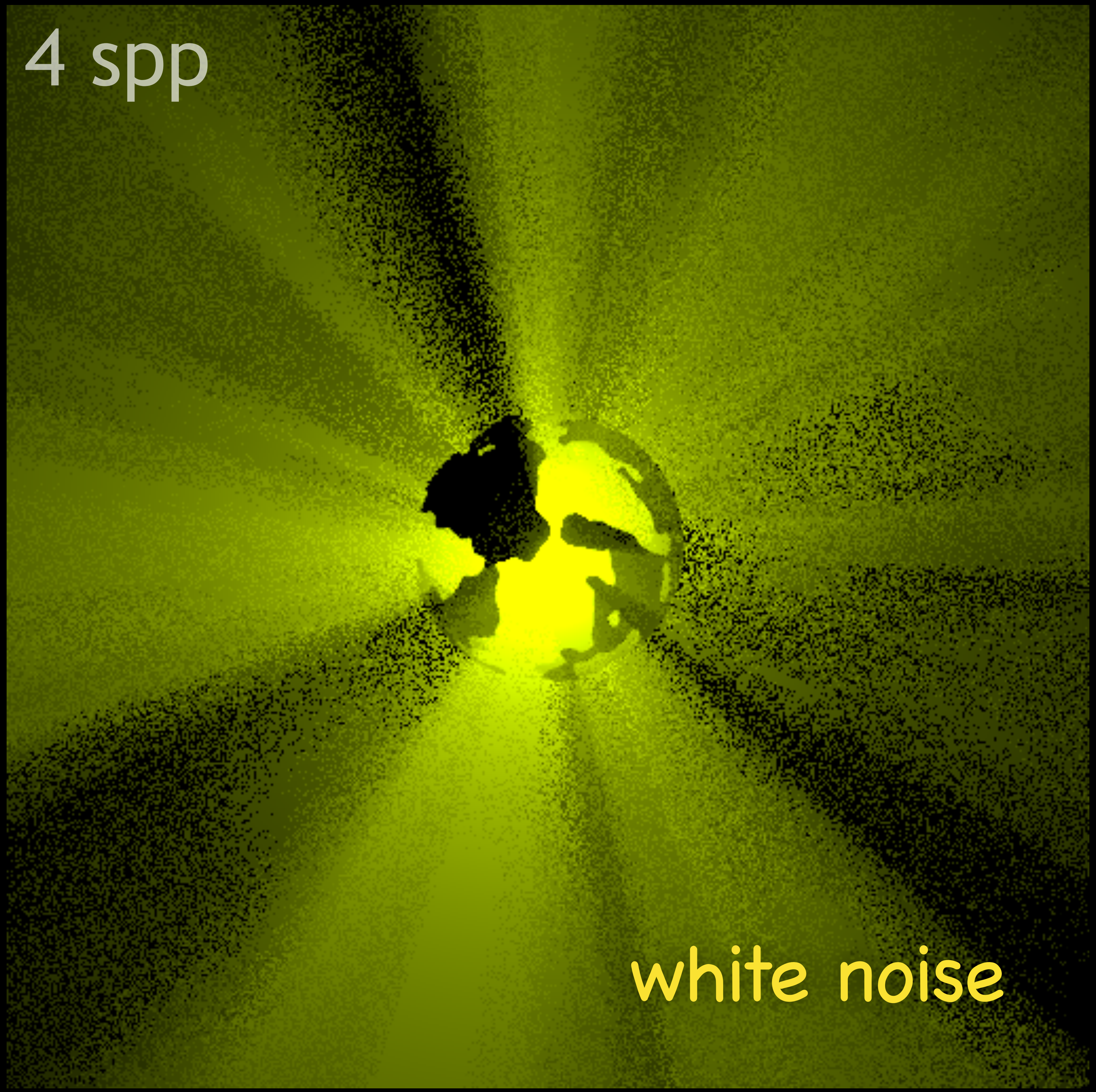
Motivation

equal error

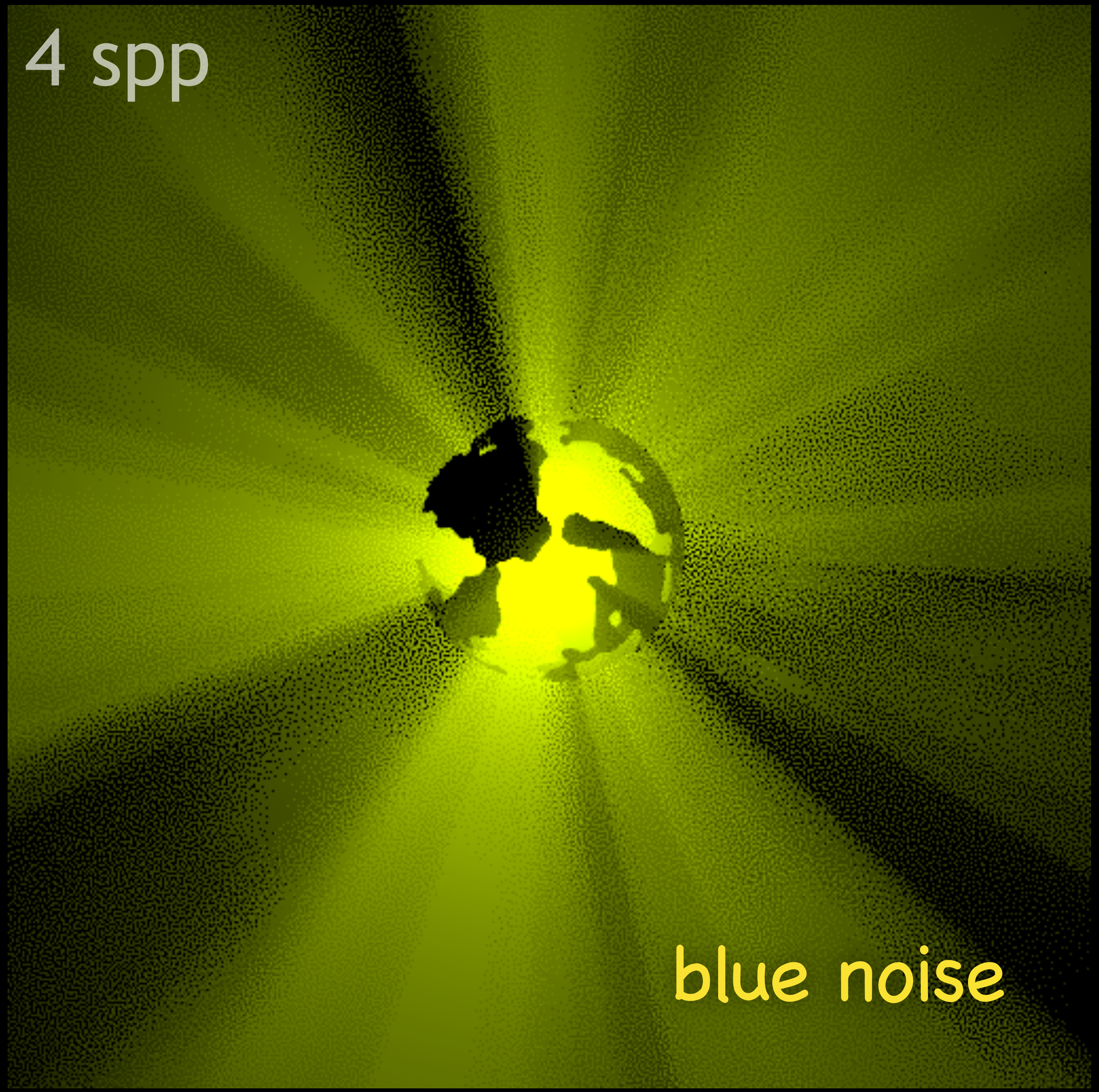


correlated sampling

positively



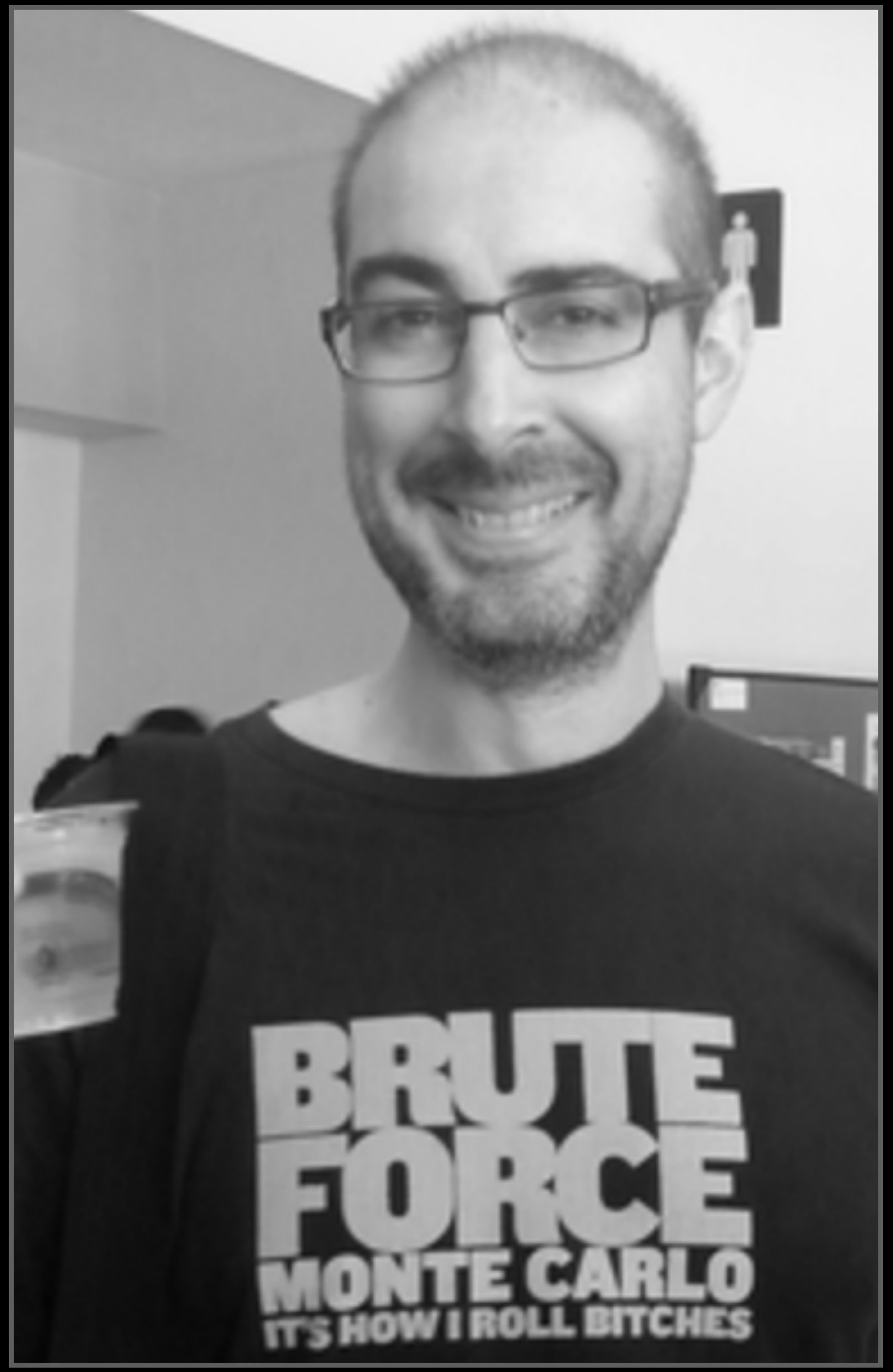
uncorrelated sampling



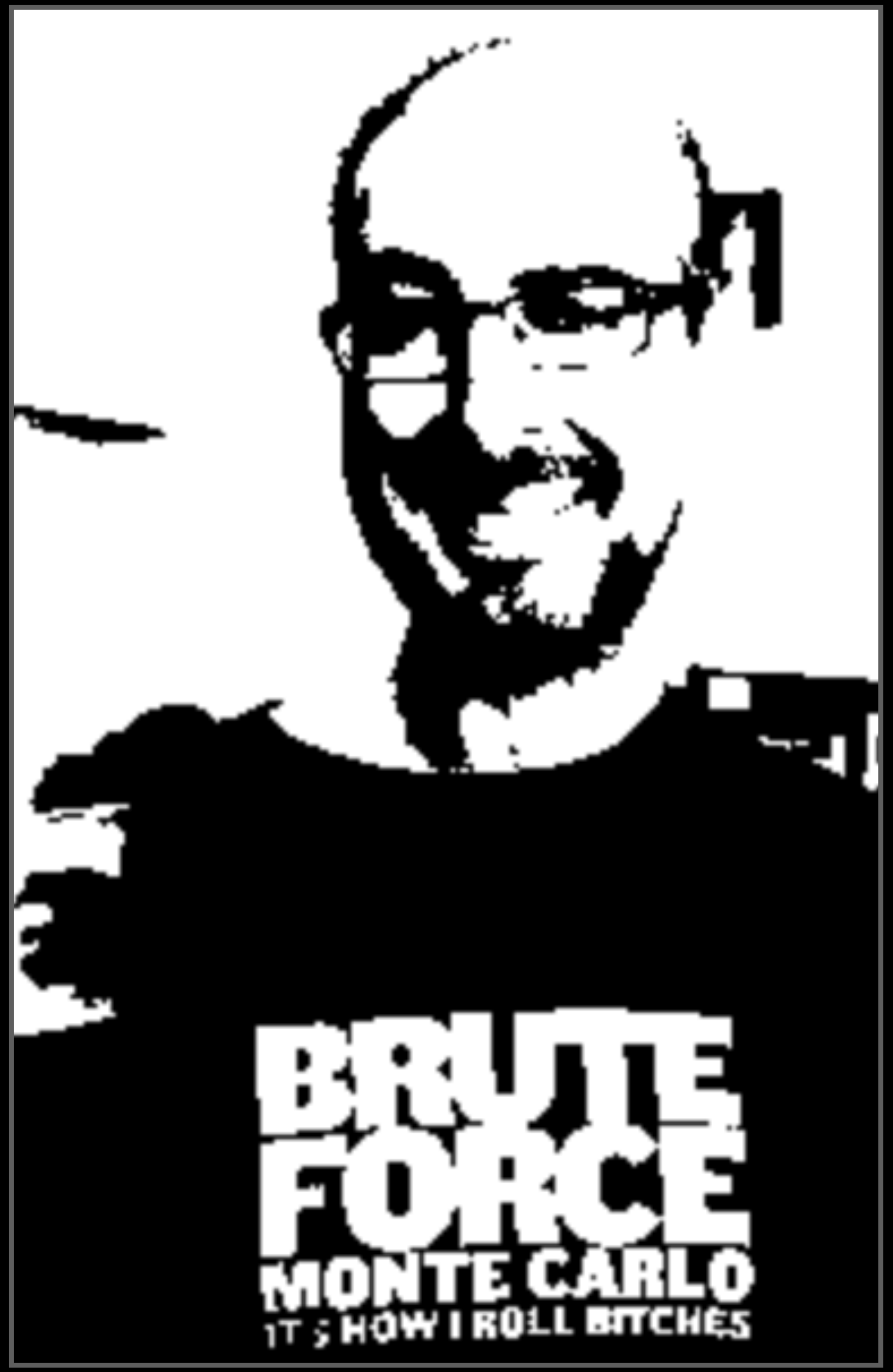
correlated sampling

negatively

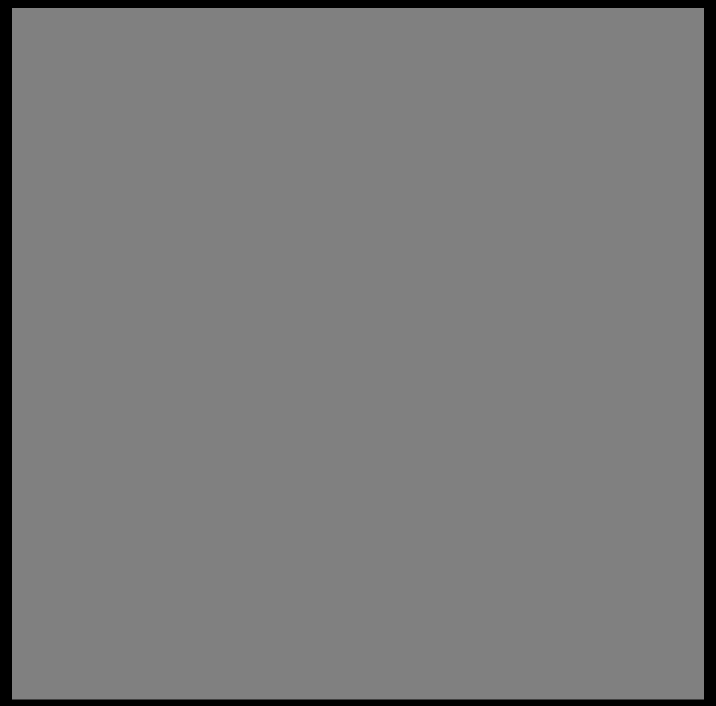
Halftoning: image thresholding

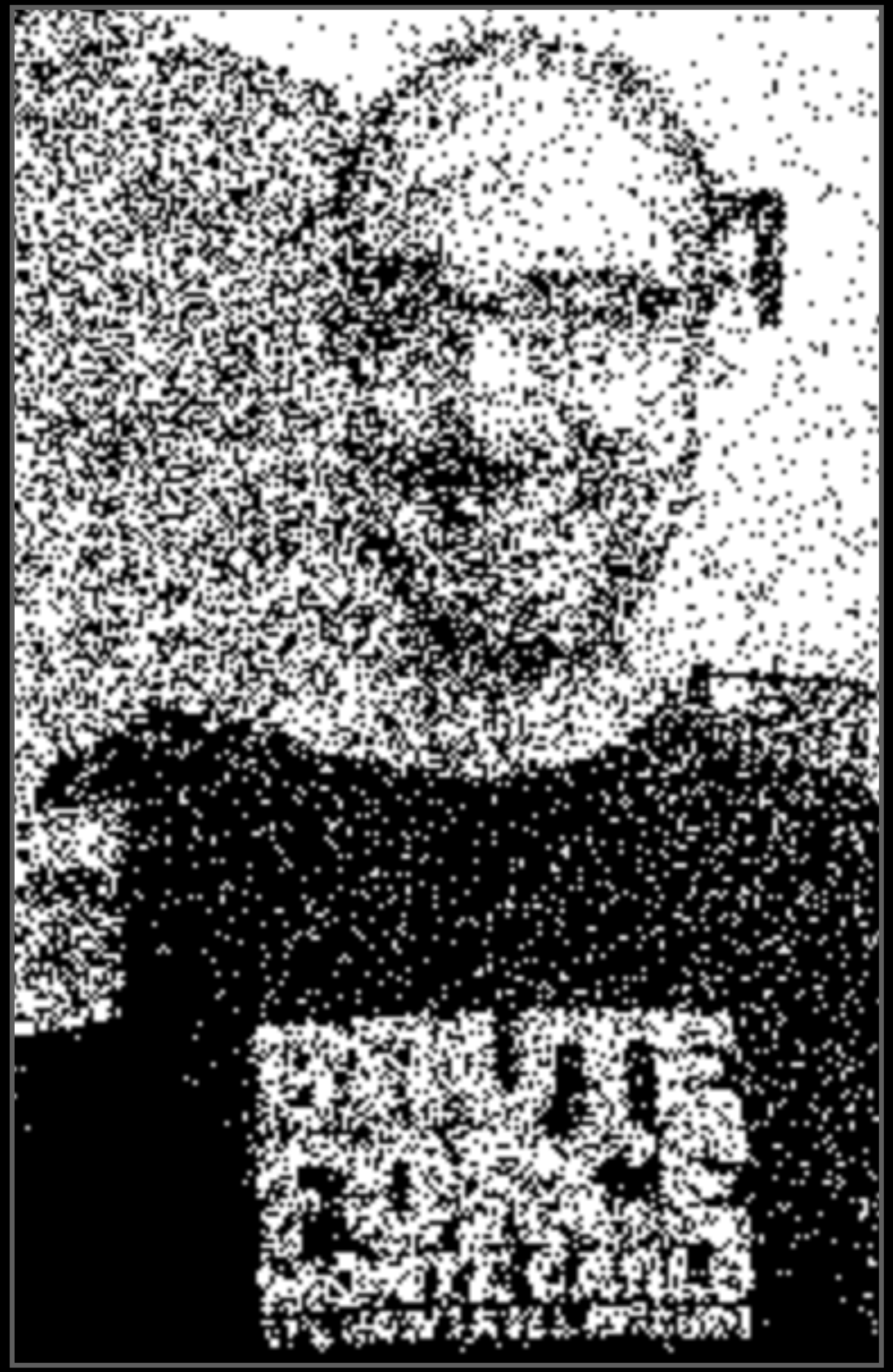


input image

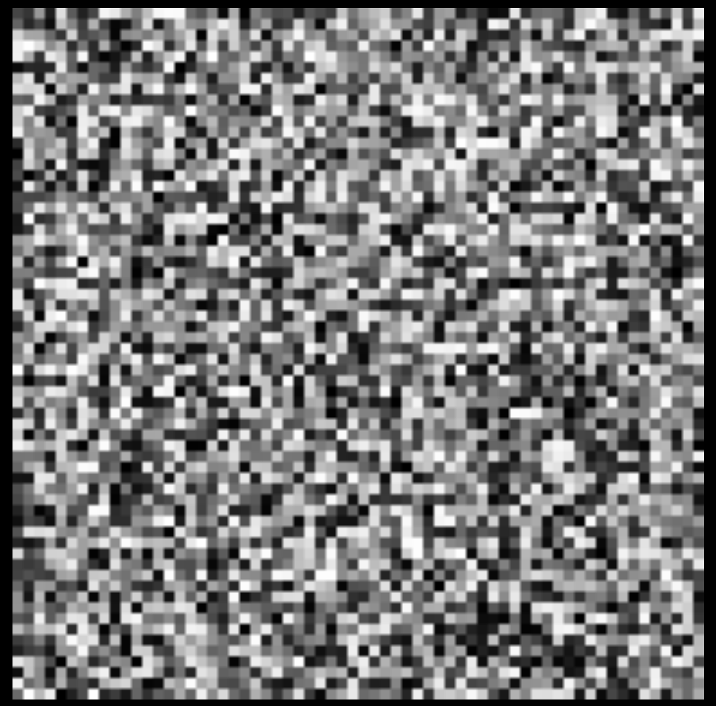


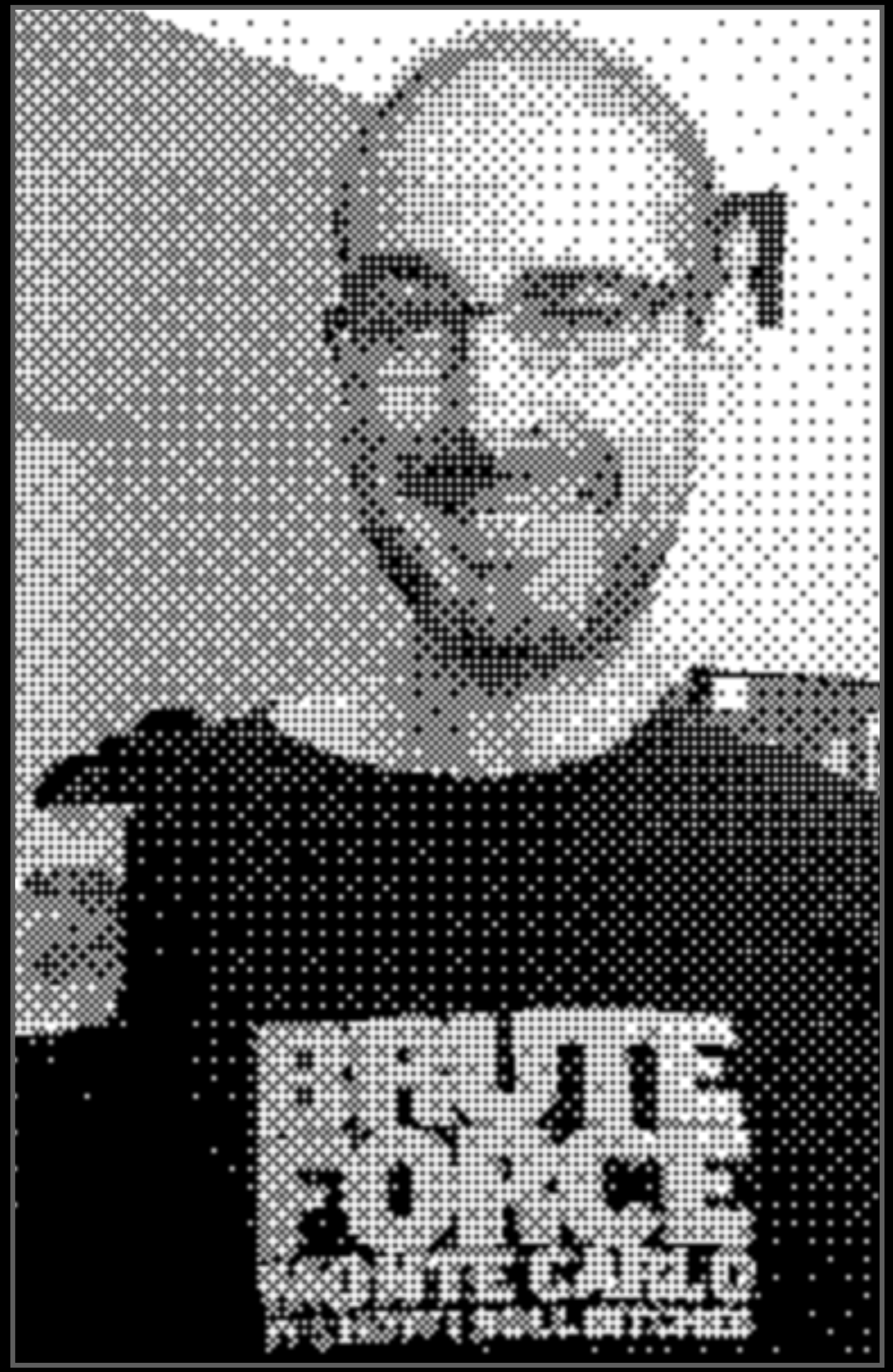
constant



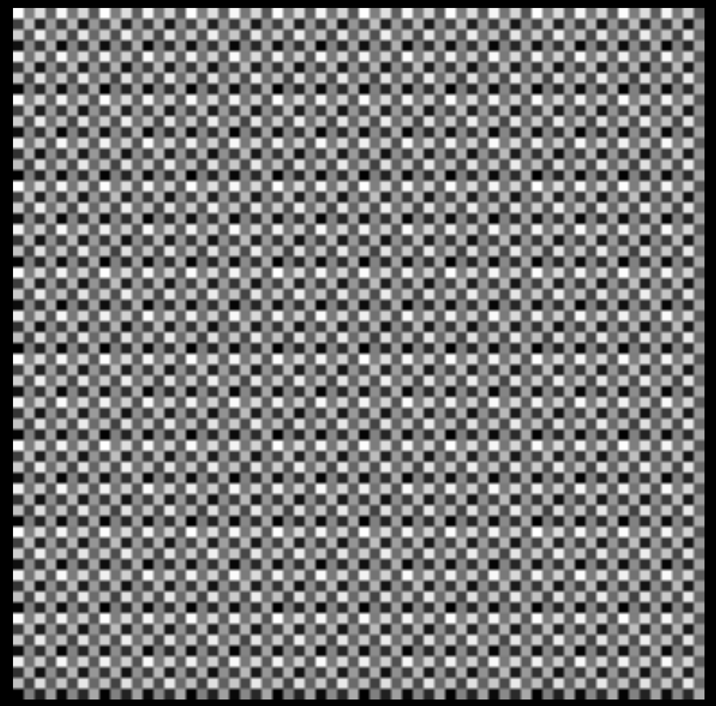


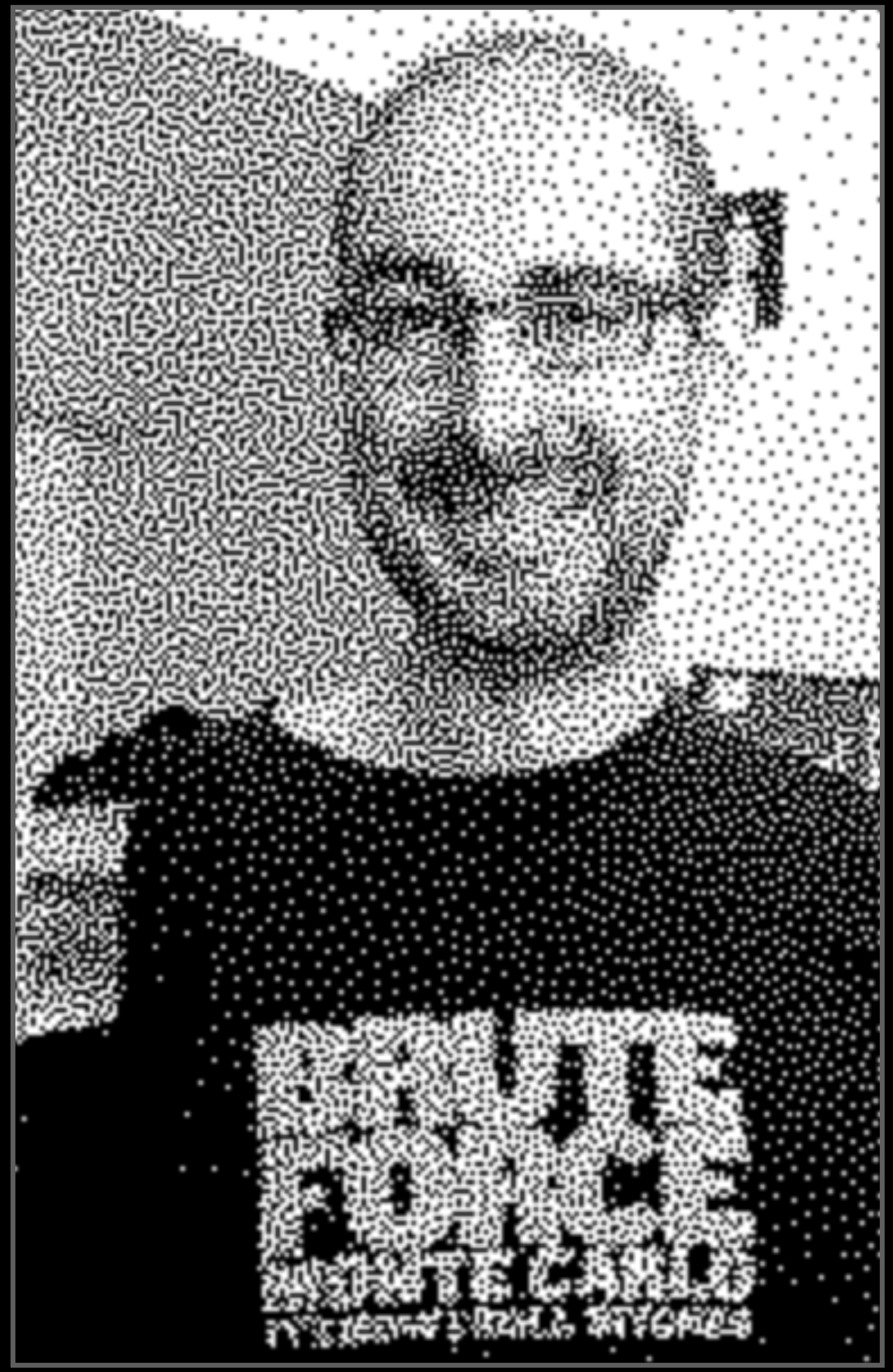
random dither



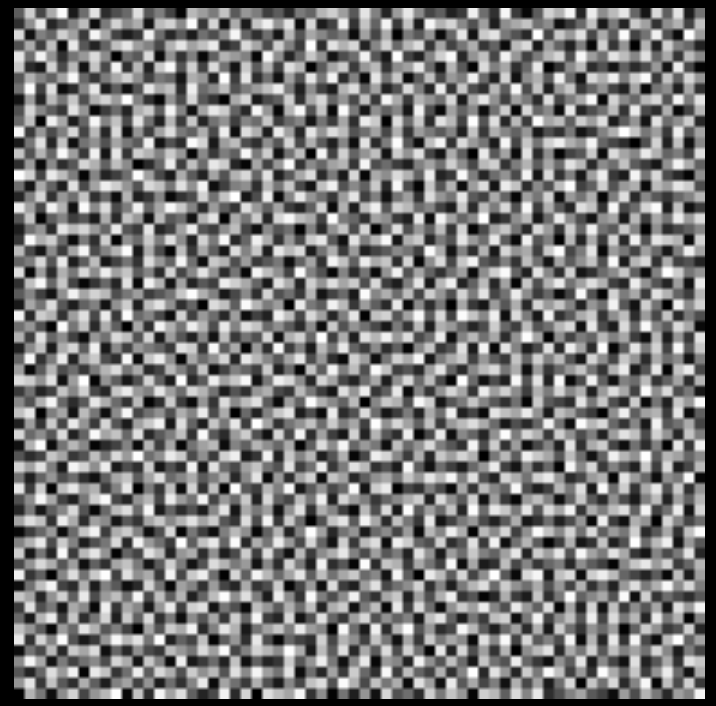


ordered dither



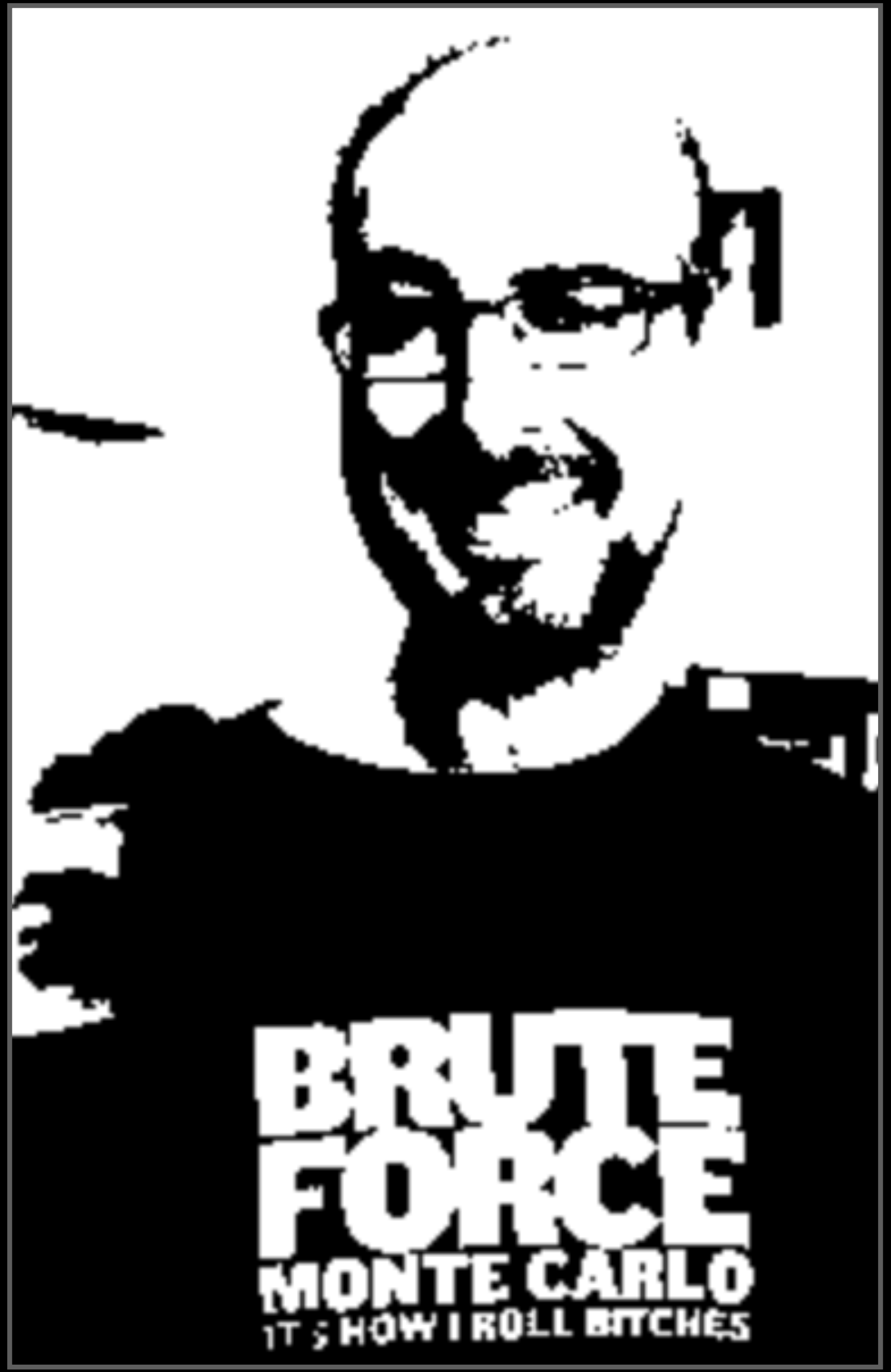


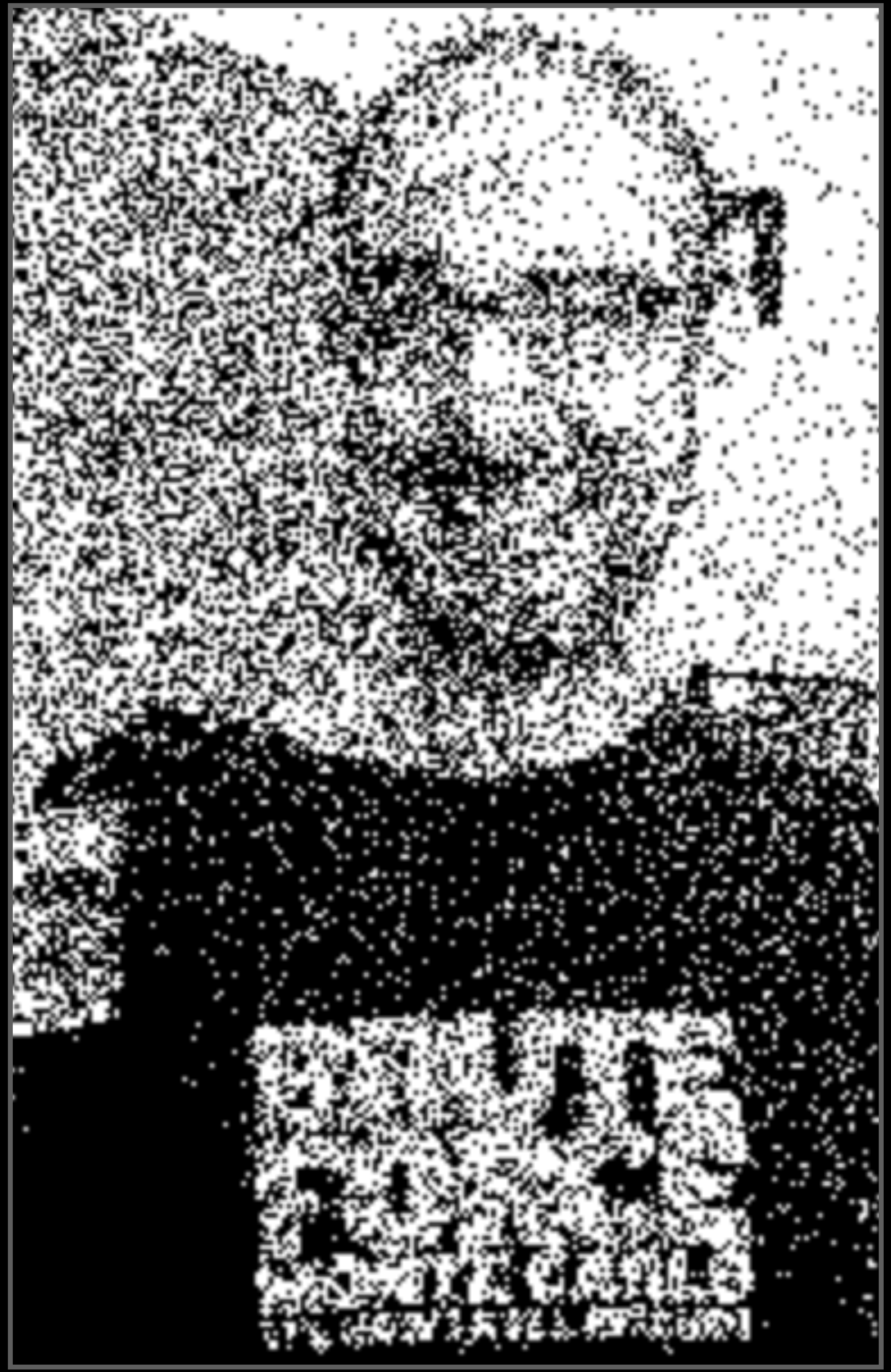
blue-noise dither

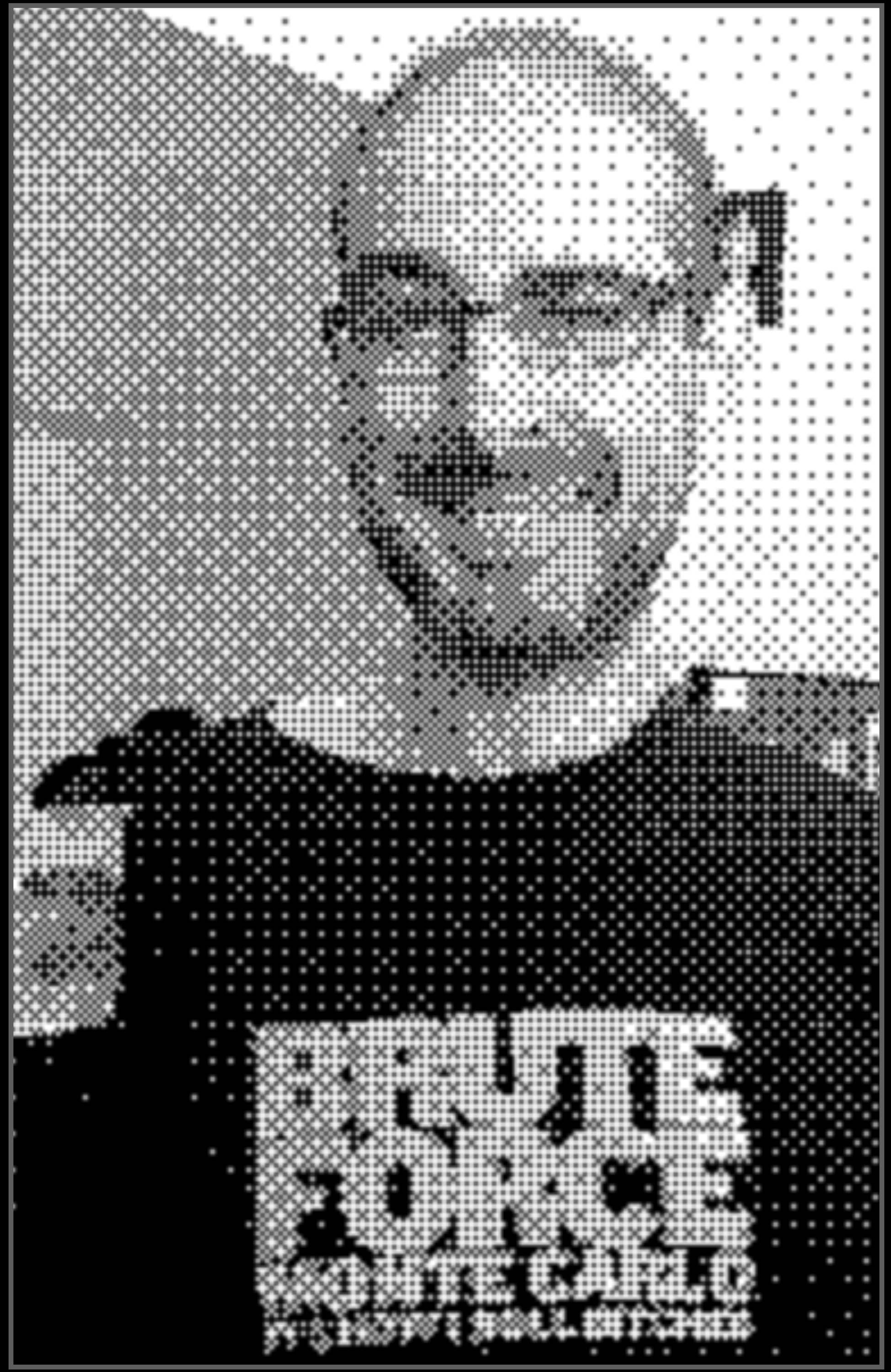


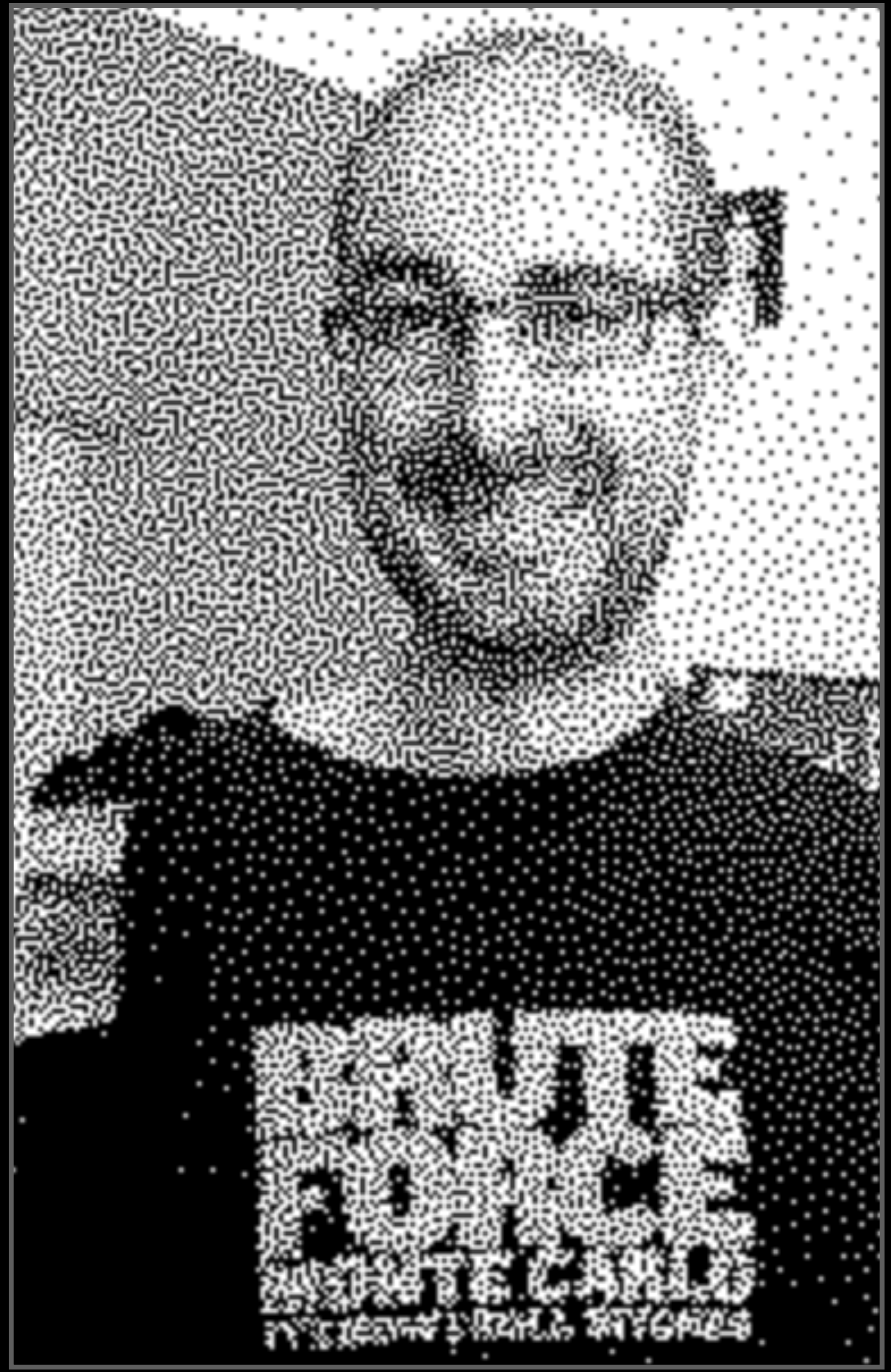
Halftoning: image thresholding

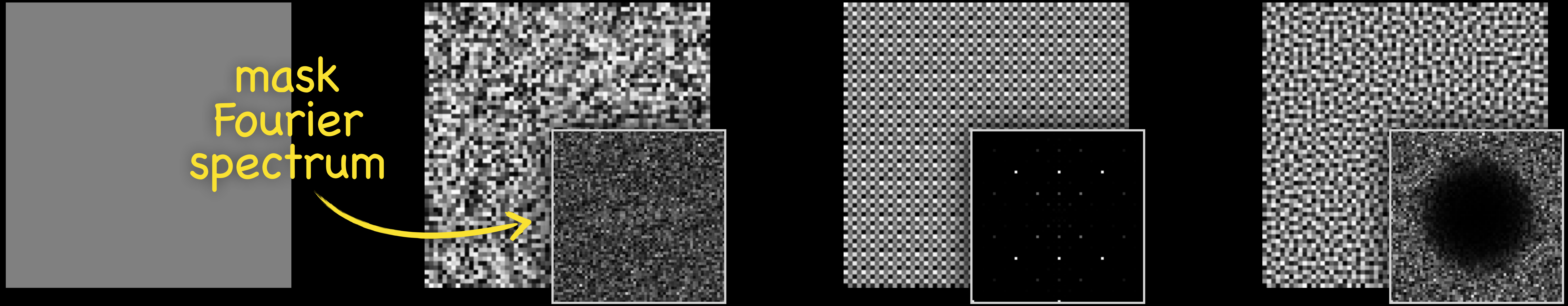
input image


constant


random dither

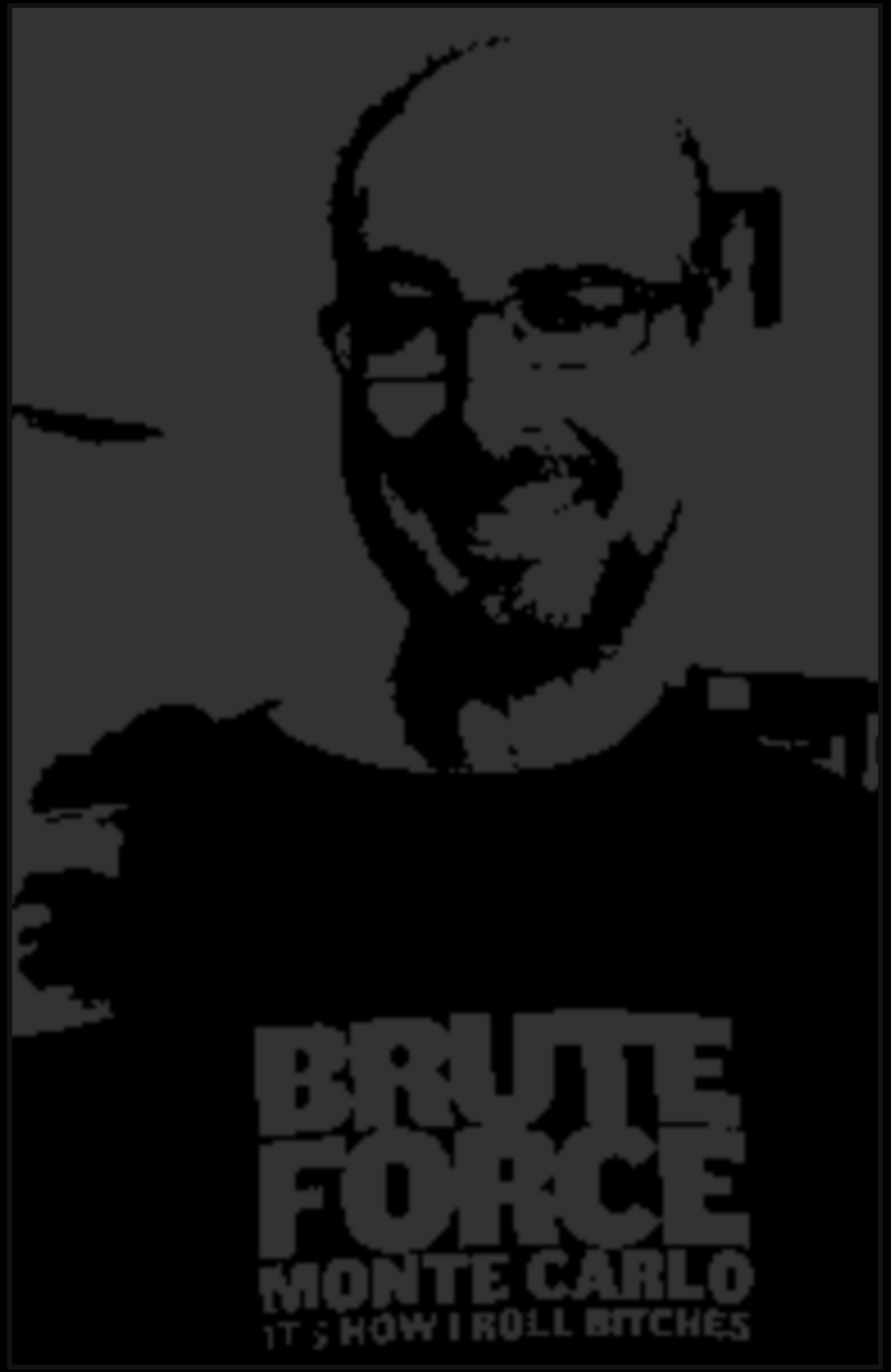

ordered dither


blue-noise dither

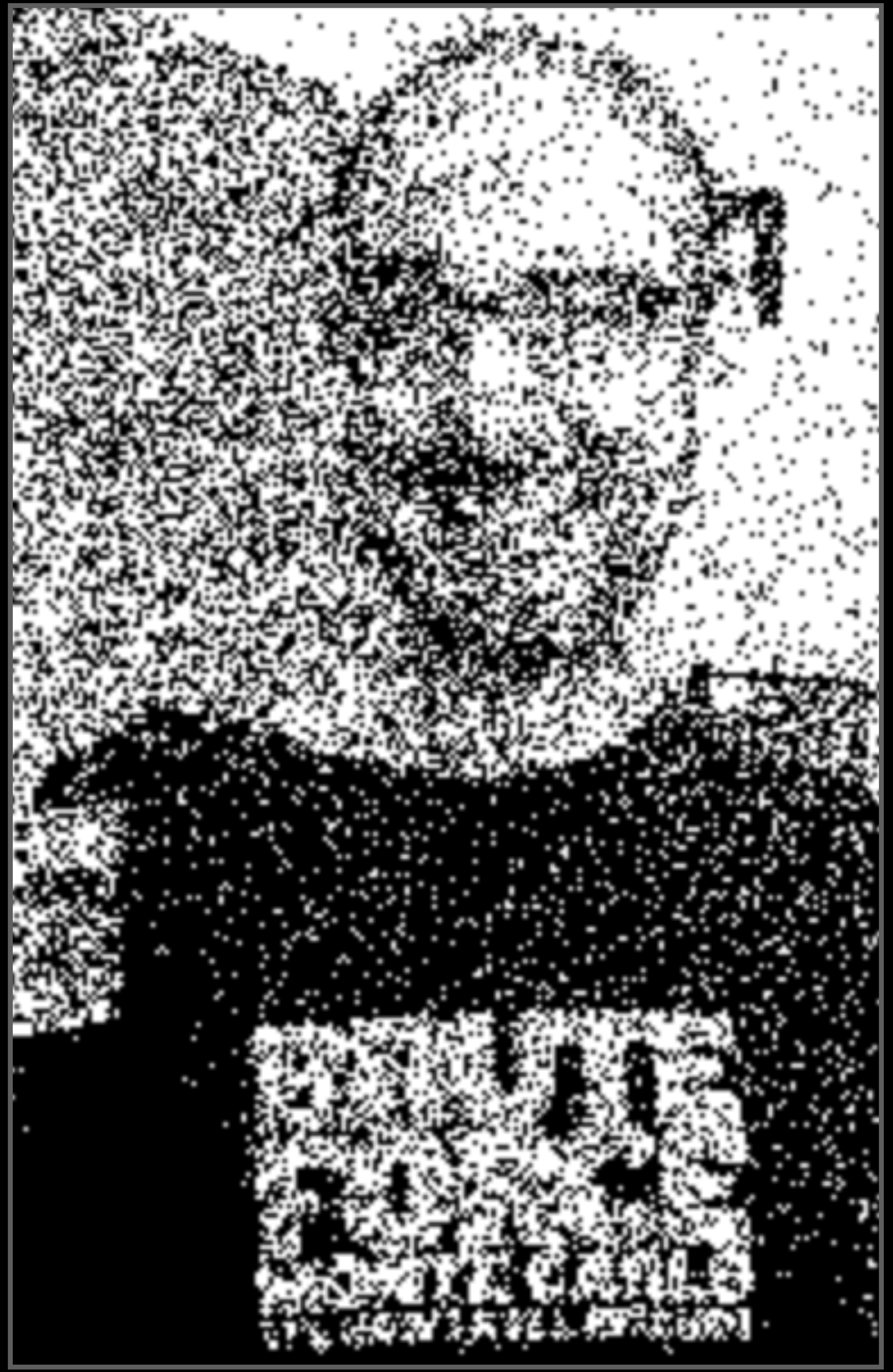


Halftoning: image thresholding

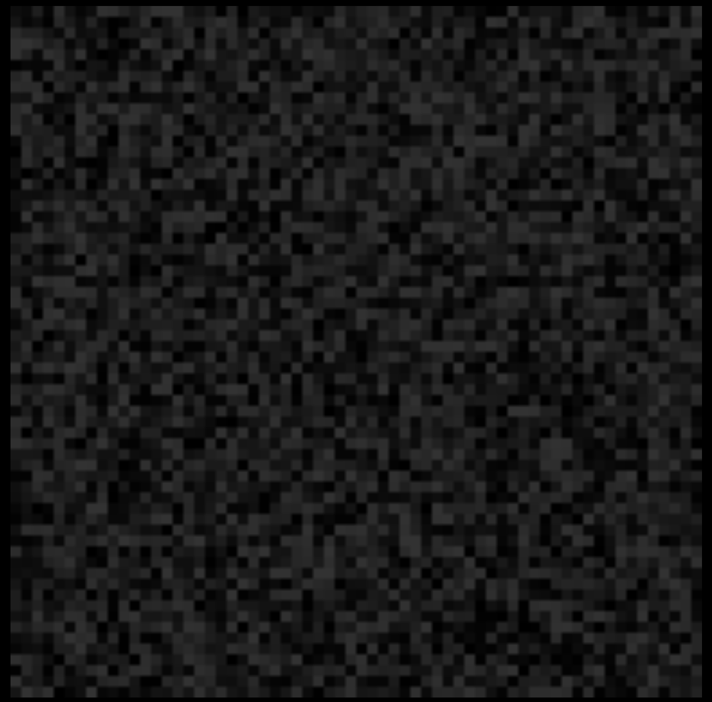
input image

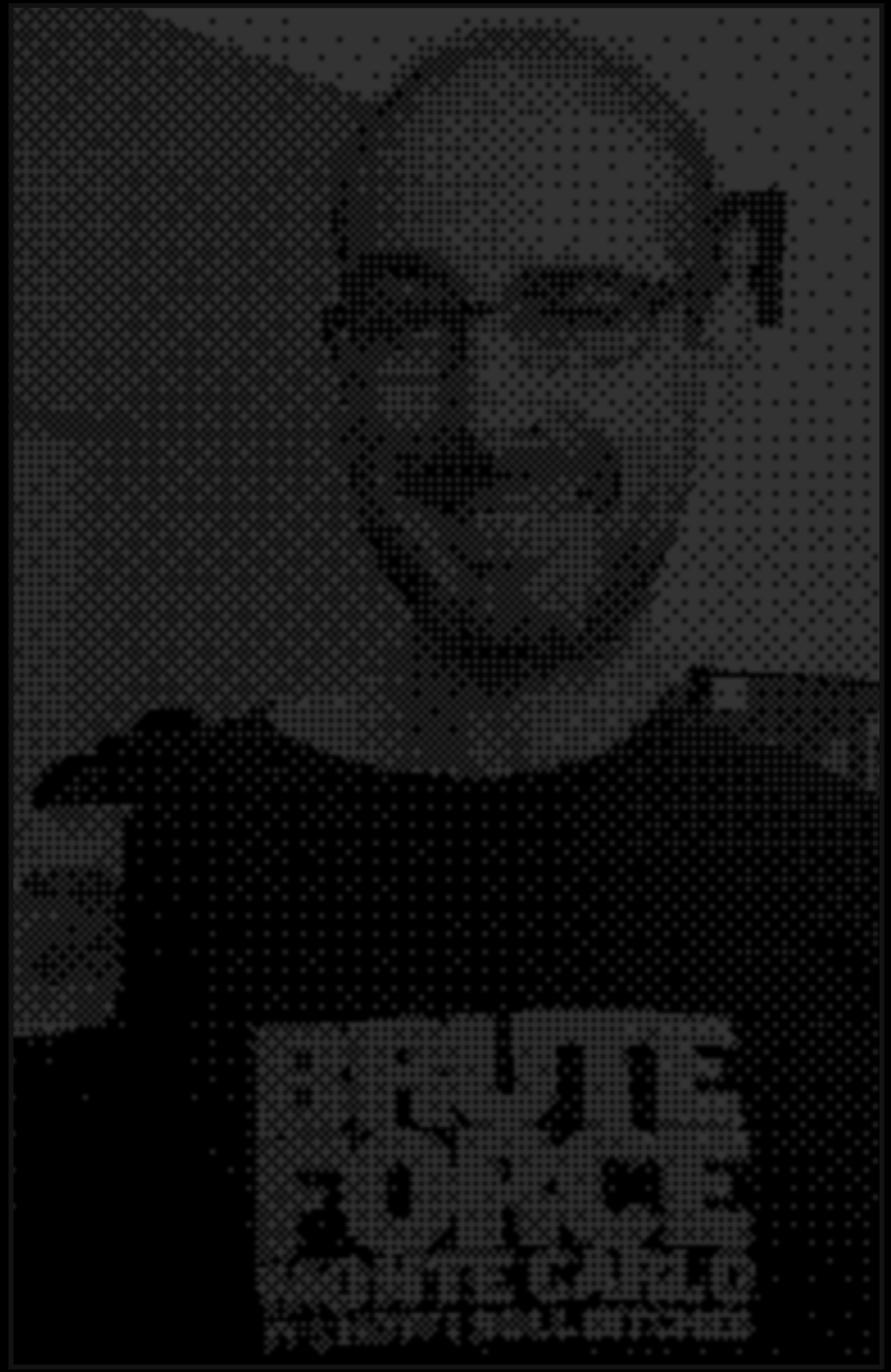


constant

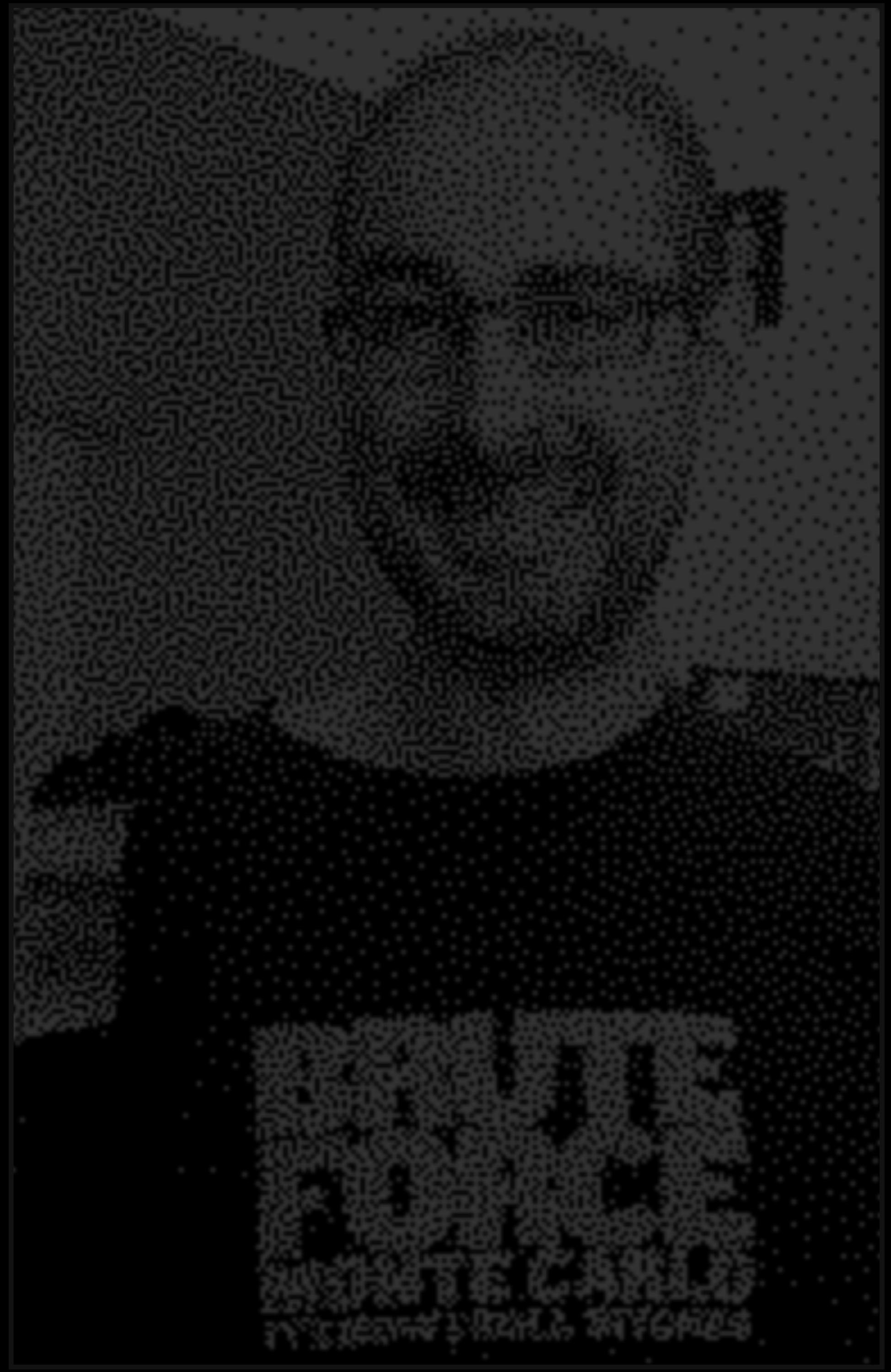


random dither

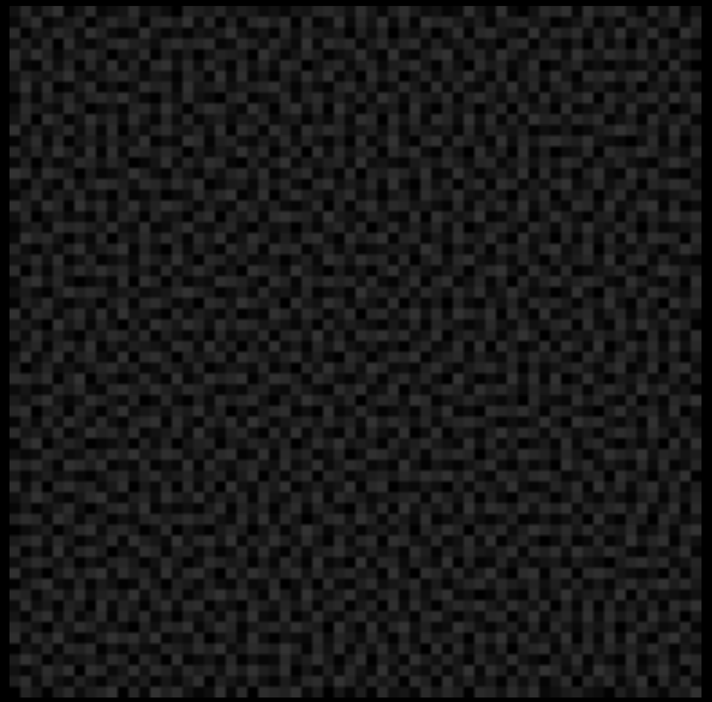




ordered dither

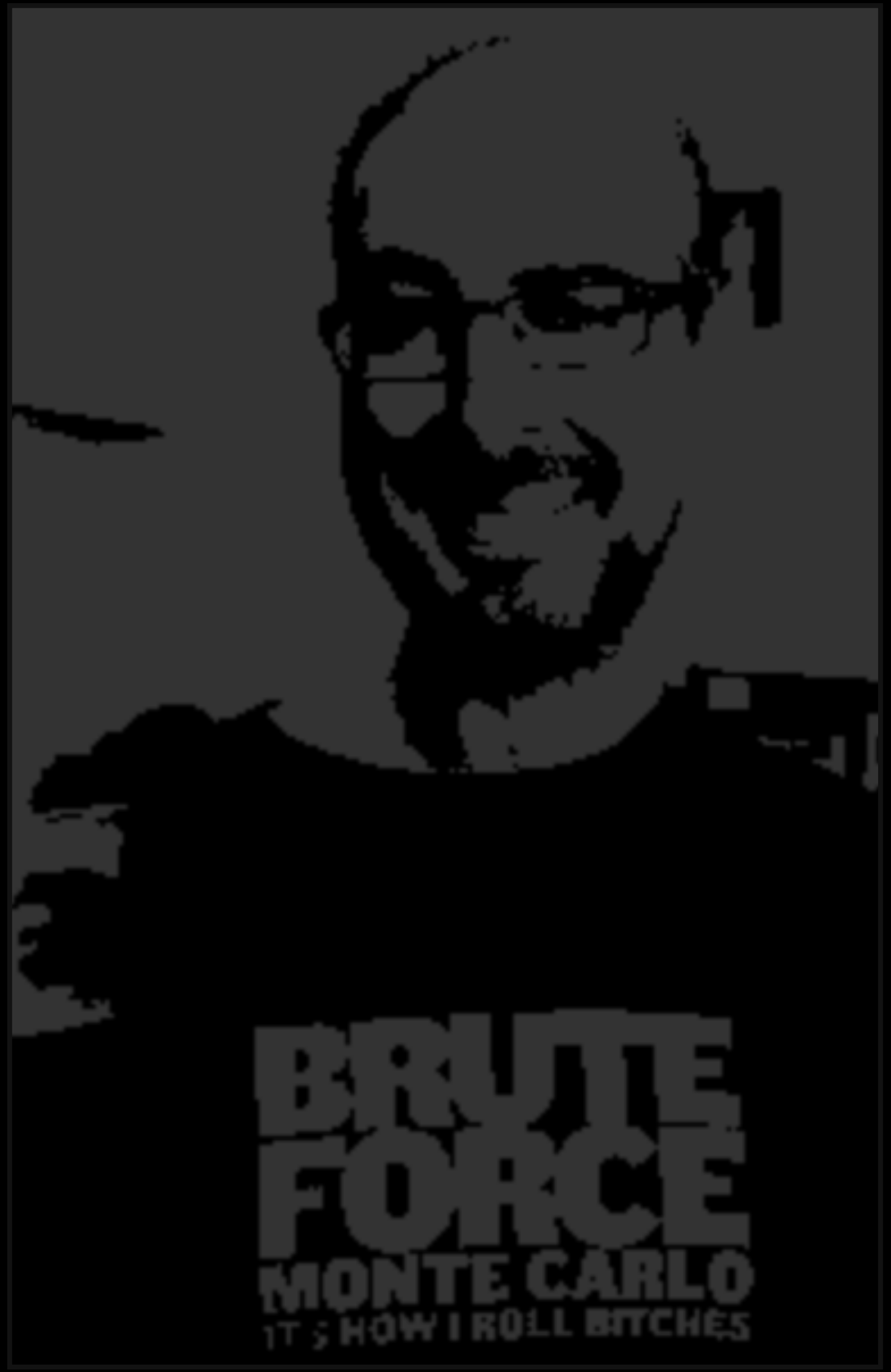


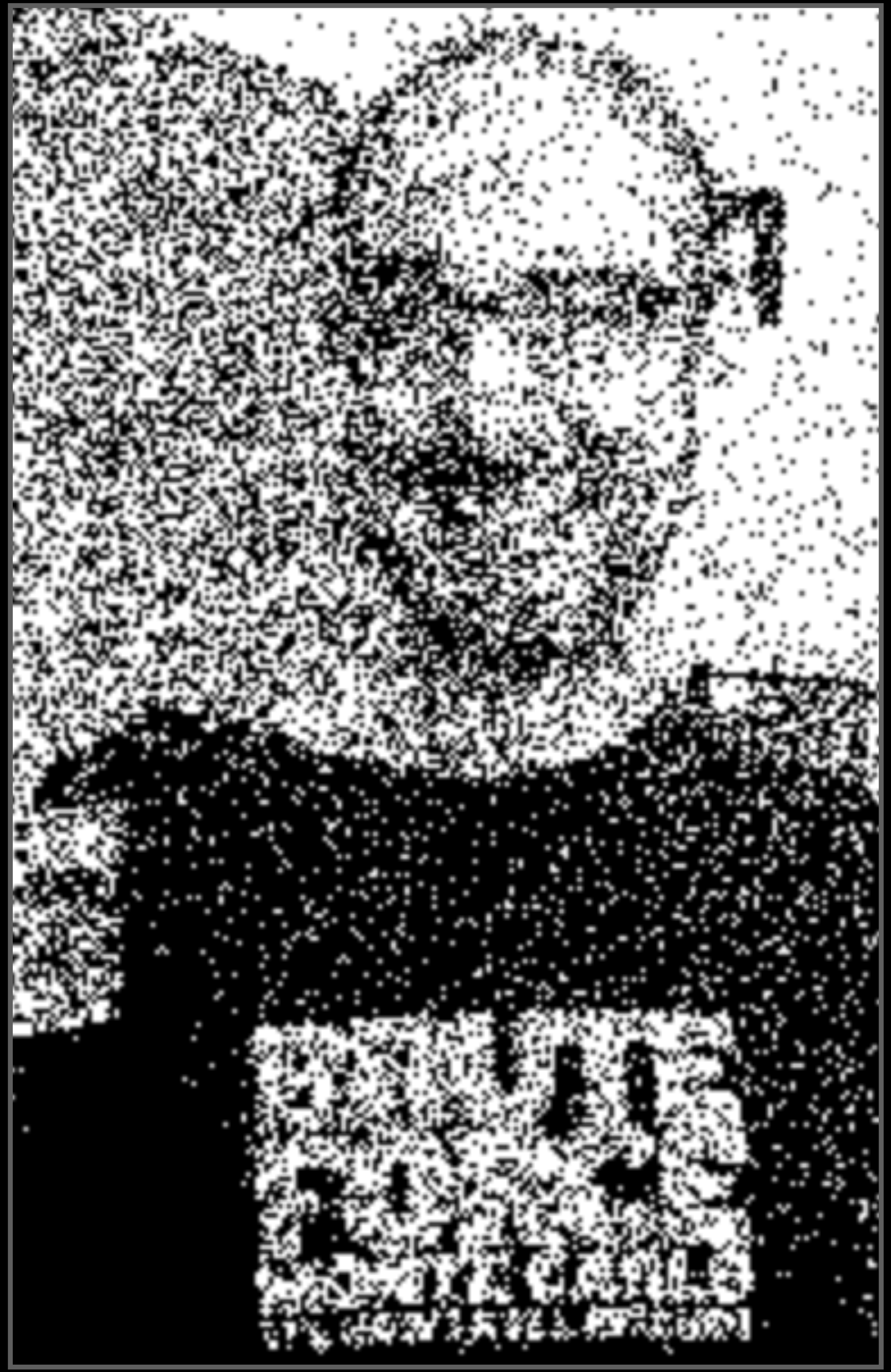
blue-noise dither

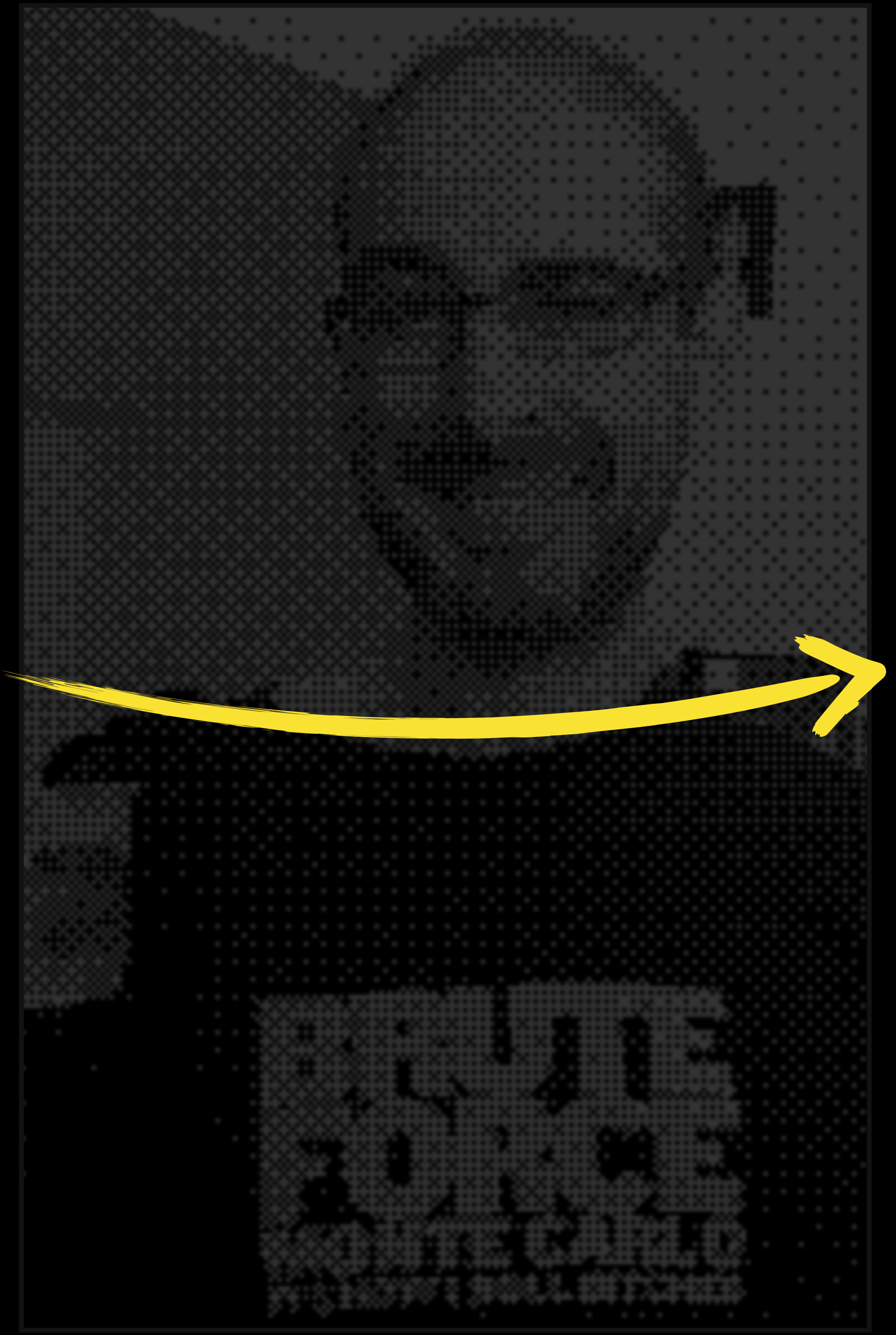


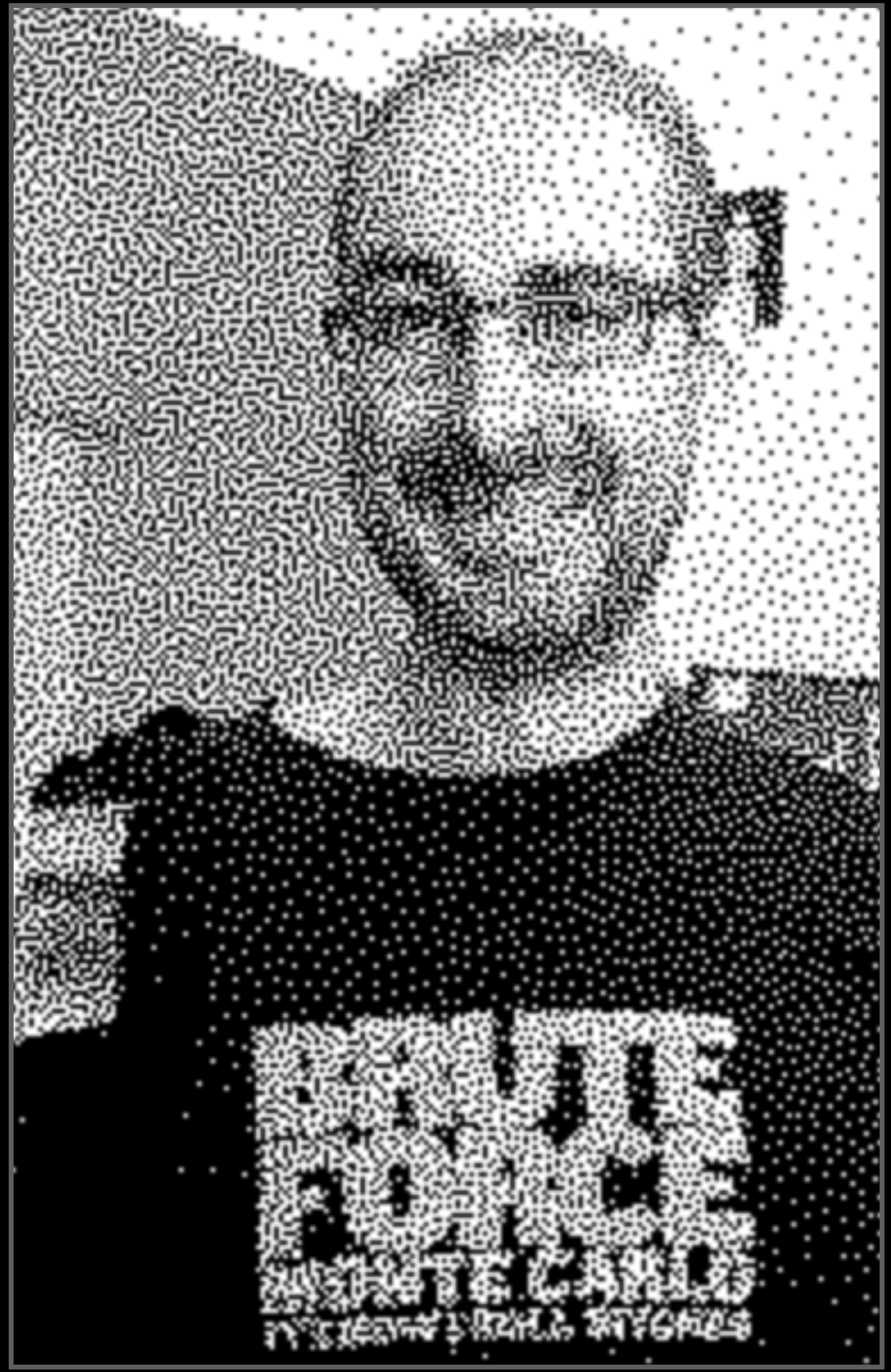
Halftoning: image thresholding

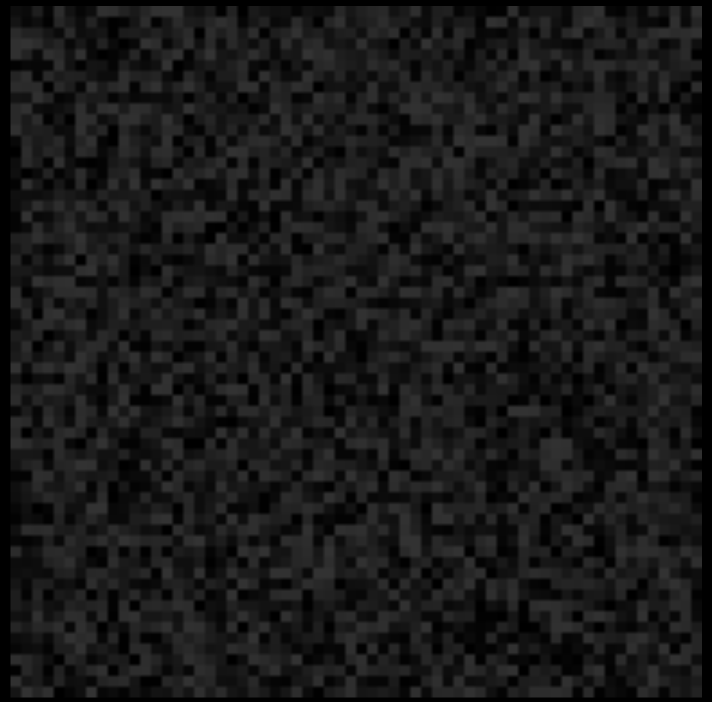
input image

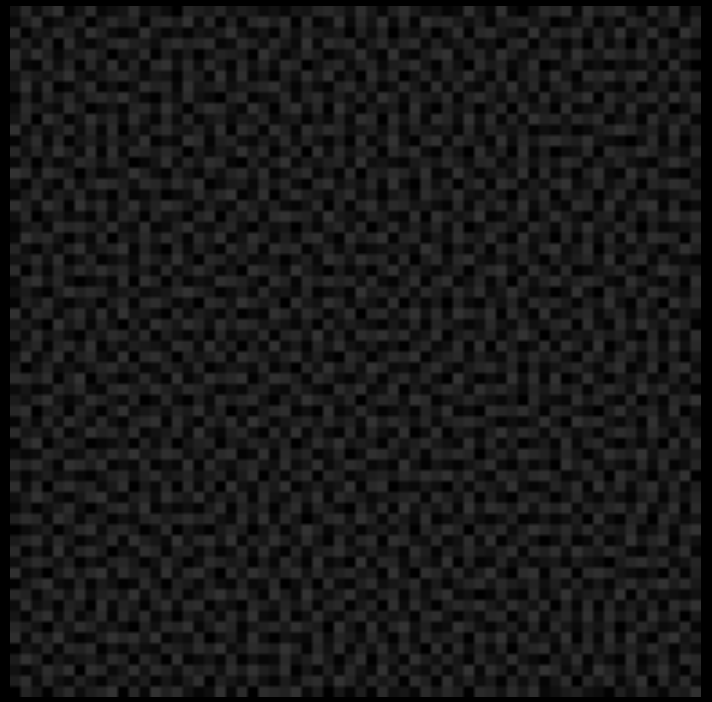

constant


random dither


ordered dither


blue-noise dither





Halftoning: image thresholding

input image

constant

random dither

ordered dither

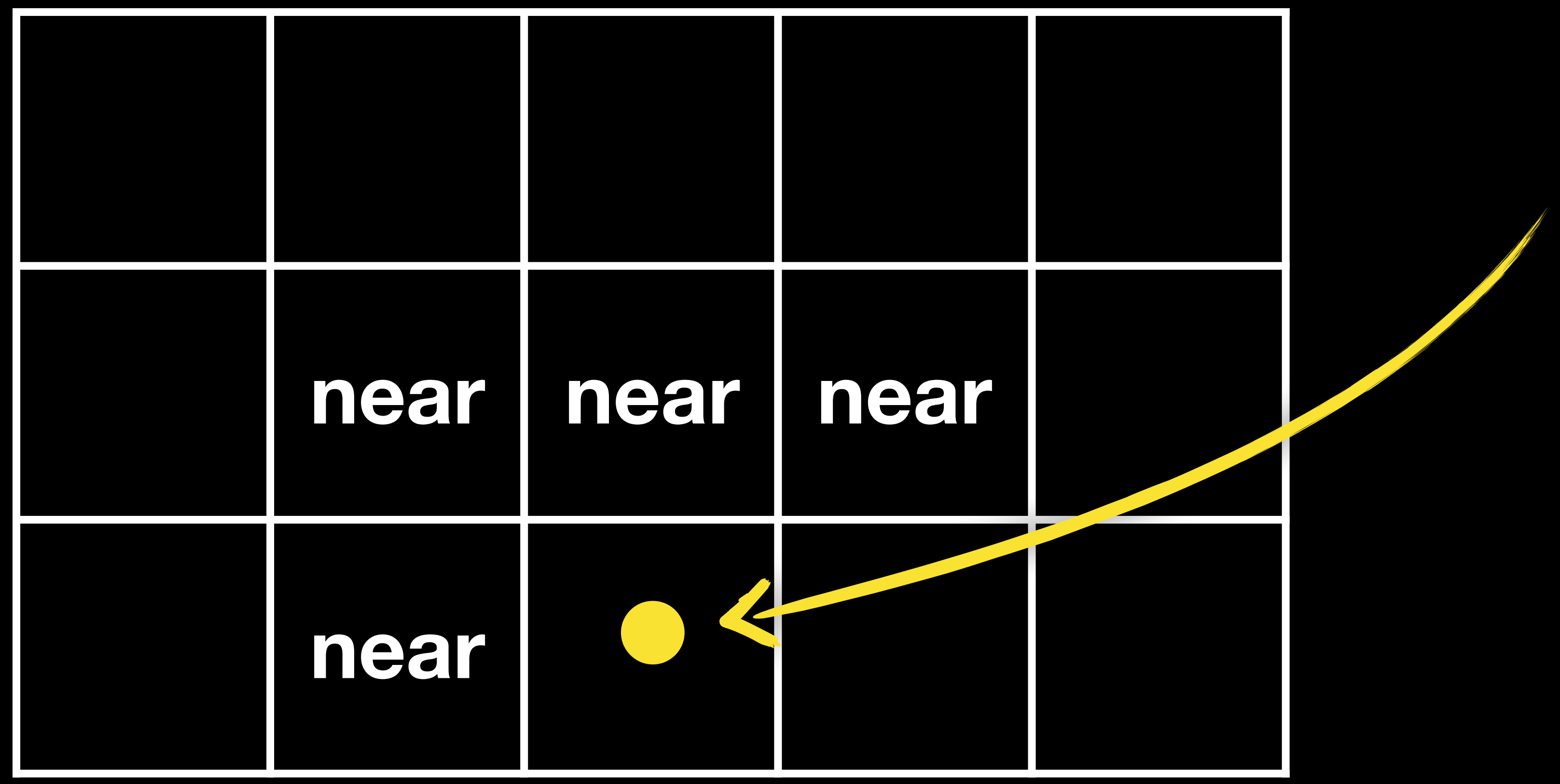
blue-noise dither

Pixel correlation: overview

- 1991 **Screen-space best-candidate** *[Mitchell]*
- 2016 **Dithered pattern offsetting** *[Georgiev & Fajardo]*
- 2019 **Dithered pattern scrambling** *[Heitz & al.]*
  Dithered seed permutation *[Heitz & Belcour]*

Halftoning-inspired

Screen-space best-candidate *[Mitchell ’91]*

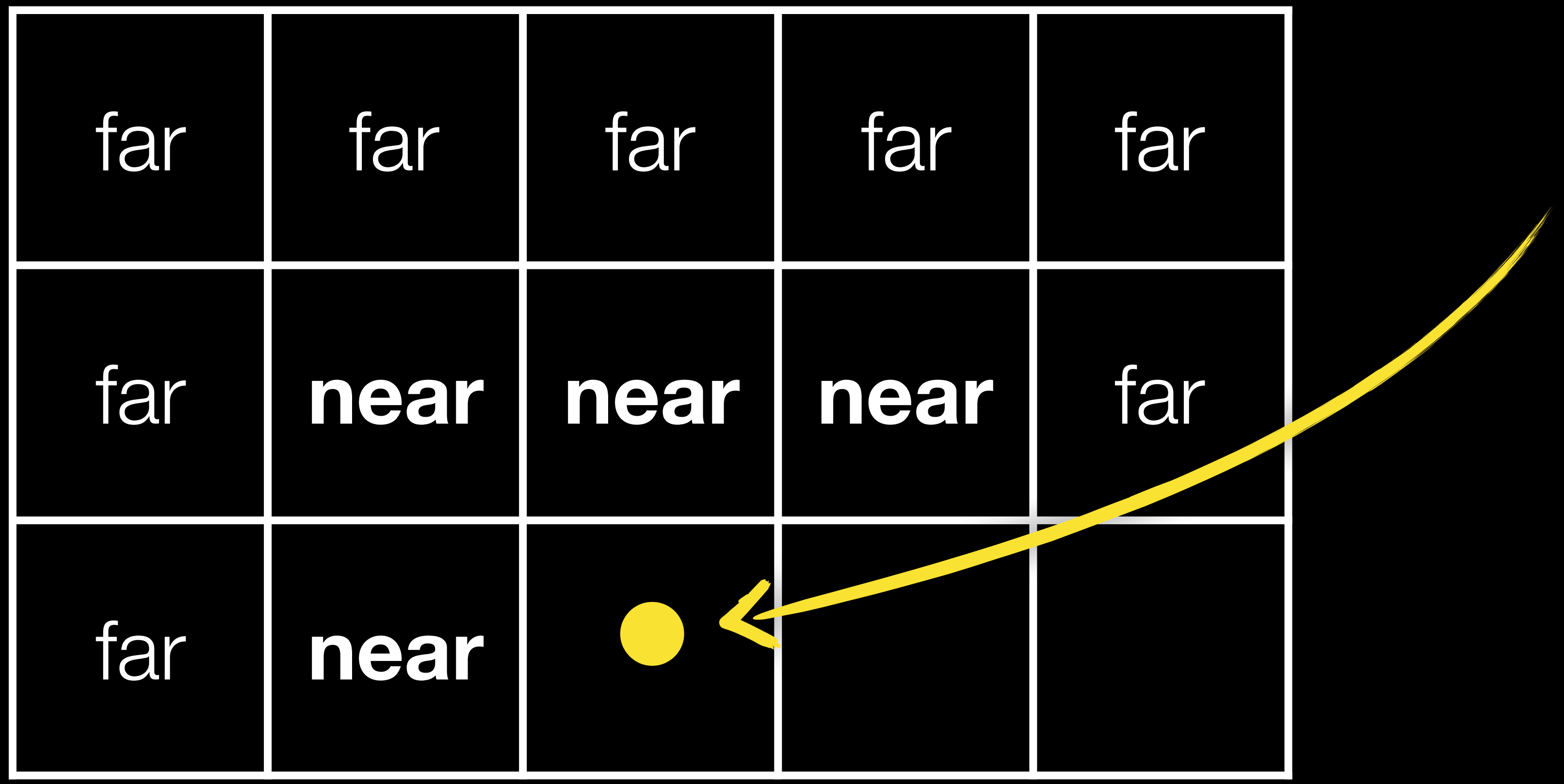


two-step resampling

generate 100 candidates:

1. select 10 farthest ones

Screen-space best-candidate *[Mitchell ’91]*



two-step resampling

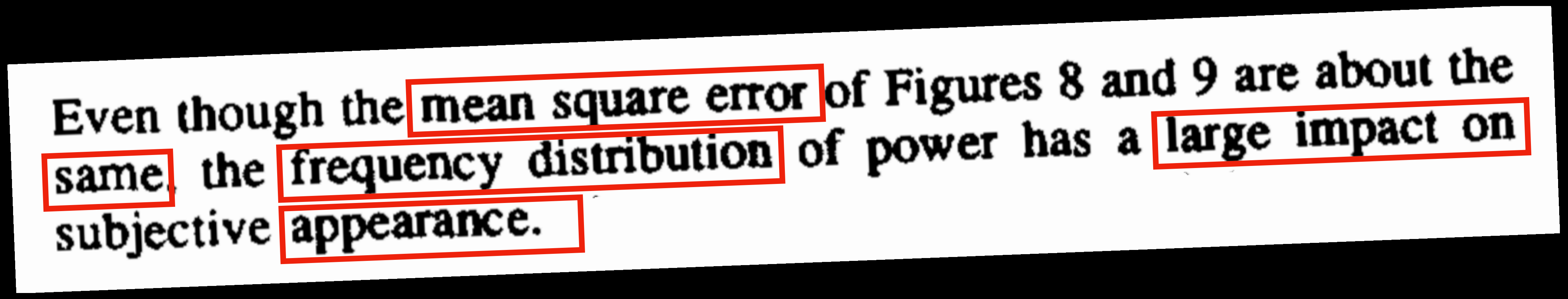
generate 100 candidates:

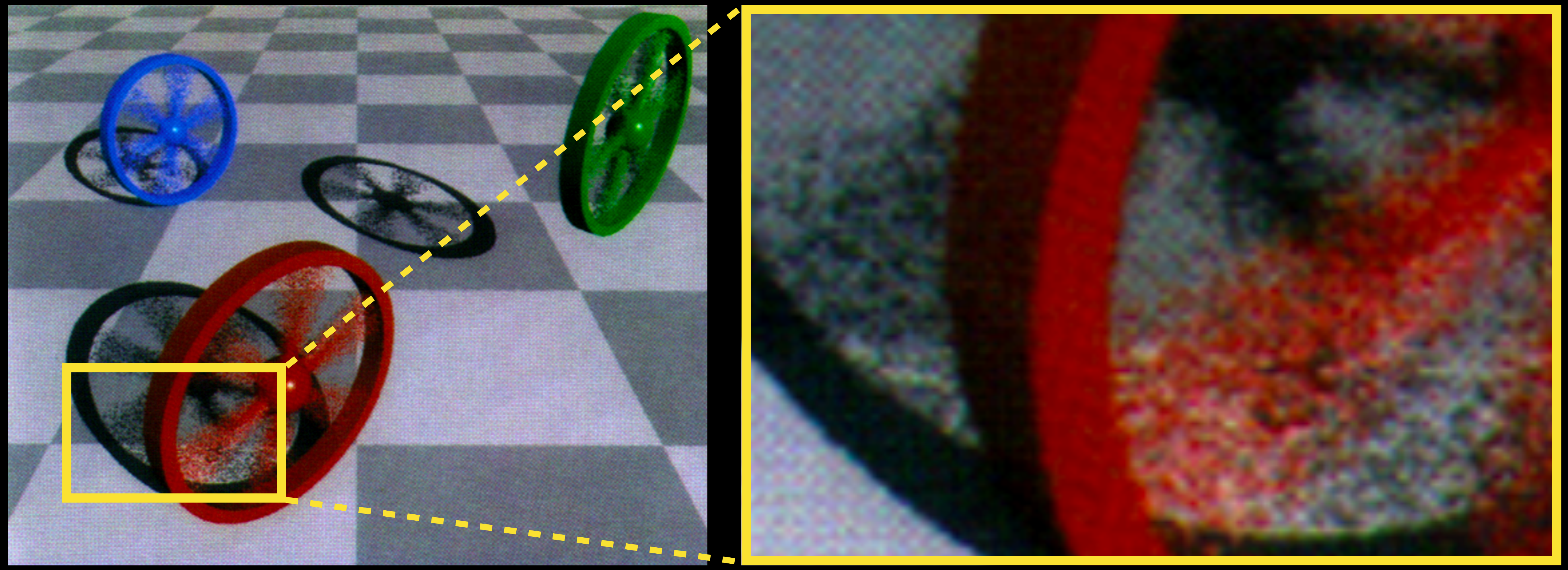
1. select 10 farthest ones
2. select farthest one

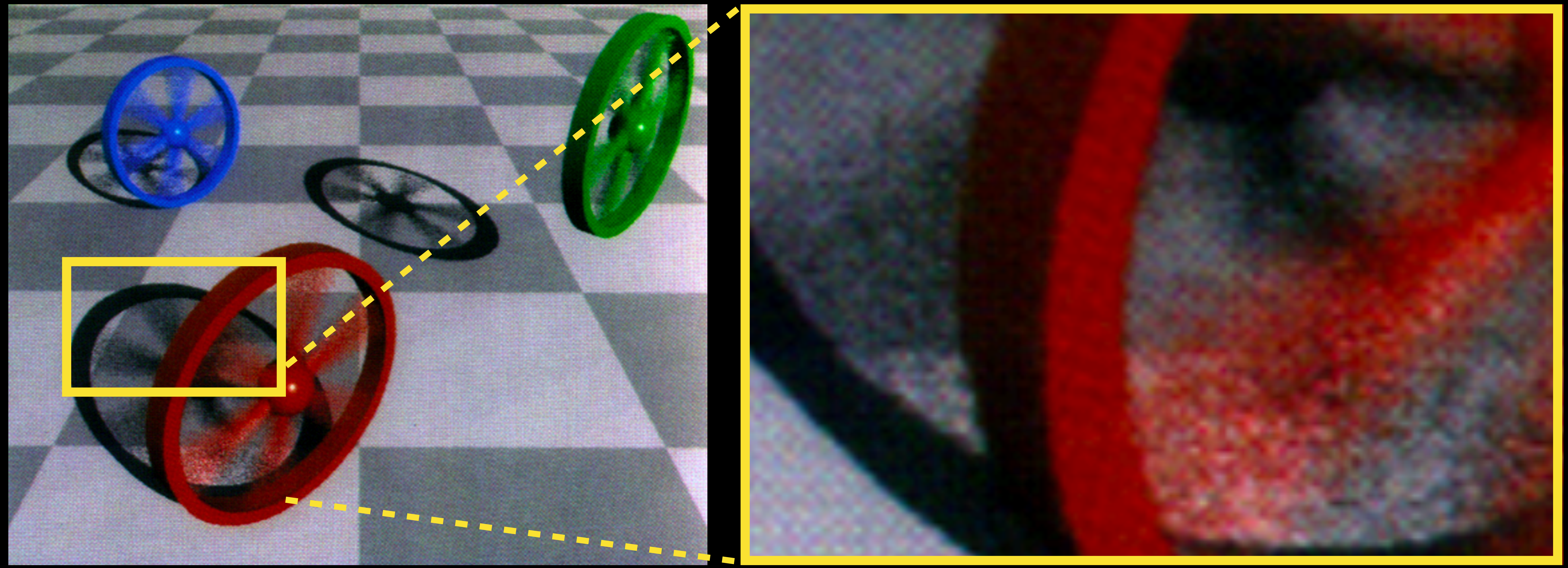
Screen-space best-candidate *[Mitchell '91]*

far	far	far	far	far
far	**near**	**near**	**near**	far
far	**near**	●		

two-step resampling


Even though the mean square error of Figures 8 and 9 are about the same, the frequency distribution of power has a large impact on subjective appearance.


uncorrelated time sampling


resampling method

Uncorrelated pixel sampling

```
pattern = generate_pattern()
for each pixel p:
    vector = random()
    pattern_p = offset(pattern, vector)
    render_pixel(p, pattern_p)
```

Uncorrelated pixel sampling

```
pattern = generate_pattern()
for each pixel p:
    vector = random()
    pattern_p = offset(pattern, vector)
    render_pixel(p, pattern_p)
```

Dithered sampling *[Georgiev & Fajardo 2016]*

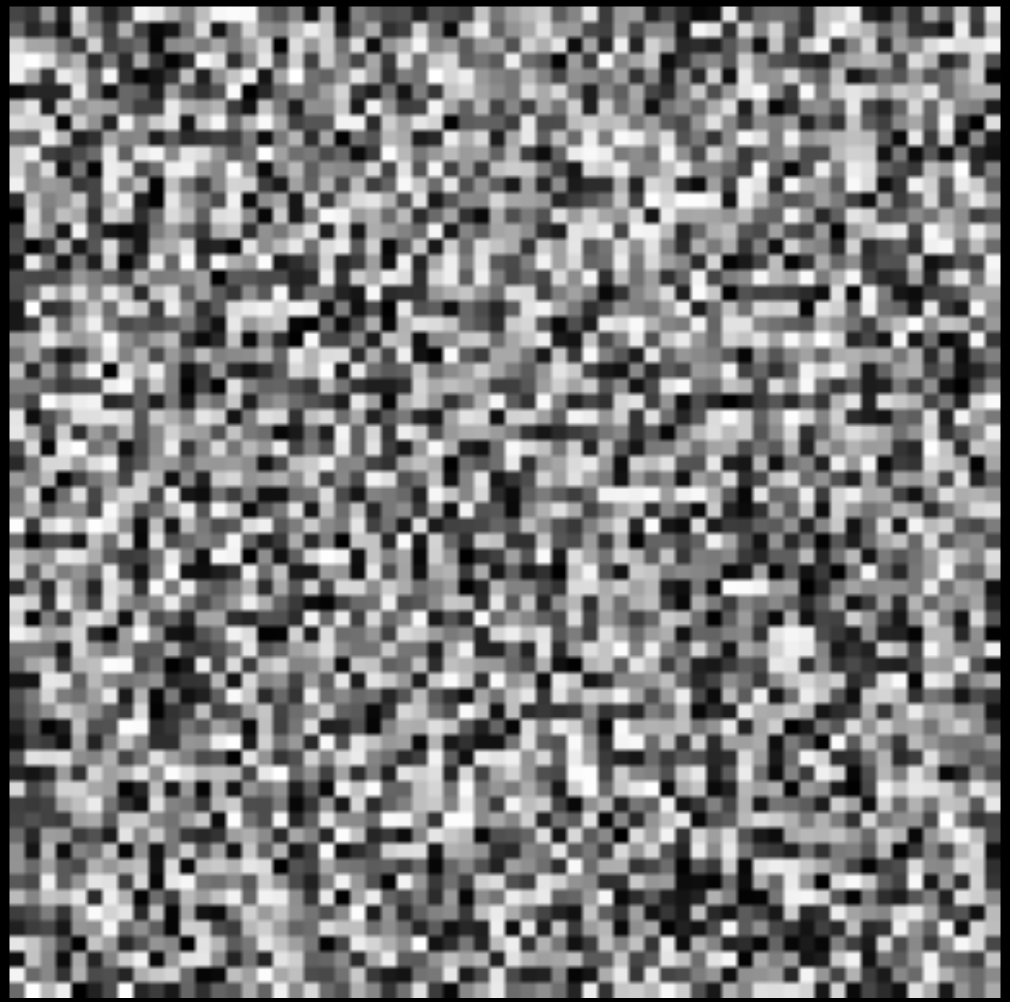
```
pattern = generate_pattern()
for each pixel p:
    vector = lookup(mask, p)
    pattern_p = offset(pattern, vector)
    render_pixel(p, pattern_p)
```

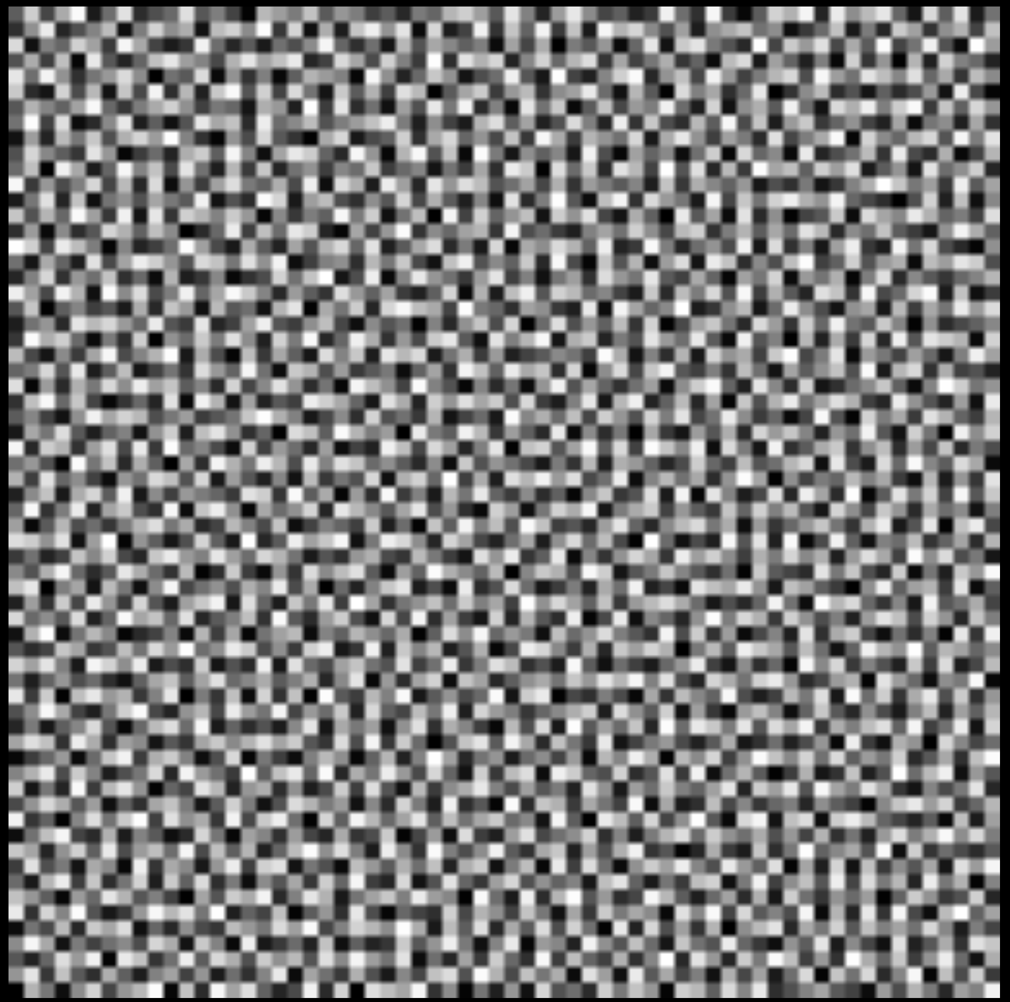
Dithered sampling *[Georgiev & Fajardo 2016]*

```
pattern = generate_pattern()
for each pixel p:
    vector = lookup(mask, p) // mask tiled over the image plane
    pattern_p = offset(pattern, vector)
    render_pixel(p, pattern_p)
```

use this, please

low-frequency


all-frequency

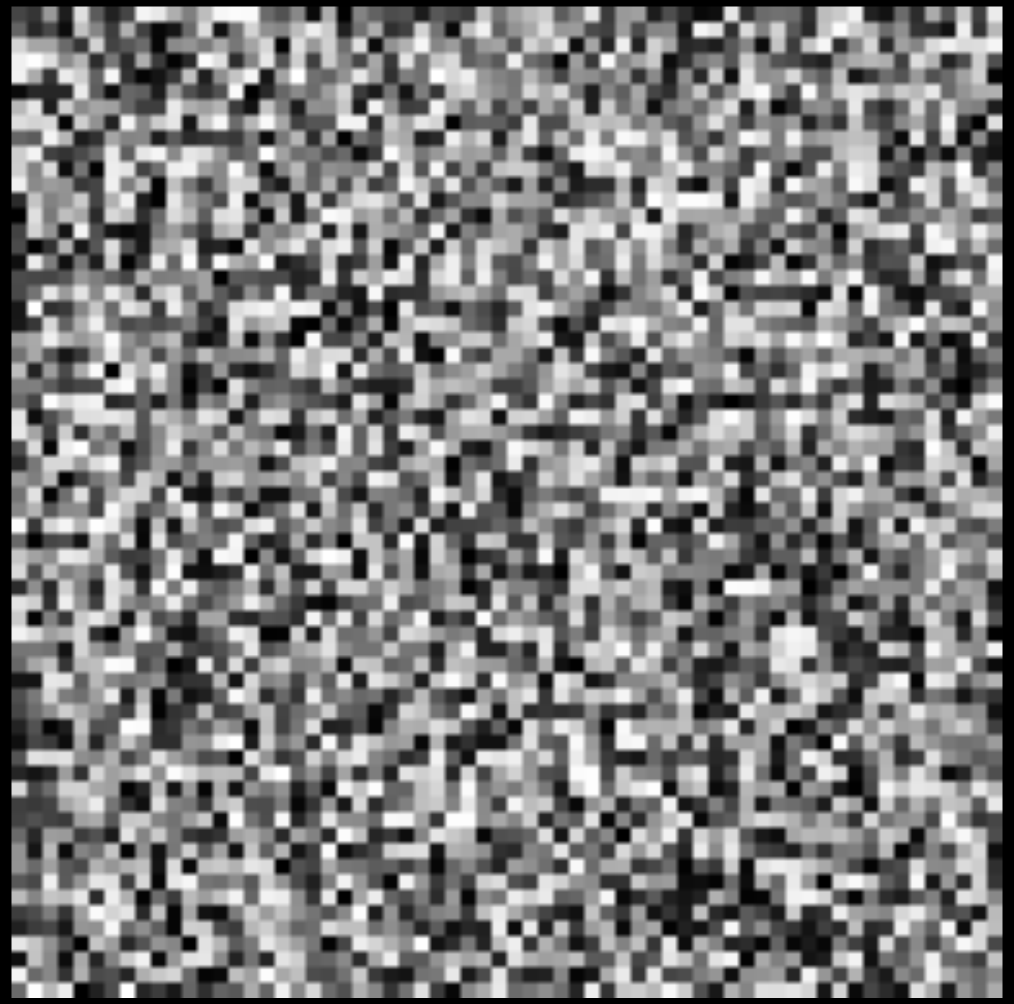

high-frequency

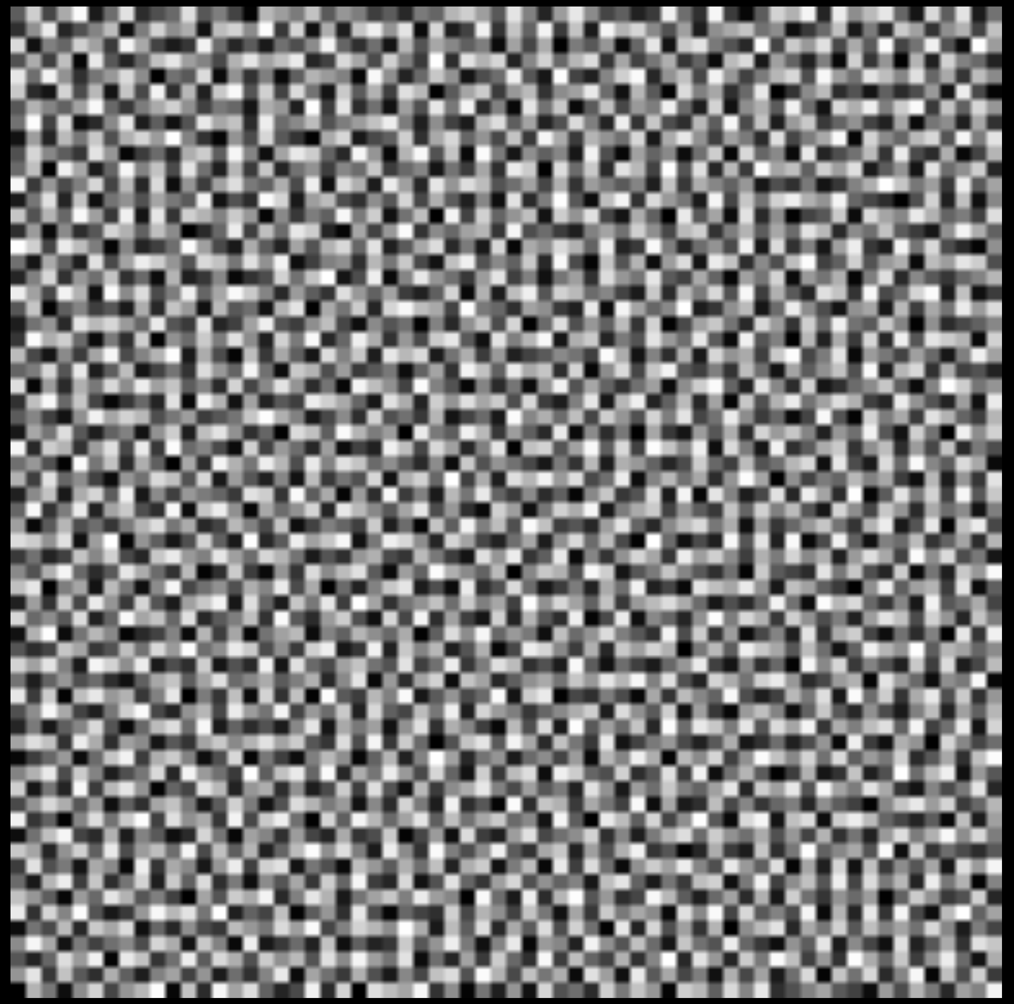
Dithered sampling *[Georgiev & Fajardo 2016]*

```
1. pattern = generate_pattern()
2. for each pixel p:
3.     vector = lookup(mask, p) // mask tiled over the image plane
4.     pattern_p = offset(pattern, vector)
5.     render_pixel(p, pattern_p)
```

use this, please

low-frequency


all-frequency


high-frequency

Dithered sampling: mask construction

```
1. M = random_mask()
2. until converged:
3.    p,q = pick_random_pixels(M)
4.    if swap_reduces_energy(M,p,q): // probabilistic
5.       swap(p,q)
```

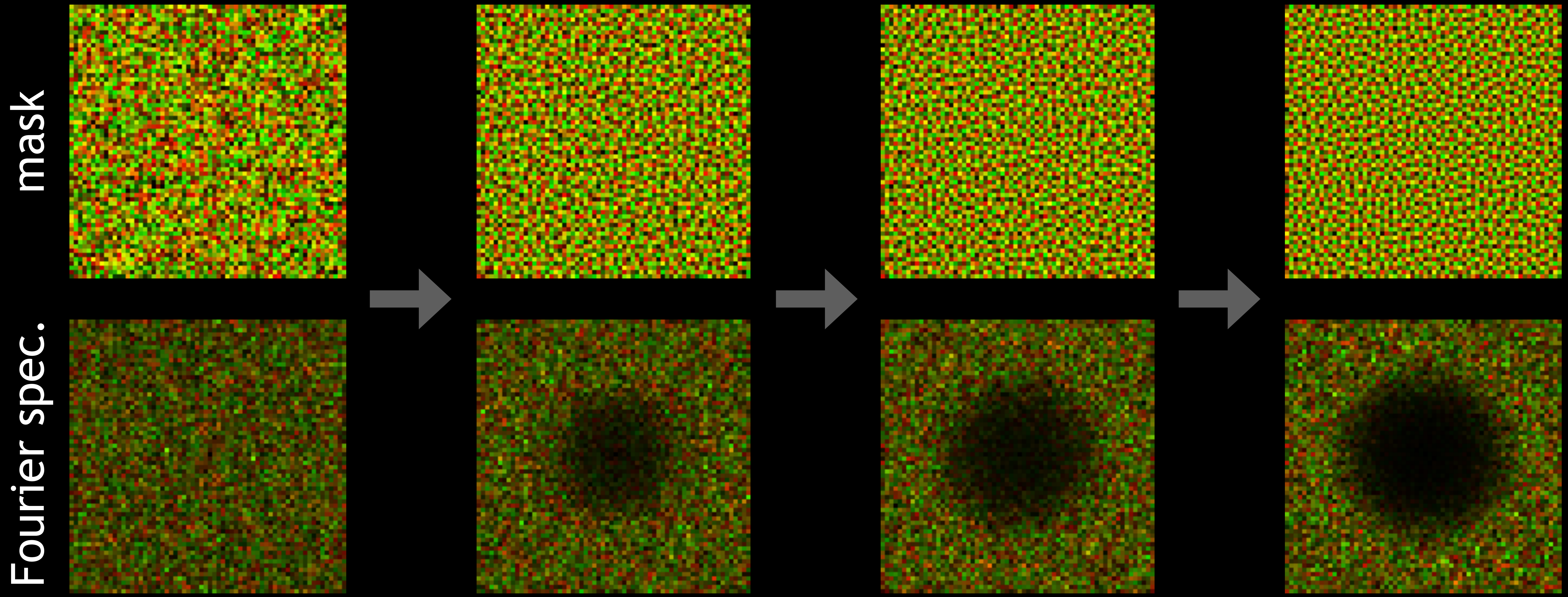
Dithered sampling: mask construction

```
1. M = random_mask()
2. until converged:
3.     p,q = pick_random_pixels(M)
4.     if swap_reduces_energy(M,p,q): // probabilistic
5.         swap(p,q)
```

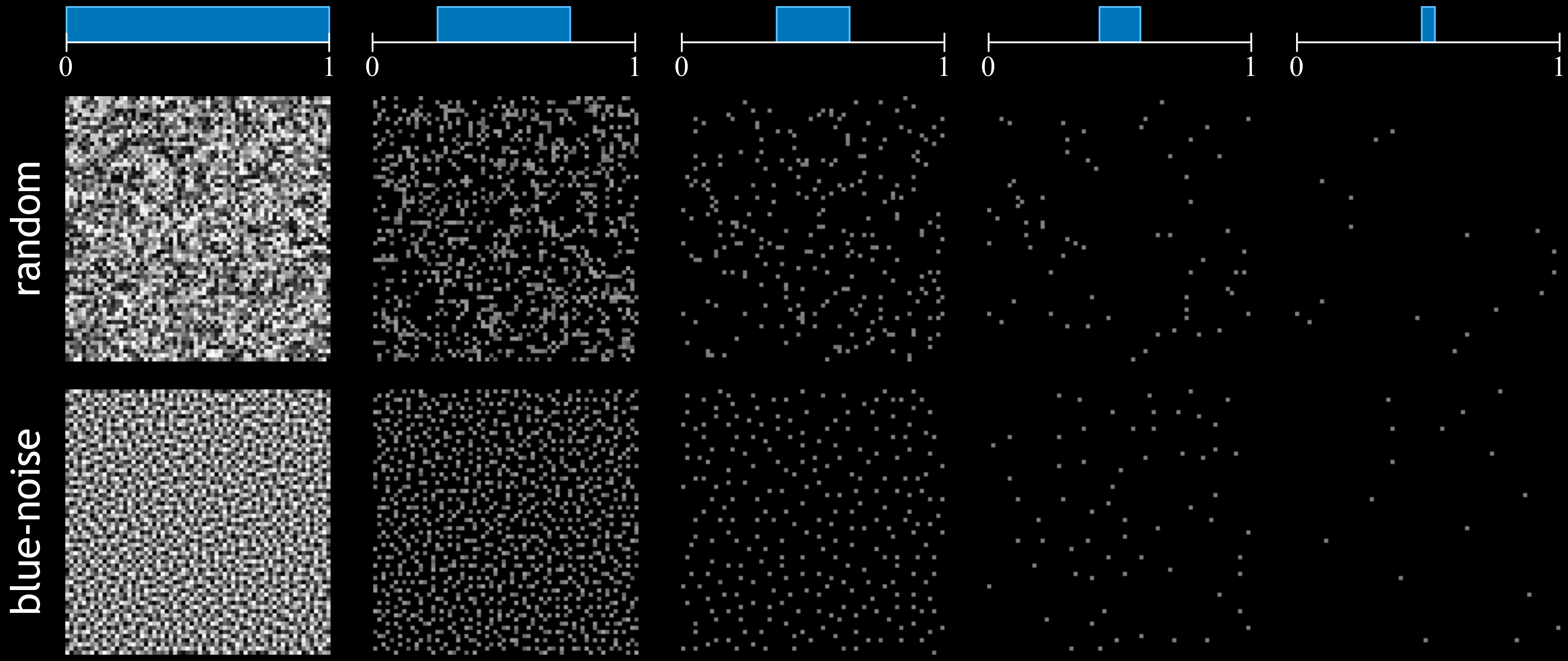
$$E(M) = \underbrace{\sum_{p \neq q}}_{\text{pixels}} E(p,q) = \sum_{p \neq q} \underbrace{\exp\left(-\frac{\overbrace{\|p_\mathrm{i} - q_\mathrm{i}\|^2}^{\text{pixel distance}}}{\sigma_\mathrm{i}^2}\right)}_{\text{image-space Gaussian}} \cdot \underbrace{\exp\left(-\frac{\overbrace{\|p_\mathrm{s} - q_\mathrm{s}\|^{d/2}}^{\text{sample distance}}}{\sigma_\mathrm{s}^2}\right)}_{\text{sample-space Gaussian}}$$

Dithered sampling: mask construction

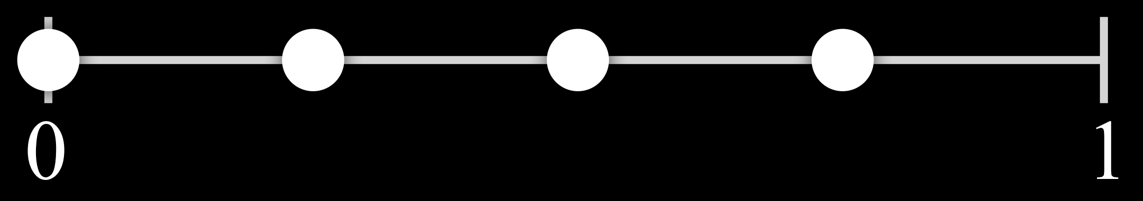
$$E(M) = \sum_{p \neq q} E(p, q) = \sum_{p \neq q} \exp\left(-\frac{\|p_\mathrm{i} - q_\mathrm{i}\|^2}{\sigma_\mathrm{i}^2} \right) \cdot \exp\left(-\frac{\|p_\mathrm{s} - q_\mathrm{s}\|^{d/2}}{\sigma_\mathrm{s}^2} \right)$$



Dithered sampling: mask sample distribution



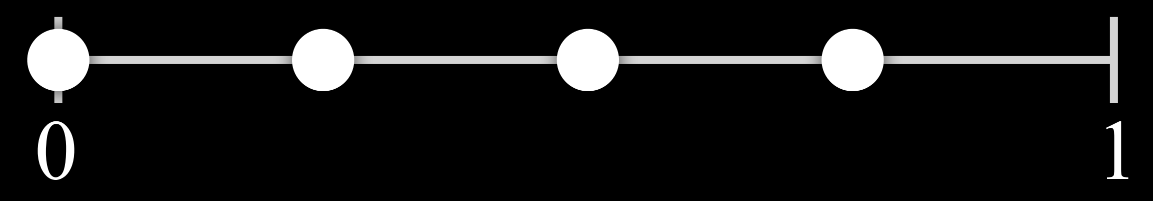
Dithered sampling: pattern offsetting

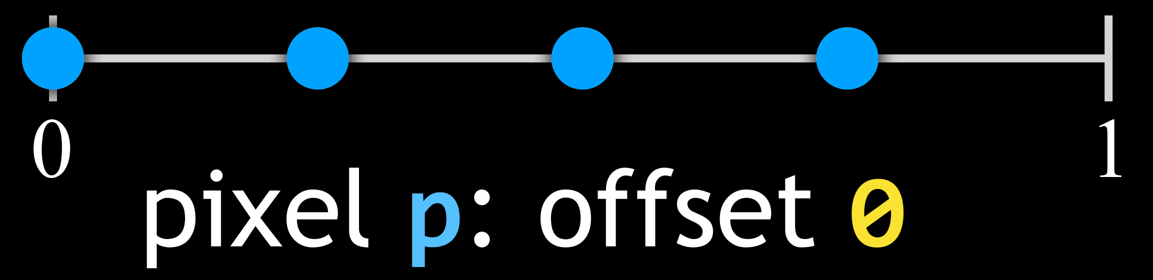


1D

input pattern

Dithered sampling: pattern offsetting



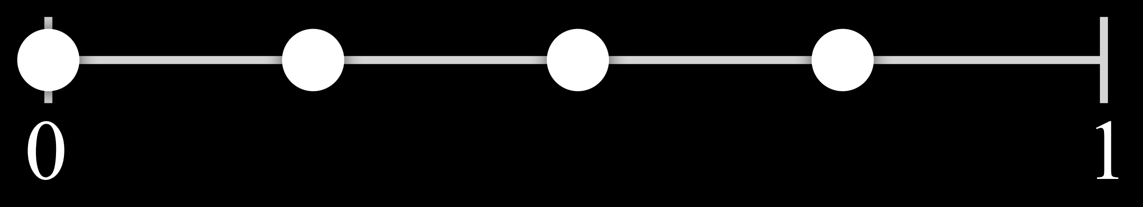


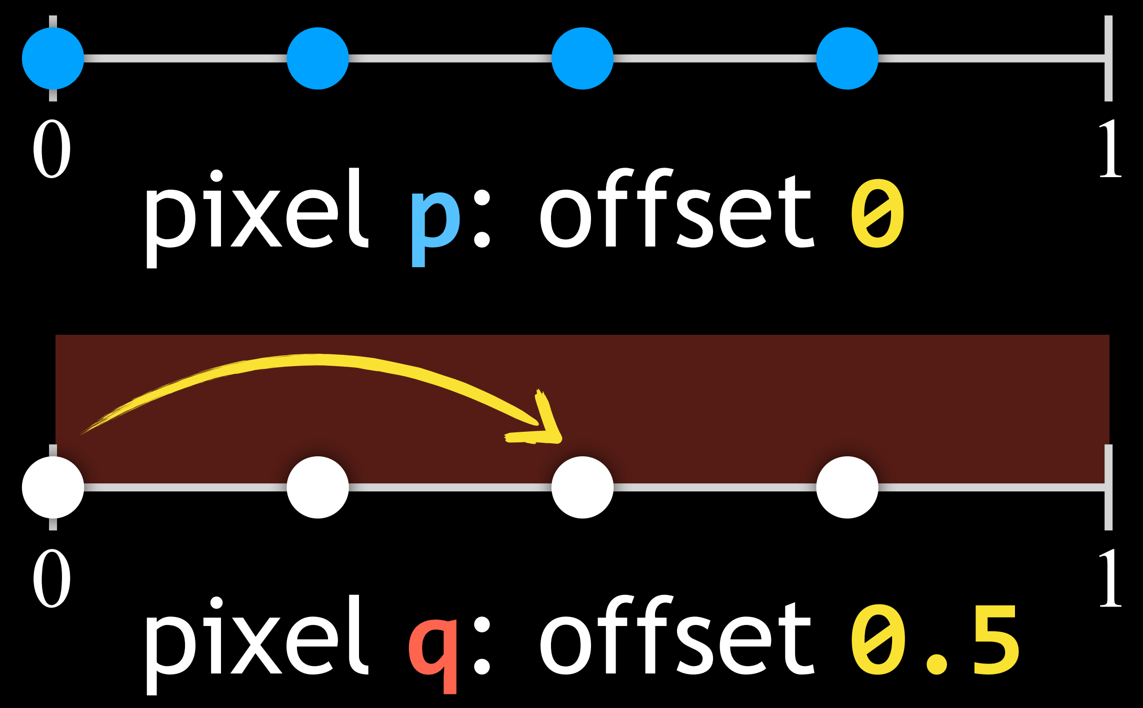
1D

input pattern

global offsetting

Dithered sampling: pattern offsetting

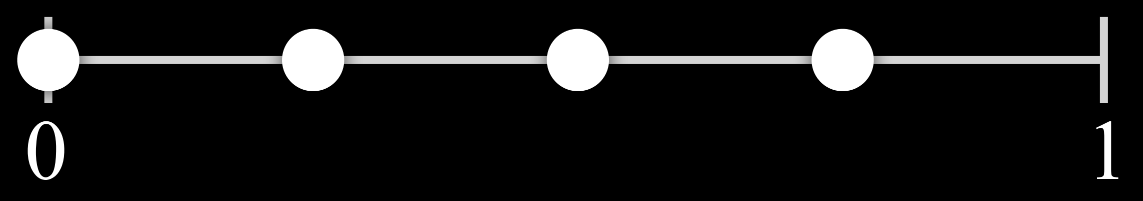


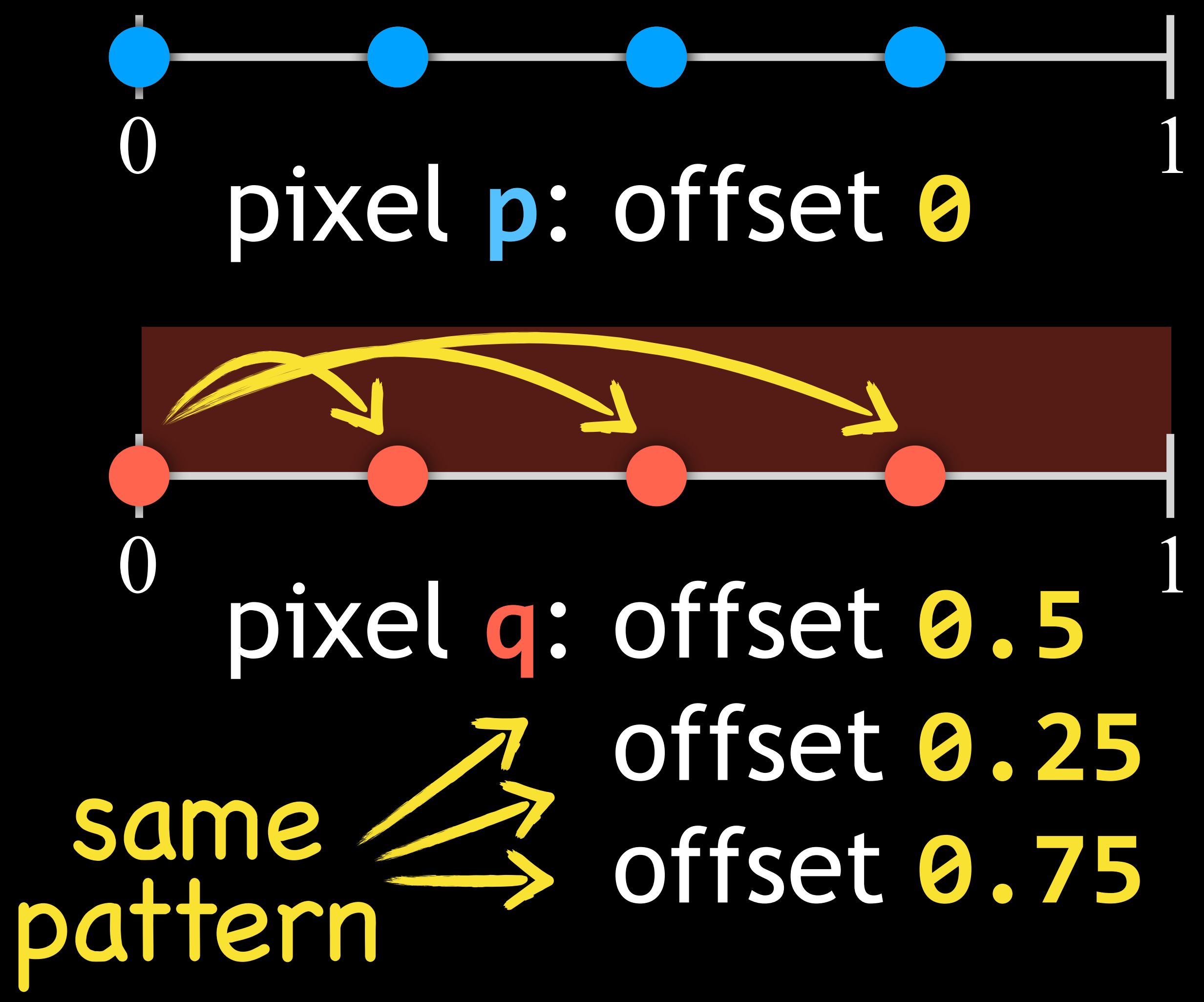


1D

input pattern	global offsetting

Dithered sampling: pattern offsetting



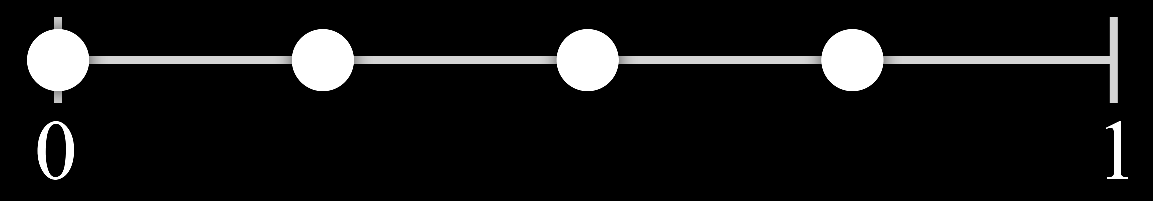


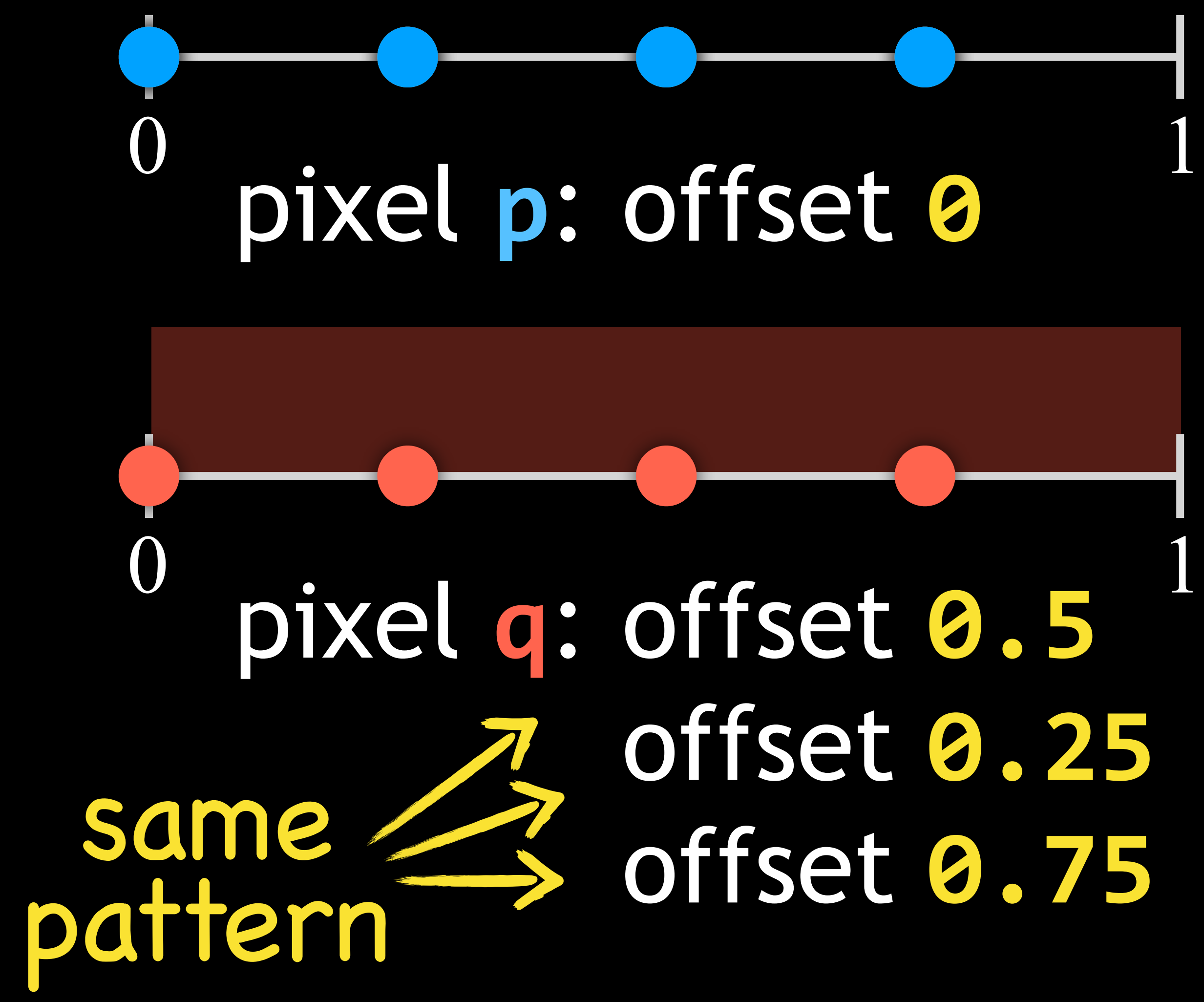
1D

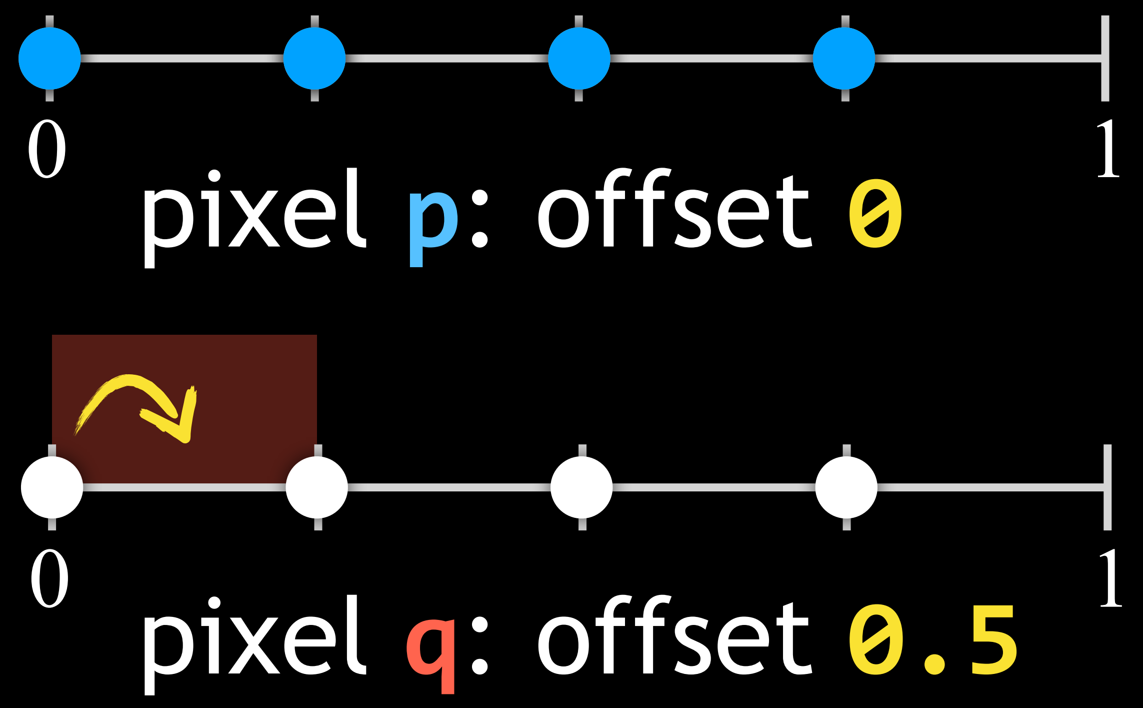
input pattern

global offsetting

Dithered sampling: pattern offsetting



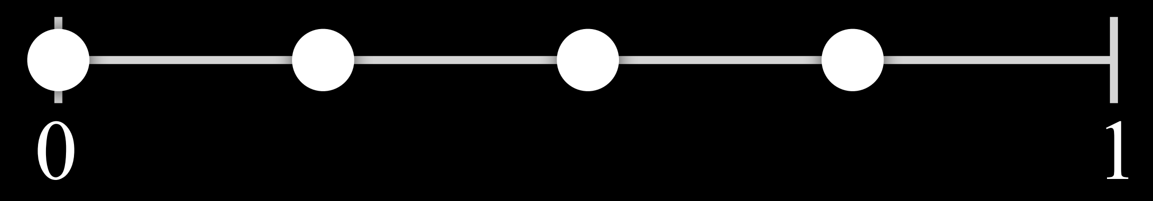




1D

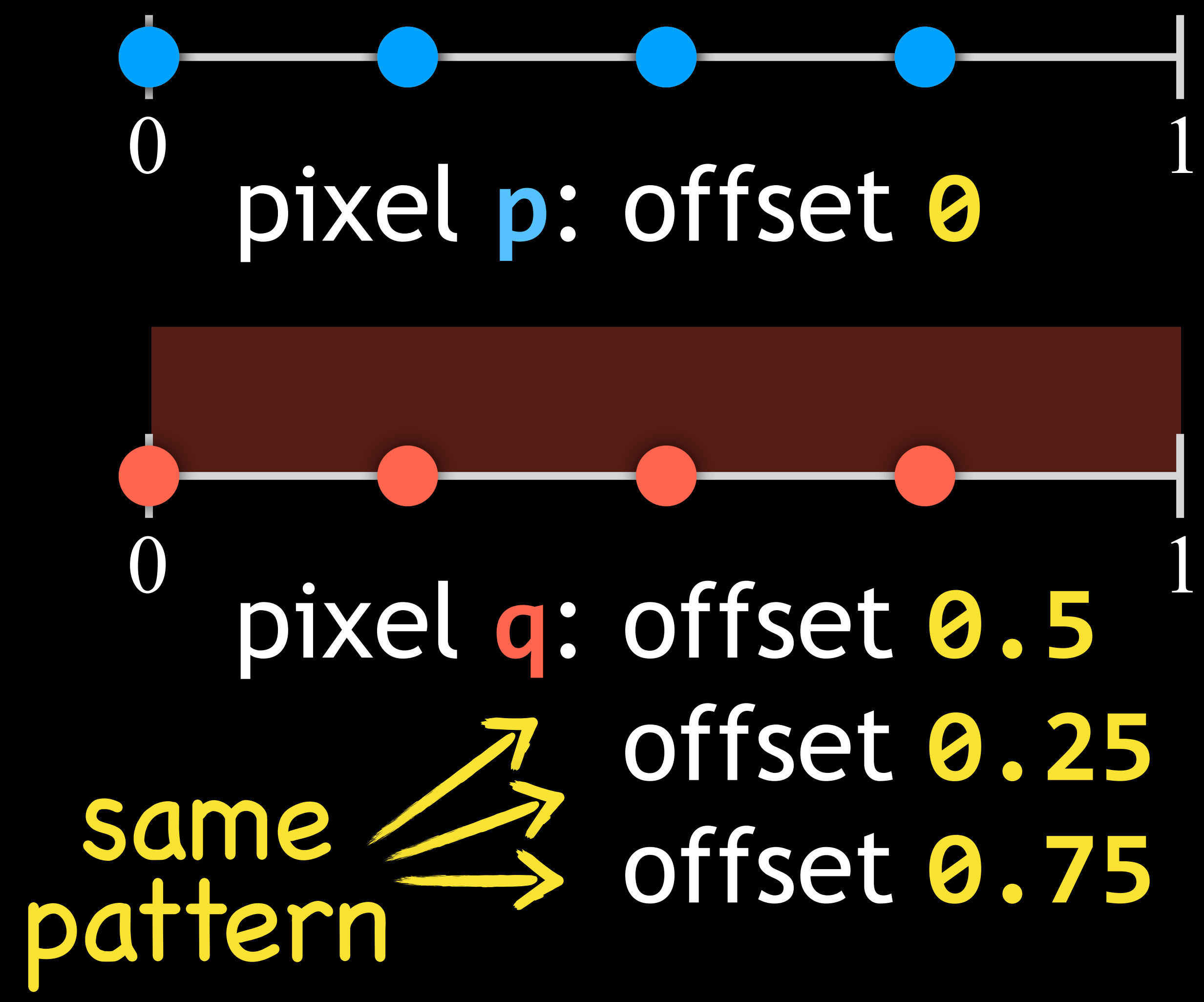
input pattern | global offsetting | stratified offsetting

Dithered sampling: pattern offsetting

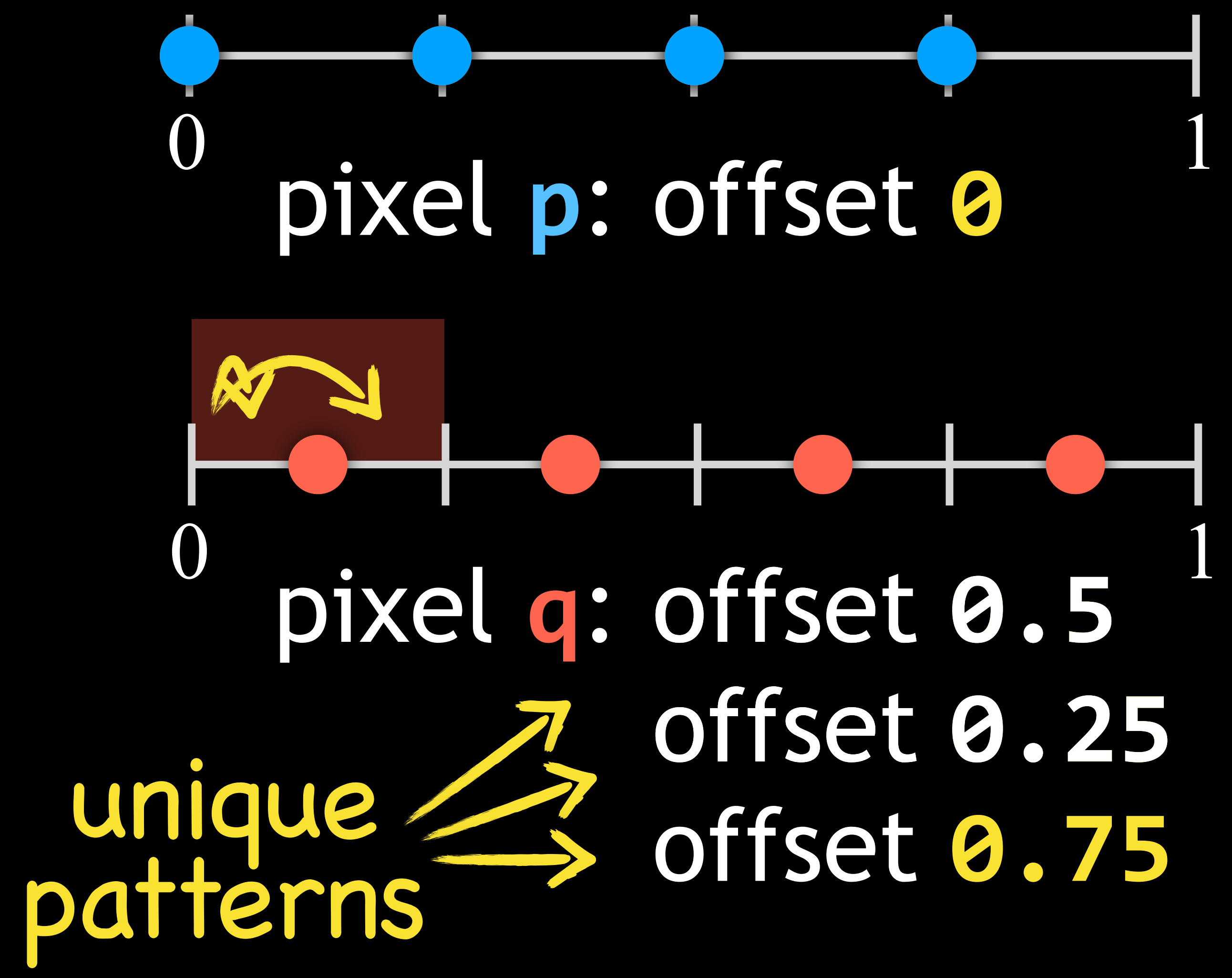


1D

input pattern

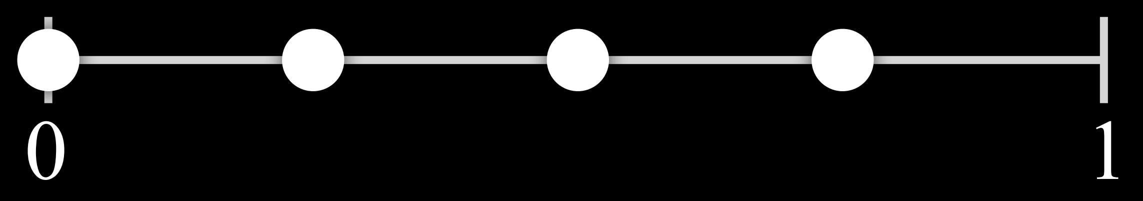


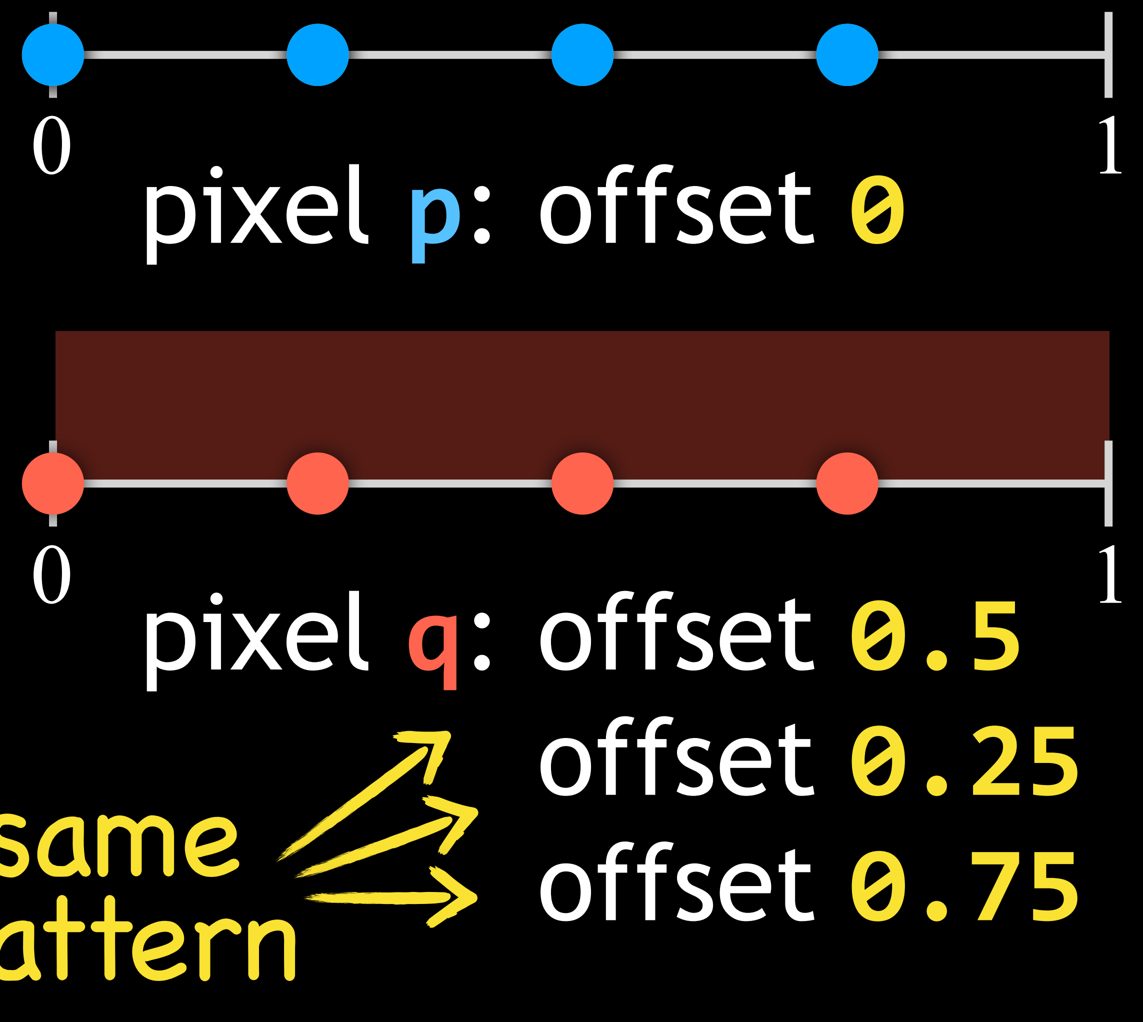
global offsetting

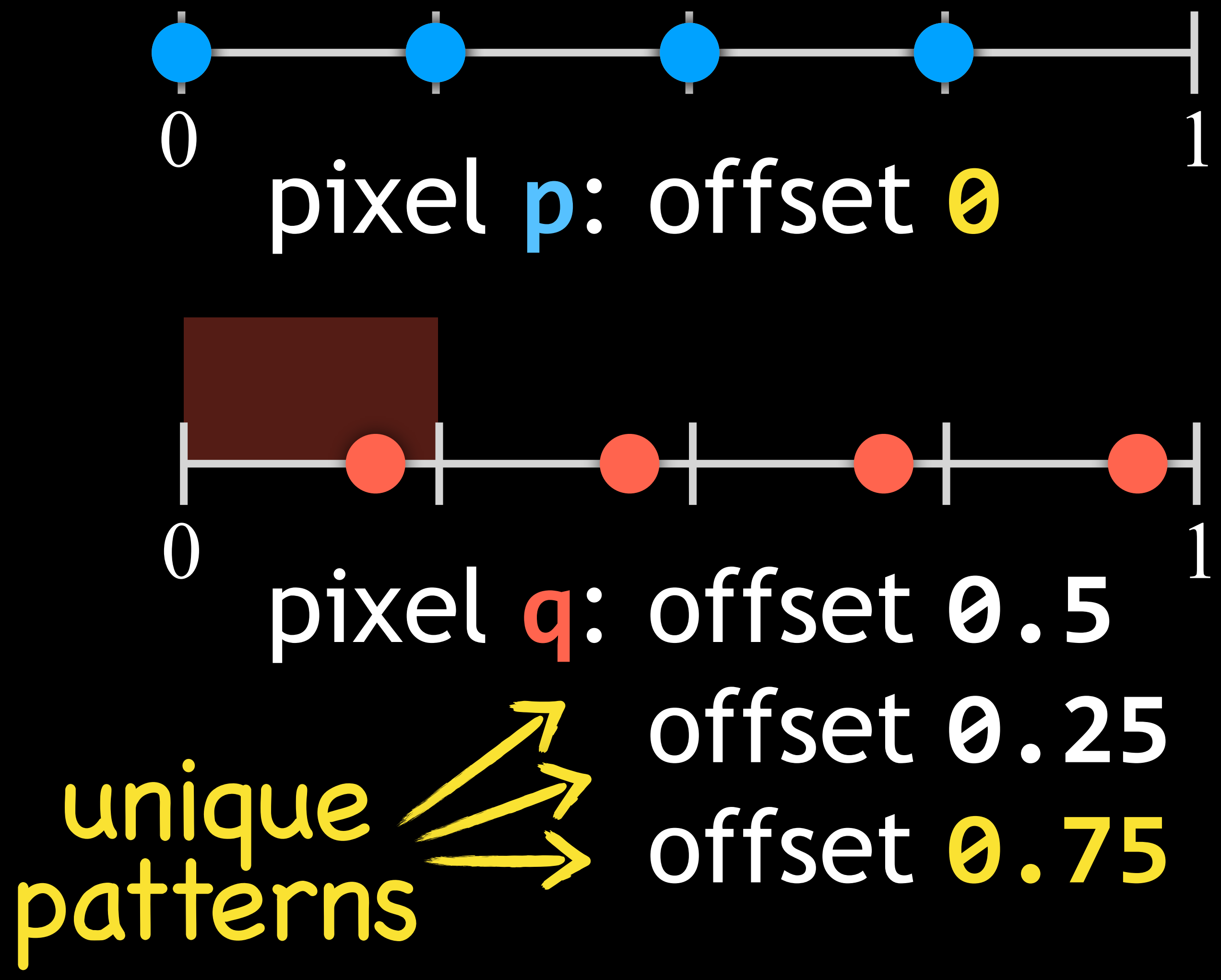


stratified offsetting

Dithered sampling: pattern offsetting



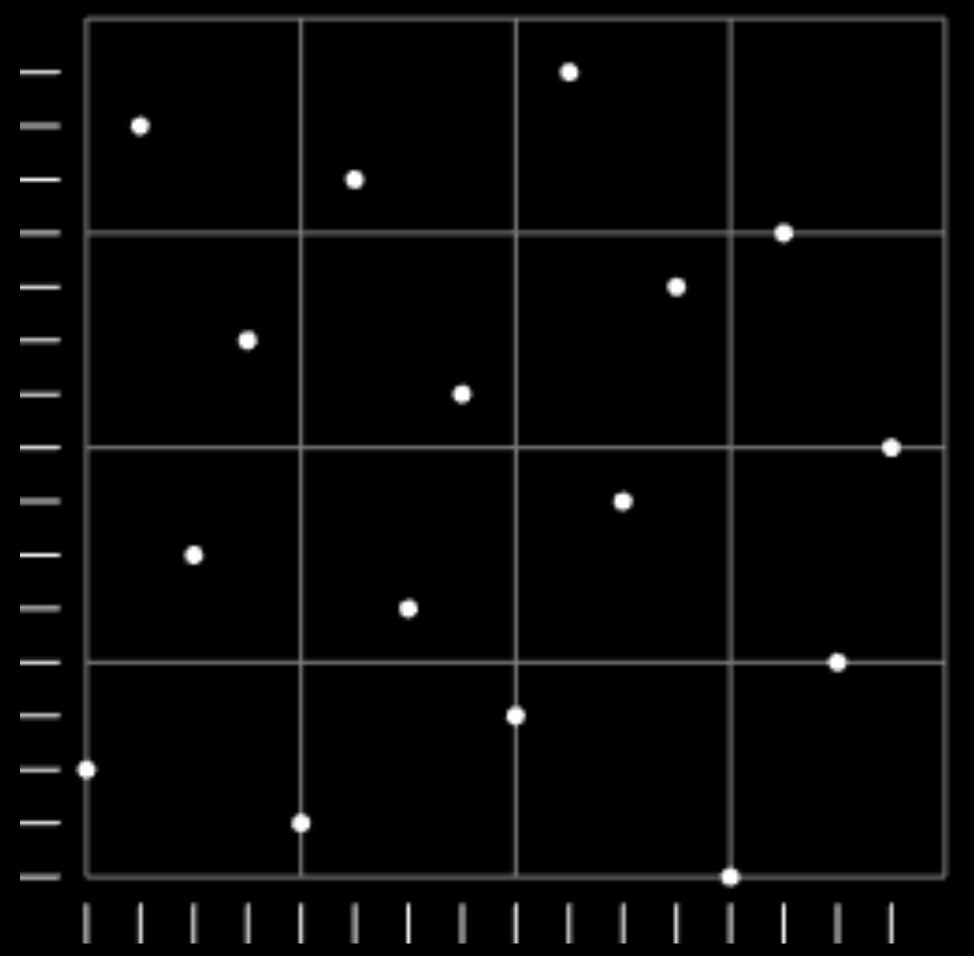




1D

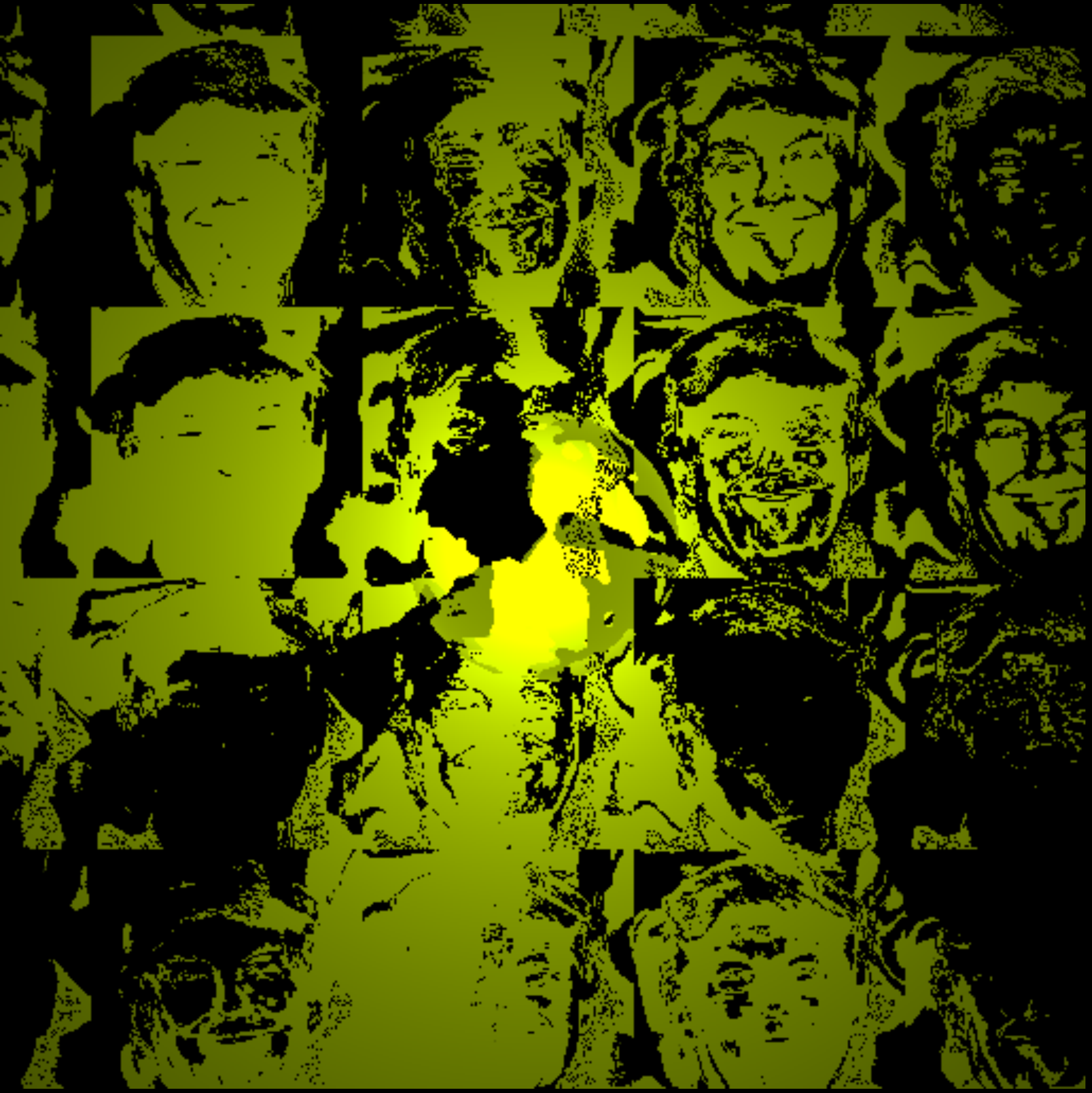
input pattern | global offsetting | stratified offsetting

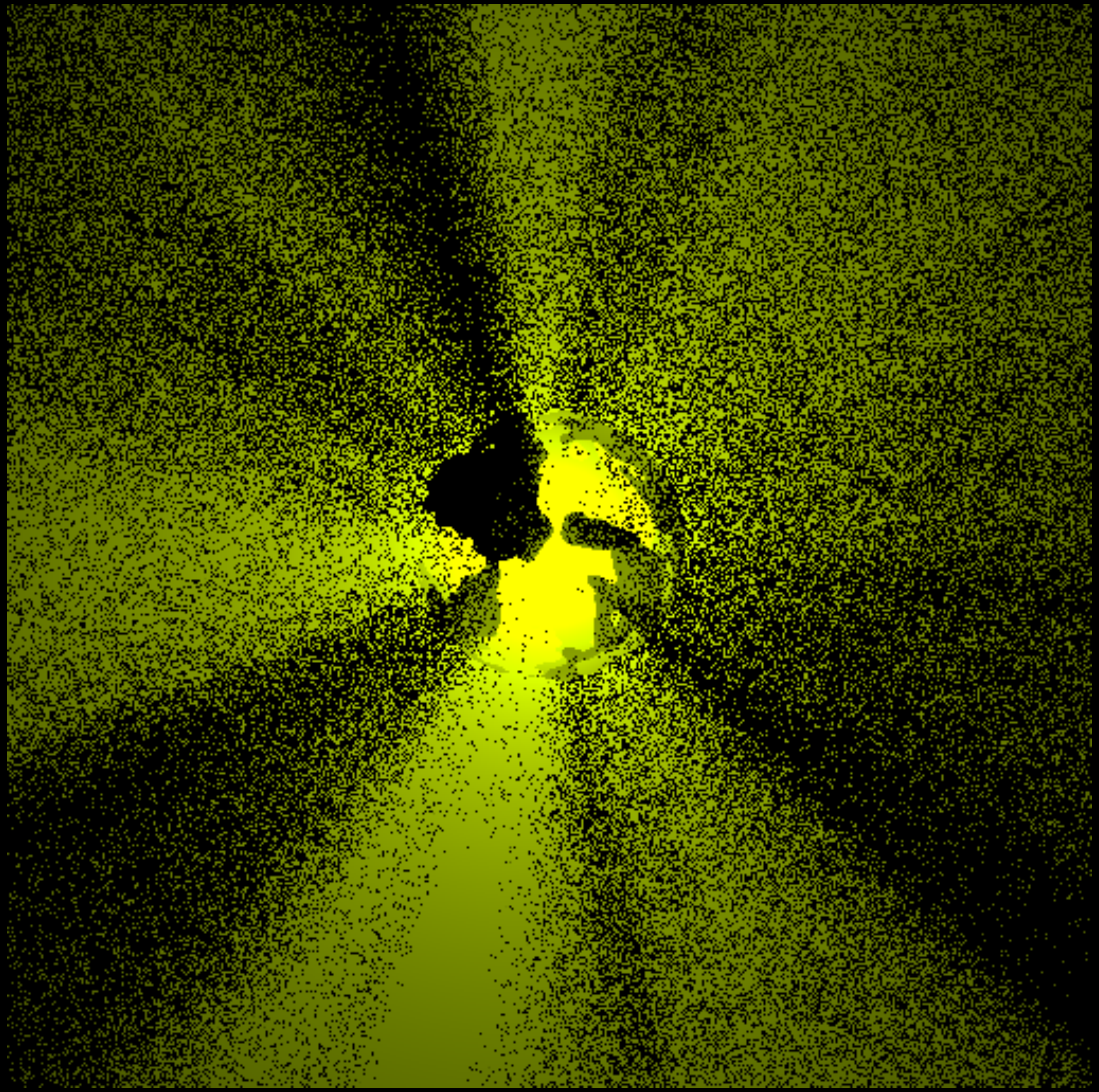
2D

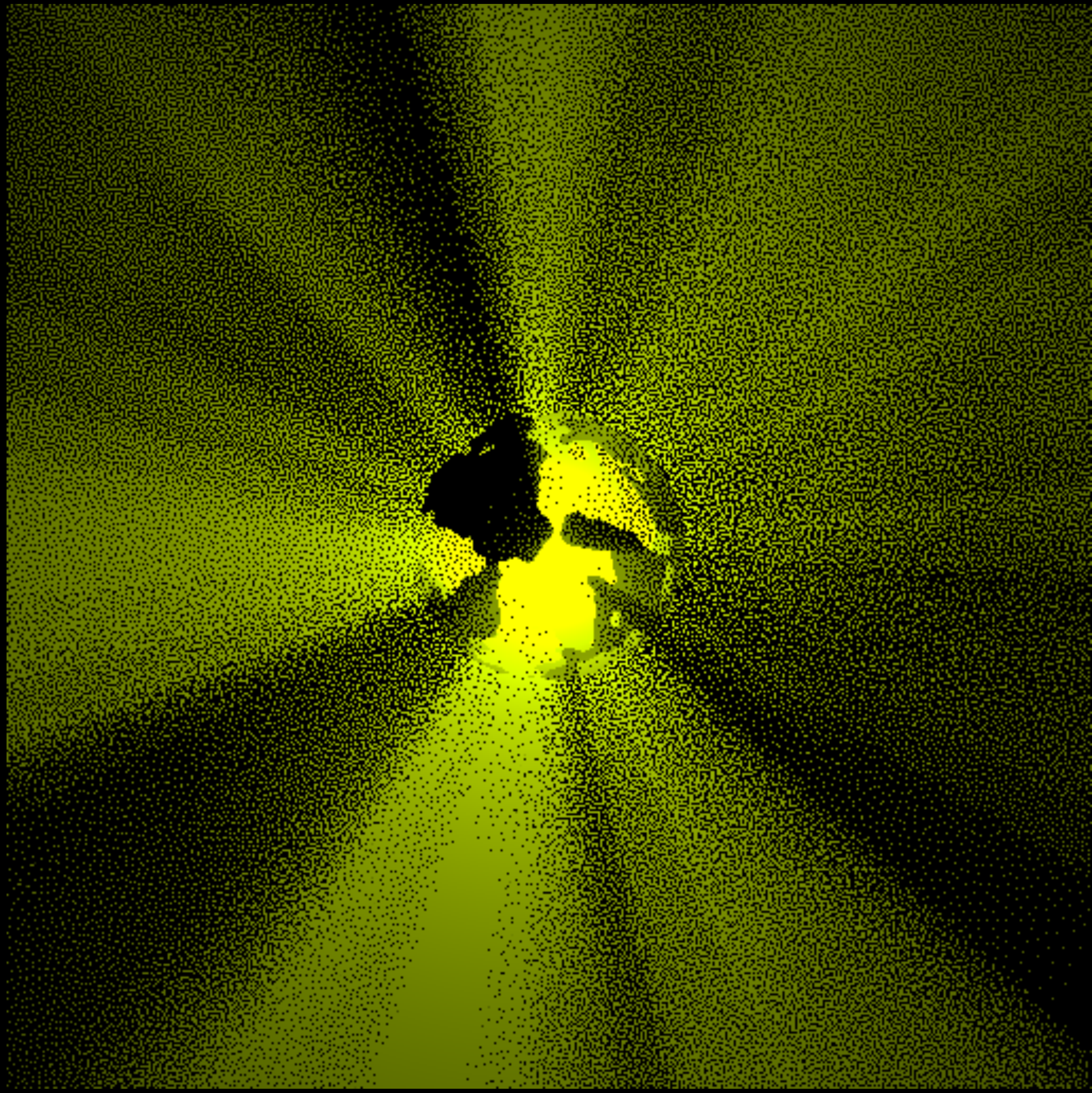


Dithered sampling: results (1D sampling)

1 spp

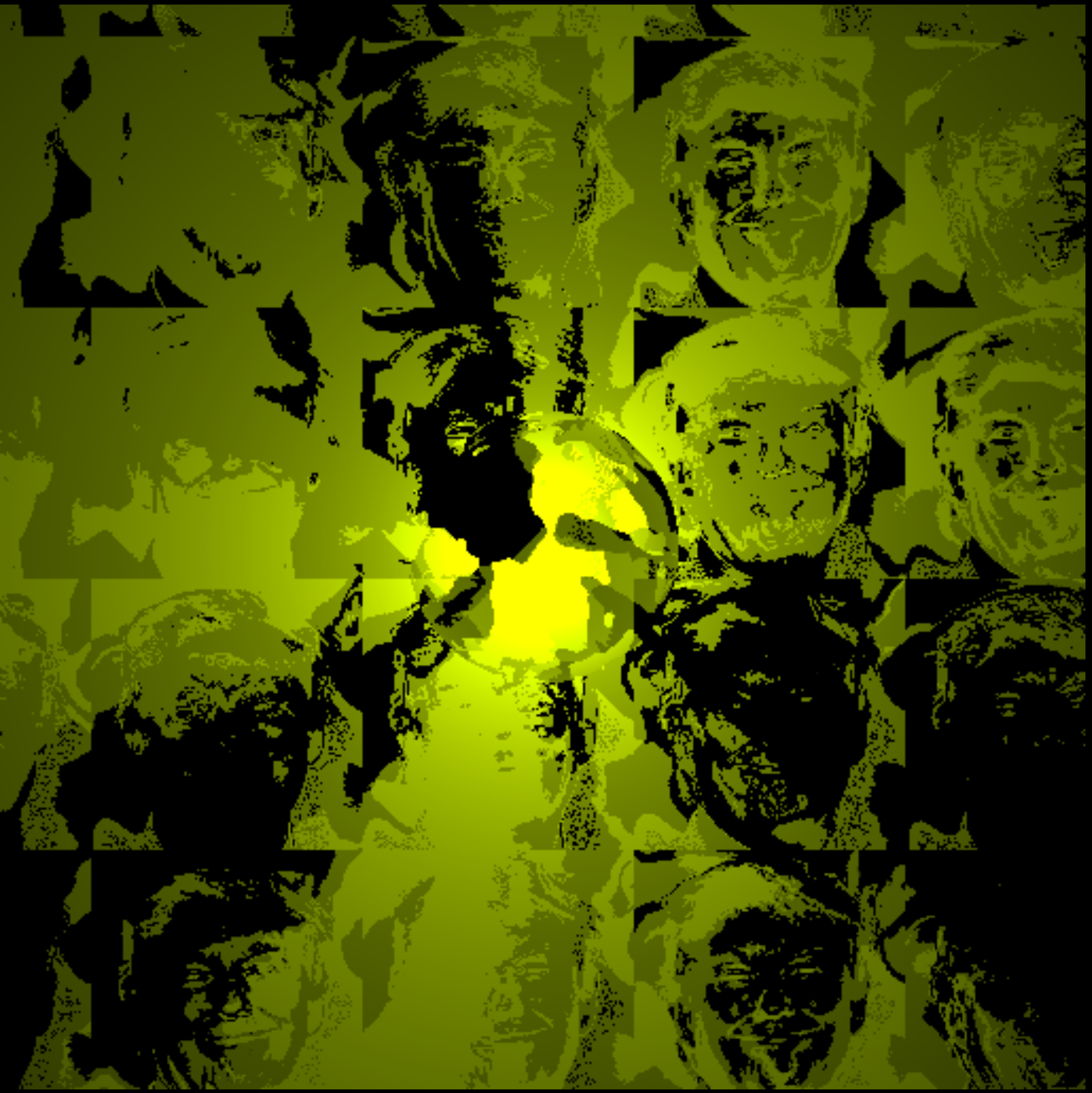

low-frequency

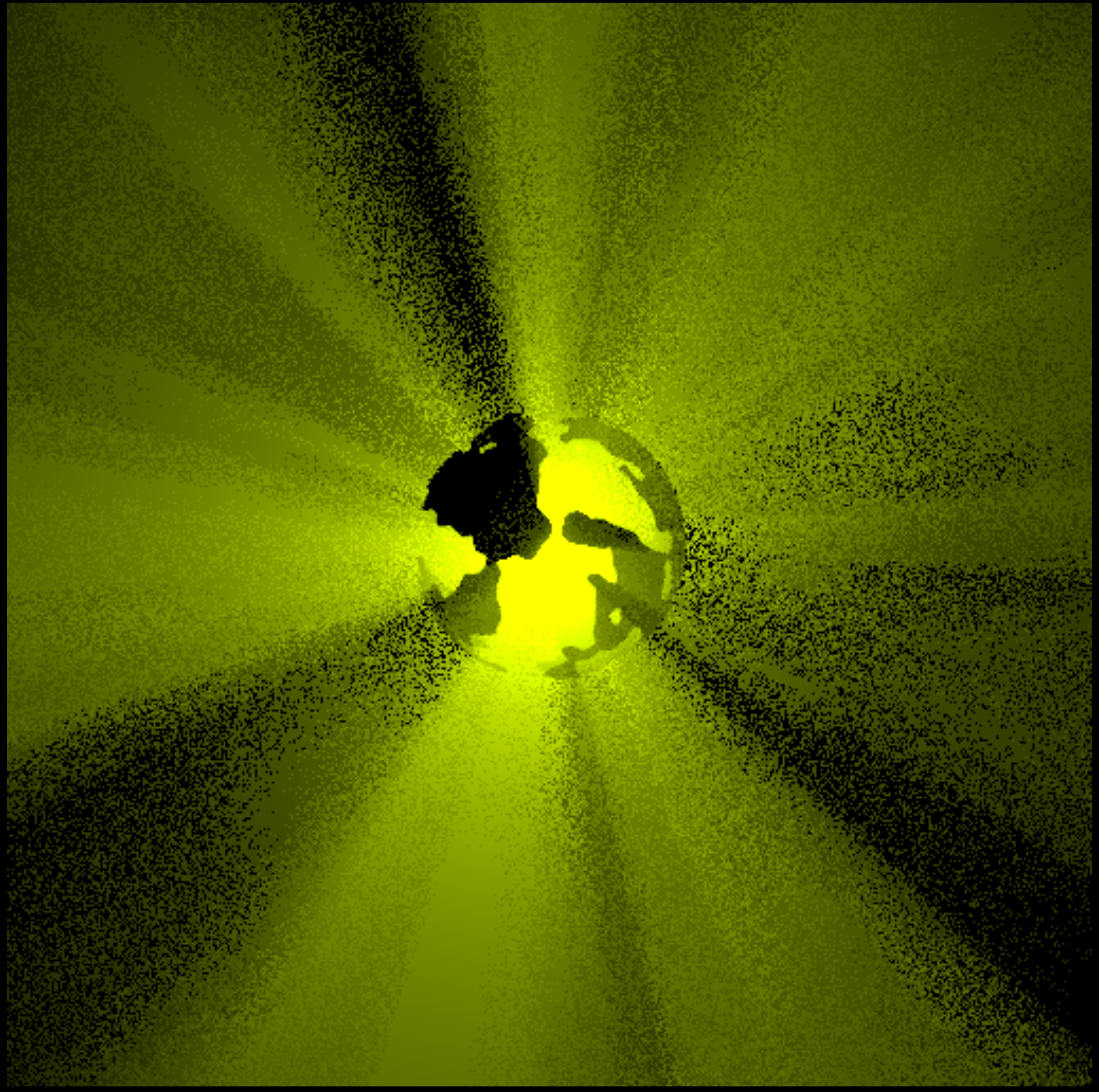

all-frequency

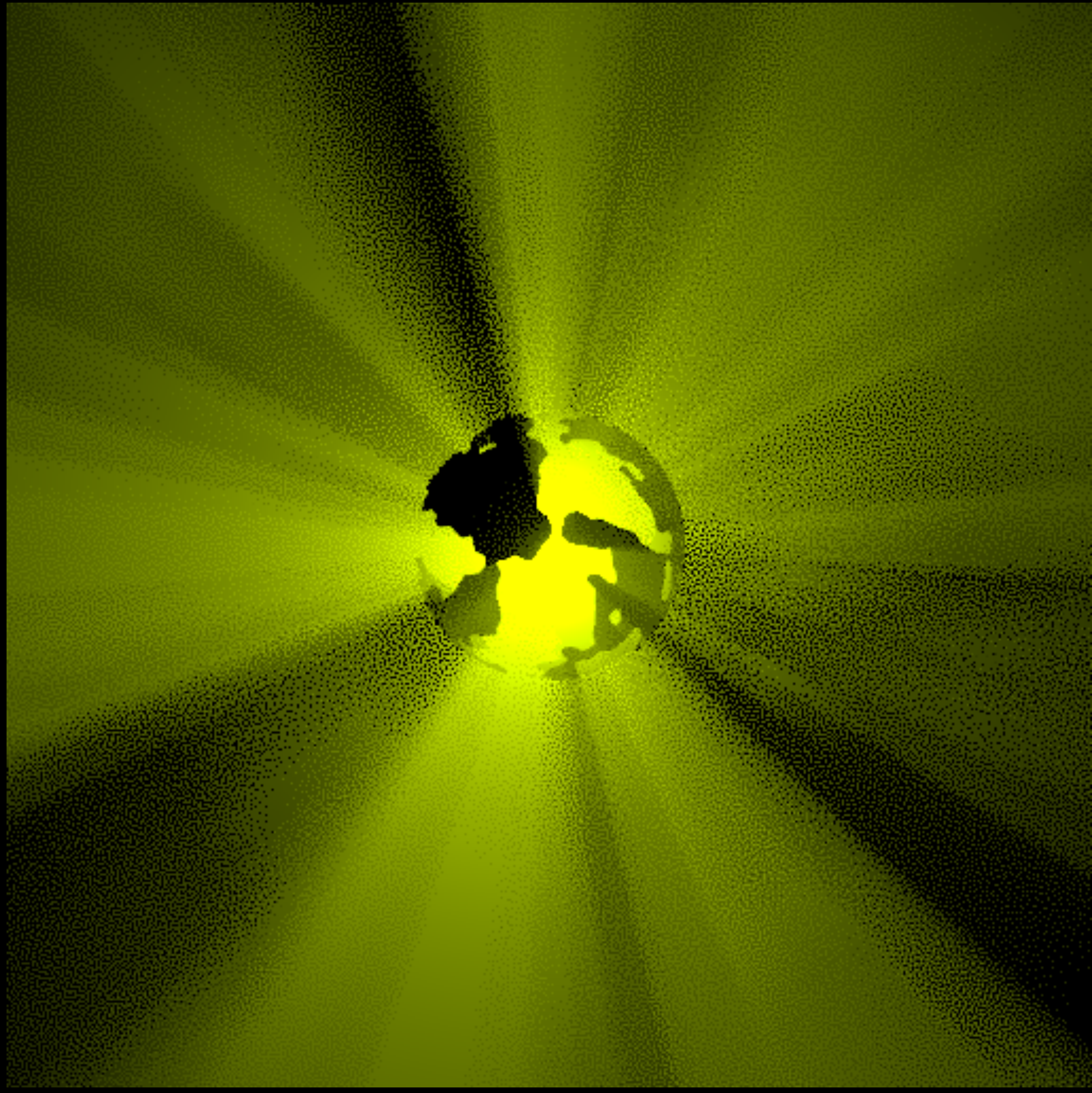

high-frequency

Dithered sampling: results (1D sampling)

4 spp

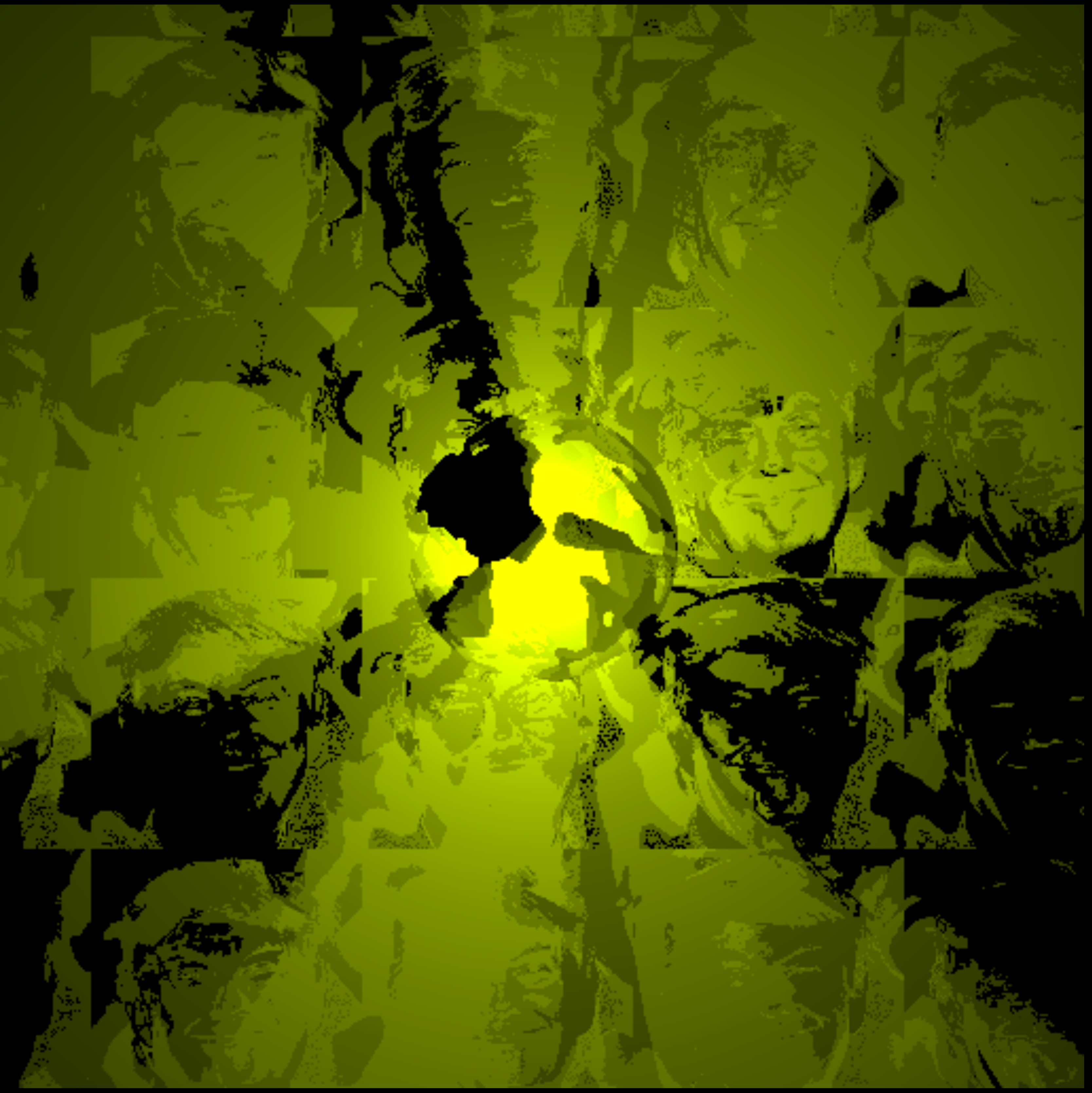

low-frequency

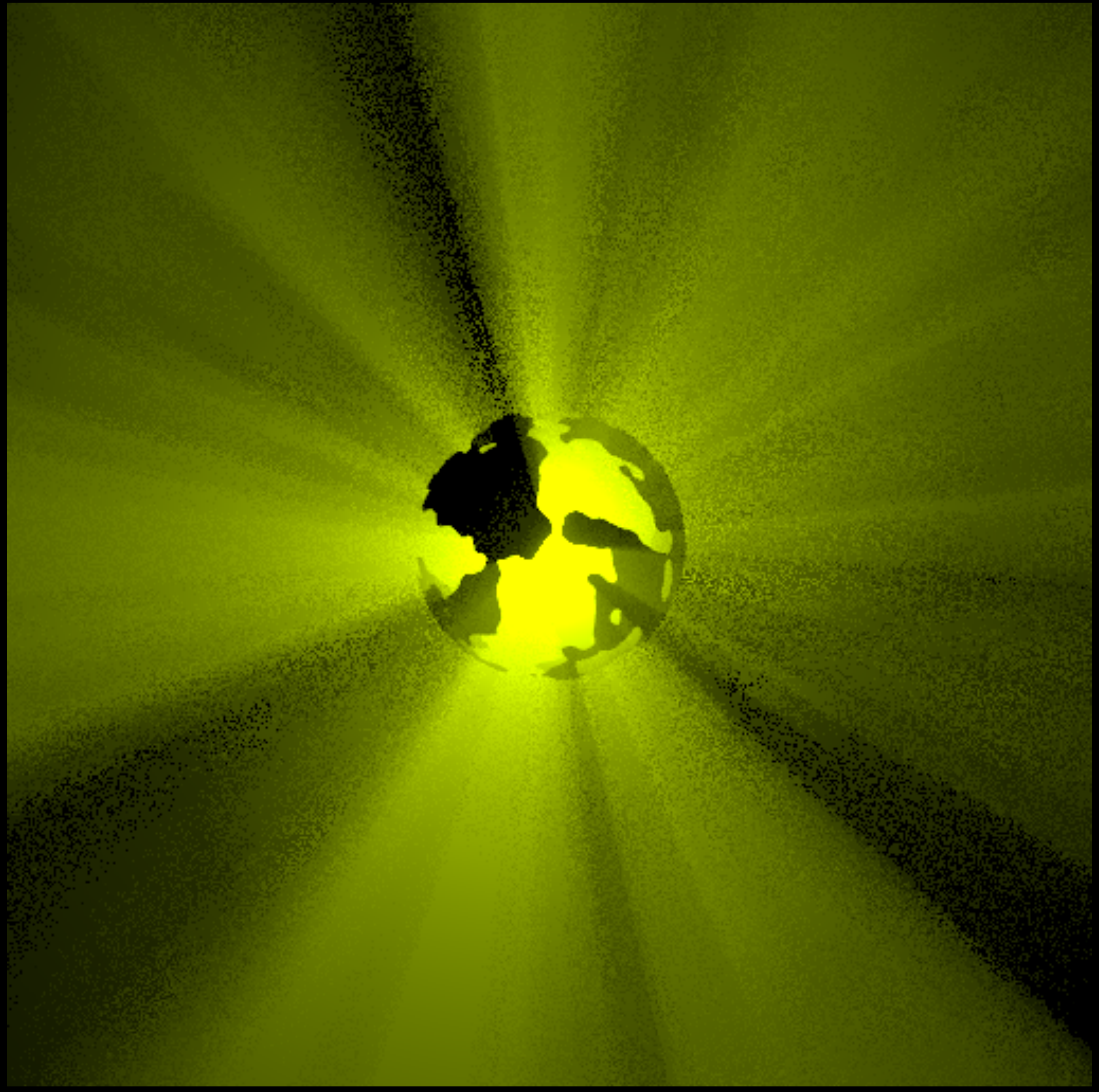

all-frequency

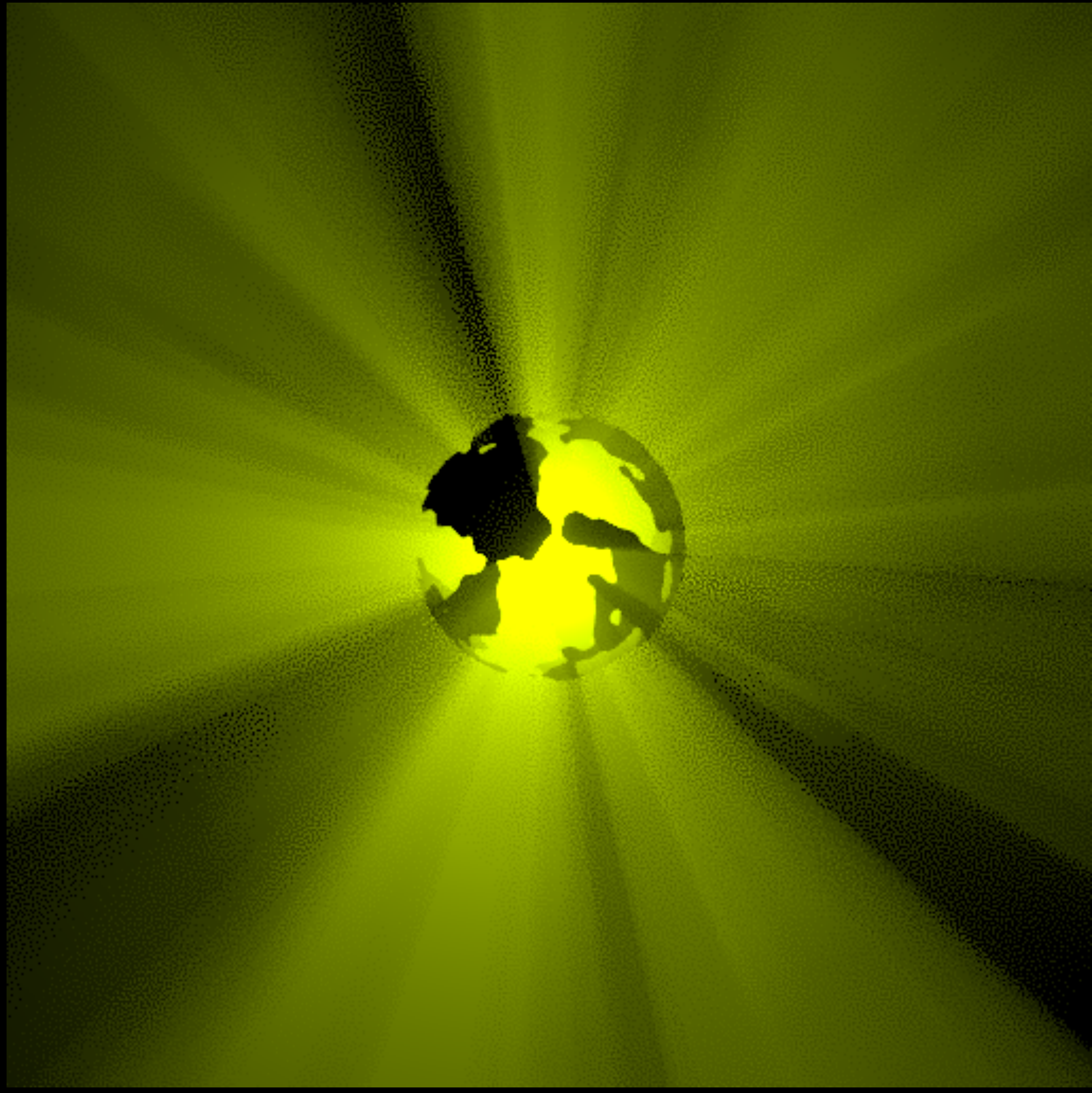

high-frequency

Dithered sampling: results (1D sampling)

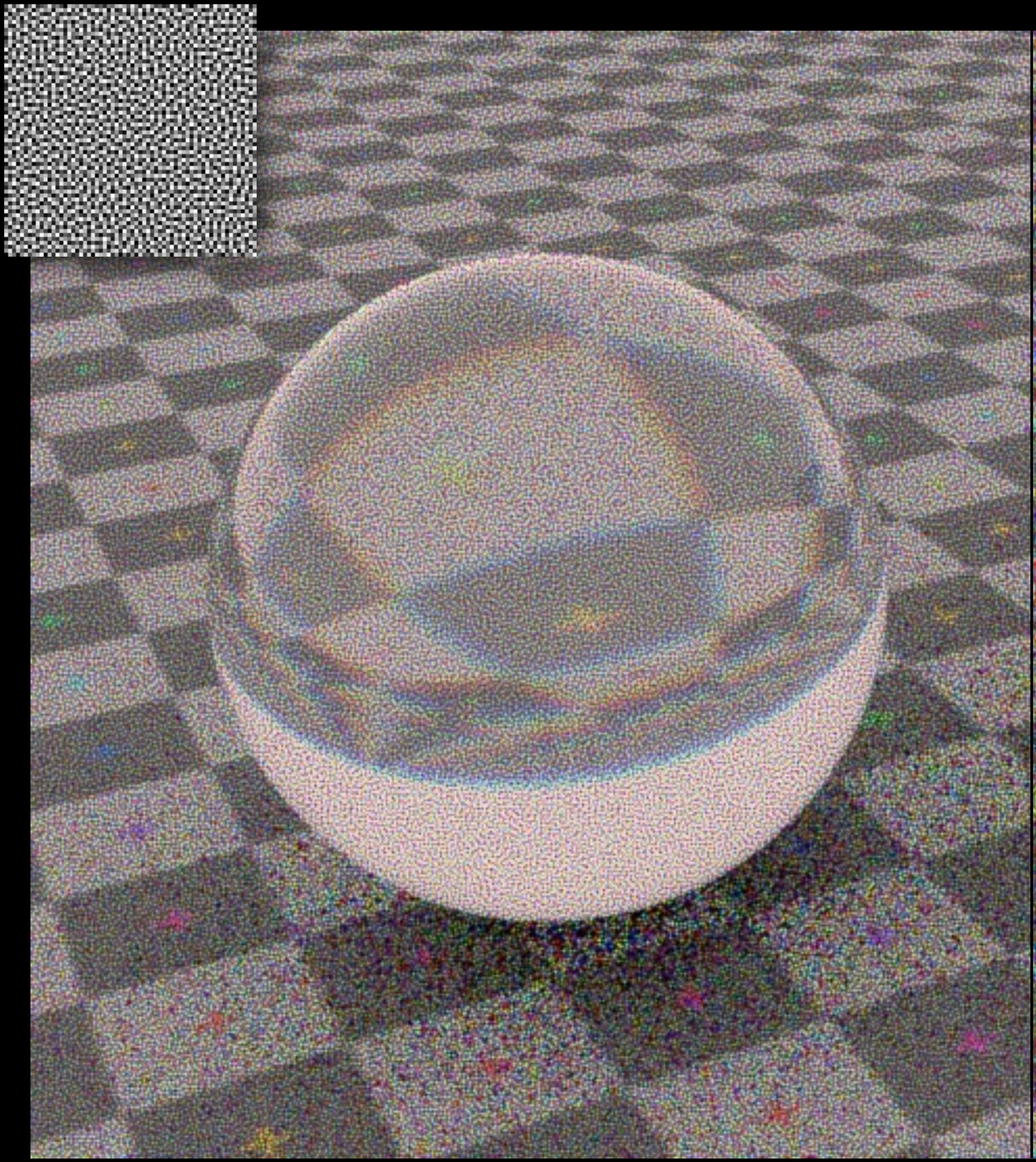
9 spp

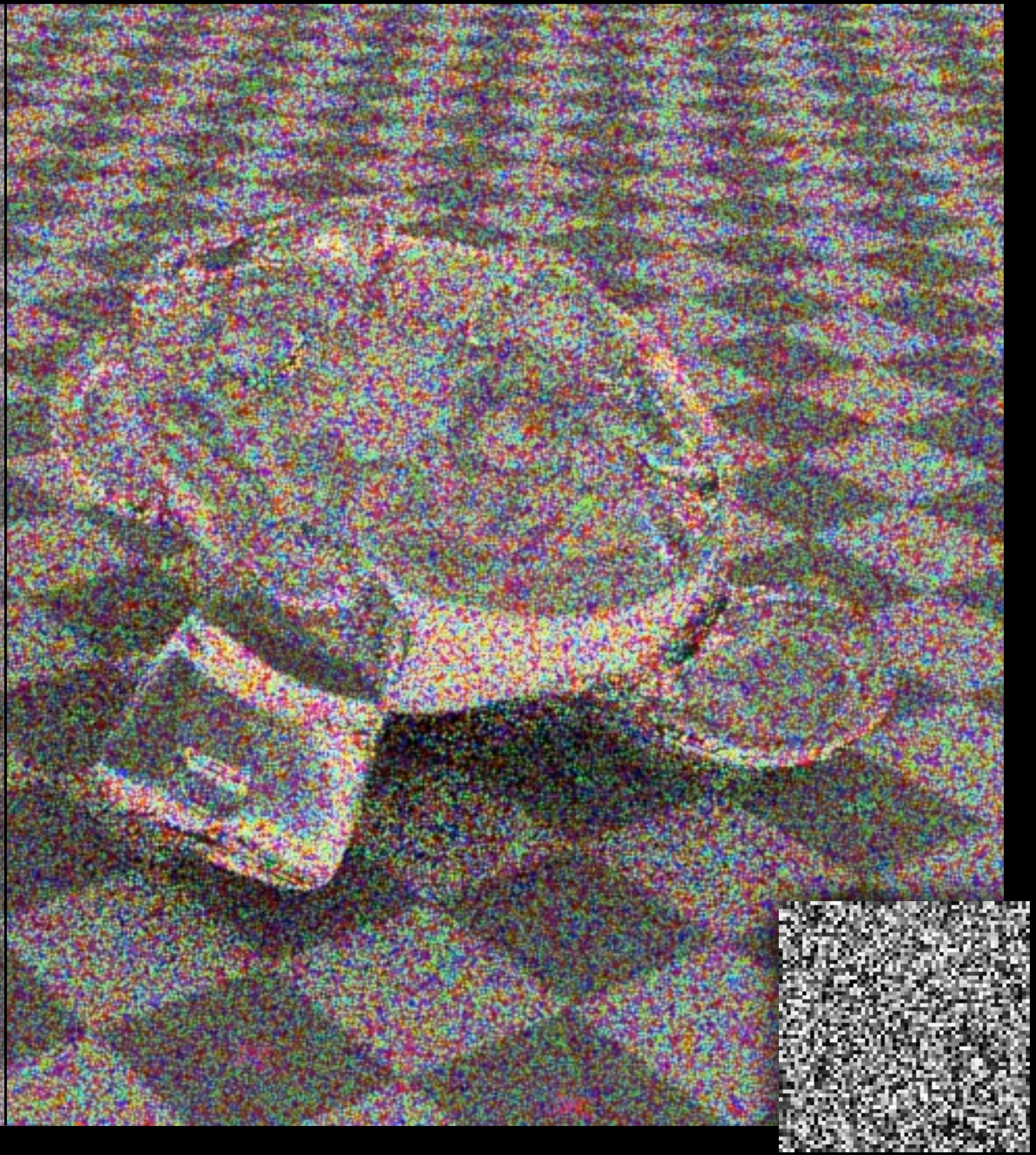

low-frequency


all-frequency


high-frequency

Dithered sampling: results (1D sampling)





Dithered sampling: results (2D sampling)

random
1
2
3
1 spp

Dithered sampling: results (2D sampling)

blue-noise
1
2
3
1 spp

Dithered sampling: results (2D sampling)

random
1
2
3
4 spp

Dithered sampling: results (2D sampling)

blue-noise
1
2
3
4 spp

Dithered sampling: results (2D sampling)

random
1
2
3
9 spp

Dithered sampling: results (2D sampling)

blue-noise
1
2
3
9 spp

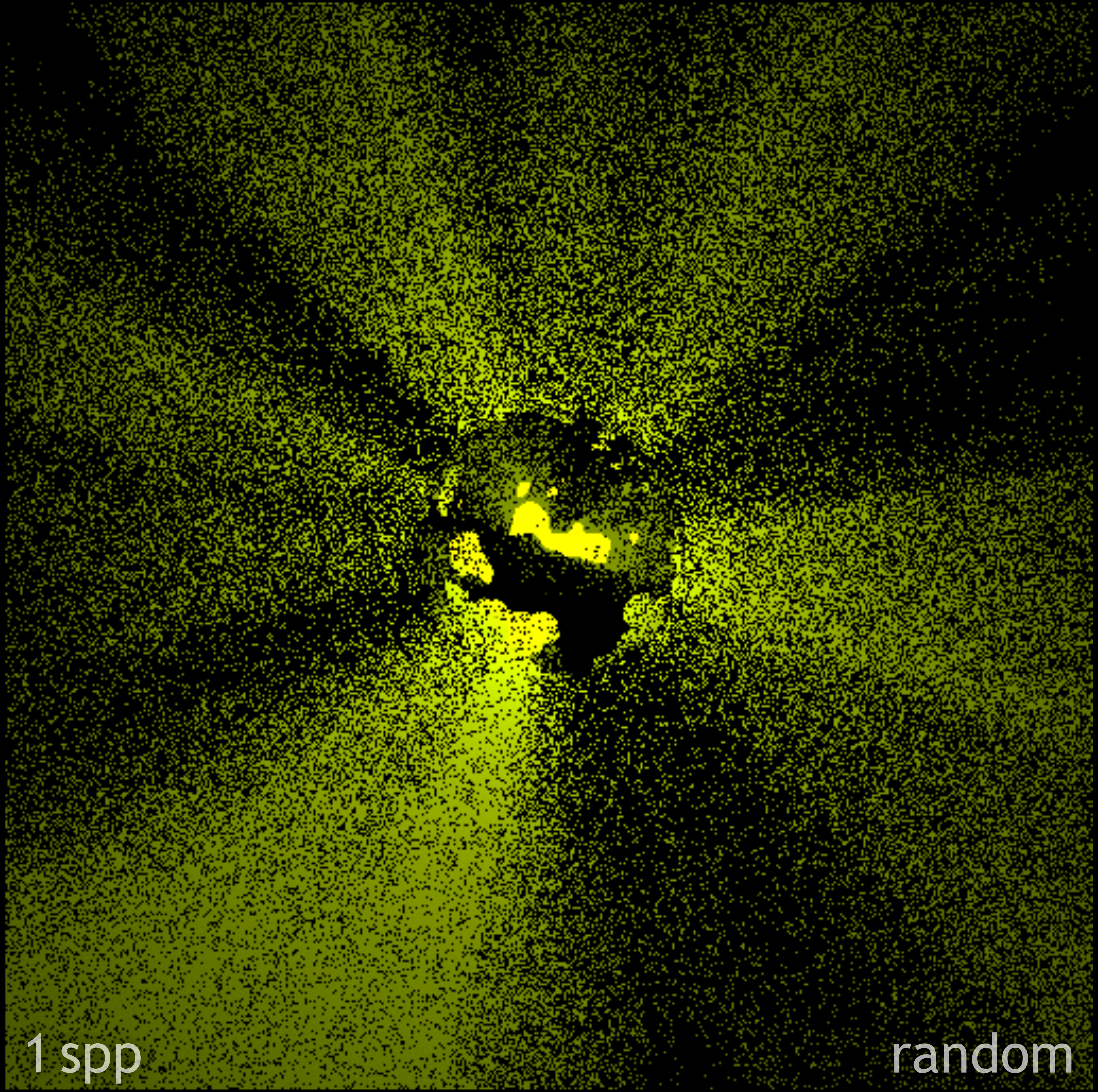
Dithered sampling: results (2D sampling)

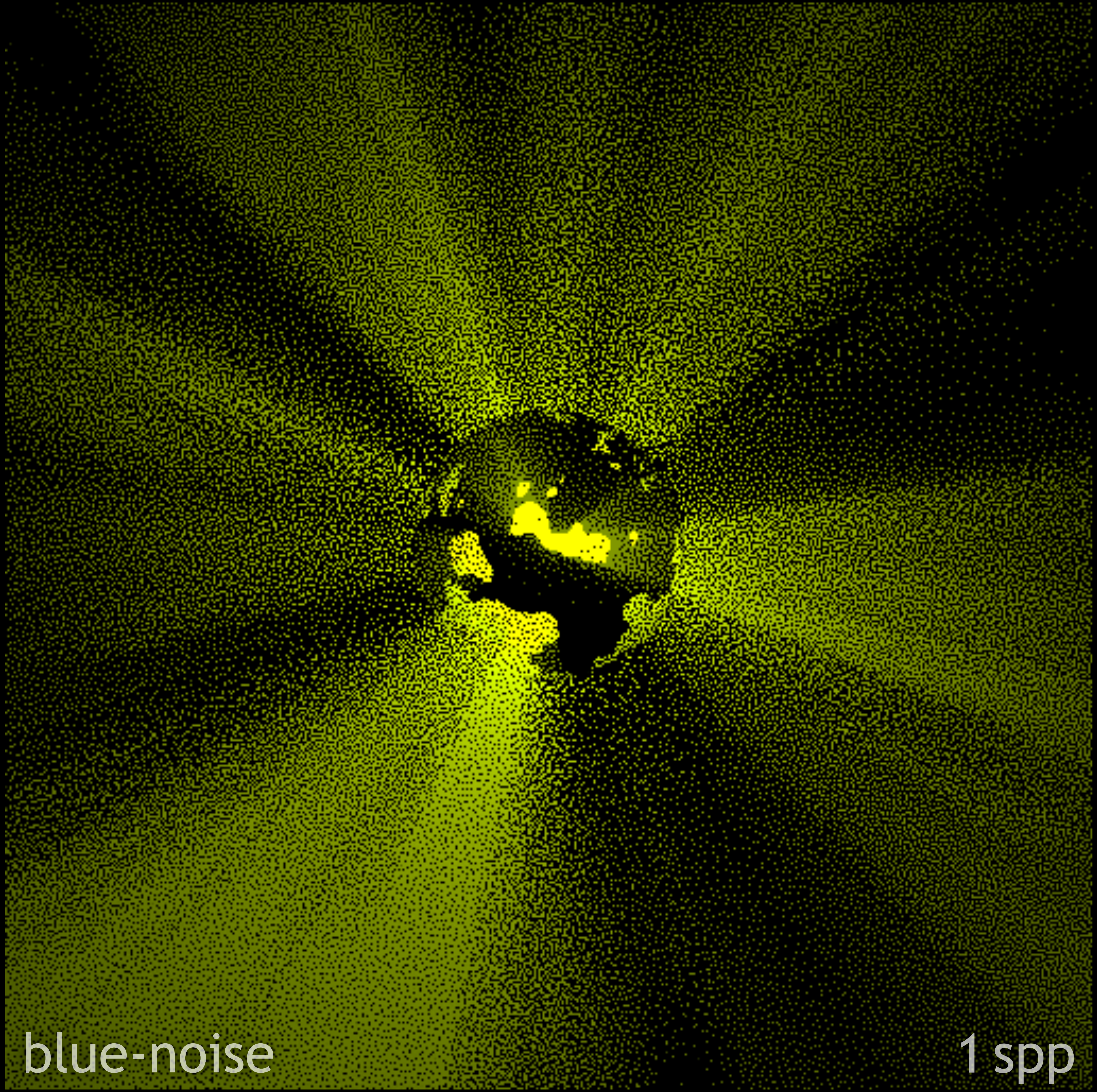
random
1
2
3
16 spp

Dithered sampling: results (2D sampling)

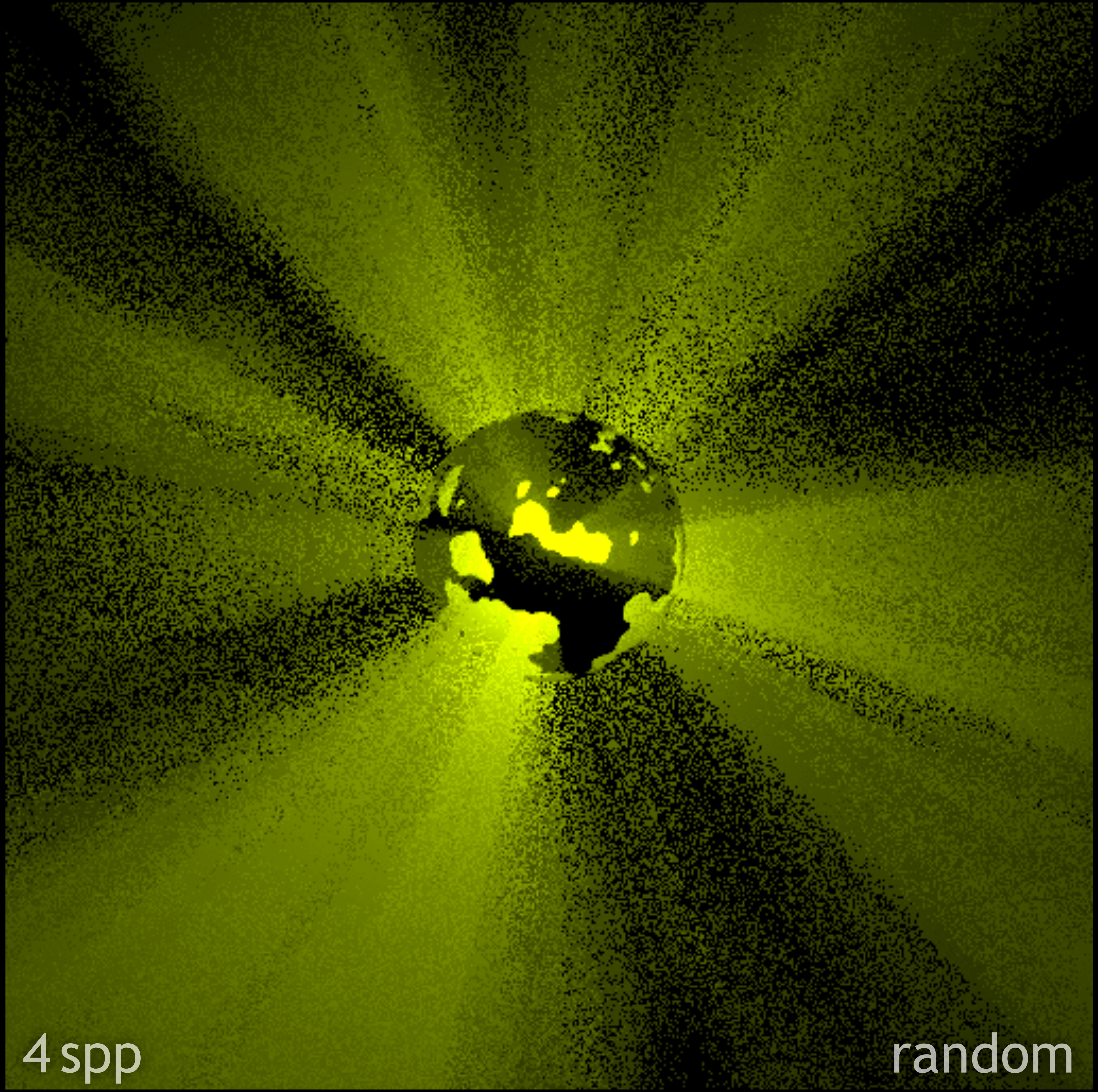
blue-noise
1
2
3
16 spp

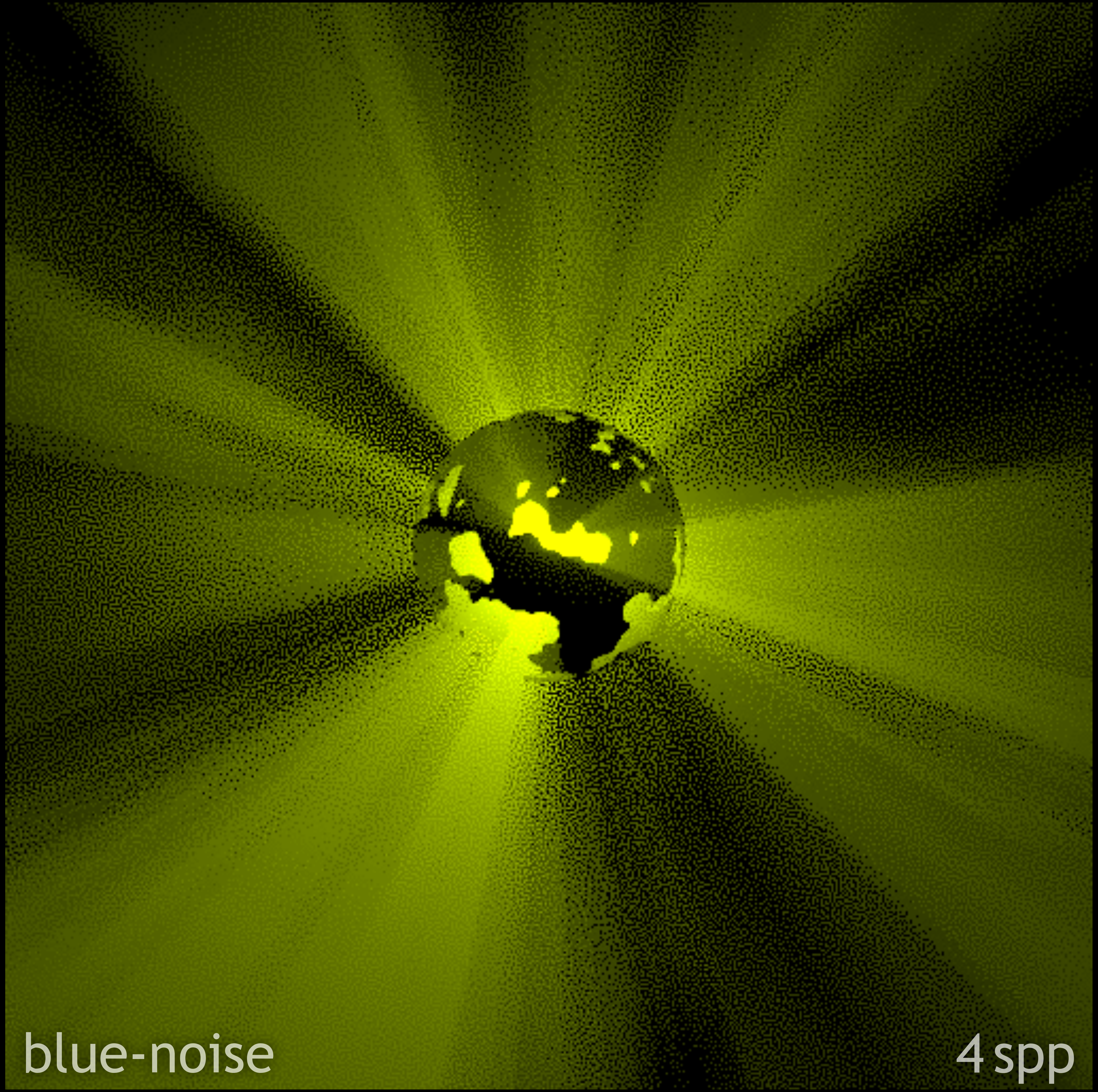
Dithered sampling: animation (1D sampling)



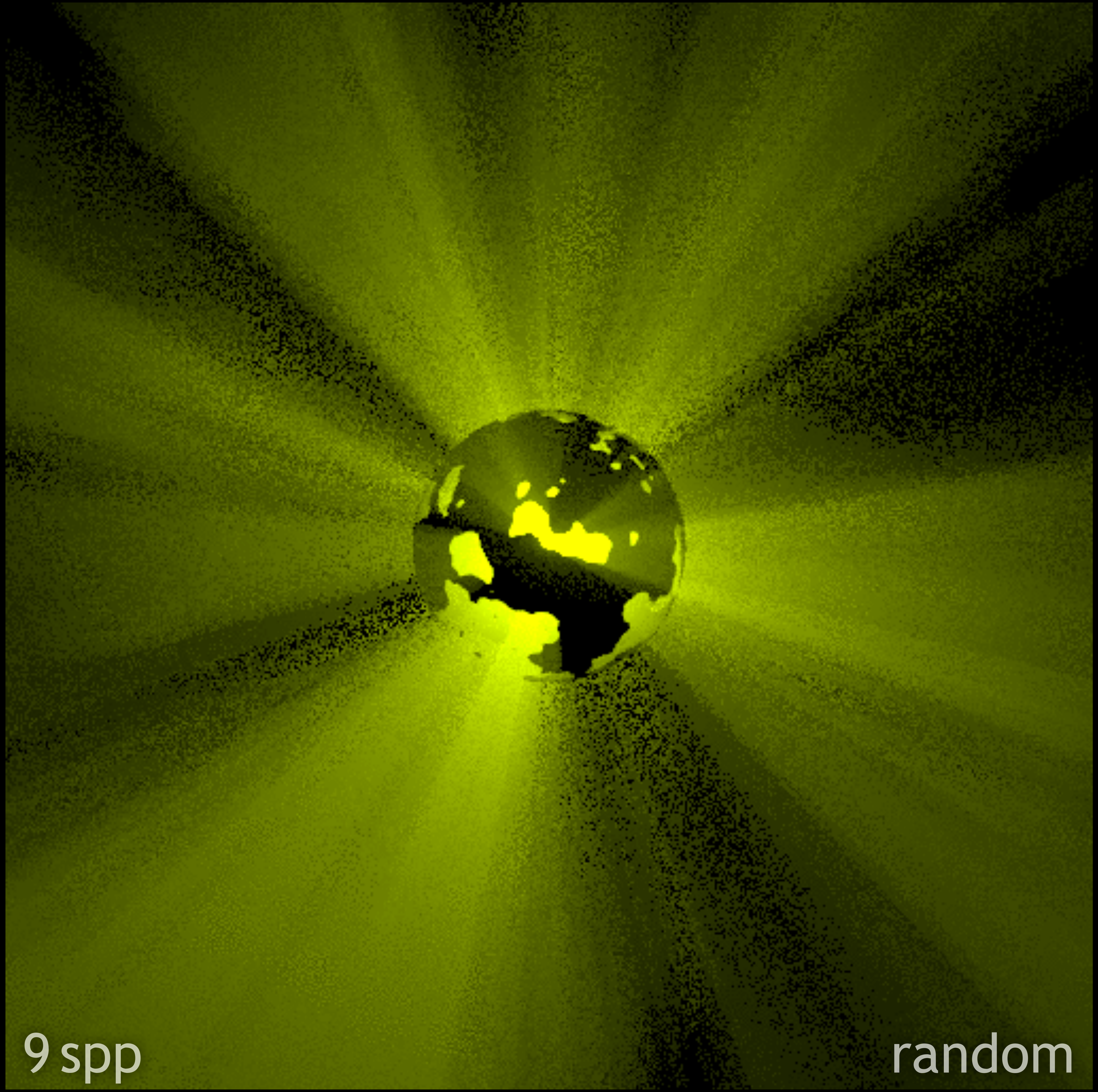


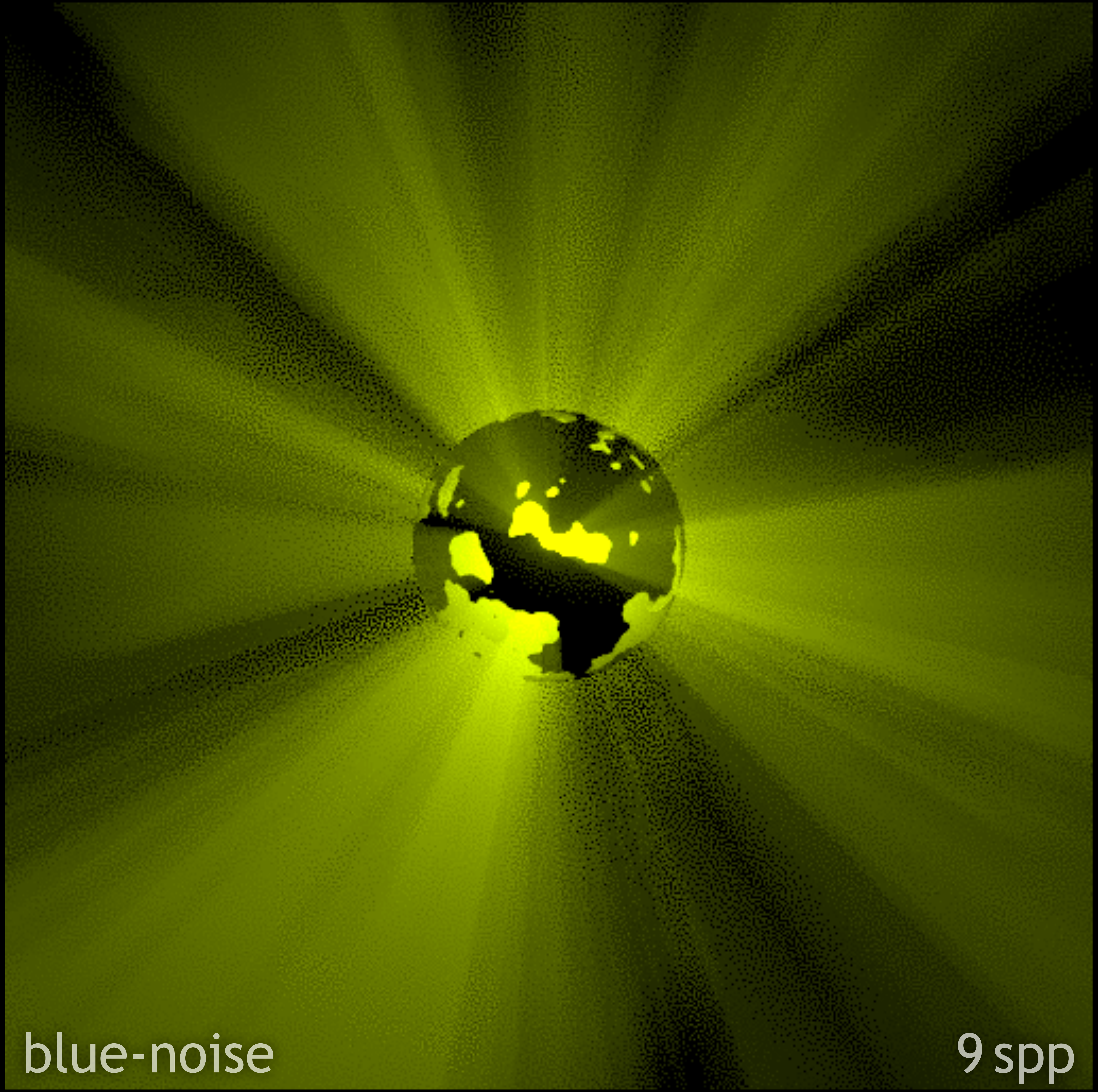
Dithered sampling: animation (1D sampling)





Dithered sampling: animation (1D sampling)





Dithered sampling: summary

Connection between halftoning and MC rendering

Blue correlation > **white** decorrelation

Simple, fast method

Limitations

- Occasional mask tiling artifacts
- Improvement only when integrand is smooth in
    - screen space
    - sample space
- Higher dimensions difficult
- Suboptimal with
    - progressive sampling
    - non-stratified patterns

Dithered sampling: summary

Connection between halftoning and MC rendering

Blue correlation > **white** decorrelation

Simple, fast method

Limitations

- Occasional mask tiling artifacts
- Improvement only when integrand is smooth in
  - screen space
  - sample space
- Higher dimensions difficult
- Suboptimal with
  - progressive sampling
  - non-stratified patterns

Addressed by recent work

Pattern-offset masks *[Georgiev & Fajardo 2016]*

```
pattern = generate_pattern()
for each pixel p:
    vector = lookup(mask, p)
    pattern_p =   offset(pattern, vector)
    render_pixel(p, pattern_p)
```

Pattern-scrambling masks *[Heitz & al. 2019]*

```
pattern = generate_pattern()
for each pixel p:
    seed      = lookup(mask, p)
    pattern_p = scramble(pattern, seed)
    render_pixel(p, pattern_p)
```

Pattern-scrambling masks *[Heitz & al. 2019]*

```
pattern = generate_pattern()
for each pixel p:
    seed      = lookup(mask, p)
    pattern_p = scramble(pattern, seed)
    render_pixel(p, pattern_p)
```

Mask construction

```
M = random_mask()
until converged:
    p,q = pick_random_pixels(M)
    if swap_reduces_energy(M,p,q): // probabilistic
        swap(p,q)
```

A Low-Discrepancy Sampler that Distributes Monte Carlo Errors as a Blue Noise in Screen Space [Heitz et al. 2019]

Pattern-scrambling masks *[Heitz & al. 2019]*

```
1. pattern = generate_pattern()
2. for each pixel p:
3.     seed    = lookup(mask, p)
4.     pattern_p = scramble(pattern, seed)
5.     render_pixel(p, pattern_p)
```

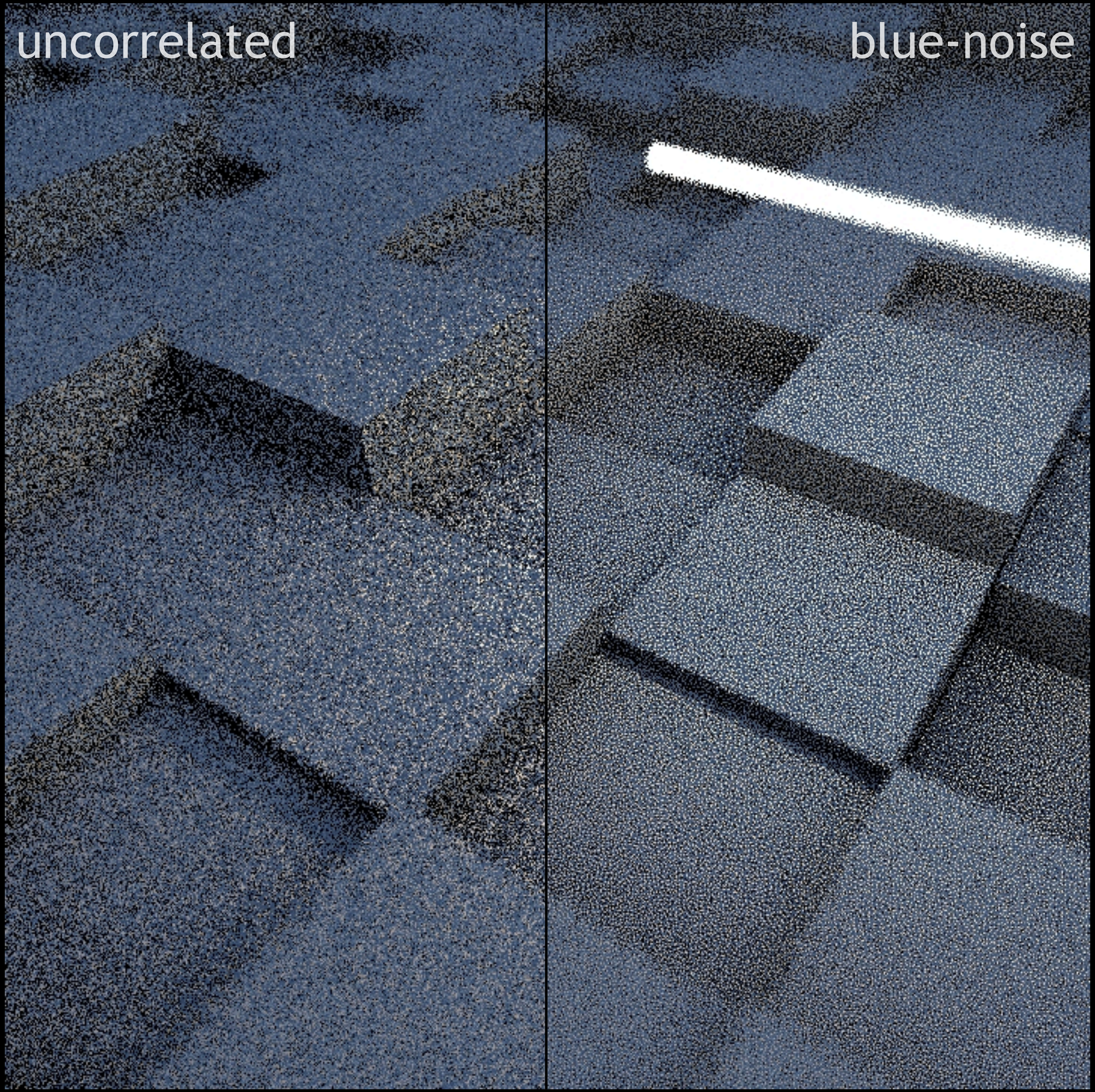
Mask construction

```
1. M = random_mask()
2. until converged:
3.     p,q = pick_random_pixels(M)
4.     if swap_reduces_energy(M,p,q): // probabilistic
5.         swap(p,q)
```

Laurent's talk,
later today

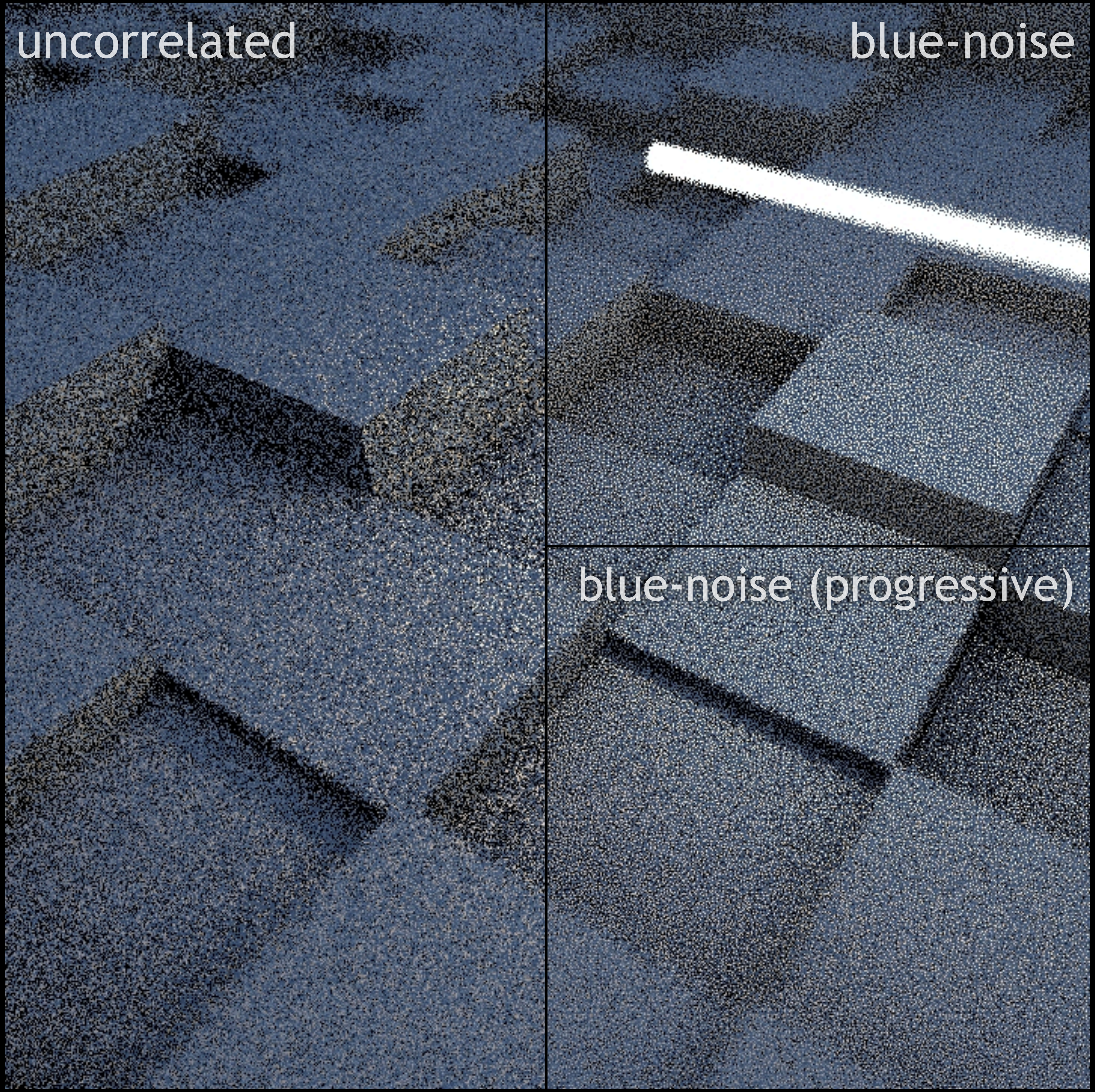
Pattern scrambling: results teaser

1 spp



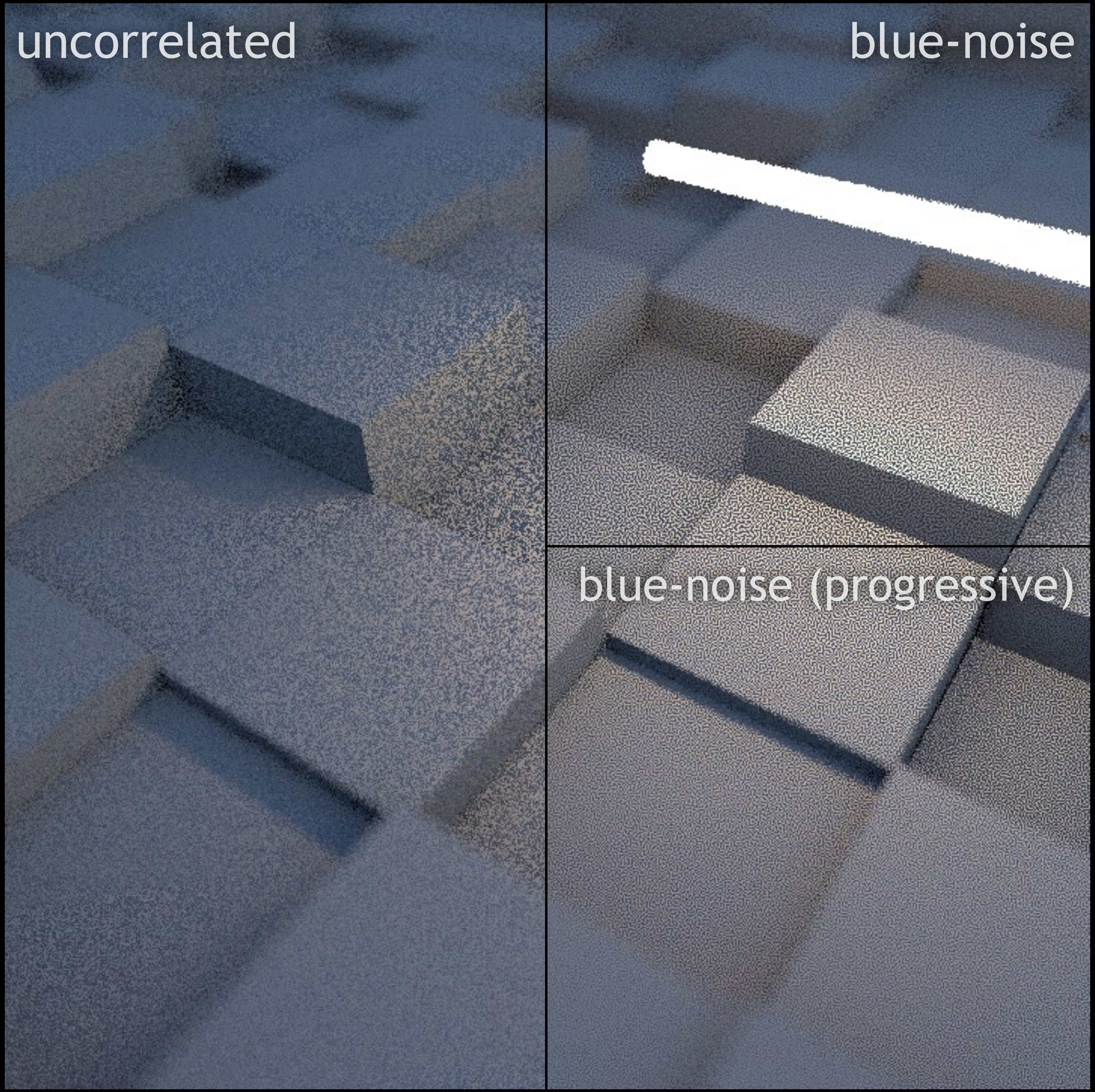
Pattern scrambling: results teaser

1 spp



Pattern scrambling: results teaser

8 spp



Pattern scrambling: advantages

Progressive sampling

Preserves pattern integration qualities

Higher dimensions

Seriously,
Laurent’s talk,
today at 3pm

Pattern scrambling: advantages

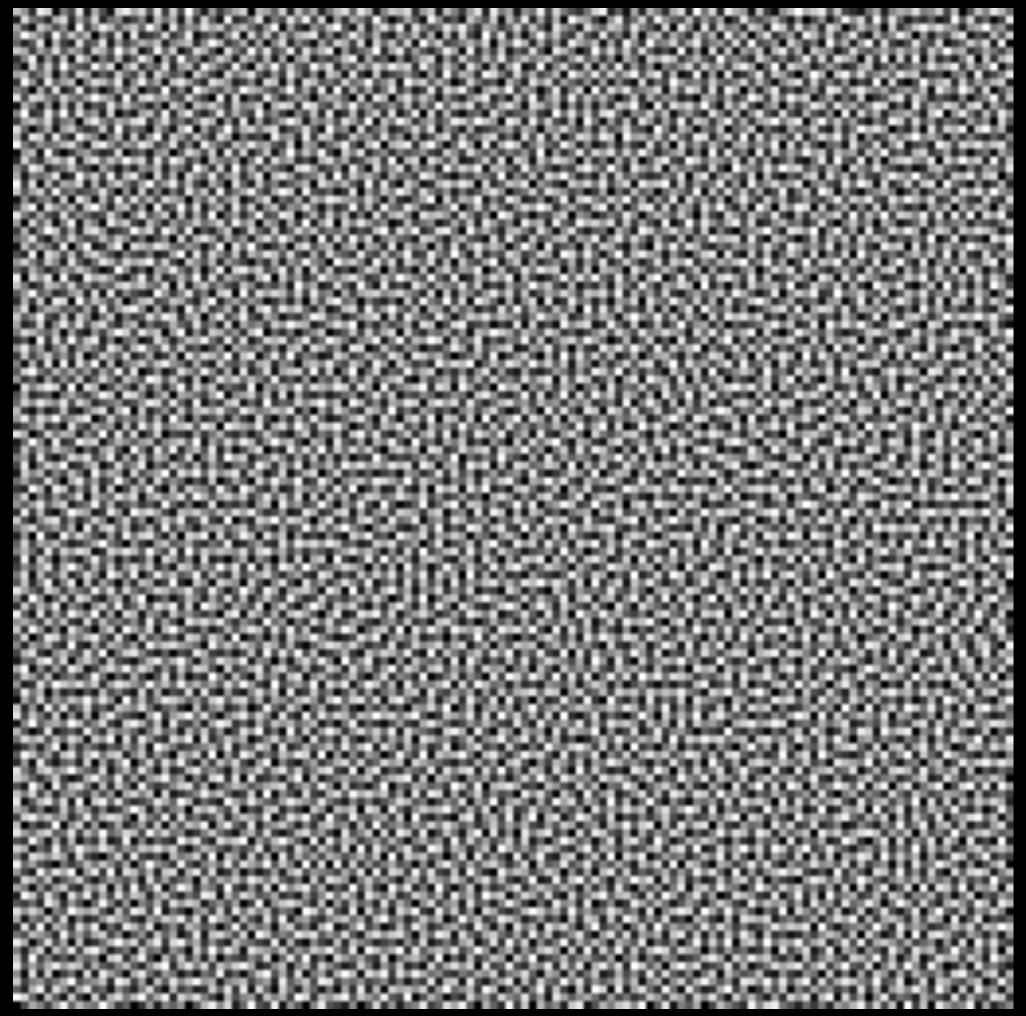
Progressive sampling

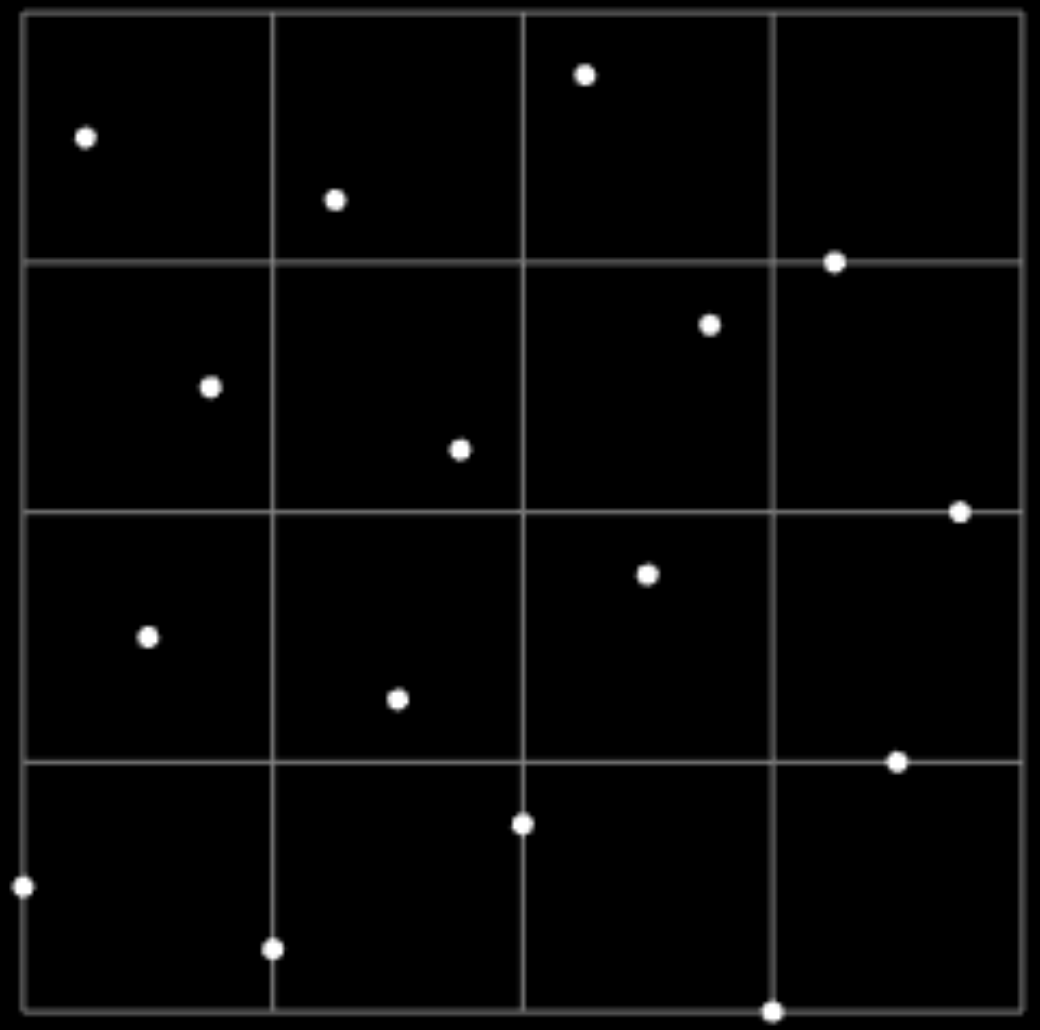
Preserves pattern integration qualities

Higher dimensions

- Not too many

Dithered offsetting/scrambling limitations





smooth integrand,
few dimensions

Dither pixel estimates more directly?

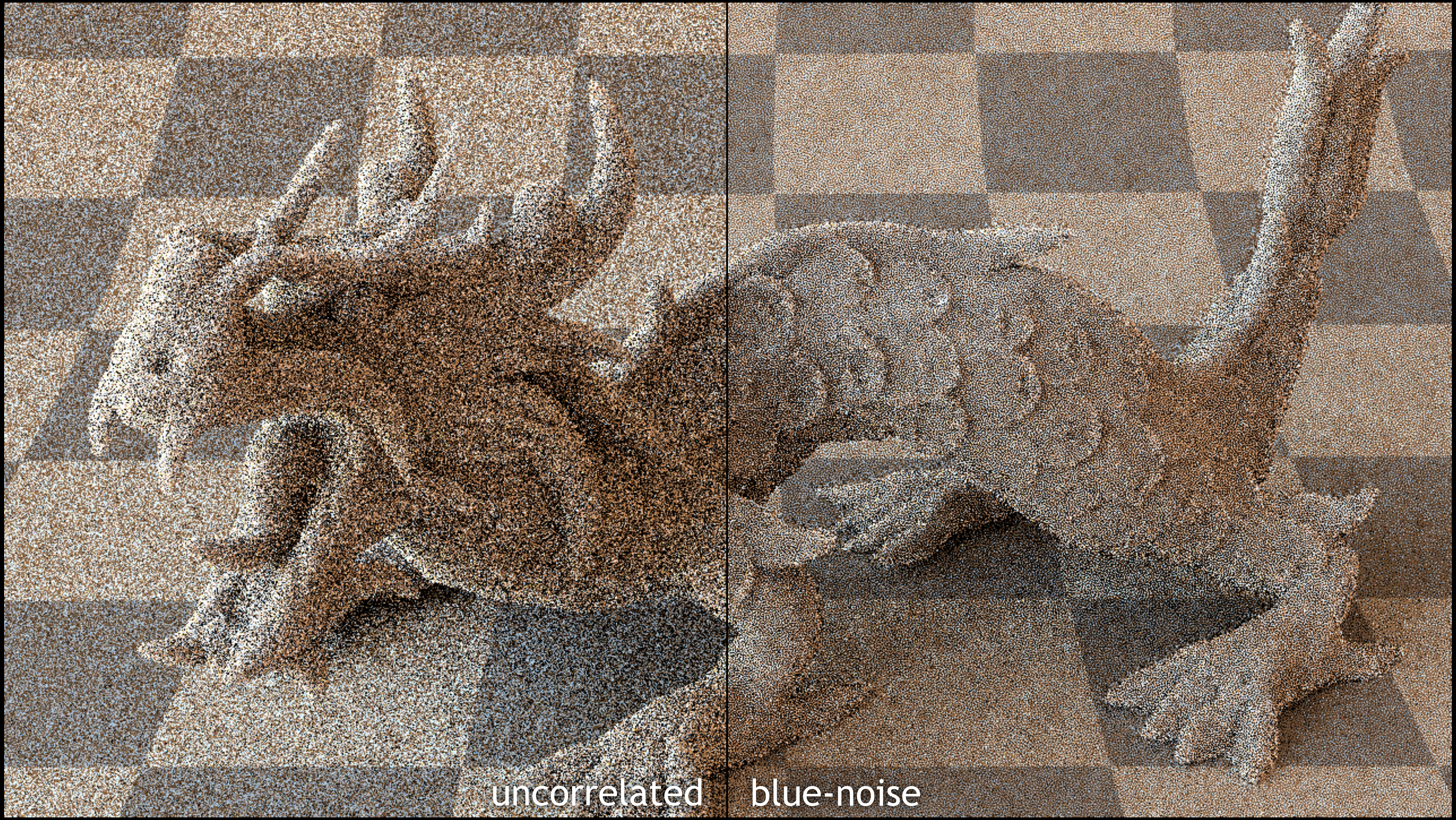
Pixel seed permutation *[Heitz & Belcour 2019]*

44630351
83957812
39572374
59258734
14572398
26379452
63814677
53684257
93246778

Distributing Monte Carlo Errors as a Blue Noise in Screen Space by Permuting Pixel Seeds Between Frames [Heitz et al. 2019]

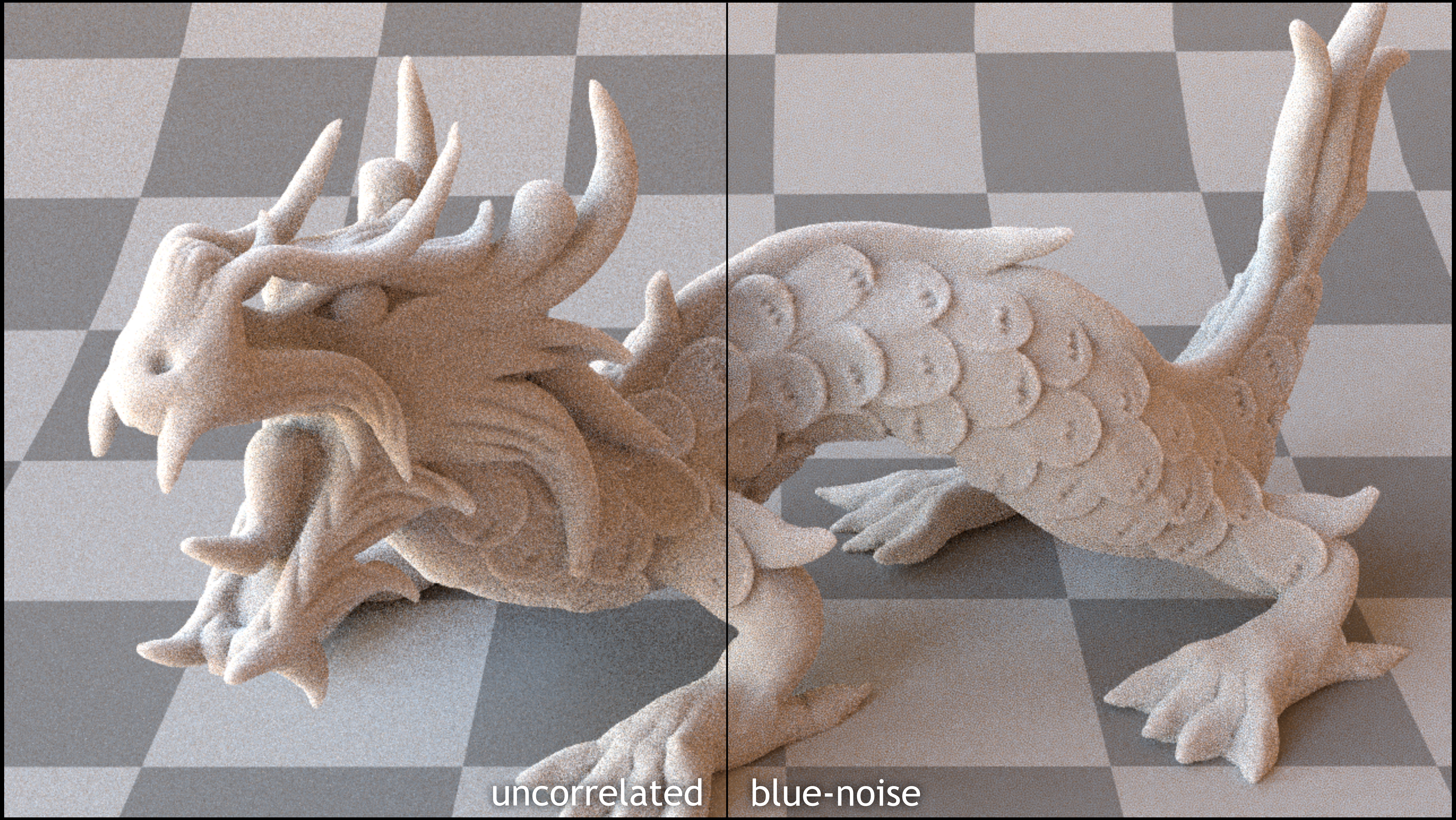
Pixel seed permutation: 10 light bounces

1 spp

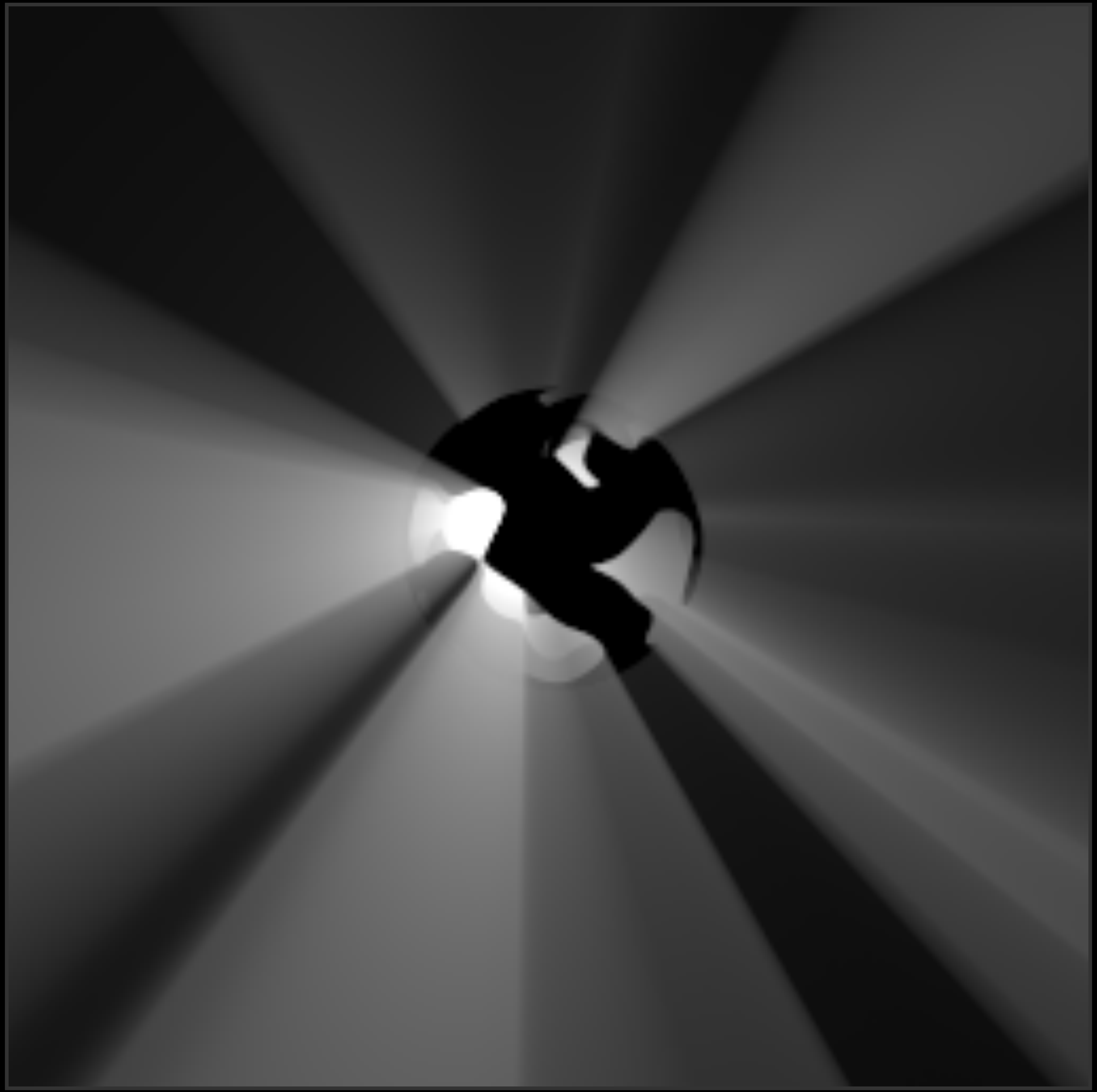


Pixel seed permutation: 10 light bounces

16 spp

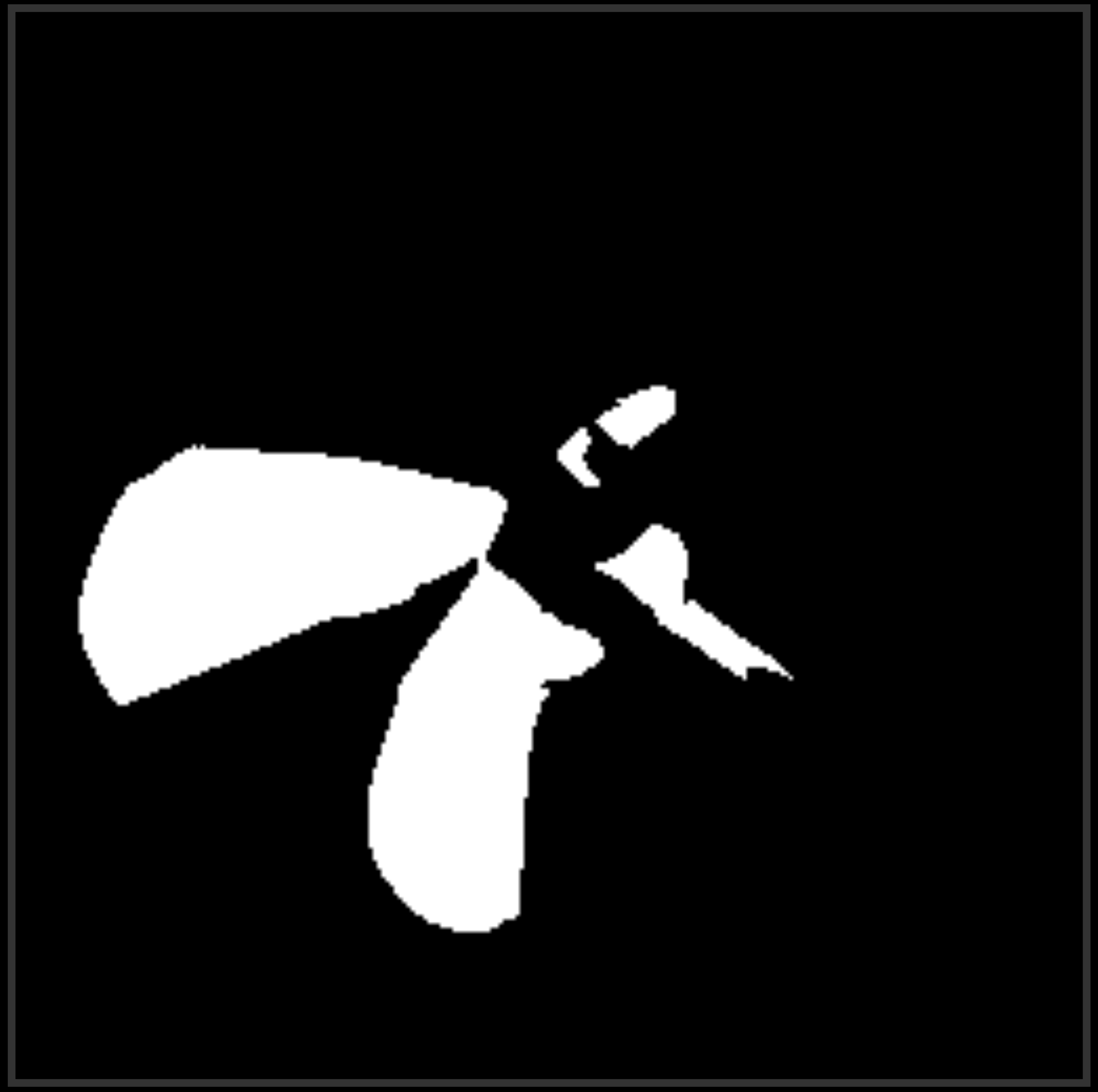
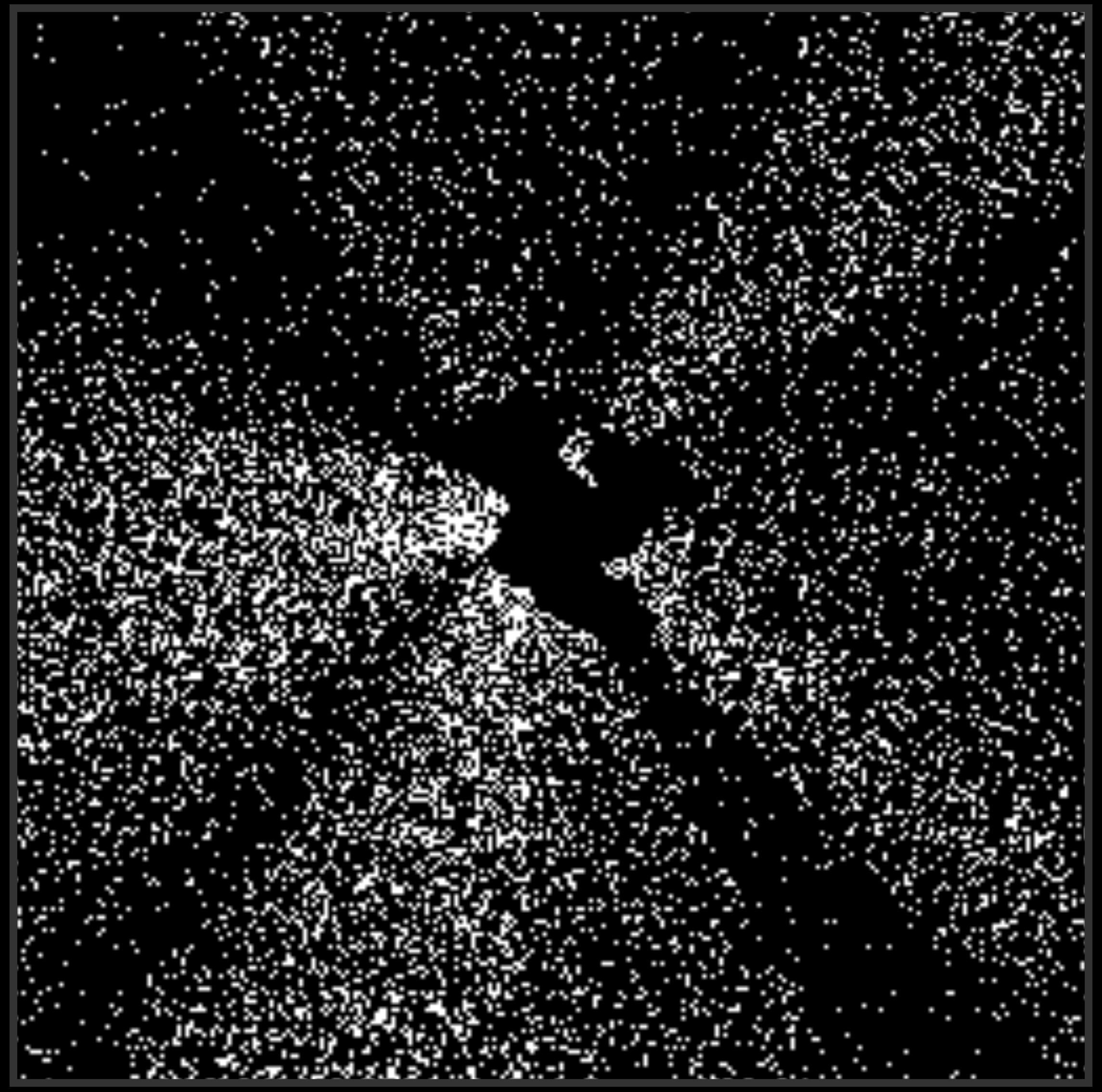
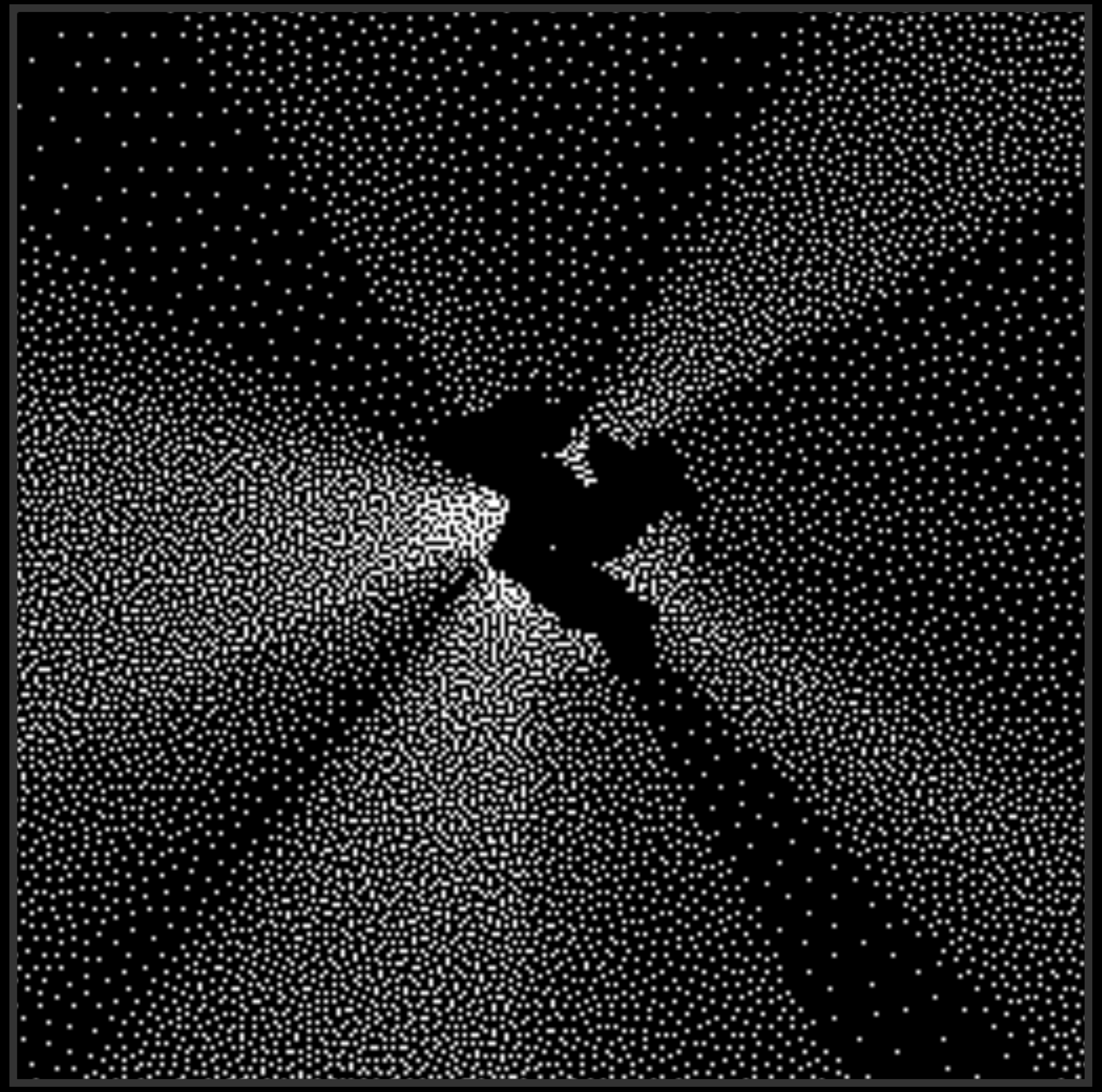


Takeaway: not *all* correlation is bad!



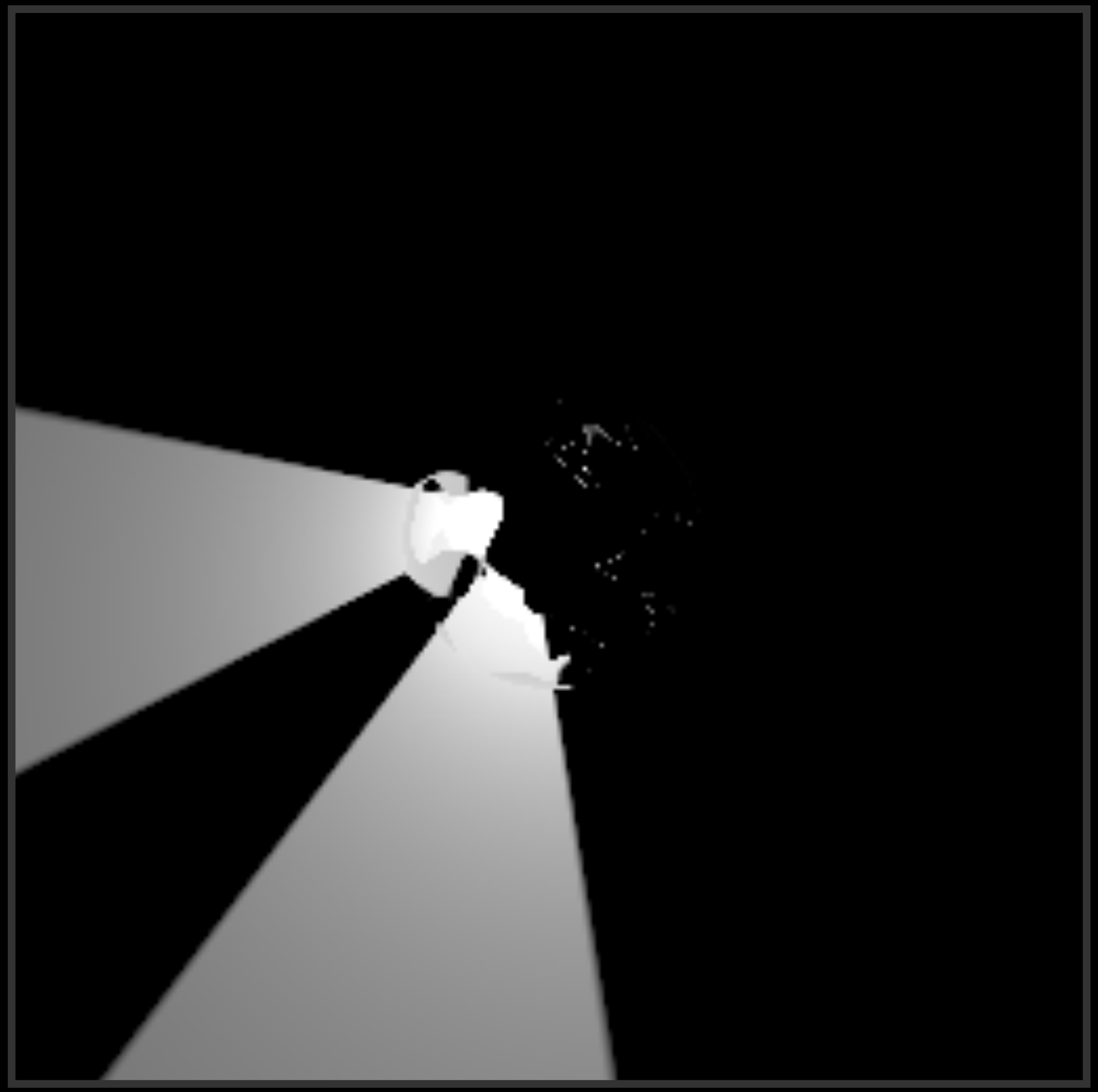
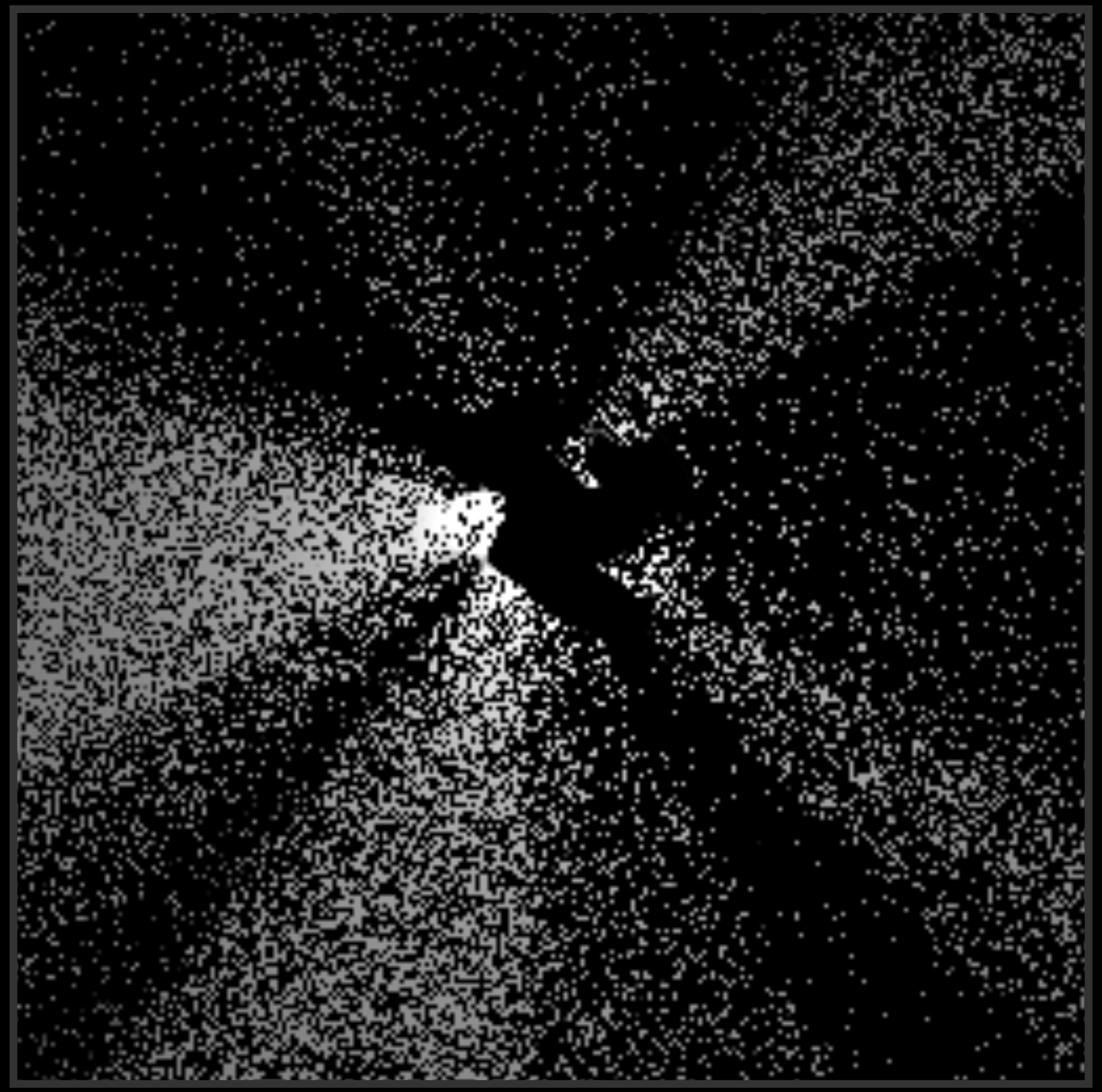
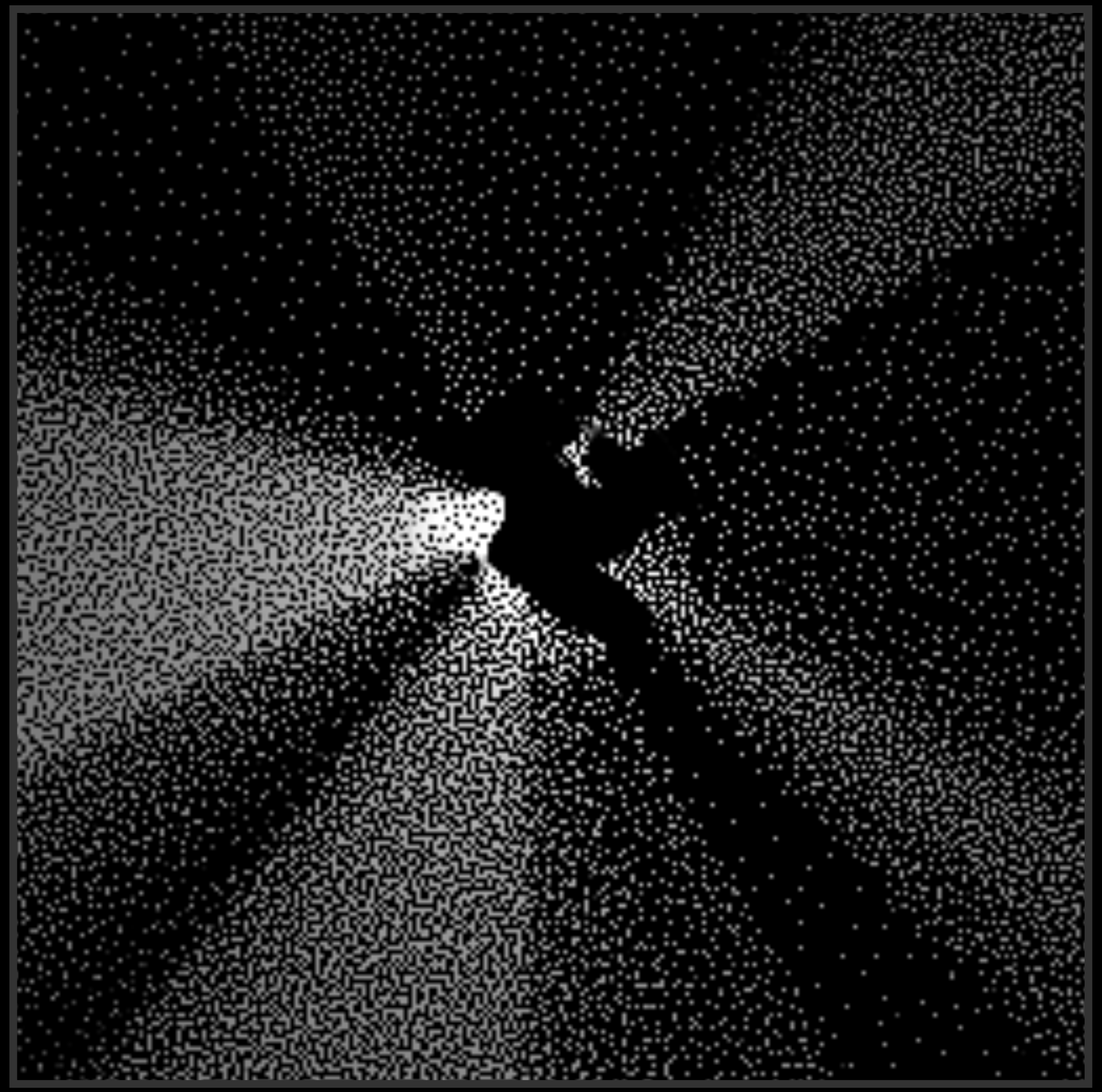
Takeaway: not *all* correlation is bad!

halftoning

constant *(correlated)* random *(uncorrelated)* blue-noise *(correlated)*

rendering

Summary

Correlation ≠ bias

Blue-noise error dithering ≠ blue-noise point sampling

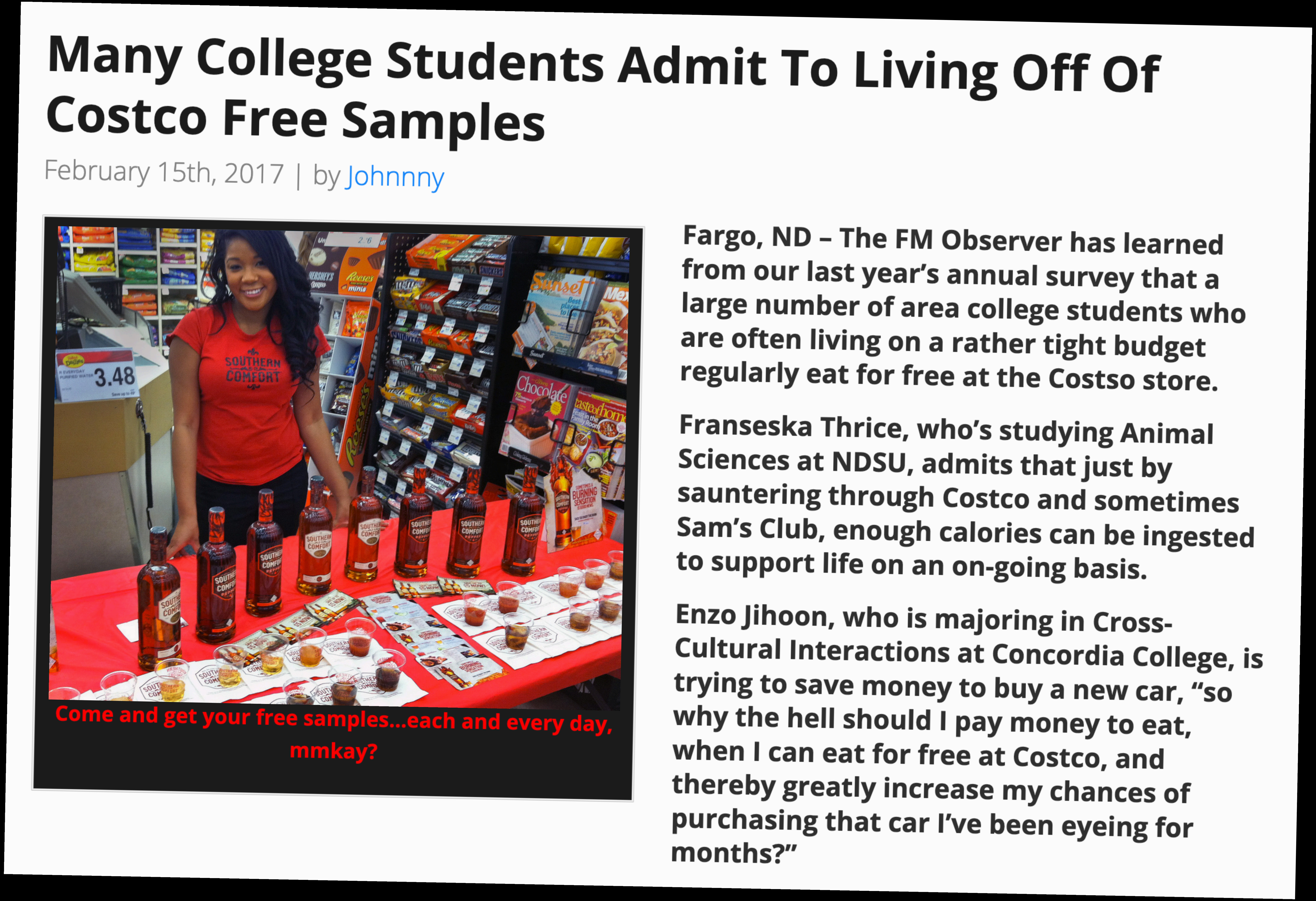
Dithering + denoising = BFF

Opinion: ‘random’ is never the best

Debits:

~~Credits:~~ Jorge Schwarzhaupt, Marios Papas, Johannes Hanika, Marco Manzi

My favorite samples



Many College Students Admit To Living Off Of Costco Free Samples

February 15th, 2017 | by Johnnny

Come and get your free samples...each and every day, mmkay?

Fargo, ND – The FM Observer has learned from our last year's annual survey that a large number of area college students who are often living on a rather tight budget regularly eat for free at the Costso store.

Franseska Thrice, who's studying Animal Sciences at NDSU, admits that just by sauntering through Costco and sometimes Sam's Club, enough calories can be ingested to support life on an on-going basis.

Enzo Jihoon, who is majoring in Cross-Cultural Interactions at Concordia College, is trying to save money to buy a new car, "so why the hell should I pay money to eat, when I can eat for free at Costco, and thereby greatly increase my chances of purchasing that car I've been eyeing for months?"

My favorite samples

Anything clear?