Fractal terrain generation

Statisticaly self-similar fractal

but also

Application of noise functions

Fractal terrain generation

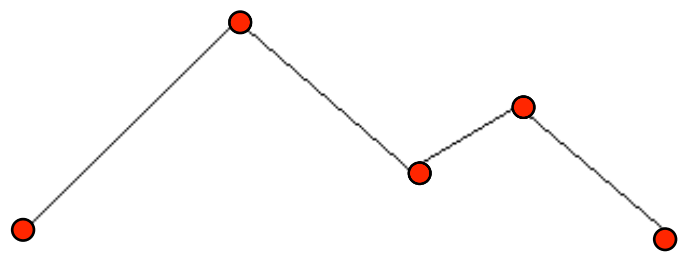
- Midpoint displacement/heightfield refinement

- Frequency space filtered noise

- Band-limited noise functions (Perlin noise)

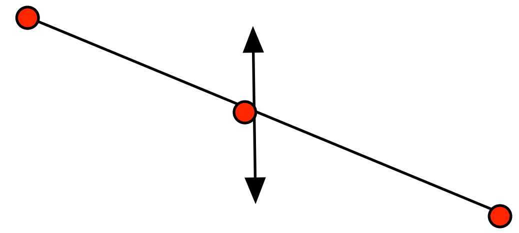
Random midpoint-displacement

Good for fractal terrain generation



Initiator

Desired rough overall whape



Generator

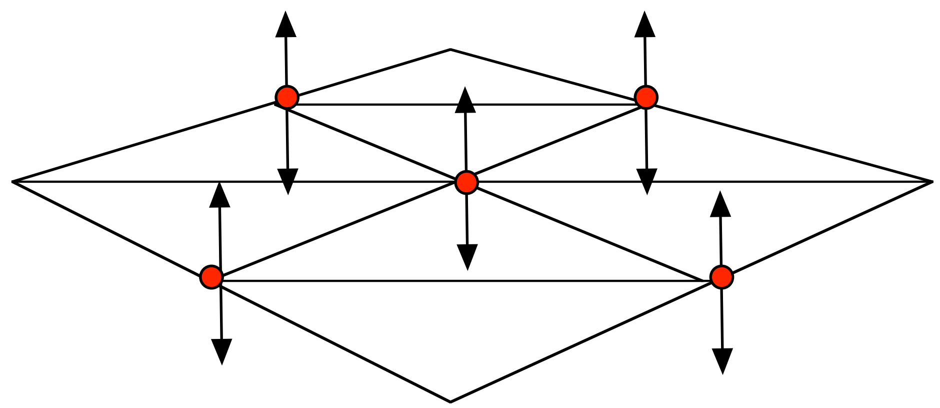
Find midpoint, displace along y only

7 iterations

Fractal terrain generation in 3D

Split a square to four

Displace midpoints of each side and middle



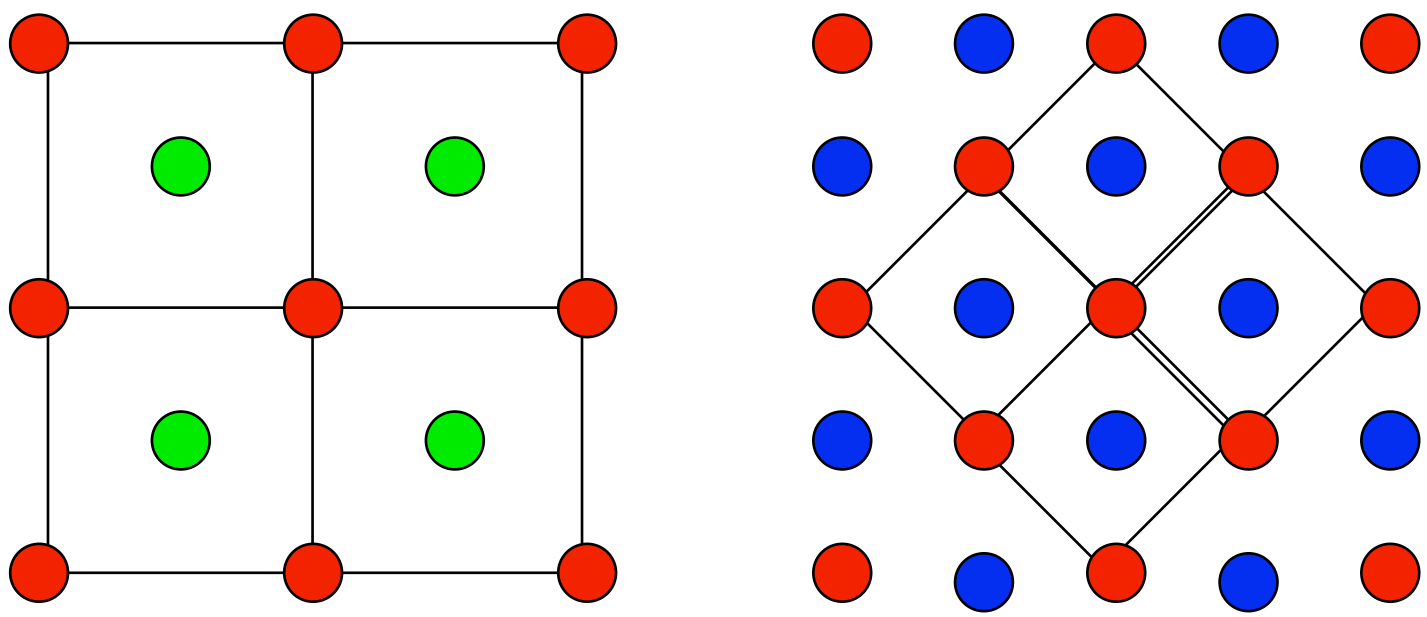
Middle point can be independent or calculated from the others

Edge points must match neighbor patches

Diamond-square algorithm

1) Midpoint from corners

2) Edge from coners and midpoints



Repeat to desired resolution

Diamond-square algorithm

Random offset at each stage

Proportional to size of the side of the square

=> Scale down by sqrt(2) for each phase!

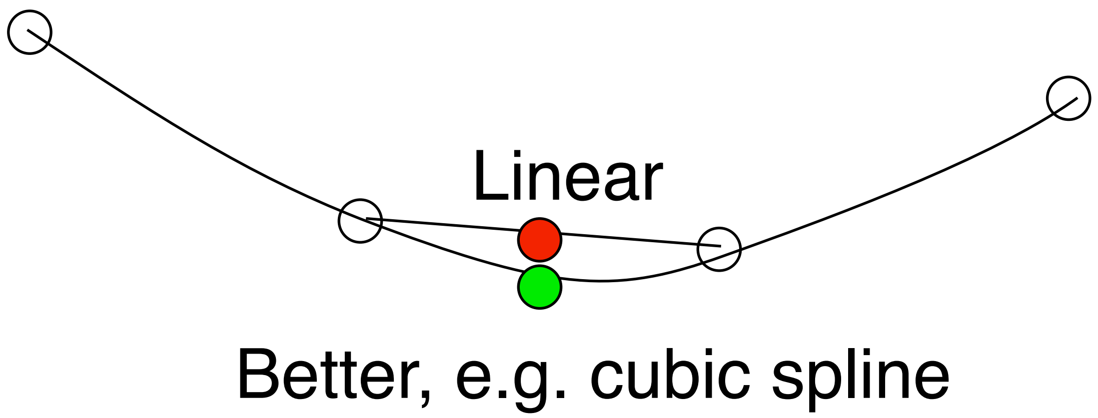
(Not by 2 for every two phases! Popular misconception!)

Diamond-square algorithm filtering

Important feature! We are *reconstructing a signal* from samples! (And then add HF detail.)

Simple and fast: Averaging (linear interpolation)

Better: Higher precision filter from larger neighborhood. Usual signal processing rules apply! Use a 4x4 neighborhood.



”Heightfield approach”

”Square-square” algorithm

Terrain level k is array of resolution 2^k x 2^k

The next level has 4x the resolution

Generate new 2x2 block from one, or filter over a small neighborhood

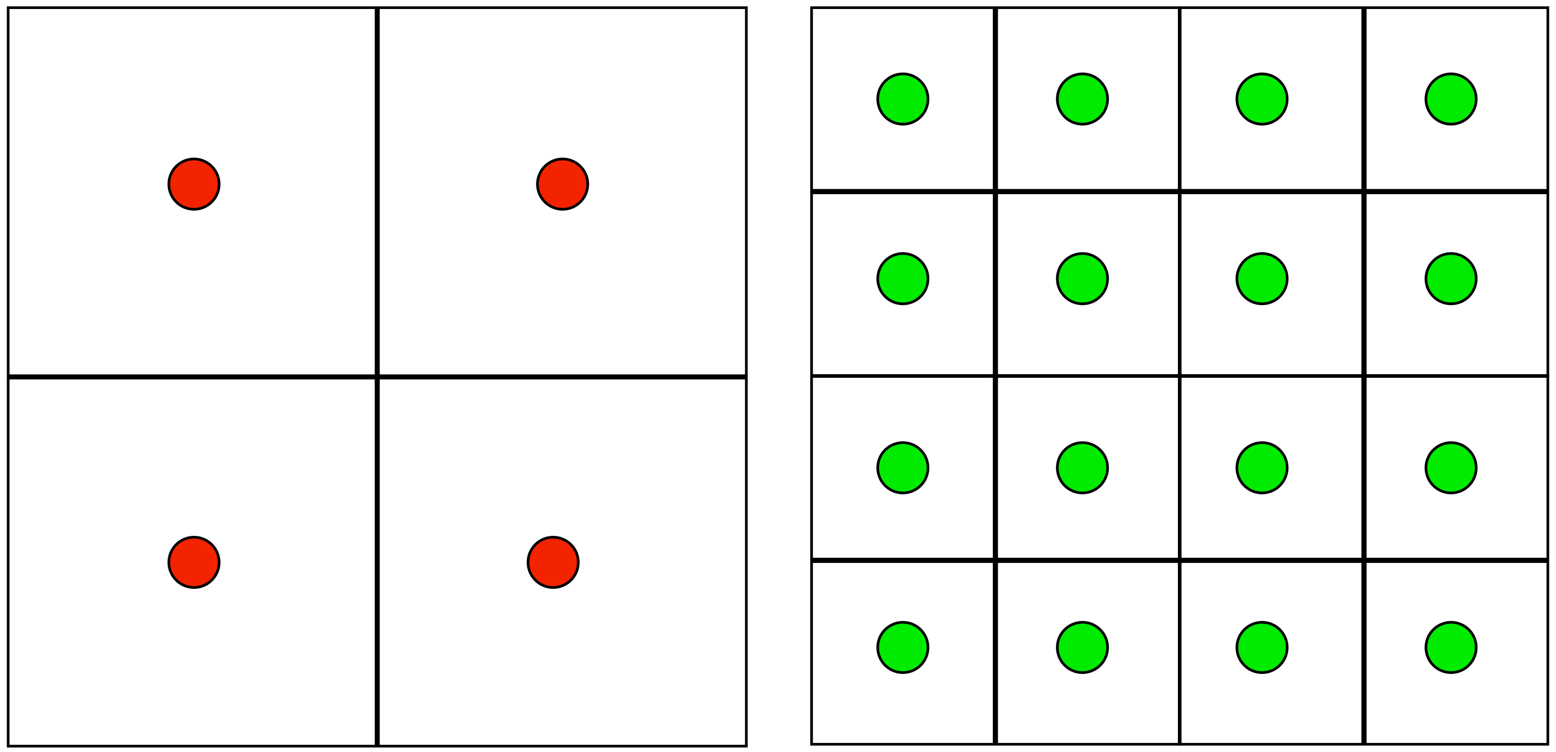
Add random offset to all values

Offset should be smaller for higher k

=> magnitude of frequency components inverse proportional to frequency!

Square square

Image upsampling + add noise



Again: Linear interpolation is simple and fast, better filter gives better result

Noise functions

Fractals and noise functions are closely related

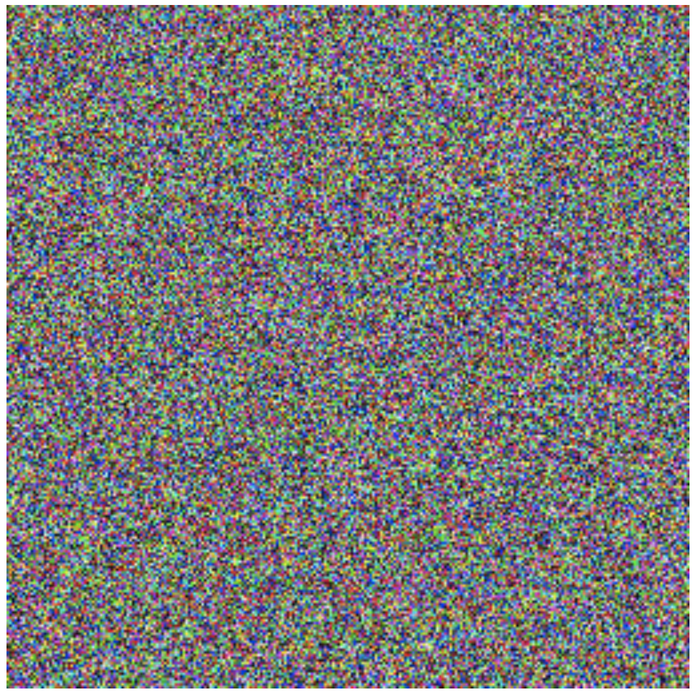
Noise can look natural... but when?

- **white noise**
- **colored noise**
- **value noise**
- **gradient noise**

White noise

Same amplitude in all frequencies

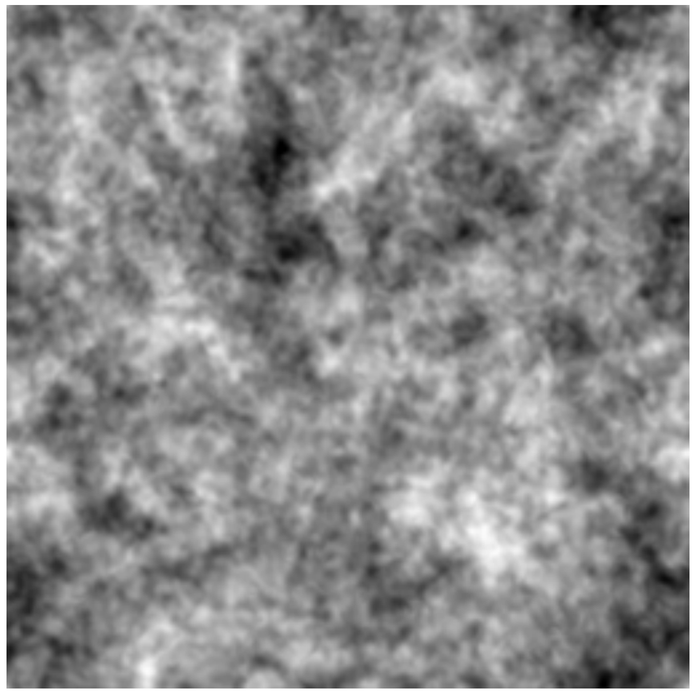
Useless as it is, but can be processed to something better.

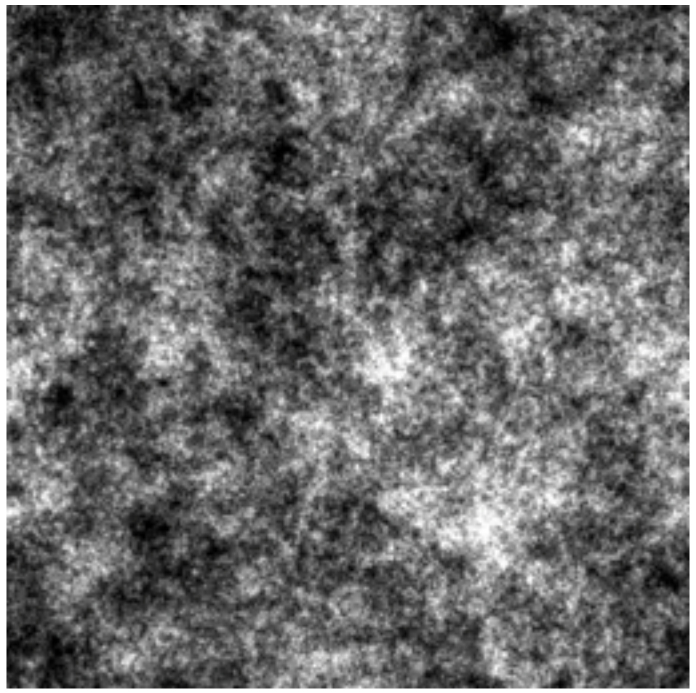


Colored noise

Amplitude varies with frequencies

With the right variation, it can look nice - natural!





Colored noise

Can be processed with filters, e.g. frequency plane functions

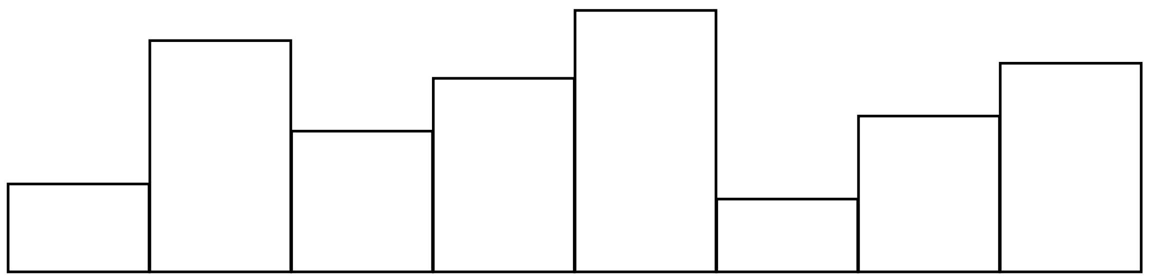
Considered too computationally heavy. (Questionable!)

Therefore other methods became popular: Simplex noise, Perlin noise.

Value noise

If you just fill your pixels with values in some range, you get *value noise* (essentially white noise).

Value noise is perfectly useful after proper filtering, possibly combining several frequency bands.



Colored noise by filtering in the frequency plane

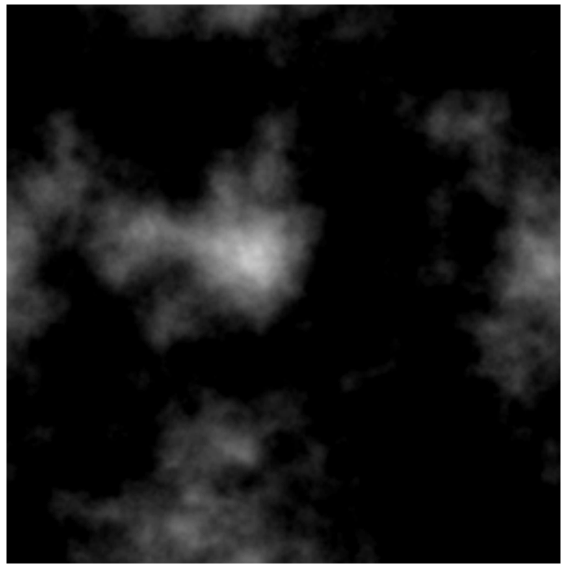
Fill frequency space (2D) with random numbers (white noise)

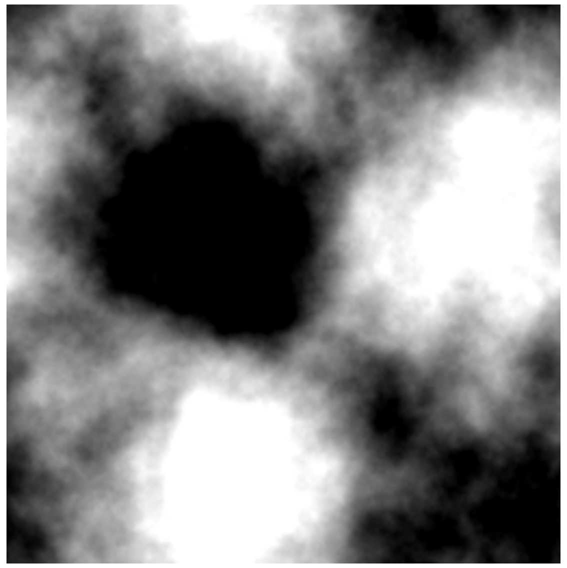
Filter by G(f) = F(f) * 1/|f|

Convert to spatial image with FFT

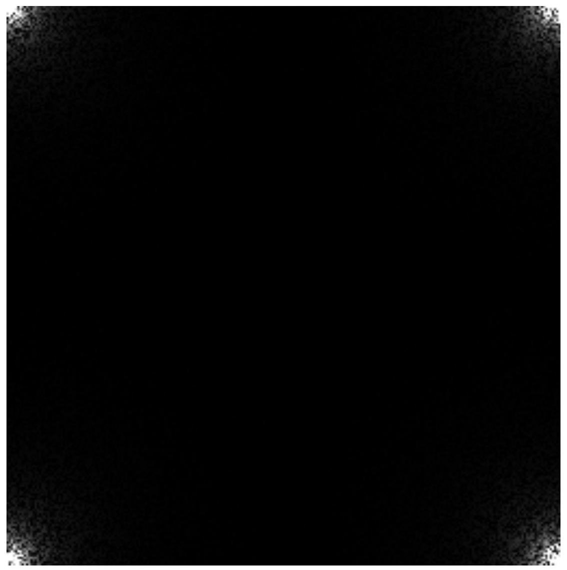
Filter white noise by 1/f

Examples



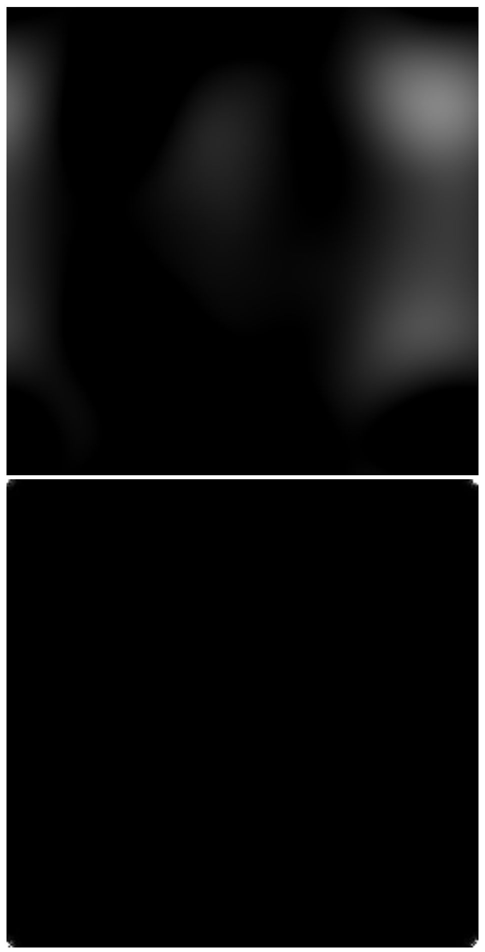


Frequency space:



Other falloffs than 1/|f|

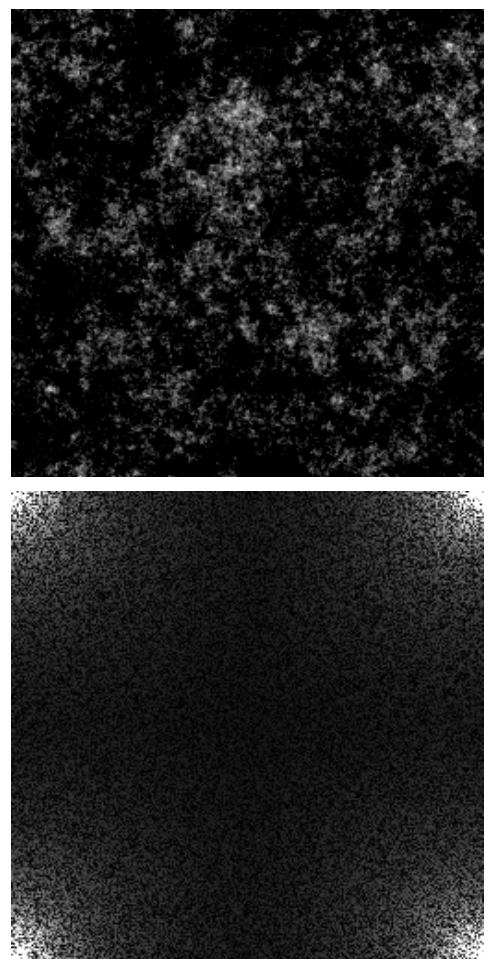
$1/|f|^2$



Signal space:

Frequency space:

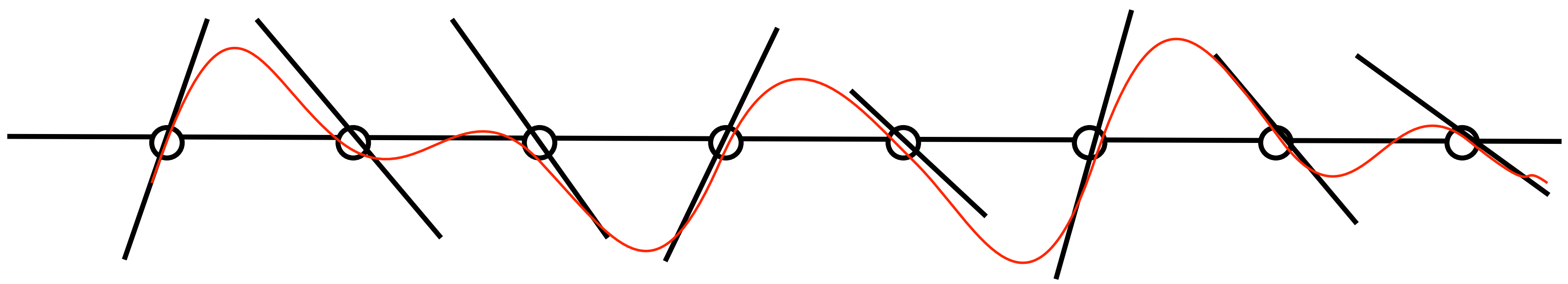
1/sqrt(|f|)



Gradient noise

If the values are used for gradients instead of height, we get *gradient noise*. (Perlin noise.)

The function is interpolated to match the gradients.



Simplex noise

Gradient noise based on triangles/tetrahedrons.
Ken Perlin’s replacement for ”Perlin noise” (gradient noise on quads).

Gradient noise vs FFT

Gradient noise claimed to be very fast. (Compared to what?)

Frequency space processing much simpler algorithm (simple weighting curve, based on 1/f, FFT) and great control, but requires O(NlogN) operations.

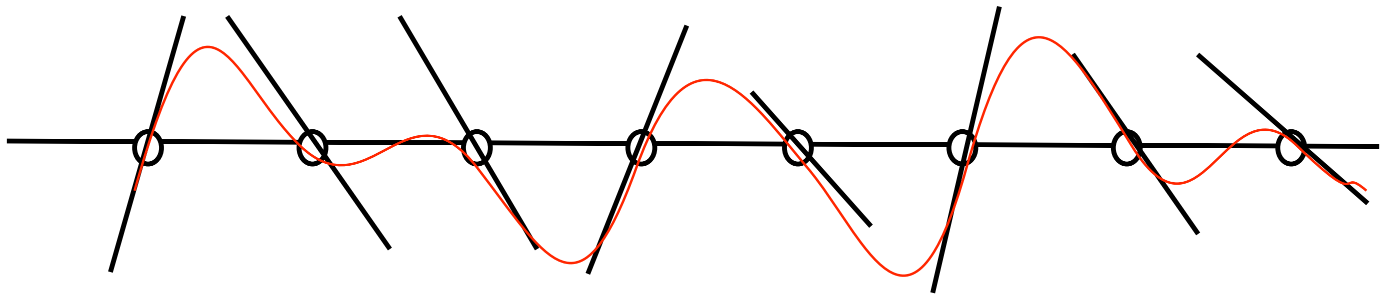
One pass Gradient noise faster... but don't we need many?

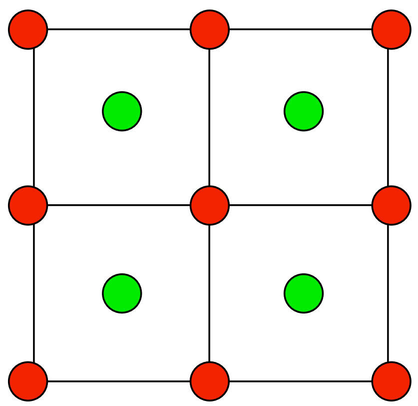
Artifacts

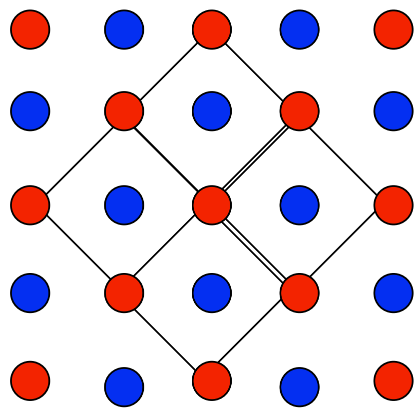
Diamond square and Perlin noise are incomplete! They both "lock" in certain points, only producing certain phases of the signal.

I.e. produce only the cosine part of a signal and skipping the sin!

This can be corrected by generating two sets of the signal, with a proper offset!







Applications of noise functions and randomness

Terrains

Textures, texture detail

Water

Smoke

Animations (particle systems etc)

Etc...

I told you noise is beautiful!

Feature comparison

Scalability: Diamond square and Square square very easy to scale

Perlin easy to scale if you add additional octaves - which degrades performance

Control: Frequency plane filtering has extreme control. The others depend on weights on octaves

Your application needs may decide

Performance comparison

Produce an NxN image

Diamond square: $O(N^2)$

Square square: $O(N^2)$

Single octave Perlin: $O(N^2)$

Multi octave Perlin: $O(N^2 \log N)$

Frequency plane filtered: $O(N^2 \log N)$

All produce similar results except single octave Perlin.

BUT: Square square and FP filter gives highest quality!

And then...?

- Add water, calculate rivers and lakes
- Erosion effects (esp along rivers)
- Roads
- Vegetation and buildings
- Expand into new patches
- Multitexturing for different kinds of locations (slopes, height)
- Different generation for different climates (mountains, deserts...?)

Conclusions of terrain generation

The backbone of procedural environment generation!

Fractal or noise? Same thing!

Higher frequencies - lower amplitudes. (Typical for natural images as well as a rule in fractals.)

Don't assume Perlin noise is best just because it is famous. Proper (traditional) signal processing methods will compete very favorably - if you filter properly!