Solid and procedural textures

15-468, 15-668, 15-868
Physics-based Rendering
Spring 2021, Lecture 6

http://graphics.cs.cmu.edu/courses/15-468

Course announcements

- Take-home quiz 2 posted, due next Tuesday at 23:59.
- Programming assignment 1 posted, due Friday 2/26 at 23:59.
  - How many of you have looked at/started/finished it?
  - Any questions?
- Weekly reading groups.
  - Make sure to complete the poll on Piazza: https://piazza.com/class/kklw0l5me2or4?cid=30

Overview of today's lecture

- Go over take-home quiz 1.
- 3D textures.
- Procedural textures.
- Generating "realistic noise".

Slide credits

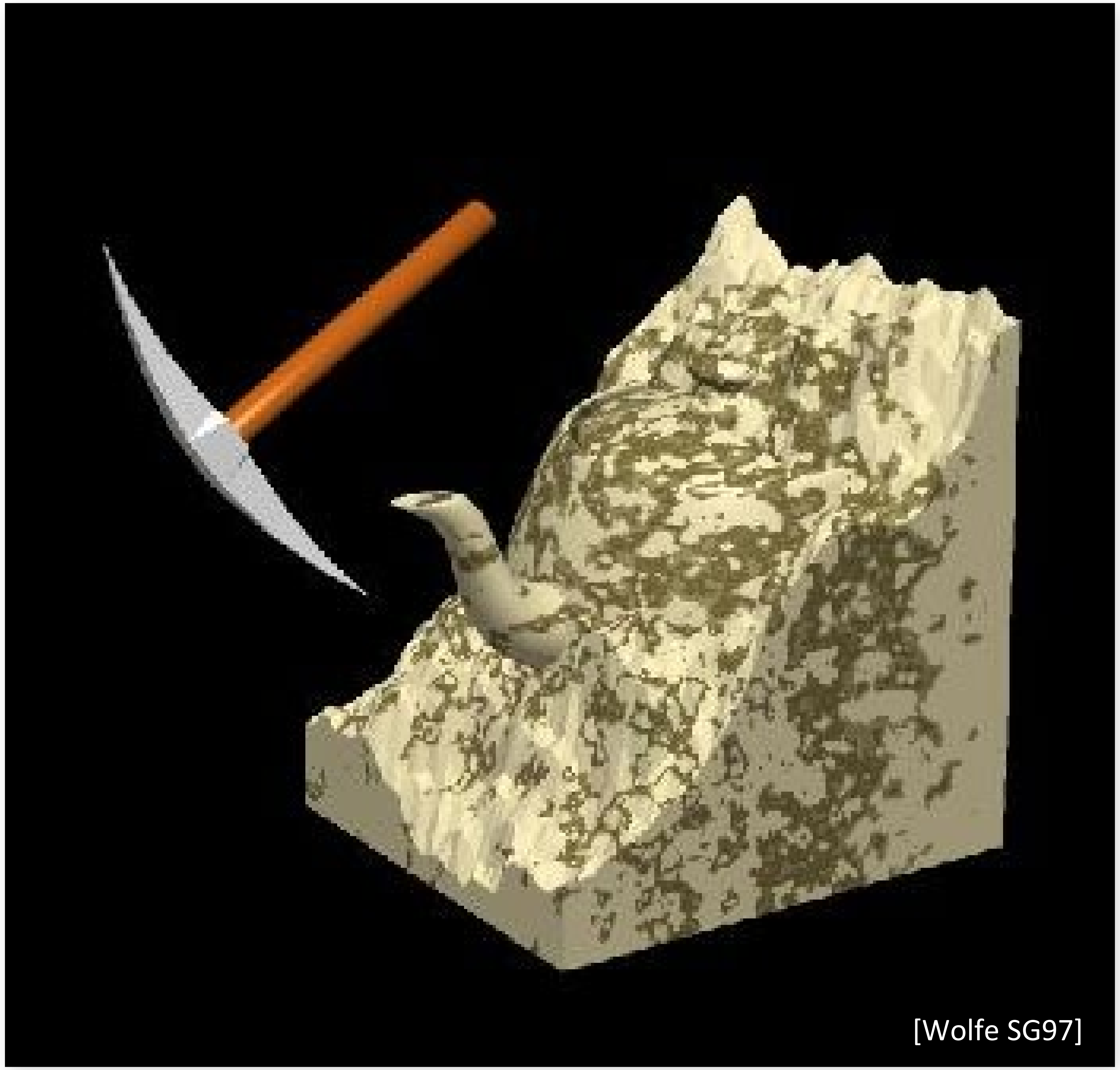
Most of these slides were directly adapted from:

- Wojciech Jarosz (Dartmouth).

3D textures

Texture is a function of (u, v, w)

- can just evaluate texture at 3D surface point
- good for solid materials
- often defined procedurally



[Wolfe SG97]

Procedural texturing

Instead of using rasterized image data, define texture procedurally

Simple example:

- color = 0.5*sin(x) + 0.5

Often called “solid texturing” because texture can easily vary in all 3 dimensions.

- but you can also do 2D or 1D procedural textures

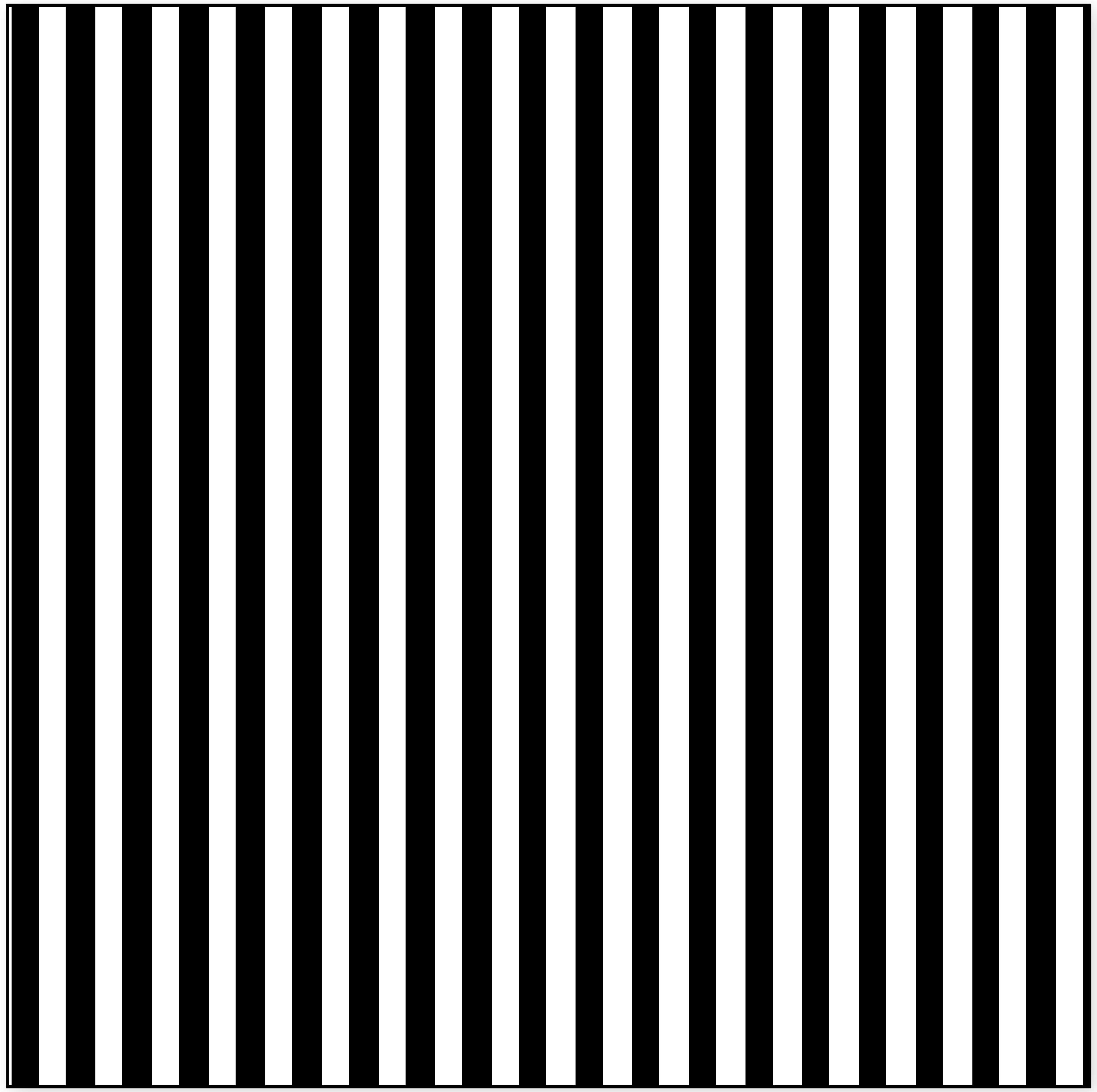
Raster vs. procedural textures

Why use procedural textures?

- low memory usage
- infinite resolution
- solid texture: no need to parametrize surface

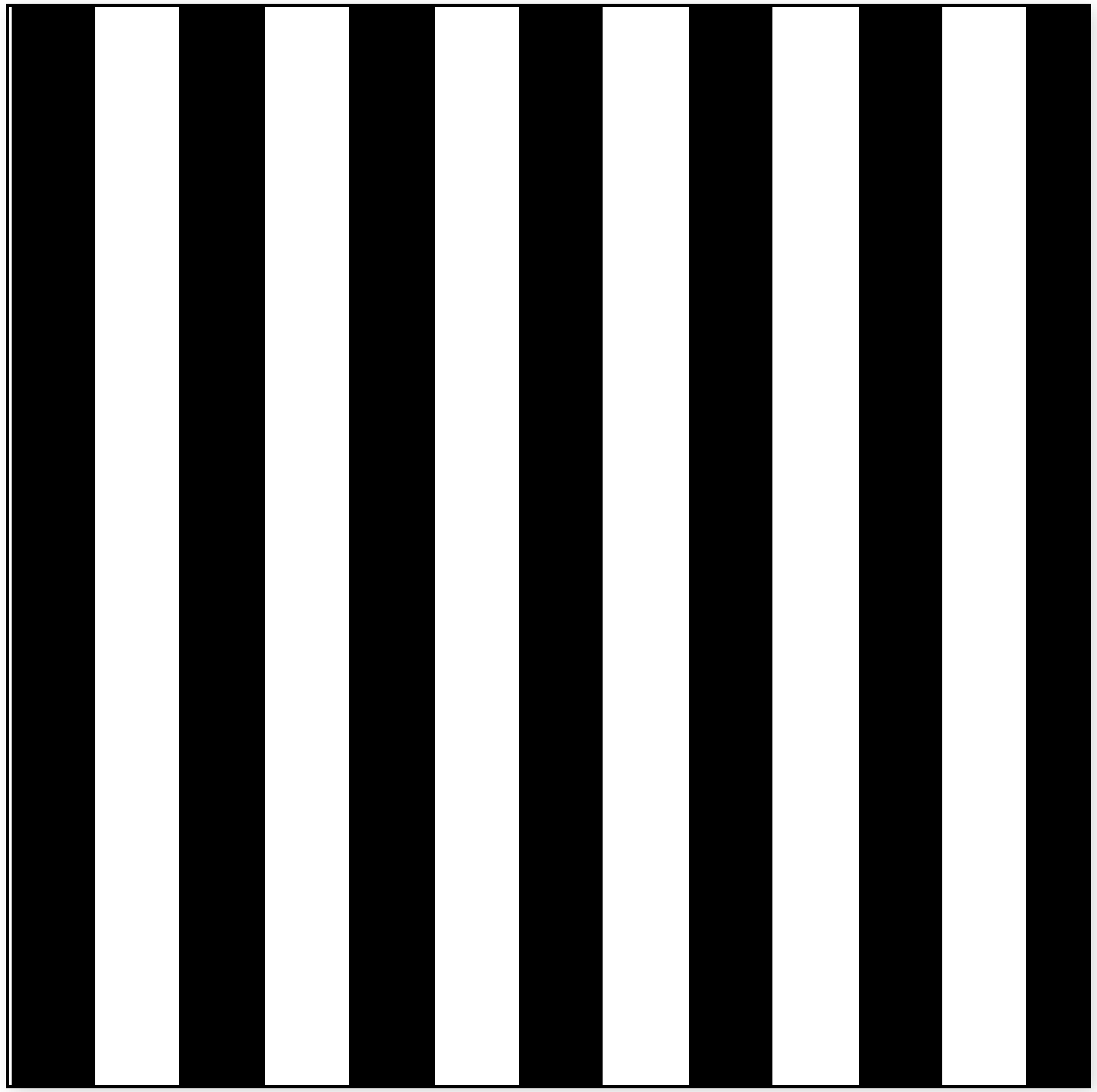
3D stripe texture

```
color stripe(point p):
    if (sin(p_x) > 0)
        return c_0
    else
        return c_1
```



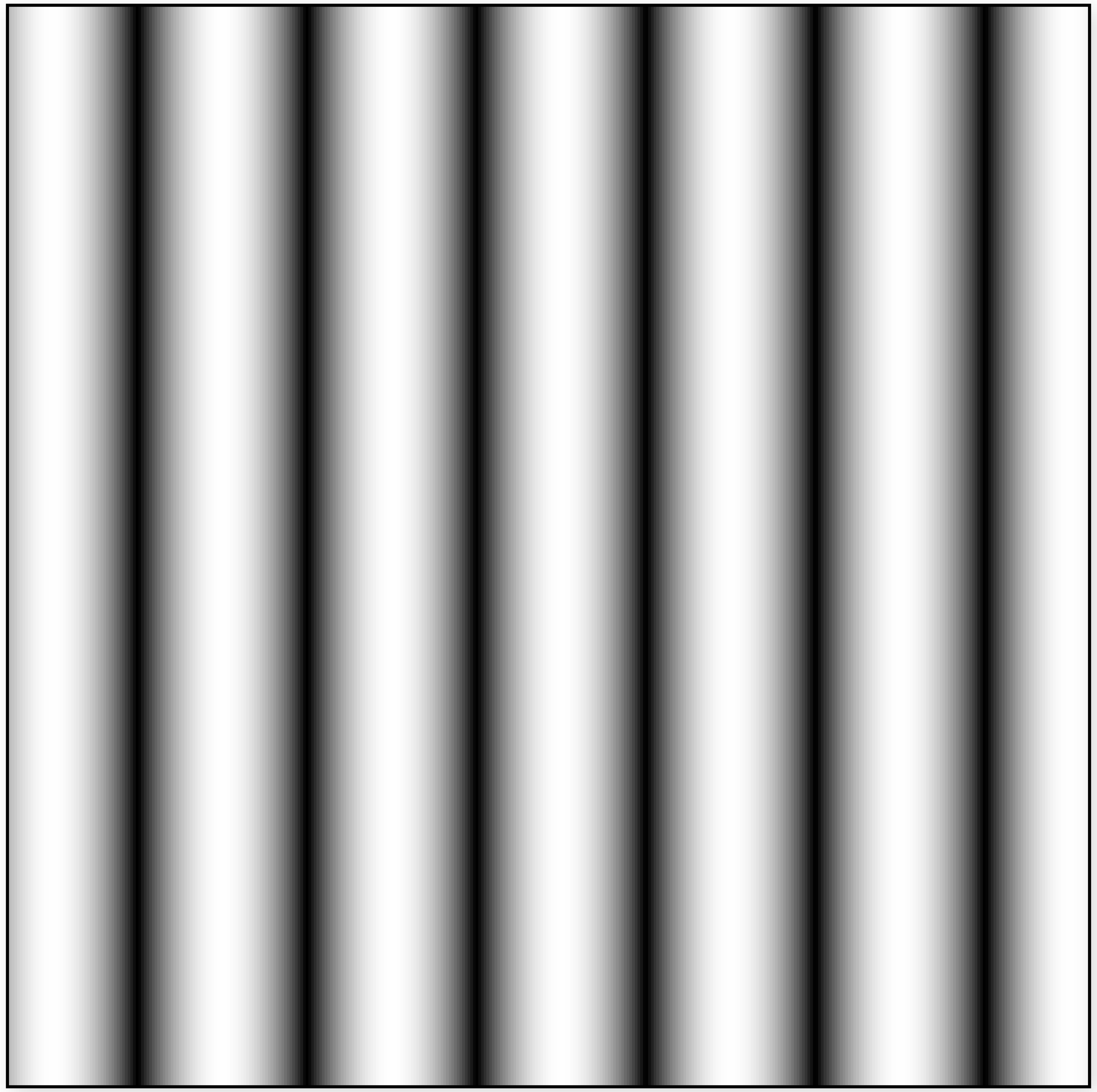
3D stripe texture

```
color stripe(point p, real w):
    if (sin(πpx/w) > 0)
        return c0
    else
        return c1
```



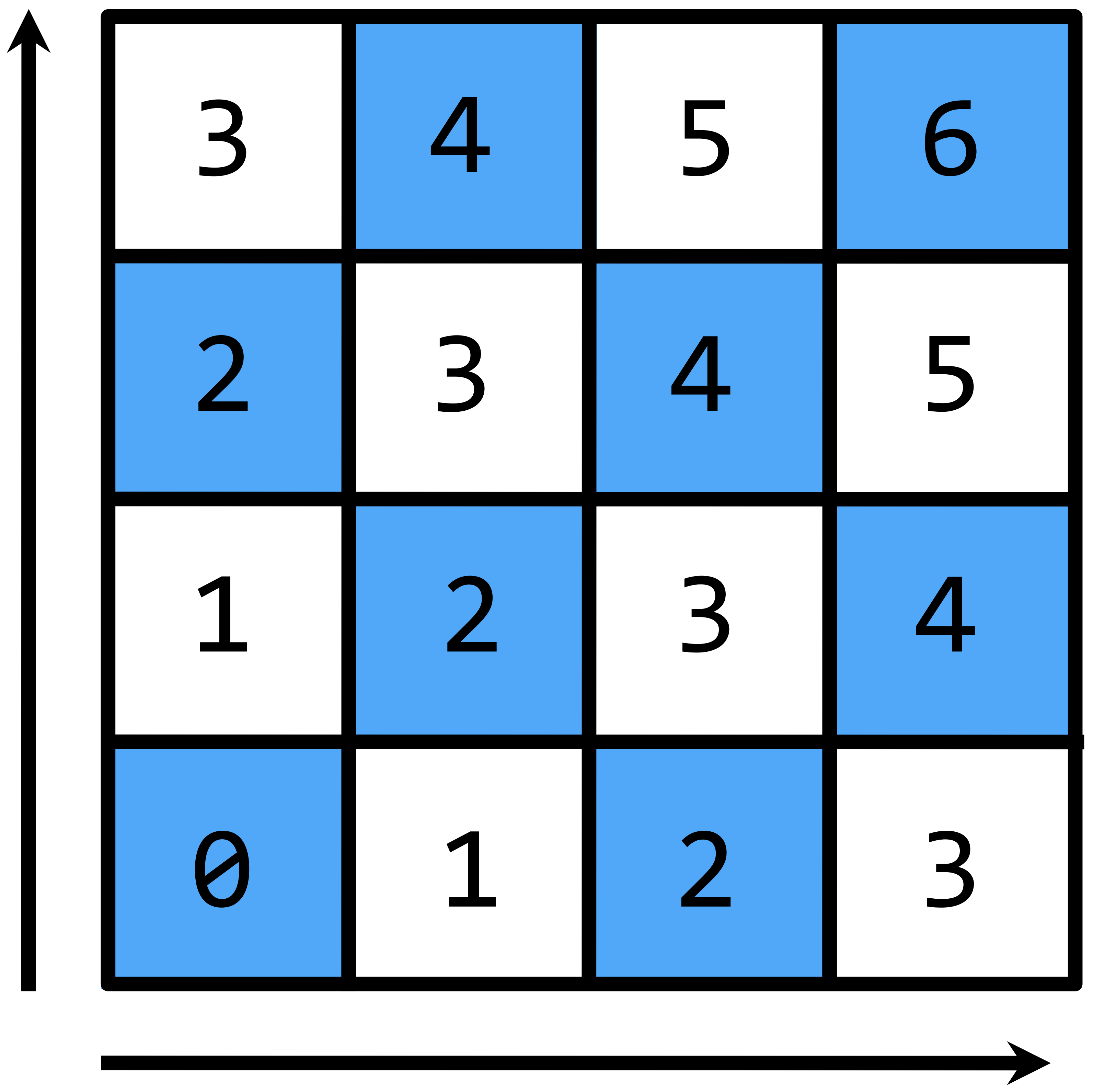
3D stripe texture

```
color stripe(point p, real w):
    t = (1 + sin(π p_x/w))/2
    return lerp(c_0, c_1, t)
```



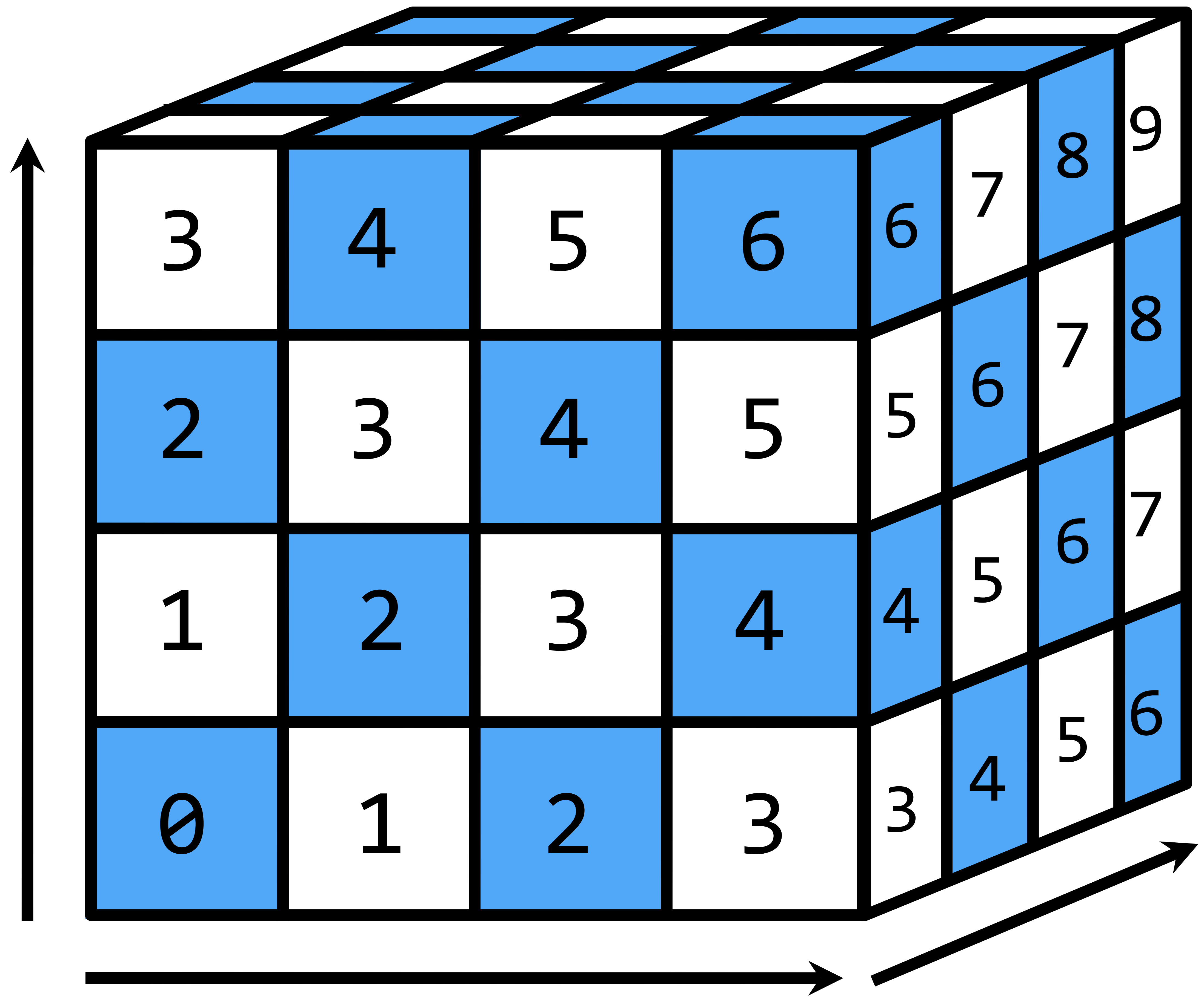
2D checkerboard texture

```
color checkerboard(point p):
    real a = floor(px)
    real b = floor(py)
    real val = a+b
    if (isEven(val))
        return c0
    else
        return c1
```



3D checkerboard texture

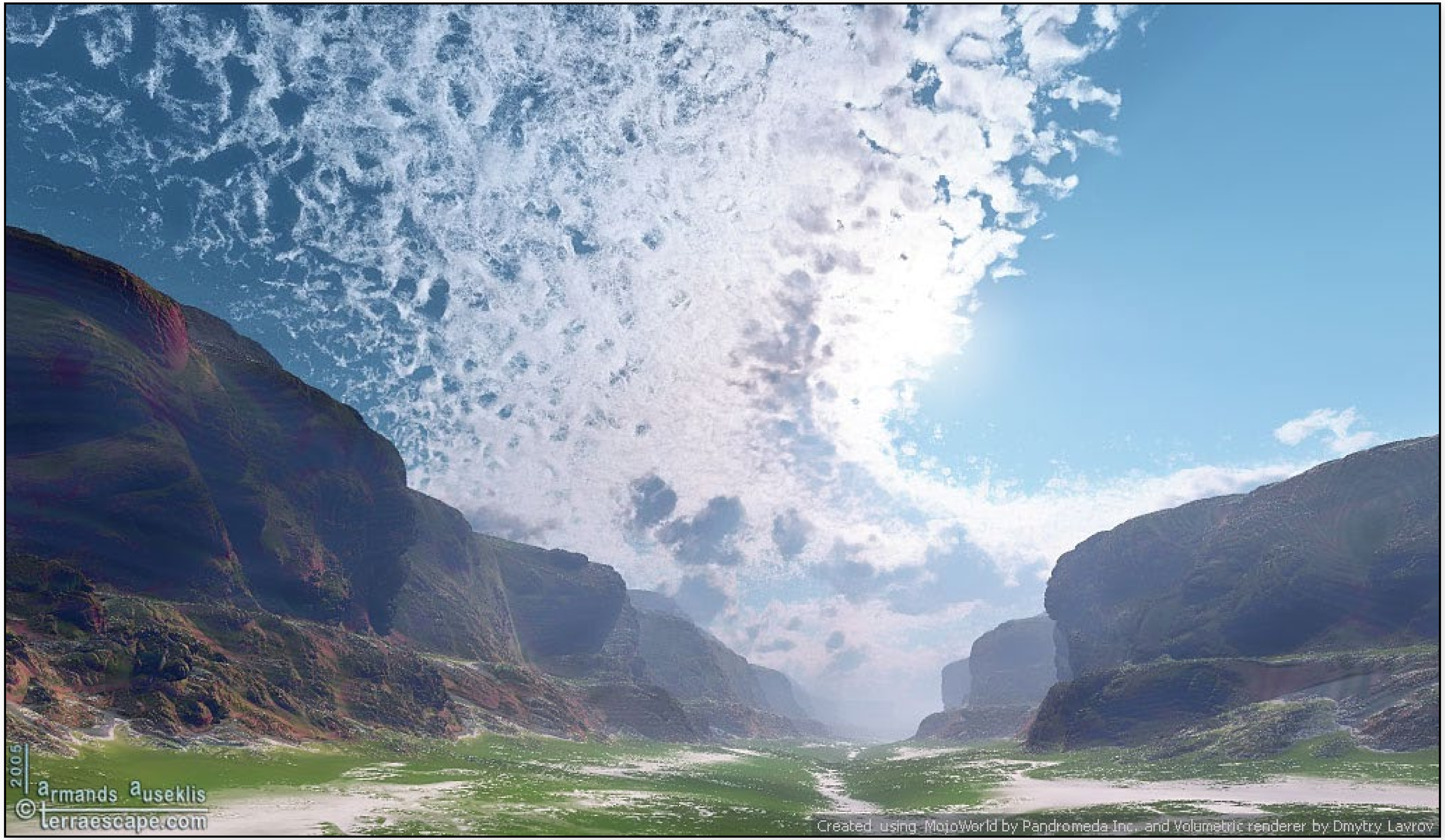
```
color checkerboard(point p):
    real a = floor(px)
    real b = floor(py)
    real c = floor(pz)
    real val = a+b+c
    if (isEven(val))
        return c0
    else
        return c1
```



Procedural synthesis

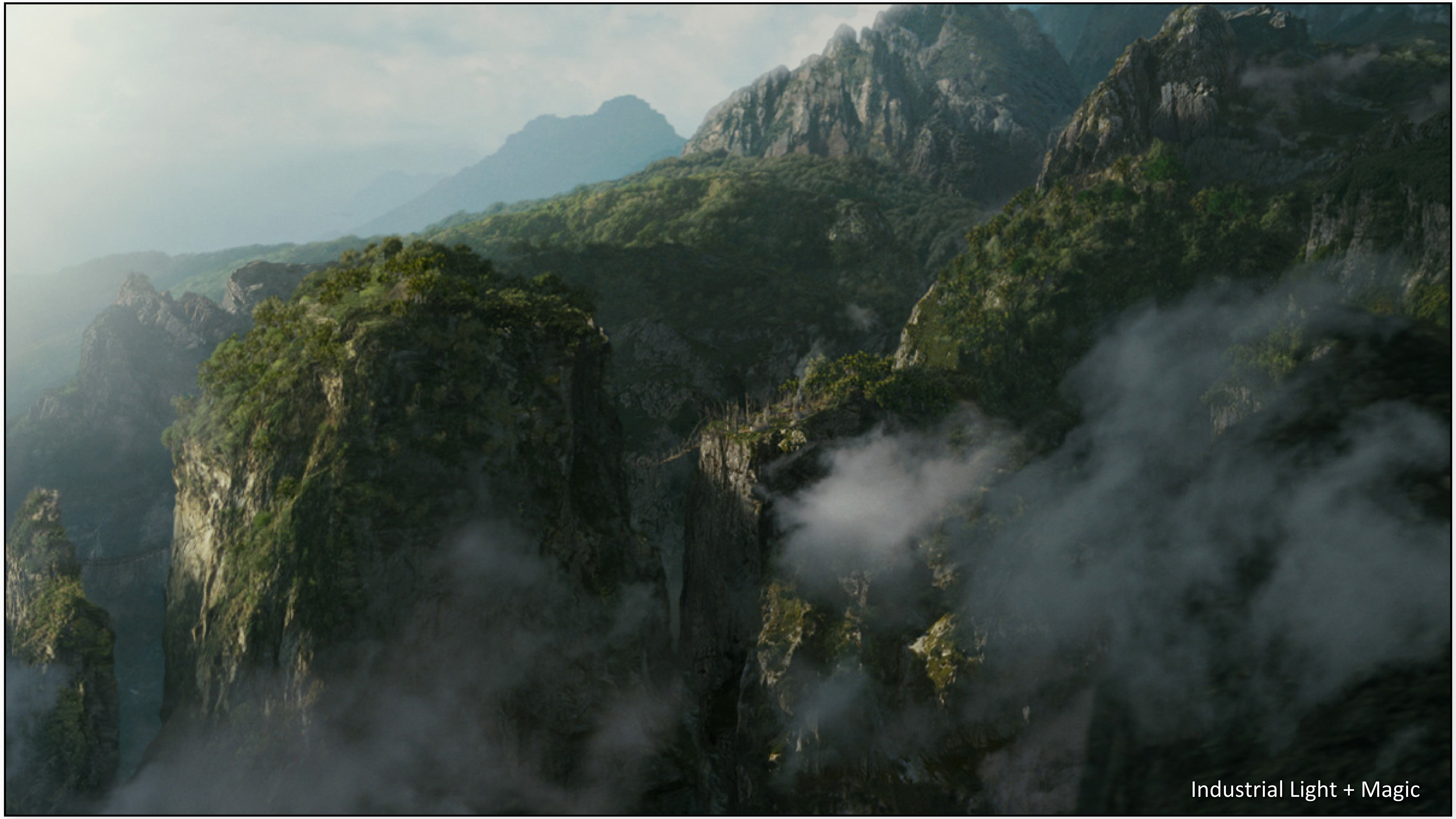
created using Terragen

Procedural synthesis



created using MojoWorld

Procedural synthesis



Digital matte painting for Pirates of the Caribbean 2; created using Vue Infinite

Procedural synthesis

created using Vue Infinite

Procedural textures

Our procedurals are "too perfect"

Often want to add controlled variation to a texture

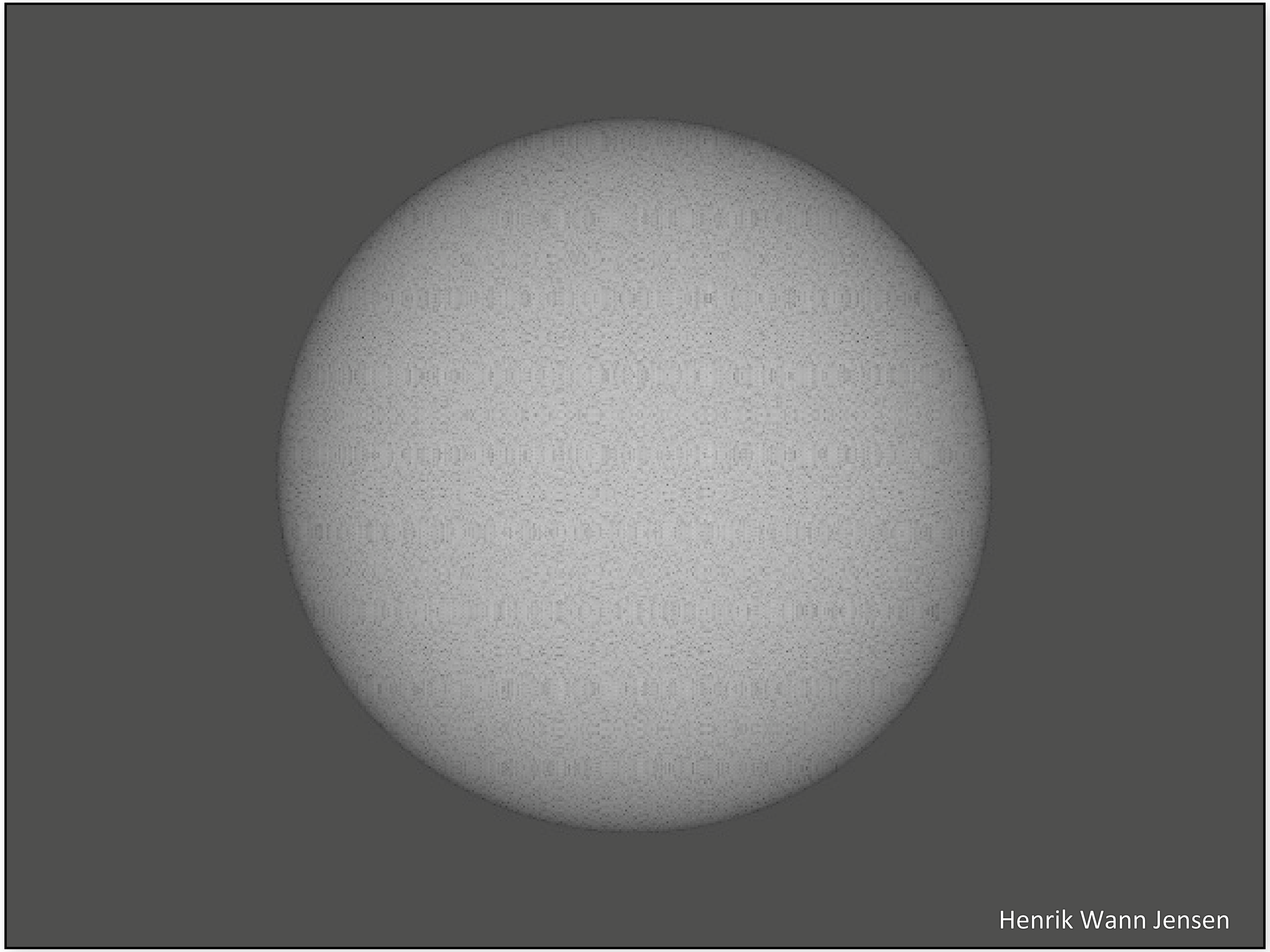
- Real textures have many imperfections

Just calling rand() is not that useful.

Random noise

albedo = randf();

Not band-limited, white noise.



Noise functions

Function: $\mathbf{R}^n \longrightarrow [-1, 1]$, where $n = 1,2,3,...$

Desirable properties:

- no obvious repetition
- rotation invariant
- band-limited (i.e. not scale-invariant)

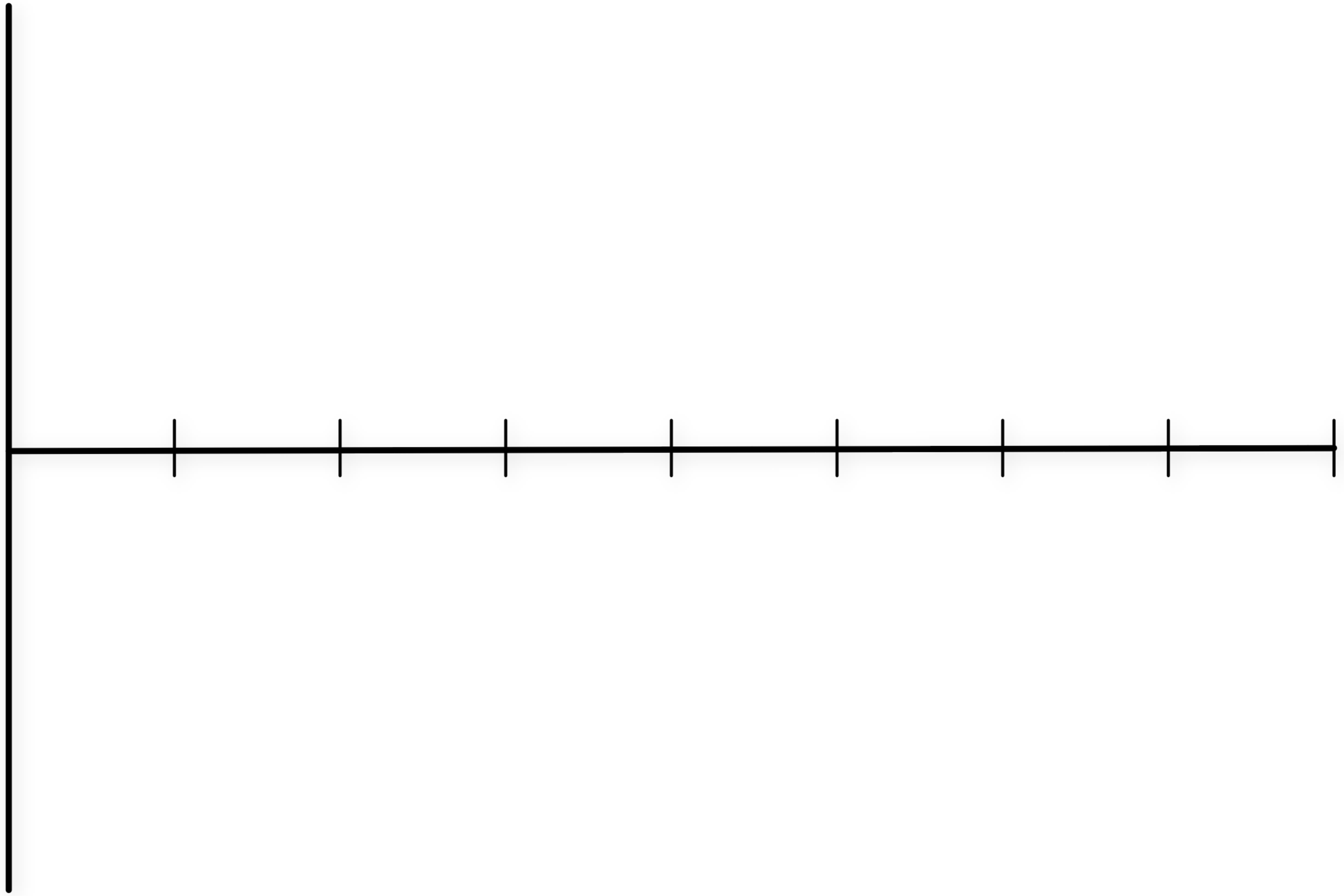
Fundamental building block of most procedural textures

Value noise

Values associated with integer lattice locations

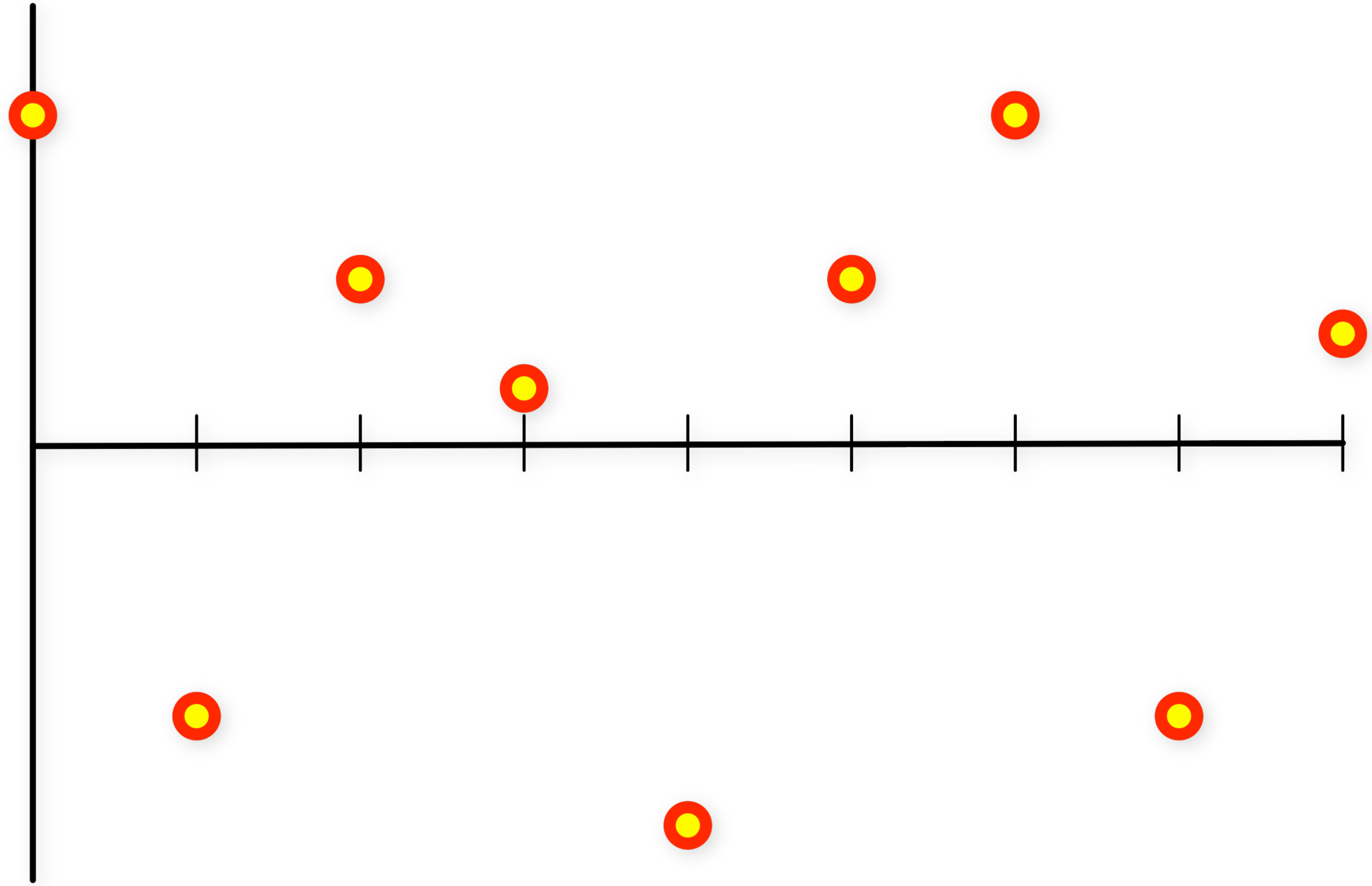
Given arbitrary position, interpolate value from neighboring lattice points

Value noise example



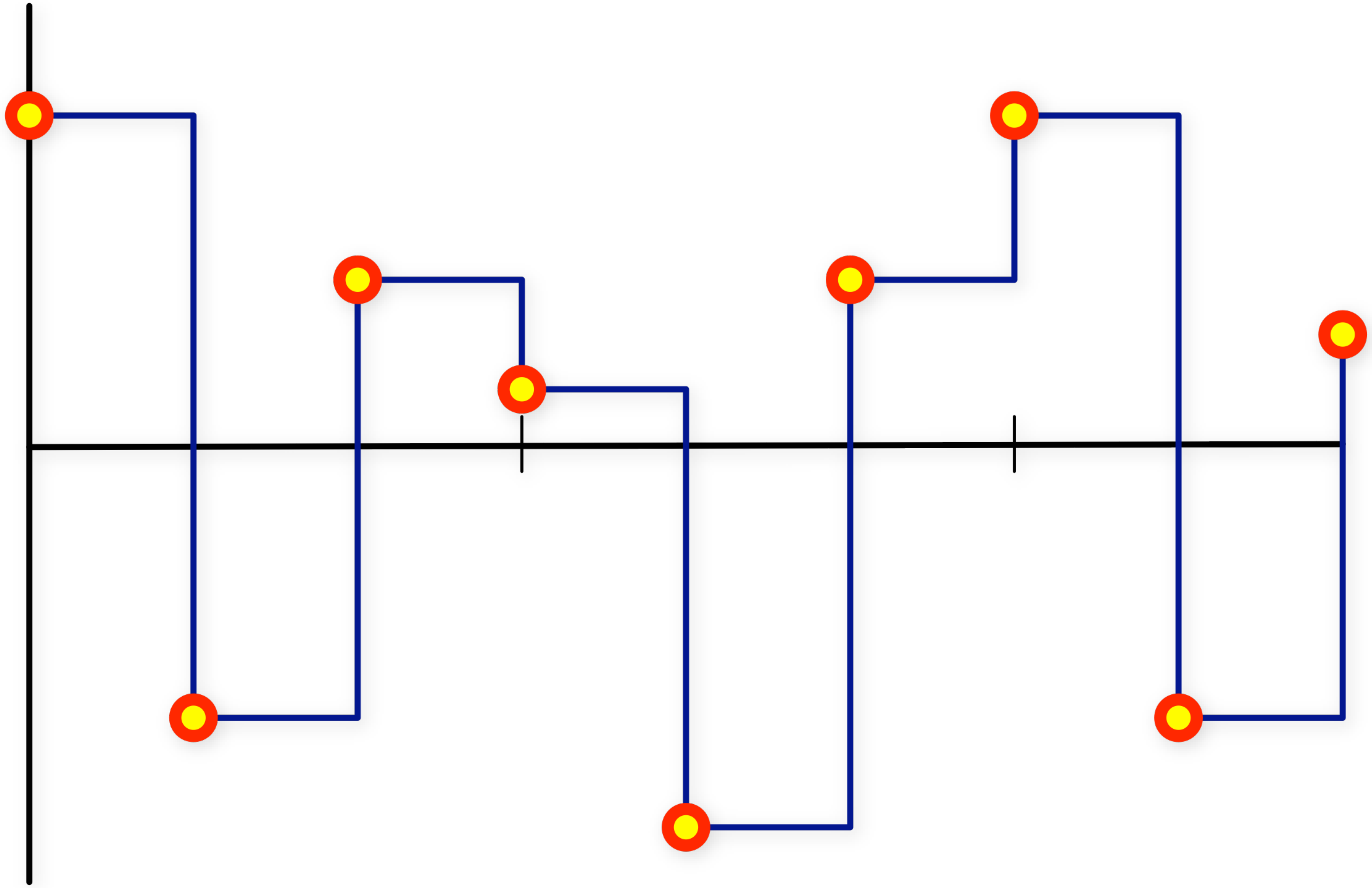
Value noise example

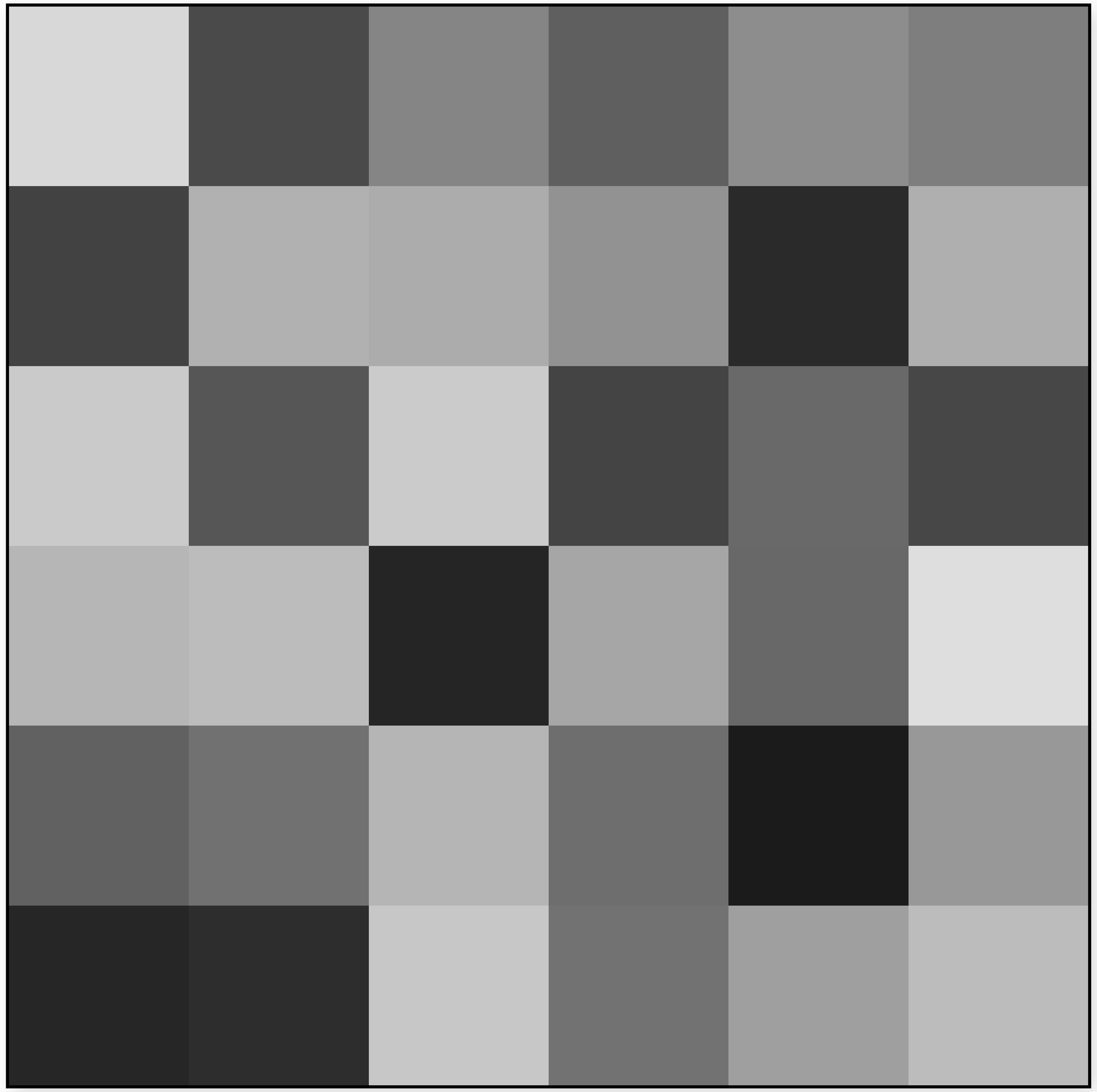
Random values on grid



Value noise example

Random values on grid

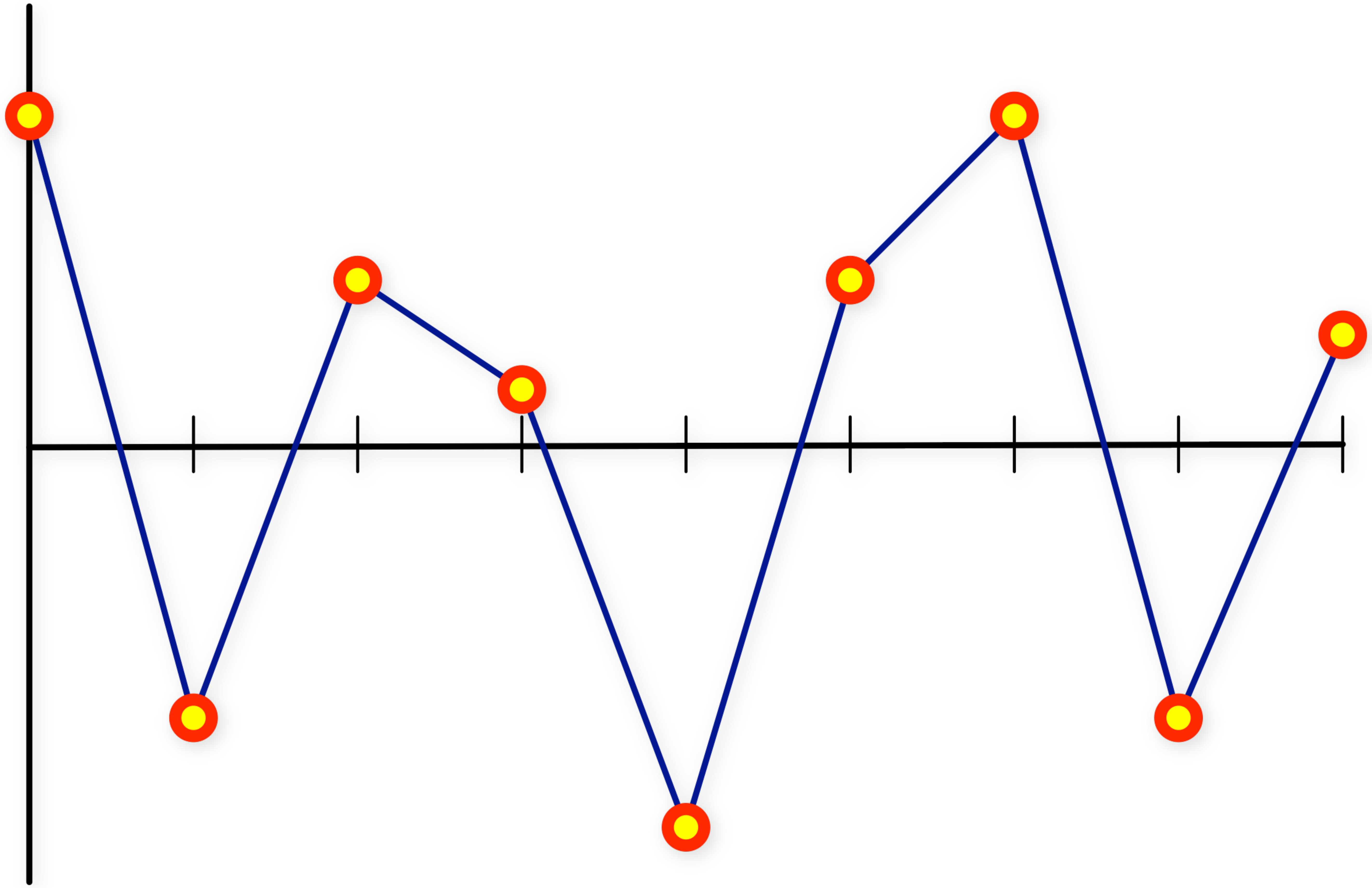


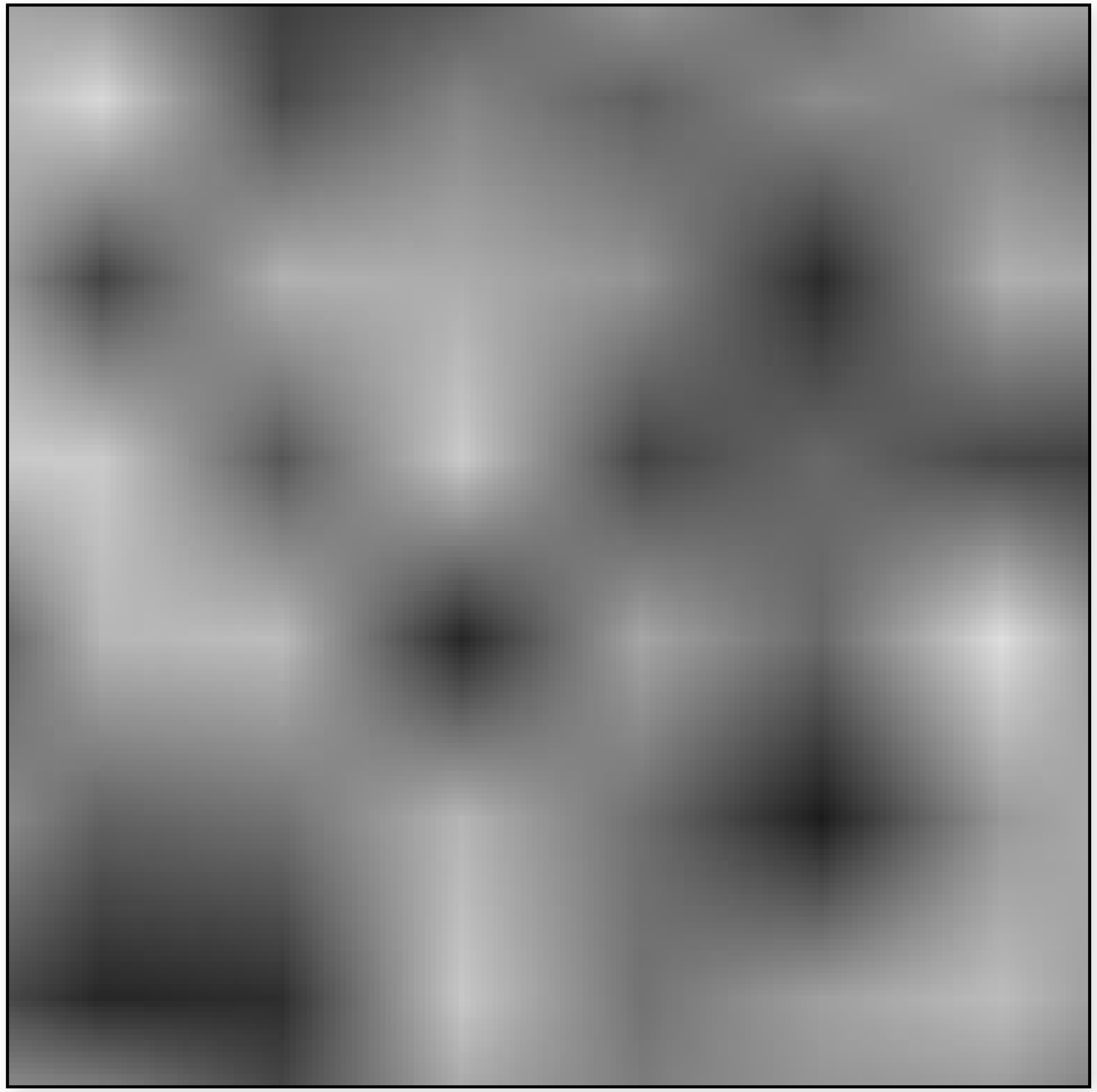


Cell noise: use value of nearest point on grid

Value noise example

(Bi-) linearly interpolated values

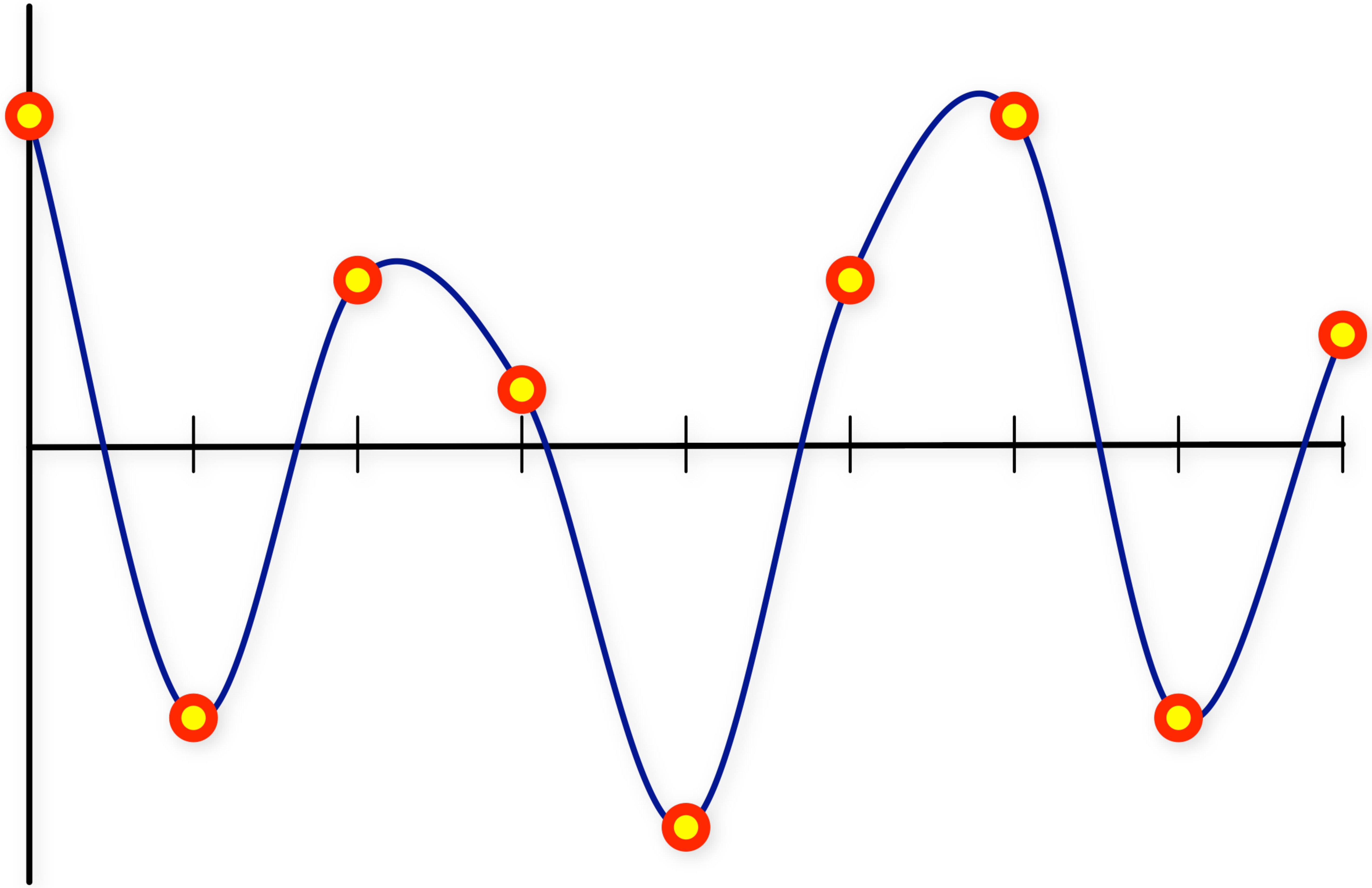


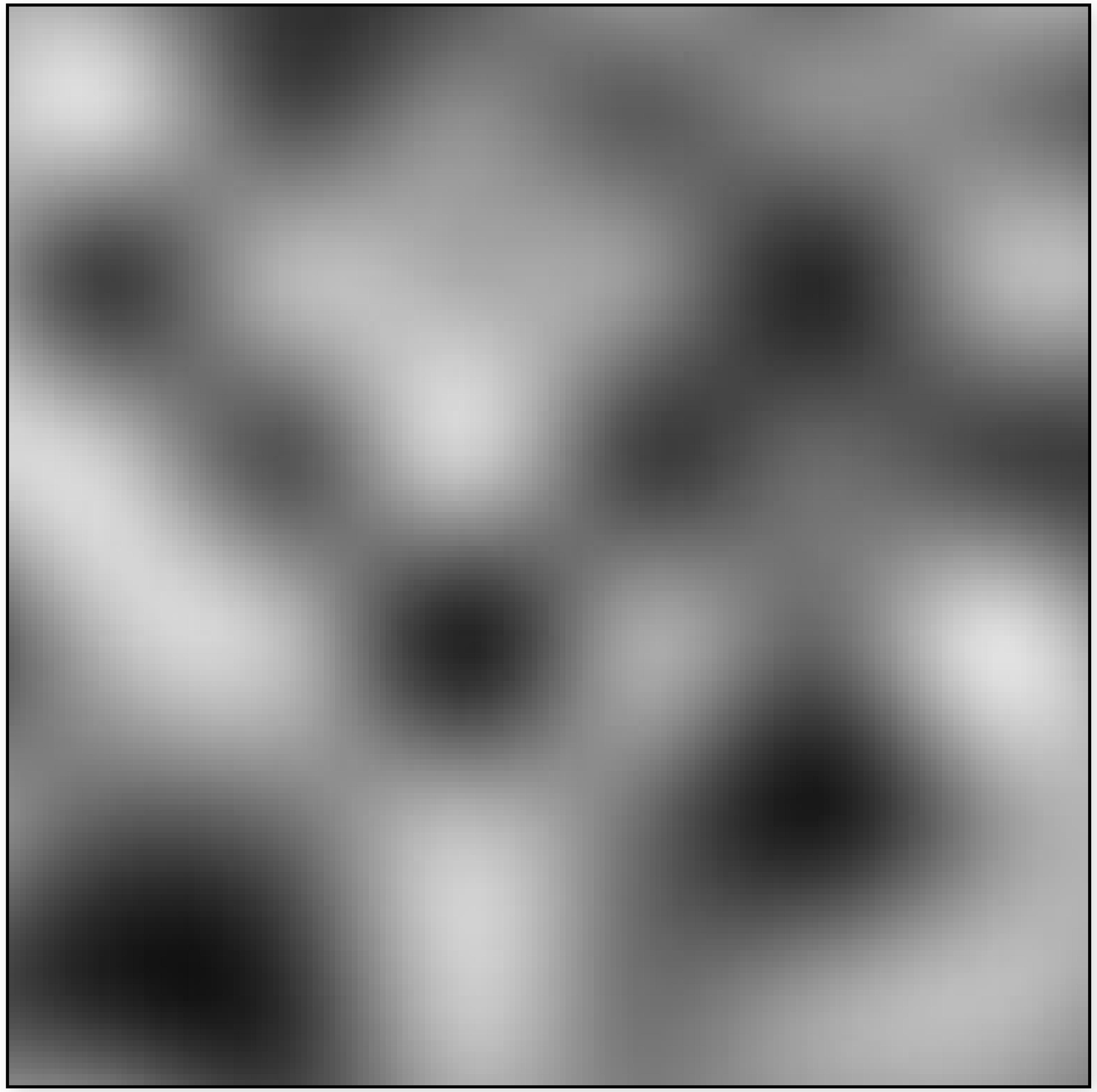


Interpolate between 2^n nearest grid points

Value noise example

(Bi-) cubic interpolation





Interpolate between 4^n nearest grid points

Value noise - implementation issues

Not feasible to store values at all integer locations

- pre-compute an array of pseudo-random values
- use a randomized hash function to map lattice locations to pseudo-random values

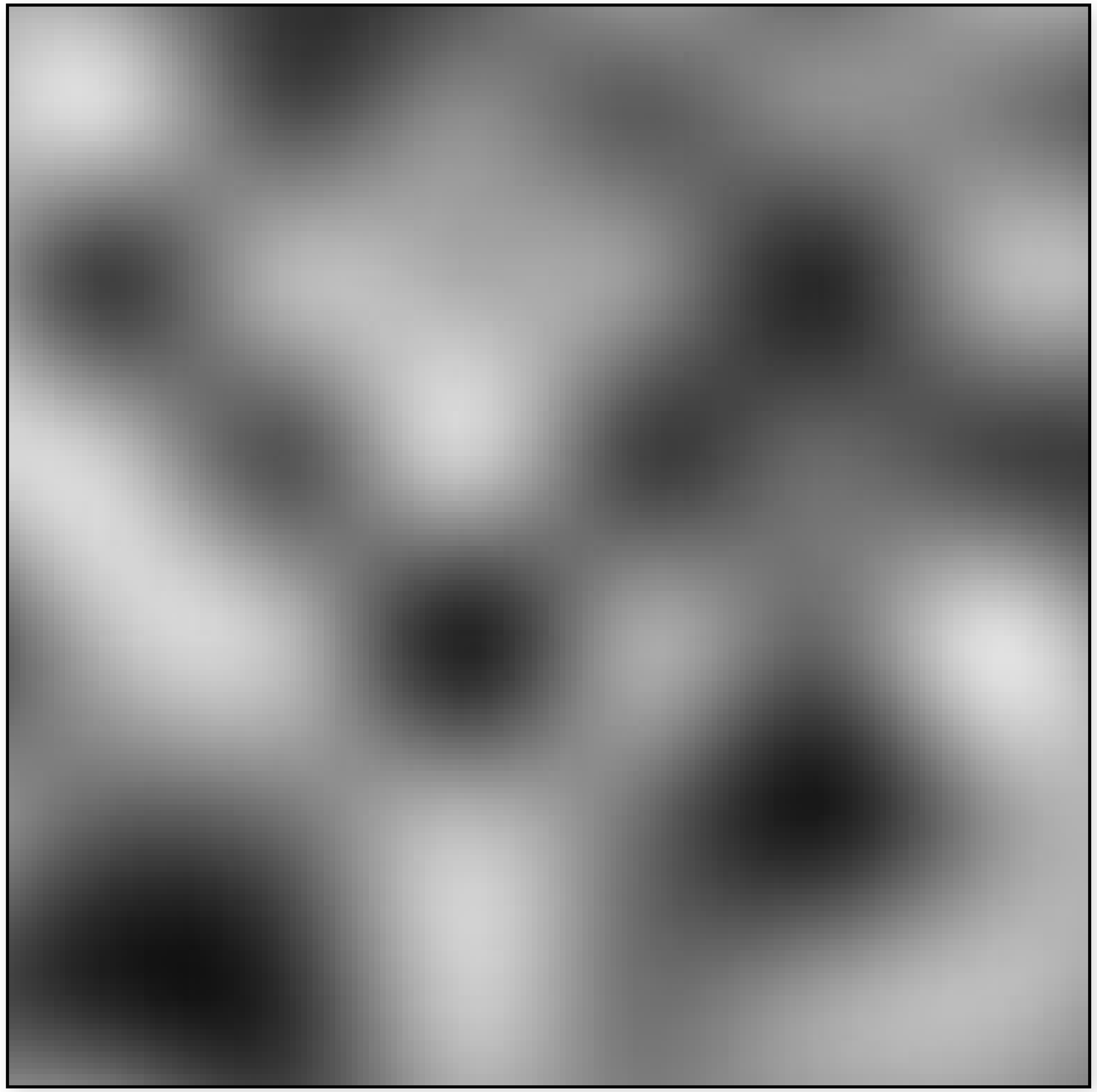
Value noise - implementation details

```
// randomly permuted array of 0...255, duplicated
const unsigned char values[256*2] = [1, 234, ...];

float noise1D(float x)
{
   int xi = int(floor(x)) & 255;
   return lerp(values[xi], values[xi+1], x-xi)/128.0-1;
}

// 2D hashing:
// values[xi + values[yi]];
// 3D hashing:
// values[xi + values[yi + values[zi]]];
// etc.
```

Value noise - limitations



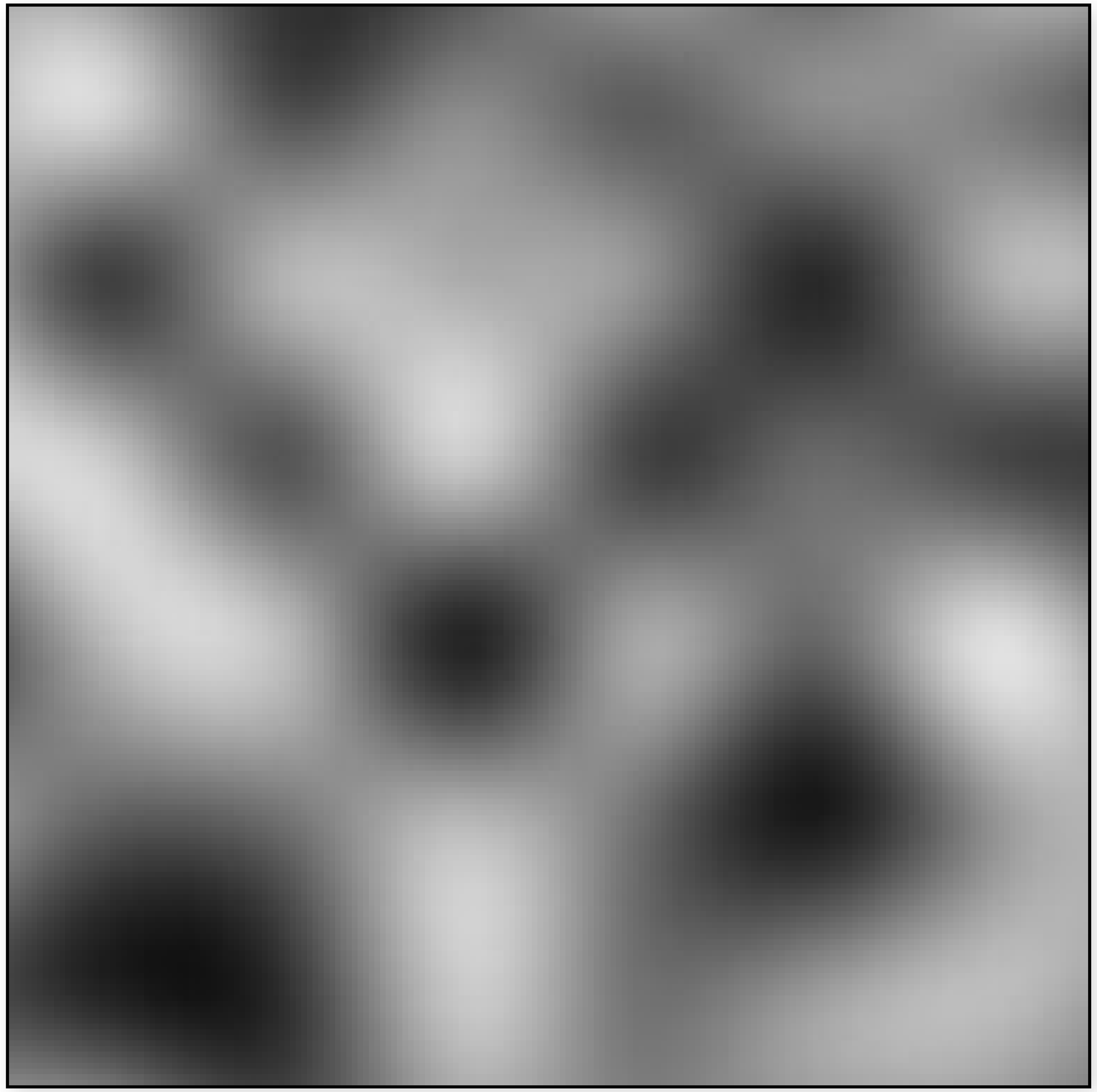
Value noise - limitations

Lattice structure apparent

- Minimal/maxima always on lattice

Slow/many lookups

- 8 values for trilinear
- 64 values for tricubic
    - 4^n for n dimensions



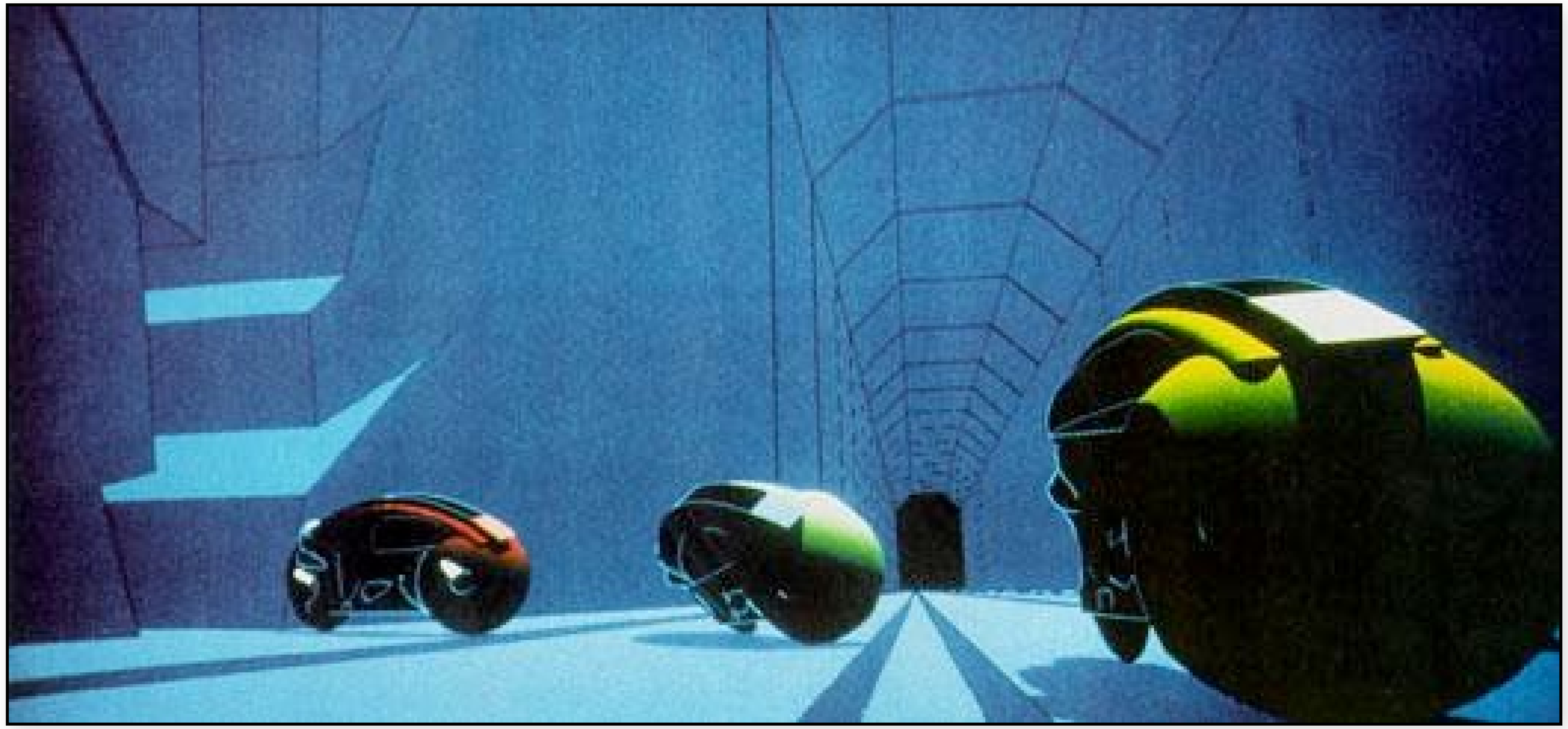
Perlin noise

Perlin noise, invented by Ken Perlin in 1982

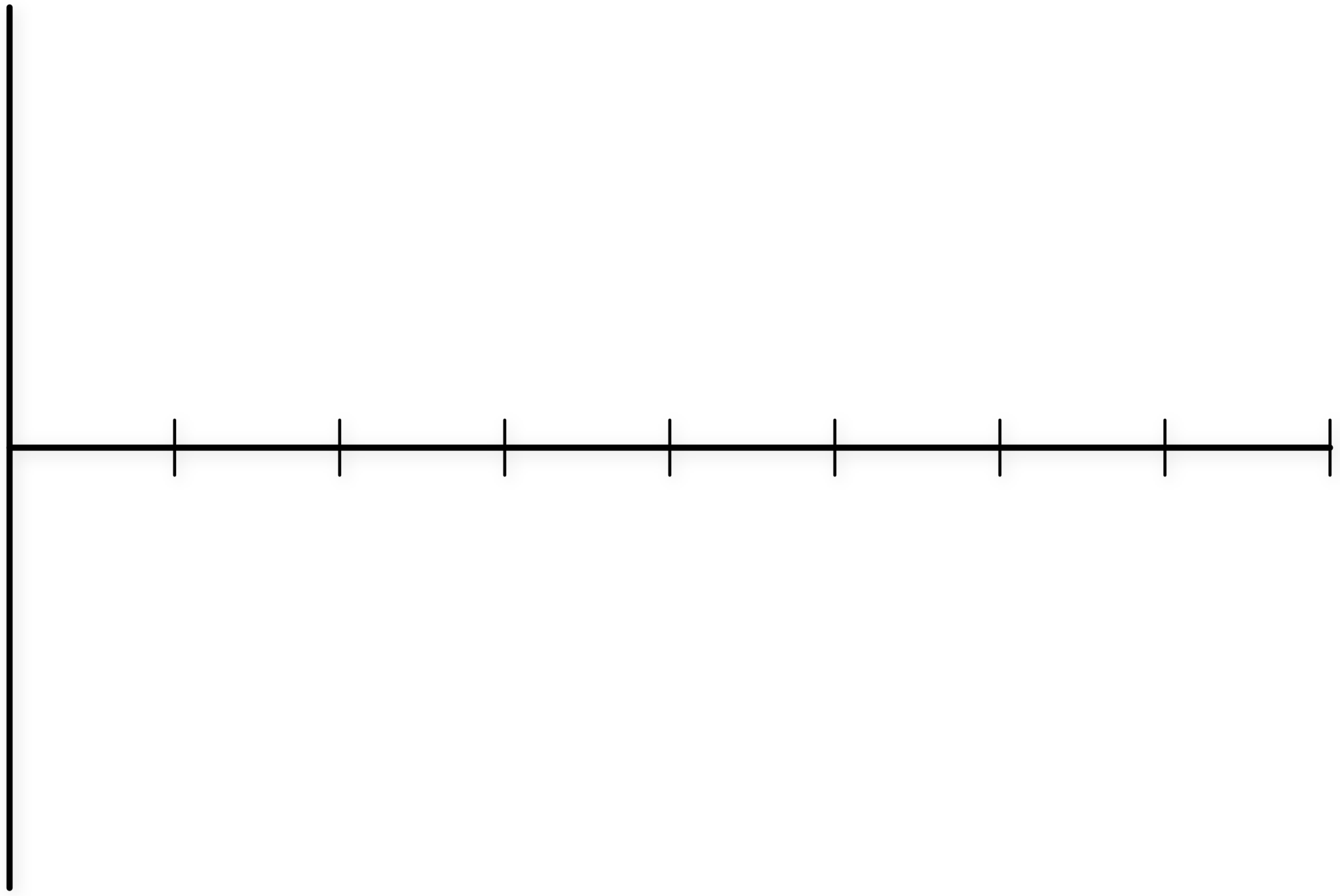
- First used in the movie Tron!

Store random vectors/gradients on lattice

- Use Hermite interp.
- a.k.a. "gradient noise"

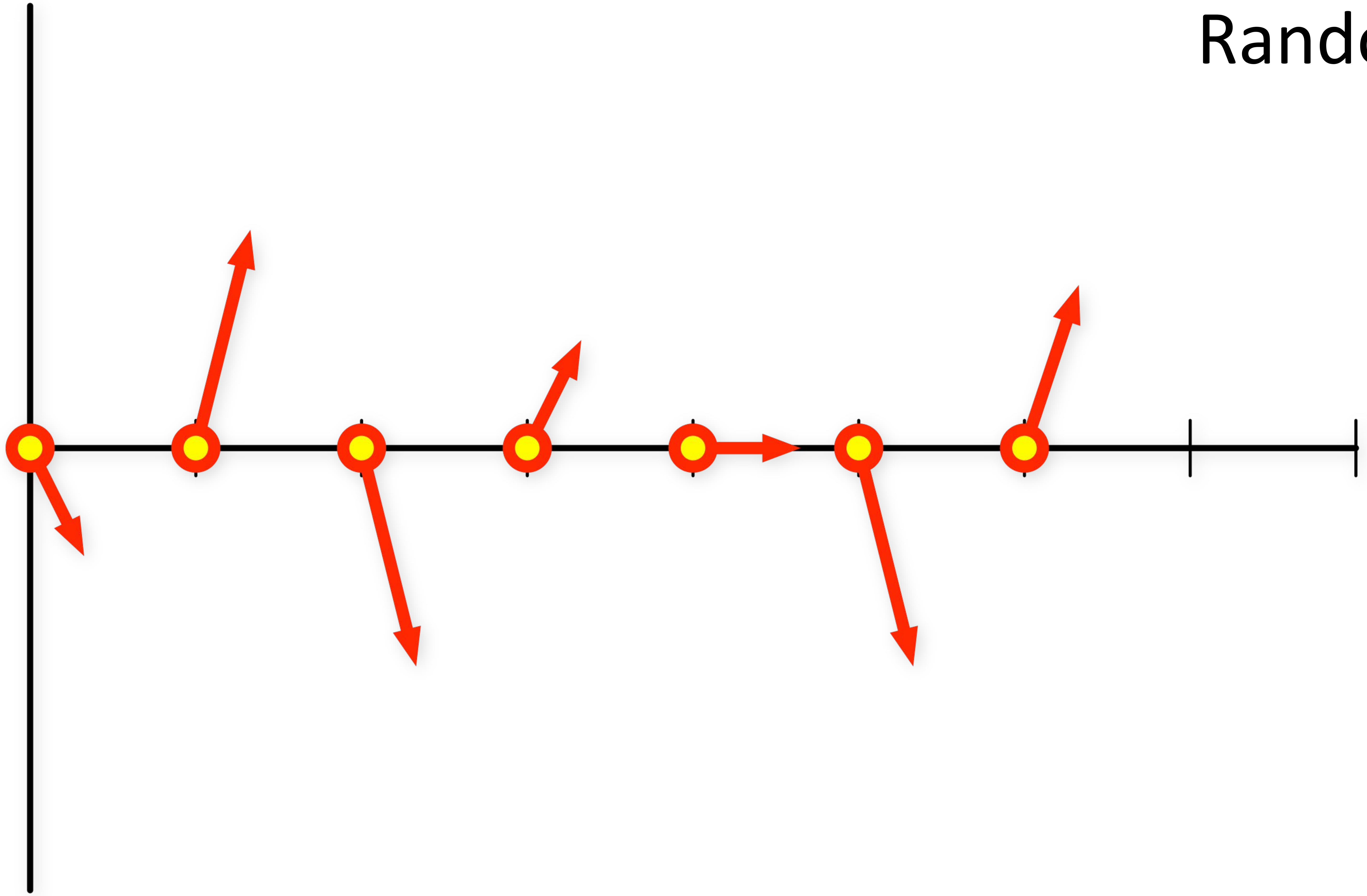


Classic Perlin noise



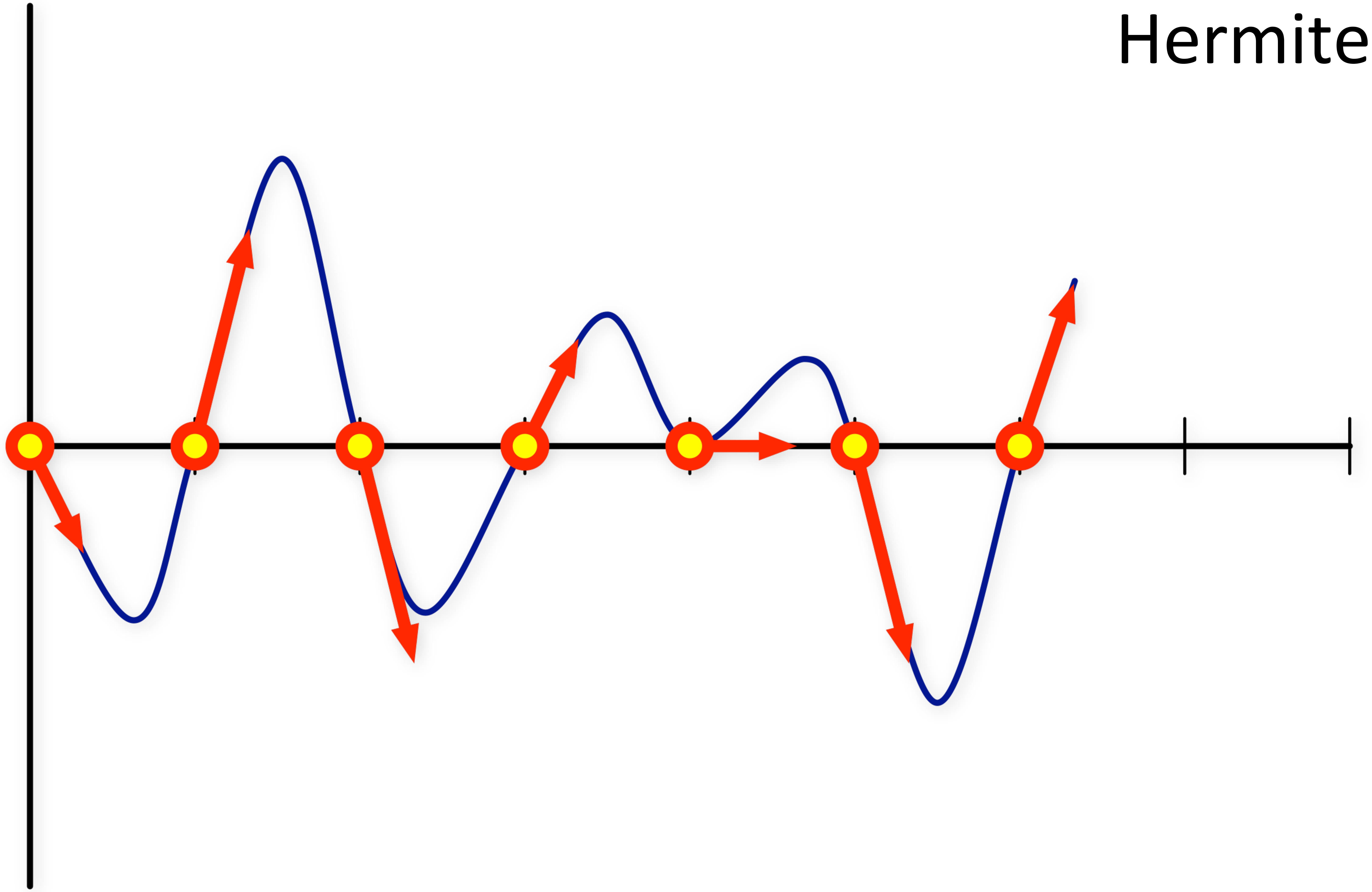
Classic Perlin noise

Random gradients on grid



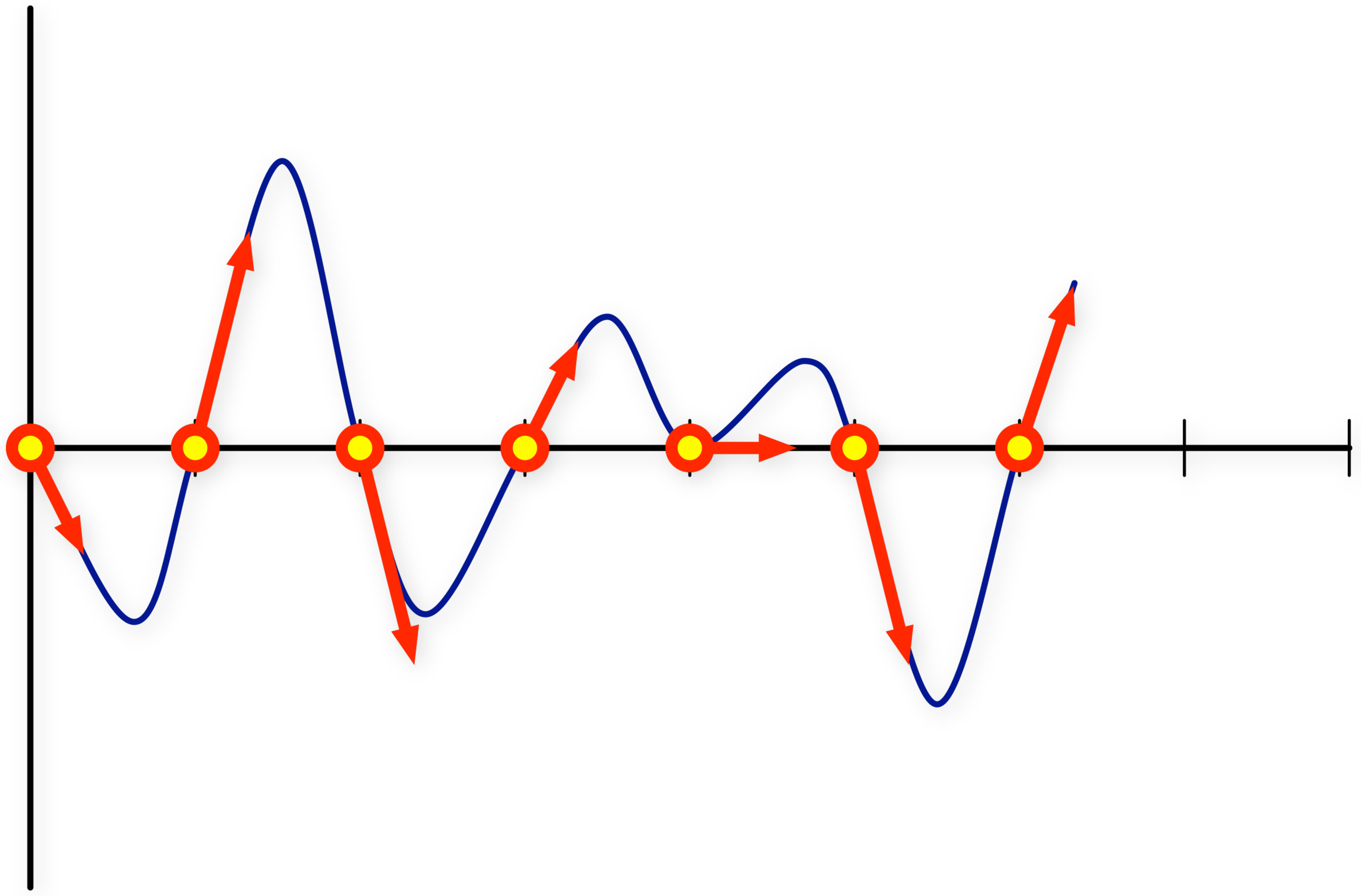
Classic Perlin noise

Hermite-interpolated values

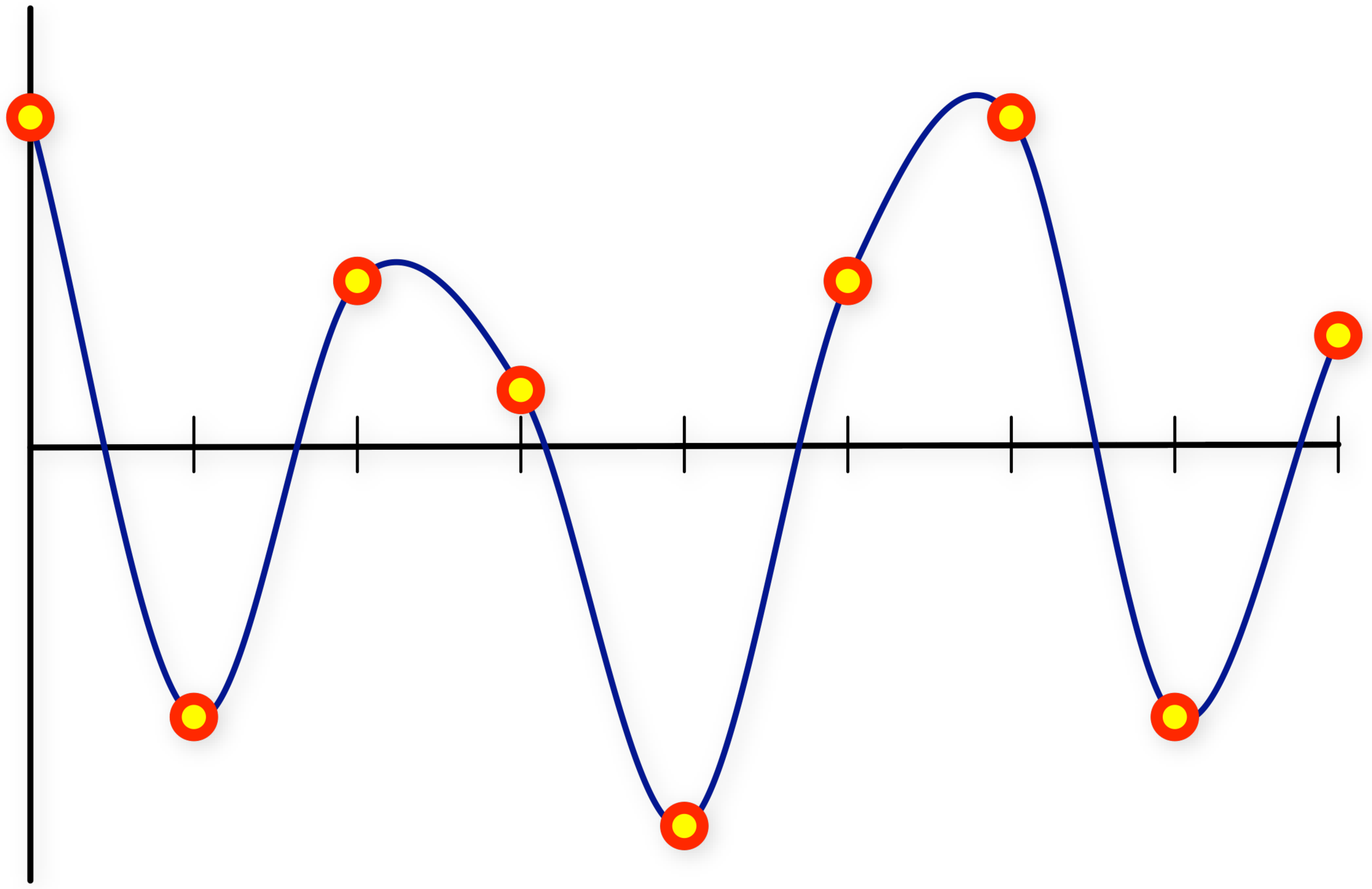


Perlin noise vs. value noise

Perlin Noise
(gradient noise)



Cubic Value Noise

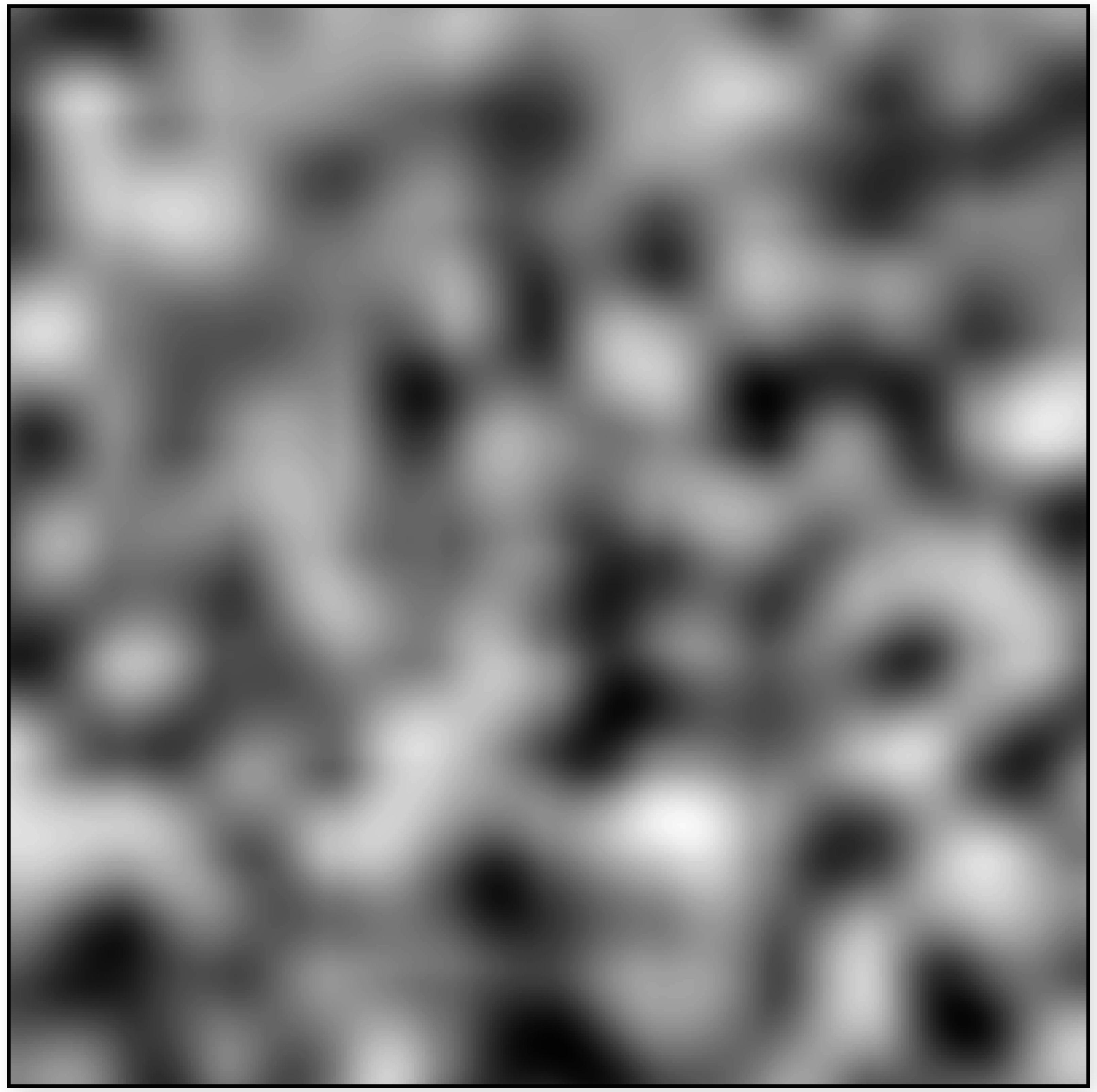


Why is Perlin noise better?

Perlin noise

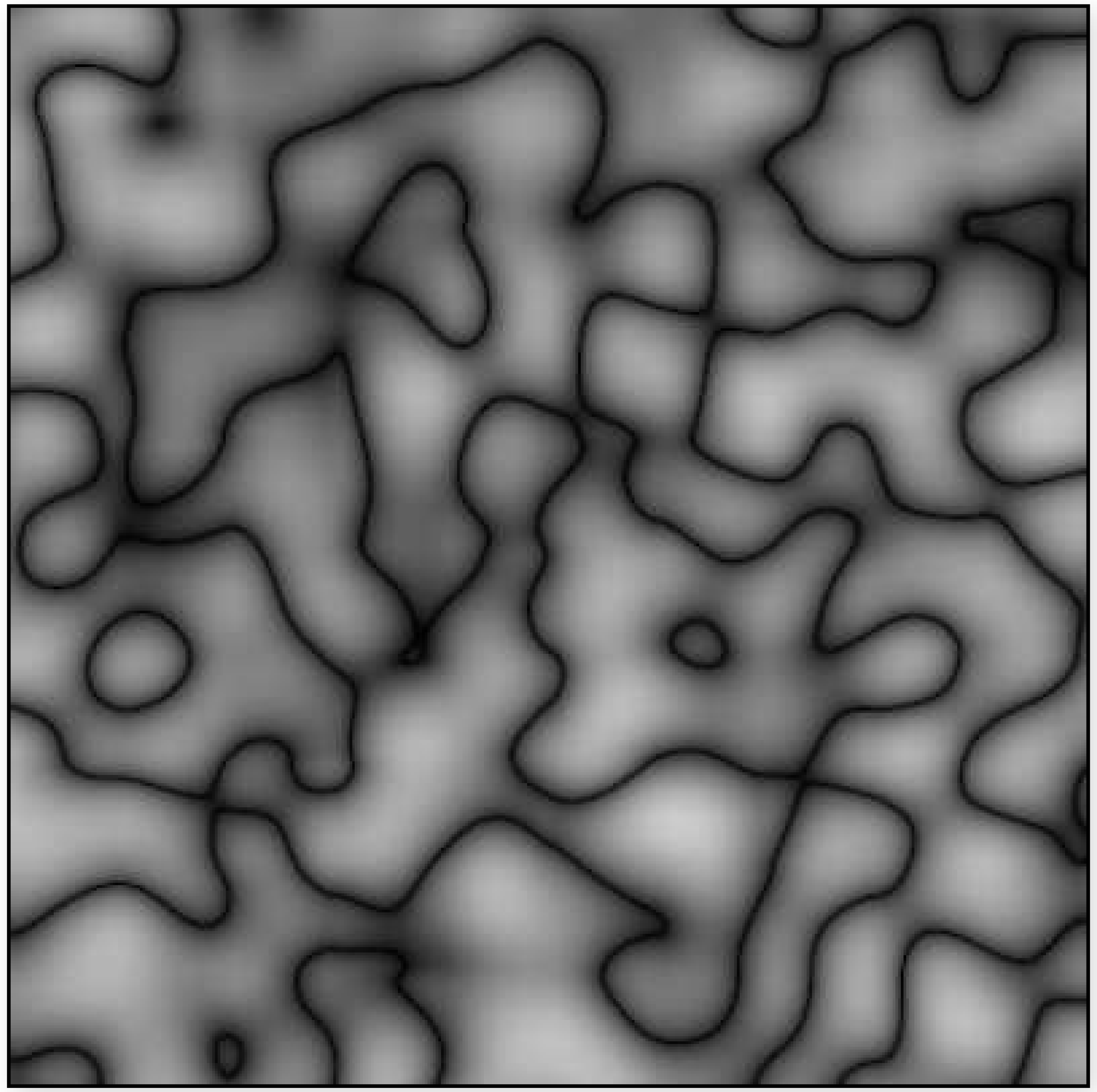
Typically signed by default, ~in [-1,1] with a mean of 0

offset/scale to put into [0,1] range



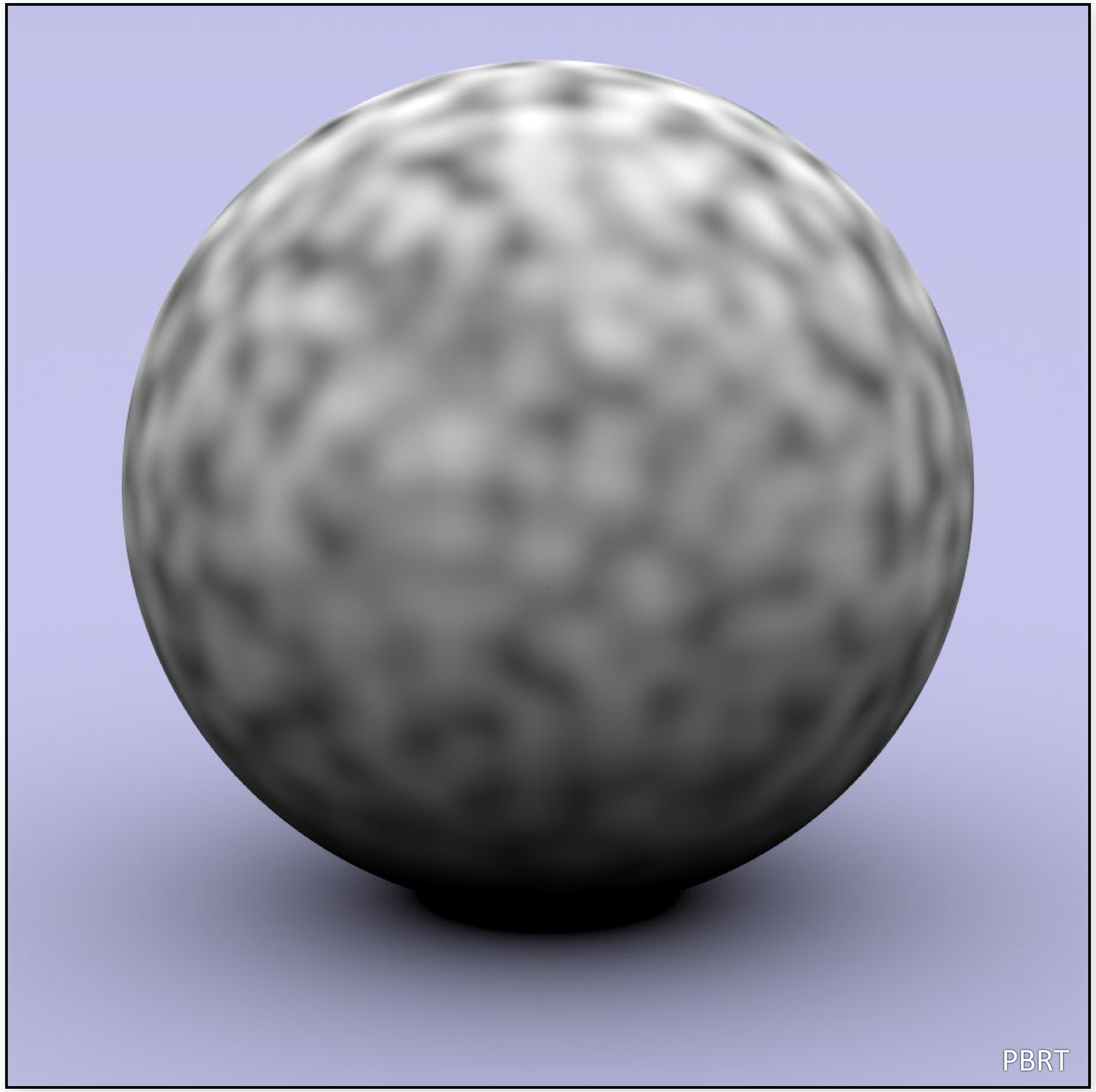
(noise(**p**)+1)/2

take absolute value

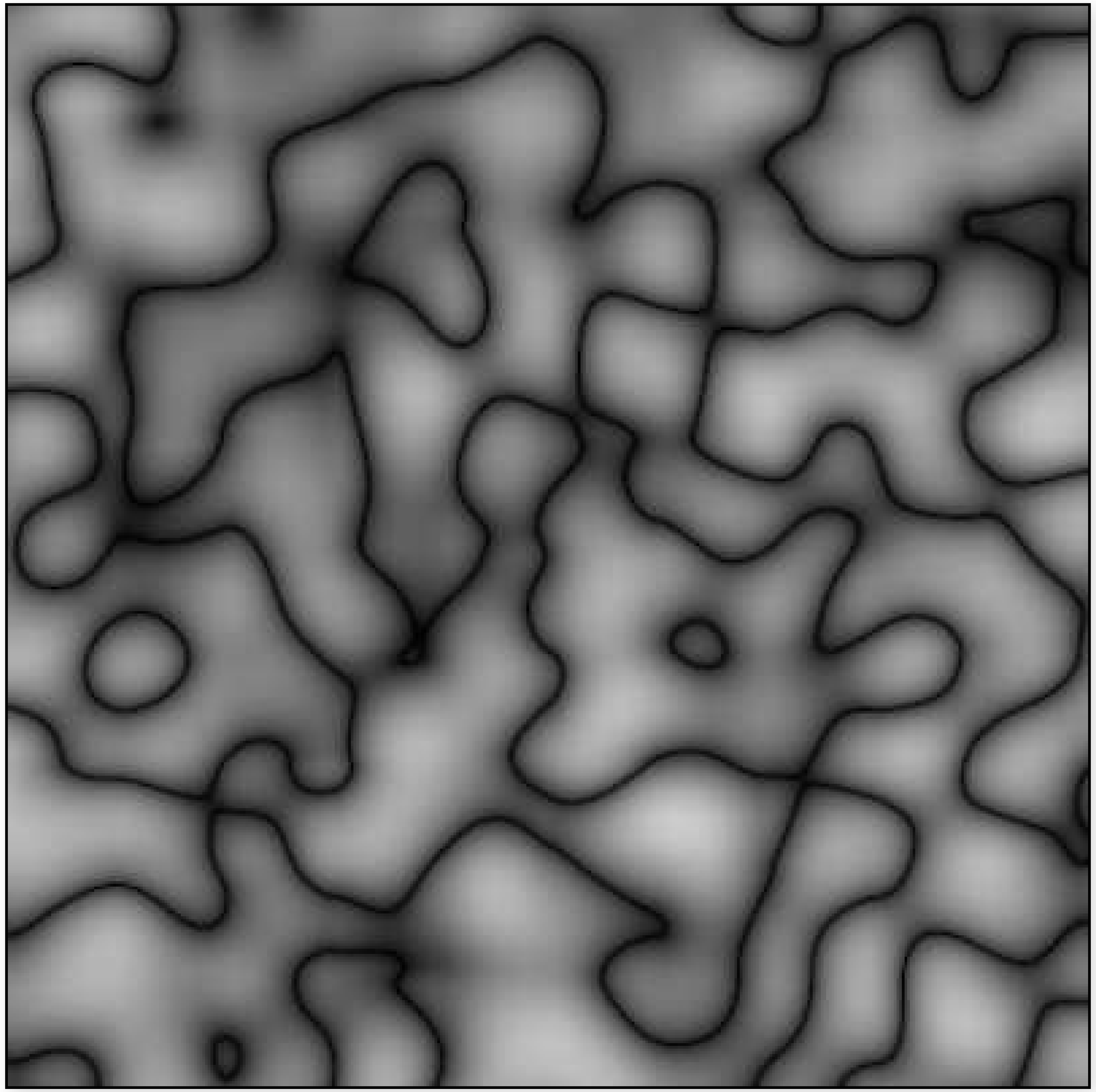


|noise(**p**)|

3D Perlin noise



Absolute value of noise



Perlin noise

Change frequency: ?

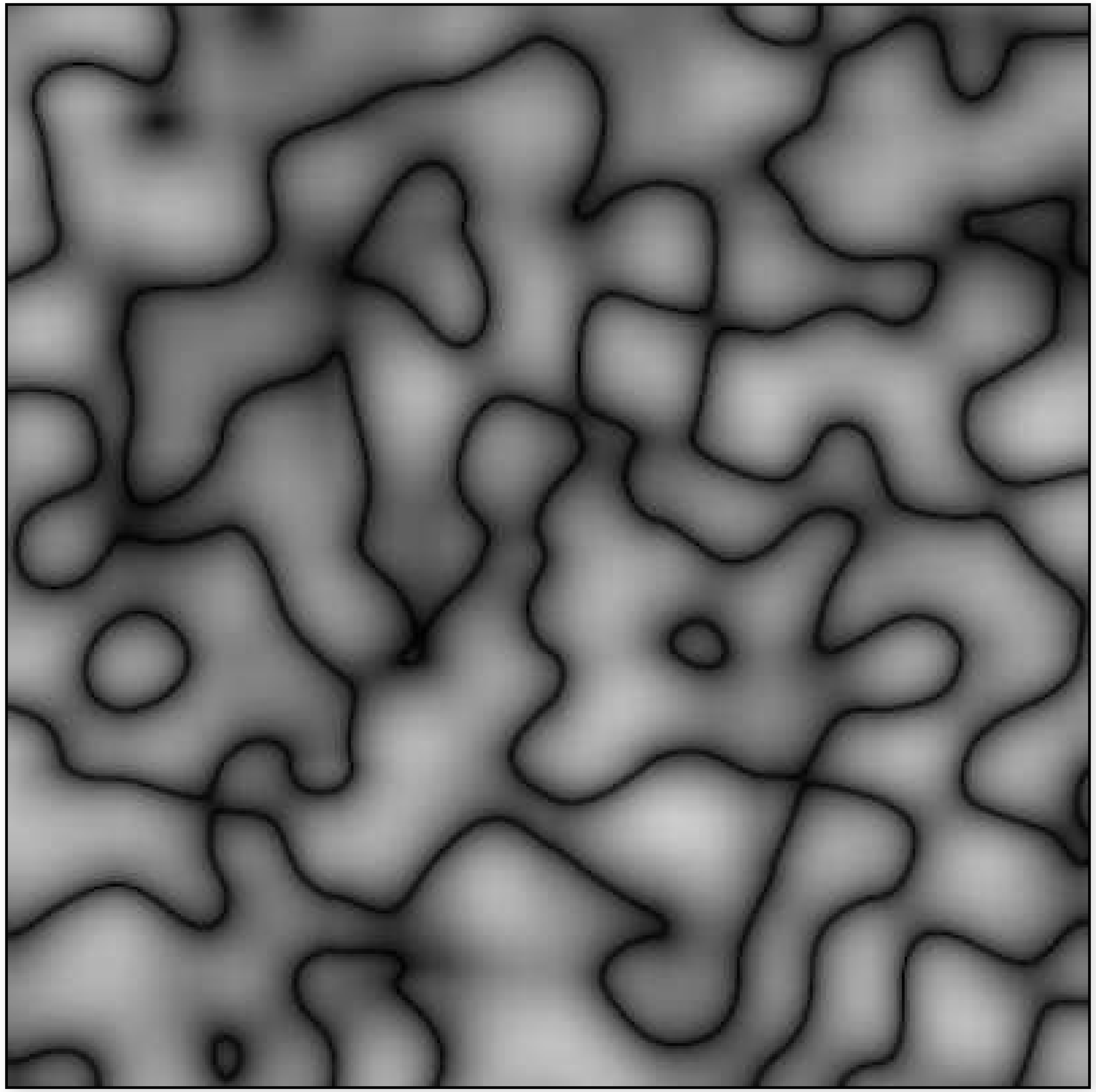
Change amplitude: ?

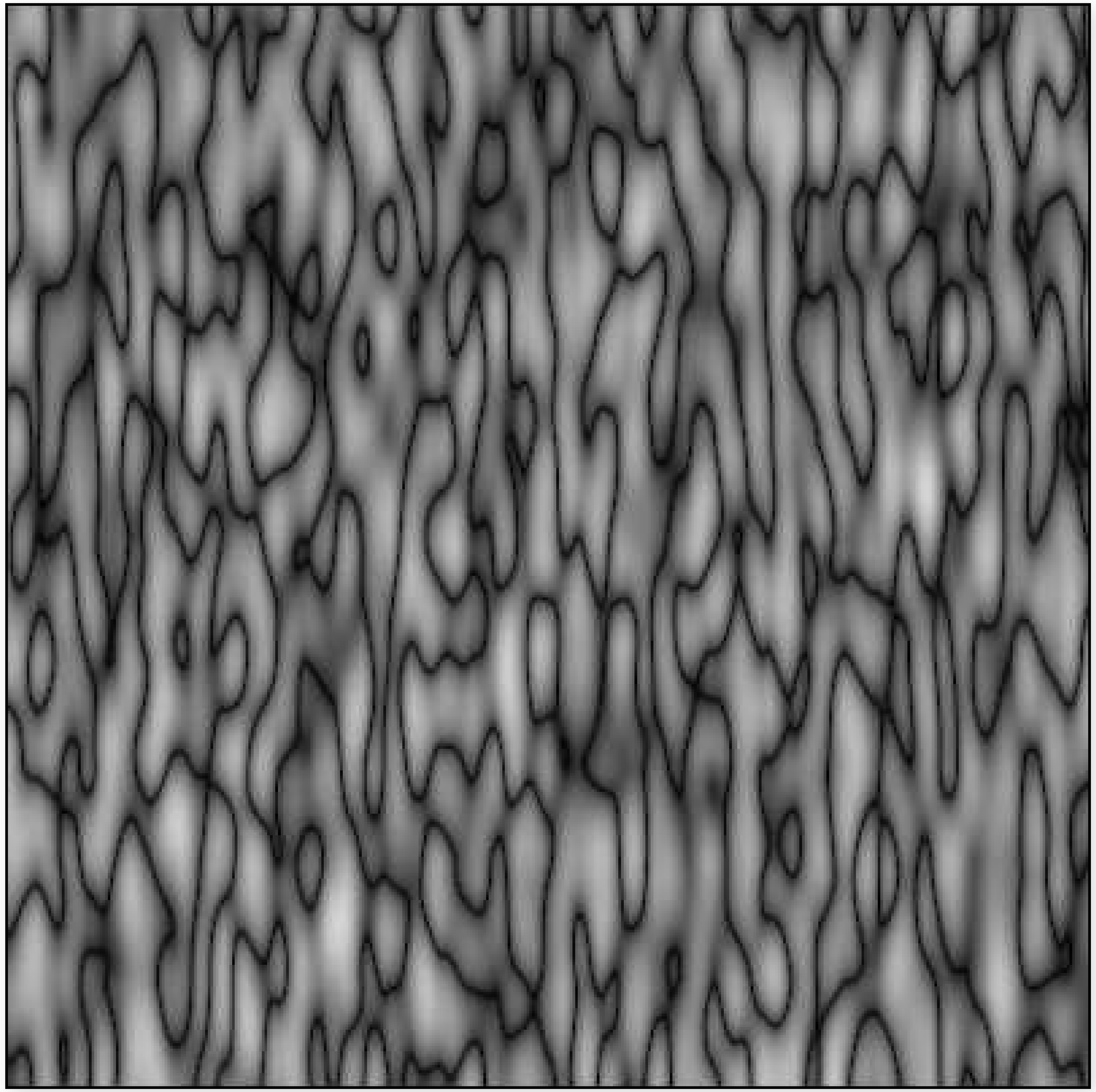
Perlin noise

Change frequency: noise(10*x)

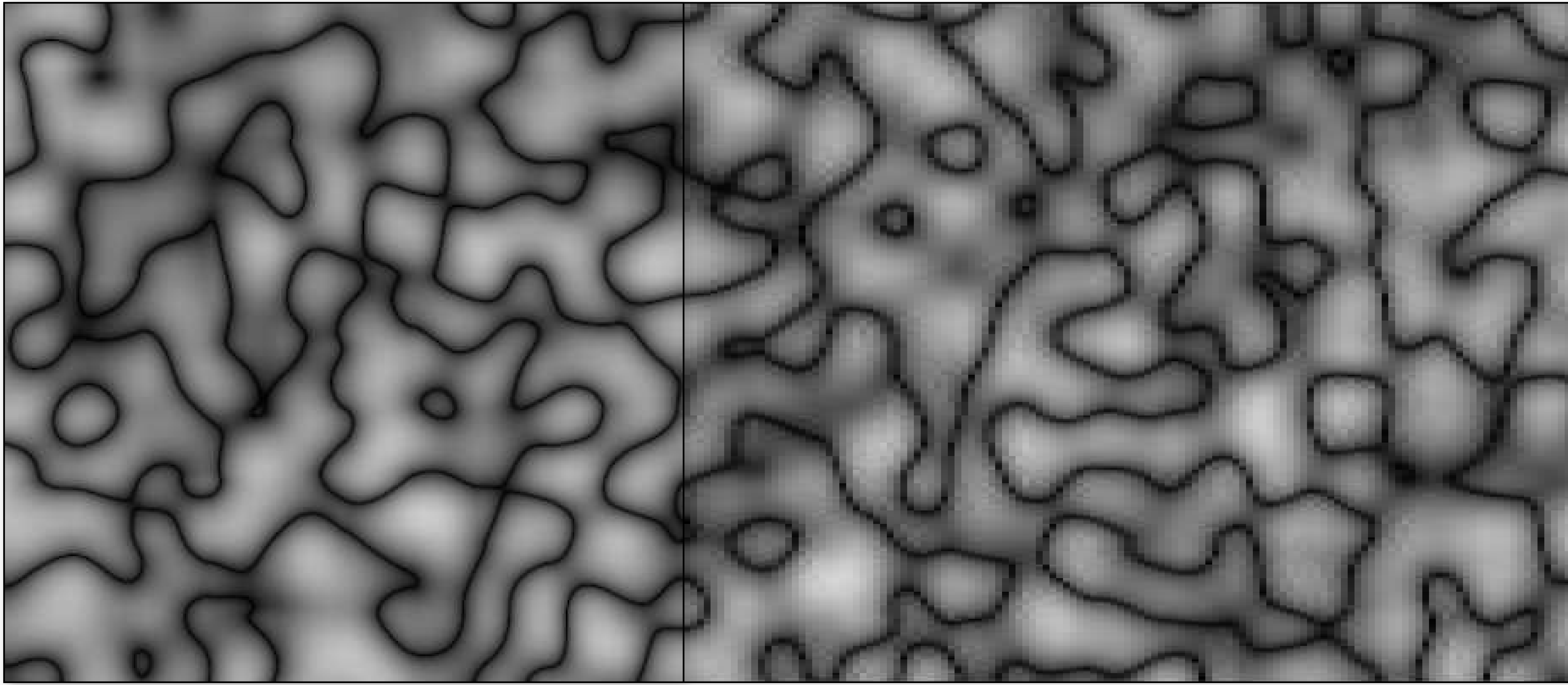
Change amplitude: 10*noise(x)

Absolute value of noise

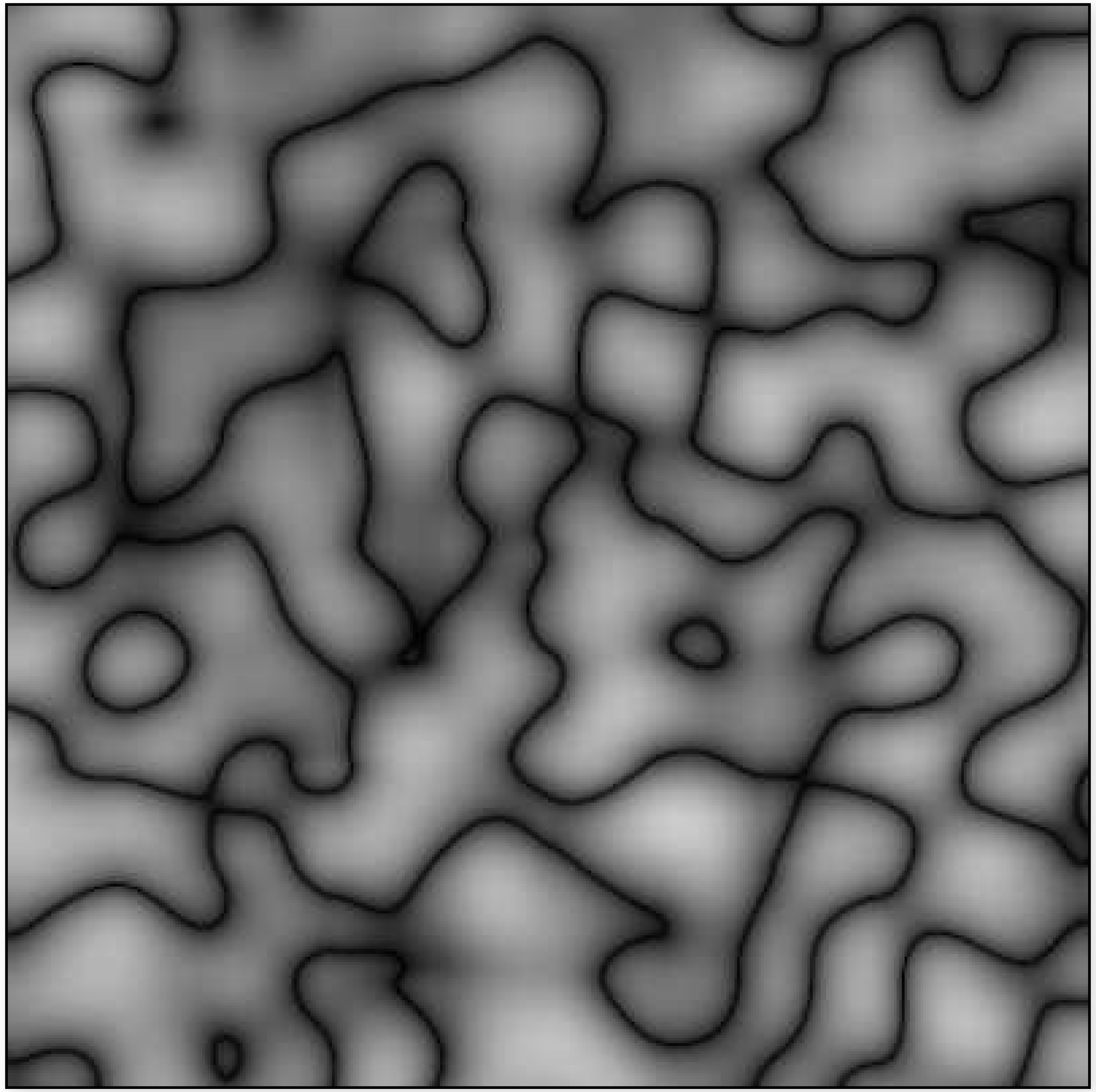


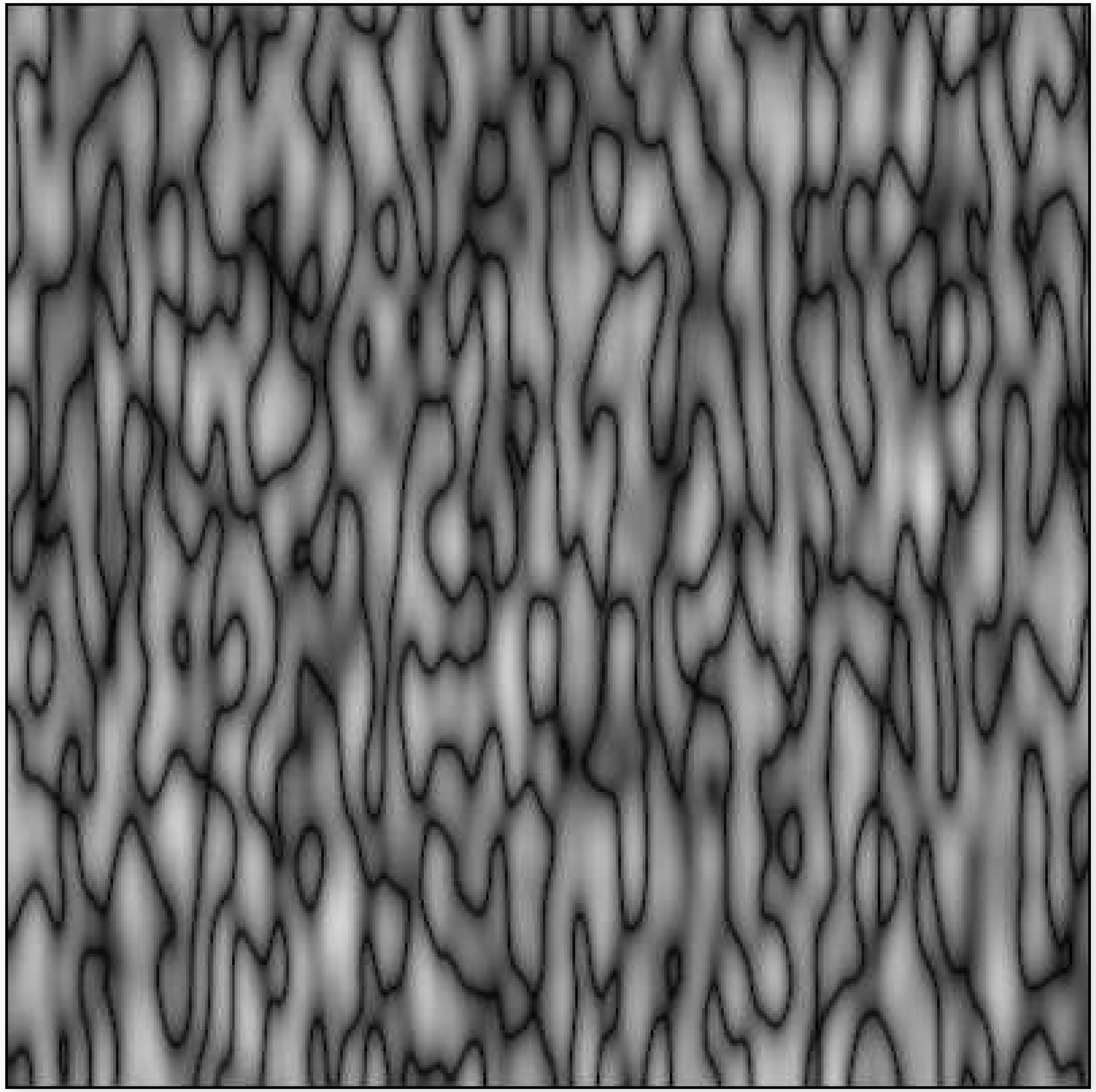


Absolute value of noise

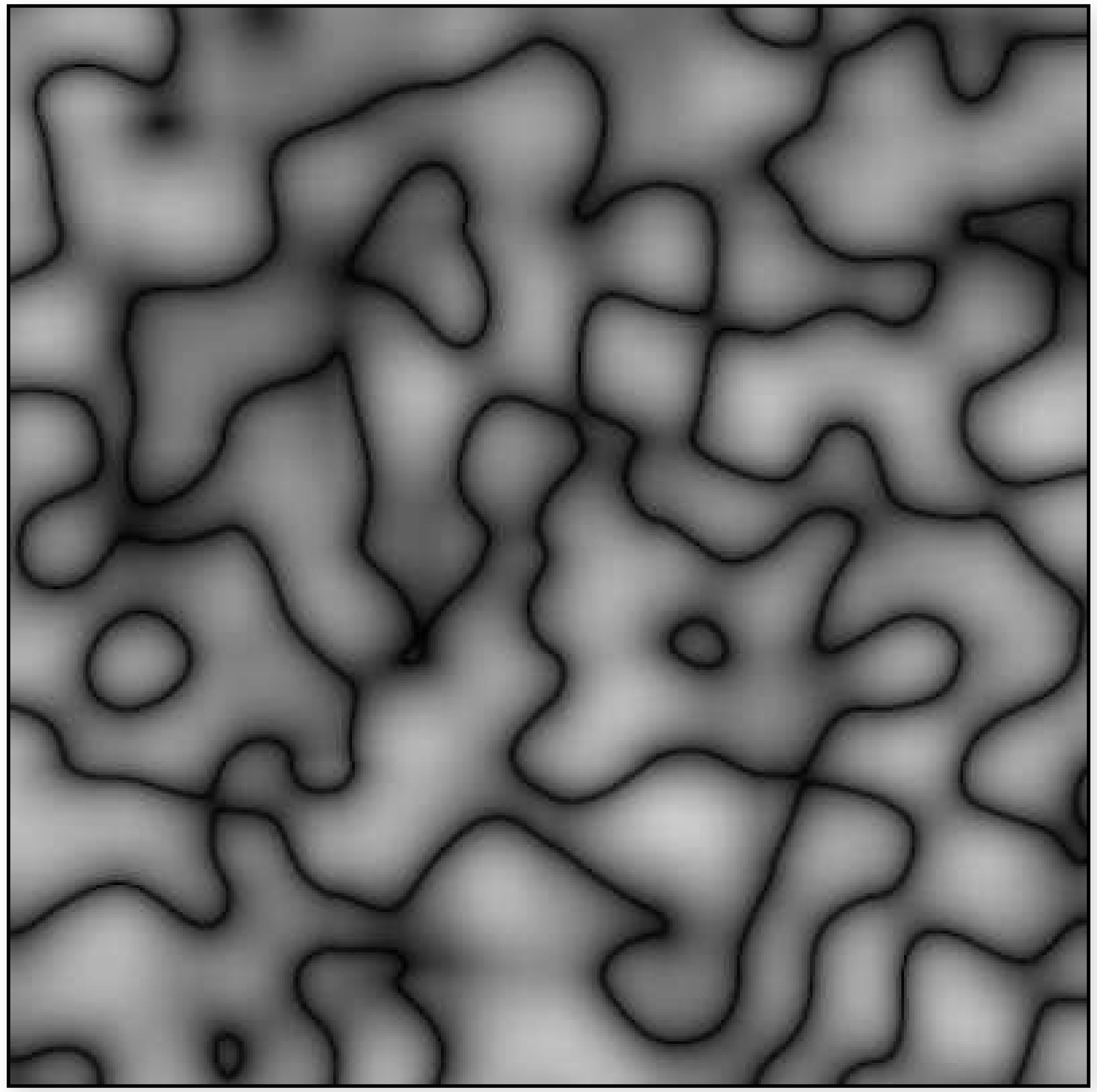


Absolute value of noise

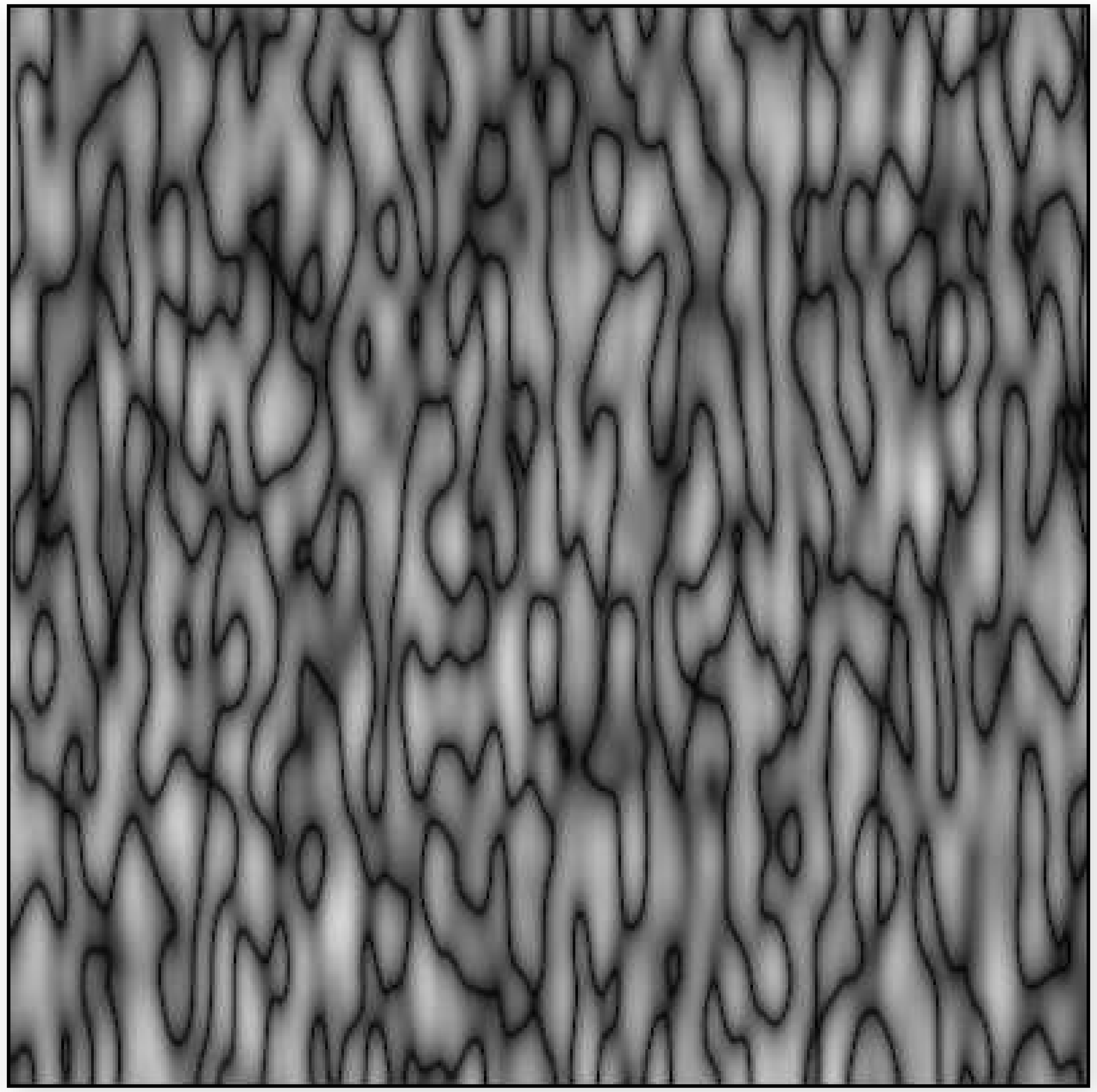




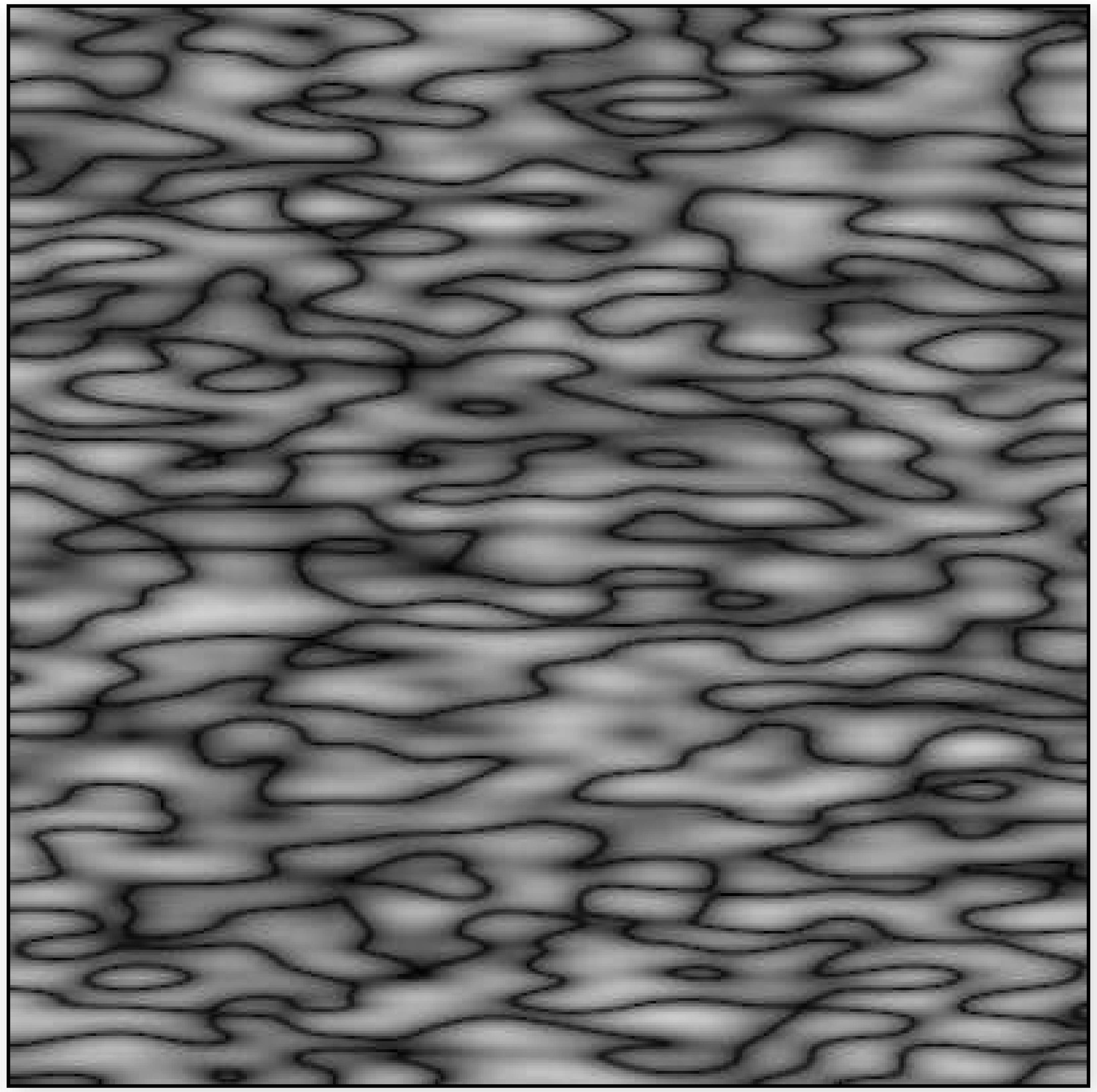
Absolute value of noise



$|\text{noise}(\mathbf{p})|$

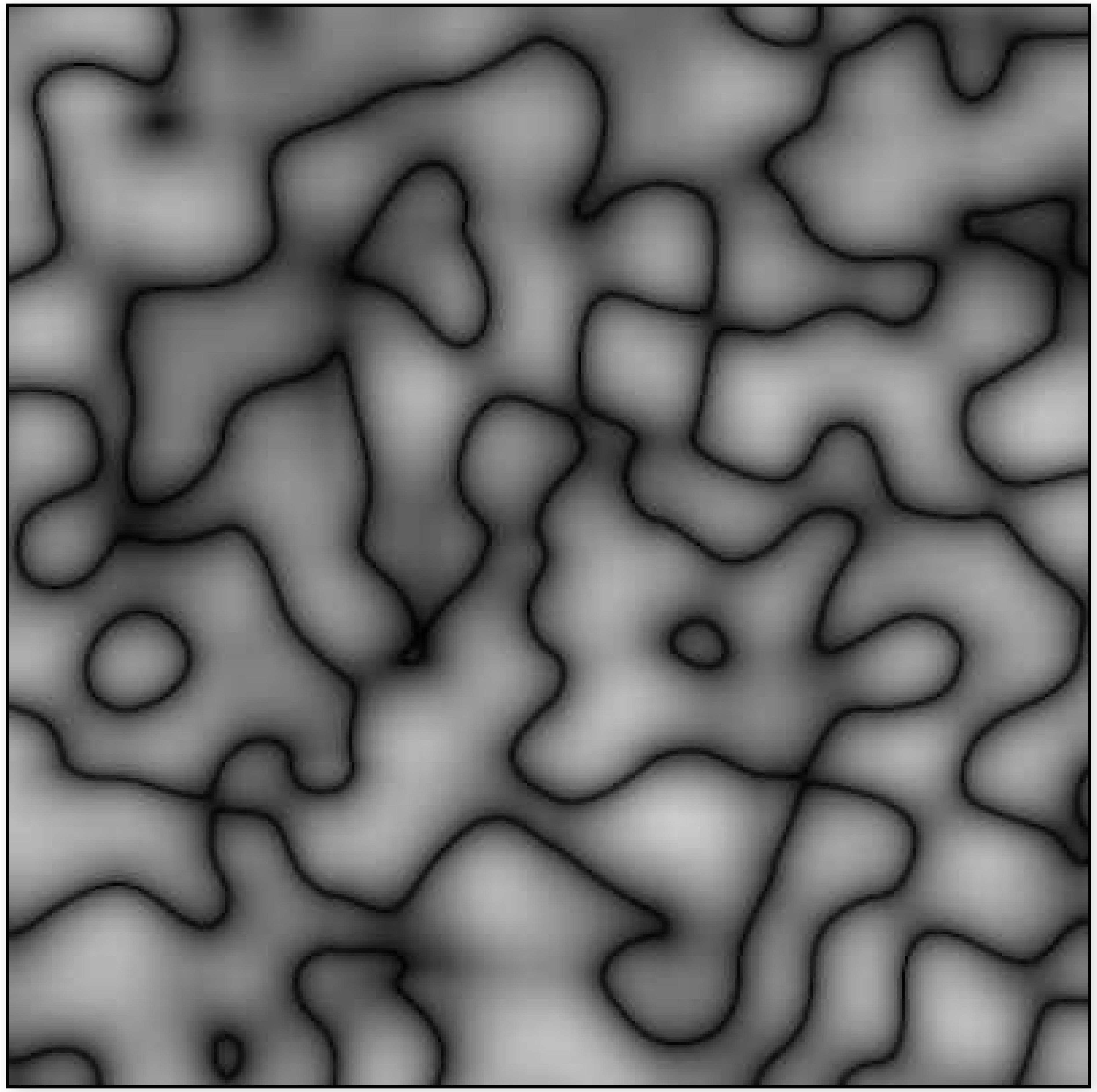


$|\text{noise}(4\,\mathbf{p}_x,\mathbf{p}_y,\mathbf{p}_z)|$



$|\text{noise}(\mathbf{p}_x,4\,\mathbf{p}_y,\mathbf{p}_z)|$

Perlin noise - limitations



Perlin noise - limitations

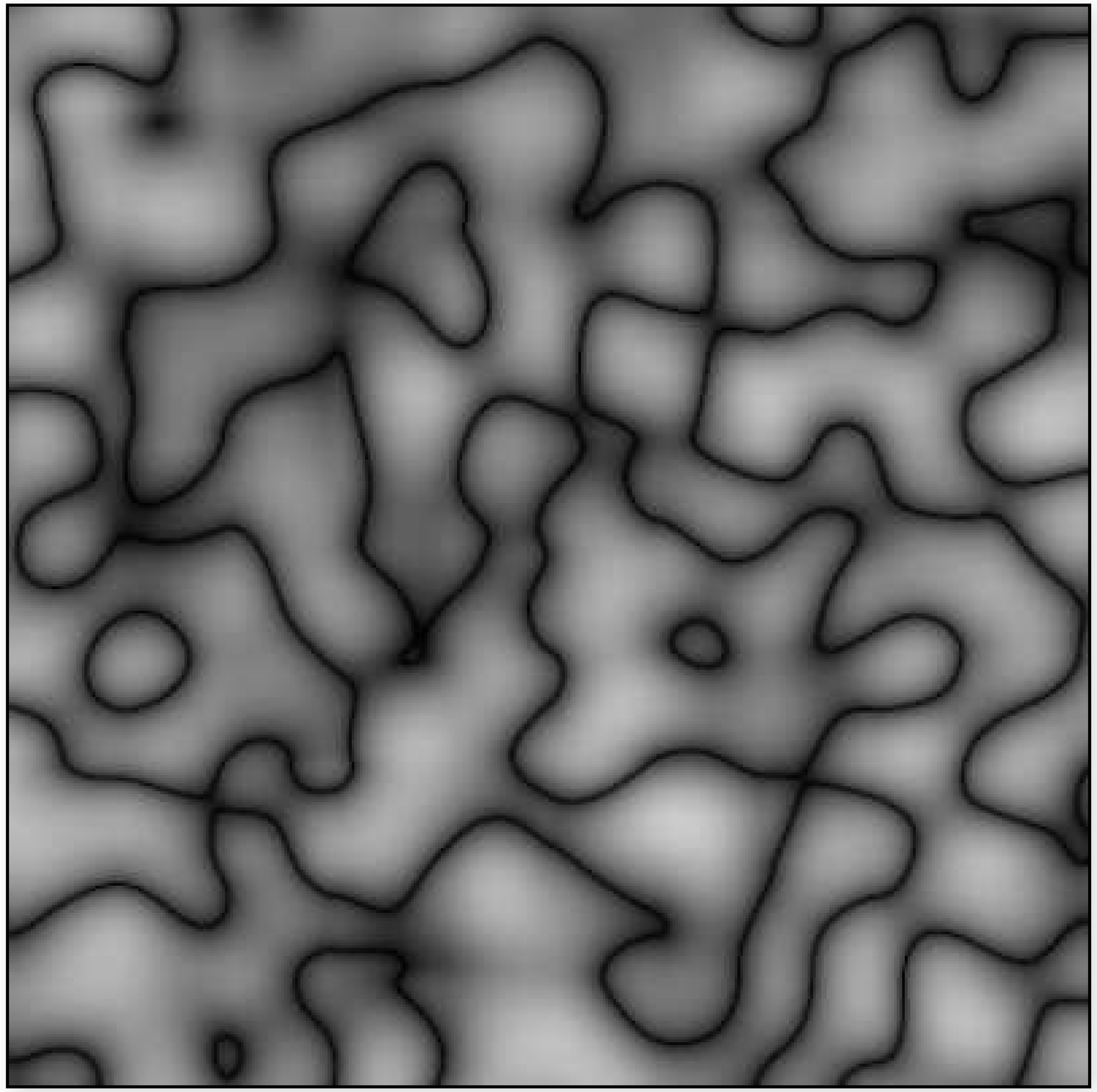
Lattice structure apparent for |noise|

- all lattice locations have value 0

Lookups faster, but still slow:

- Perlin is 2^n for *n* dimensions instead of 4^n for value noise
- other variations: simplex noise (O(*n*))

Not quite rotation invariant



More reading

Fantastic explorable explanation by Andrew Kensler at Pixar

- eastfarthing.com/blog/2015-04-21-noise

Spectral synthesis

Representing a complex function $f_s(\mathbf{p})$ by a sum of weighted contributions from a scaled function $f(\mathbf{p})$:

$$f_s(\mathbf{p}) = \sum_i w_i f(s_i\, \mathbf{p})$$

Called a “fractal sum” if w_i and s_i are set so:

- increasing frequencies have decreasing amplitude, e.g.: $w_i = 2^{-i}$, $s_i = 2^i$
- when $s_i = 2^i$, each term in summation is called an “octave”

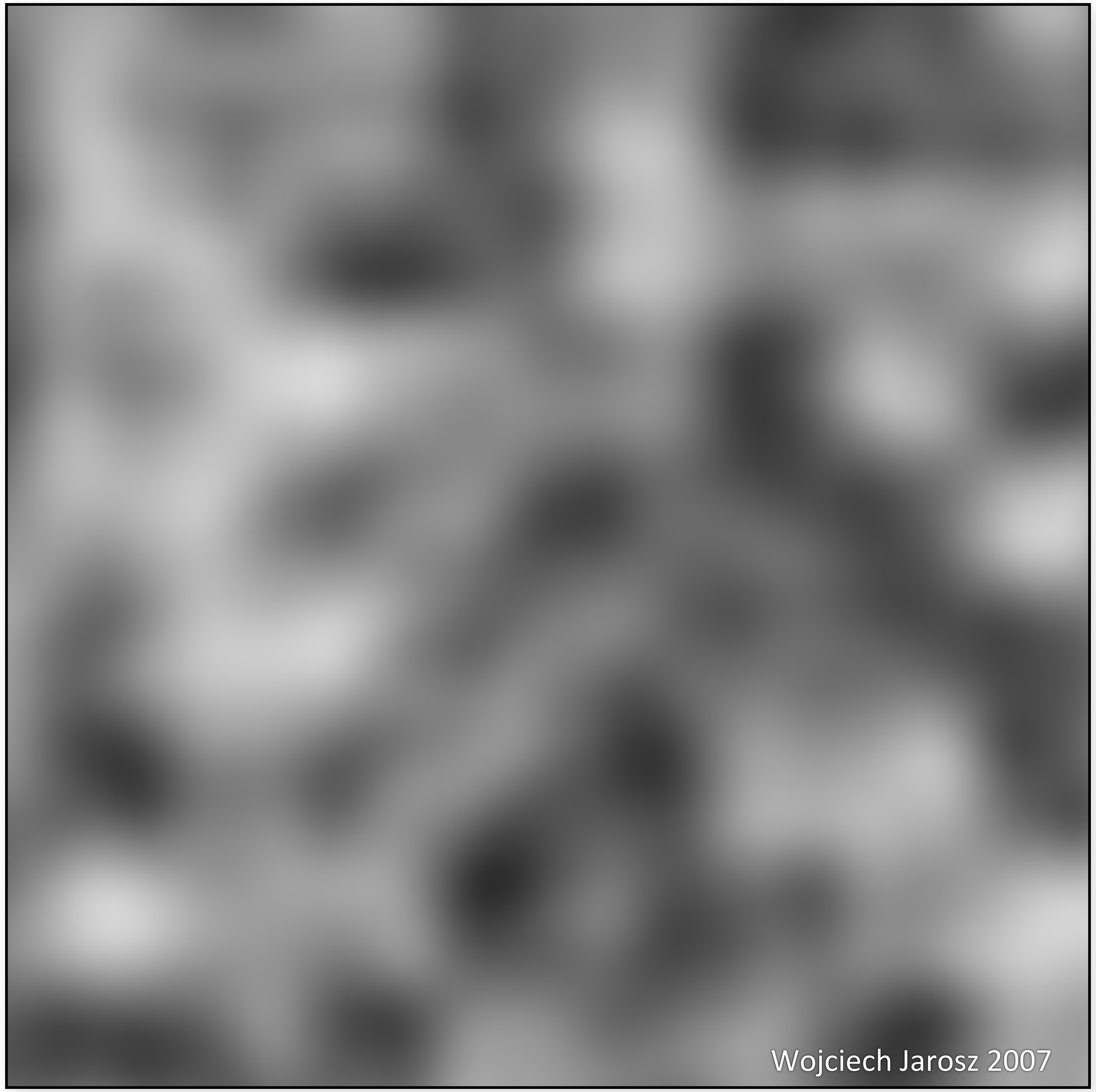
What function $f(\mathbf{p})$ should we use?

fBm - fractional Brownian motion

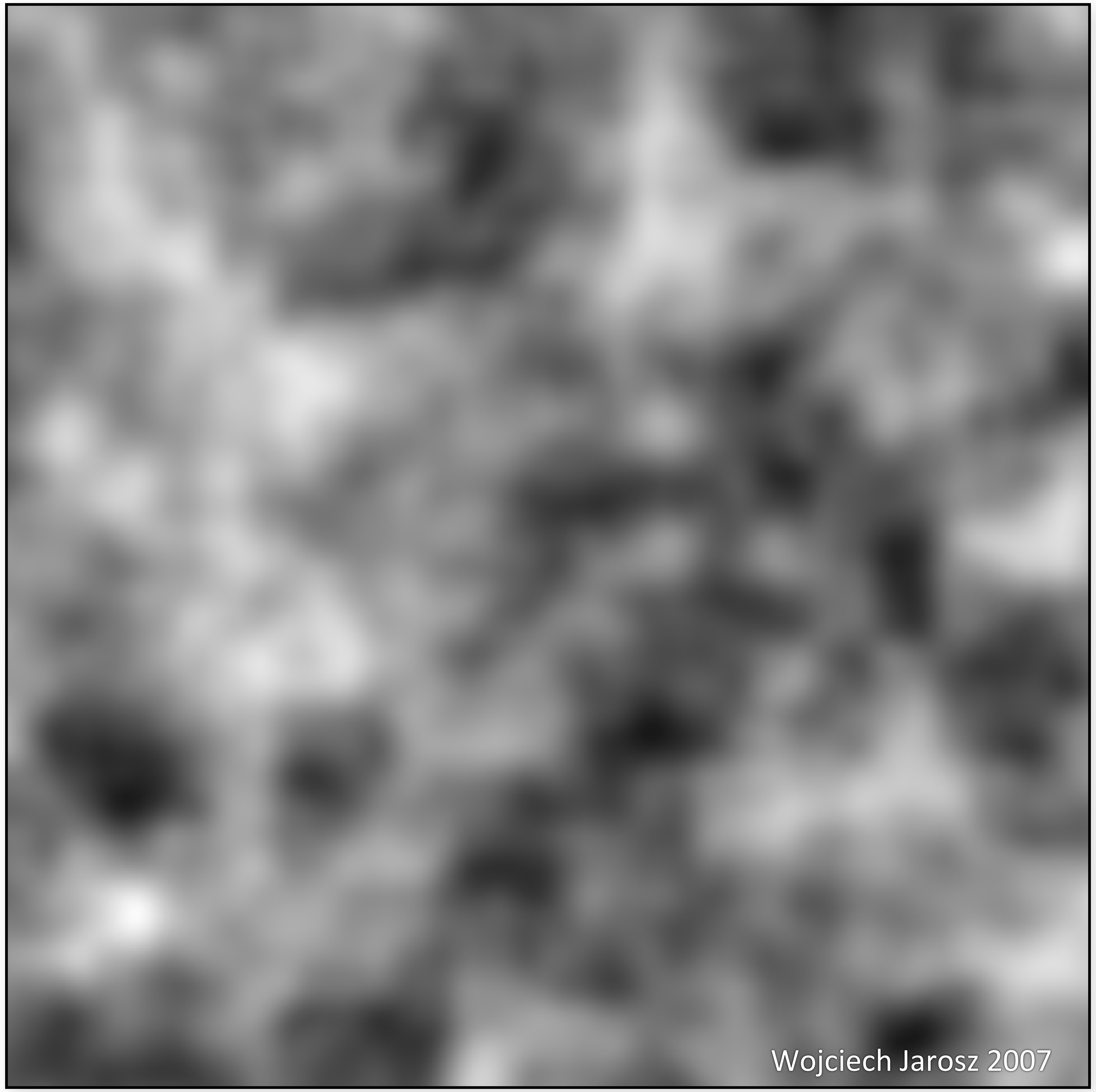
In graphics:

- Fractal sum of Perlin noise functions
- “Fractal noise”

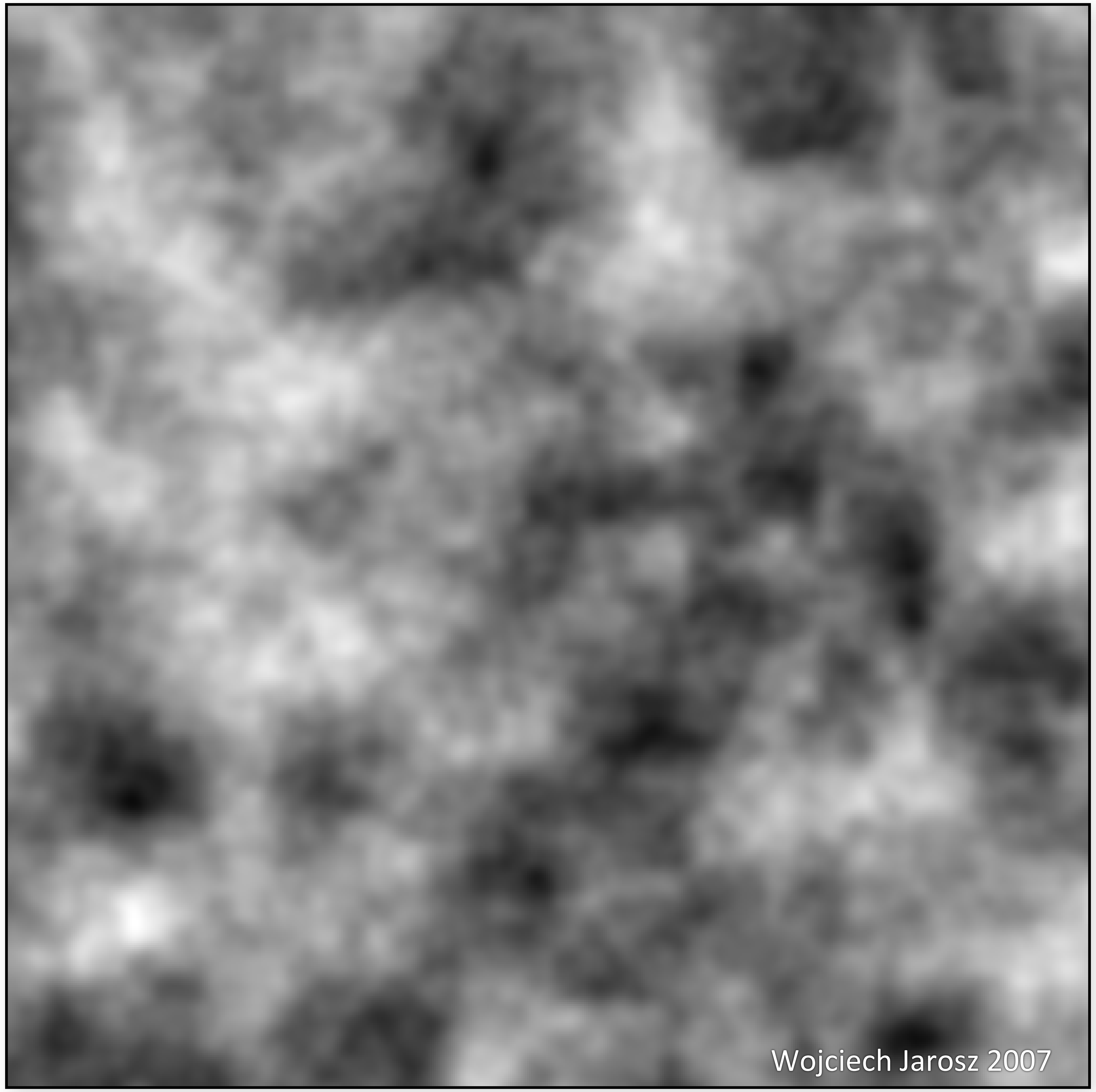
fBm - 1 octave



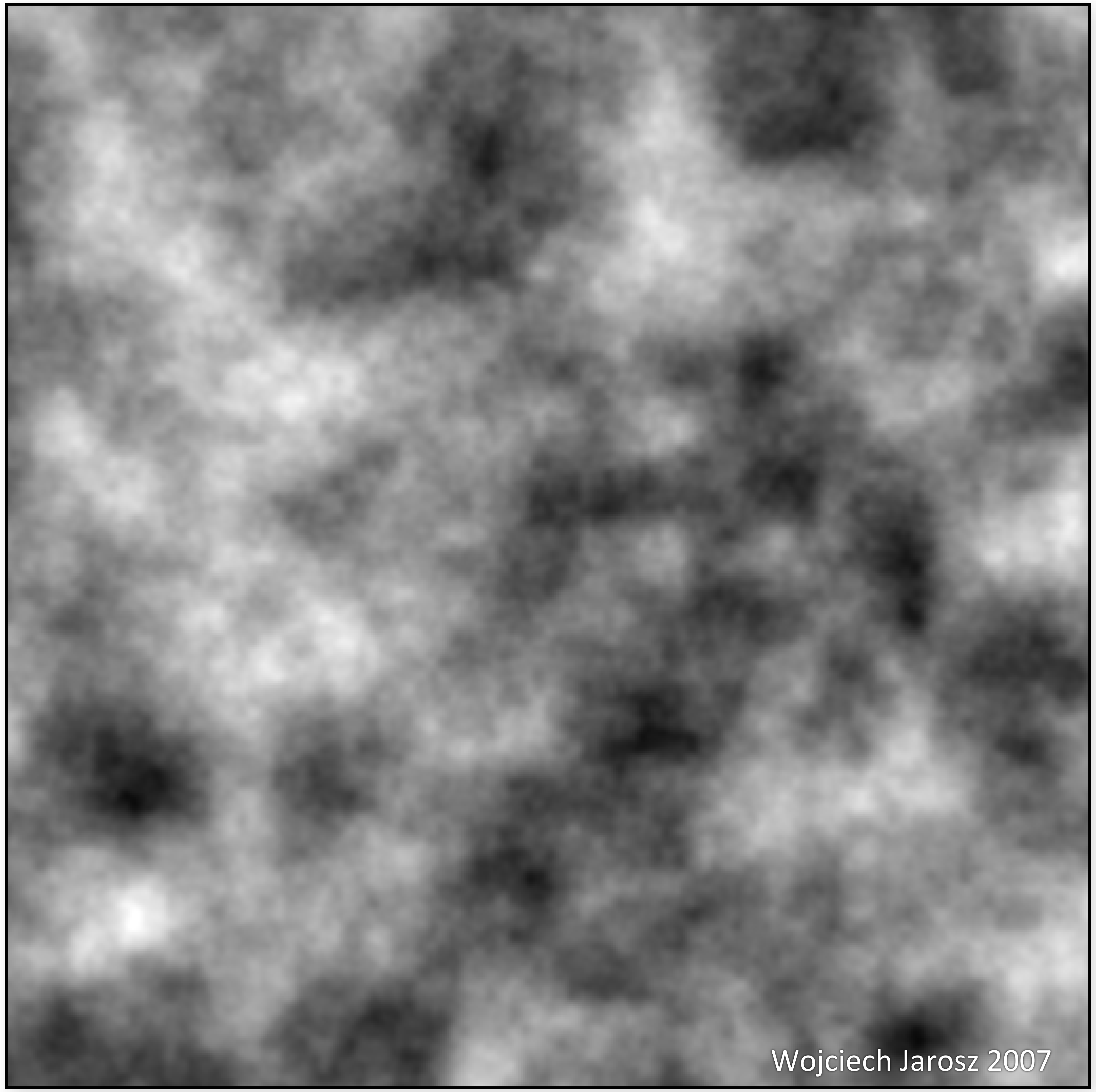
fBm - 2 octaves



fBm - 3 octaves



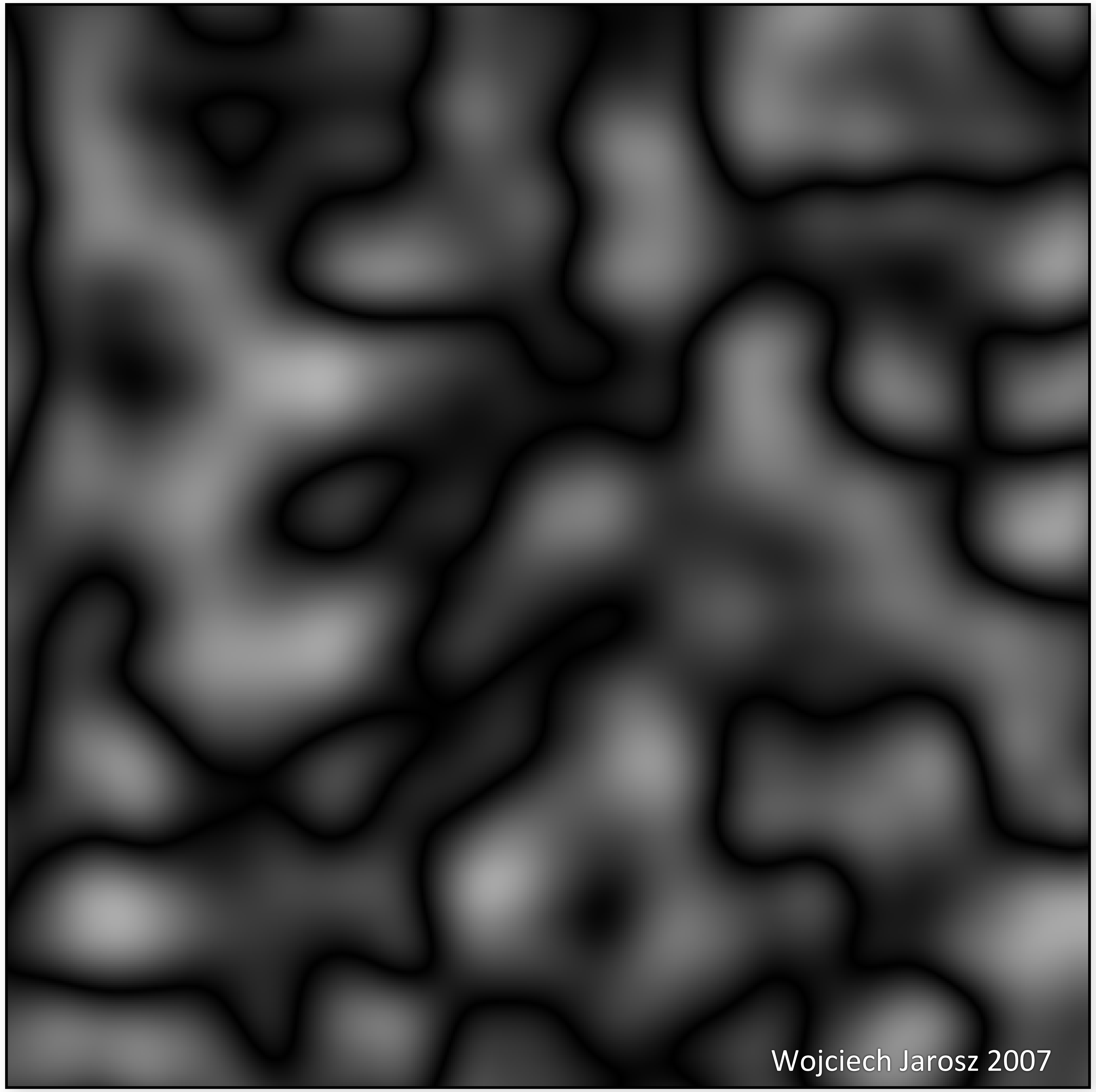
fBm - 4 octaves



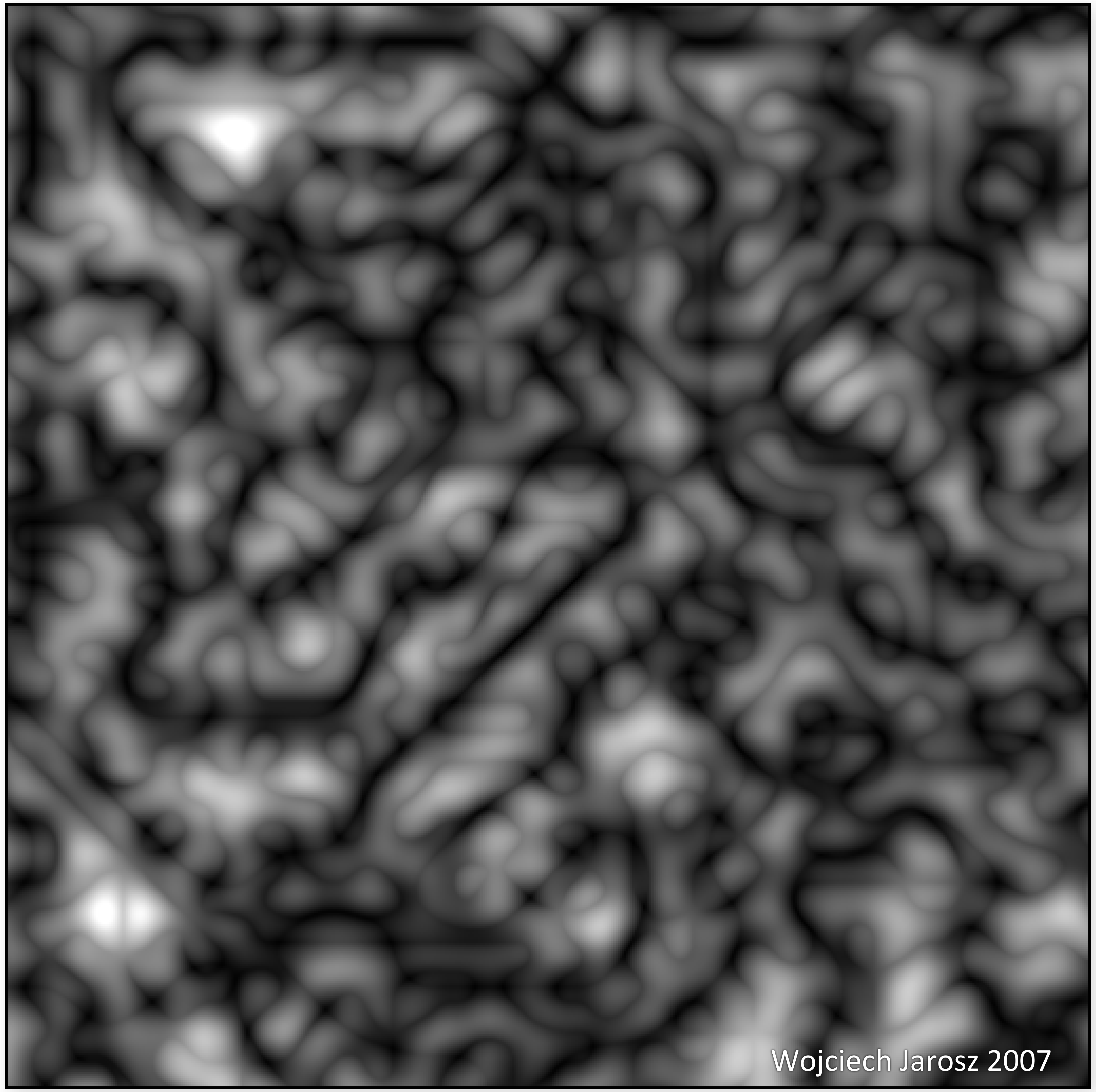
Turbulence

Same as fBm, but sum absolute value of noise function

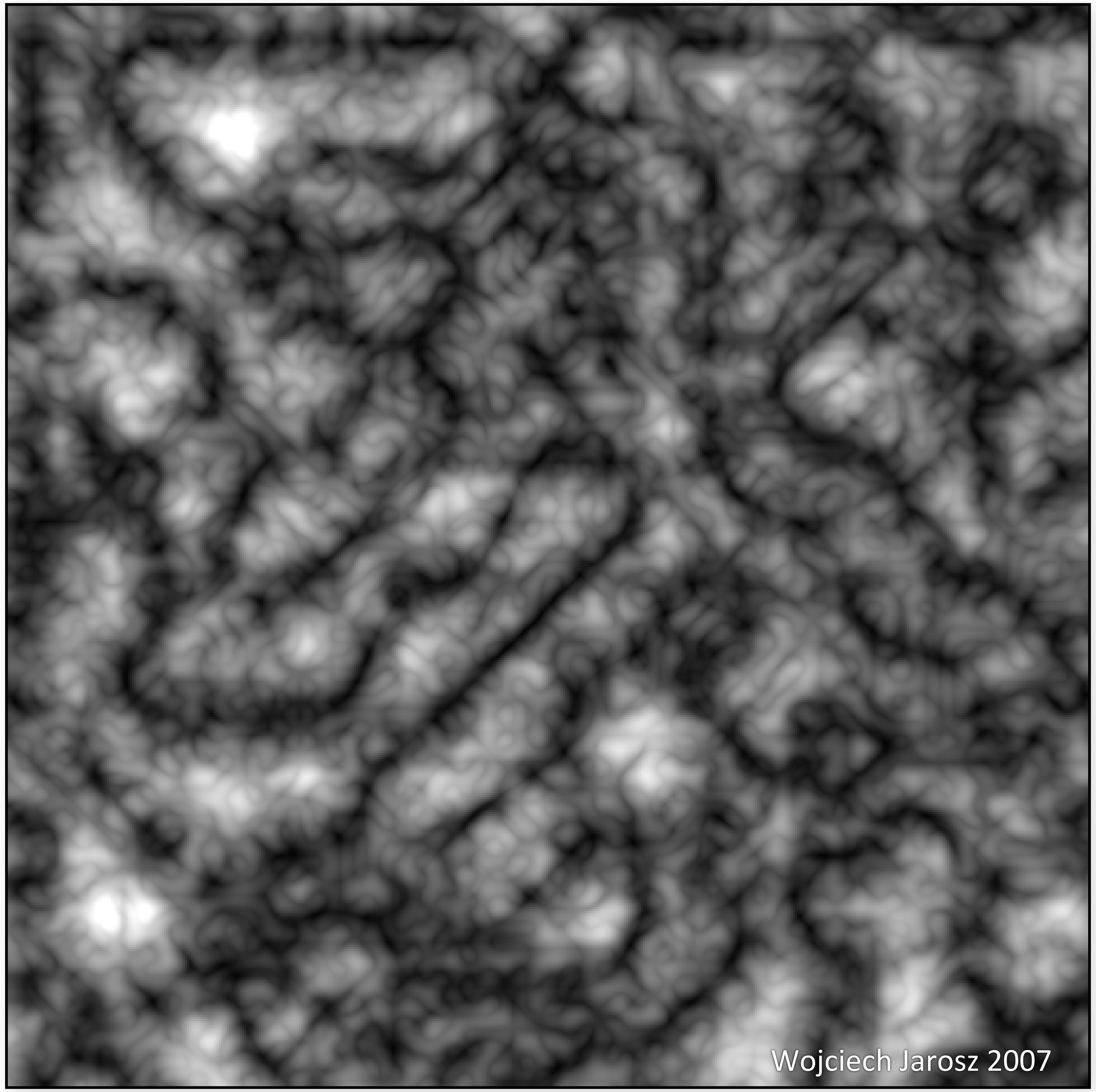
Turbulence - 1 octave



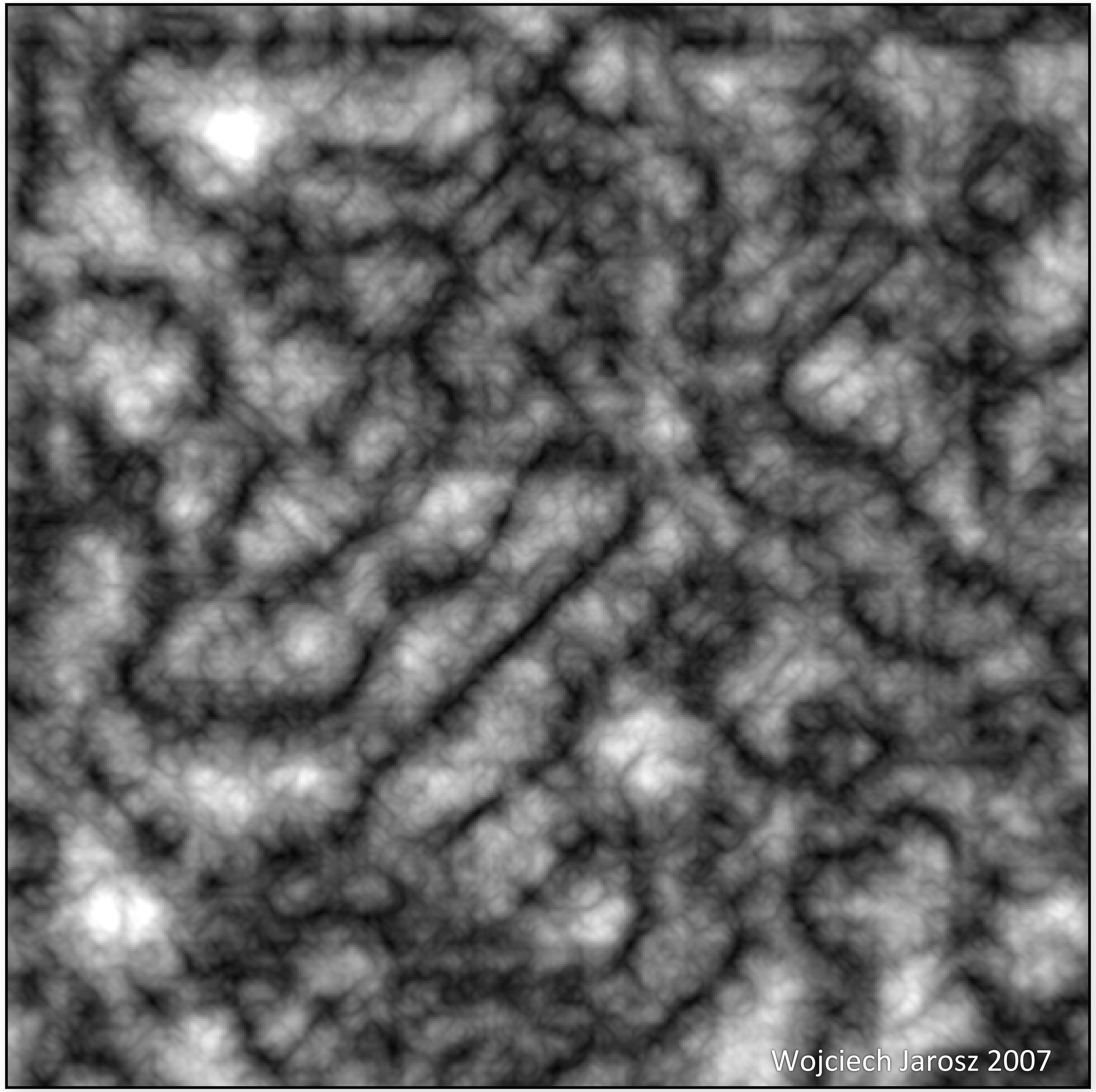
Turbulence - 2 octaves



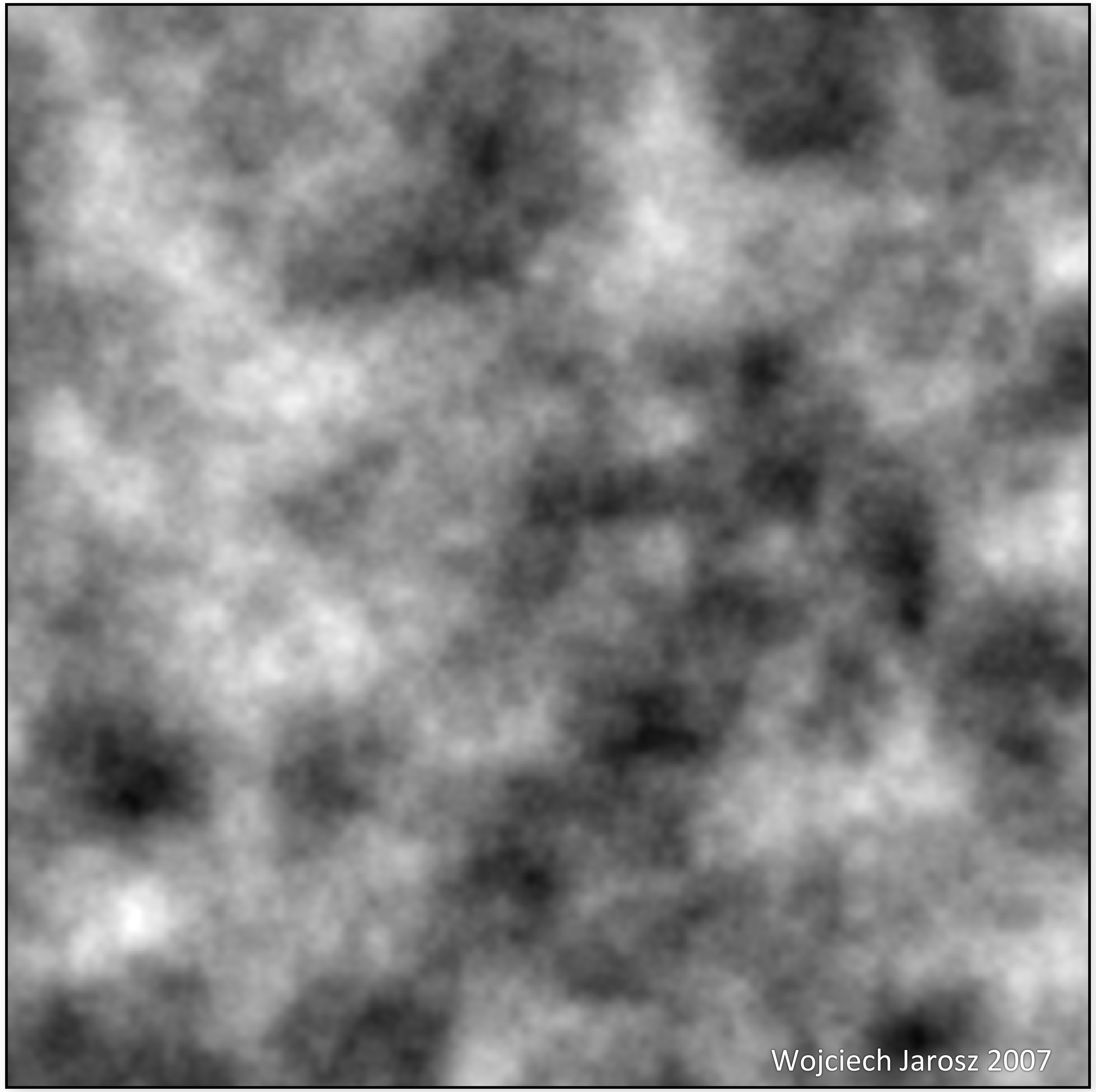
Turbulence - 3 octaves

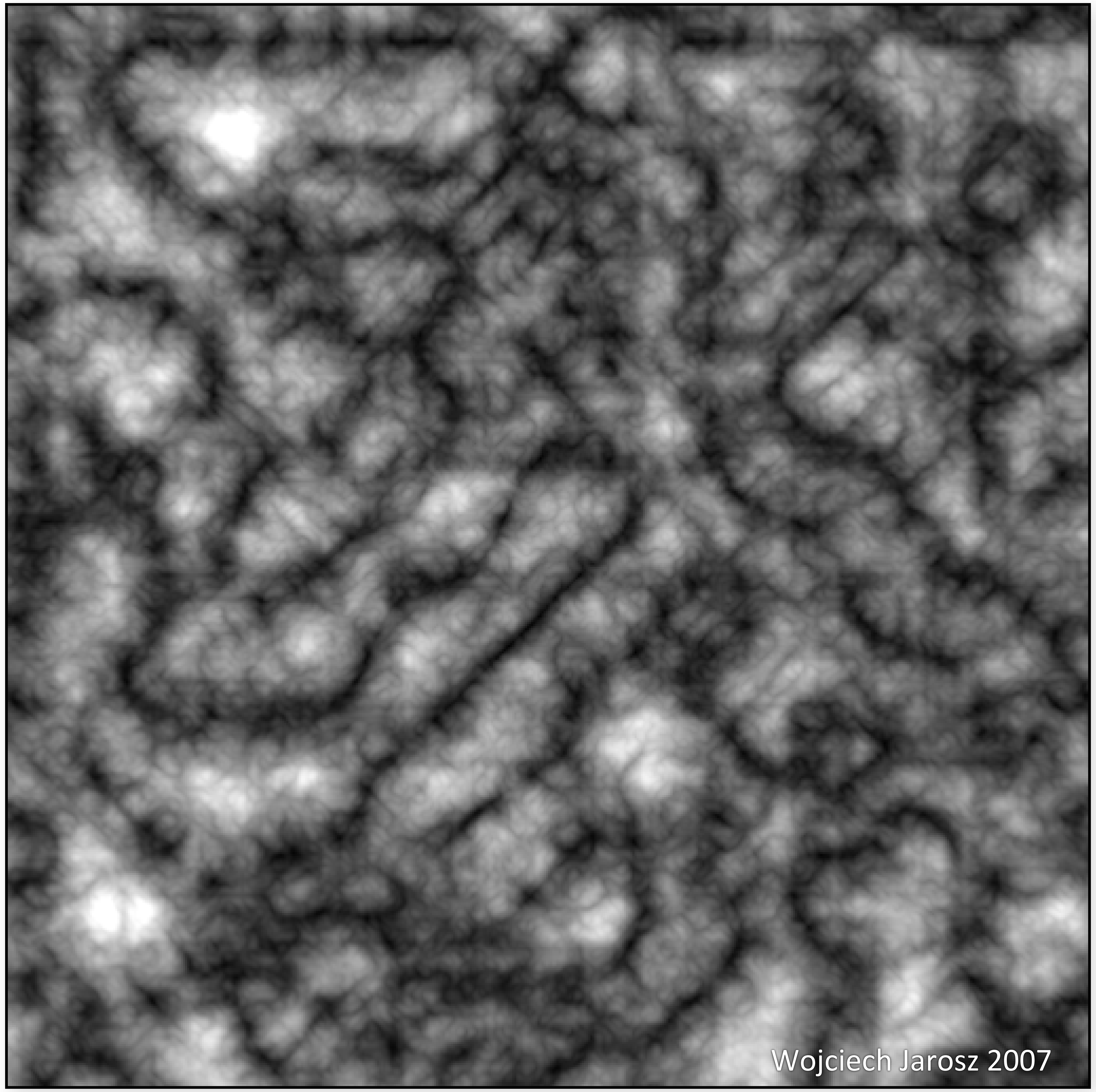


Turbulence - 4 octaves

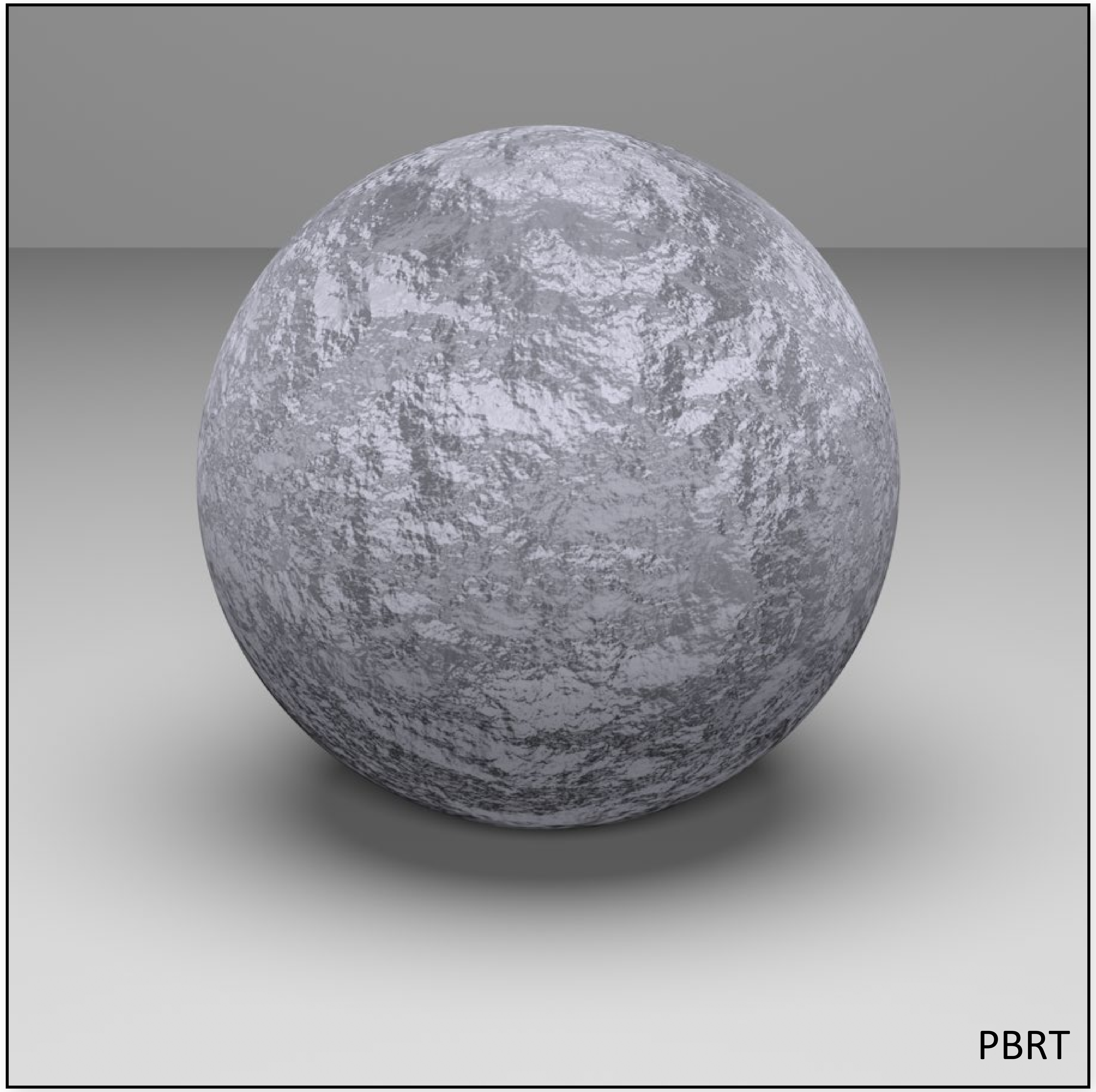


fBm vs Turbulence

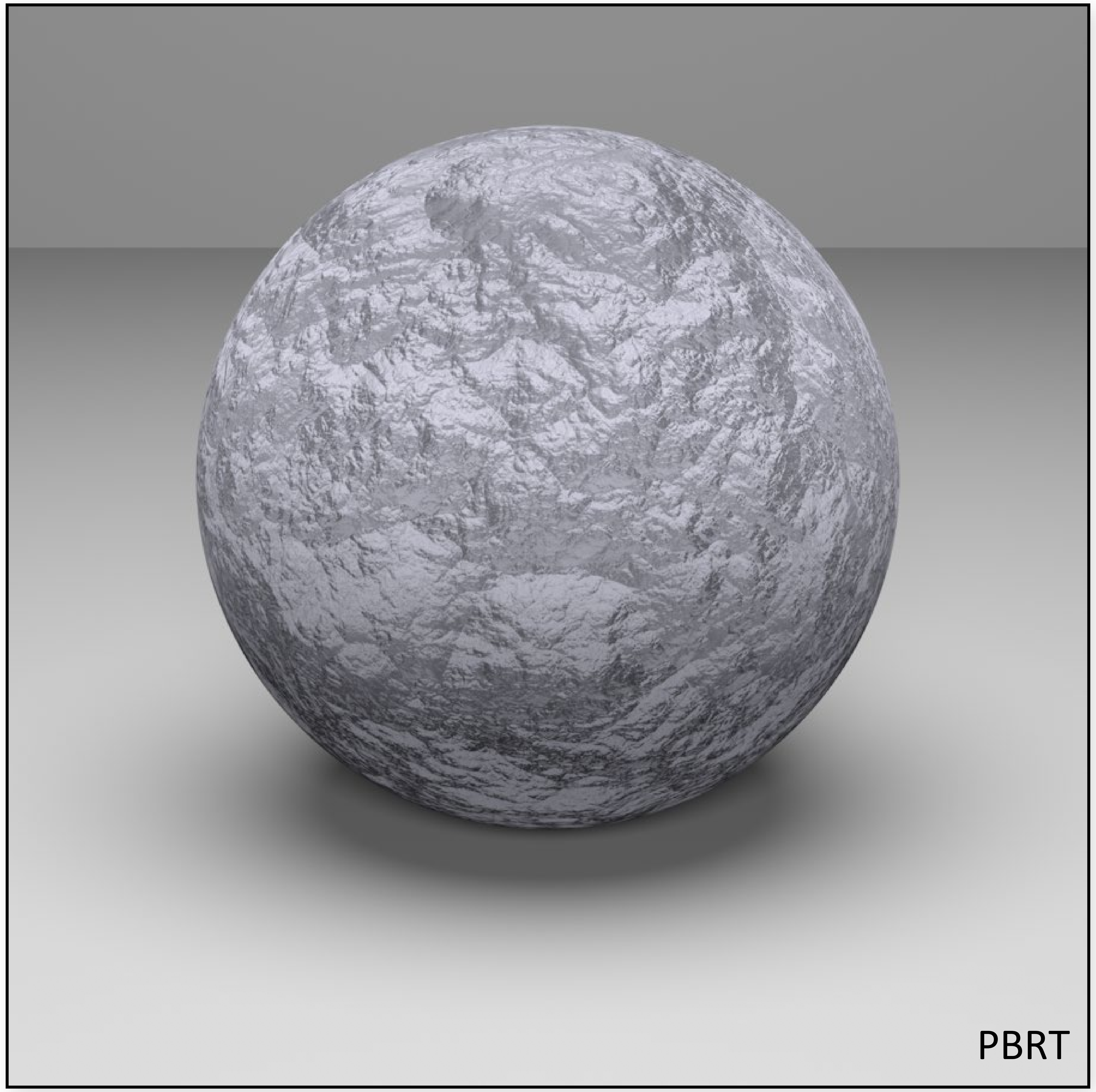




Bump mapping

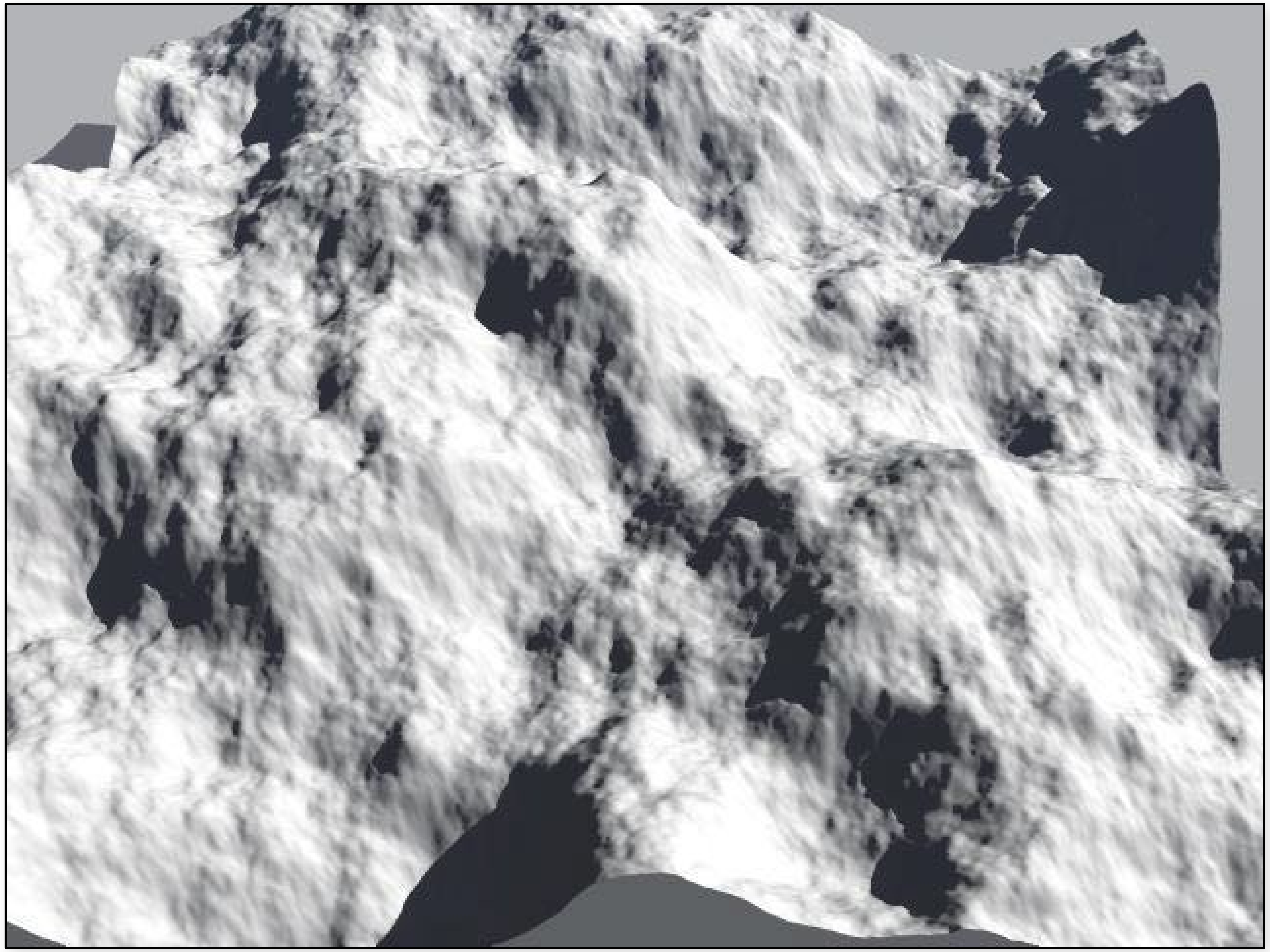


fBm



Turbulence

2D fBm



A fractional Brownian motion (fBm) terrain patch of fractal dimension ~2.1.

source: Ken Musgrave

Fractal dimension

Fractals have *fractional* dimension, e.g. D = 1.2.

- under some appropriate definition of dimension…

Integer component indicates the underlying Euclidean dimension of the fractal, in this case a line ("1" in 1.2).

Fractional part is called the fractal increment (".2" in 1.2).

Fractal increment varies from .0 to .999...

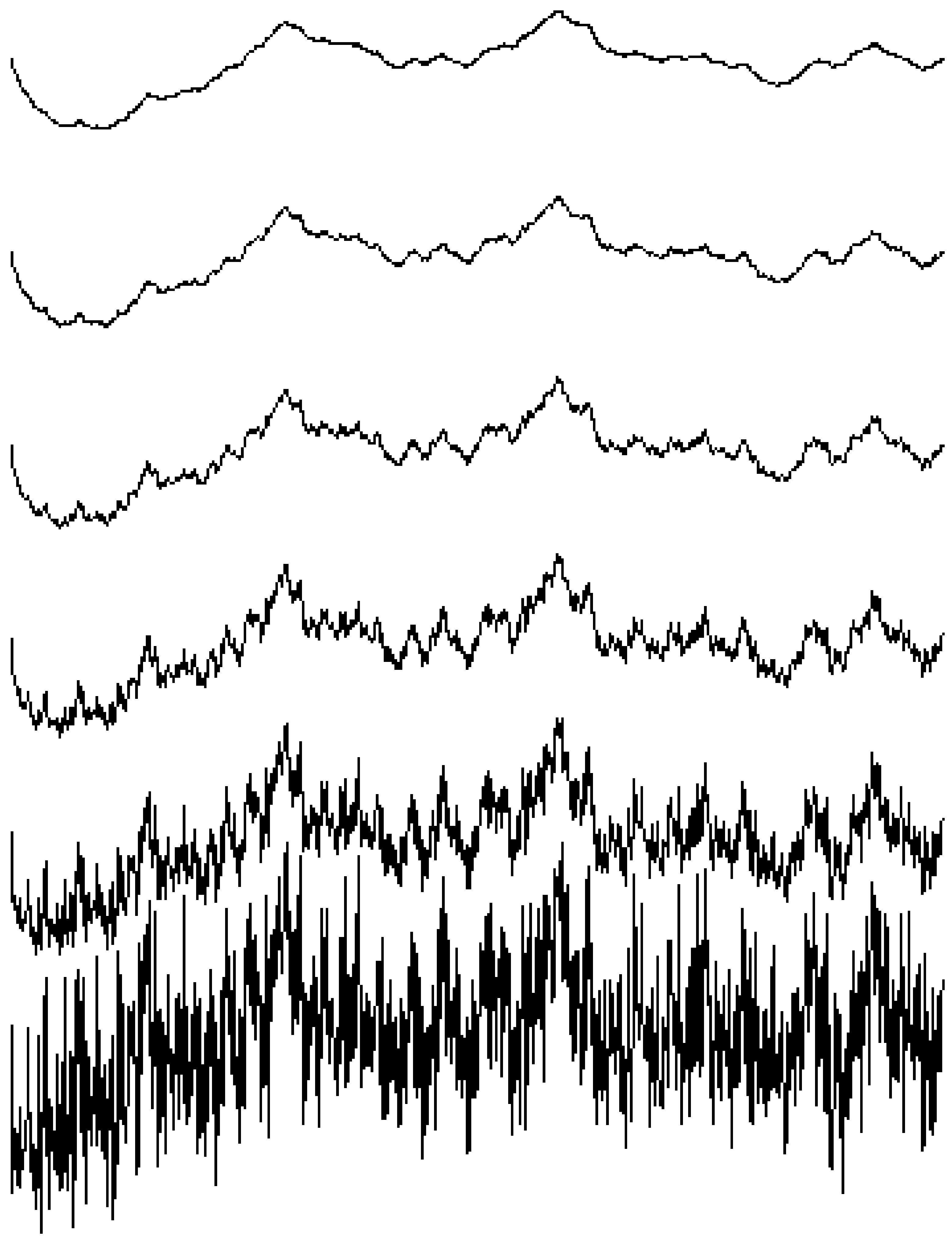
- fractal goes from (locally) occupying only its underlying Euclidean dimension (the line), to filling some part of the next higher dimension (the plane).

Continuous "slider" for the visual complexity of a fractal

- “smoother” ⟺ “rougher”

What determines the dimension of fBm?

Fractal dimension of fBm



Traces of fBm for H varying from 1.0 to 0.0 in increments of 0.2

source: Ken Musgrave

fBm

fBm is statistically homogeneous and isotropic.

- Homogeneous means "the same everywhere"
- Isotropic means "the same in all directions"

Fractal phenomena in nature are rarely so simple and well-behaved.

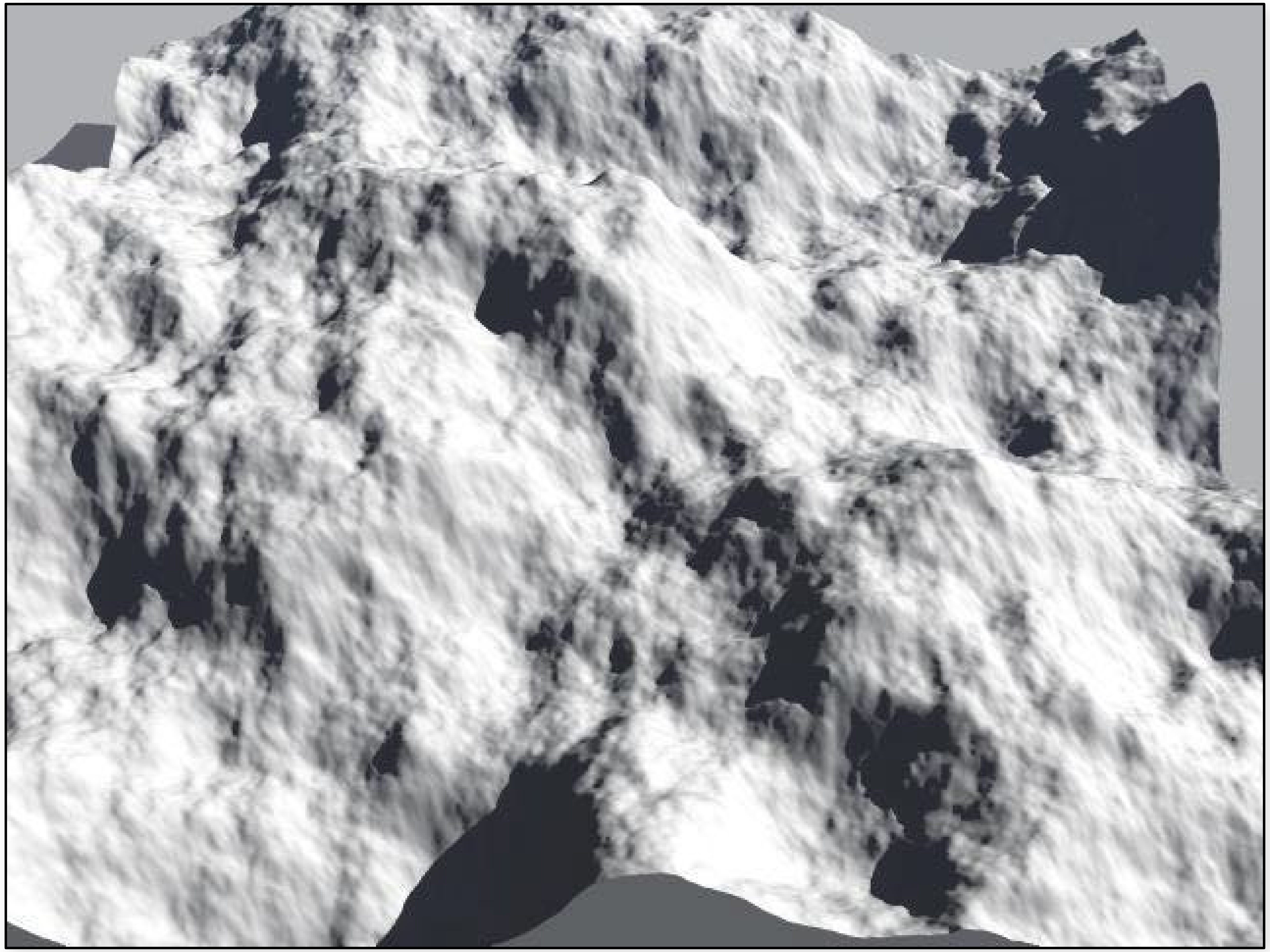
Multifractals

Fractal system which has a different fractal dimension in different regions

Heterogeneous fBm

- Scale higher frequencies in the summation by the value of the previous frequency.
- Many possibilities: hetero terrain, hybrid multifractal, ridged multifractal

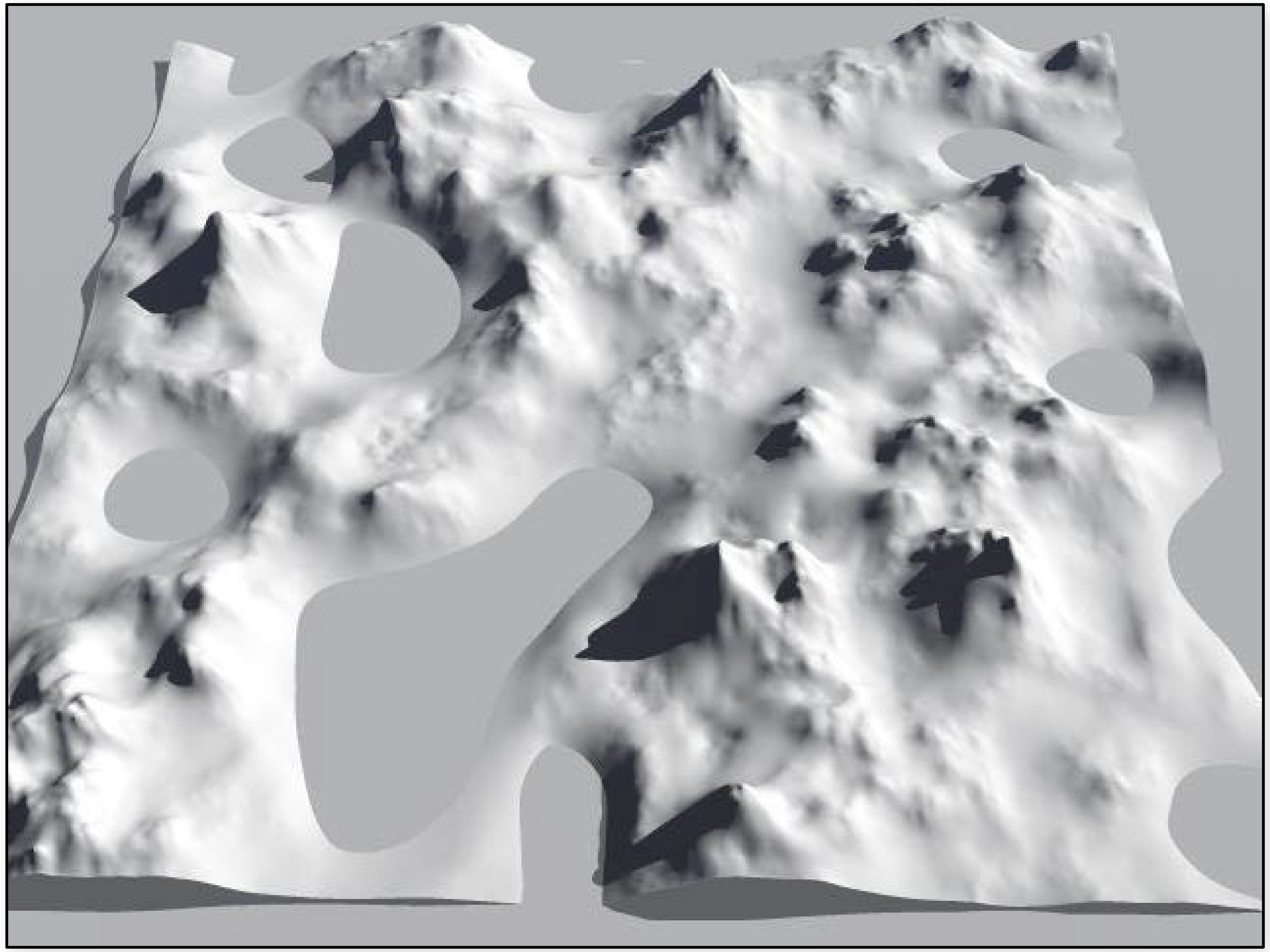
2D fBm



A fractional Brownian motion (fBm) terrain patch of fractal dimension ~2.1.

source: Ken Musgrave

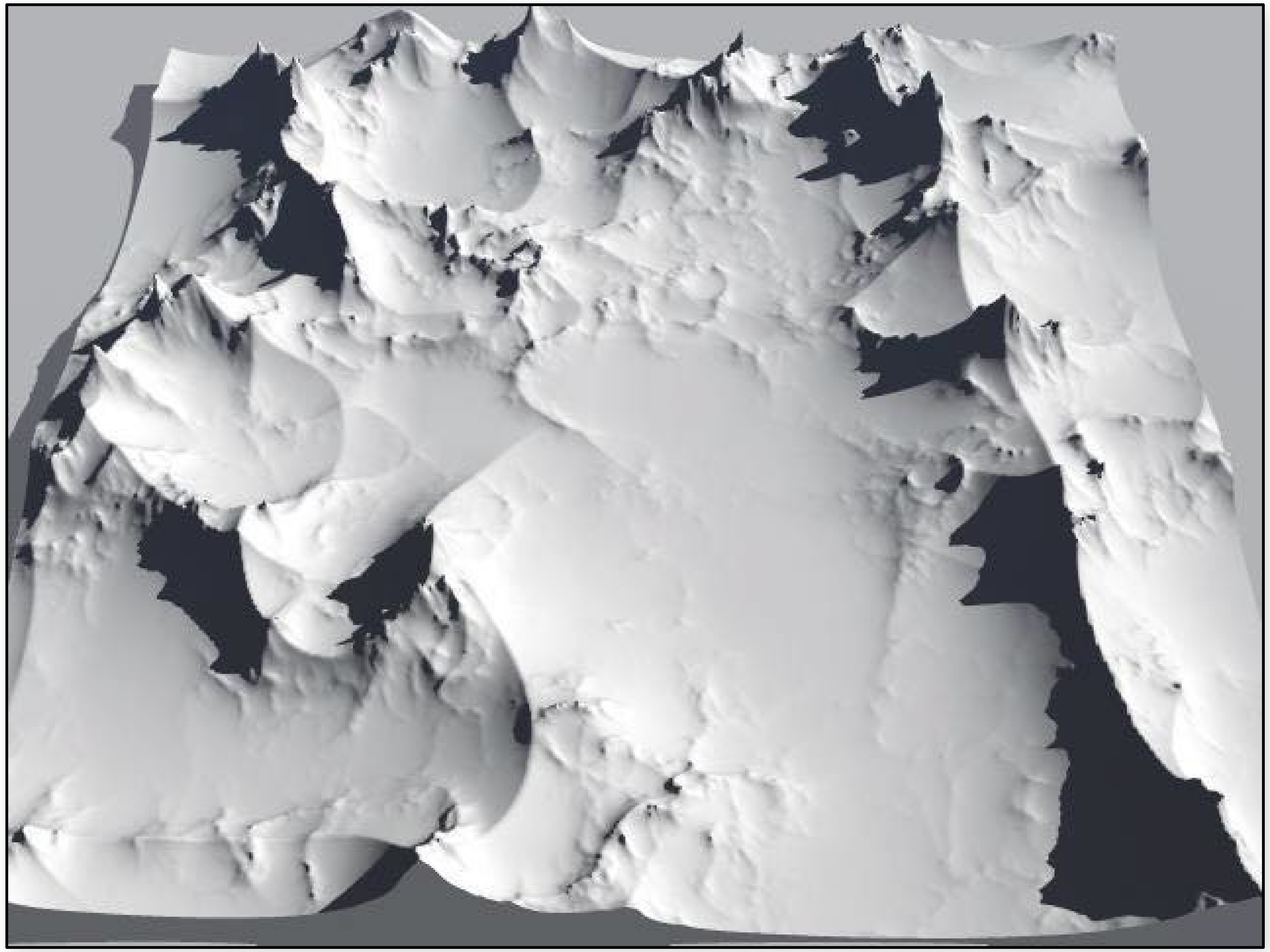
Heterogeneous fBm



A hybrid multifractal terrain patch made with a Perlin noise basis: the "alpine hills" Bryce 4 terrain model.

source: Ken Musgrave

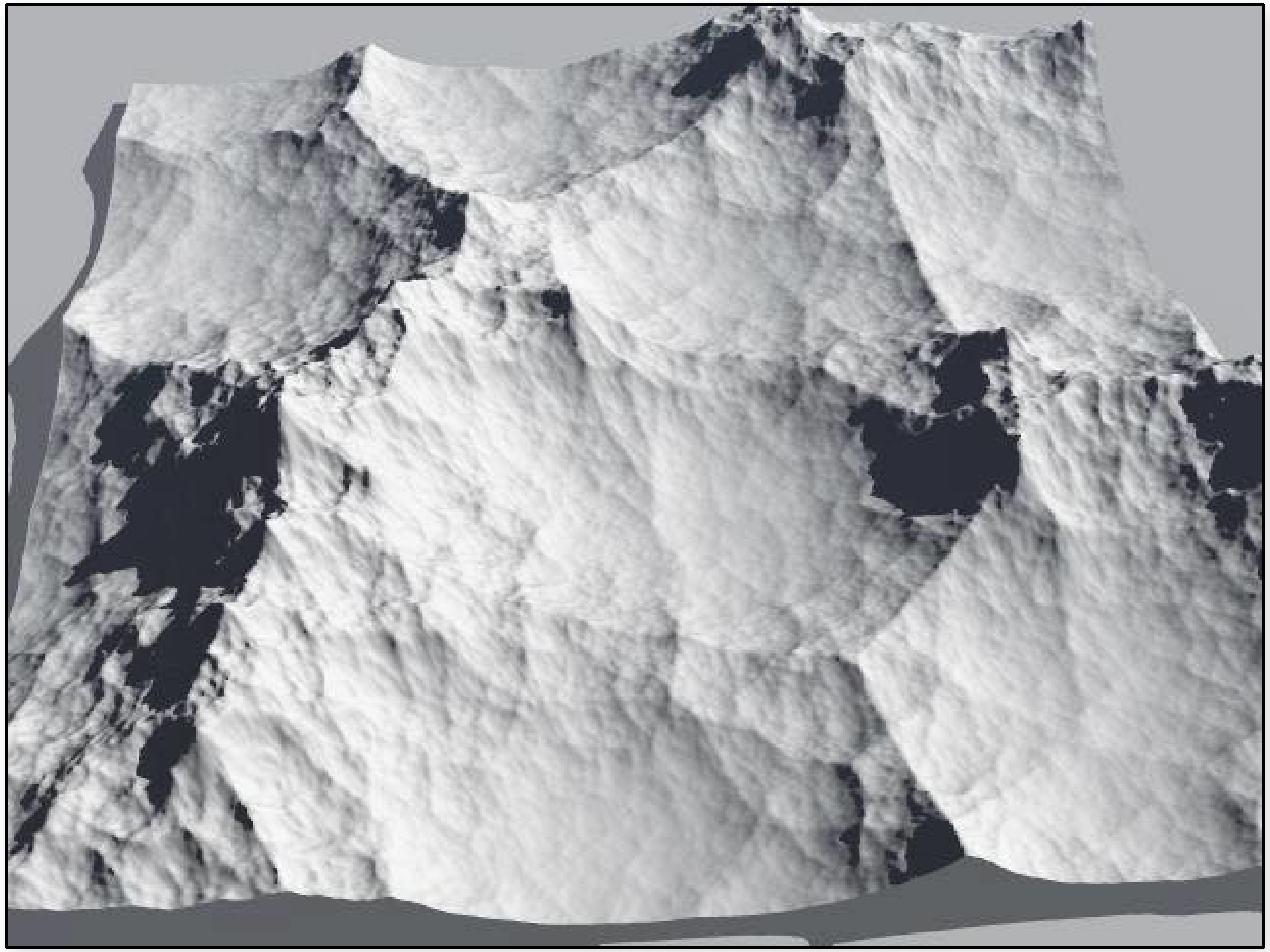
Heterogeneous fBm



The “ridges” terrain model from Bryce 4: a hybrid multifractal made from one minus the absolute value of Perlin noise.

source: Ken Musgrave

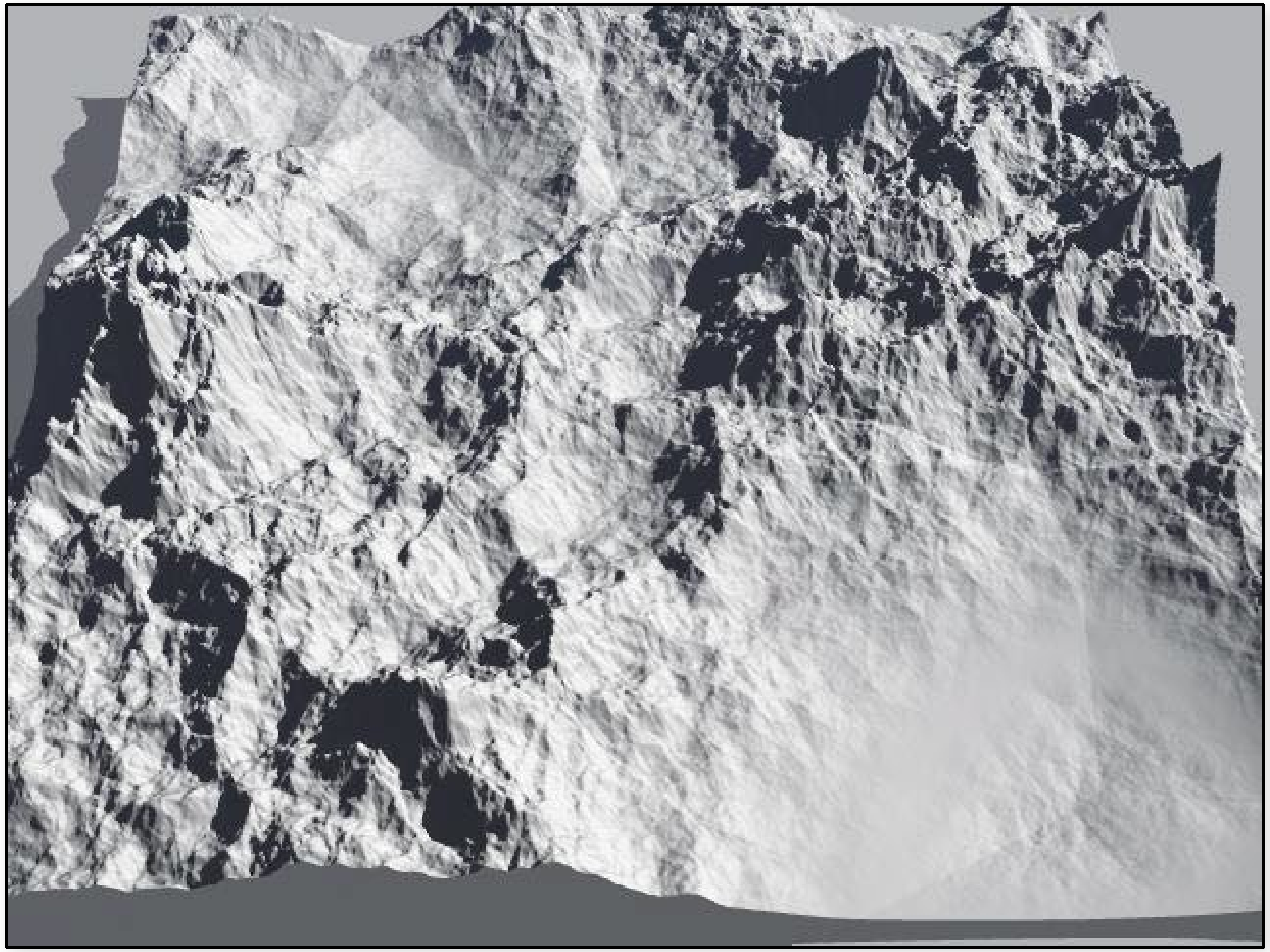
Heterogeneous fBm



A hybrid multifractal made from Worley's Voronoi distance-squared basis

source: Ken Musgrave

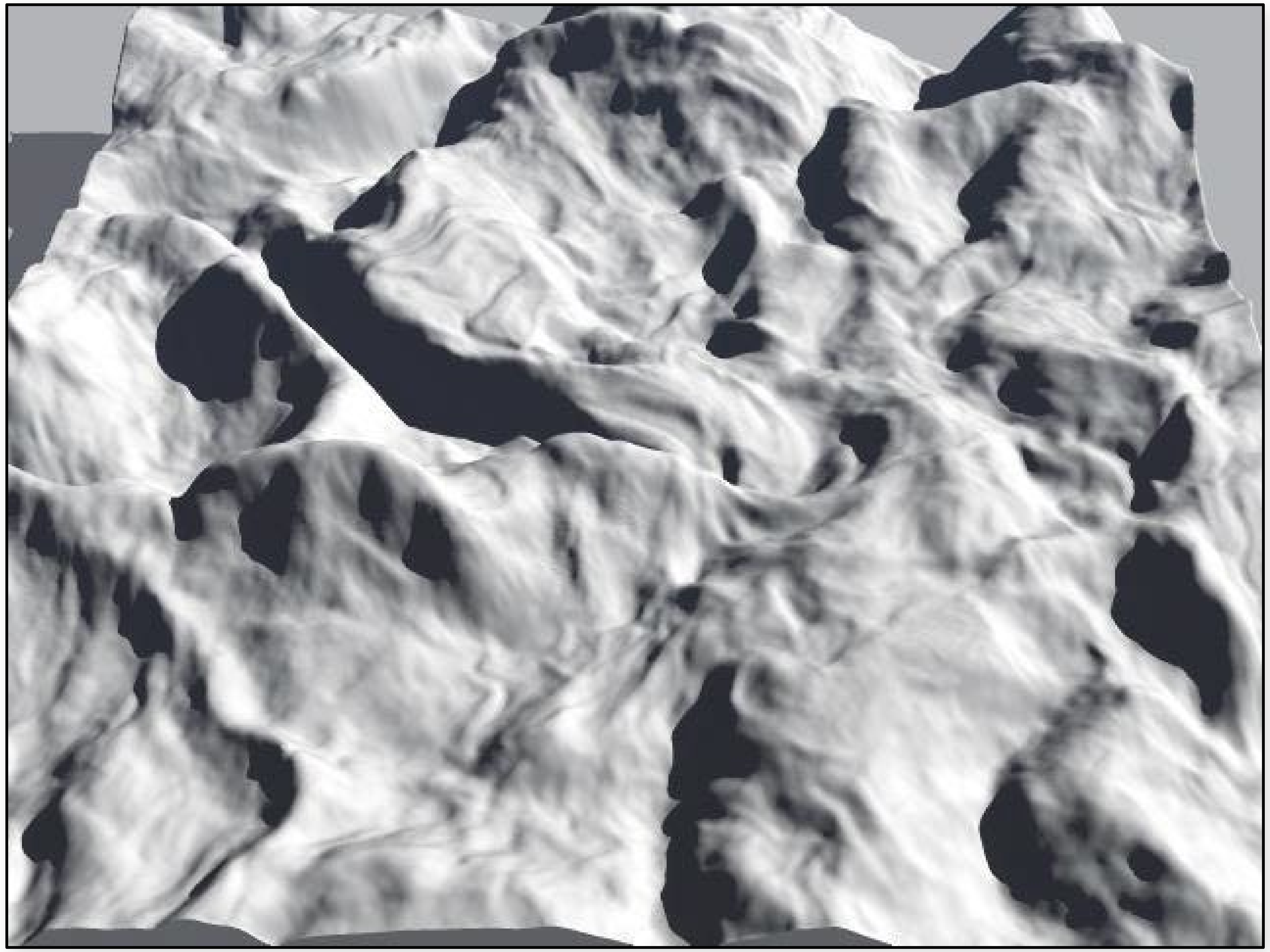
Heterogeneous fBm



A hybrid multifractal made from Worley's Voronoi distance basis

source: Ken Musgrave

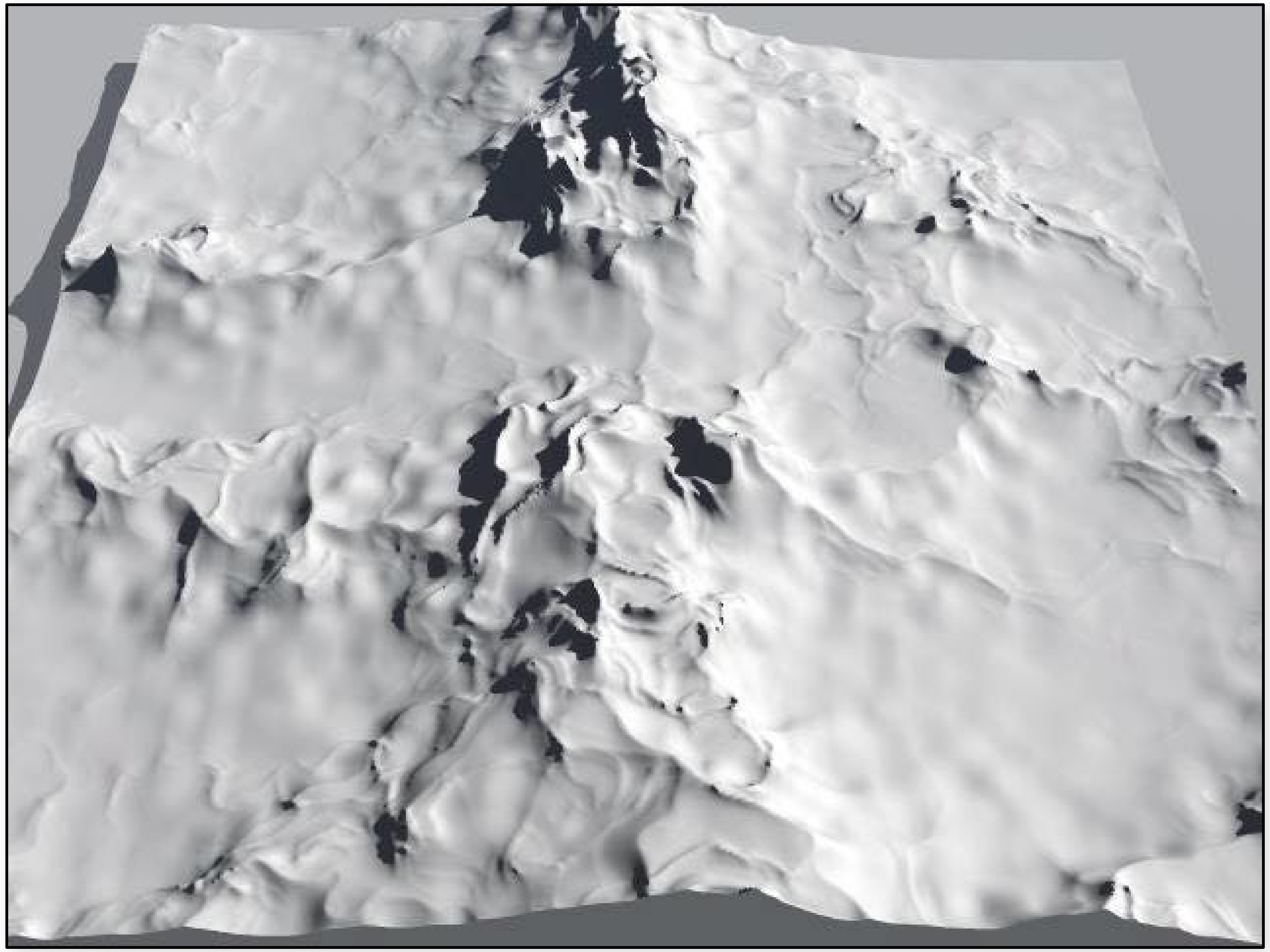
Domain Distortion



fBm distorted with fBm

source: Ken Musgrave

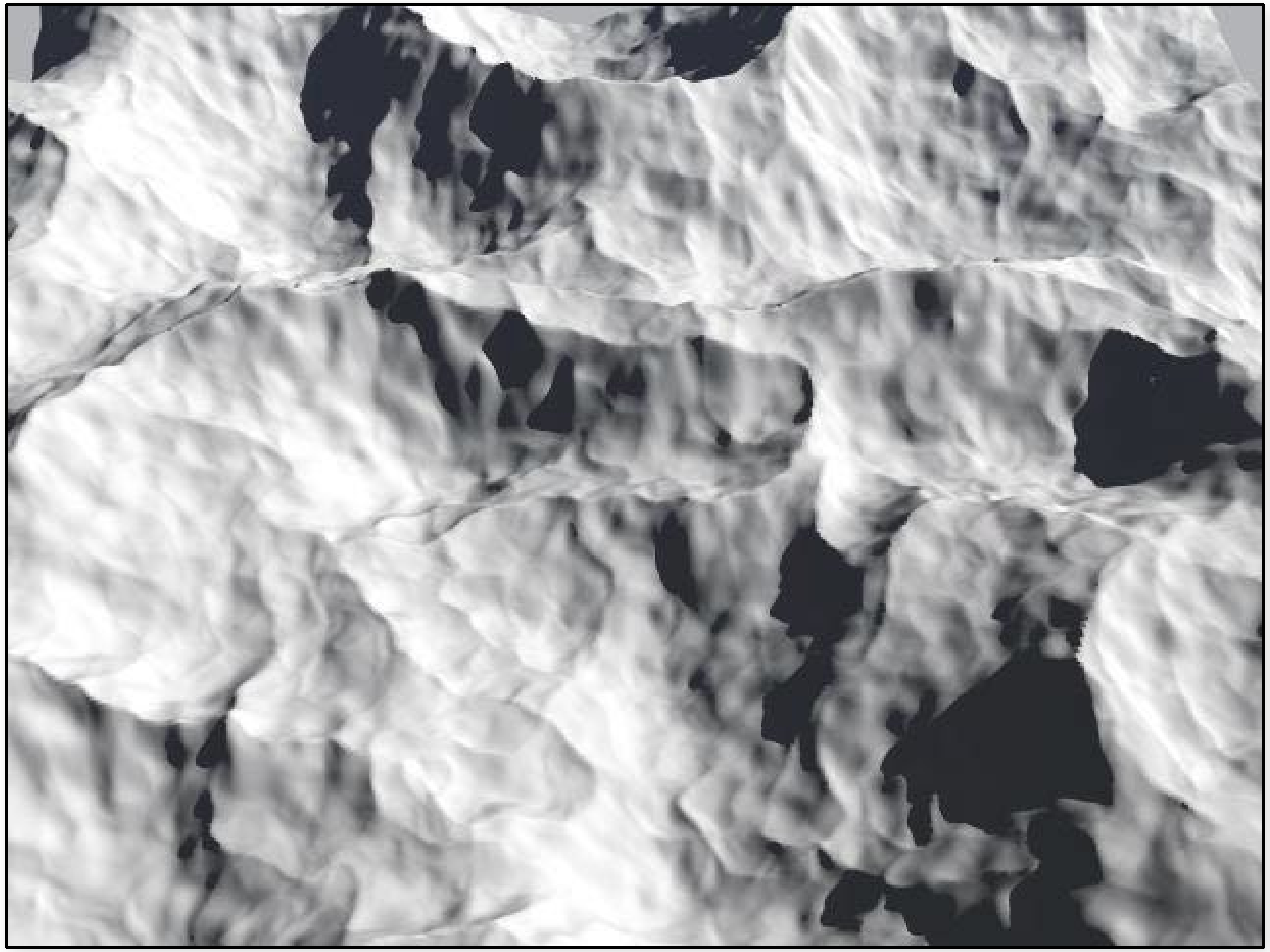
Domain Distortion



A sample of the “warped ridges” terrain model in Bryce 4: the “ridges” model distorted with fBm.

source: Ken Musgrave

Domain Distortion

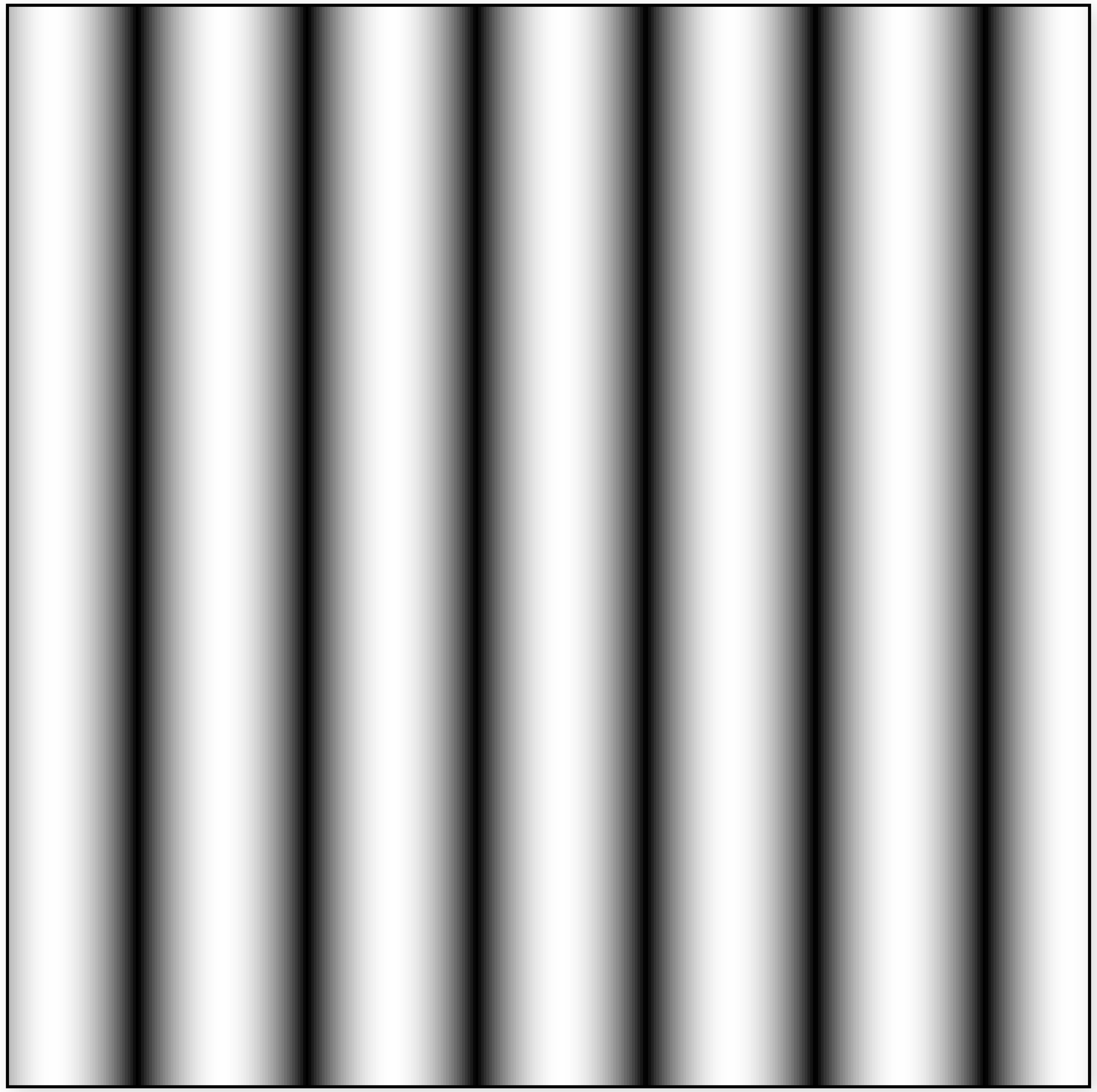


A sample of the “warped slickrock” terrain model in Bryce 4: fBm constructed from one minus the absolute value of Perlin noise, distorted with fBm.

source: Ken Musgrave

Recall: 3D stripe texture

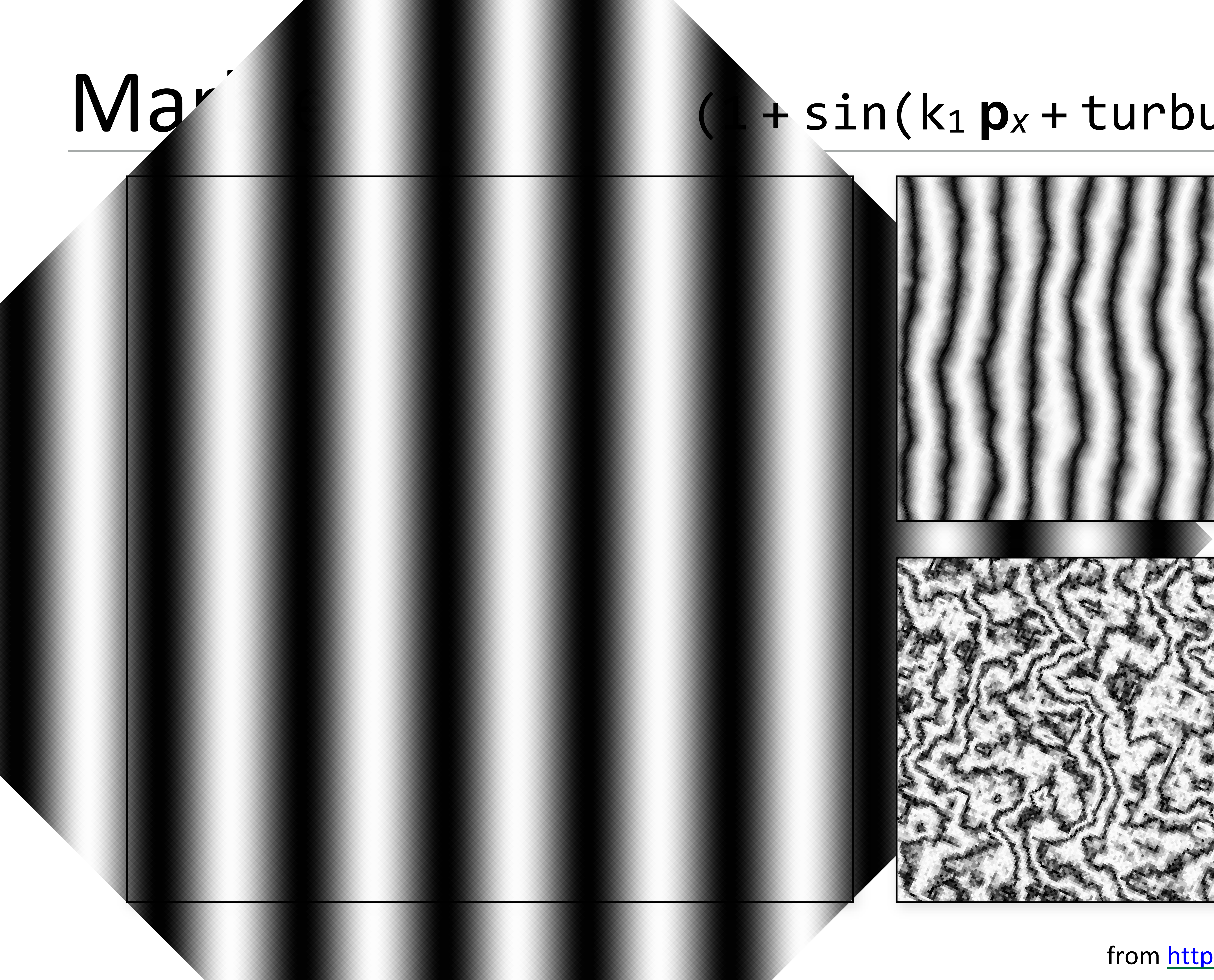
```
color stripe(point p, real w):
    t = (1 + sin(π p_x/w))/2
    return lerp(c_0, c_1, t)
```

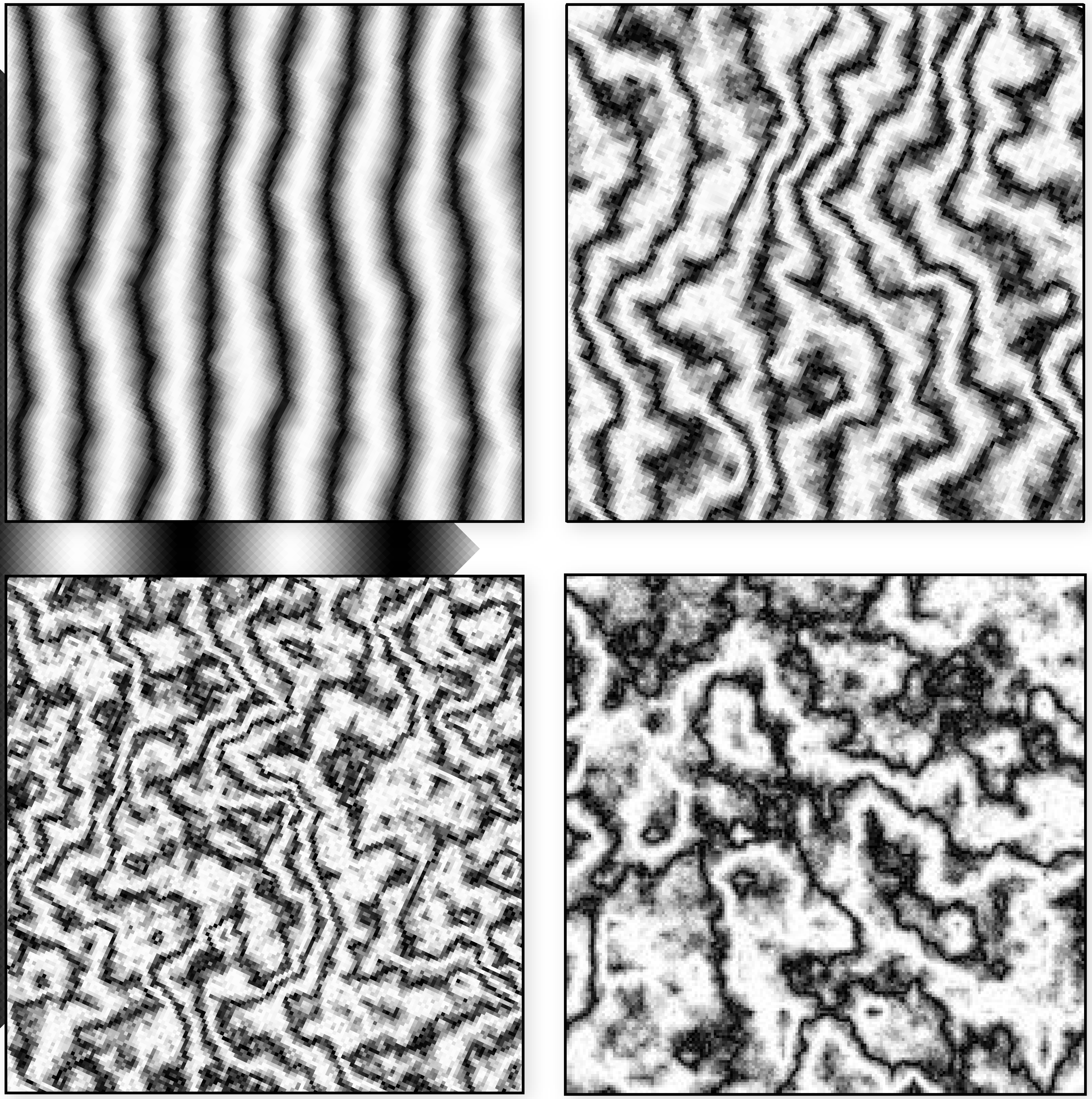


How can we make this less structured (less "boring")?

Mar

$(1 + \sin(k_1\,\mathbf{p}_x + \mathrm{turbulence}(k_2\,\mathbf{p}))/w)/2$

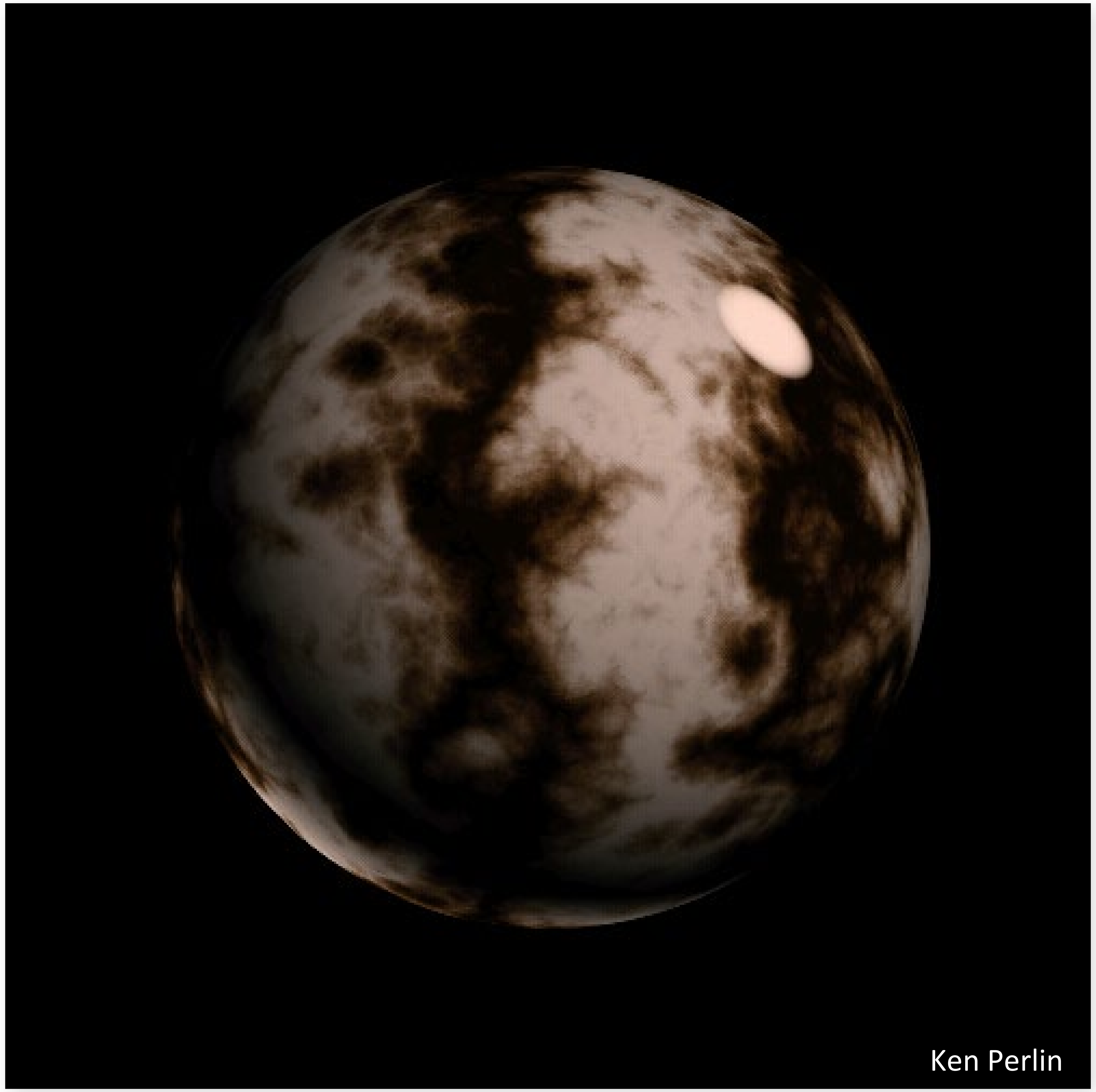




from http://lodev.org/cgtutor/randomnoise.html

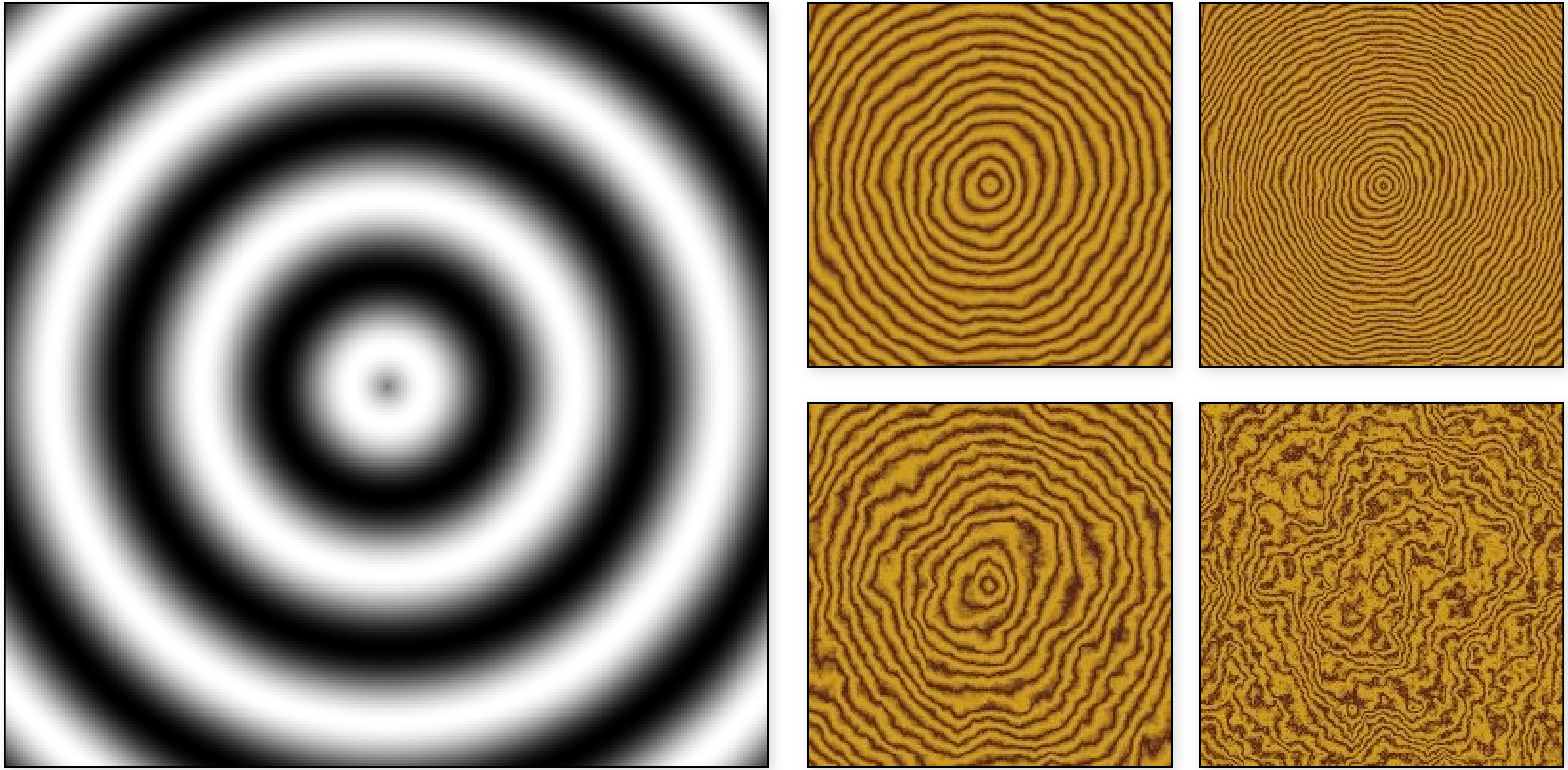
Marble

$(1 + \sin(k_1\, \mathbf{p}_x + \mathrm{turbulence}(k_2\, \mathbf{p}))/w)/2$



Wood

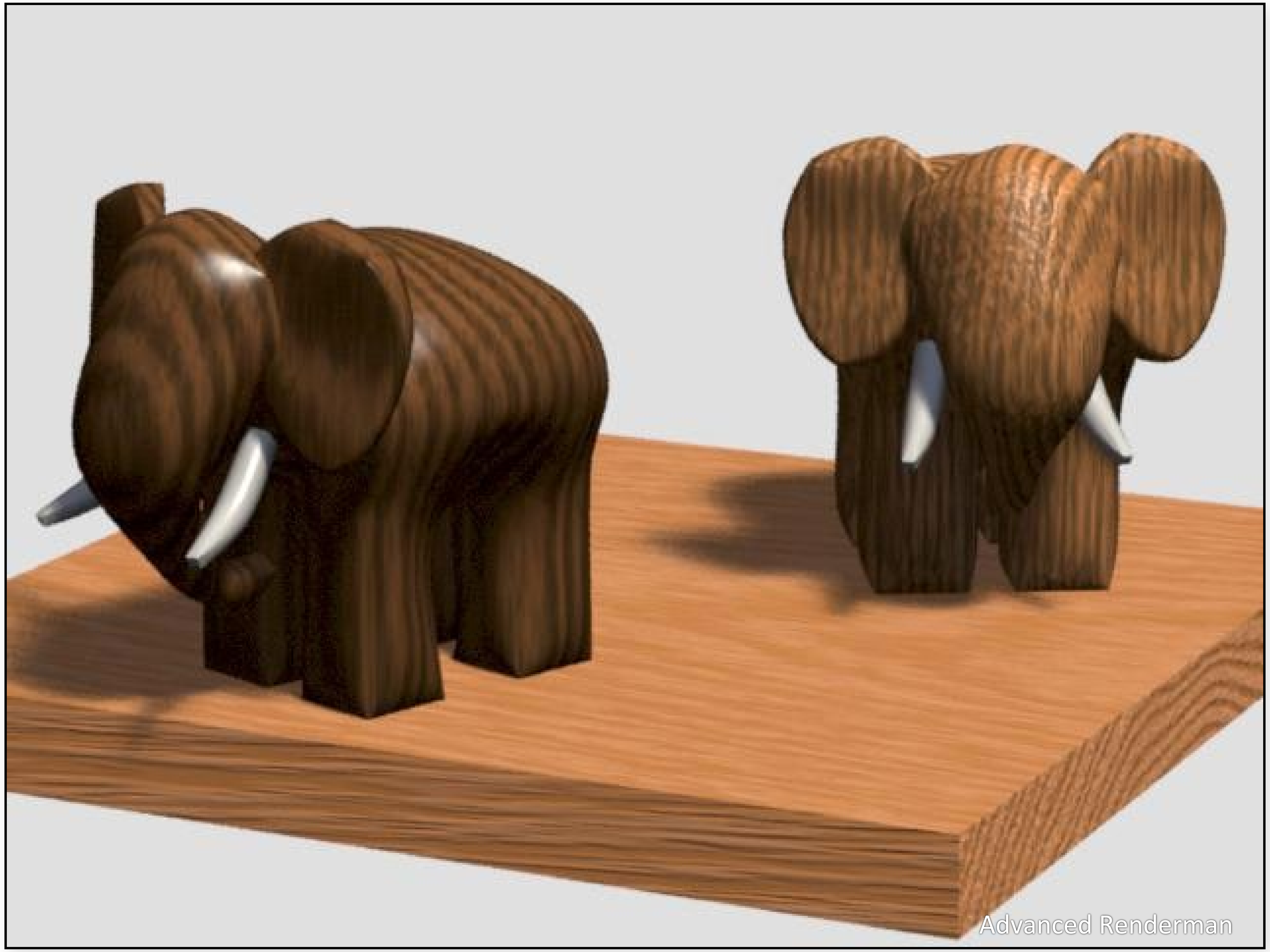
$$(1 + \sin(\text{sqrt}(\mathbf{p}_x^2 + \mathbf{p}_y^2) + \text{fBm}(\mathbf{p})))/2$$



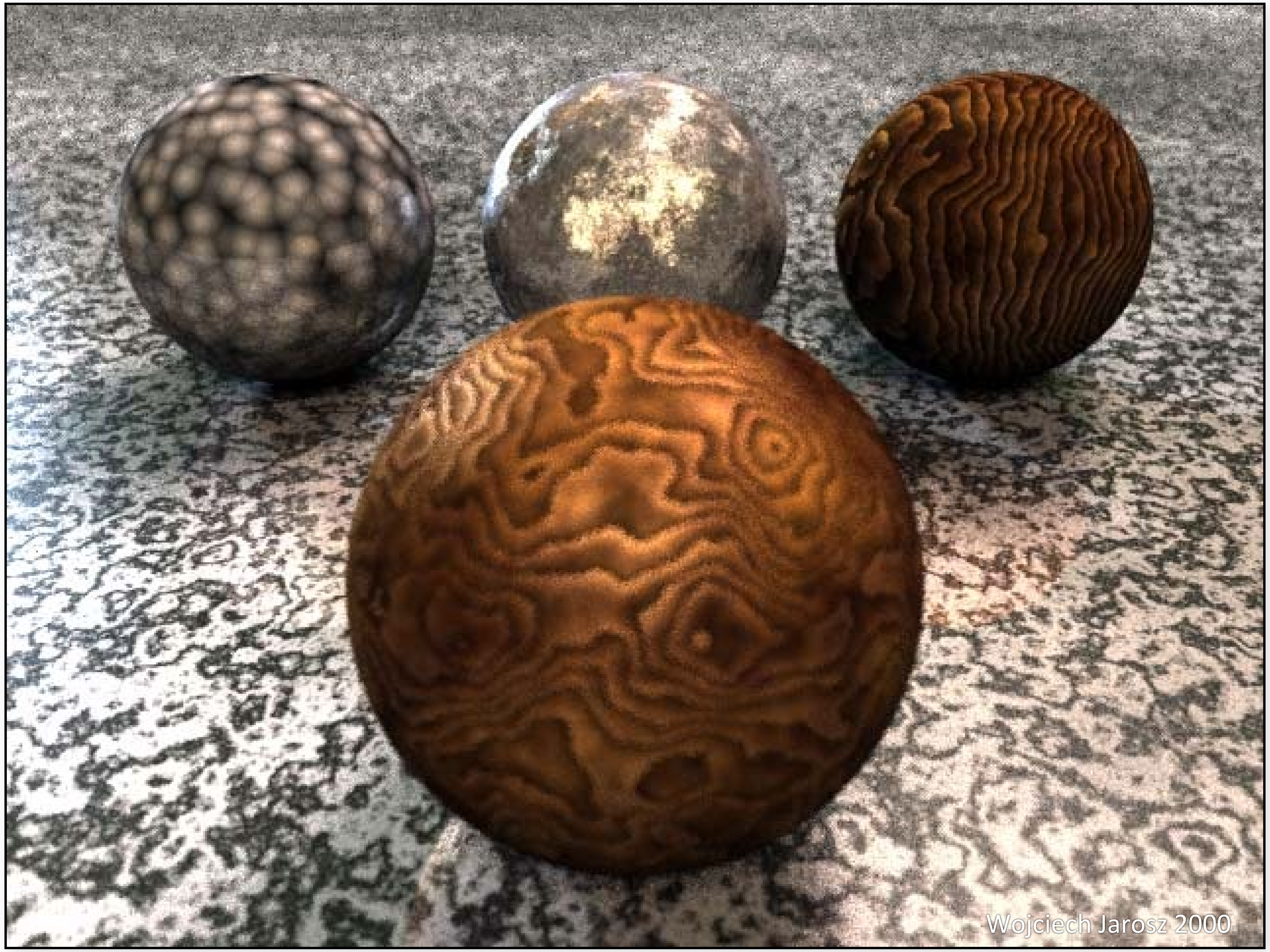
from http://lodev.org/cgtutor/randomnoise.html

Wood

$$(1 + \sin(\mathrm{sqrt}(\mathbf{p}_x^2 + \mathbf{p}_y^2) + \mathrm{fBm}(\mathbf{p})))/2$$



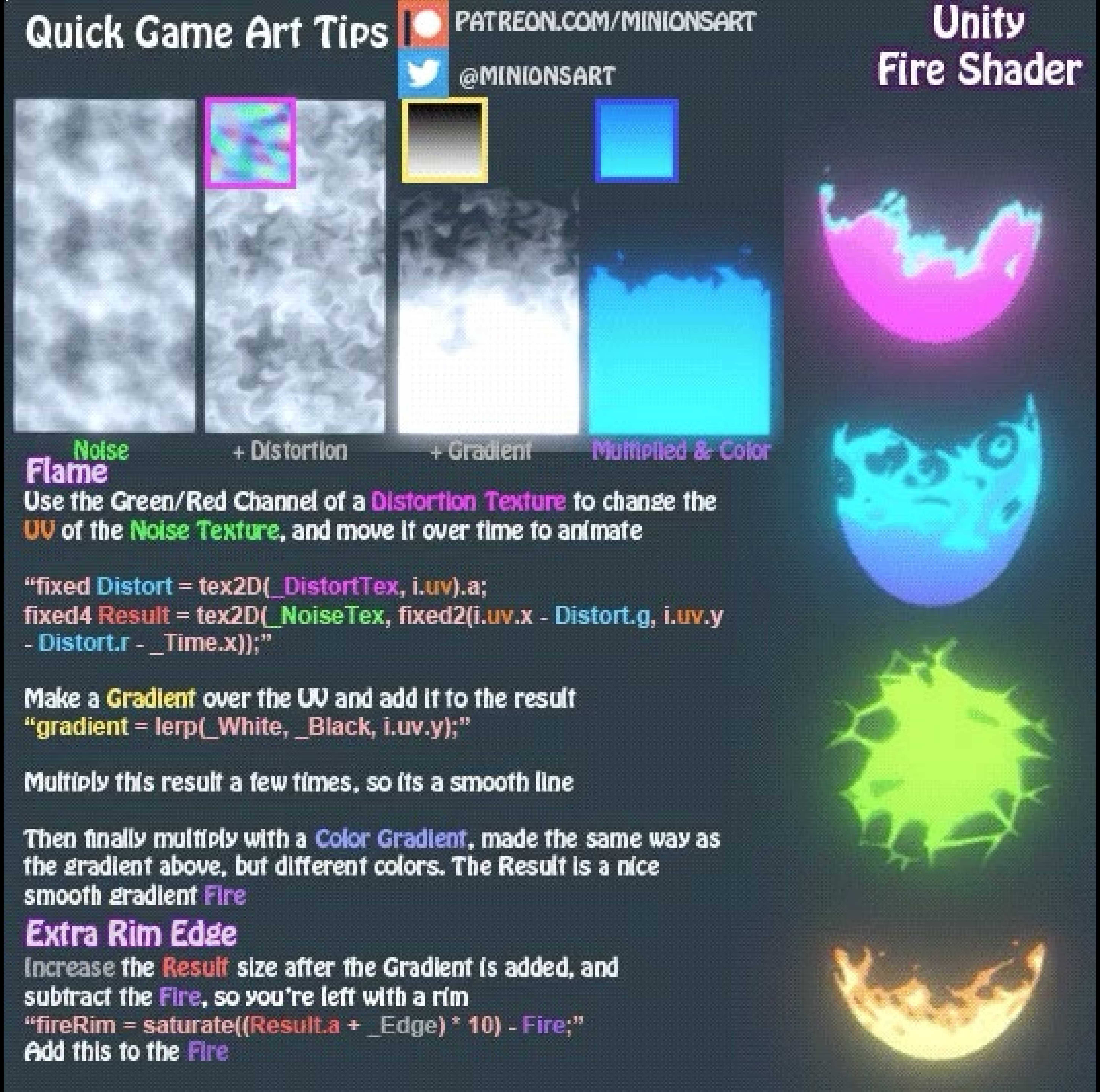
and more...



and more...


SPACE RANGER
LIGHTYEAR
LASER

Source: twitter/ciaccodavide



Source: twitter/minionsart

Worley noise

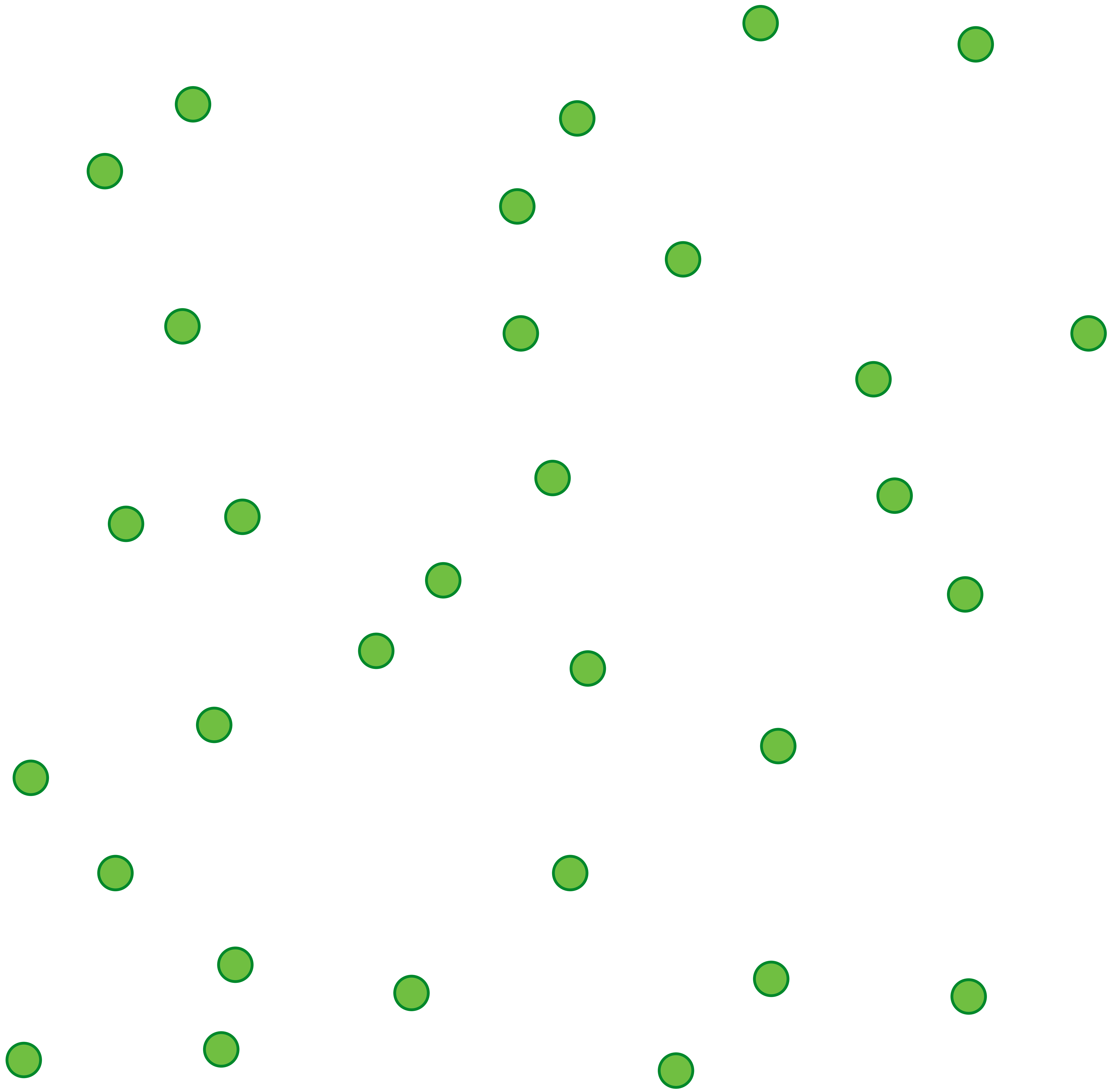
"Cellular texture" function

- Introduced in 1996 by Steve Worley
- Different from *cell* texture!

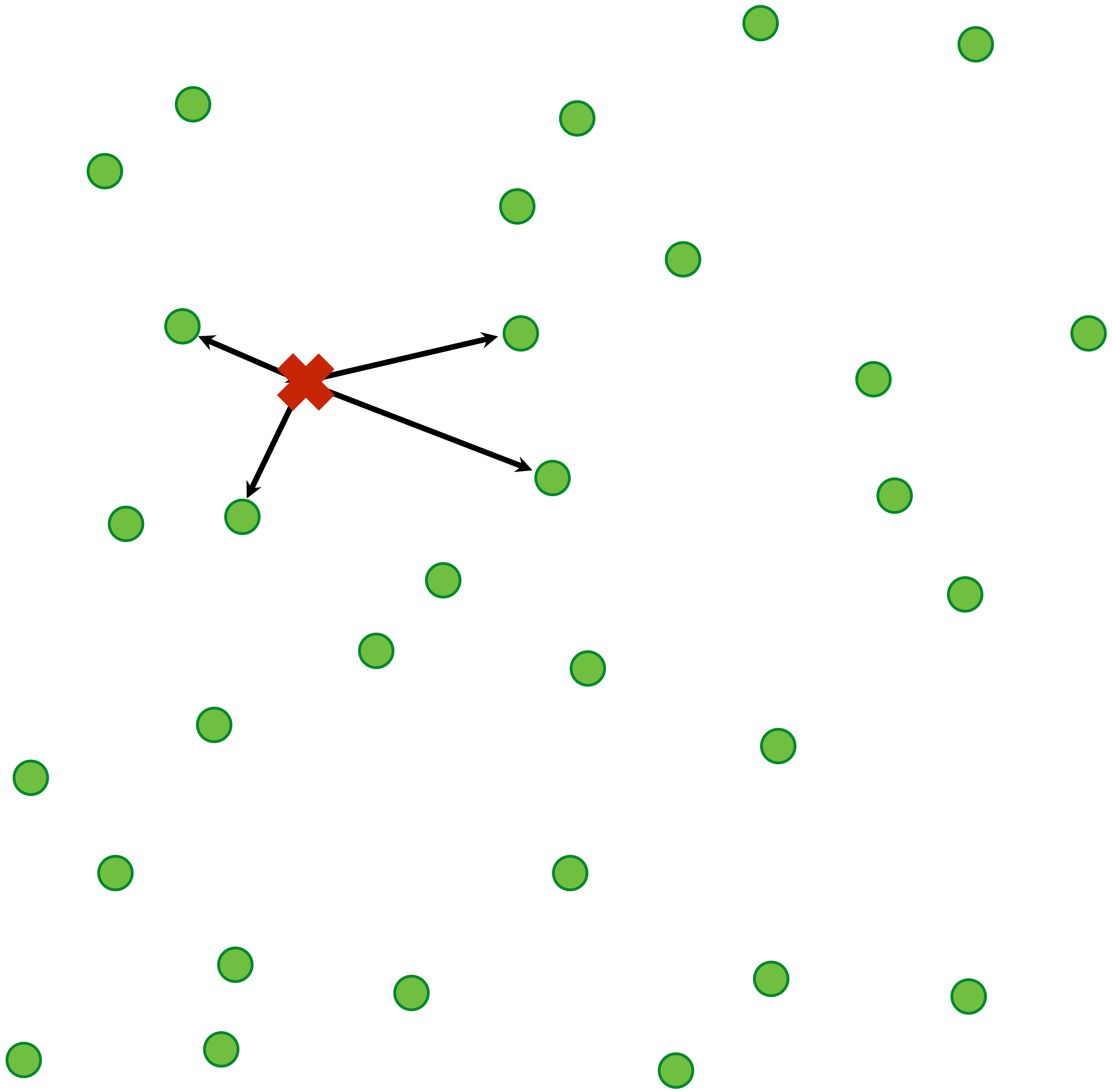
Randomly distribute "feature points" in space

- $f_n(x)$ = distance to n^{th} closest point to x

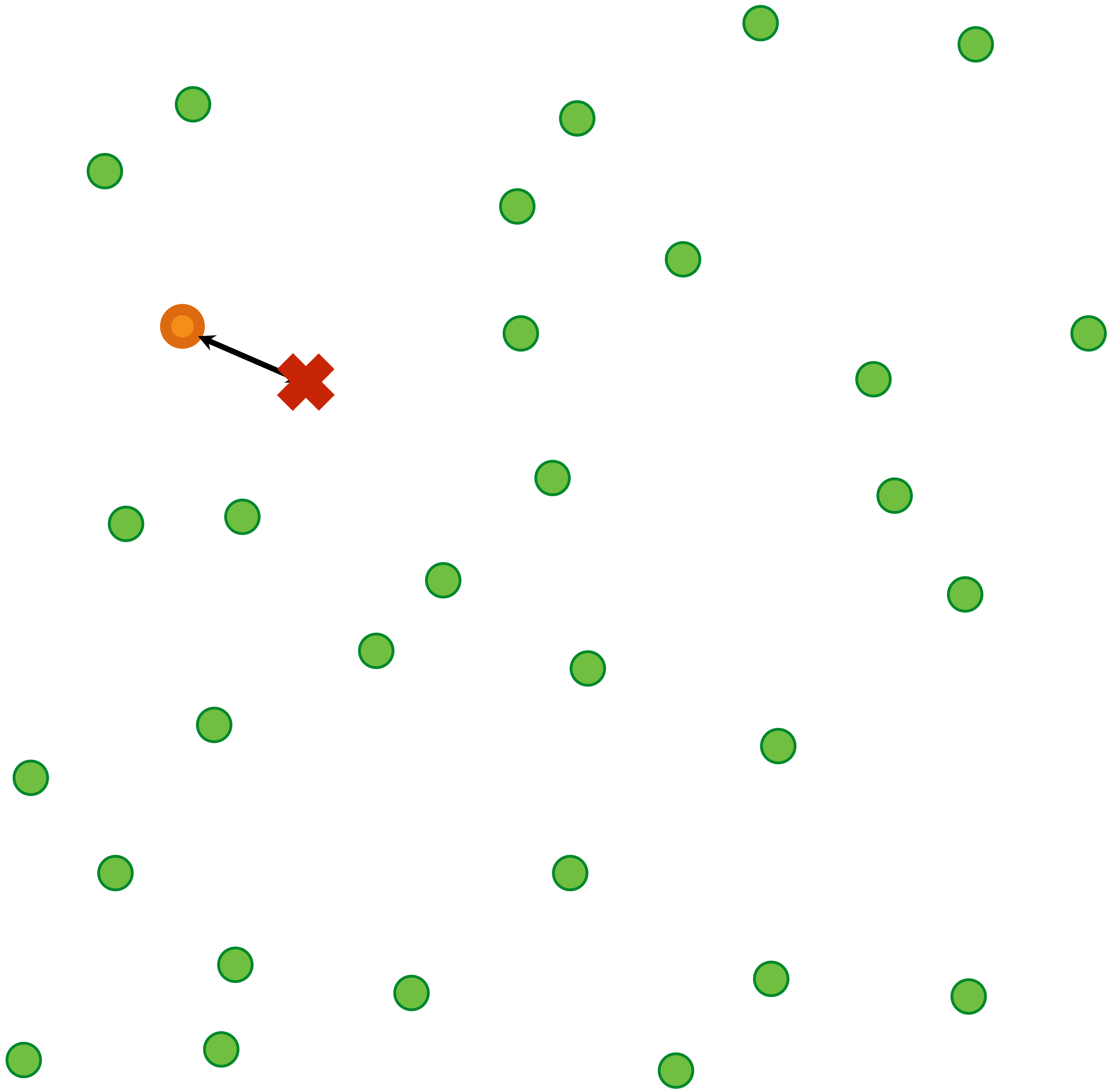
2D Worley noise: f1



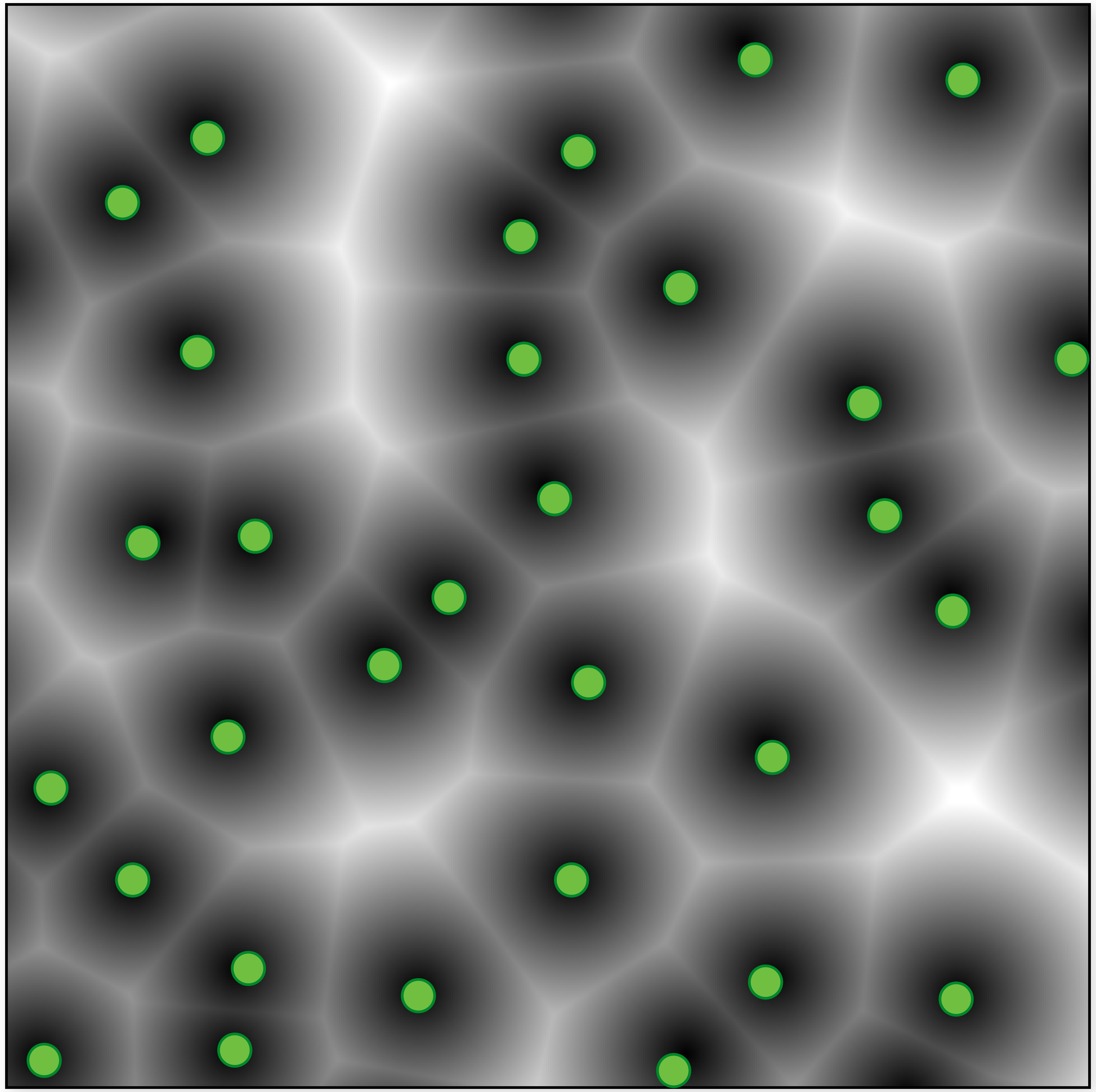
2D Worley noise: f1



2D Worley noise: f1

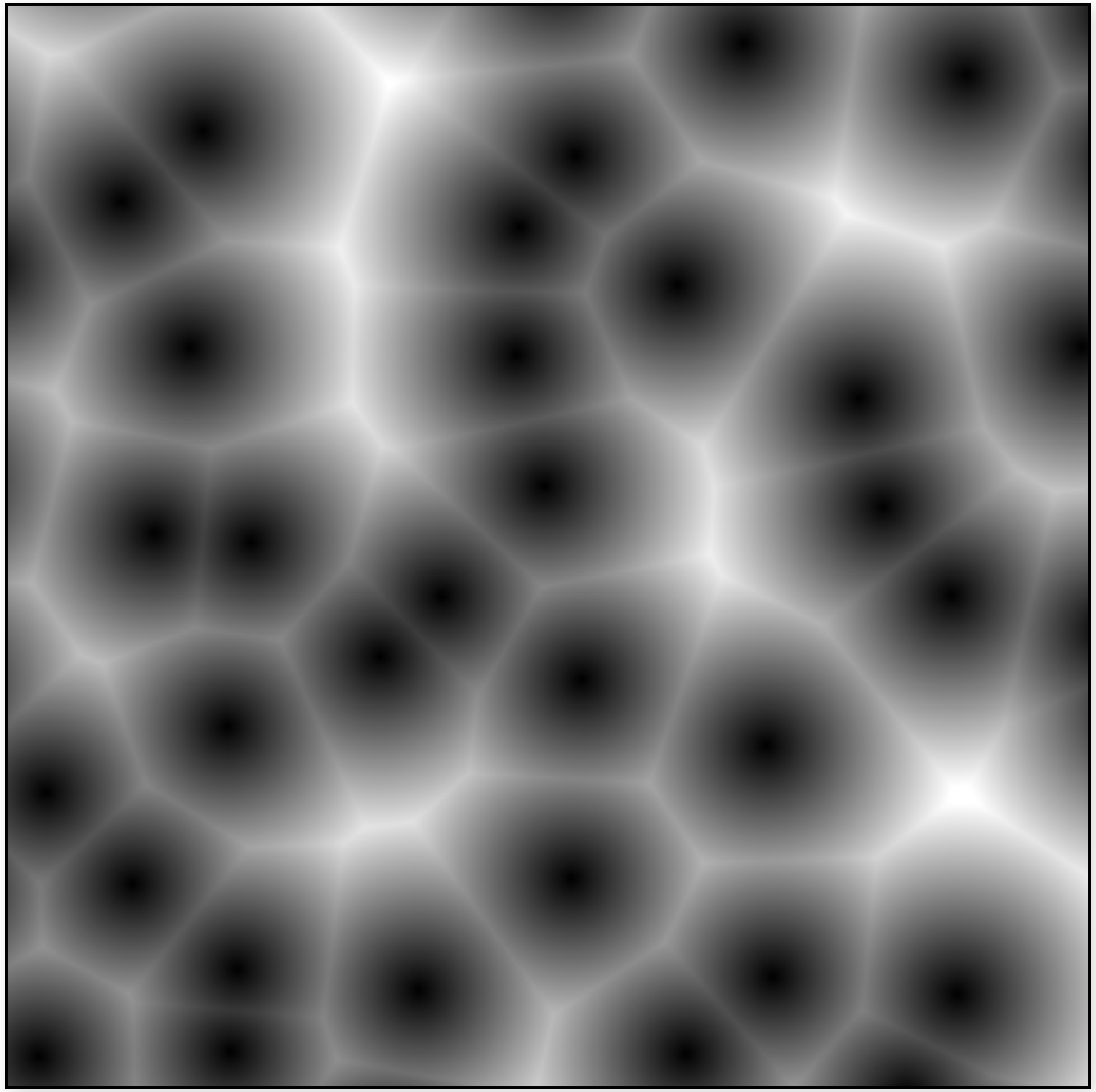


2D Worley noise: f_1



What do we call this image in geometry?

2D Worley noise: f_1



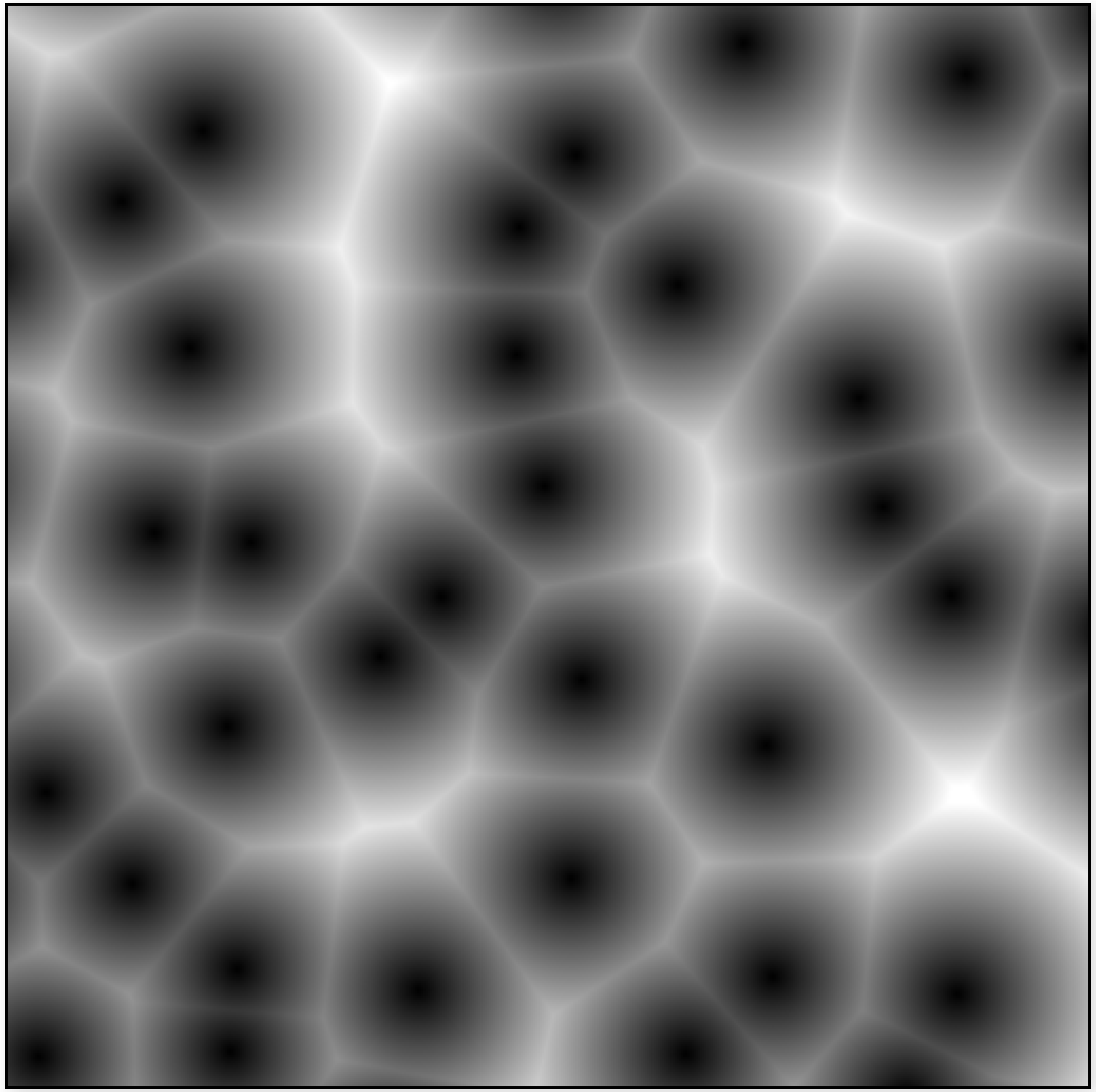
Worley Noise

fractal F1, bump map

Worley Noise

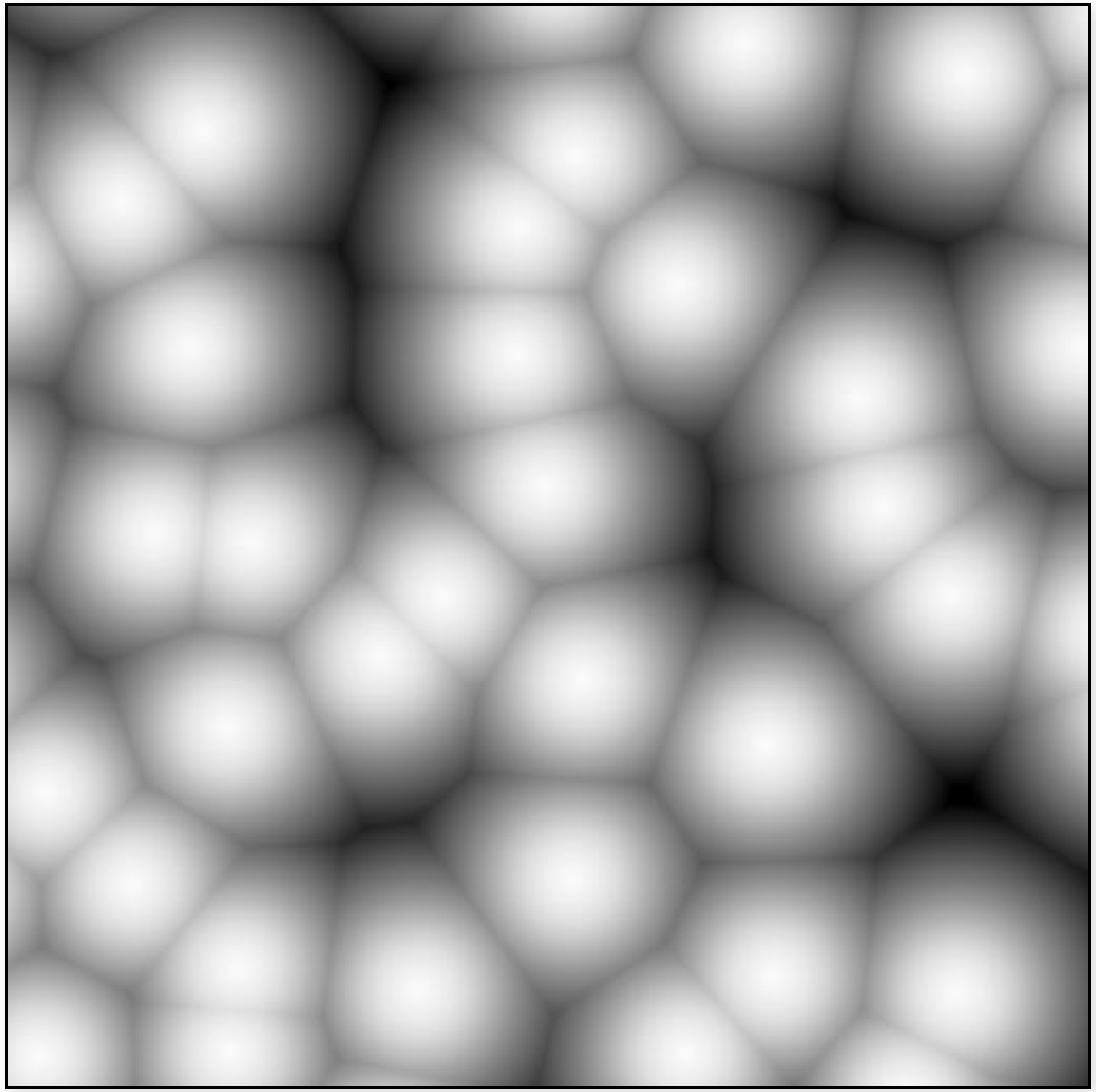
fractal F1, bump map

2D Worley noise: f_1



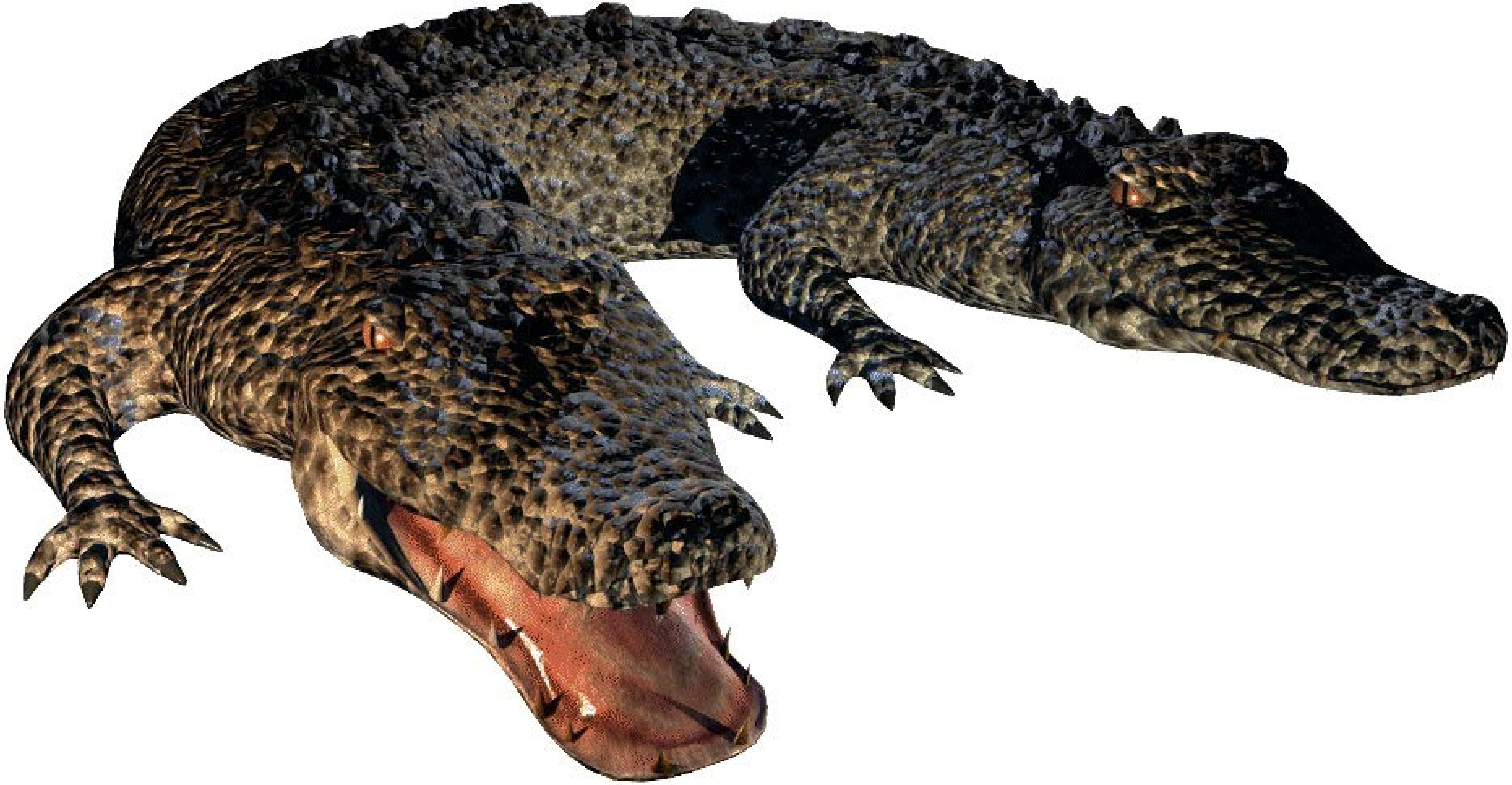
Wojciech Jarosz 2007

2D Worley noise: $1\text{-}f_1$



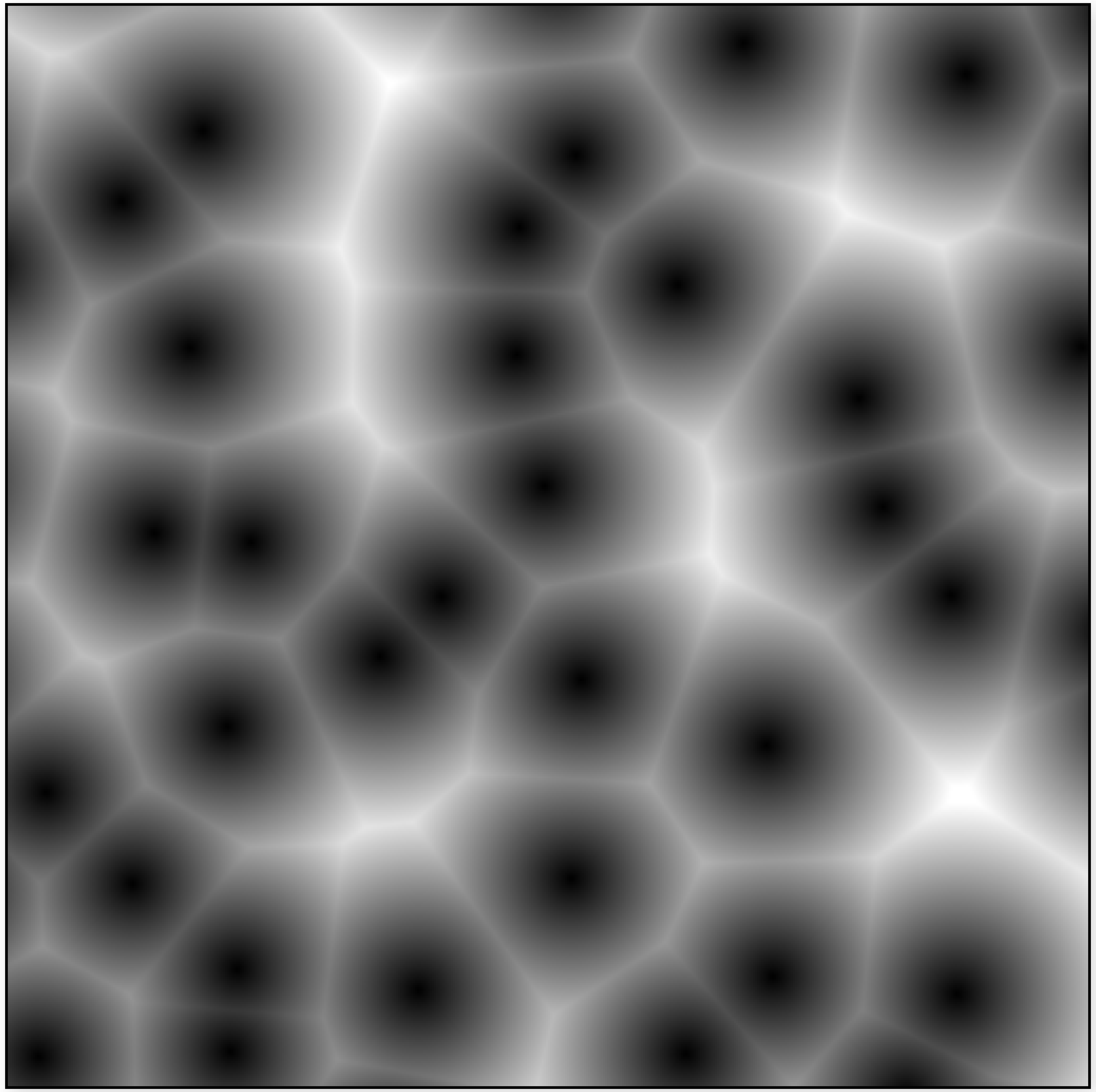
Wojciech Jarosz 2007

Worley Noise



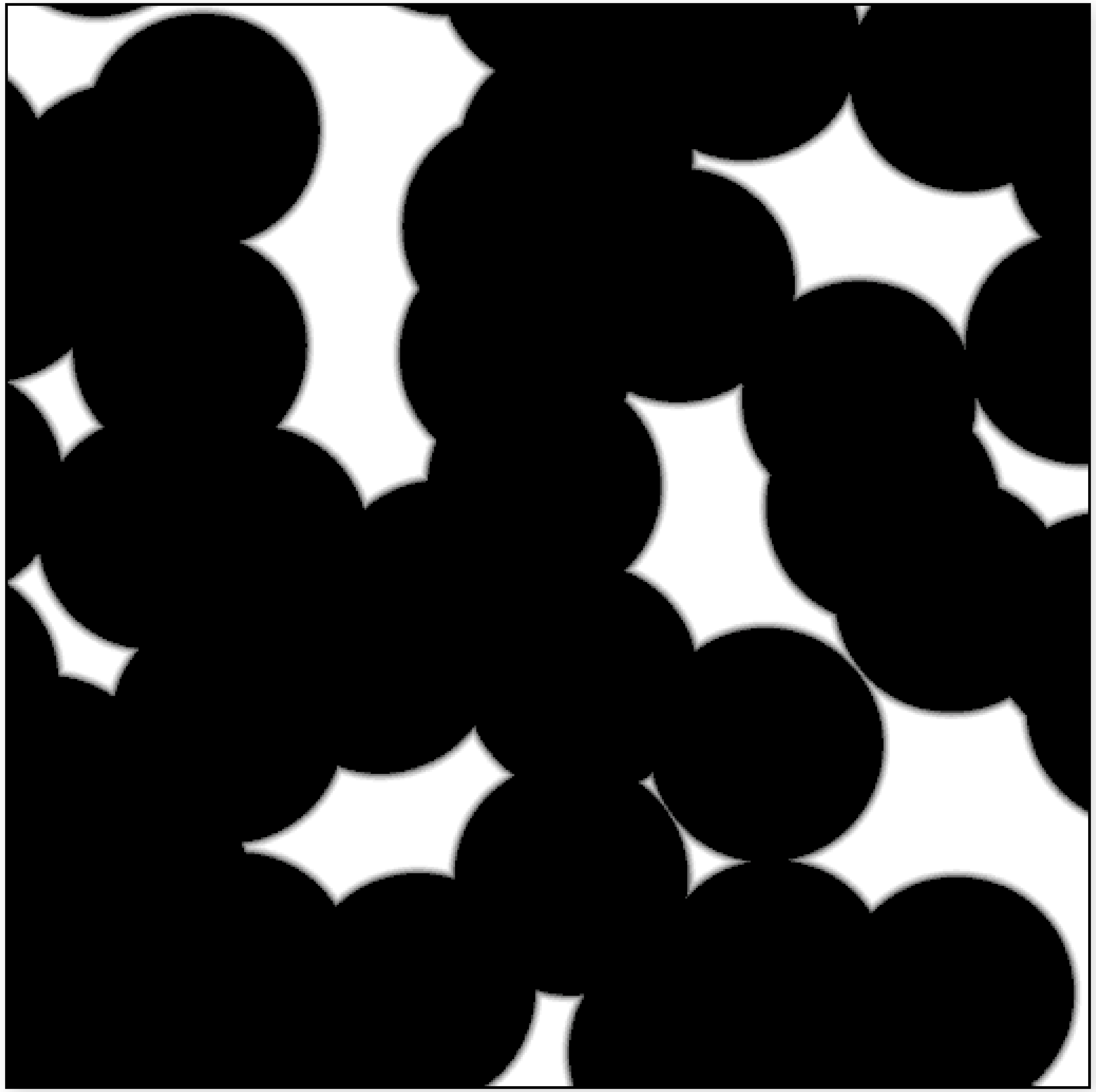
fractal f_1, color and bump map

2D Worley noise: f_1



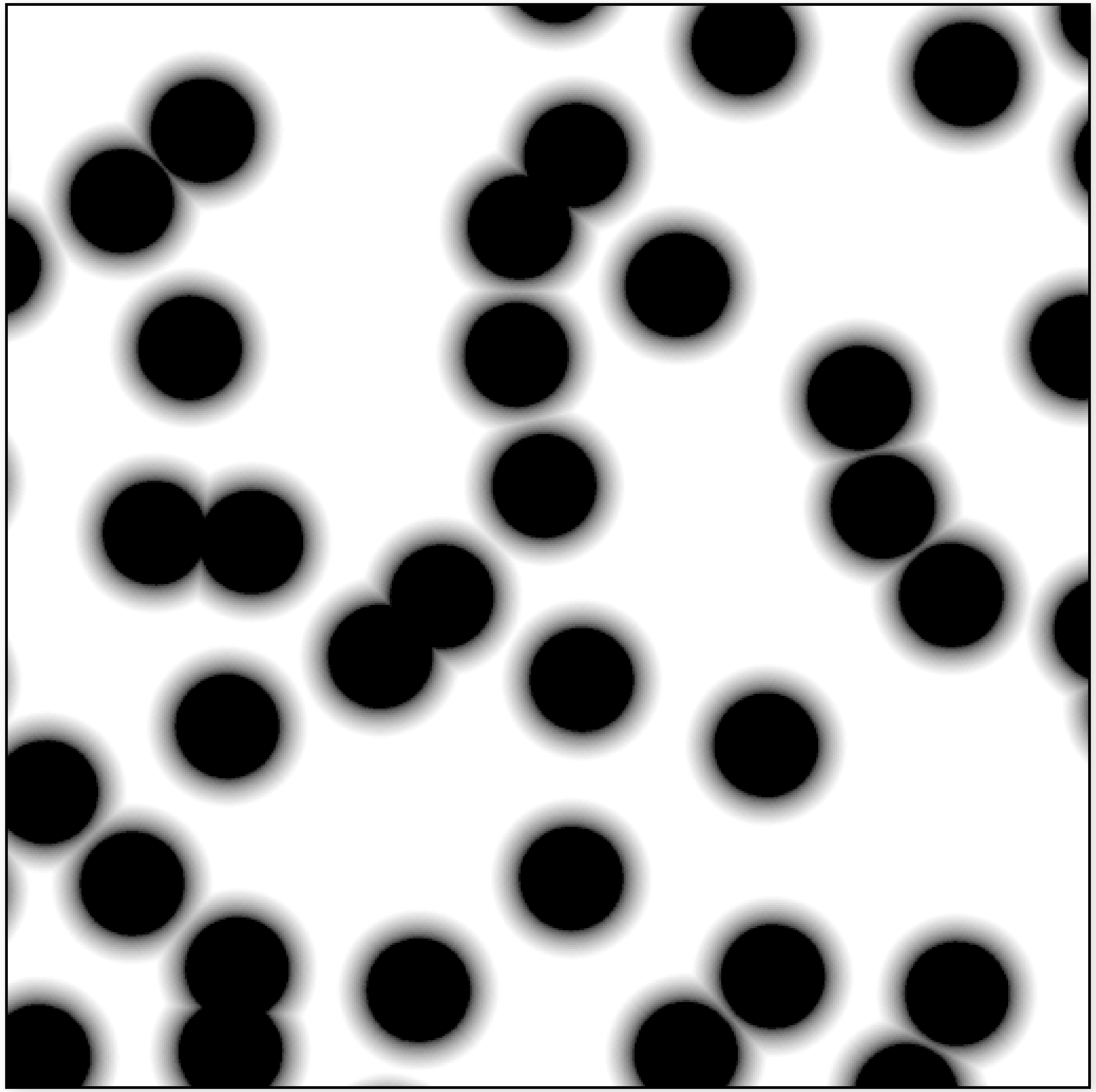
Wojciech Jarosz 2007

2D Worley noise: f_1, thresholded



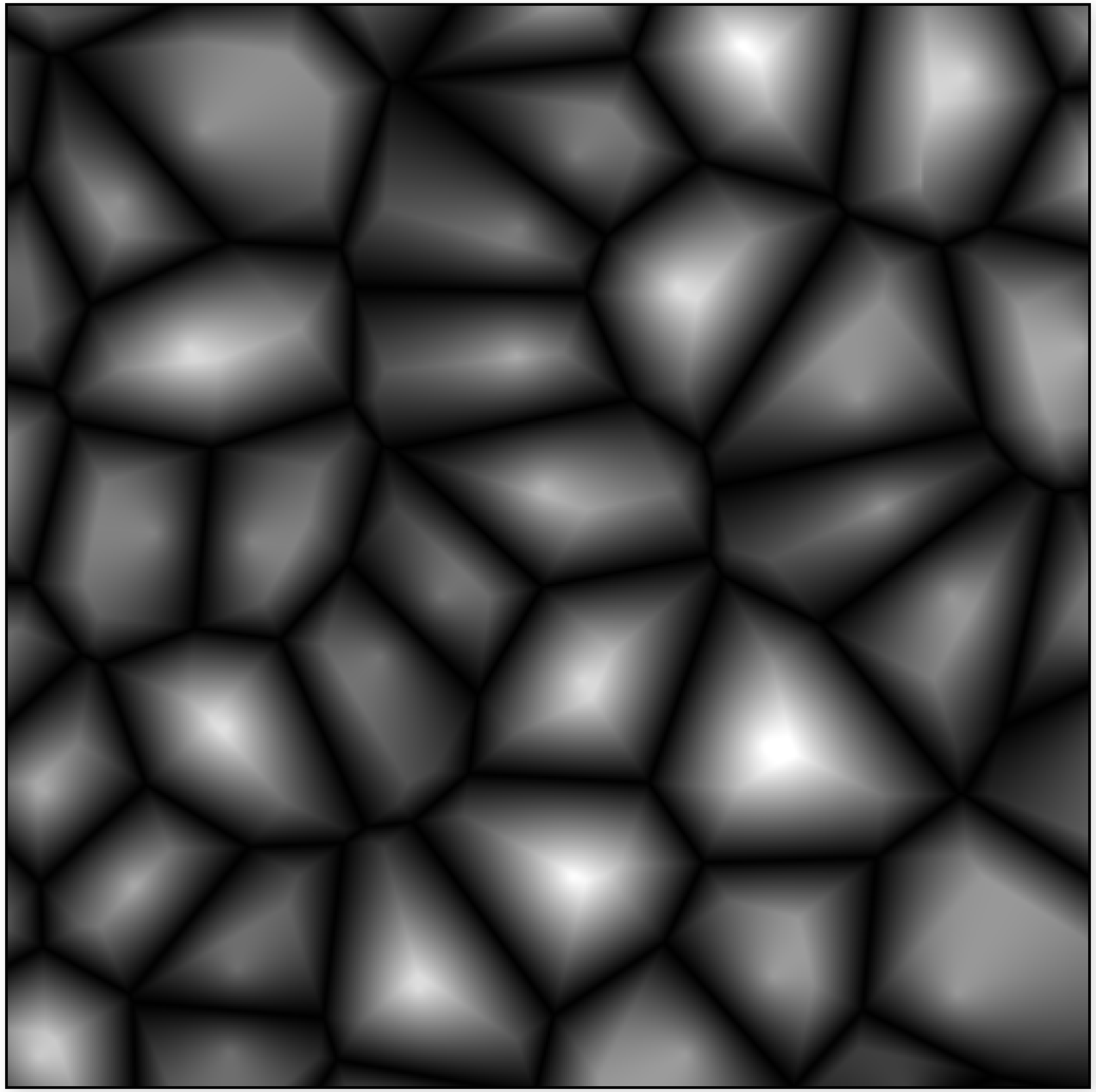
Wojciech Jarosz 2007

2D Worley noise: f_1, thresholded



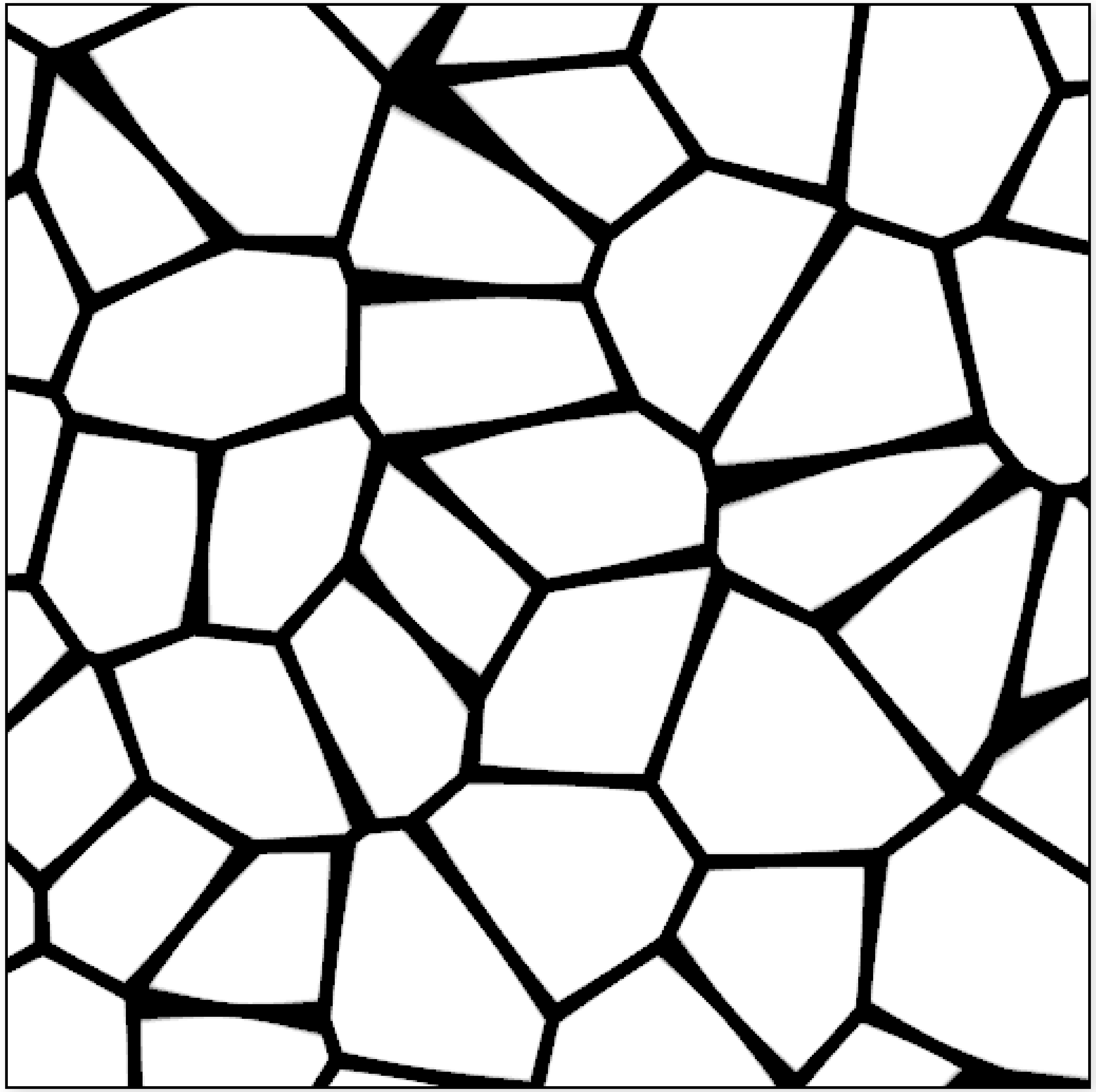
Wojciech Jarosz 2007

2D Worley noise: f_2-f_1



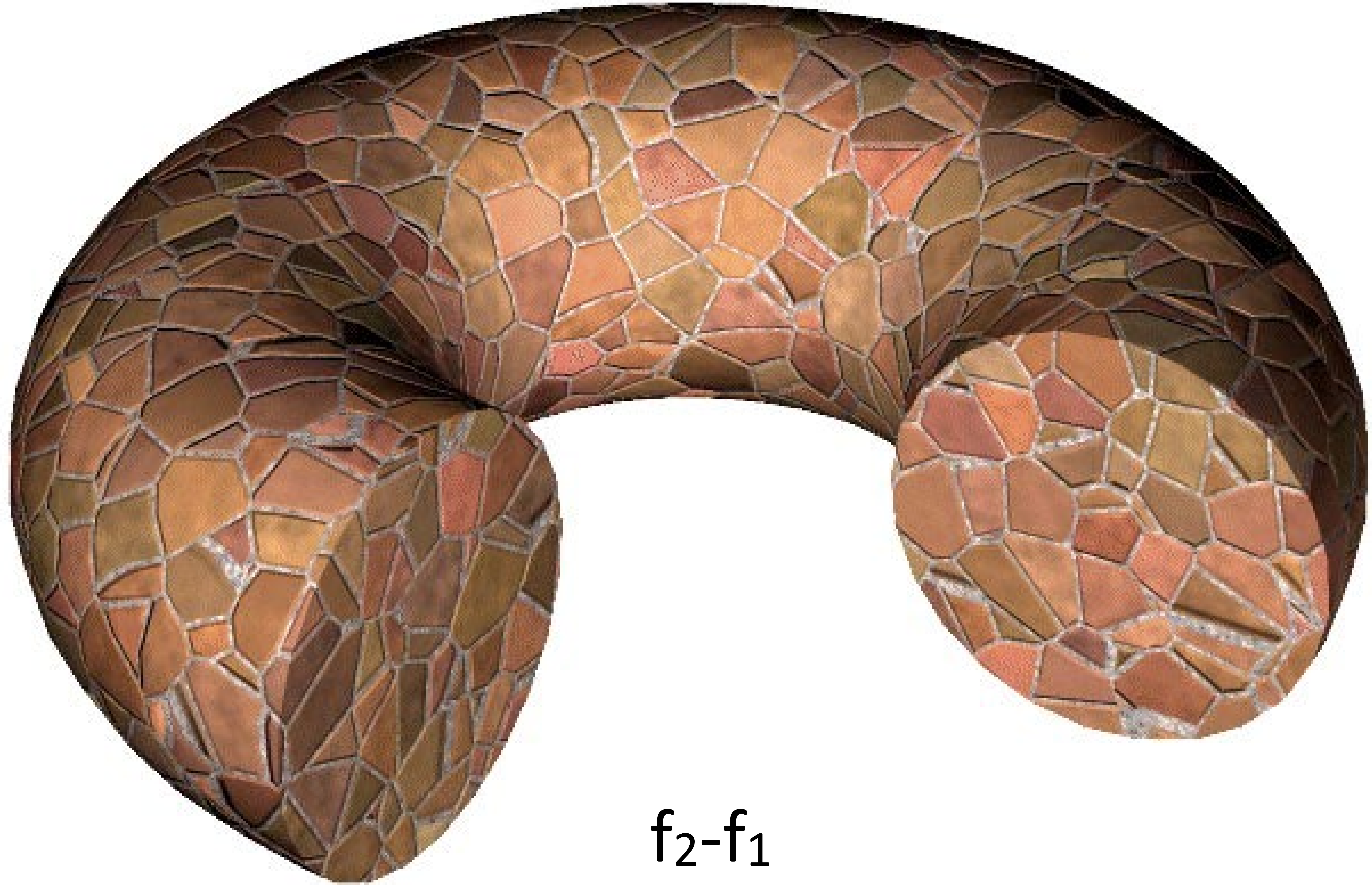
Wojciech Jarosz 2007

2D Worley noise: f_2-f_1, thresholded



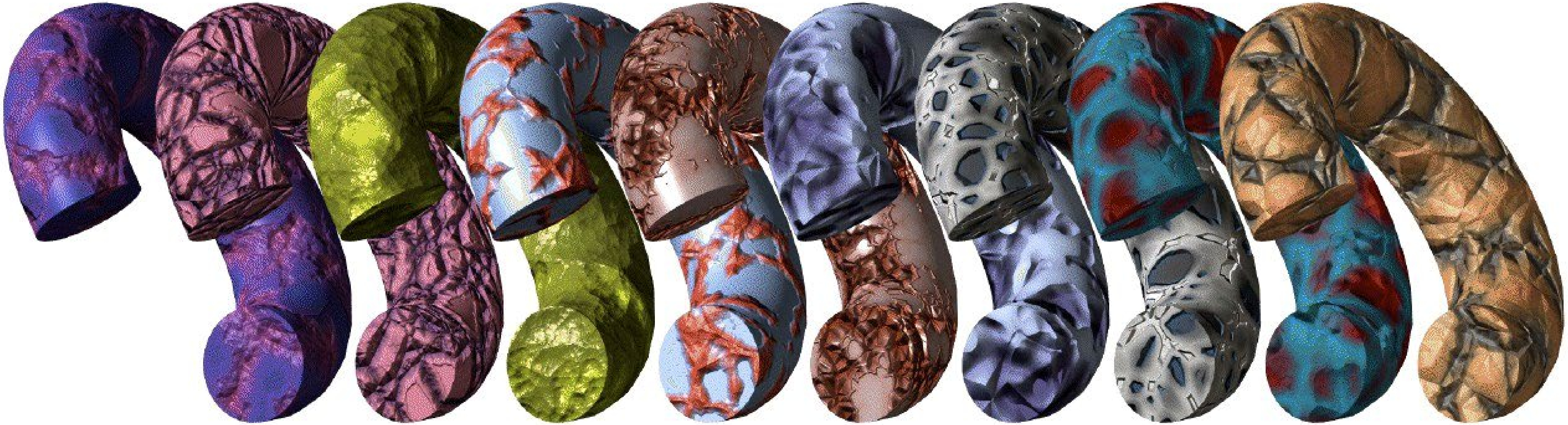
Wojciech Jarosz 2007

3D Worley noise



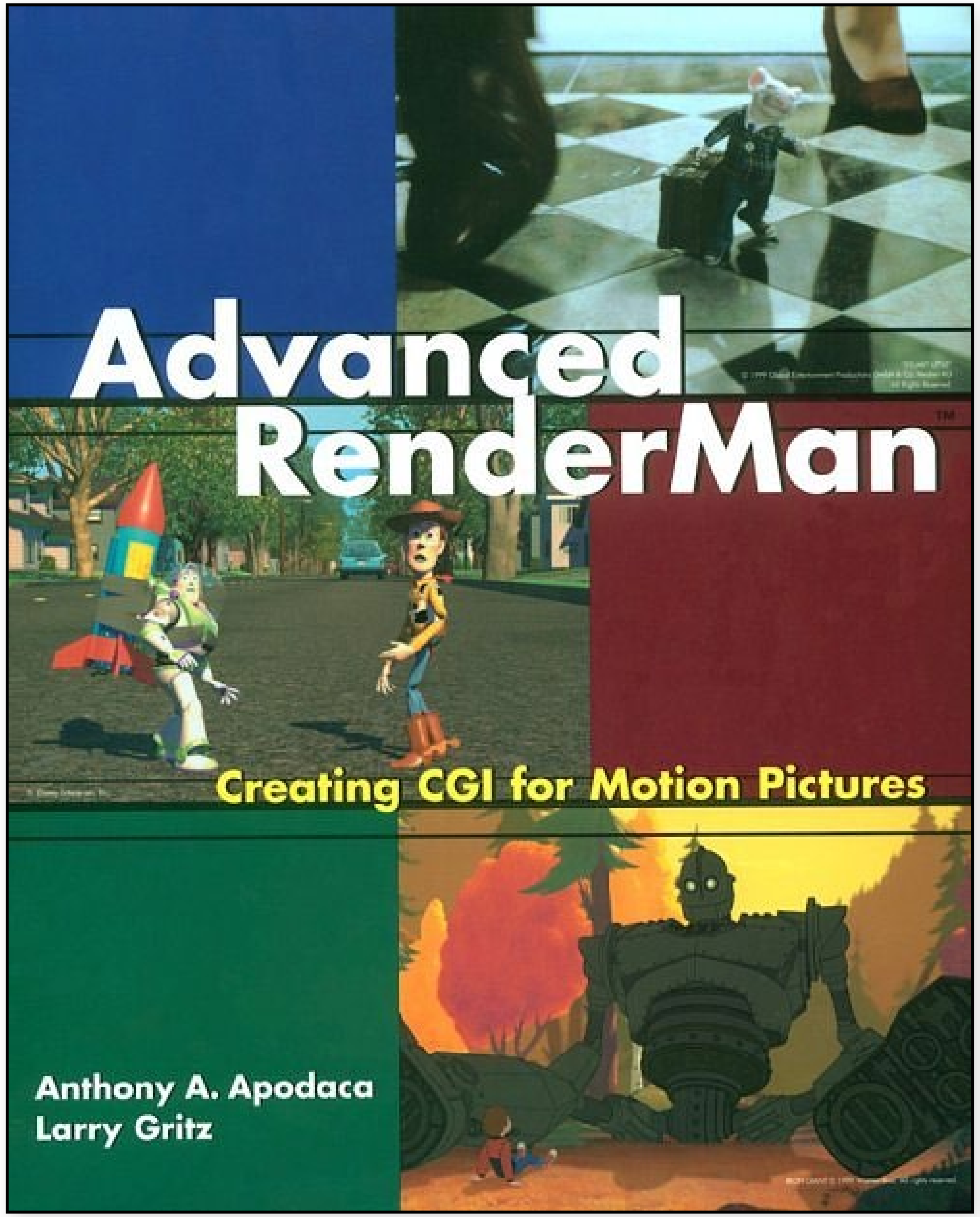
f_2-f_1

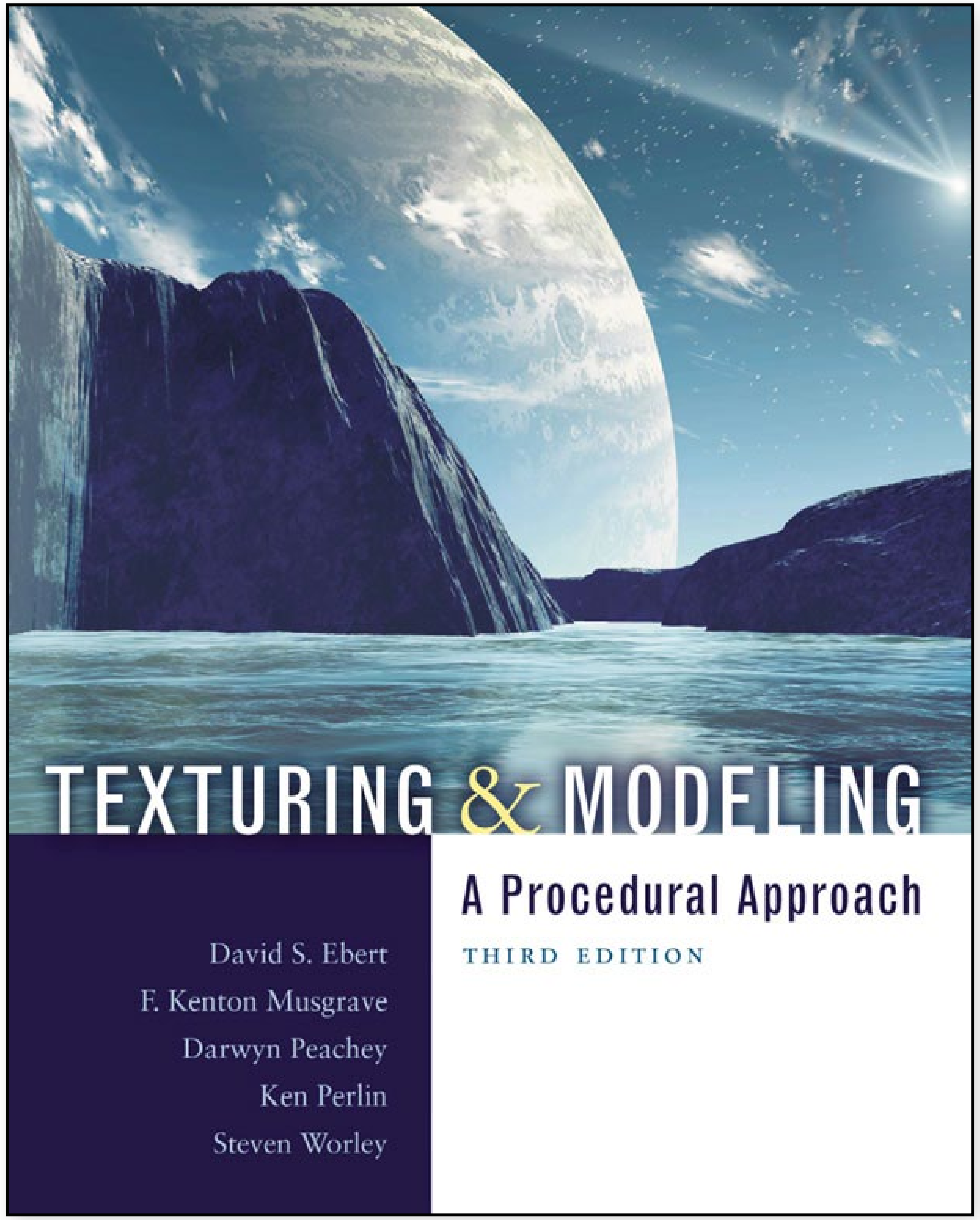
Worley Noise



fractal f1-f4 combinations

Other Resources





Demos

Amazing realtime demos using fractal noise:

- http://www.iquilezles.org/www/articles/morenoise/morenoise.htm
- https://www.shadertoy.com/view/4ttSWf
- https://www.shadertoy.com/view/XttSz2