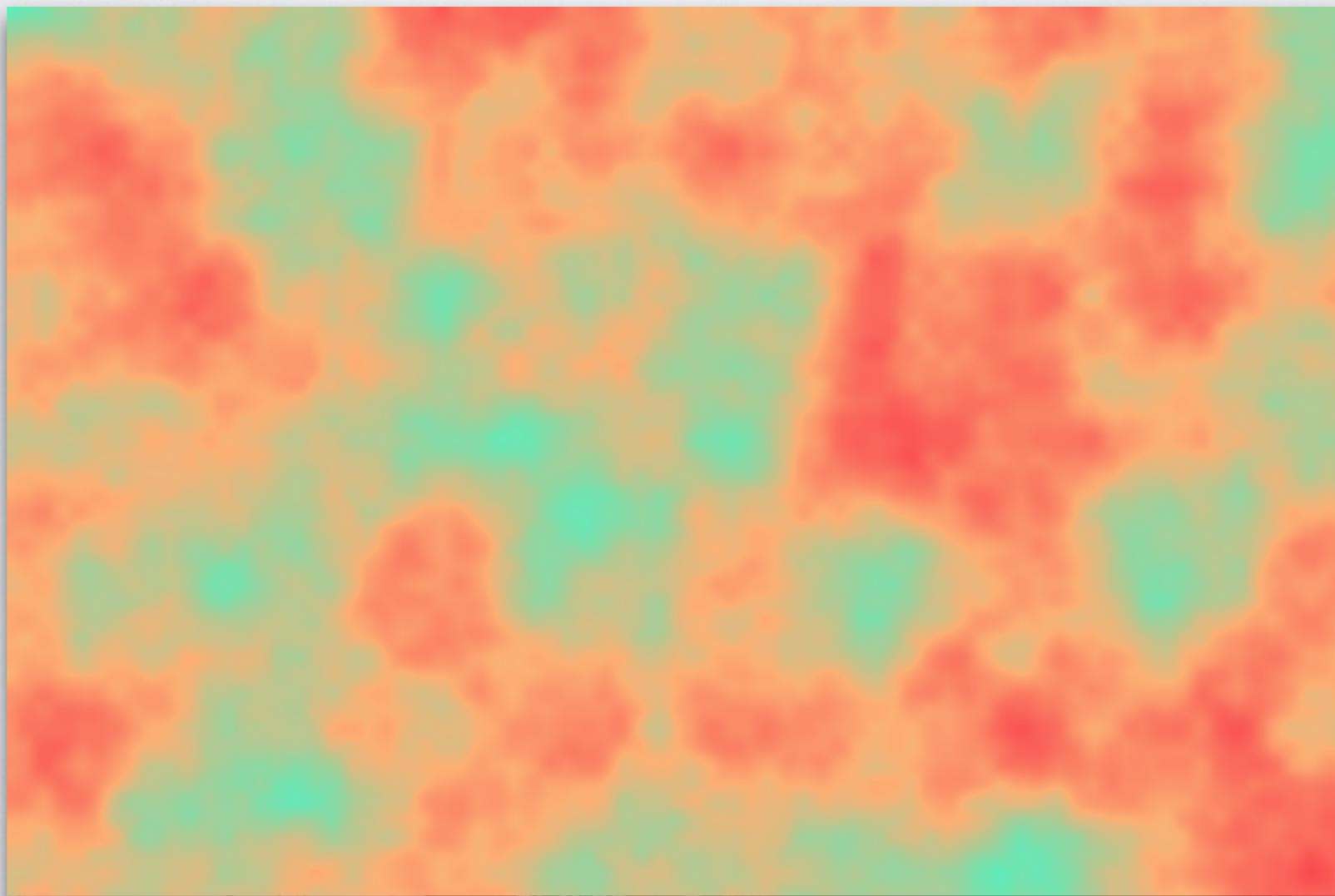


CIS 700

NOISE FUNCTIONS



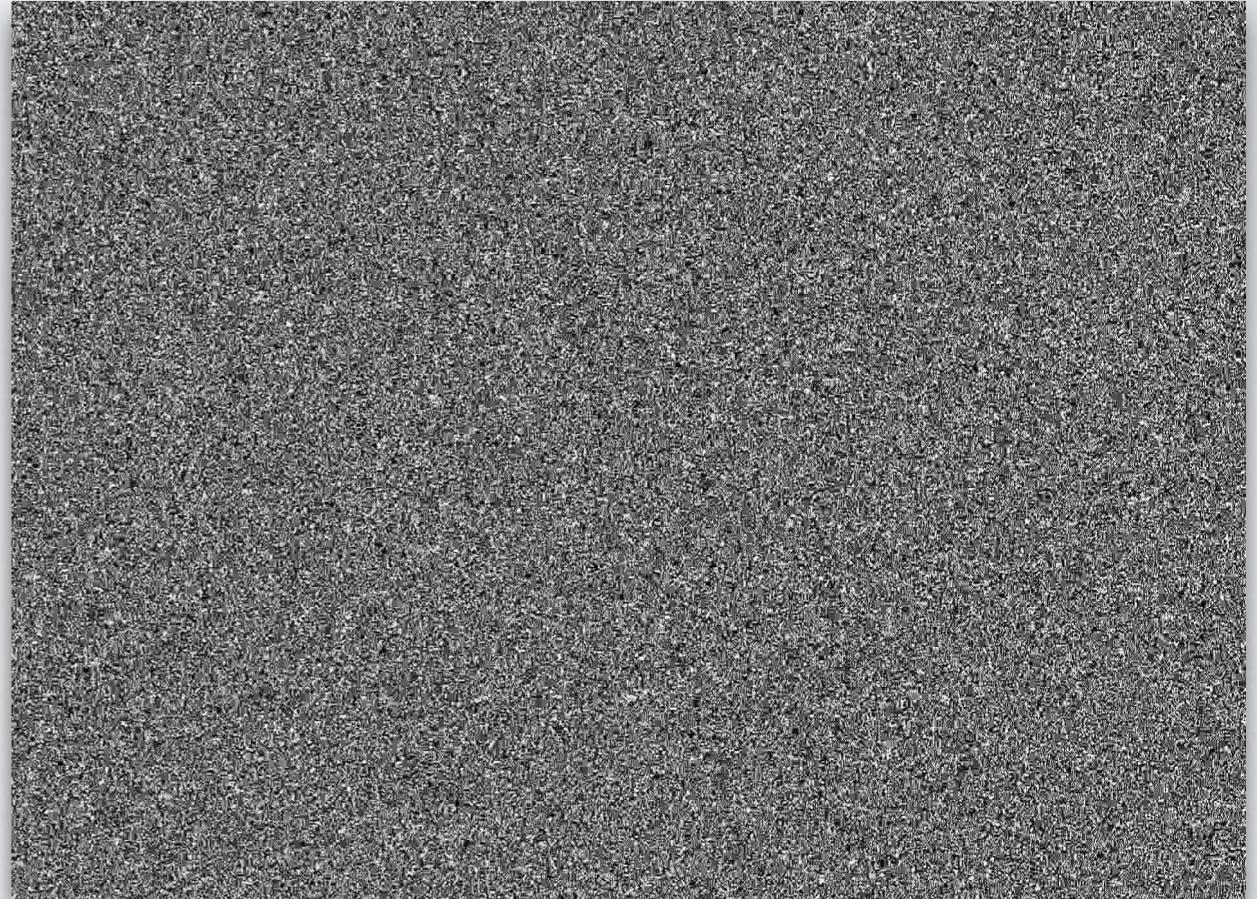
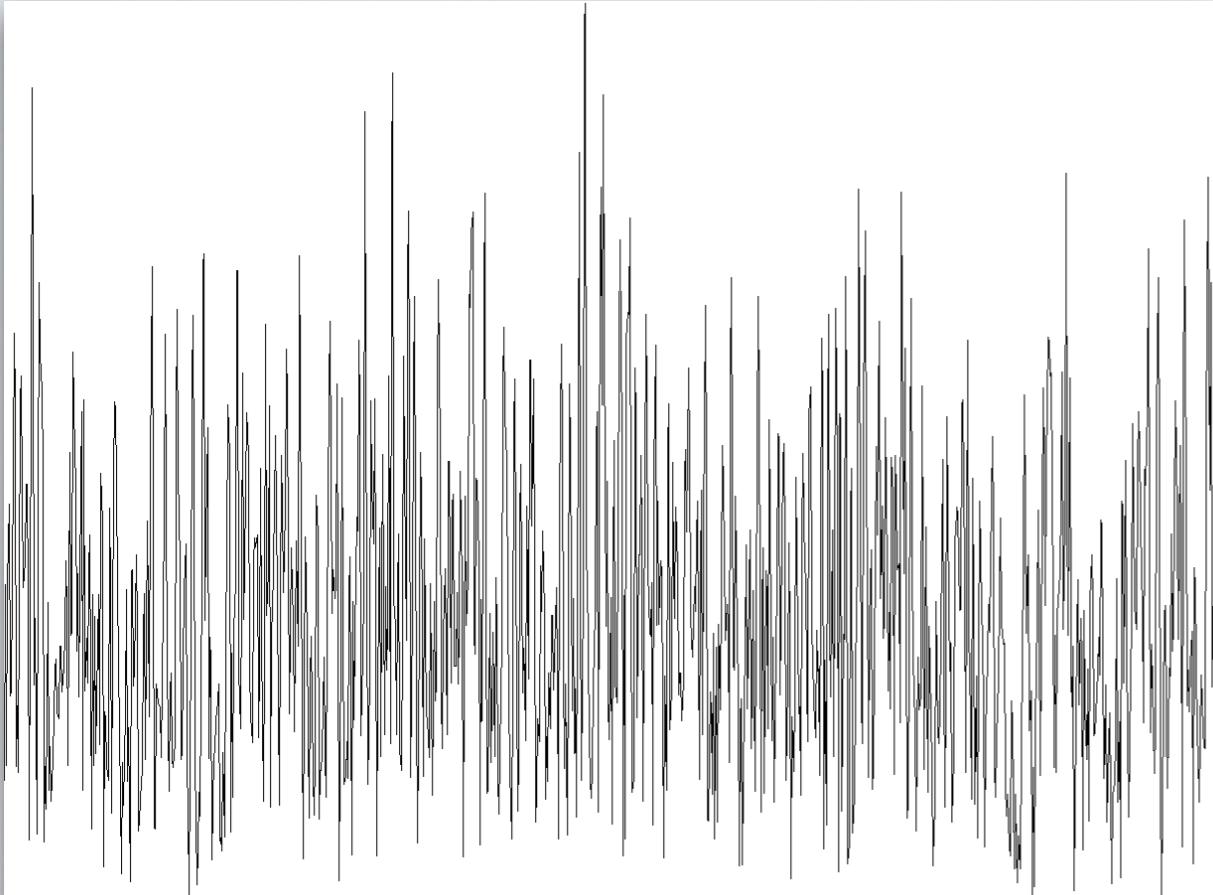
engineering the appearance of randomness

USES OF RANDOMNESS



- Designed systems need variation
- Natural systems appear to combine structure & randomness
 - Which belies more underlying structure
 - ...that we can fake with noise.
Simulation is expensive!
- Any domain: location, color, transparency, normals...

IS RANDOM TOO RANDOM?



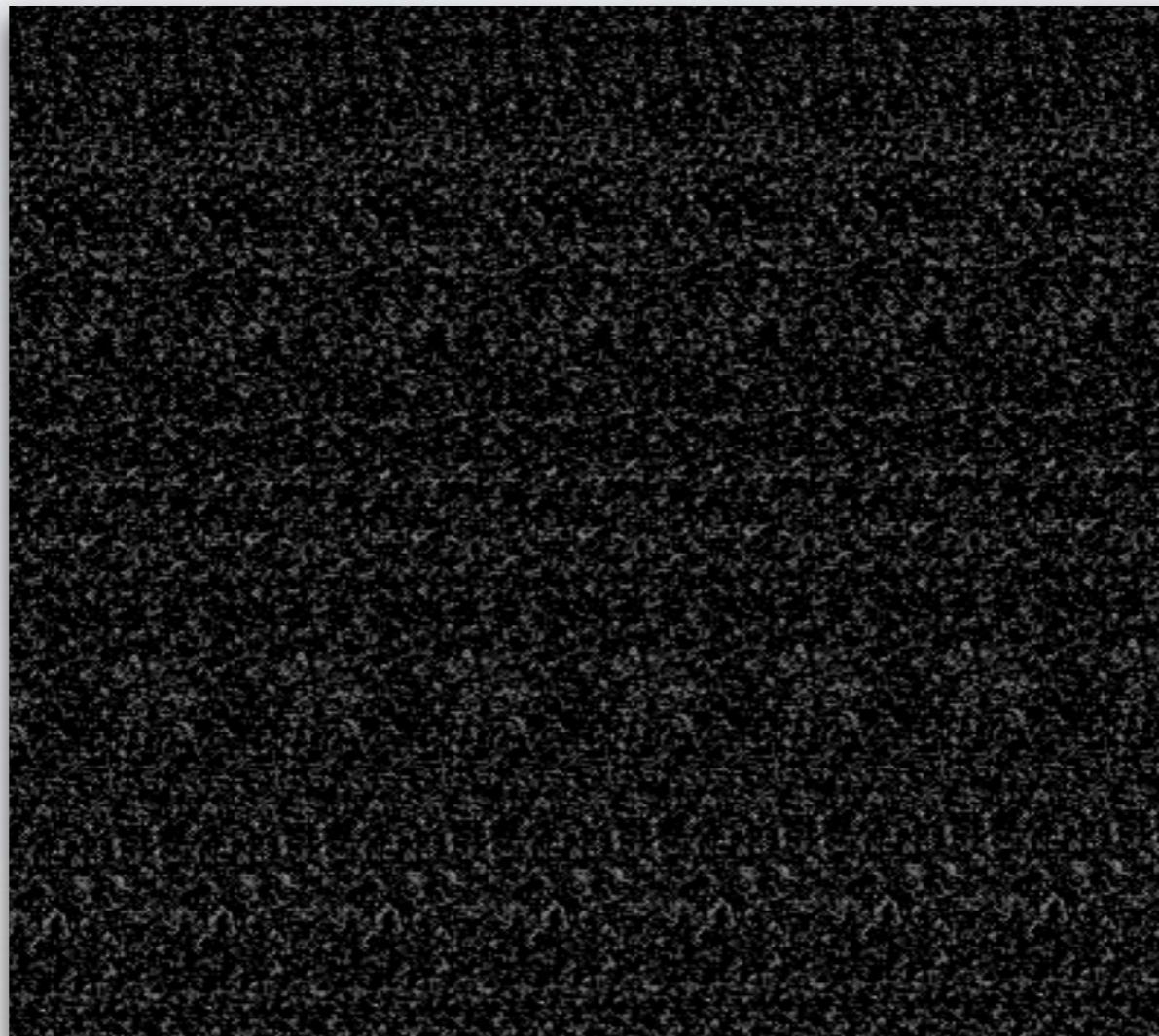
- A) True random values don't guarantee the same results each time
- B) True randomness lacks coherency — no relationship between adjacent points.

NOISE FUNCTIONS

- A noise function is a seeded random number generator
 - Given some input parameters, returns a random-looking number: $y = \text{noise}(x)$
 - Same input parameters will always produce the same output.
- Good-looking, varied output depends on choosing and manipulating noise functions.
- Many generation methods, both implicit and explicit.

PSEUDO-RANDOM NOISE

hash functions!



Using math operations, we can transform input in a that looks random... but actually repeats after a long period.

PSEUDO-RANDOM NOISE

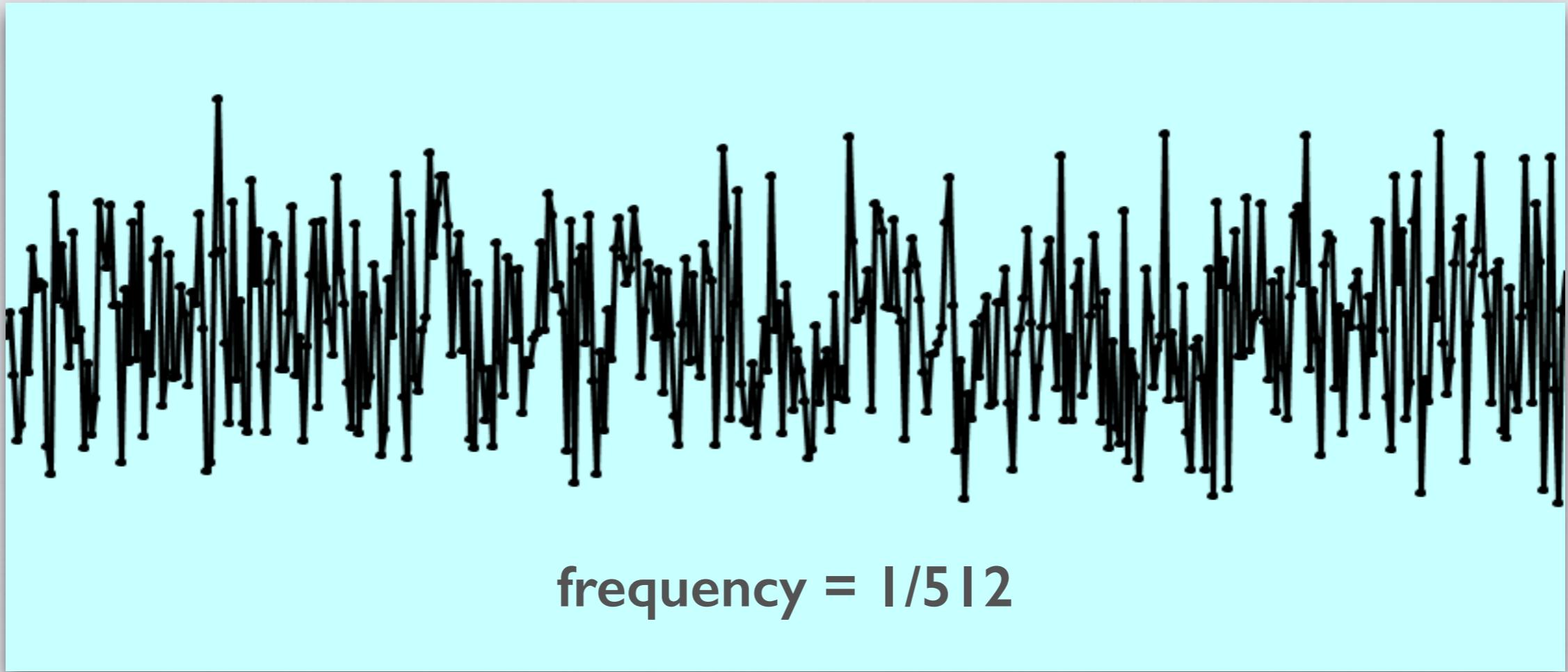
- Can be n-dimensional
- Typically uses bitwise operations and/or large prime numbers
- Warning: know your output range! Usually assume $[-1, 1]$.
- This is the corner-stone of a lot of demos. See [IQ's blog](#) for optimization tips.

Examples:

```
float noise_gen1(int x) {
    x = (x << 13) ^ x;
    return (1.0 - (x * (x * x * 15731 + 789221) + 1376312589) & 7fffffff) / 10737741824.0;
}
```

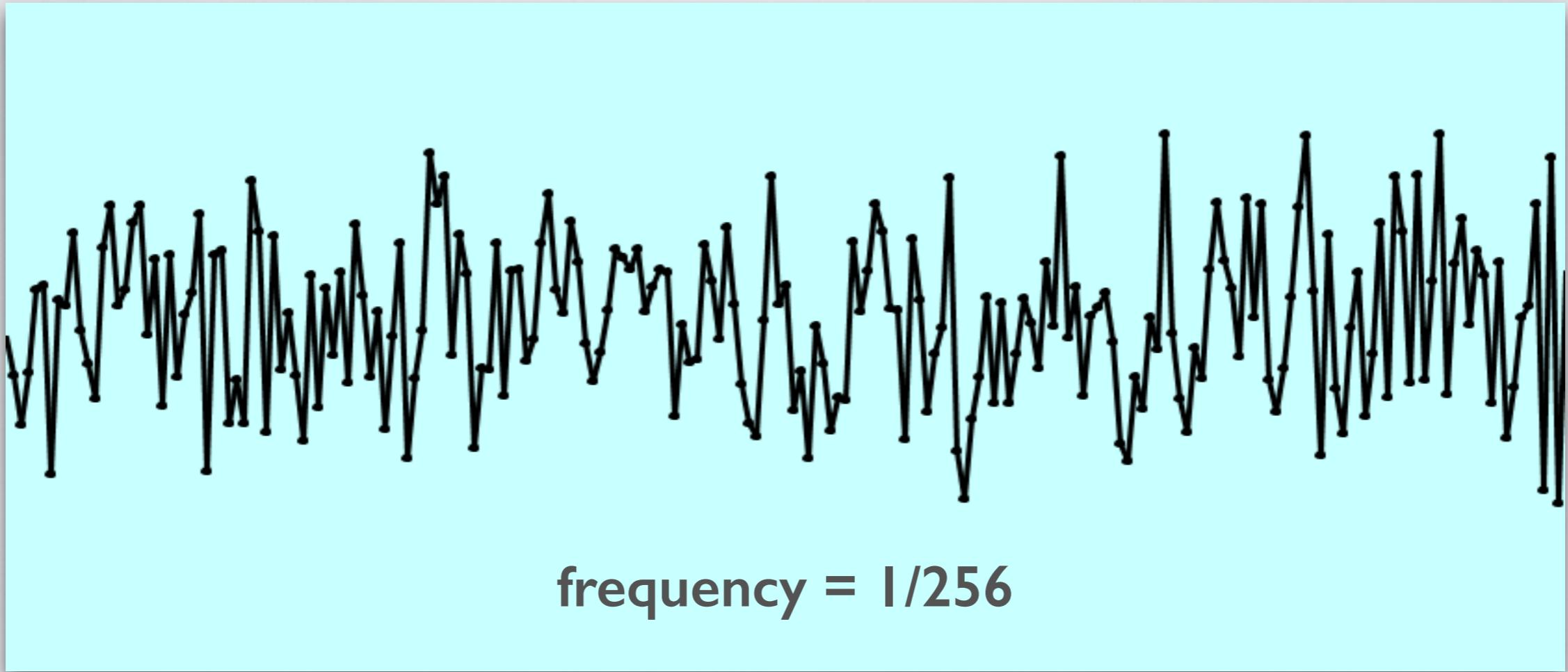
```
float noise_gen2(int x, int y) {
    return fractional_component(sin(dot(vec2(x, y), vec2(12.9898, 78.233))) * 43758.5453);
}
```

FREQUENCY



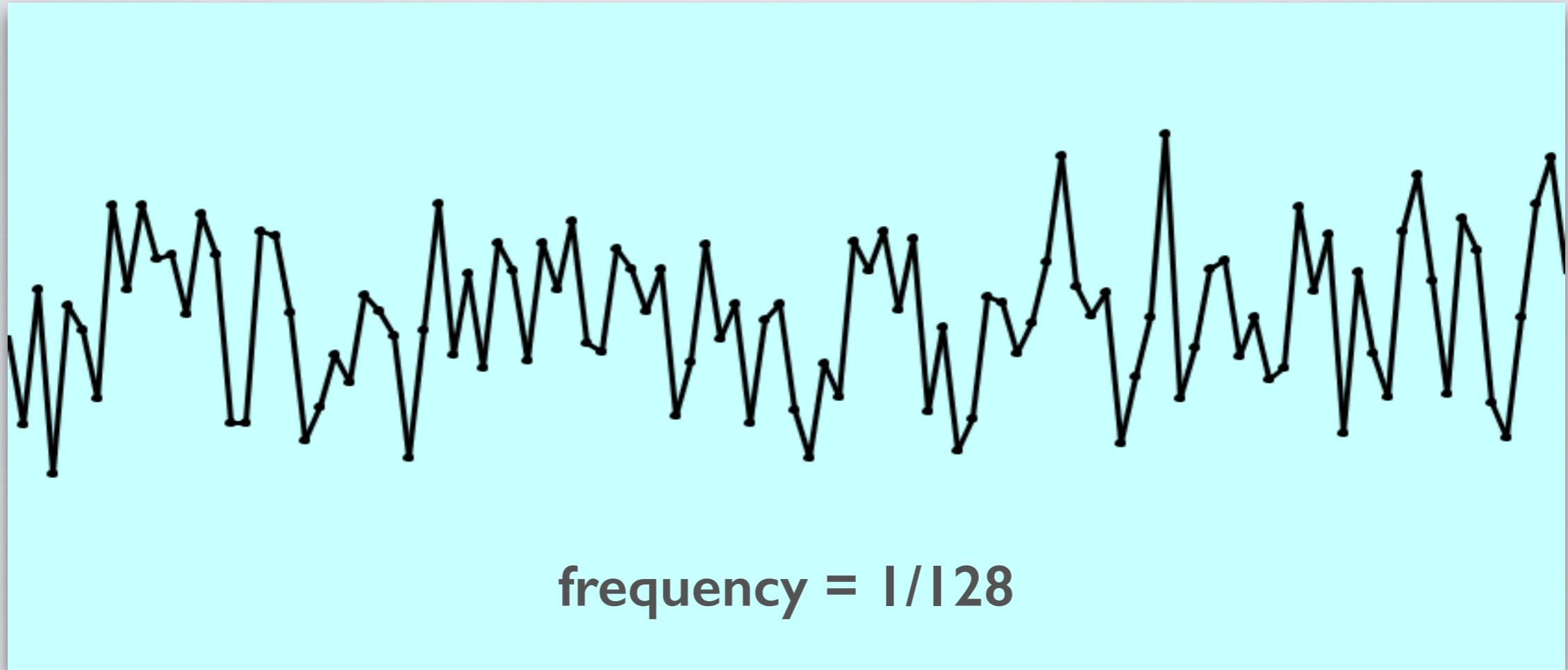
- We can sample noise at different frequencies and interpolate between points.
- Controls ‘bumpiness’
- Creates coherency — similar input creates similar output

FREQUENCY



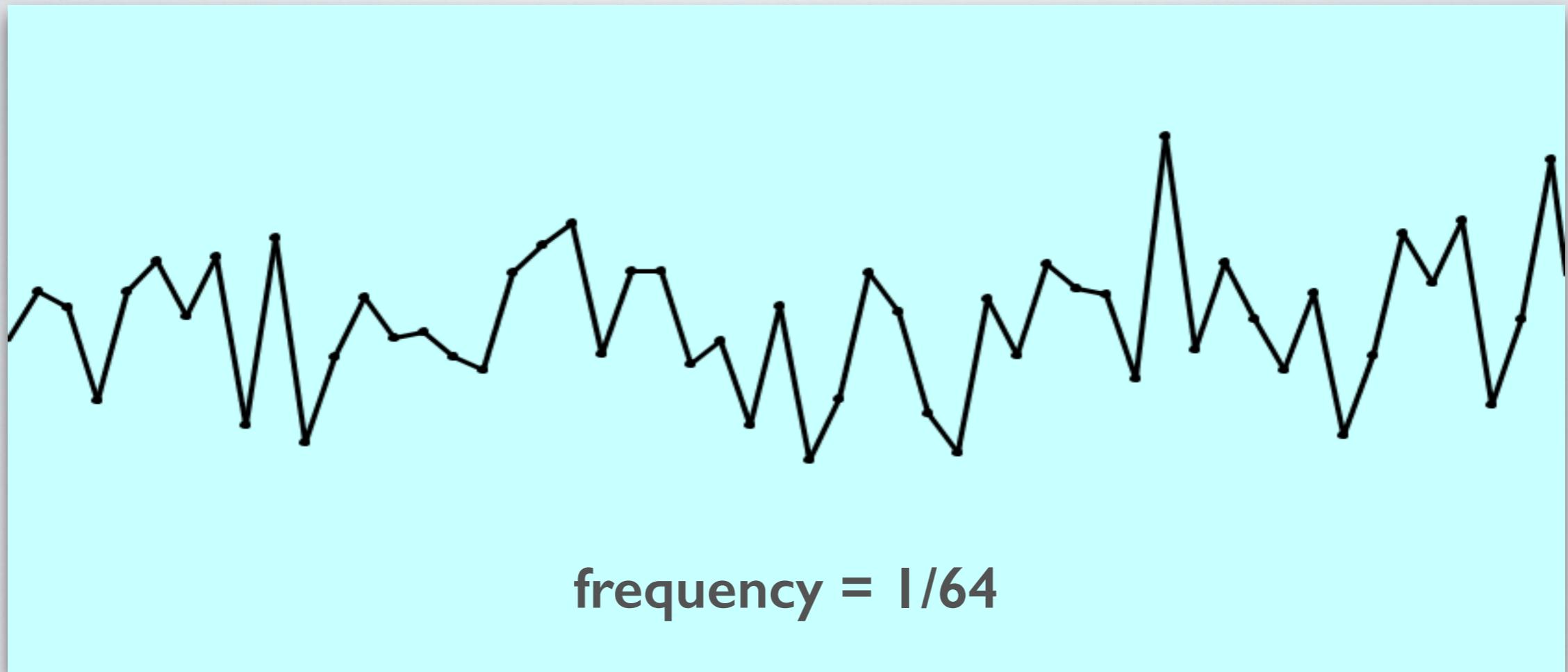
- We can sample noise at different frequencies and interpolate between points.
- Controls ‘bumpiness’
- Creates coherency — similar input creates similar output

FREQUENCY



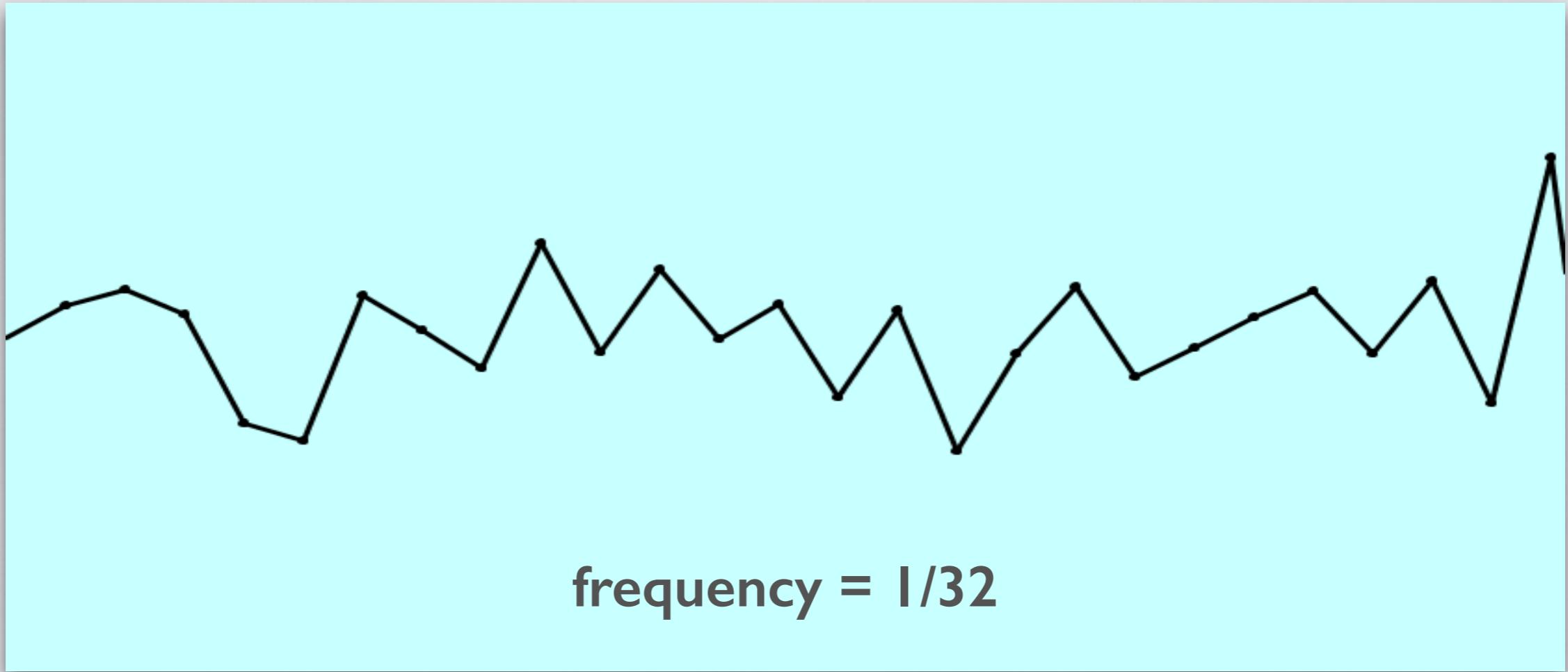
- We can sample noise at different frequencies and interpolate between points.
- Controls ‘bumpiness’
- Creates coherency — similar input creates similar output

FREQUENCY



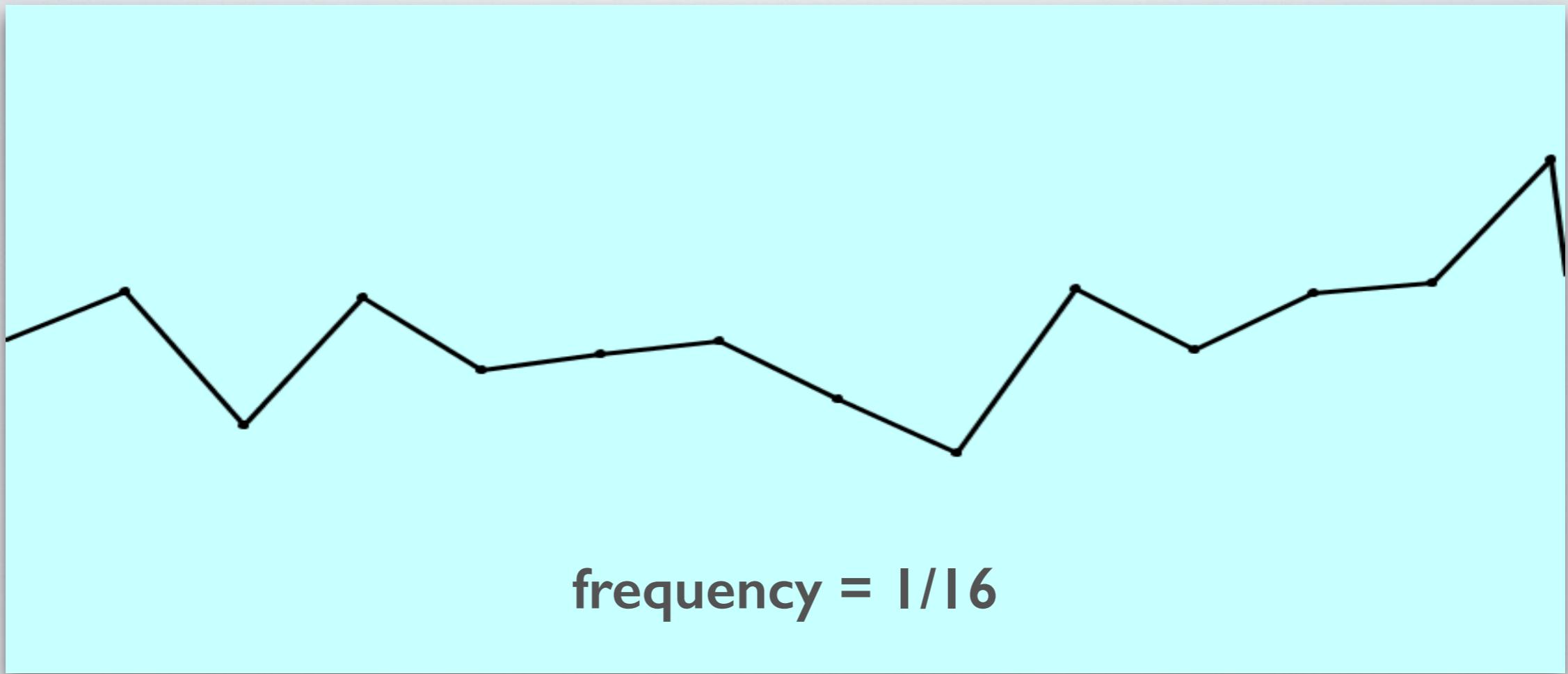
- We can sample noise at different frequencies and interpolate between points.
- Controls ‘bumpiness’
- Creates coherency — similar input creates similar output

FREQUENCY



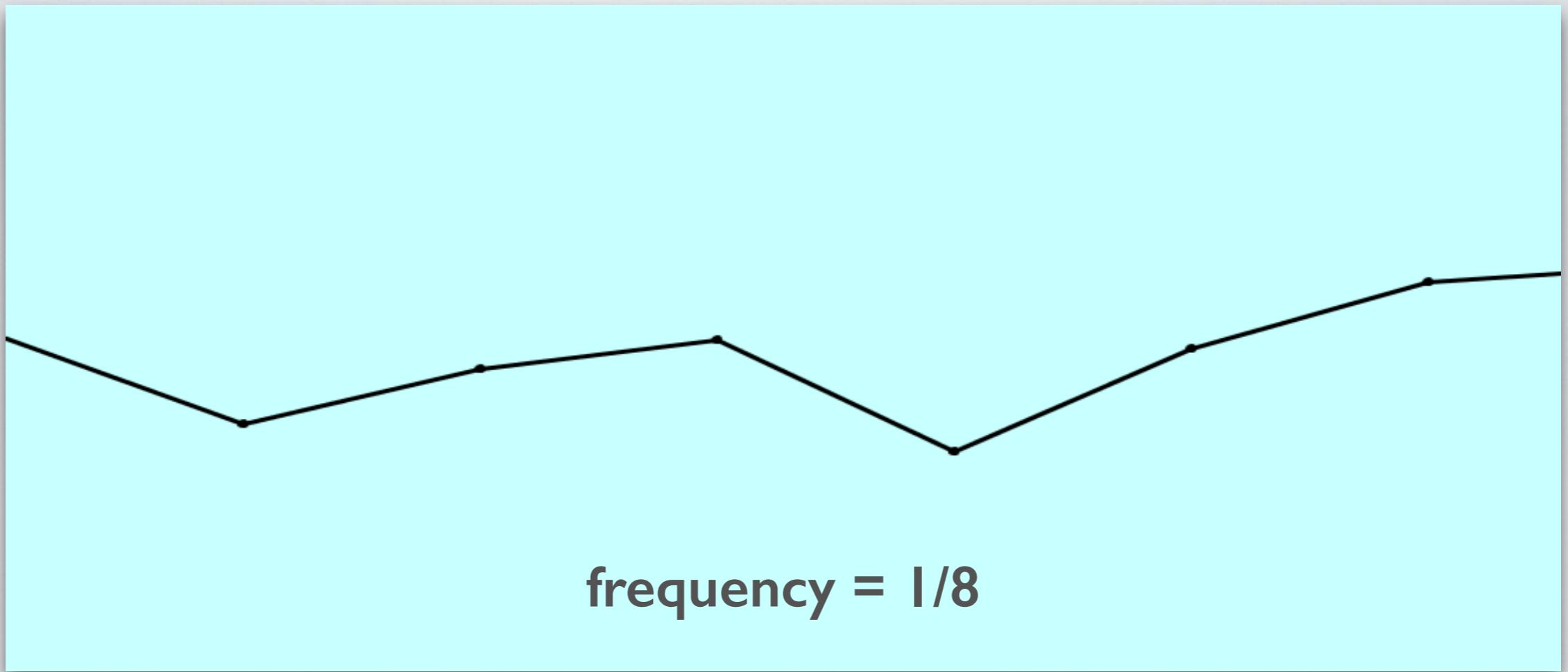
- We can sample noise at different frequencies and interpolate between points.
- Controls 'bumpiness'
- Creates coherency — similar input creates similar output

FREQUENCY



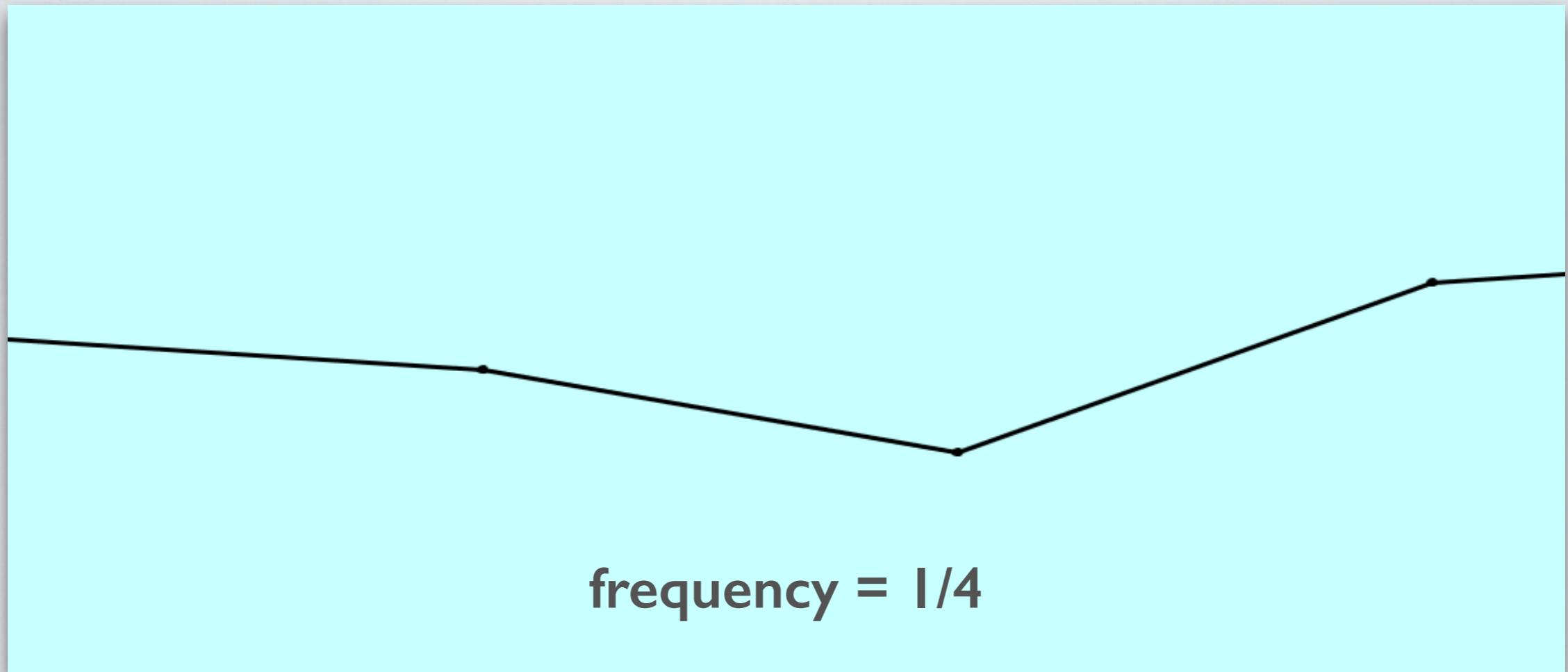
- We can sample noise at different frequencies and interpolate between points.
- Controls 'bumpiness'
- Creates coherency — similar input creates similar output

FREQUENCY



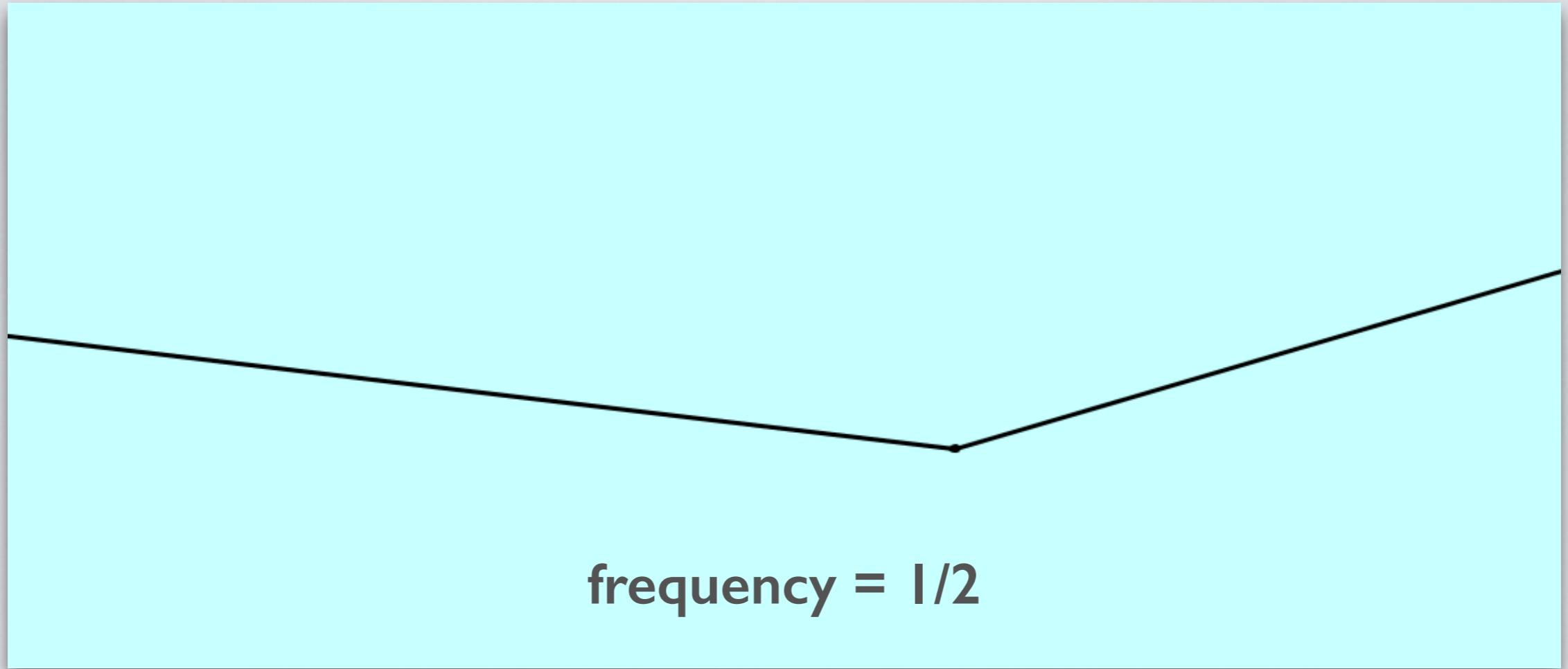
- We can sample noise at different frequencies and interpolate between points.
- Controls 'bumpiness'
- Creates coherency — similar input creates similar output

FREQUENCY



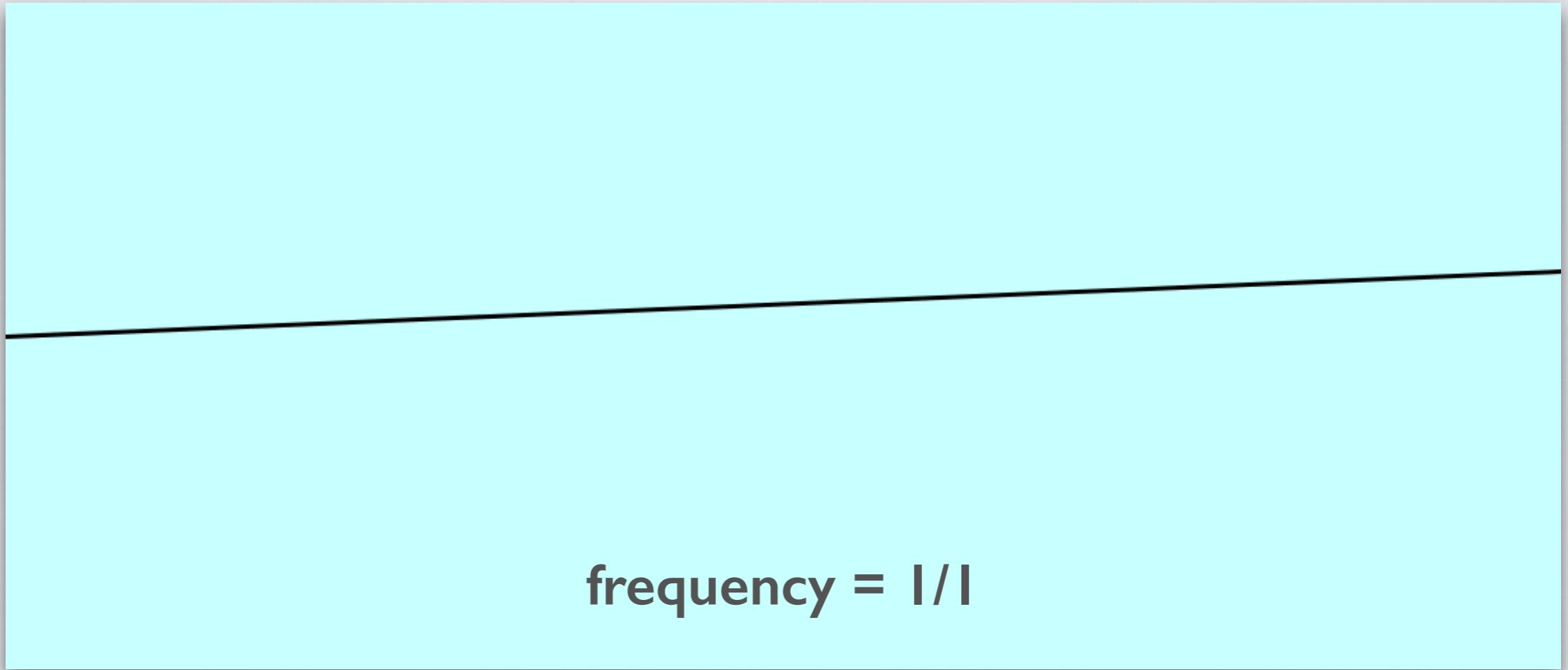
- We can sample noise at different frequencies and interpolate between points.
- Controls ‘bumpiness’
- Creates coherency — similar input creates similar output

FREQUENCY



- We can sample noise at different frequencies and interpolate between points.
- Controls ‘bumpiness’
- Creates coherency — similar input creates similar output

FREQUENCY



$$\text{frequency} = I/I$$

- We can sample noise at different frequencies and interpolate between points.
- Controls ‘bumpiness’
- Creates coherency — similar input creates similar output

PERLIN NOISE



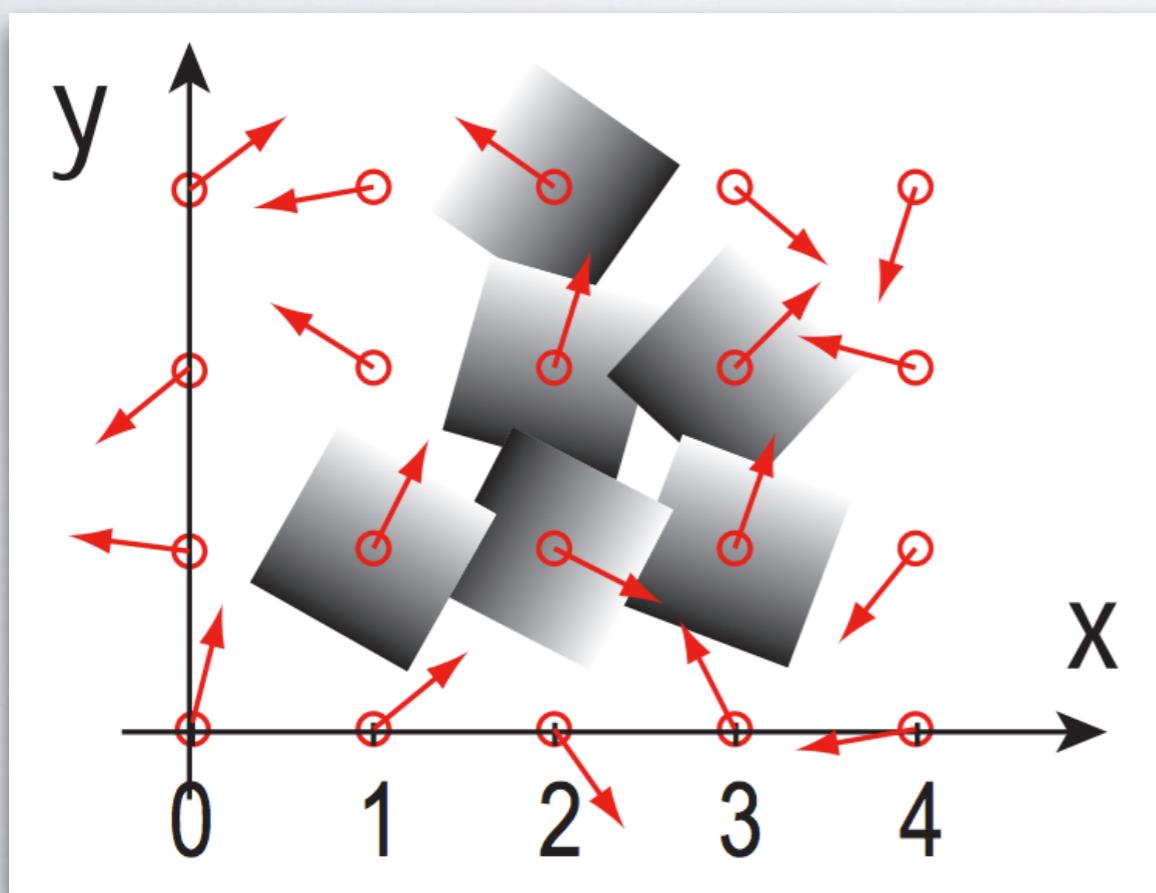
- Can we do better than value noise?
- Rather than defining a value at each sample point, we can define a gradient.
- Invented by Ken Perlin (1983), frustrated with the “machine-like” look of computer graphics
- Won an academy award for technical achievement!

Ken Perlin ([source](#))

PERLIN NOISE

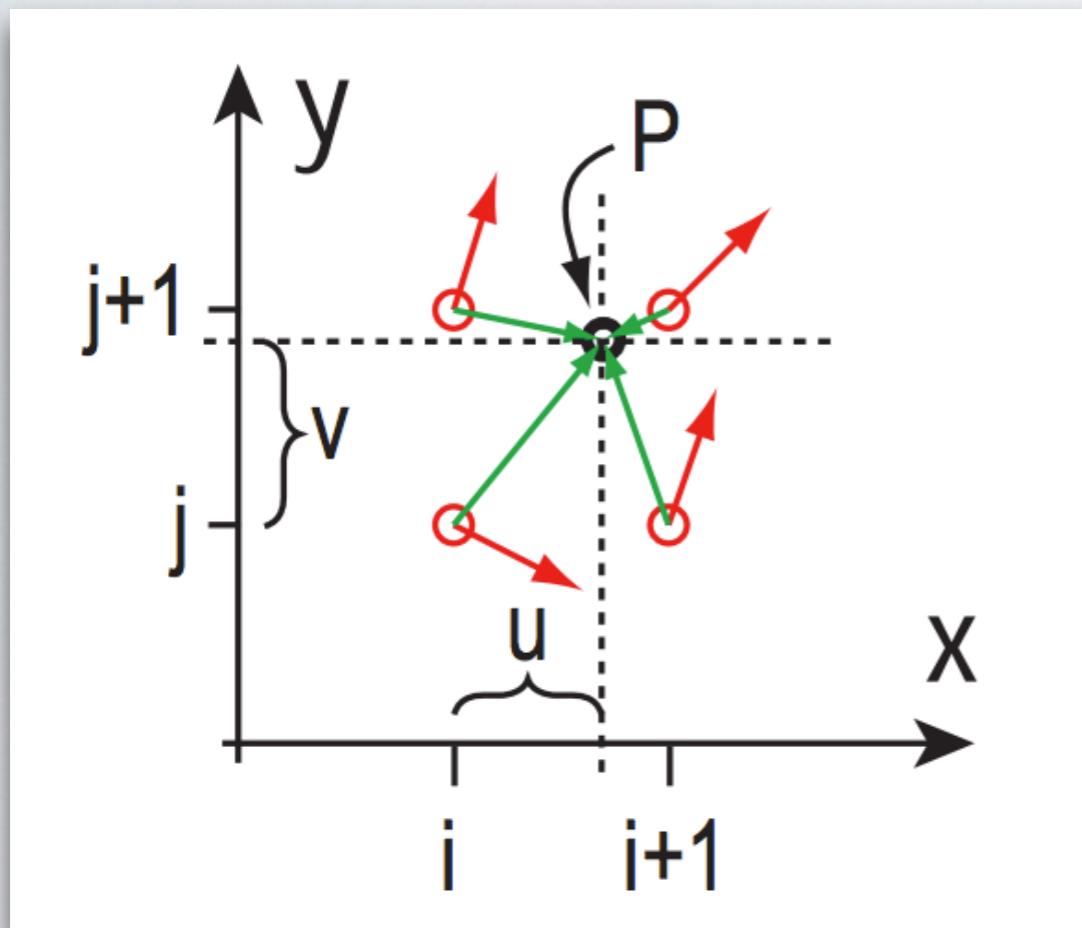


- Begin by defining a gradient, rather than a value at each integer point on the lattice.
- Any selection of gradients will do, as long as they are well-distributed.
- Perlin recommends vectors to each of the edges of a unit square / cube.



Stephan Gustavson ([source](#))

PERLIN NOISE

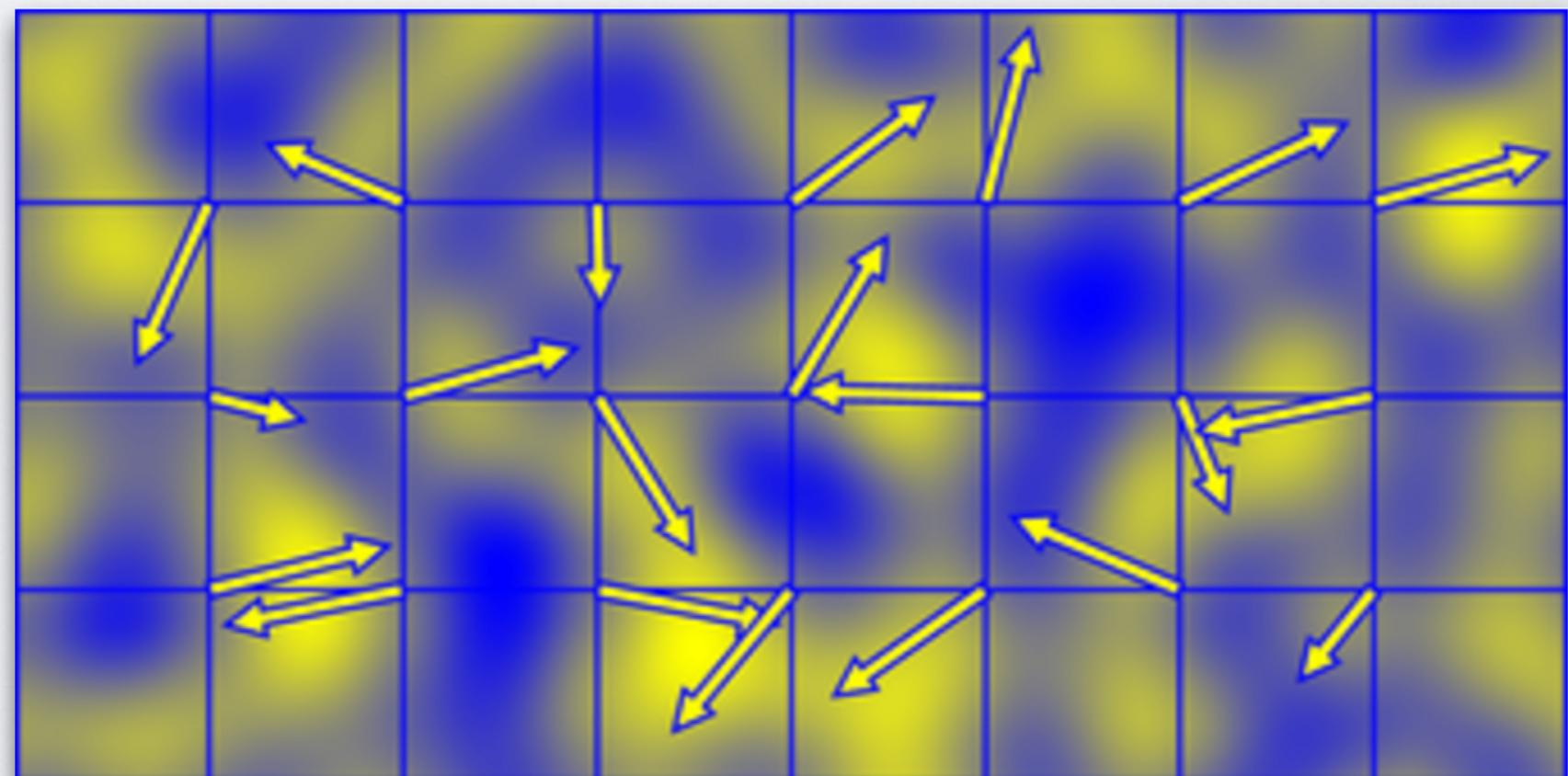
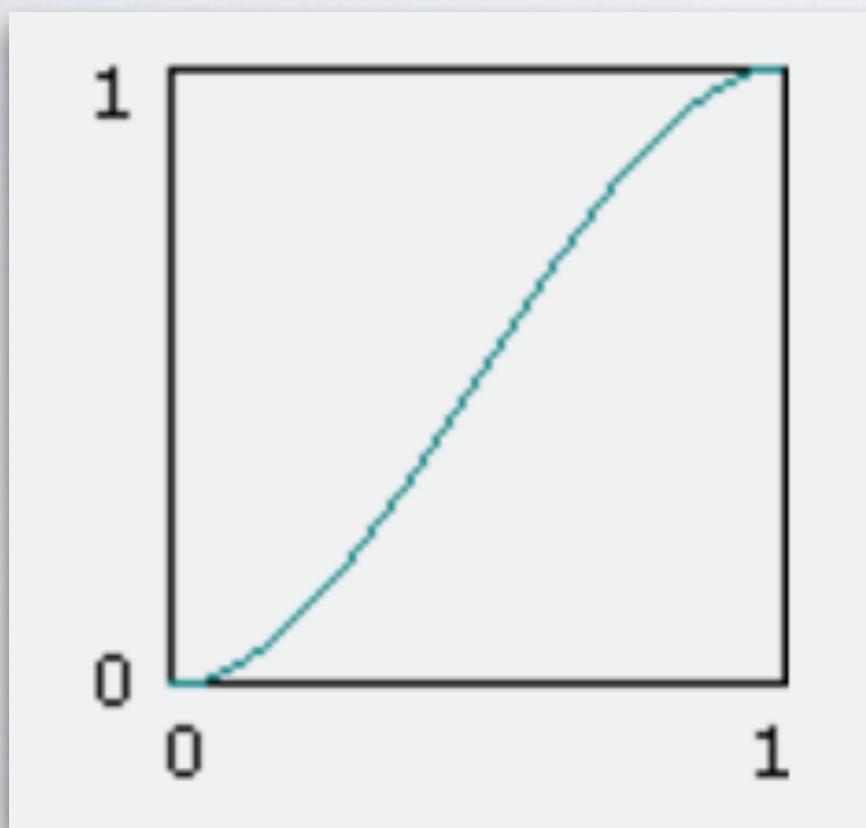


Stephan Gustavson ([source](#))

- Next calculate the **distance vectors**: $(P - V)$ for each surrounding lattice point V .
- For each surrounding lattice point, **influence_vector = dot(gradient_vector, distance_vector)**
- Interpolate between all the influence vectors to get your final value.

PERLIN NOISE

- Rather than just using linear interpolation, we can use a fade function.
- $\text{new_t} = 6t^5 - 15t^4 + 10t^3$



Boojum ([source](#))

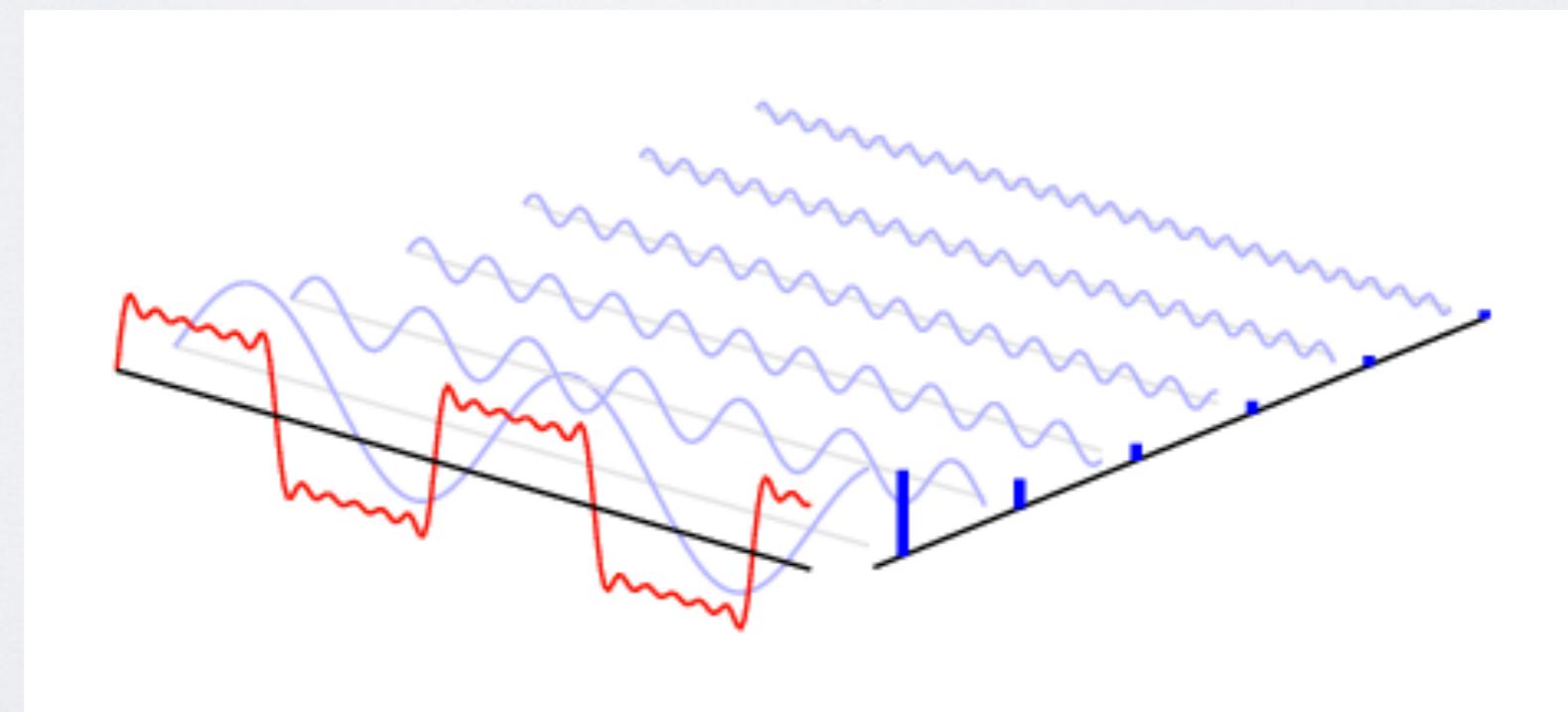
MANY, MANY METHODS

- Main types of noise generation approaches
 - Lattice noise (value or gradient)
 - Interpolates or convolutes values and/or gradients defined on an integer lattice
 - Explicit noise
 - Generate values in a preprocess step and store it. (Not strictly procedural)
 - Sparse convolution noise
 - Sum contributions from randomly positioned and weighted kernel functions.

DESCRIBING NOISE

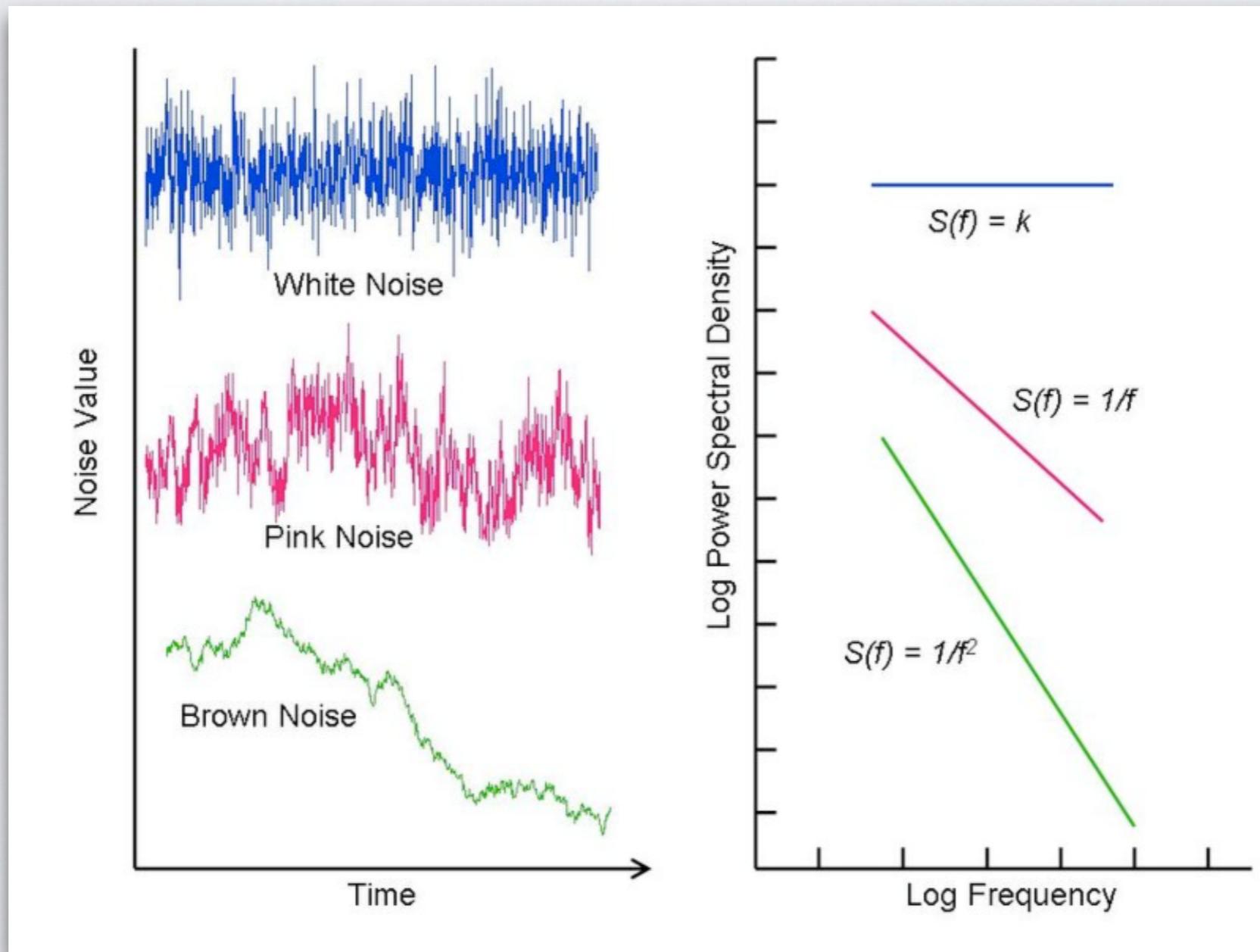
COMPONENT FREQUENCIES

- Brief signal processing aside...
- Fourier transform insight: any function, or signal, can be decomposed into simple component signals at different frequencies.
- Formally, noise functions are described by their power spectrum, the distribution of its energy among its different frequency components.



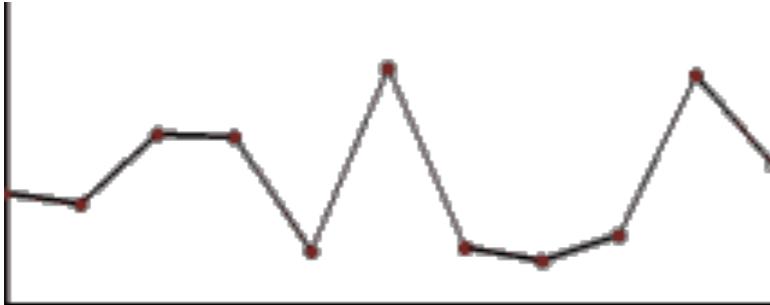
THE COLORS OF NOISE

- Color describes component frequencies



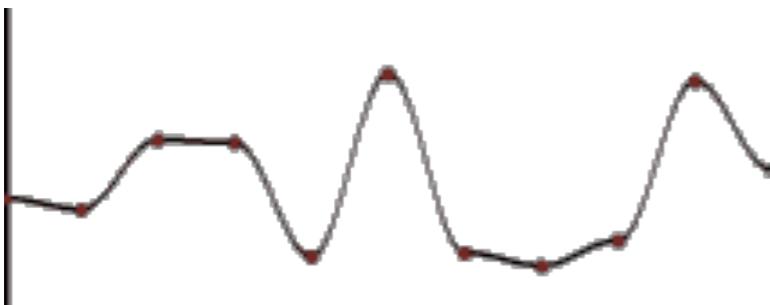
MODIFICATIONS

INTERPOLATION METHODS



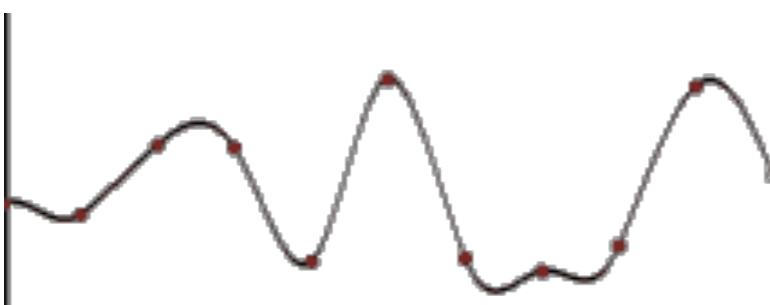
linear

```
float linear_interpolate(float a, float b, float t) {  
    return a * (1 - t) + b * t;  
}
```



cosine

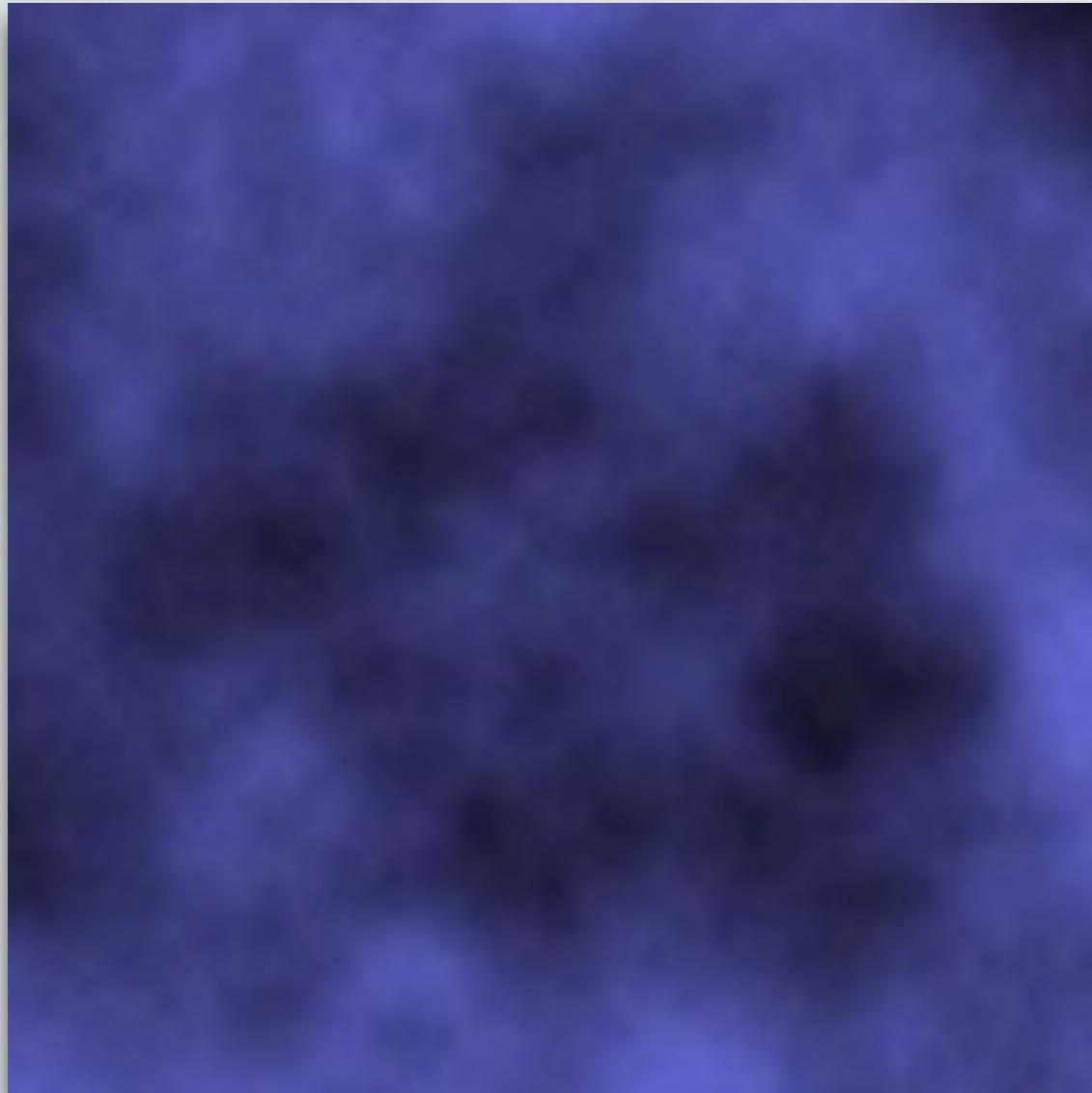
```
float cosine_interpolate(float a, float b, float t) {  
    float cos_t = (1 - cos(t * M_PI)) * 0.5f;  
    return linear_interpolate(a, b, cos_t);  
}
```



cubic

```
float cubic_interpolate(float v0, float v1, float v2, float v3, float t) {  
    float A = (v3 - v2) - (v0 - v1);  
    float B = (v0 - v1) - A;  
    float C = v2 - v0;  
    float D = v1;  
    return A * pow(t, 3) + B * pow(t, 2) + C * t + D;  
}
```

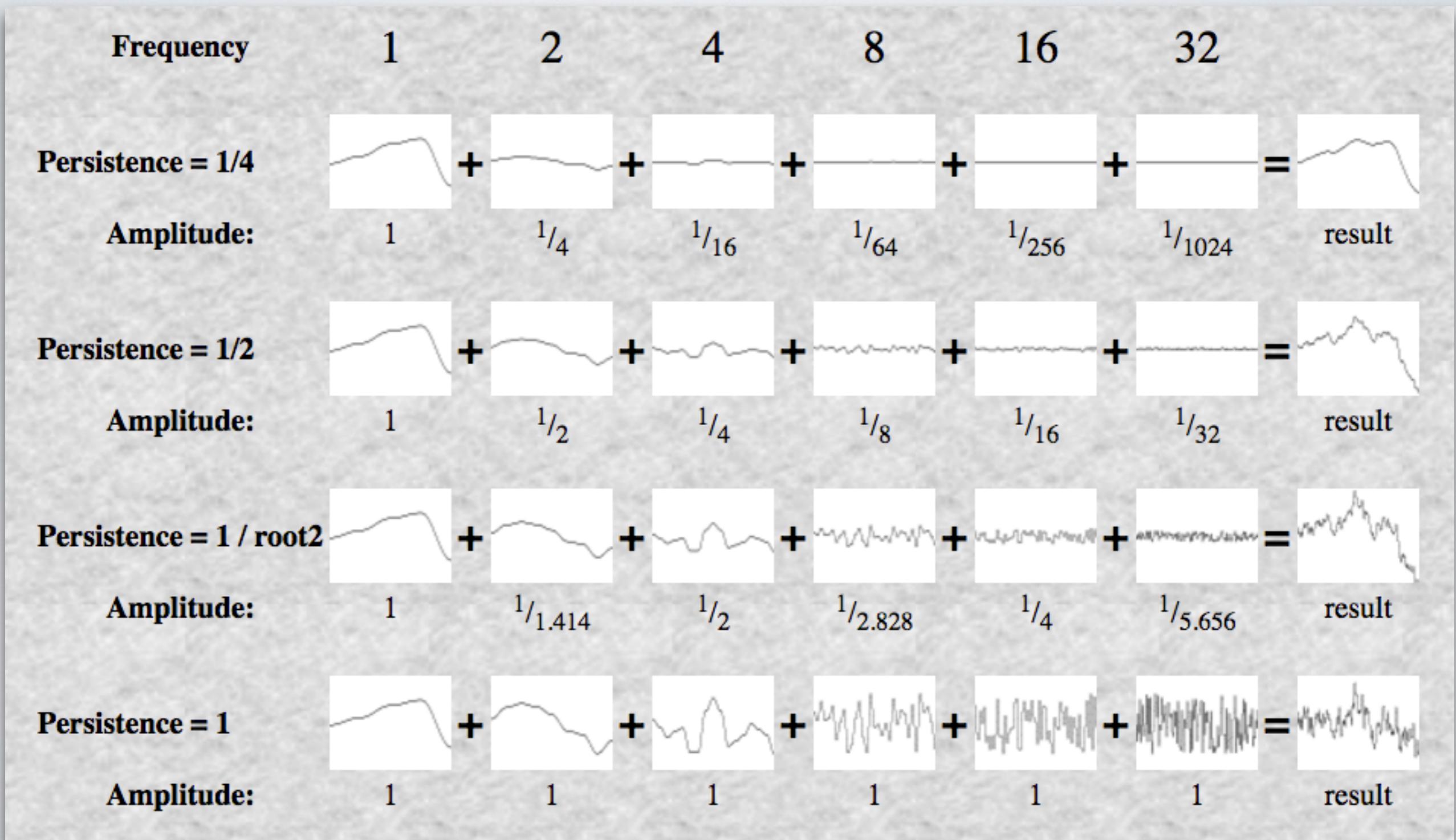
MULTI-OCTAVE NOISE (FBM)



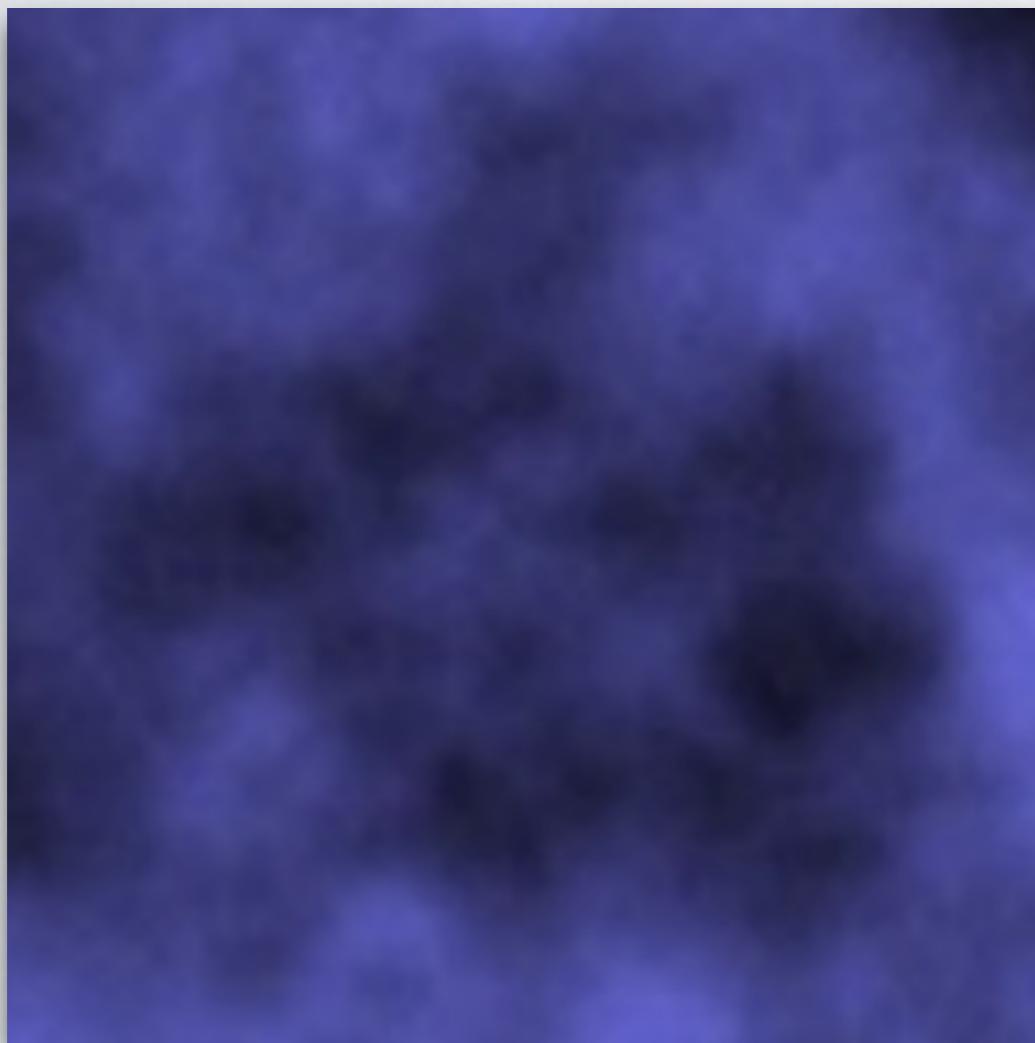
- Observation: organic patterns have multiple levels of detail
- Idea: sum multiple noise functions (“octaves”) at different frequencies and amplitudes.
- AKA “fractional brownian motion”
- Octave contributions are modulated using
 - **frequency:** sample rate
 - **persistence:** decay of amplitude as frequency increases.

Hugo Elias ([source](#))

MULTI-OCTAVE NOISE (FBM)

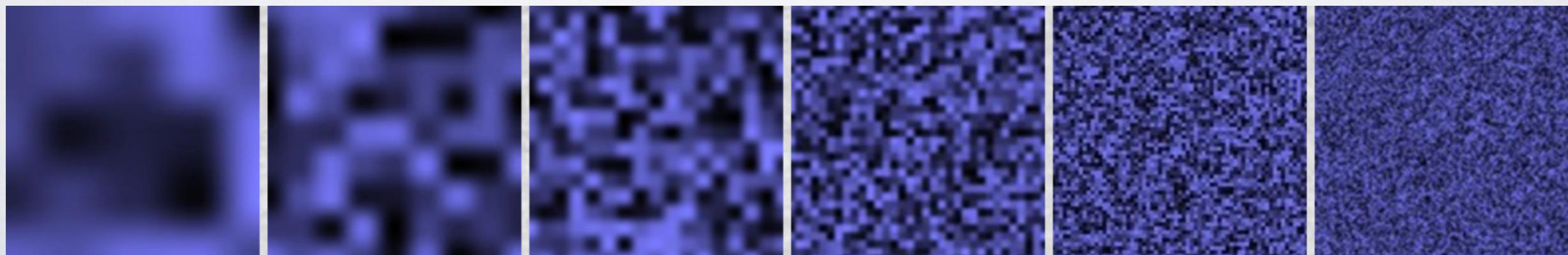


MULTI-OCTAVE NOISE (FBM)



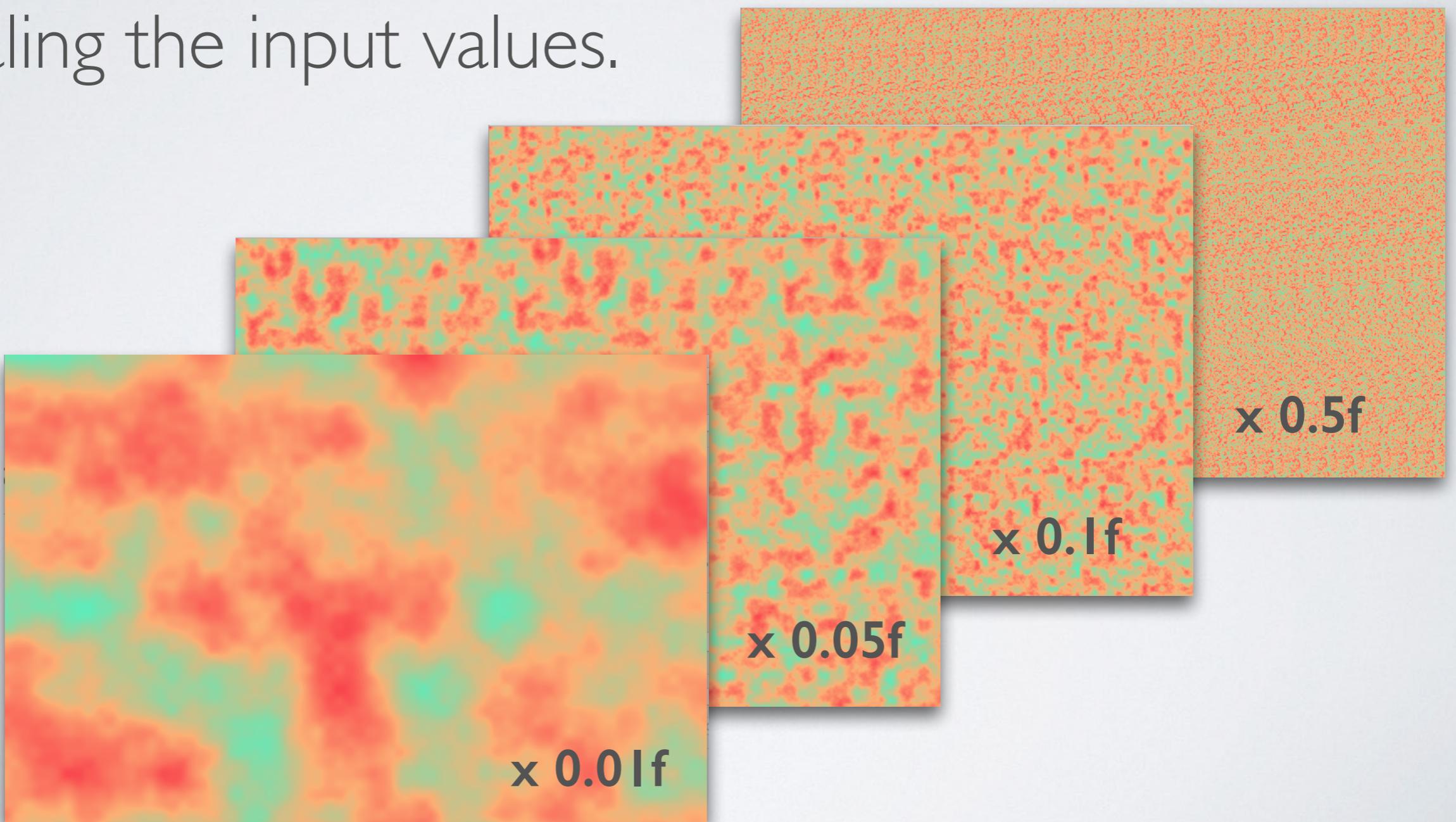
Applicable in n dimensions!

```
PerlinNoise2d(float x, float y) {  
    float total = 0;  
    float persistence = 1 / 2.0f;  
  
    // Loop for some number of octaves  
    for (int i = 0; i < N_OCTAVES; ++i) {  
        float frequency = pow(2, i);  
        float amplitude = pow(persistence, i);  
  
        // Accumulate contributions in total  
        total += sampleNoisei(x, y, frequency);  
    }  
    return total;  
}
```



“ZOOMING IN”

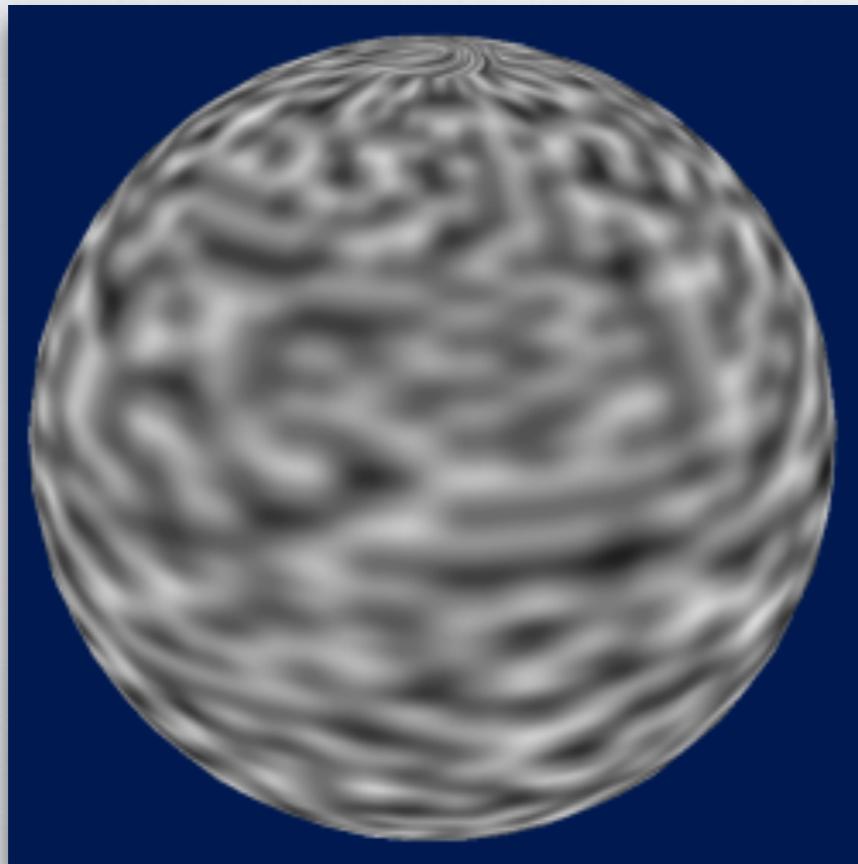
You can adjust the feature size of your noise output by scaling the input values.



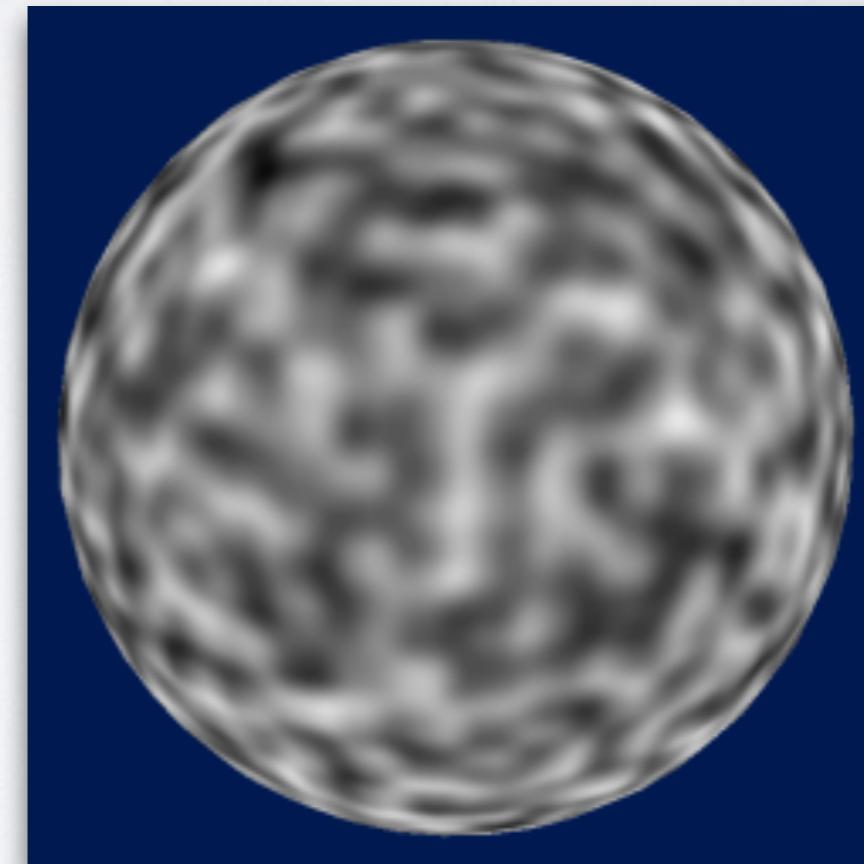
SOLID TEXTURING

Avoid mapping problems that arise from applying a 2D texture to a 3D surface.

2D



3D

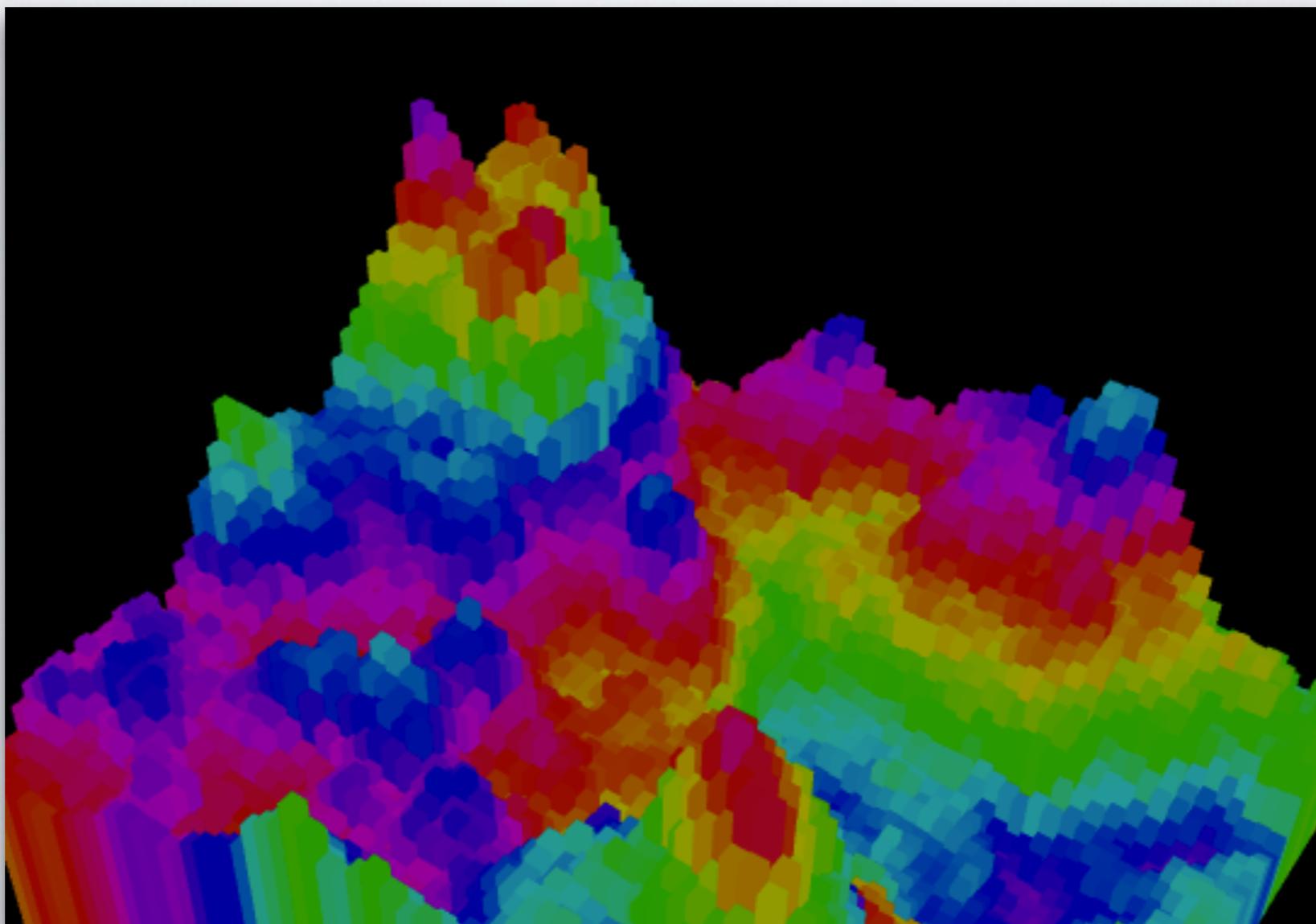


Stephan Gustavson ([source](#))

APPLICATIONS

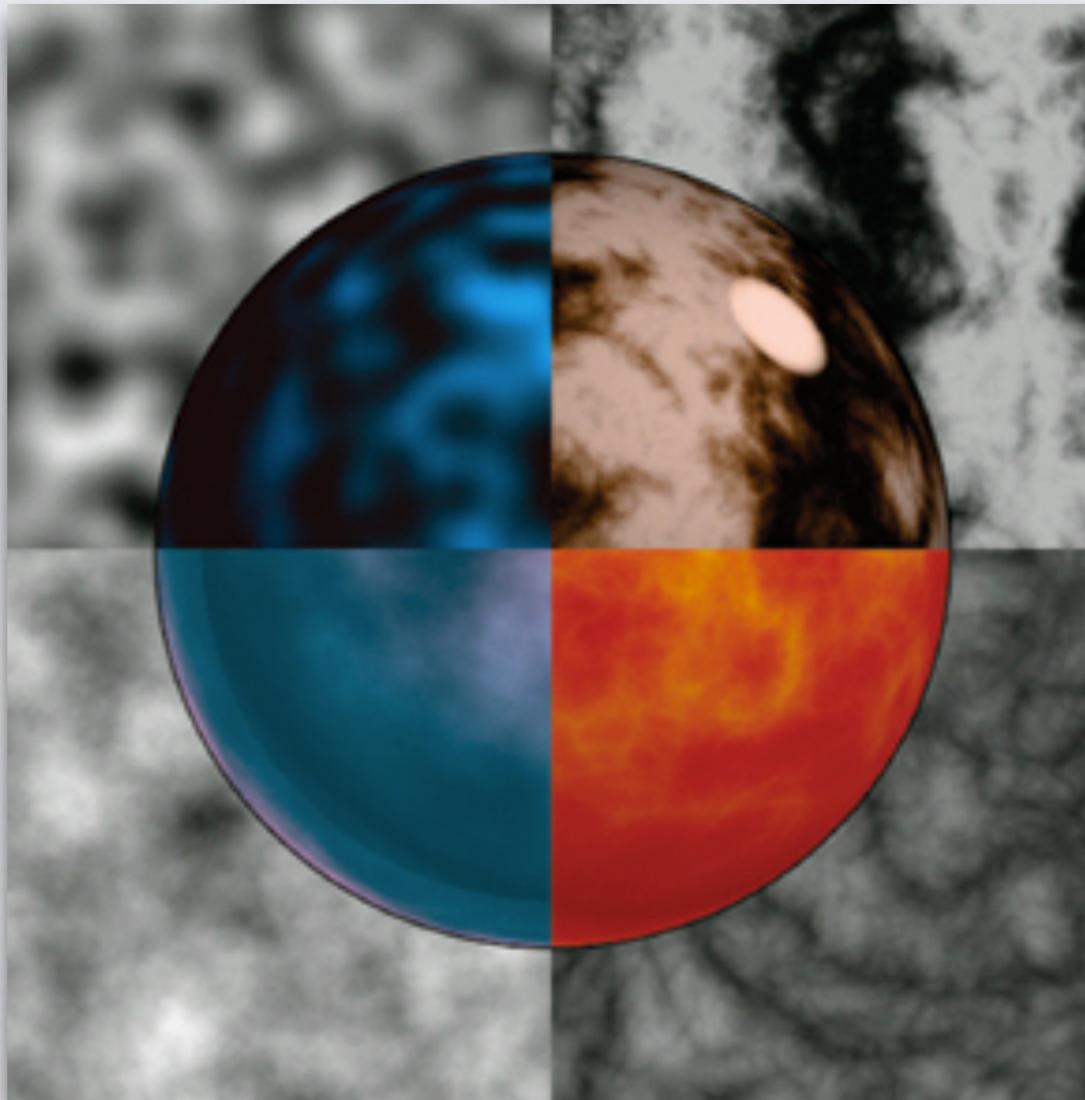
HEIGHT / DISPLACEMENT

- Use noise as a displacement along normals to make terrain.



COLOR

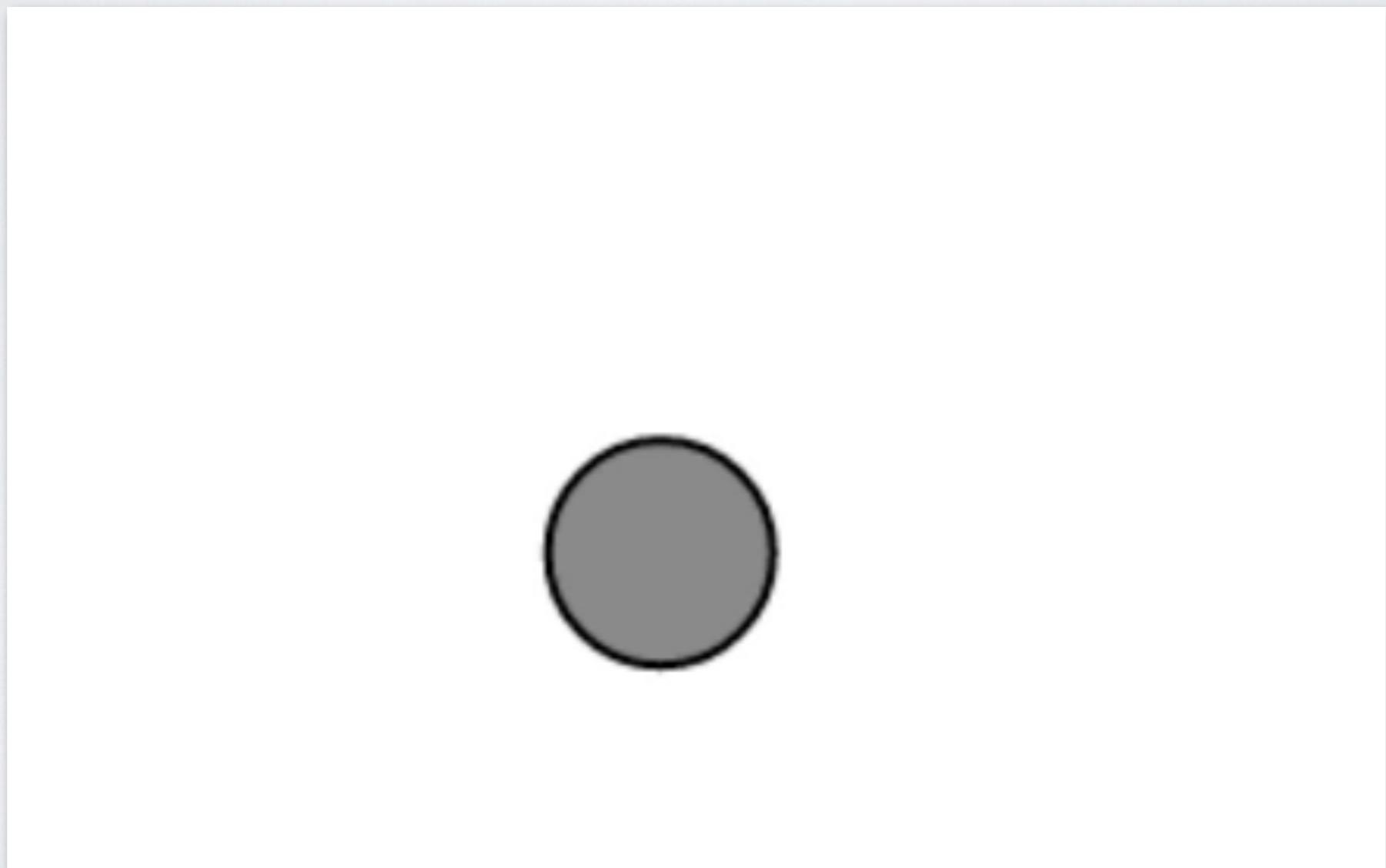
- Map noise output to color, gradient or bins.



Ken Perlin ([source](#))

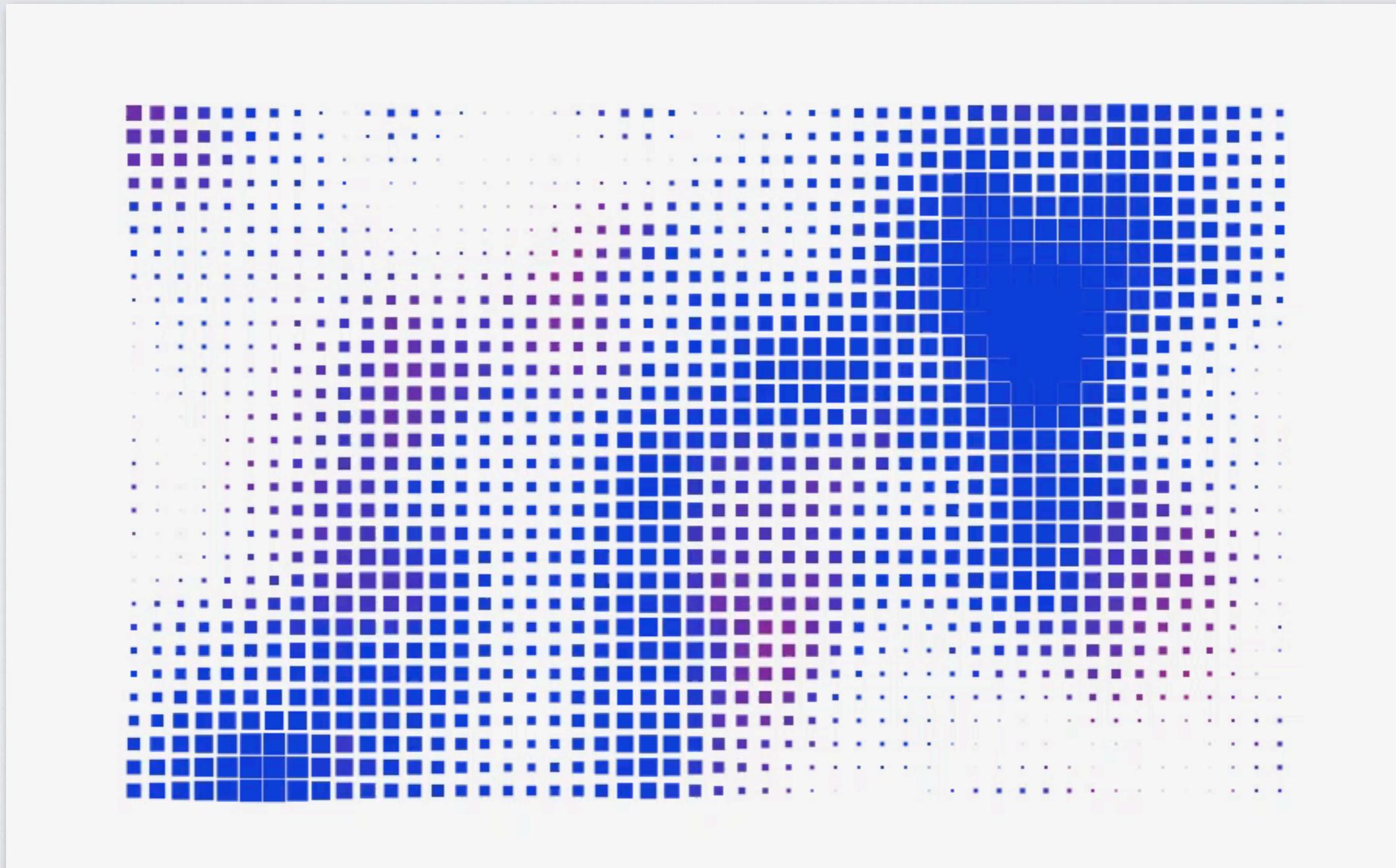
ANIMATION

- Use noise as a velocity or offset for natural movement



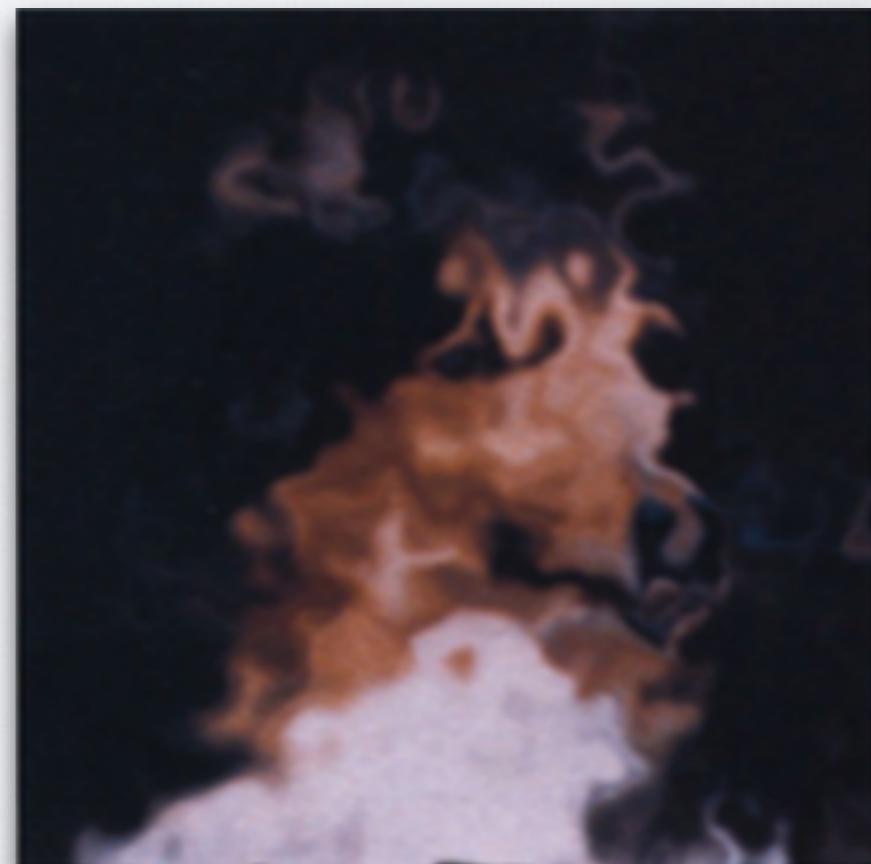
ANIMATION

- Use time as an extra parameter to create ripples



PERTURBATION

- Offset regularly-spaced coordinates (eg. image-texture reads) with noise to create a turbulent effect.



Ken Perlin ([source](#))

IN SUMMARY

- We characterize noise by its power spectrum — component frequencies
- Many generation methods
 - Directly generating pseudo-random noise
 - using a value or gradient lattice
- Ways to modify noise output
 - Sample at different frequencies
 - Add multiple noise functions together
 - Apply a smoothing function, modify interpolation
 - Scale input parameters to “zoom”

REFERENCES

- Papers
 - Original Perlin Noise
 - Improved Noise
 - Survey of Procedural Noise Techniques
- Noise explanations
 - Layman's intro to noise concepts
 - Gentle explanation of Perlin noise with reference code
 - Ken Perlin explains Perlin noise and its usage
 - Good explanations of both Perlin and Simplex noise
- Bonus
 - Some fbm value noise tips from IQ
 - Layman's introduction to Fourier transformation

ASSIGNMENT

- Create an animated cloud using multi-octave noise.
- Displace a sphere vertices along normals using noise.
- Also apply noise based color patterns.
- All computation should be done in the shader!



Ken Hermann