

Playing with Perlin Noise: Generating Realistic Archipelagos



Yvan Scher · Follow

5 min read · Nov 7, 2017



Listen

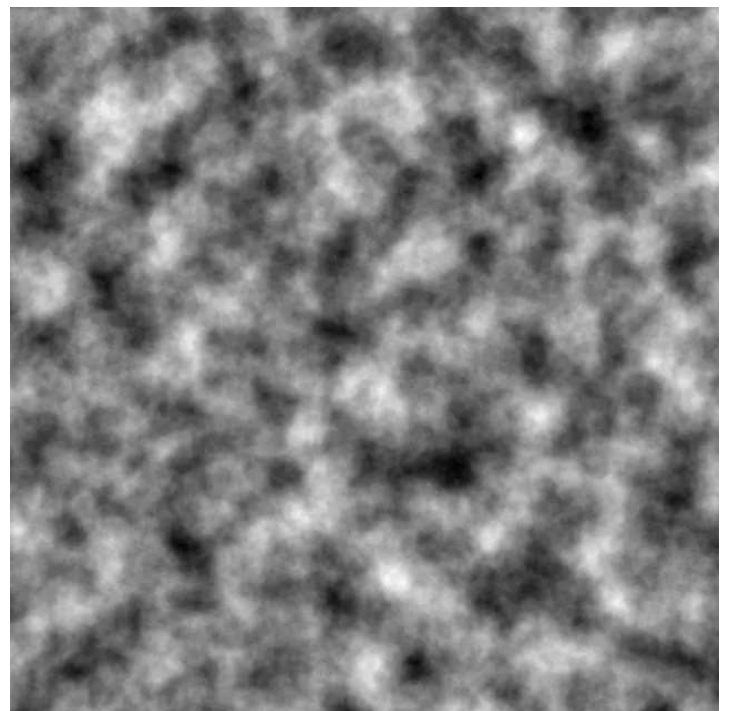
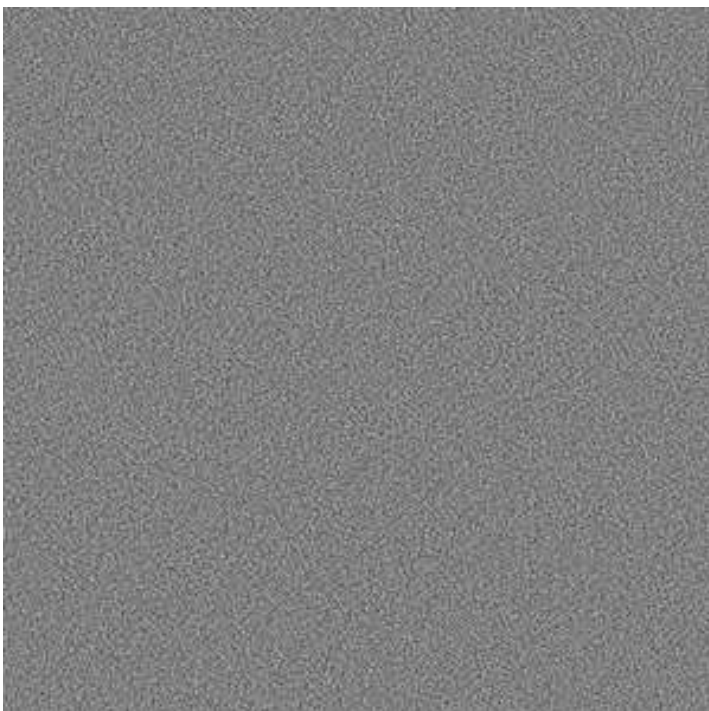


Share

If you are still reading this on medium I moved my blog to <http://yvanscher.com/blog.html>

I haven't seen a ton of great examples of making maps with perlin noise in python. So here we go!

Perlin noise is a mathematical formula used to generate 'realistic' structures. It's noise but unlike regular noise it has some coherent structure. Here is regular noise vs. Perlin noise:



regular static and perlin noise

In the python noise module there are a few parameters that affect what you see when you generate your perlin noise:

1. **scale:** number that determines at what distance to view the noisemap.
2. **octaves:** the number of levels of detail you want your perlin noise to have.
3. **lacunarity:** number that determines how much detail is added or removed at each octave (adjusts frequency).
4. **persistence:** number that determines how much each octave contributes to the overall shape (adjusts amplitude).

We won't worry about scale too much, you can use it to zoom out (bigger scale) or in (smaller scale).

Perlin noise combines multiple functions called 'octaves' to produce natural looking surfaces. Each octave adds a layer of detail to the surface. For example: octave 1 could be mountains, octave 2 could be boulders, octave 3 could be the rocks.

Lacunarity of more than 1 means that each octave will increase its level of fine grained detail (increased frequency). Lacunarity of 1 means that each octave will have the same level of detail. Lacunarity of less than one means that each octave will get smoother. The last two are usually undesirable so a lacunarity of 2 works quite well.

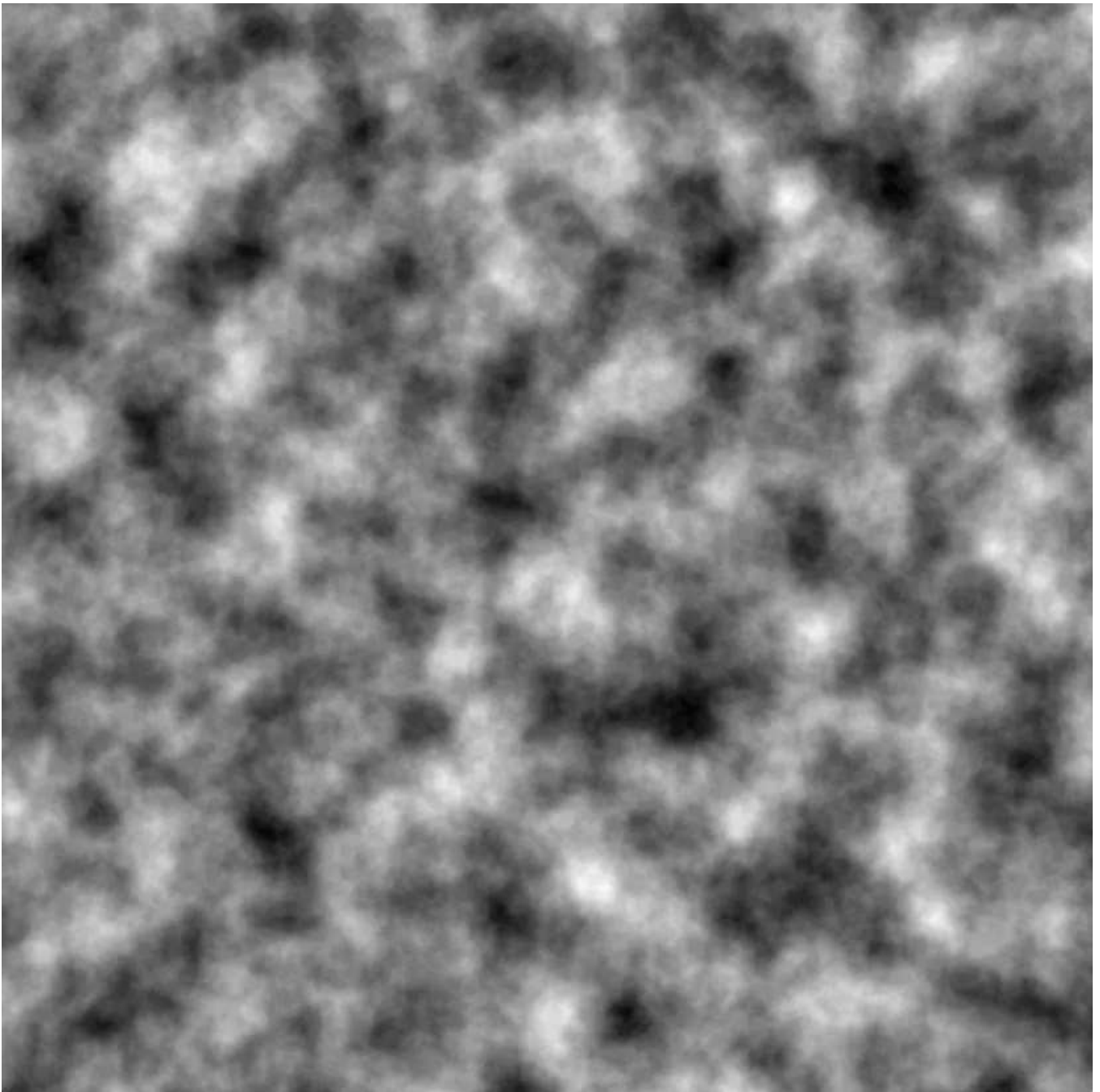
Persistence determines how much each octave contributes to the overall structure of the noise map. If your persistence is 1 all octaves contribute equally. If your persistence is more than 1 successive octaves contribute more and you get something closer to regular noise (spoiler the regular noise image above is actually a perlin noise with a persistence of 5.0). A more default setting would be a persistence of less than 1.0 which will decrease the effect of later octaves.

Enough chatting though! Let's run some experiments. First let's start with default perlin noise, and its accompanying image:

```
1  import noise
2  import numpy as np
3  from scipy.misc import toimage
4
5  shape = (1024,1024)
6  scale = 100.0
7  octaves = 6
8  persistence = 0.5
9  lacunarity = 2.0
10
11 world = np.zeros(shape)
12 for i in range(shape[0]):
13     for j in range(shape[1]):
14         world[i][j] = noise.pnoise2(i/scale,
15                                     j/scale,
16                                     octaves=octaves,
17                                     persistence=persistence,
18                                     lacunarity=lacunarity,
19                                     repeatx=1024,
20                                     repeaty=1024,
21                                     base=0)
22
23 toimage(world).show()
```

default_noise.py hosted with ❤ by GitHub

[view raw](#)



The way this perlin noise looks in our script is a 2D array of values between -1 and 1. The values that are darker on the map are lower values, the values that are close to 1 are lighter. What I want to try next is assigning two colors to different ranges of values in this map to produce some terrain:

```
1  blue = [65,105,225]
2  green = [34,139,34]
3  beach = [238, 214, 175]
4
5  def add_color(world):
6      color_world = np.zeros(world.shape+(3,))
7      for i in range(shape[0]):
8          for j in range(shape[1]):
9              if world[i][j] < -0.05:
10                 color_world[i][j] = blue
11             elif world[i][j] < 0:
12                 color_world[i][j] = beach
13             elif world[i][j] < 1.0:
14                 color_world[i][j] = green
15
16         return color_world
17
18 color_world = add_color(world)
19 toimage(color_world).show()
```

first_terrain.py hosted with ❤ by GitHub

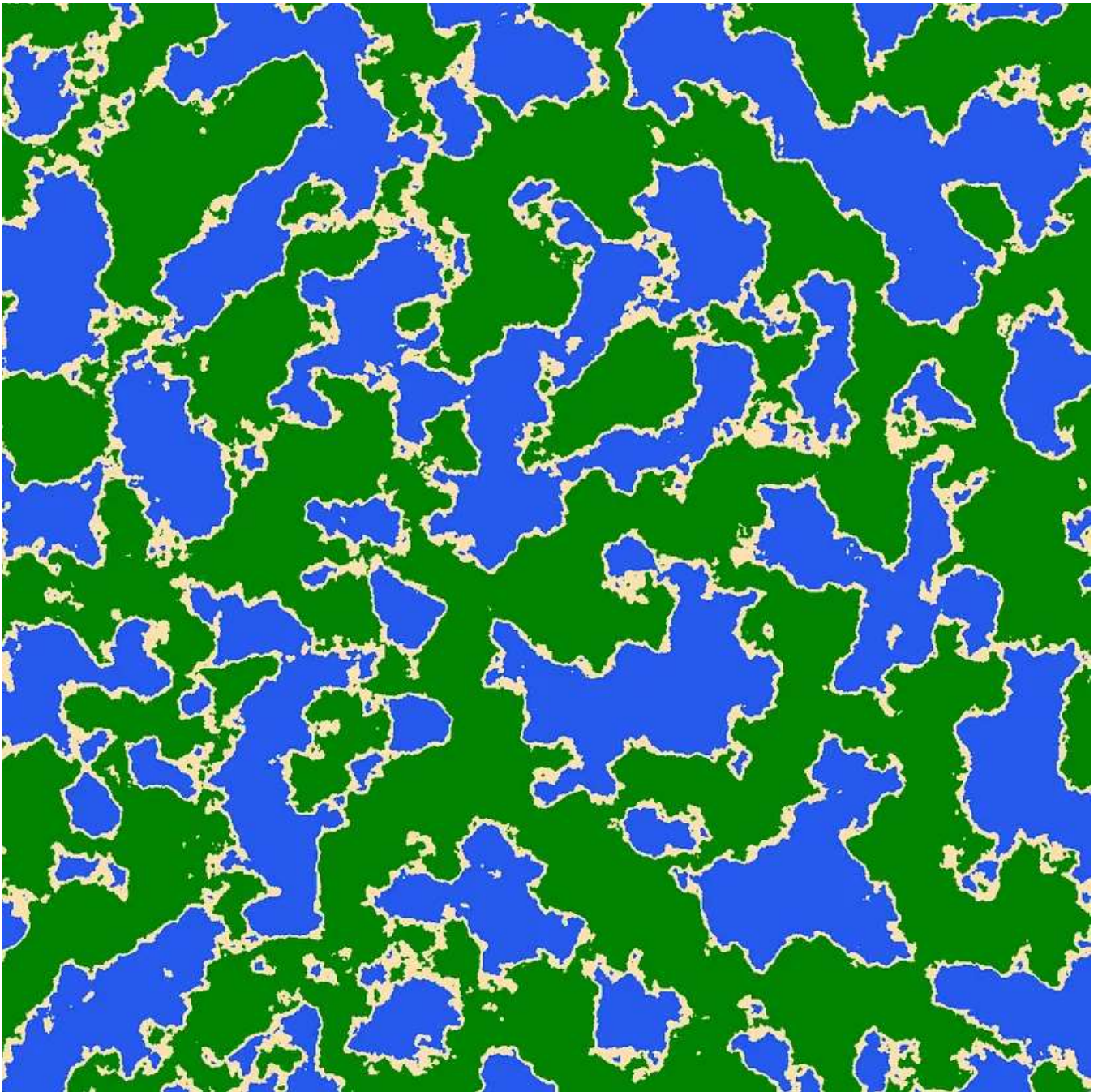
[view raw](#)

[Open in app](#) ↗

[Sign up](#)

[Sign In](#)



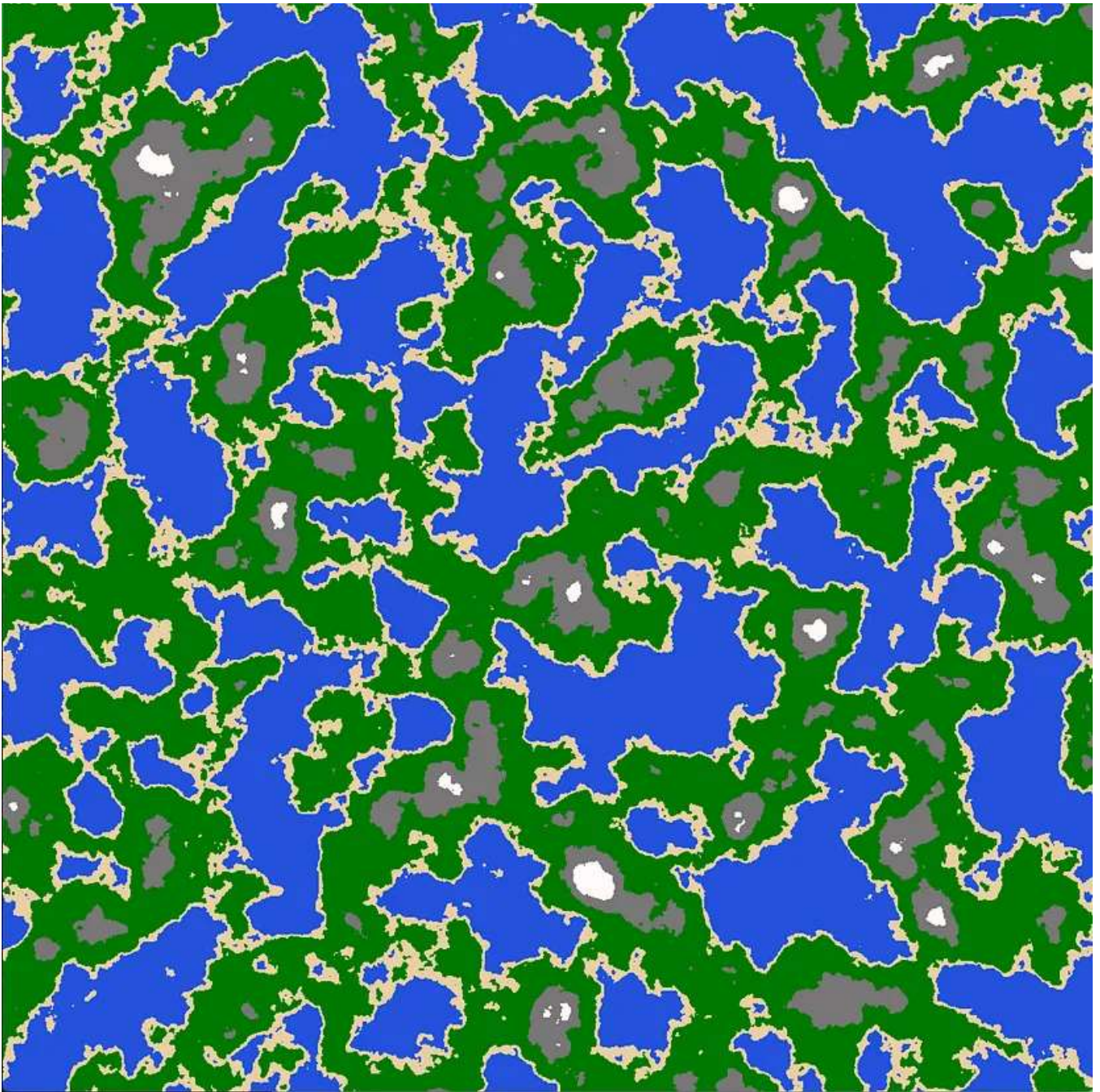


This terrain map is pretty neat; it has jagged coasts, beaches, and lots of water. while I have never observed natural terrain that looks like this if we look at any one part of the map it seems 'realistic.' Let's take it a step further and add mountains and snow:

```
1  blue = [65,105,225]
2  green = [34,139,34]
3  beach = [238, 214, 175]
4  snow = [255, 250, 250]
5  mountain = [139, 137, 137]
6
7  def add_color(world):
8      color_world = np.zeros(world.shape+(3,))
9      for i in range(shape[0]):
10         for j in range(shape[1]):
11             if world[i][j] < -0.05:
12                 color_world[i][j] = blue
13             elif world[i][j] < 0:
14                 color_world[i][j] = beach
15             elif world[i][j] < 1.0:
16                 color_world[i][j] = green
17             elif world[i][j] < 0.35:
18                 color_world[i][j] = mountain
19             elif world[i][j] < 1.0:
20                 color_world[i][j] = snow
21
22         return color_world
23
24 color_world = add_color(world)
25 toimage(color_world).show()
```

second_terrain.py hosted with ❤ by GitHub

[view raw](#)



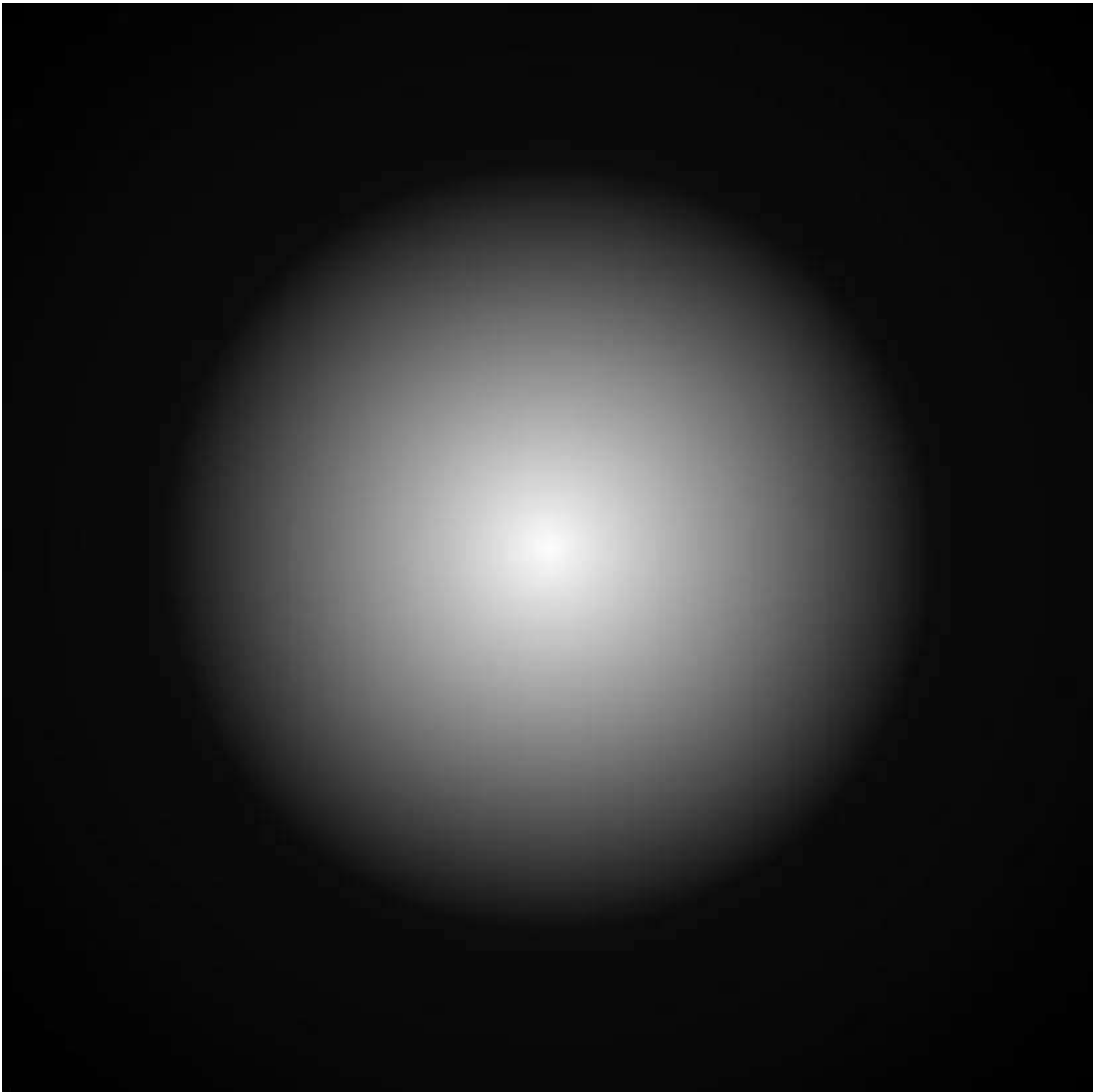
This is cool but this terrain pattern is clearly not natural. To make it more natural we will use a circular filter to get rid of all the peripheral perlin noise:



We'll call it Perlin's World!

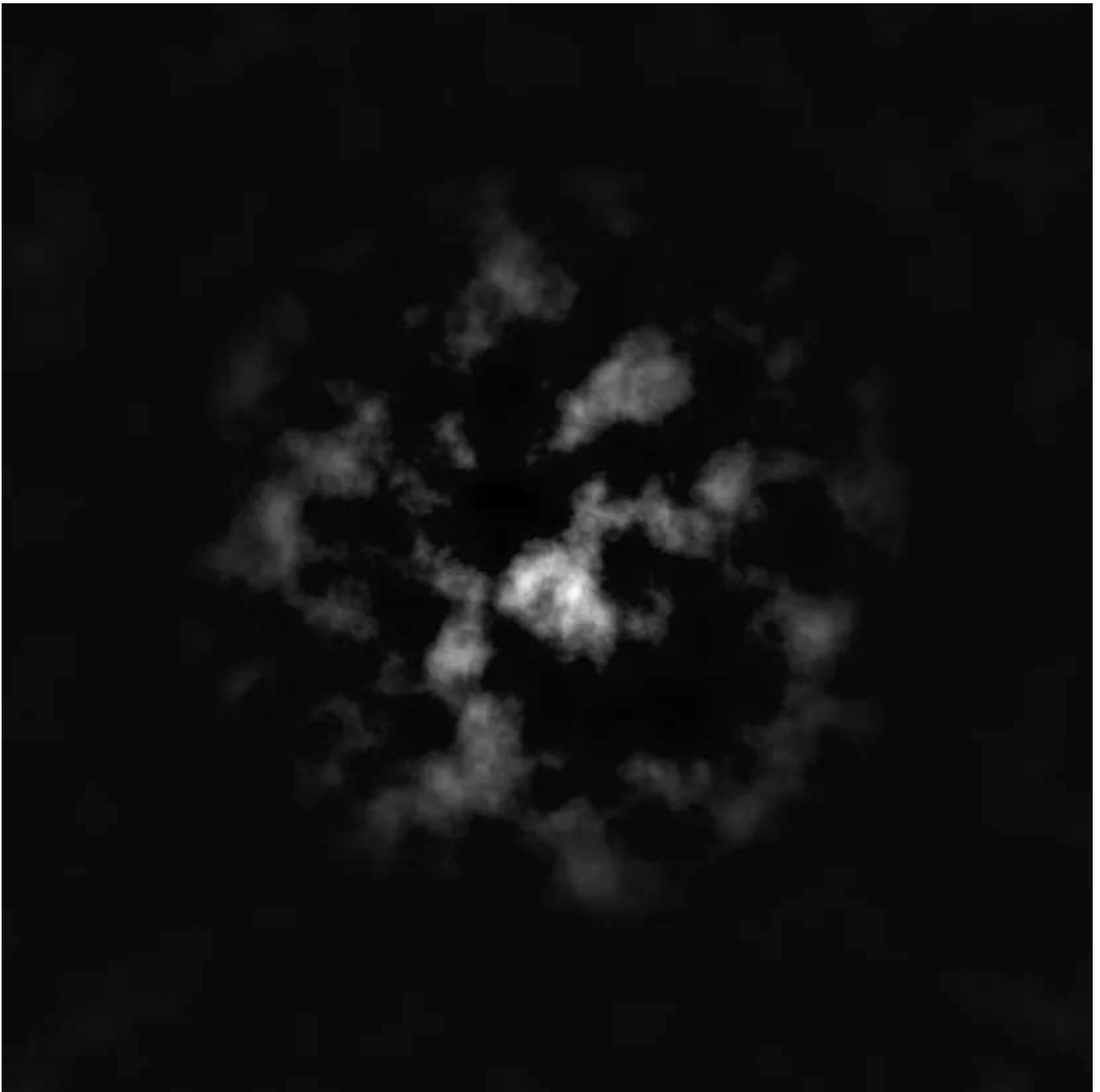
Here I was trying to create an island so I made a circular filter and then applied it to the color_world perlin noise array. What I ended up was a planet floating in an ocean. I changed the ocean color to black and it looks pretty cool! That said what I wanted was an island so let's try again. This time we're going to calculate a circular gradient and then apply that over the perlin noise as a filter.

Circular Gradient:



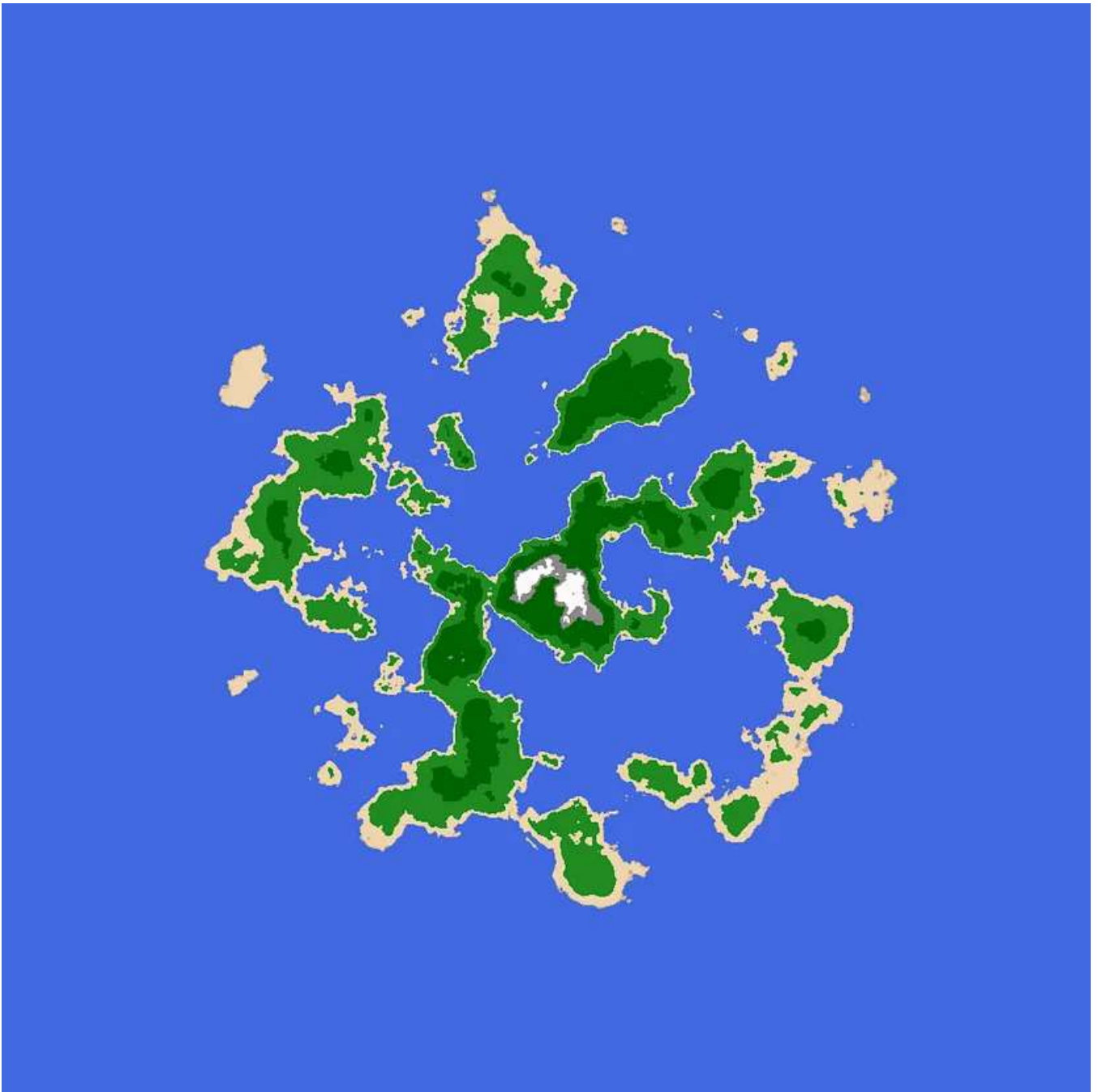
I struggled a lot with this part. I'm sure there is a more efficient way to get the gradient like this but the above was what I came up with. I calculated a distance metric from the center of the map and then normalized, shrunk, and renormalized those distances to produce this spherical gradient. Again lighter means the value is closer to 1, darker colors are closer to 0. Next I apply this circular gradient to the perlin noise we created before.

Circular Gradient + Noise:

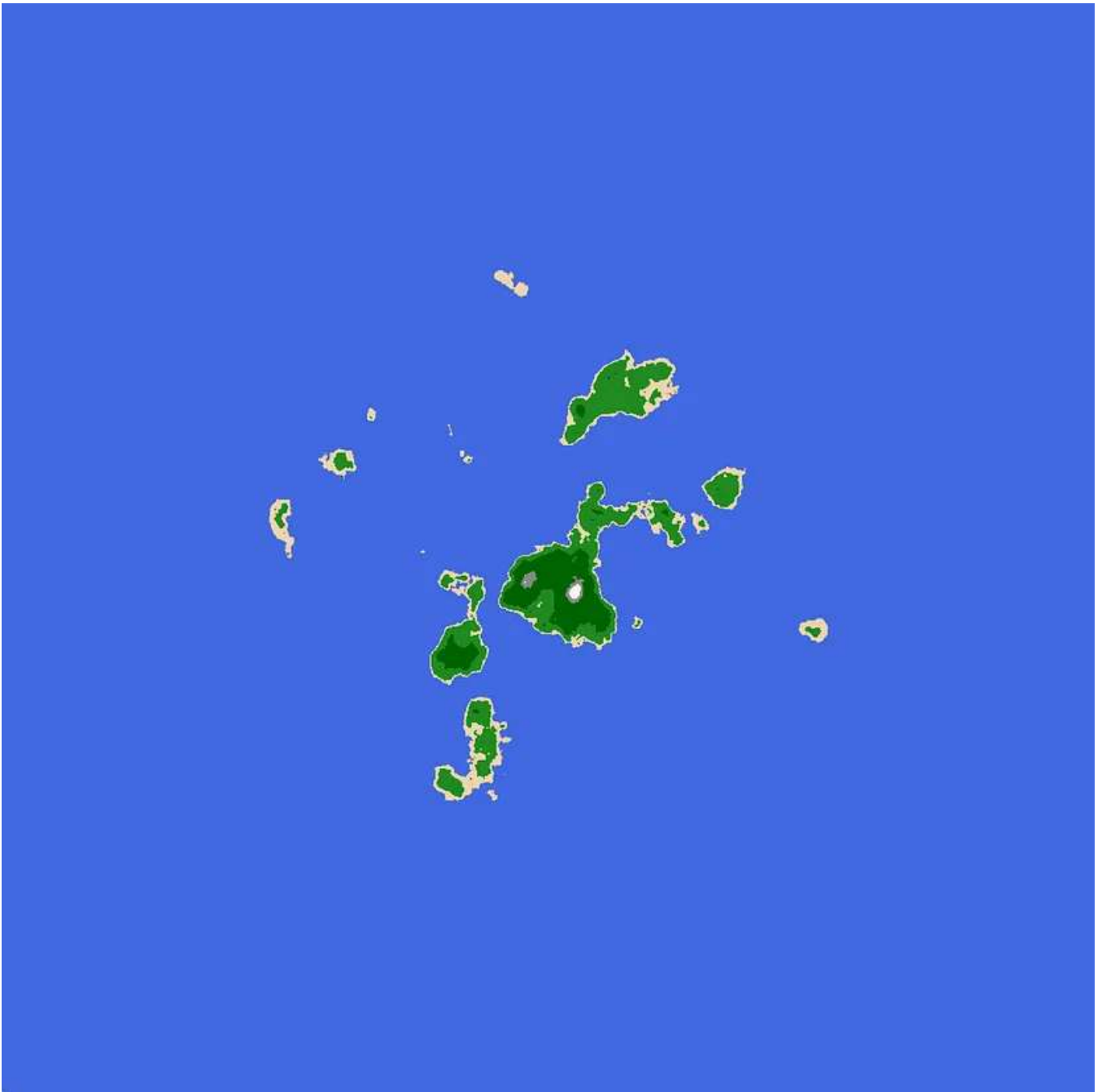


This part was less tricky but still a pain. I multiply the perlin noise by the circle gradient and then I increase the contrast by multiplying positive (lighter values) by 20. Then I renormalize to make it 0–1 again.

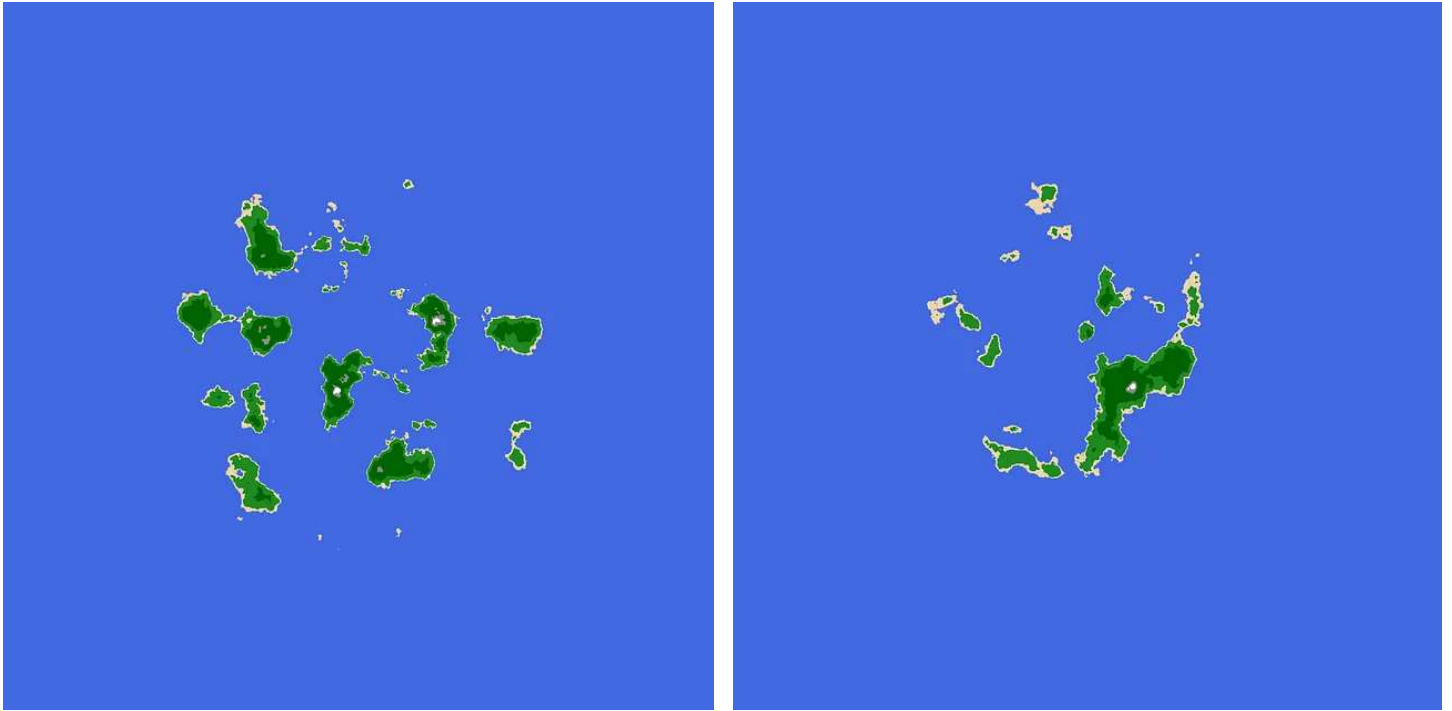
Colorized Circular Gradient + Noise:



This is really cool and it looks like a much more natural archipelago. I encourage you to try different shading methods and maybe randomly removing some sections. I'm going to change the threshold value and set it as `threshold = 0.2`. That will produce a smaller but more realistic archipelago as so:



There we are! We have a natural looking island archipelago! So now that we have our islands you may notice that no matter how often you rerun this script perlin noise will produce the same islands. To get new islands you can set the `base` parameter of the `pnoise2` function to a random integer number, let's try `base=5`, `base=100` :



Archipelagos generated from base 5 and base 100.

Conclusion

So we started with some simple noise and ended up with a way to generate a relatively unlimited number of unique and natural looking archipelagos! I hope you enjoyed this post!

[The full notebook with all code is here.](#)

If you enjoyed this post you can sign up for my newsletter here and stay up to date on cool and interesting projects.

Game Development

Mathematics

Programming

Art

Design



Follow

Written by Yvan Scher

294 Followers

Computers

More from Yvan Scher

```
00024  
00025 PROCEDURE DIVISION.  
00026 0001-MAIN.  
00027     INSPECT FUNCTION REVERSE(STR-1)  
00028           TALLYING WS-LEN1 FOR LEADING SPACES.  
00029     COMPUTE WS-LEN = LENGTH OF STR-1 - WS-LEN1.  
00030     DISPLAY WS-LEN.  
00031     MOVE 1 TO I.  
00032     MOVE WS-LEN TO J.  
00033     PERFORM REV-PARA WS-LEN TIMES.  
00034     DISPLAY STR-1.  
00035     DISPLAY STR-2.  
00036     GOBACK.  
00037 REV-PARA.  
00038     MOVE STR-1(J:1) TO STR-2(I:1).  
00039     SUBTRACT 1 FROM J.  
00040     ADD 1 TO I.  
00041     EXIT.  
***** Bottom of Data *****
```



Yvan Scher

7 cobol examples with explanations.

You may have heard COBOL before. If you search for it you will find images like this:

8 min read · Aug 1, 2018



479



6





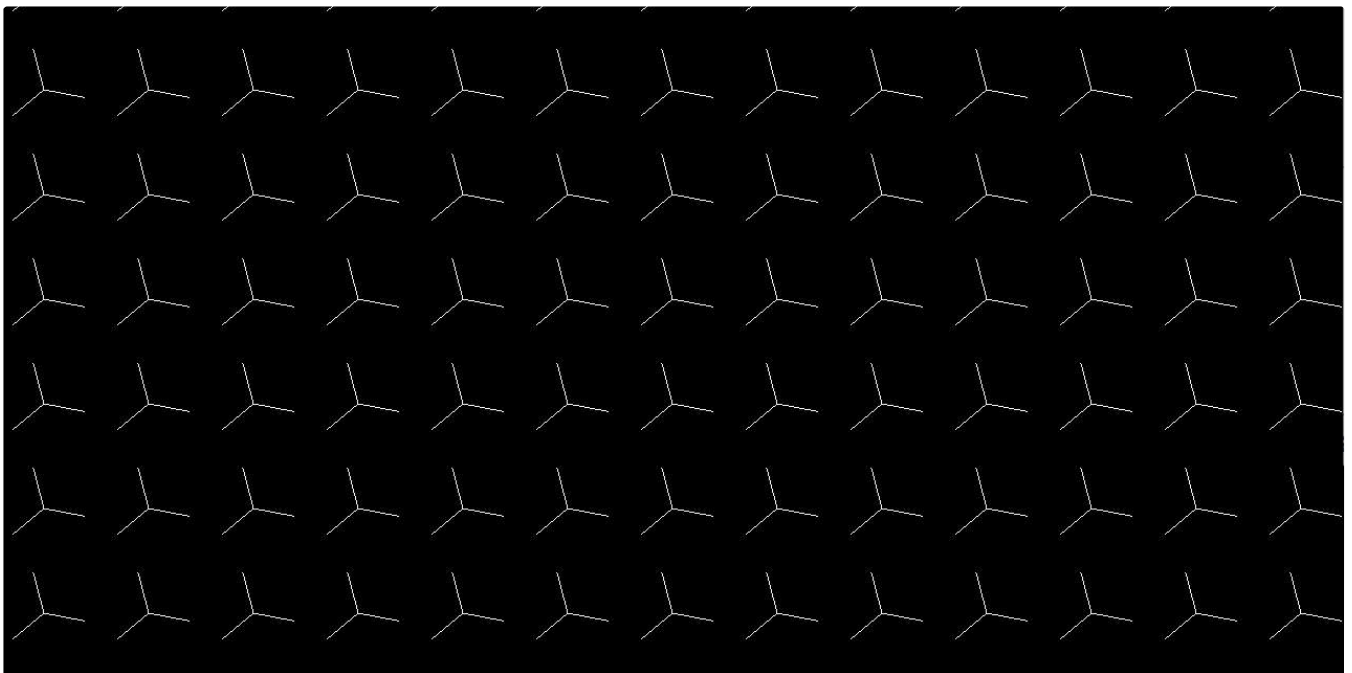
 Yvan Scher

Making a Game AI With Machine Learning

Making a mini game AI using pysc2 and Q learning

7 min read · Jan 5, 2018

 177 



 Yvan Scher

Opengl and pyglet basics

A pyglet micro-tutorial.

5 min read · Jan 31, 2018

 110

 2






Yvan Scher

Explanation of the c3d file format

A quick reference for anyone new to the format.

2 min read · Feb 4, 2018

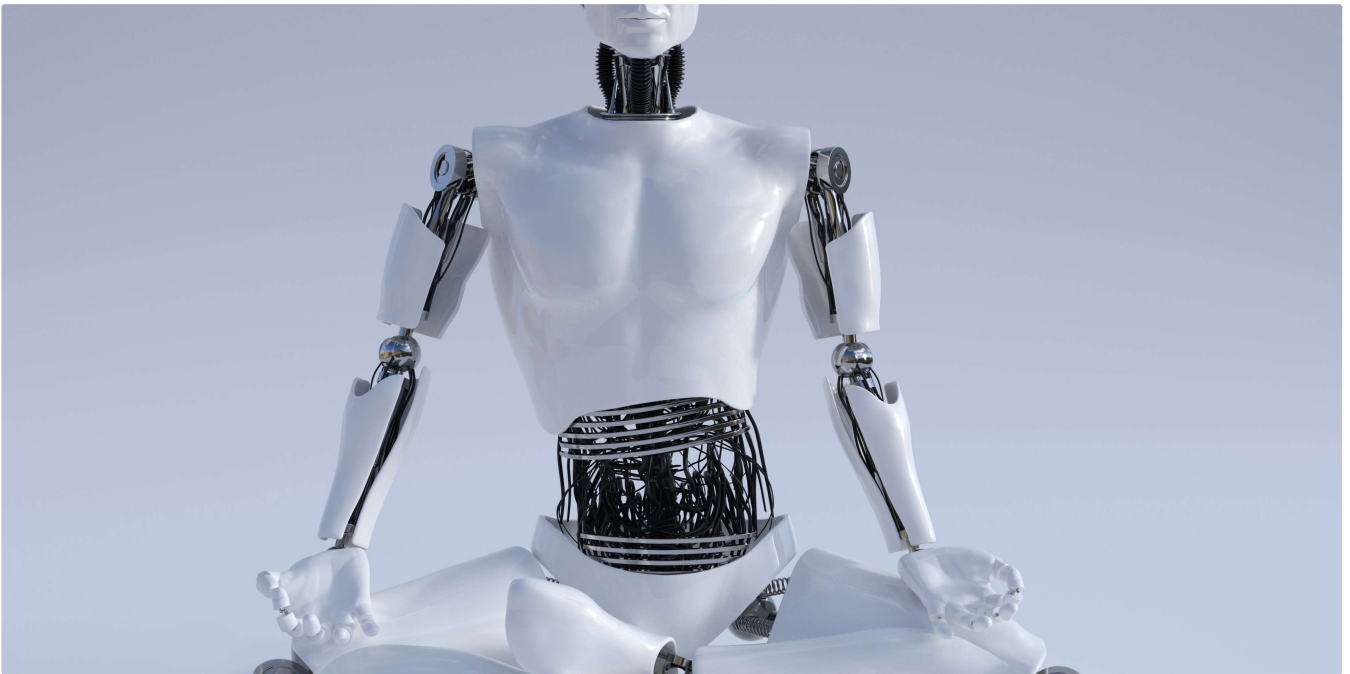
 11





See all from Yvan Scher

Recommended from Medium



The PyCoach in Artificial Corner

You're Using ChatGPT Wrong! Here's How to Be Ahead of 99% of ChatGPT Users

Master ChatGPT by learning prompt engineering.

🌟 · 7 min read · Mar 17



16.3K



286



Unbecoming

10 Seconds That Ended My 20 Year Marriage

It's August in Northern Virginia, hot and humid. I still haven't showered from my morning trail run. I'm wearing my stay-at-home mom...

★ · 4 min read · Feb 16, 2022



46K



744



Somnath Singh in JavaScript in Plain English

Coding Won't Exist In 5 Years. This Is Why

Those who won't adapt would cease to exist.

★ · 8 min read · Jan 20



7.6K



261





 Alexander Nguyen in Level Up Coding

Why I Keep Failing Candidates During Google Interviews...

They don't meet the bar.

★ · 4 min read · Apr 13

 3.1K  97



 Bryan Ye in Better Humans

How To Wake Up at 5 A.M. Every Day

An unconventional and compassionate guide to becoming an early bird

★ · 15 min read · Oct 3, 2019



85K



633



Tell me how ChatGPT works.



ChatGPT is a large language model that uses deep learning techniques to generate human-like text. It is based on the GPT (Generative Pre-trained Transformer) architecture, which uses a transformer neural network to process and generate text. The model is pre-trained on a massive dataset of text, such as books, articles, and websites, so it can understand the patterns and structure of natural language. When given a prompt or a starting point, the model uses this pre-trained knowledge to generate text that continues the given input in a coherent and natural way.



Molly Ruby in Towards Data Science

How ChatGPT Works: The Models Behind The Bot

A brief introduction to the intuition and methodology behind the chat bot you can't stop hearing about.

★ · 8 min read · Jan 30



7.3K



125



See more recommendations