

Painting with noise

Randomness is an important part of making art, music, or games with computers. Randomness is how you can make things be interesting and surprising, how you can generate worlds no one has seen before without having to painstakingly craft them by hand every time. Randomness is how you can create surprising new melodies, craft coastlines, and animate plumes of smoke and fire that look realistic.

But not *all* randomness is created equal. When we think of "random" we usually mean *uniformly* random, where every outcome is equally likely.

If you put the following code into the p5js.org editor you'll see a visualization of how jagged true randomness can be:

```
function setup() {
  createCanvas(400, 400);
  noLoop();
}

function draw() {
  background(220);
  fill(0);
  stroke(0);
  beginShape();
  vertex(0, 400)
  for(let x = 0; x < 400; x++) {
    vertex(x, 400 * random());
  }
  vertex(400, 400)
  endShape();
}
```

Here we've let the y-axis be a random number between 0 and 400 and you can see that there's just jaggedness, no connection at all between the numbers, which is how it should be!

But what if you want a melody that smoothly climbs then falls again, or terrain that has smooth climbs to its highest peaks rather than sudden cliffs? Then *uniform* randomness doesn't actually help. Instead, you need something that's *random* but *smooth*. That's what "Perlin noise" is! It's an algorithm invented by an early computer graphics pioneer named Ken Perlin that is random *in certain spots* and then creates a smooth curve between those random points. It's not a terribly complicated algorithm, but it's slightly more than we can cover in this article, which is why all our examples are using P5.js since it has a built-in implementation of Perlin noise in its `noise` function.

So here's a similar visualization with the `noise` function:

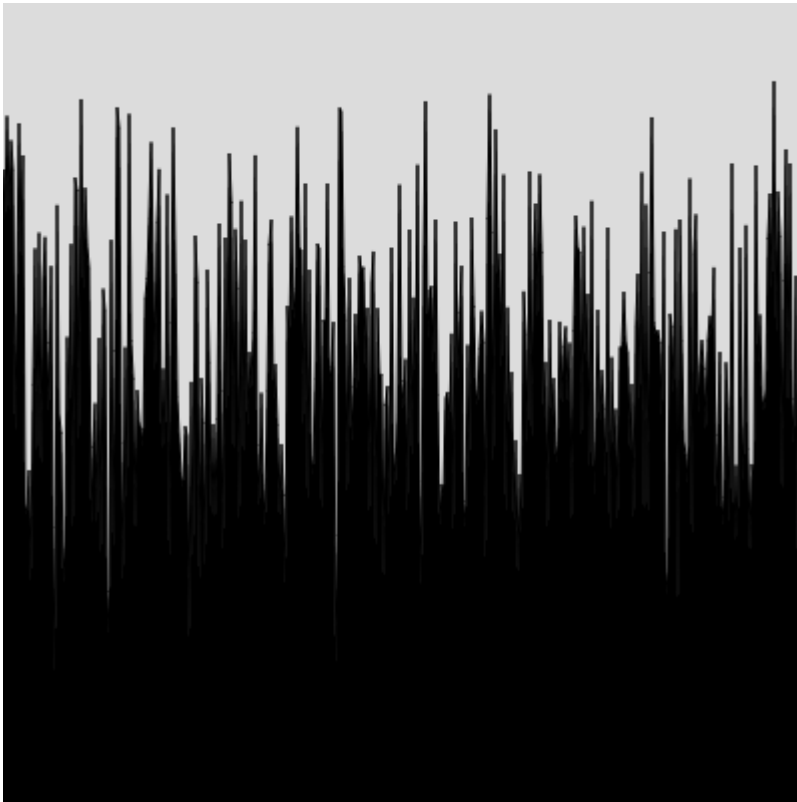
```
let zoom = 1;
function setup() {
  createCanvas(400, 400);
}

function draw() {
  background(220);
  fill(0);
  stroke(0);
```

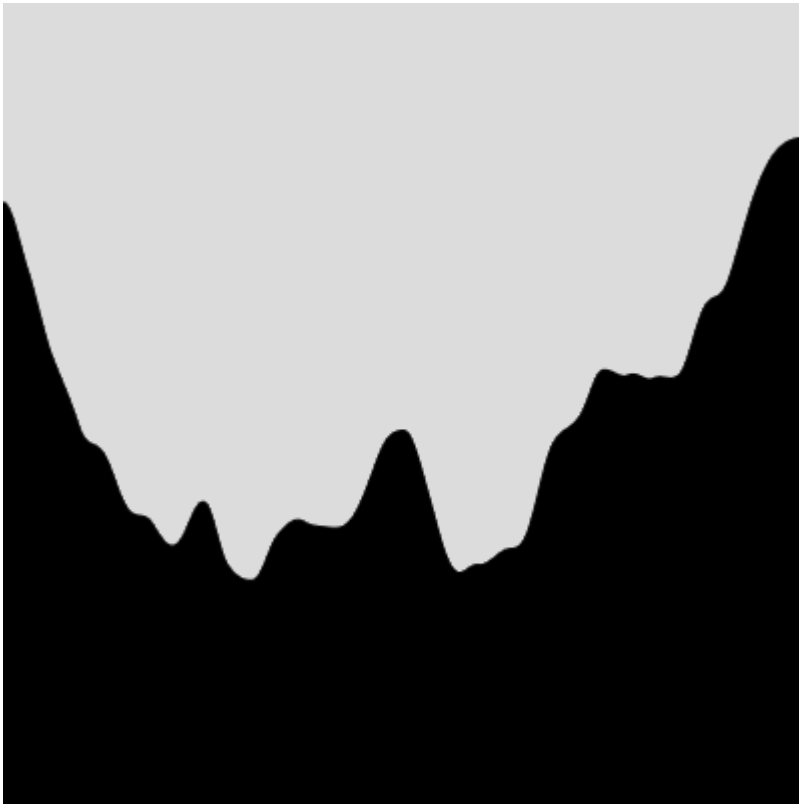
```
beginShape()  
  vertex(0,400)  
  for(let x = 0; x < 400; x++){  
    vertex(x,400*noise(x/zoom));  
  }  
  vertex(400,400)  
endShape()  
}
```

It might look a little less jagged, but go ahead and change the `zoom` variable to 10 or 100 or even 1000 and suddenly it looks a lot smoother than it did before, unlike the truly random version. Unlike `random` we have to give `noise` arguments to represent the point in space and/or time you're at. It makes sense that you'd need this in order to get the smoothness!

Perlin noise with a zoom of 1



Perlin noise with a zoom of 100



Now that we've shown that the noise is *smooth* let's come up with some fun examples with it:

Here's one where we use `noise` with an `x` *and* a `y` argument in order to create smooth terrain:

```
let zoom = 100;
let waterLevel = 0.4;

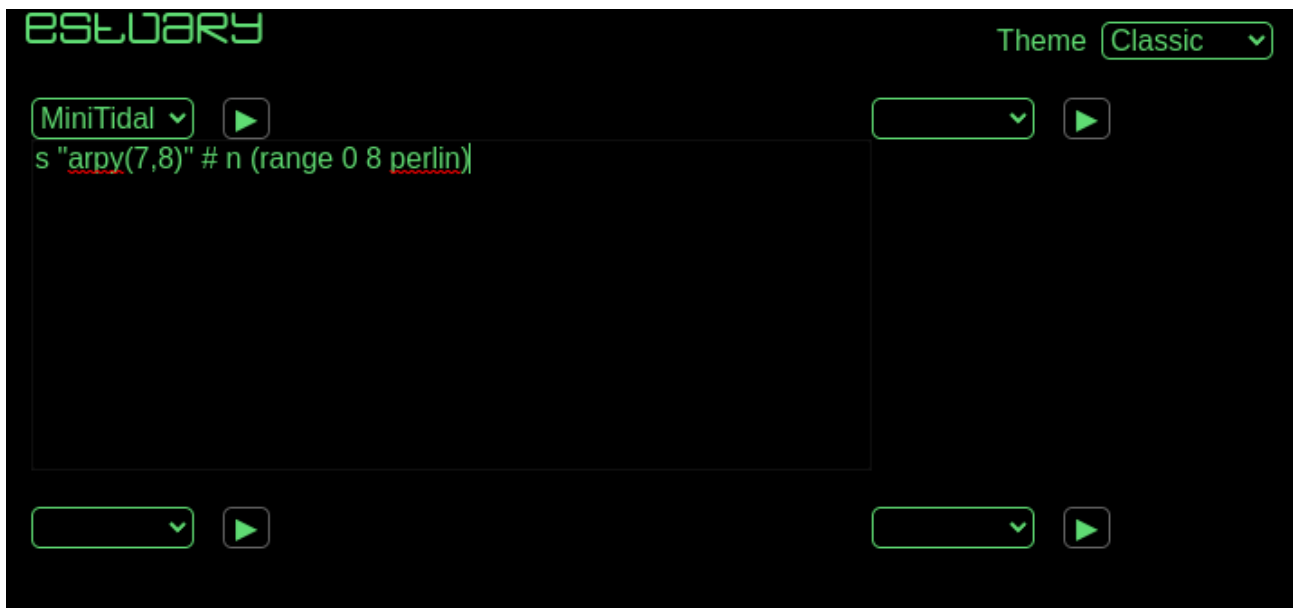
function setup() {
  createCanvas(400, 400);
  noLoop();
}

function draw() {
  for(let x = 0; x < 400; x++) {
    for(let y = 0; y < 400; y++) {
      let h = noise(x/zoom, y/zoom);
      if(h < waterLevel) {
        stroke(color("#0677D7"));
        point(x, y);
      }
      else {
        stroke(240*h);
        point(x, y);
      }
    }
  }
}
```

The basic idea is that we let the result of the `noise` function represent elevation, we set all pixels below a certain "elevation" to a blue color that represents water, and then *above* that level let all the pixels be grey and transitioning to white the higher they are.

There's a lot more you can do with noise like create fire or smoke effects, or even use it in things like music!

If you go to estuary.mcmaster.ca, click solo mode, and type `s "arpy(7,8)" # n (range 0 8 perlin)` into a miniTidal window like this



then hit the play button you'll hear a simple little melody that is both random and yet noodles up and down in time smoothly. This is a toy example but I've actually used a more complicated version of this trick to create computer generated melodies for lo-fi piano purposes!

No matter what kind of generative art you want to make, odds are pretty good you'll be using Perlin noise at some point!