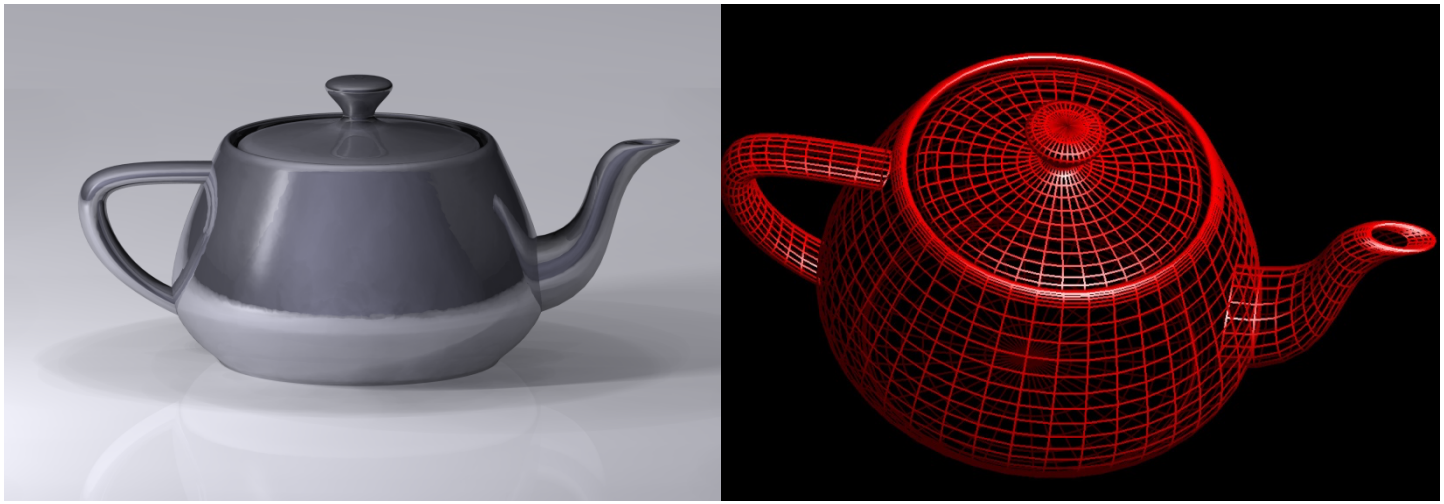


Foundations of Computer Graphics

Saurabh Ray

We will start at 16:07.

Tuesday, April 7.

The exam will be an assignment on NYU classes that starts at 16:00 and closes at 17:25.

You can submit pictures of your handwritten solutions.

Topics. Everything until lecture 14 except 'procedural noise'.

Please practice the exams from past offerings (on NYU classes).

I don't have solutions but I am happy answer questions during office hours.

Office hours. Today and on Monday from 17:30 - 19:00 on Zoom.

Today:.

Procedural Noise.

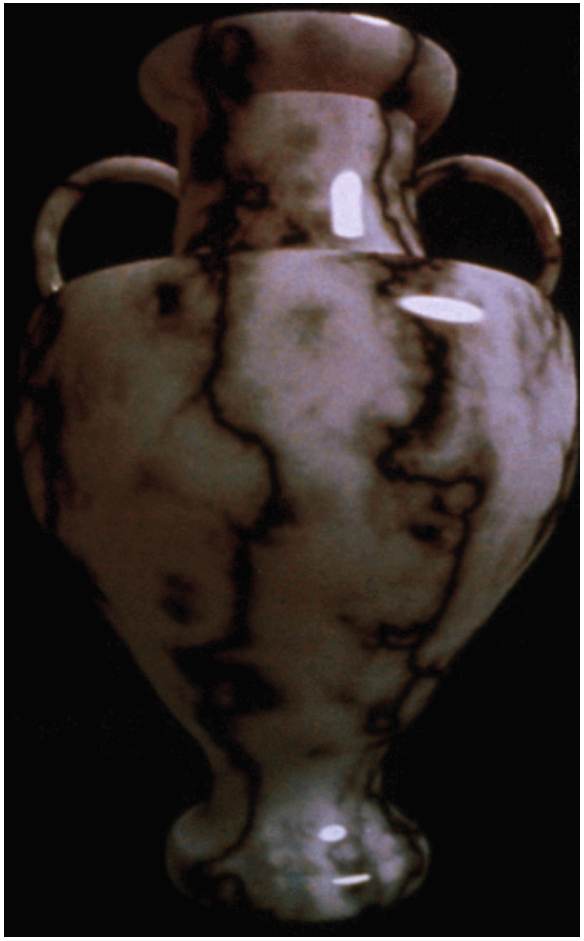
Lattice Based Noise: Value Noise, Gradient Noise.
Adding Self Similarity: Fractional Brownian Motion, Turbulence.
Extending to higher dimensions, Simplex Noise.

WebGL examples.

Reading: https://www.cs.umd.edu/class/fall2018/cmsc425/Lects/lect14-perlin.pdf

Code: https://github.abudhabi.nyu.edu/sr194/CG-2020/tree/master/Procedural%20Noise

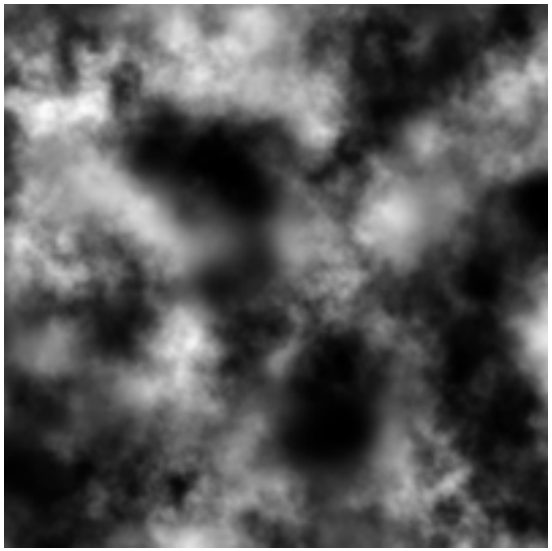
Procedural Generation



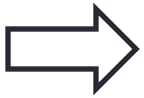
Many natural phenomena can be simulated procedurally.

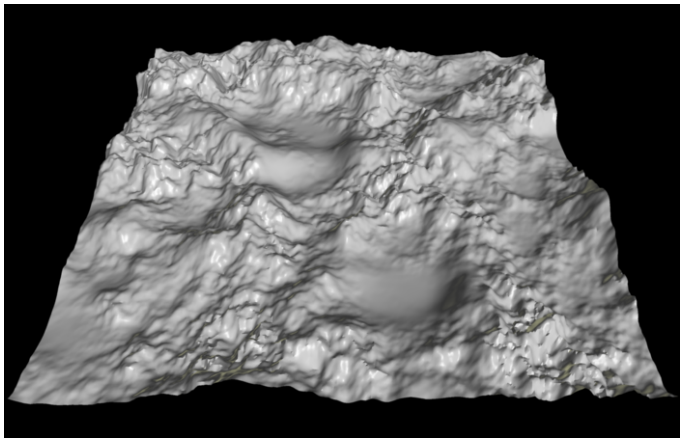
Key ingredient: Randomness.

Terrains using Height Maps



Height Map





Generated Terrain

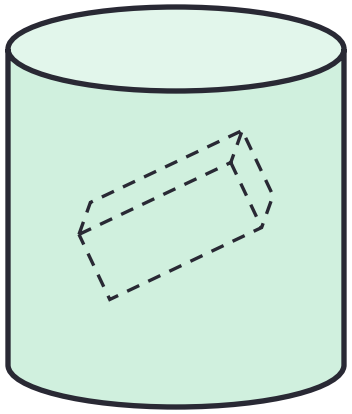
We can use a terrain texture like this →
or we can color according to height.

Solid (3d) Textures

Looks better than a 2d texture pasted on the surface of the model.

Storing the texture in a file limits resolution and requires more space, but is faster.

Procedural generation is slower but requires less storage space and has potentially unlimited resolution.



We want a function that given any point in the cylinder outputs the wood color at that point.

Creating a function so that each horizontal cross section has perfectly circular rings with two colors is easy.

Can be made more realistic using perturbations.

Procedural Noise

Natural phenomena contain a mixture of structure and randomness.

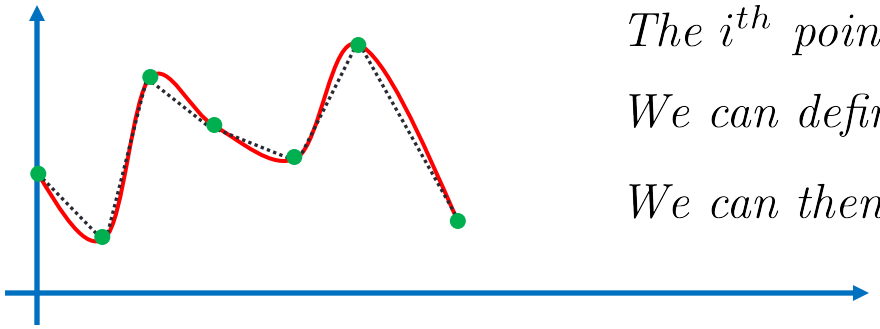
For example, altitude in a terrain isn't totally random.

There is random but gradual change as we move around in the terrain.

And this happens at all scales.

How do we obtain a smooth random function using a pseudo random number generator (PRNG)?

Idea. Interpolation of randomly generated points.



The i^{th} point has coordinates $(i, rand(i))$.

We can define the function on $[0, n]$ s.t. $f(0) = f(n)$.

We can then make it periodic: $g(t) = f(t \bmod n)$.

This is called **Value Noise**.

Procedural Noise

How do we achieve randomness at every scale?

$$N(t) = g(t) + \frac{1}{2}\, g(2t) + \frac{1}{4}\, g(4t) + \frac{1}{8}\, g(8t) + \cdots$$

We add terms with exponetially increasing frequency and exponentially decreasing amplitude.

This also gives us *self similarity.*

We could also add an *offset* to each scaled copy.

The functions $g(t), g(2t), \ldots$ are often called **octaves**.

In practice, we don't need an infinite sum since the amplitudes decrease exponentially.

This technique is often referred to as **fractional brownian motion.**

Closely related: **Turbulence.** $N(t) = |g(t)| + \frac{1}{2}\, |g(2t)| + \frac{1}{4}\, |g(4t)| + \cdots$

Gradient Noise.

We change how we obtain the initial function $f(t)$.

For each integer $t \in [0, n]$:

- we prescribe $f(t)$ using a PRNG

- we prescribe the slope $f'(t)$ using a PRNG

How do we define a "smooth" function $f(t)$ on $[0, n]$ to match the prescriptions at each integer in $[0, n]$?

Is there a unique way to construct such a function? **No!**

Procedural Noise

Here is one solution. For $t \in [i, i+1]$, define

$$f(t) = \begin{aligned}&(1-(t-i)) \cdot [\, f(i) + f'(i) \cdot (t-i)\,] \\ &+(t-i) \cdot [\, f(i+1) - f'(i+1) \cdot (i+1-t)\,]\end{aligned}$$

Instead of these, we can use $(1-\alpha(t-i))$ *and* $\alpha(t-i)$ *where* $\alpha(x)$ *is a smooth 'fading' function that goes from* 0 *to* 1 *as* x *goes from* 0 *to* 1.

e.g. $\alpha(x) = 3x^2 - 2x^3$ *or* $\alpha(x) = 6x^5 - 15x^4 + 10x^3$

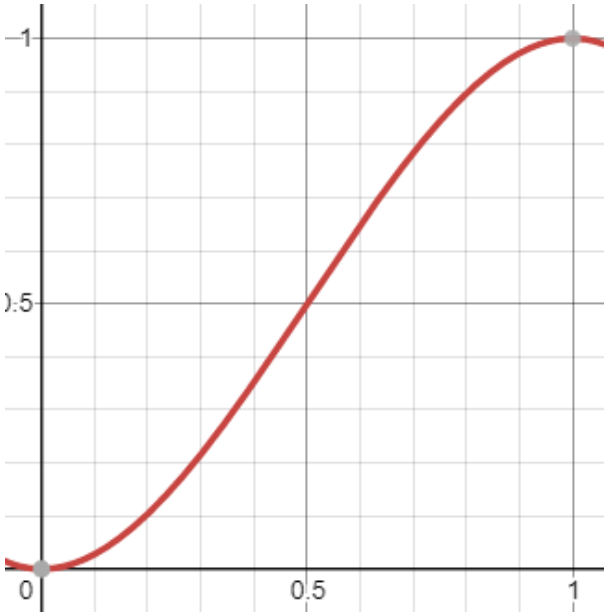
$$\begin{aligned}f(t) &= (1-\alpha(t-i)) \cdot (\, f(i) + f'(i) \cdot (t-i)\,) \\ &\quad + \alpha(t-i) \cdot (\, f(i+1) - f'(i+1) \cdot (i+1-t)\,)\end{aligned}$$

$$\begin{aligned}&= (1-\alpha(t-\lfloor t \rfloor)) \cdot (\, f(\lfloor t \rfloor) + f'(\lfloor t \rfloor) \cdot (t-\lfloor t \rfloor)\,) \\ &\quad + \alpha(t-\lfloor t \rfloor) \cdot (\, f(\lceil t \rceil) - f'(\lceil t \rceil) \cdot (\lceil t \rceil - t)\,)\end{aligned}$$

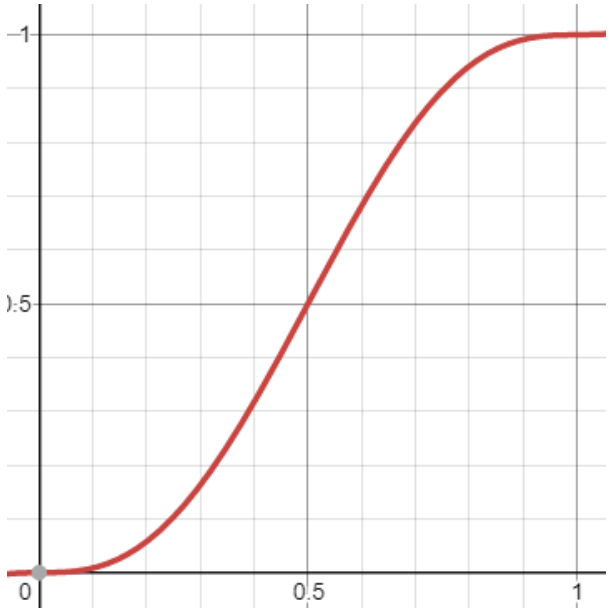
since $i = \lfloor t \rfloor$ *and* $i+1 = \lceil t \rceil$.

Procedural Noise

Plots of the ‘fading’ functions:



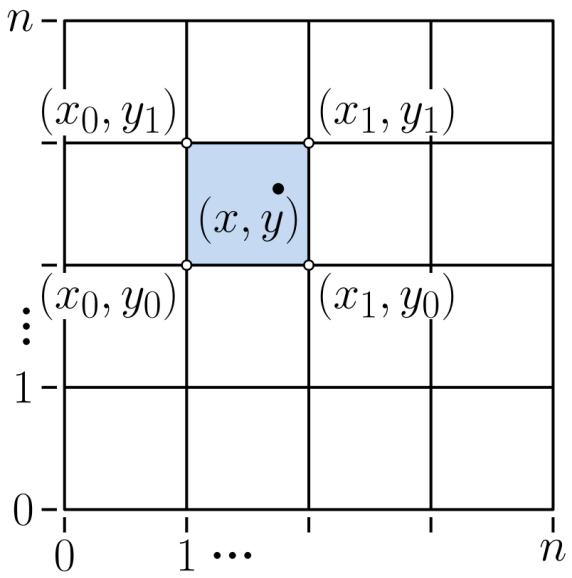
$\alpha(x) = 3x^2 - 2x^3$



$\alpha(x) = 6x^5 - 15x^4 + 10x^3$

Procedural Noise

How do we generalize value noise to two dimensions?



At each grid point in $[0, n-1] \times [0, n-1]$, *the values are randomly generated.*

For any other point $p = (x, y)$, *we bilinearly interpolate the values at the four corners of the grid cell containing* p.

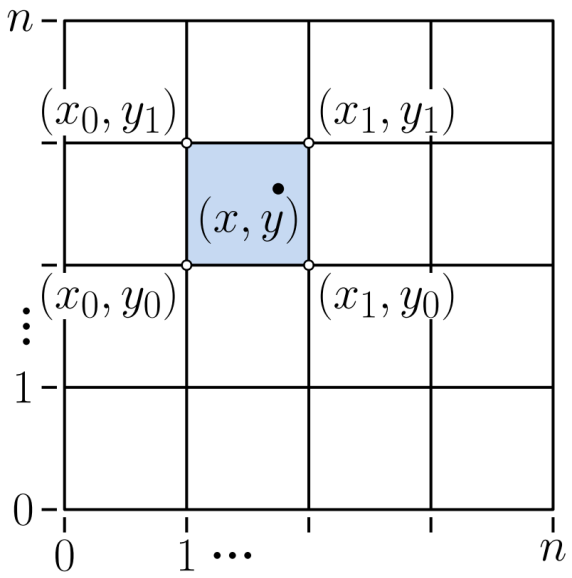
Let $x_0 = \lfloor x \rfloor$, $x_1 = (x_0 + 1) \bmod n$,
$y_0 = \lfloor y \rfloor$, $y_1 = (y_0 + 1) \bmod n$,
$a = x - x_0$, $b = y - y_0$.

$$\begin{aligned} f(x,y) = \quad & (1-a)(1-b) \cdot f(x_0, y_0) \\ & +(1-a)b \cdot f(x_0, y_1) \\ & +a(1-b) \cdot f(x_1, y_0) \\ & +ab \cdot f(x_1, y_1). \end{aligned}$$

As before, we can use a smooth fading function.

Procedural Noise

How do we generalize value noise to two dimensions?



At each grid point in $[0, n-1] \times [0, n-1]$, *the values are randomly generated.*

For any other point $p = (x, y)$, *we bilinearly interpolate the values at the four corners of the grid cell containing* p.

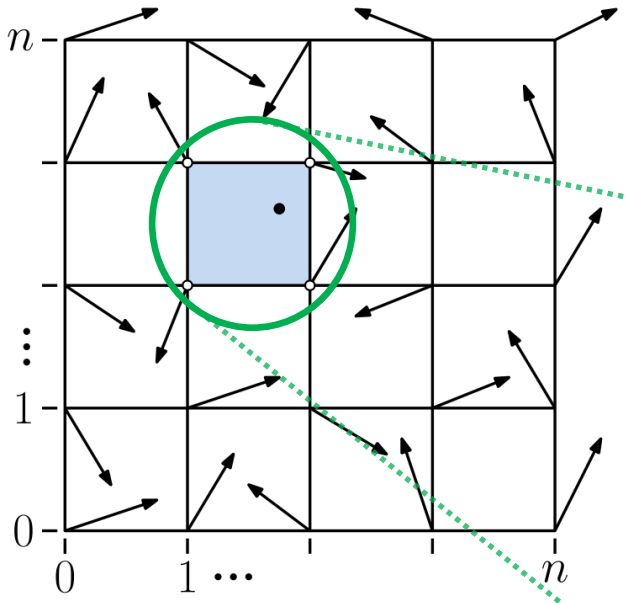
Let $x_0 = \lfloor x \rfloor$, $x_1 = (x_0 + 1) \bmod n$,
$y_0 = \lfloor y \rfloor$, $y_1 = (y_0 + 1) \bmod n$,
$a = x - x_0$, $b = y - y_0$.

$$\begin{aligned} f(x,y) = \quad & \alpha(1-a)\,\alpha(1-b) \cdot f(x_0, y_0) \\ & +\alpha(1-a)\,\alpha(b) \cdot f(x_0, y_1) \\ & +\alpha(a)\,\alpha(1-b) \cdot f(x_1, y_0) \\ & +\alpha(a)\,\alpha(b) \cdot f(x_1, y_1). \end{aligned}$$

Using a smooth fading function α.

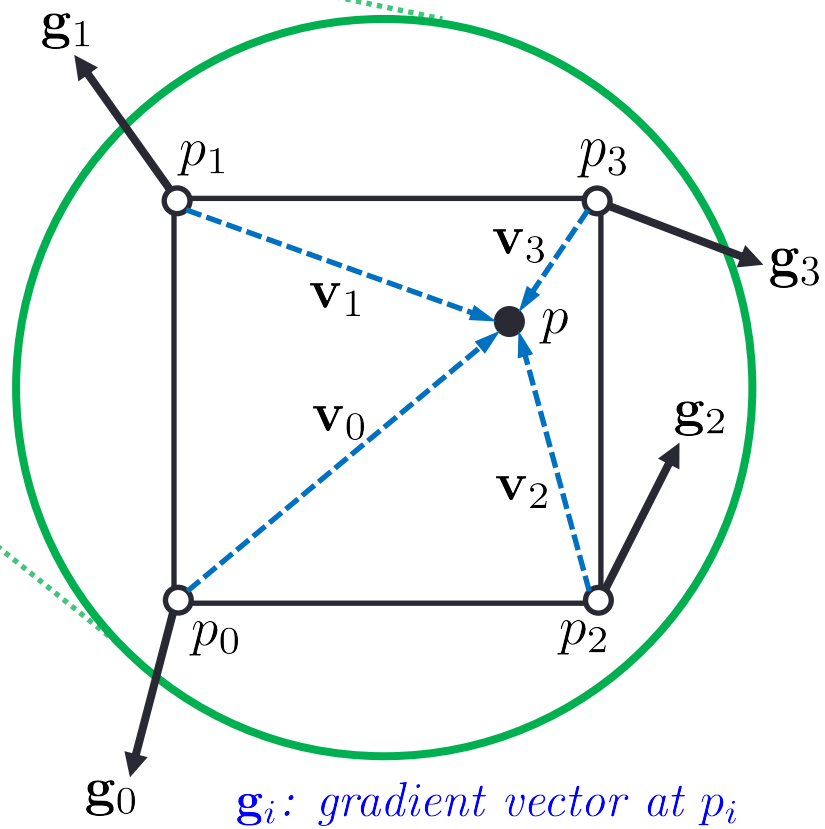
Procedural Noise

How do we generalize gradient noise to two dimensions?



For each grid point in $[0, n-1] \times [0, n-1]$, *we also prescribe a random gradient vector.*

We then construct a "smooth" function matching the prescribed values and gradients at each point.



Let $\lambda_i = f(p_i) + \mathbf{g}_i \cdot \mathbf{v}_i$,
$\forall i \in \{0, 1, 2, 3\}$.

$$
\begin{aligned}
f(p) = \; & \alpha(1-a)\,\alpha(1-b) \cdot \lambda_0 \\
& + \alpha(1-a)\,\alpha(b) \cdot \lambda_1 \\
& + \alpha(a)\,\alpha(1-b) \cdot \lambda_2 \\
& + \alpha(a)\,\alpha(b) \cdot \lambda_3 .
\end{aligned}
$$

$\mathbf{g}_i$: *gradient vector at* p_i

$\mathbf{v}_i$: $p - p_i$

Procedural Noise

As before, we can add octaves to obtain self similarity at different scales.

$$N(p) = \sum_{i=0}^{k} \frac{1}{r^i} \cdot f(s^i\, p)$$

r and s are some constants > 1.

How do we generalize this to three dimensions?

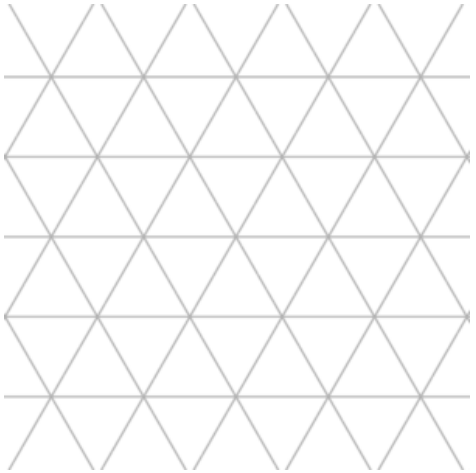
Time complexity in d dimensions? $O(d\, 2^d)$

Can be improved to $O(d^2)$ using **simplex noise**.

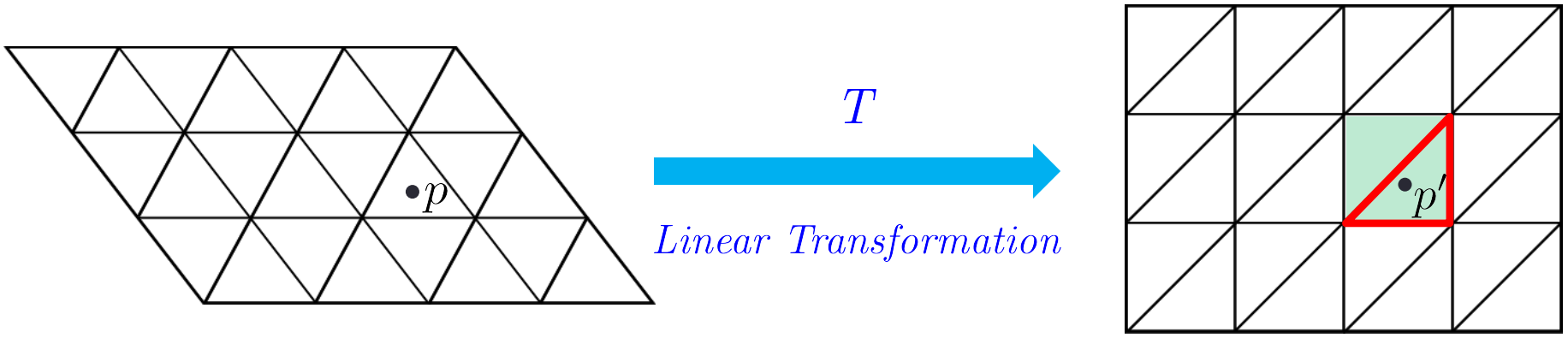
Idea. Tile the space with simplices instead of cubes.

Instead of 2^d corners of a cube, now we need to interpolate only using the values and gradients at the $d+1$ corners of a simplex.

Given a point p, how do we figure out the corners of the simplex containing it?



Procedural Noise

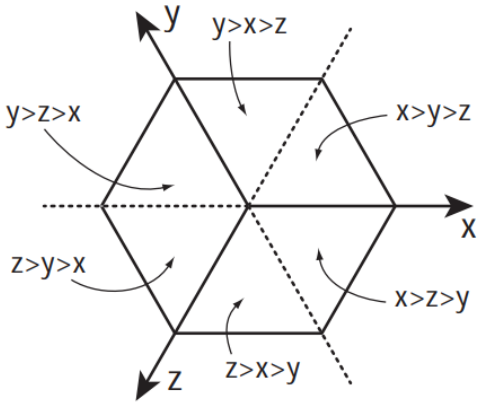


Given a point p in the first grid, we compute $p' = T(p)$.

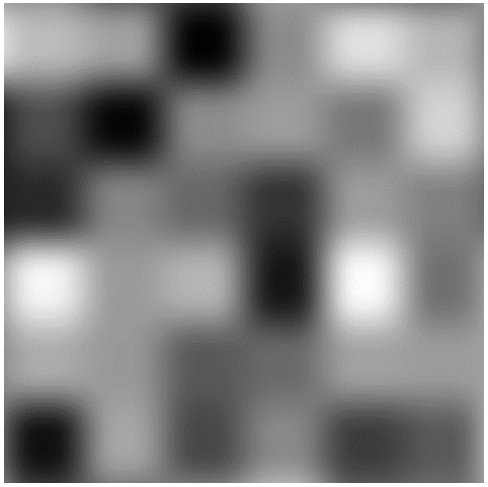
In the second grid, we can figure out the cubic grid cell containing p' using the floor function.

Within that cell, we can figure out the corners of the simplex σ' containing p' using the sorted order of the coordinates of p'. *Non-trivial!*

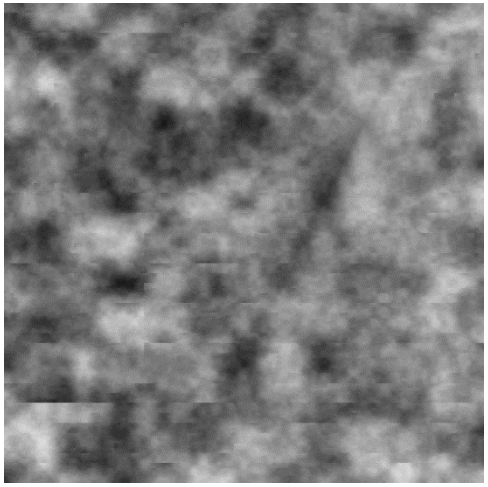
The simplex containing p is $\sigma = T^{-1}(\sigma)$.



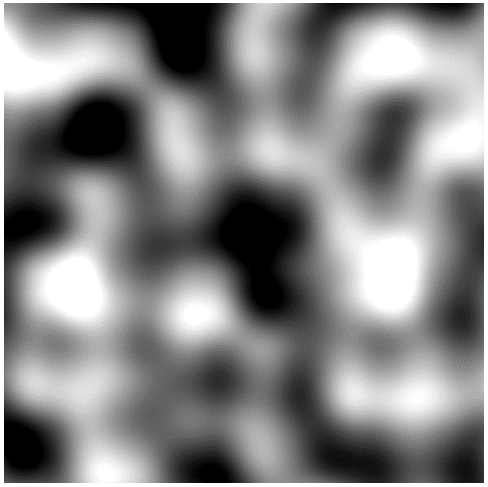
Procedural Noise



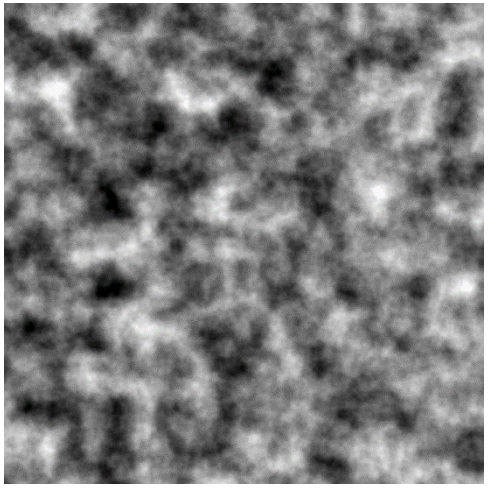
Value Noise



Sum of 8 octaves of Value Noise



Gradient Noise

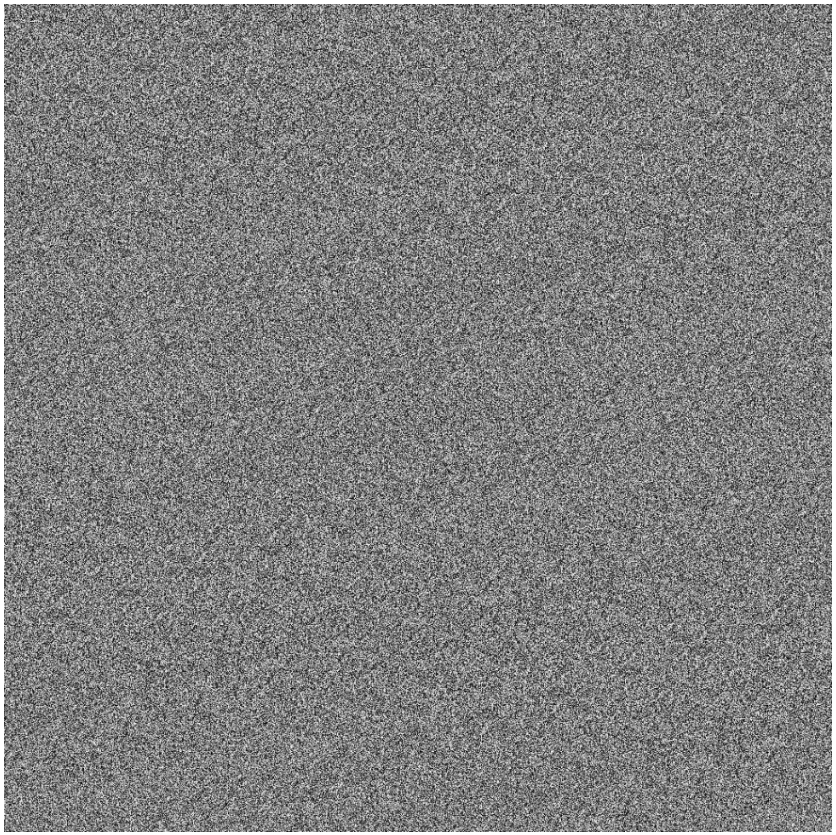


Sum of 8 octaves of Gradient Noise

Procedural Noise

Generating random numbers in GLSL:

```
float random(vec2 v, float salt){
    // returns a value in [0,1)
    float a = 5678.1234;
    vec2 b = vec2(12.3456, 78.910);
    float c = 1.1235+salt;
    return fract(a*sin(dot(b,v)+c));
}
```



This is actually a hash function.