Catlike Coding

NumberFlow Documentation

Noise Library

This library provides methods that can be used to sample various flavors of pseudorandom noise. All noise types can be sampled in 2D and 3D space. 3D noise can be used to fill a volume or to animate noise on flat surfaces. Multi-frequency samples can be taken to produce fractal versions. All noisy types can be tiled in up to three dimensions.

The library has an example scene that visualizes each noise type with a colored height field. There is also an online version.

1 Noise Parameters

Single-frequency noise only needs a sample point and a frequency as input parameters. Multi-frequency noise also need octaves, lacunarity, and an optional offset. The example images use 3D Perlin noise.

1.1 Sample Point

You sample noise for a specific point in 3D. If you are working in 2D you can leave one dimension at zero. You can also vary this dimension over time to animate the noise.

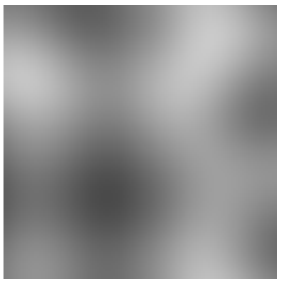
1.2 Offset

When using tiling noise, the samples will be confined to a small region of the noise domain. You can use the offset to change the position of this domain, which results in different noise output. The integer part of the offset is used to move to a different tiling region. The fractional part is used as an offset inside the tile. Offsets for dimensions that aren't tiled are simply added to the sample point.

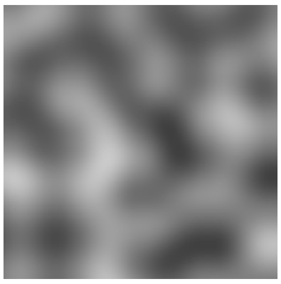
1.3 Frequency

Frequency is effectively the scale of the noise. Higher frequency noise fluctuates faster.

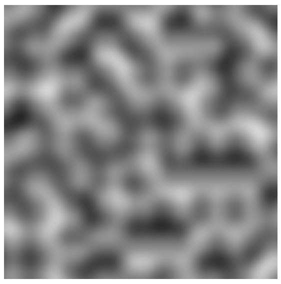
2



4



8

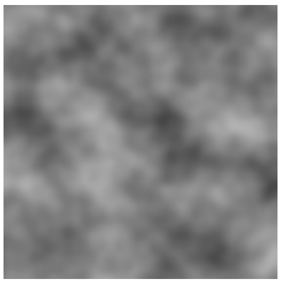


1.4 Octaves

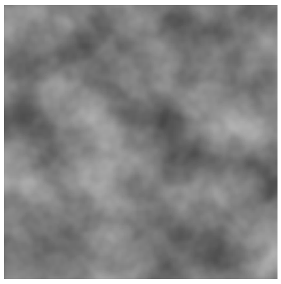
How many octaves of noise to use. This is how often the noise is sampled and added to itself. The sum of all octave samples is normalized so the final value remains in the 0–1 range. The examples below have base frequency 4.

2

3



4

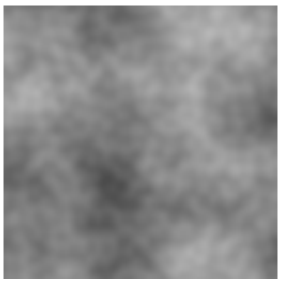


1.5 Lacunarity

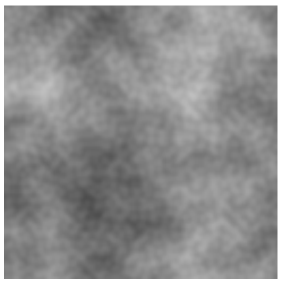
Lacunarity is the gap or distance between octaves. It is a factor by which the frequency is multiplied for each successive octave. It is typically set to 2. The examples below have base frequency 2 and 3 octaves.

2

3



4



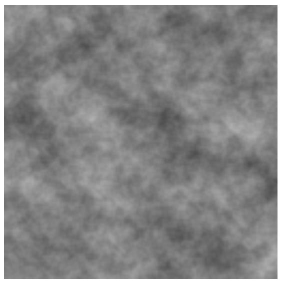
1.6 Persistence

Persistence defines how much successive octaves contribute to the final result. It is a factor by which the sample result of an octave is multiplied. It is multiplied with itself for each octave. It is typically set to 0.5. Negative persistence values should not be used, as it breaks the normalization of the final sum. The example below have base frequency 4 and 5 octaves.

0.25

0.50

0.75



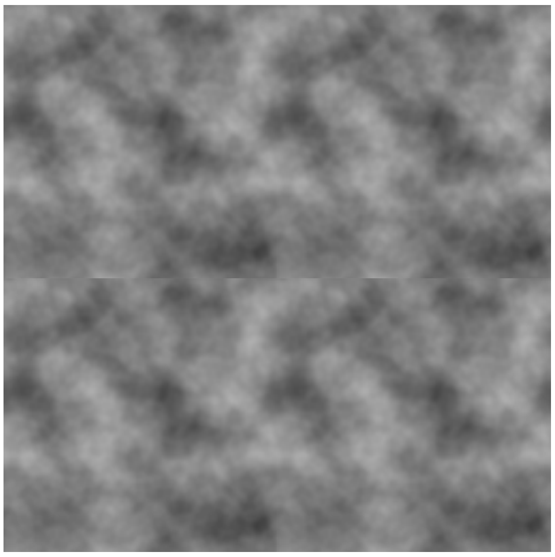
2 Tiling

It is possible to tile the X dimension, the X and Y dimensions, and all three dimensions. Besides creating tilable textures, you can also create looping animations by moving through a tiling dimension. If you want to tile another dimension, you can do so by swapping axes for the sample point.

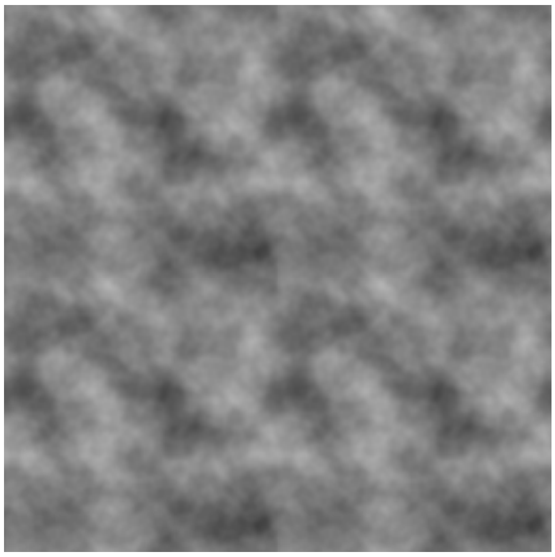
Tiling happens inside a sampling range of 0–1. To sample over a larger effective range, use a higher base frequency.

Seamless tiling is achieved by matching permutation values used on opposite sides of the tile. This produces no tiling artifacts, but will look regular at low frequencies.

Tiling X



Tiling XY



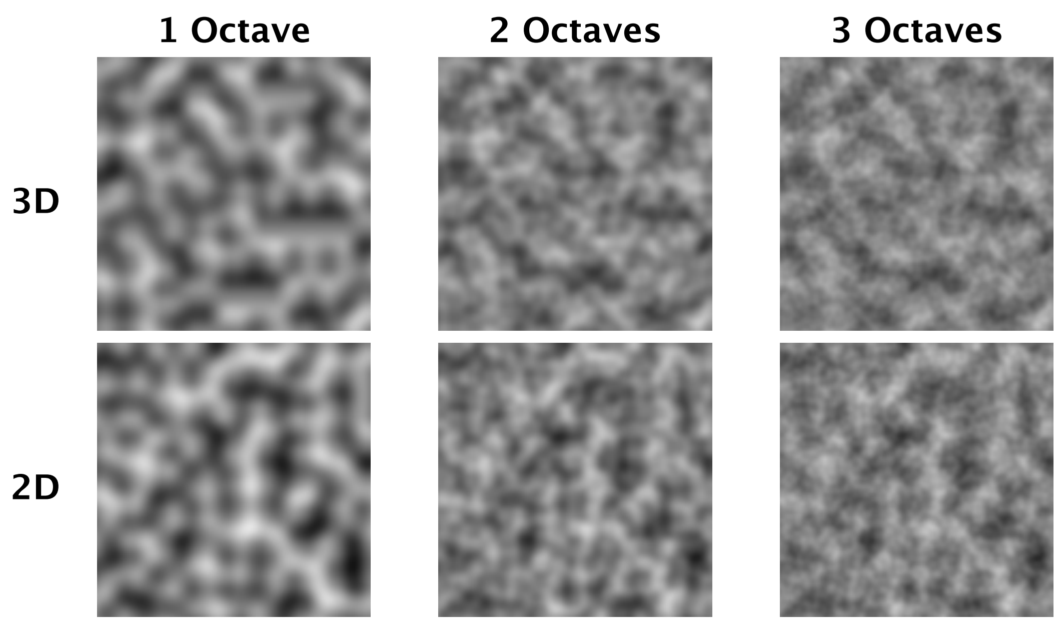
3 Noise Types

The library contains classes to sample Perlin and Turbulence noise, and various types of Voronoi noise.

3.1 Perlin

The `PerlinNoise` class produces noise that matches the reference implementation of improved 3D Perlin noise, with a few changes. Traditional Perlin noise is centered on 0 and has an effective range of roughly -0.7–0.7. The single-frequency sampling method follows this convention, except that the range has been normalized to -1–1. The multi-frequency sampling method normalizes its output so it falls in the 0–1 range, like all other noise types.

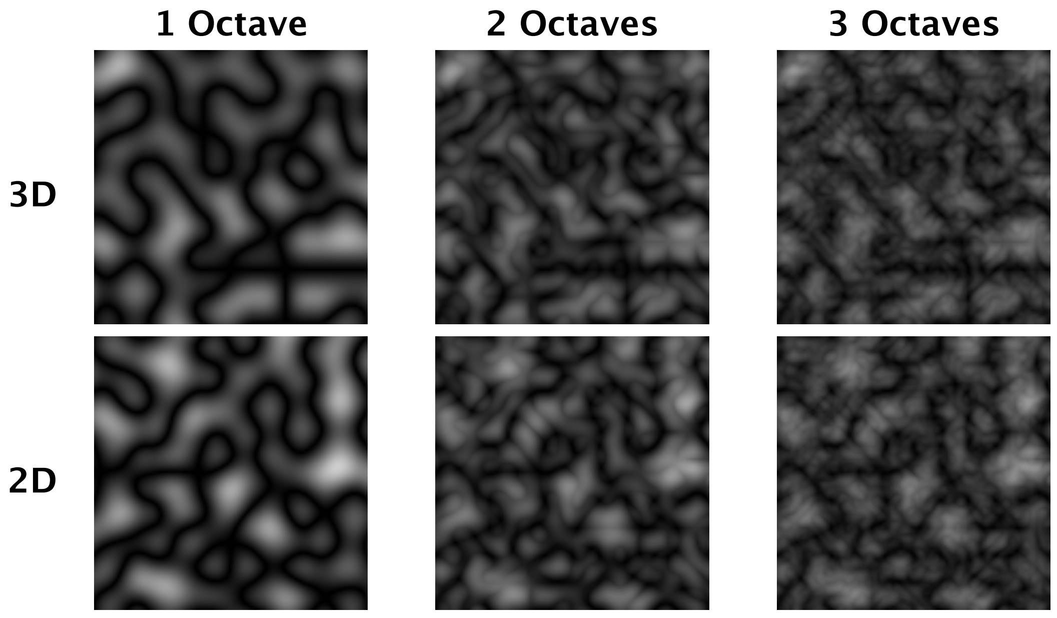
The 2D version uses 16 gradient directions instead of the typical 8. This produces a slightly move wavy pattern than the 3D version.



3.2 Turbulence

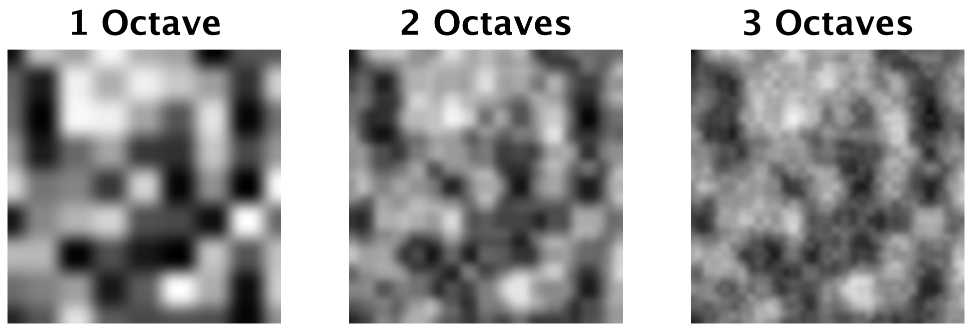
Turbulence is a variant of Perlin noise which takes the absolute value of the original noise. This results in a sharp ridge where Perlin noise would have crossed 0. As its output thus falls inside the 0–1 range, it is not further adjusted. Its sampling methods are included in the `PerlinNoise` class.

The 3D verion tends to have long straight lines with a sample value near 0. The 2D version has more slight fluctuations, because it uses 16 gradient directions.



3.3 Value

These methods in the `ValueNoise` class produce value noise. It works like Perlin noise, but blends between constant values instead of gradients. The output values are in the 0–1 range and are centered on 0.5. When animating a 2D slice by moving through the third dimension, its block structure will be very obvious.



Despite the lack of a third dimension, the 2D version has the same visual appearance as 3D versions when viewing an XY-aligned slice.

3.4 Voronoi

The `VoronoiNoise` class produces various flavors of Voronoi noise, also known as Worley noise and sometimes Cell noise.

The sampling methods each produce a 3D vector which contains three measurements of the sampled point. These values are each in the 0–1 range per octave. Besides using these values in isolation, they are also often combined.

The X coordinate contains **F1**, which is the distance to the closest cell center, with a maximum of 1.

The Y coordinate contains **F2**, which is the distance to the second closest cell center, with a maximum of 1.

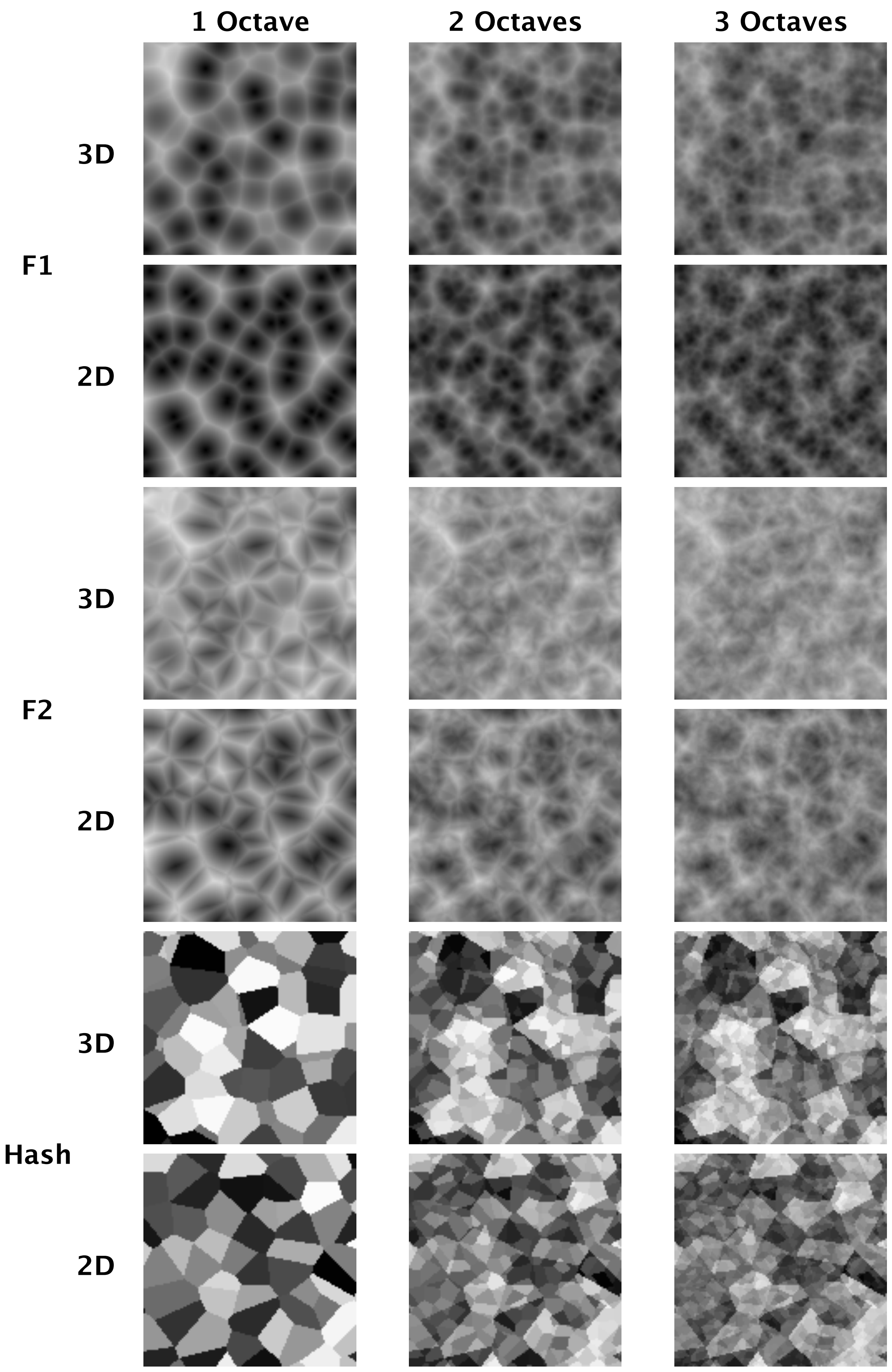
The Z coordinate contains **Hash**, which is the hash value of the cell containing the sample point.

While the original noise proposed by Worley had a variable number of voronoi cells per lattice cube, this implementation always uses two per cube. It is more common to use just one cell per cube, but two cells produce a more interesting pattern and far less distances that would exceed 1. The exception is Chebyshev noise, which needs only one cell per cube because of its short distances.

There are four ways the distances can be computed, resulting in four different patterns.

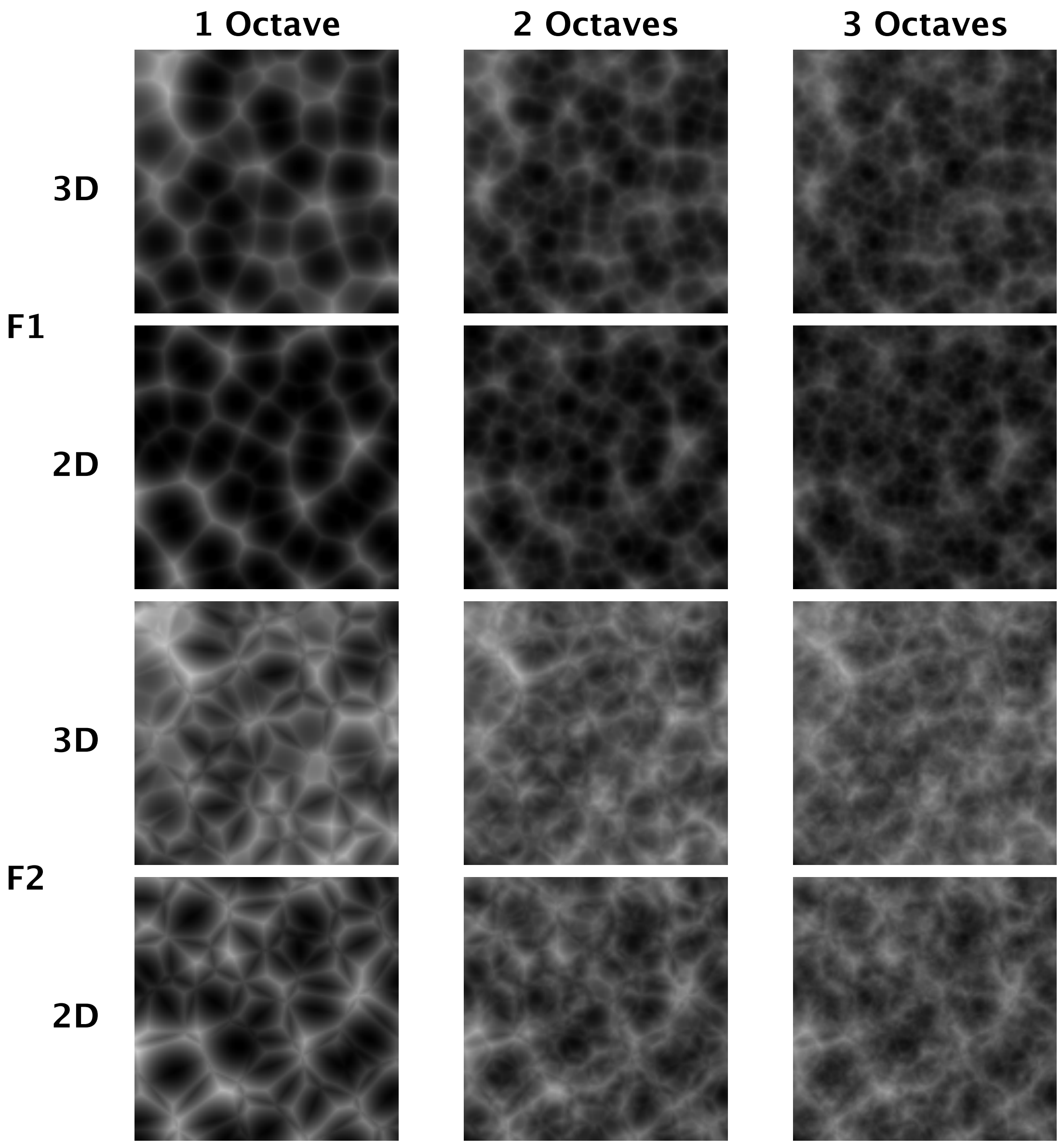
3.4.1 Linear

Linear mode uses the usual Euclidean distance. Linear is expensive to compute because it requires square root compuations.



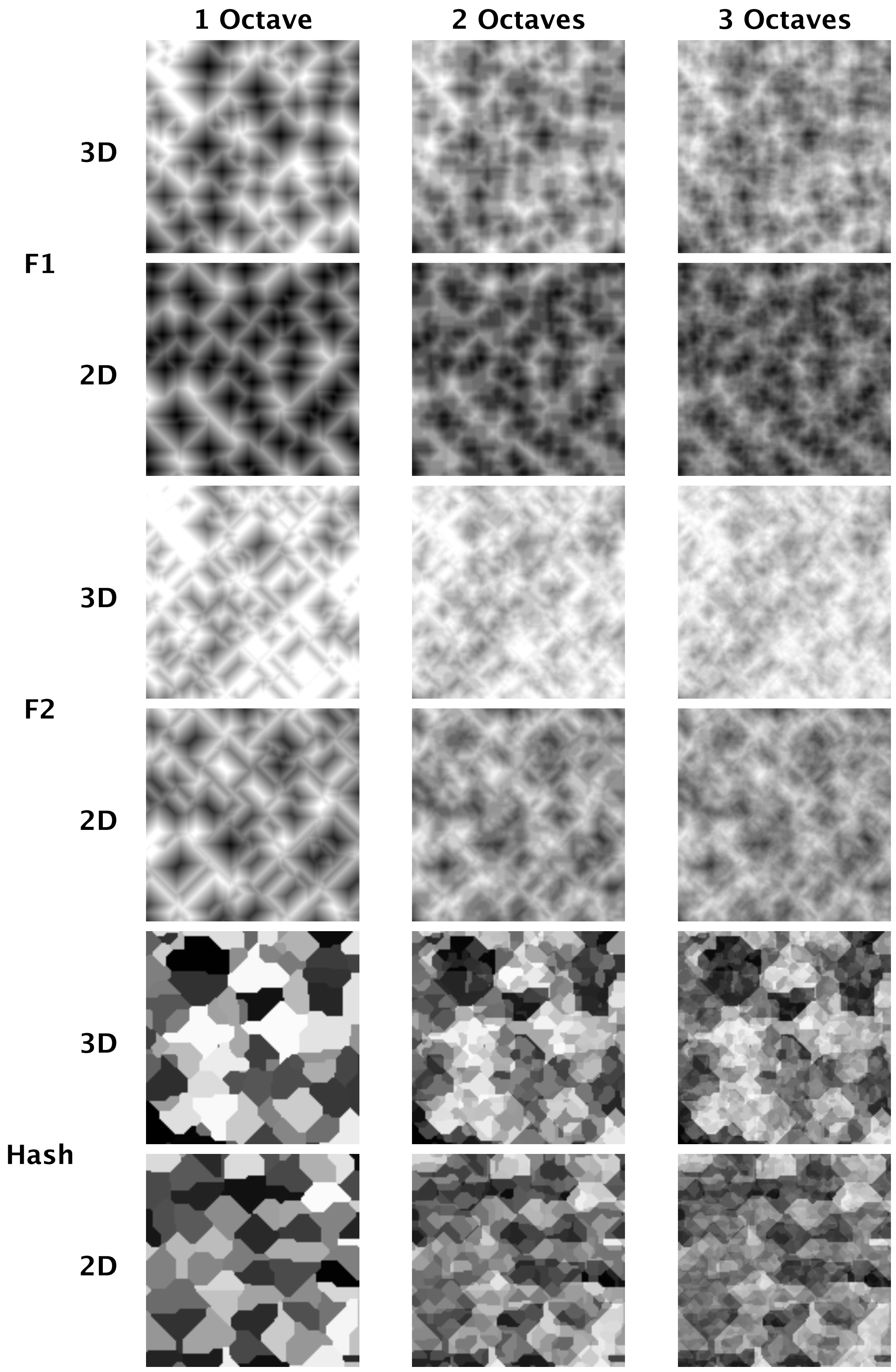
3.4.2 Squared

Square mode is the same as Linear, except that it uses squared distances. This makes it cheaper to calculate than Linear mode. Although the distances will come out smaller, it produces the same hash pattern.



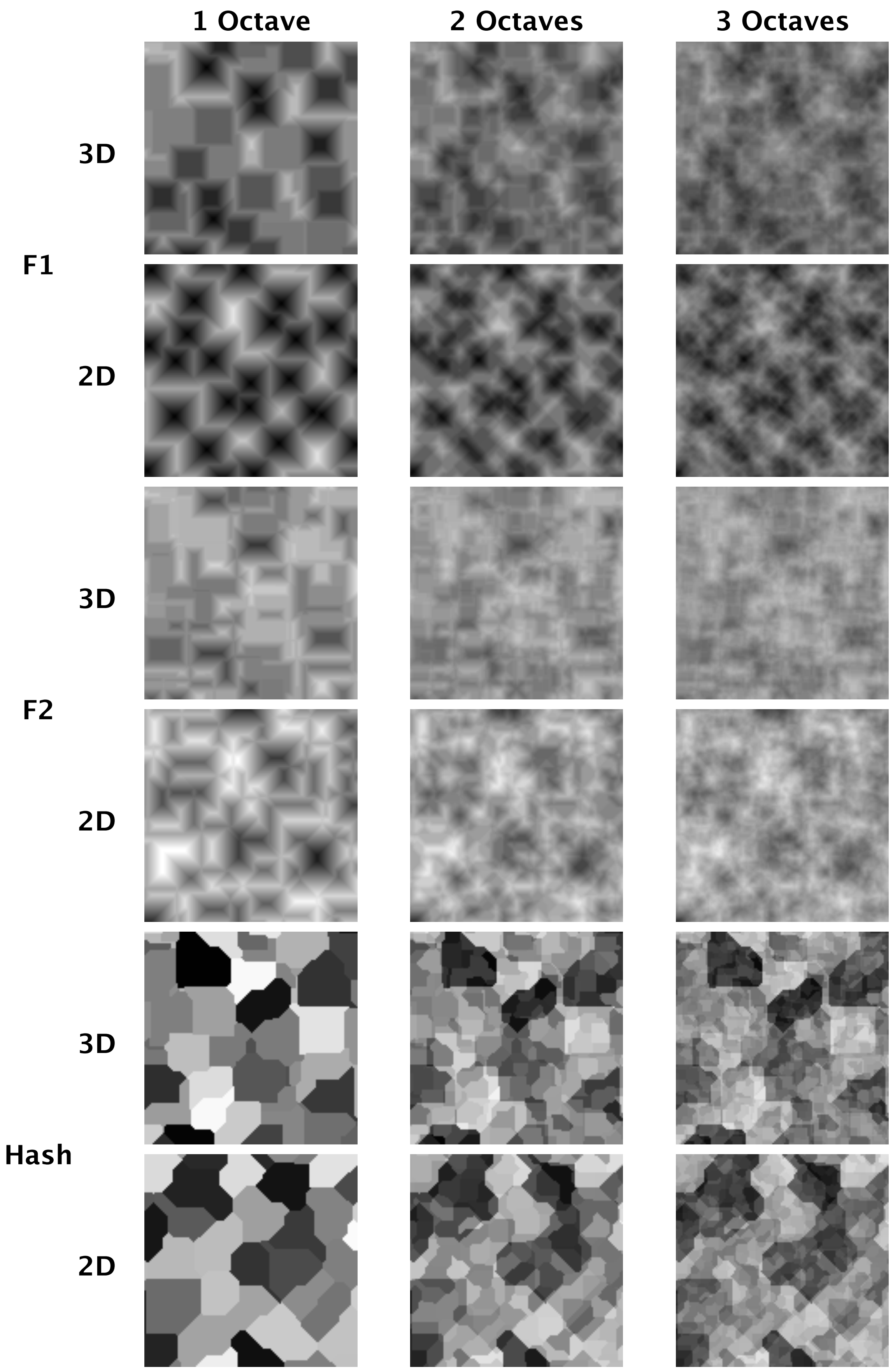
3.4.3 Manhattan

Manhattan distance is measured by only moving in straight lines along the axes. So diagonal movement is not allowed, which leads to increased distances. This type contains many regions that are clamped to the maximum distance.



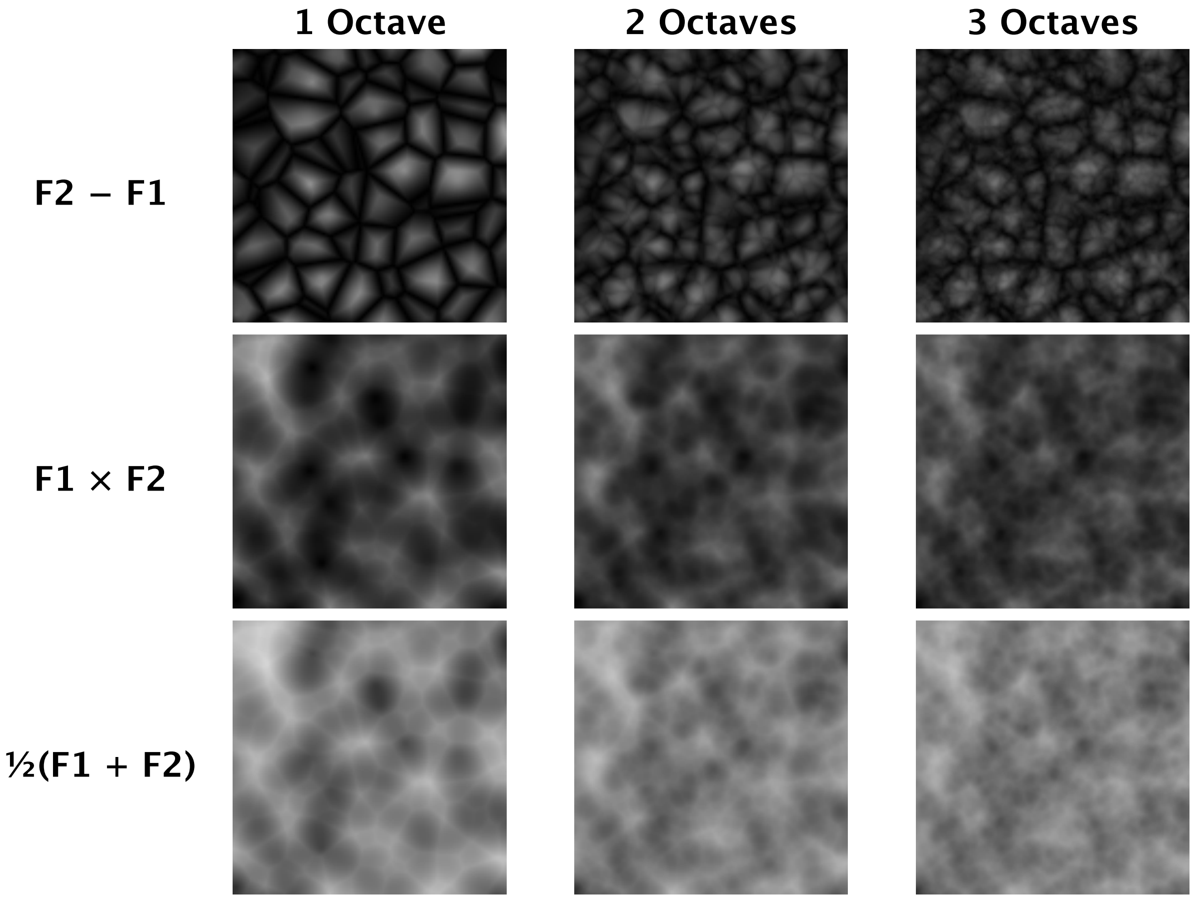
3.4.4 Chebyshev

Chebyshev distance is measures by taking the largest distance along any of the axes as the total distance. As this eliminates all but one dimension, it leads to significantly shorter distances and produces 3D regions where the distances are uniform.



3.4.5 Combining F1 and F2

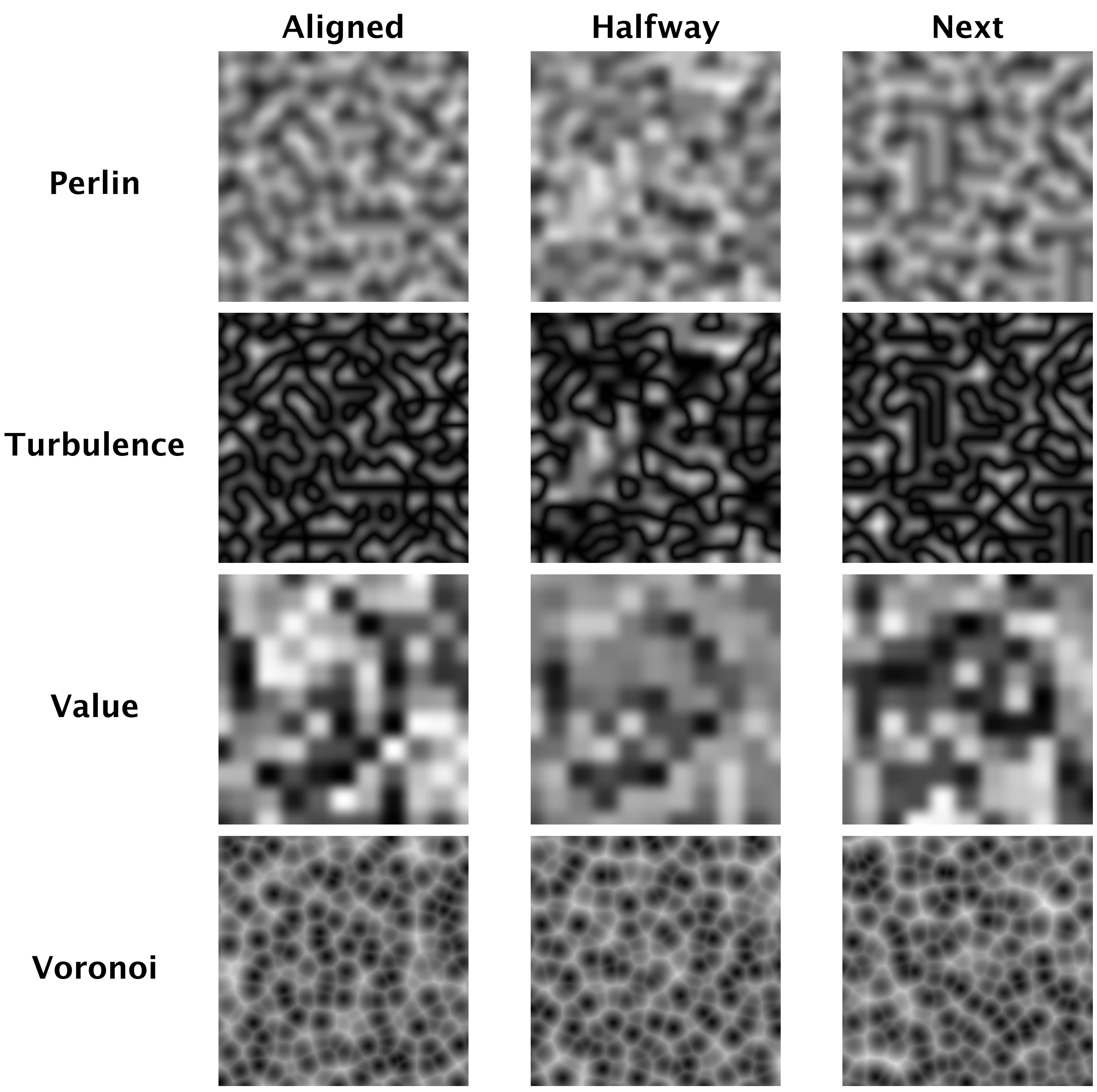
You can create more variations of Voronoi noise by combining the F1 and F2 results. Typical operations are subtraction, multiplication, and addition or averaging. Keep in mind that both samples always fall in the 0–1 range and that F2 is always at least as large as F1. The below examples use linear 3D noise.



4 Moving Through 3D

While you can animate noise textures by moving 2D slices through 3D space, you have to be aware of the inherent regularity of lattice noise. Perlin, Turbulence, and Value noise are most distinct or pure when slices align with the underlying lattice grid. For example, for a base frequency of 10 this means any Z offset that is a multiple of 0.1, for XY-aligned slices. Any other Z offset produces a blend between two successive lattice-aligned slices. This is the exact same interpolation that happens across the 2D slice itself, but it is at the same point for the entire slice. If you were to use a slice that is not axis-aligned, you'd get varying interpolations across the image.

In contrast, voronoi noise looks the same no matter where you sample it, because it doesn't interpolate between lattice points.



documentation written by Jasper Flick