

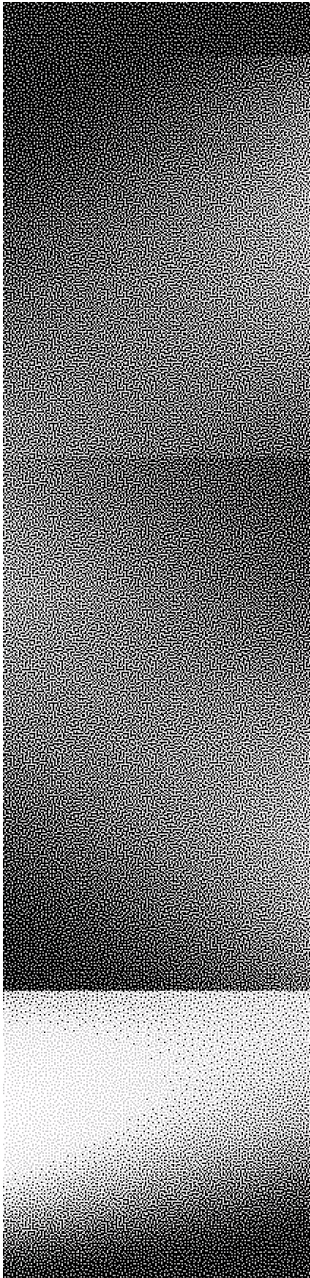
a zine by `this.xor.that`

a neither theoretical nor practical guide

# noisy pixels



# hi welcome



hello! you've found **this.xor.that**'s first zine. it's about one of my favorite things: random pixels.

## who is this.xor.that

I'm a creative coder who makes art with code. I'm also Jessica Stringham, a machine learning engineer who makes data do things with code. I use noise in both art and machine learning.

## who is this for

okay good question. I guess it's for me—a-few-months-ago. it tries to share some connections I saw. but watch out, this zine is not very practical

it'll get into the weeds and won't give you all the tools to get started

nor very theoretical

it'll gloss over both the basics and the hard work people do to make sure these things work

it's the zine I wanted to make. so maybe it's more of a love letter to noisy pixels? this zine will probably leave you wanting more. if so, there's a lot to seek out! Statisticians! Artists with interesting filters! Computer graphics engineers! Weird textures out in the world! and I guess if you do that, then this zine is for you too.

trends come from ups and downs that can feel random. sunny February days and snow in late May, but it'll be summer soon.

this.xor.that brooklyn February 2023

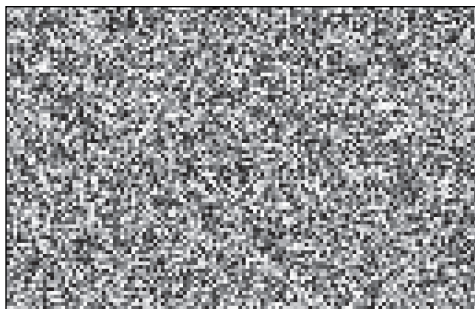
# random noise

let's start with a little square of random\* noise.

\*okay it's pseudo-random noise, but we'll get to that

plz admire this for a bit. →

this type of noise is distributed **uniformly**.



if we zoom in... →

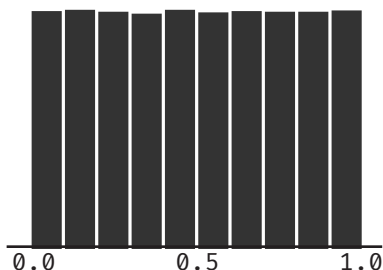


we can say each pixel represents some number between 0.0 and 1.0 ↓

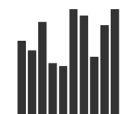
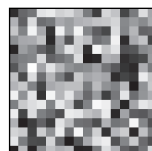
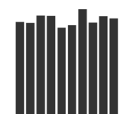
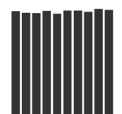


if we make a little histogram of all the different values in that image, we'd get a uniform distribution.

that's what's special about uniform distribution! every value is equally as likely. →



you can take a sample of any size, and usually you'll end up with roughly the same shape.

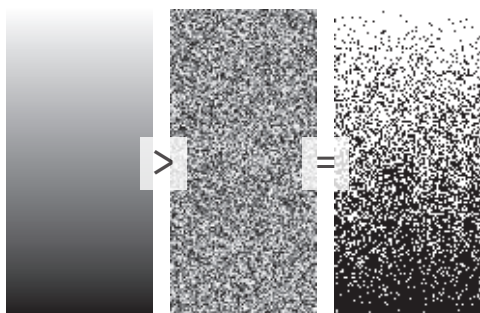
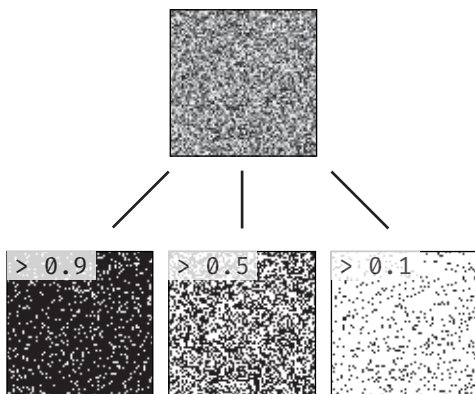


# some simple operations

you can also turn the uniformity into specks by only showing values above a certain amount.

if you want 90% of pixels to be on, you could turn values on when your random noise is  $> 0.9$ .

here are a few other examples  $\rightarrow$



you can do this on top of any grayscale image.

$\leftarrow$  for example, to this gradient

well, if you squint, you might notice that the noisy one looks darker. that's because of how we perceive things.

you can also remap the range of your noise



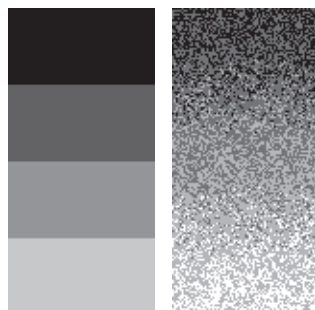
$\uparrow \times 0.2 - 0.1 = \downarrow$



and add to a gradient or other image.

this can be used to stretch a small number of grays to make a smoother gradient. though you may want to use a different type of noise.

I adjusted the range to be from 0.1 to 0.9 so that we don't go out of 0.0 and 1.0. you can also clip values!



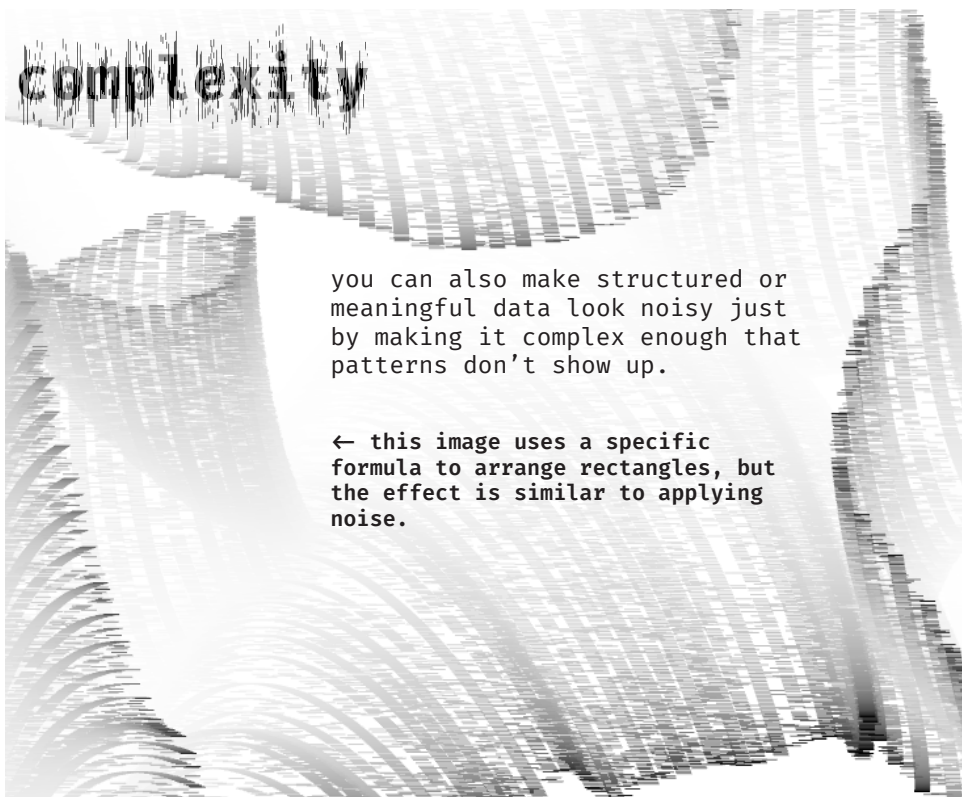
# displace



you can also use noise to displace pixels in an image. Like if we stretch out just one row of random noise, and then use that value to move pixels around.

that's how the titles of this zine were made!

## complexity



you can also make structured or meaningful data look noisy just by making it complex enough that patterns don't show up.

← this image uses a specific formula to arrange rectangles, but the effect is similar to applying noise.

another source of complexity that looks like noise is opening a file in the wrong format, e.g. an audio file as an image.

here's this ↑ opened as text →

```
?*???^????{??-6???Y}?????  
ZGbtM|7?V??2???5??h~?3N,c?N?y?^???Q?}?n  
?4s??K:??*o      ot?L??@?u?<??p???G_  
Z=?U??&?]ry?>"w?? ??rf??z?h??G+?Utw-{?4]  
 ??F??^Eg?P">W?l?
```

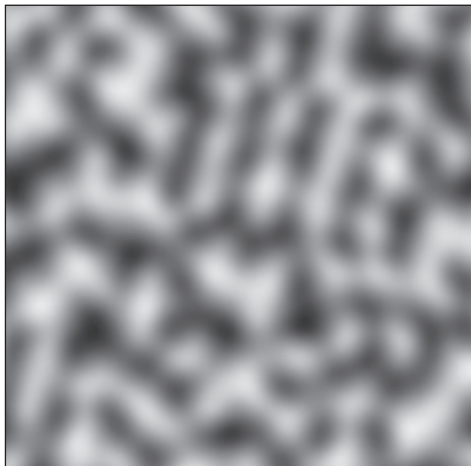
```
?F?,?w?????`G?e?@gW?z???
```

# perlin and simplex noise

Simplex and its predecessor Perlin noise are important in computer graphics. they look noisy, but are not actually independently random: values near each other have similar values.

this looks more similar to things in the real world, like mountains or clouds.

a sample from opensimplex →



## gaussian noise

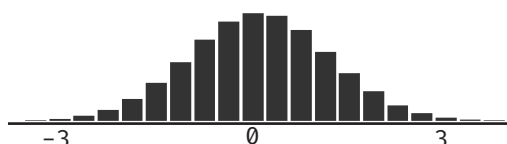
Gaussian

Uniform



Gaussian noise shows up in a bunch of places. it puts more values near the center, and values can be extreme.

here is a histogram of the kinds of values that show up. Compare it to uniform distribution on the previous page! ↓



tbh, I rarely use gaussian for noisy pixels. but the distribution is useful for other things! future zine?

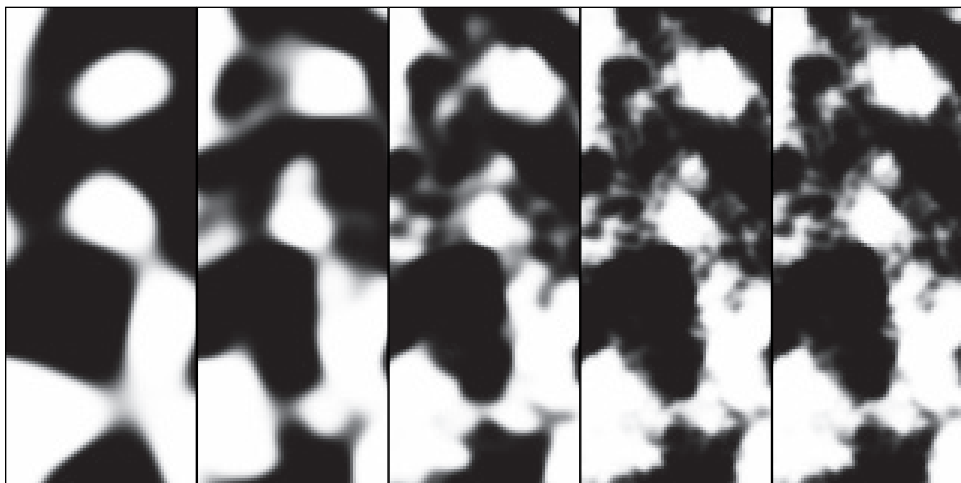
Perlin and Simplex noises work a little differently than our other sources of noise. you usually end up with a 2d or 3d space that you sample from. you can adjust the noise using the scale: taking samples really close to each other will get smooth shapes where neighbors are more influenced by each other, and taking samples really far away from each other will start to look more like random noise.

you can zoom in (top) or out (bottom) for different effects. (the values are smoothed to show the shapes a little better) →



fractals

A trick to turn something like simplex noise into clouds is to repeatedly scale it down and add it. ↓





# void-cluster

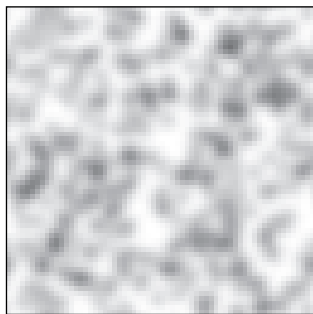
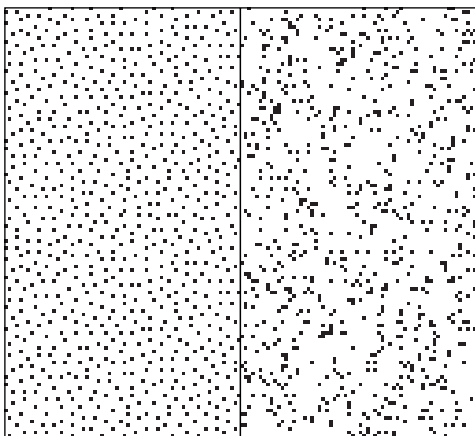
while random noise has properties that are invaluable for many applications, it does get a bit clumpy, and sometimes that's not the effect you're looking for.

a more spatially evenly distributed noisy effect uses the void-cluster algorithm.

here we select noise values  $< 0.1$  for a noisy texture created with the void-cluster algorithm and compare it to a uniform distribution →

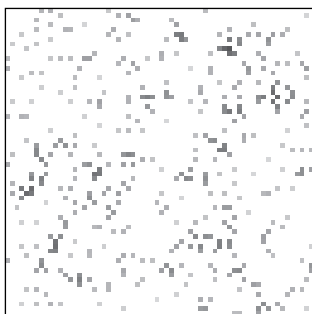
Blue  $< 0.1$

Uniform  $< 0.1$



blur the image to find the clusters and voids  
←

pick out the pixel in the brightest cluster, and swap with a pixel in a void →



void-cluster algorithm can be used to generate a blue noise texture.

the first stage is taking an image of a small amount of random noise and getting those better distributed.

there's a neat trick to find the most crowded pixel: blur the image, mask with the pixels that are on: the brightest pixel is the most in a cluster. then do the reverse and find the emptiest part of the void, and swap those two pixels.



# making noisy numbers

I want to wrap up by talking about where uniformly distributed random numbers come from.

in fields like cryptography, you want it to be extremely difficult for an adversary to guess your random number. this is usually less important in creative coding

## generating random numbers sequentially

a lot of programming languages have you initialize a random number generator (rng) object, and then call it every time you need a new random number.

some of these use linear congruential generators, which do a simple operation on the previous random number to create the next one.

```
[rng] → .237
  ↓
[rng] → .913
  ↓
[rng] → .832
  ↓
[rng] → .210
  ↓
[rng] → .513
  ↓
[rng] → .002
  ↓
[rng]
```

## generating random numbers in parallel

when you're working across a lot of pixels on a GPU or (a lot of machines), sometimes you need to generate a random number without waiting for the rng to finish the other pixels.

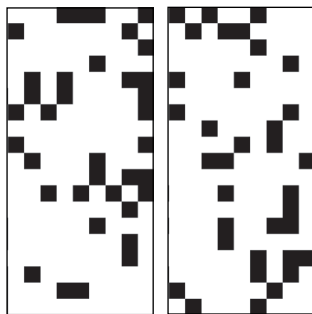
a trick is to pass an identifier (like the pixel location) through some function that gives you a number from a uniform distribution. one approach is to use hashing functions, which were made to distribute objects evenly in computer memory. you might see a function like `fract(sin(...))` used in shaders.

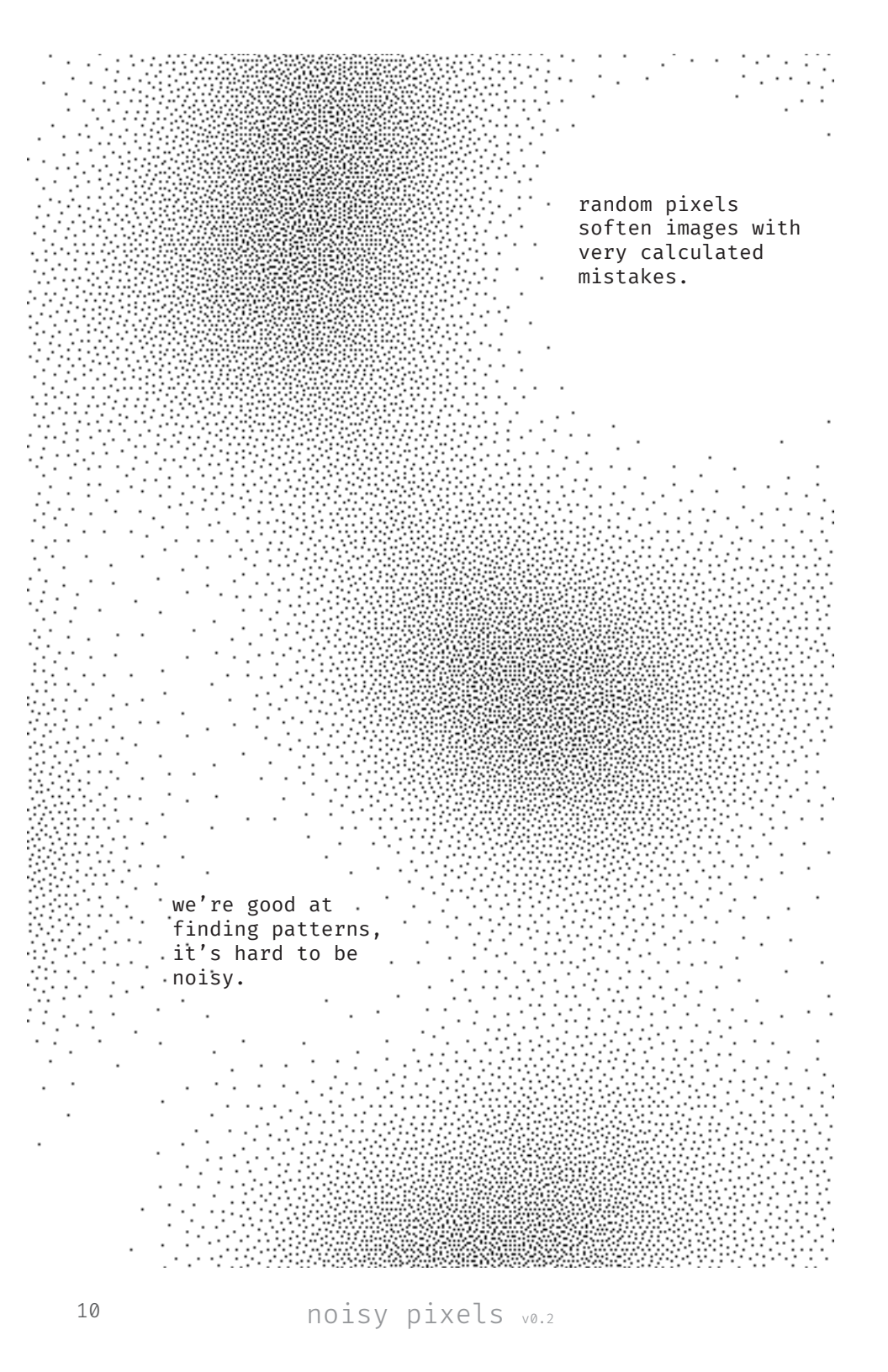
```
rng(0,0) → .237
rng(0,1) → .913
rng(0,2) → .832
rng(1,0) → .210
rng(1,1) → .513
rng(1,2) → .002
```

## deterministic random number generators

In many cases in creative coding and machine learning, it's pretty nice to be able to run the same code and get the same results! rngs do this if you set the seed, and changing the seed will give you a new set of random numbers. hash functions do this since they promise to give you the same value for the same input. you can append a seed value to the input if you want a new random number generated.

noise created with different seeds →





random pixels  
soften images with  
very calculated  
mistakes.

we're good at  
finding patterns,  
it's hard to be  
noisy.

# goodbye! thanks!

Thanks for sticking around!

## Errors

if you find an error, first check this page to see if you're on the latest version.

`https://thisxorthat.art/zines/noisypixels`

if it's still there, email me at

`jessica@thisxorthat.art`

with the error and if/how you want to be attributed.

## Attributes

this is typeset in Fira Code

the images were all created by me, `this.xor.that`, using Python (numpy and matplotlib) or by Rust (wgpu, nannou). The text was written in February and it was finished in October 2023. This was laid out in InDesign.

## Some other links

\* `https://thebookofshaders.com/` has a nice series on noise

\* `https://www.wedesoft.de/software/2022/09/21/blue-noise-dithering/` is useful for void-cluster and blue noise

