# Generating 3D Terrains from 2D Noise Maps

Shannon Williams

Computer and Information Sciences

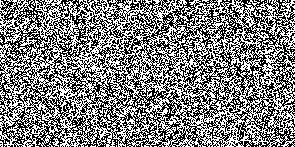
Harrisburg University of Science and Technology

***Abstract:*** *As part of the course requirements for CISC 400 - Computer Graphics, this paper will serve to aid my final project.*

*Keywords: noise, simplex, perlin*

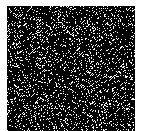
1. **Introduction**
2. **Background**
3. **Noise Height Maps**

Started with simple random generation of pixels.

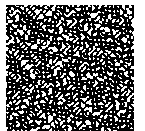


Noise generation ( Math.round( Math.random() % 32768) \* 255)

Then resorted to using simplex2 noise. Started with regular noise(x,y):



By giving a “zoom” appearance with factor of 8; I.e noise2(x/zoom, y/zoom):

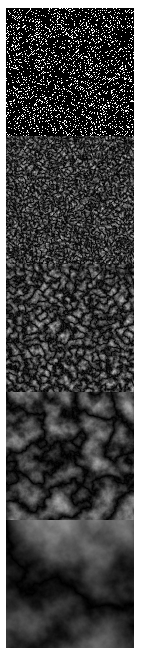


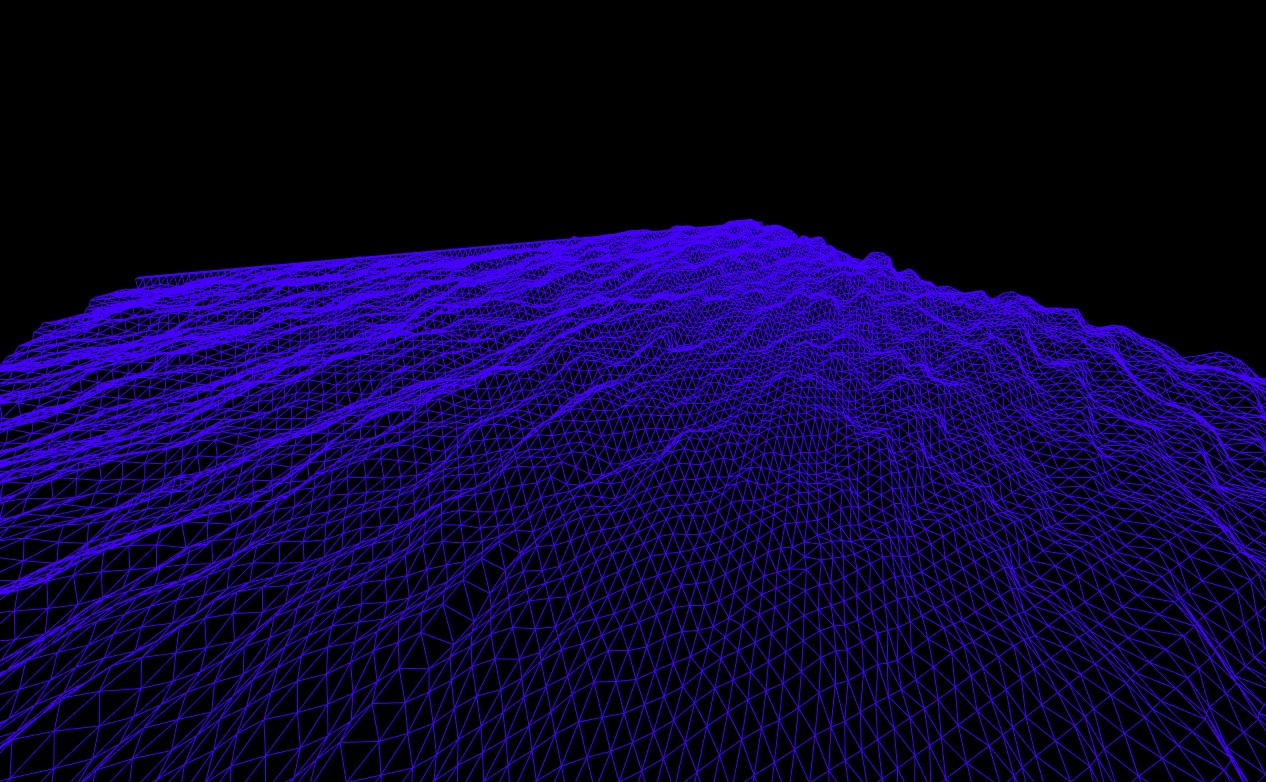
We can see we achieve a zooming effect. Lets try a zoom of 64.



What if we begin to combine these together? Now show concept of turbulence (achieving grayscale and not just black or white). See written notes for more details.

Now think of increasing the iterations. Again see notes for details. Below is a side-by-side comparison of images with turbulence with zoom factors of 1, 8, 16, 64, N=256, respectively.



Now using the same algorithm to set the height of vertices:

Applying texture in Three.JS

Creating thresholds for water, dark grass/mud, grass, dirty snow, and pure snow mountain tops. (Don’t forget to attempt adding gradients for smooth color transitions)

After getting the height for the vertice, we can set values to color our terrain based on the height. Here’s my code to achieve the mentioned:

function setTerrainTexturePixel(c, image, pxi){

let color = Math.round(c \* 255);

let gradient = THREE.Math.clamp(c,0,1);

if(color <= 50){ // give Water color

image.data[pxi] = 7;

image.data[pxi+ 1] = 72;

image.data[pxi +2] = 234;

}

else if( color >= 50 && color <= 100){ // give DARKER grass color

image.data[pxi] = 1 ;

image.data[pxi+ 1] = 33;

image.data[pxi +2] = 22;

}

else if(color > 100 && color <= 200){ // give grass color

image.data[pxi] = 3 ;

image.data[pxi+ 1] = 73;

image.data[pxi +2] = 52;

}

else if (color >200 && color <= 250){ // DIRTY snow

image.data[pxi] = 200 ;

image.data[pxi+ 1] = 200;

image.data[pxi +2] = 200;

}

else{ // pure white snow

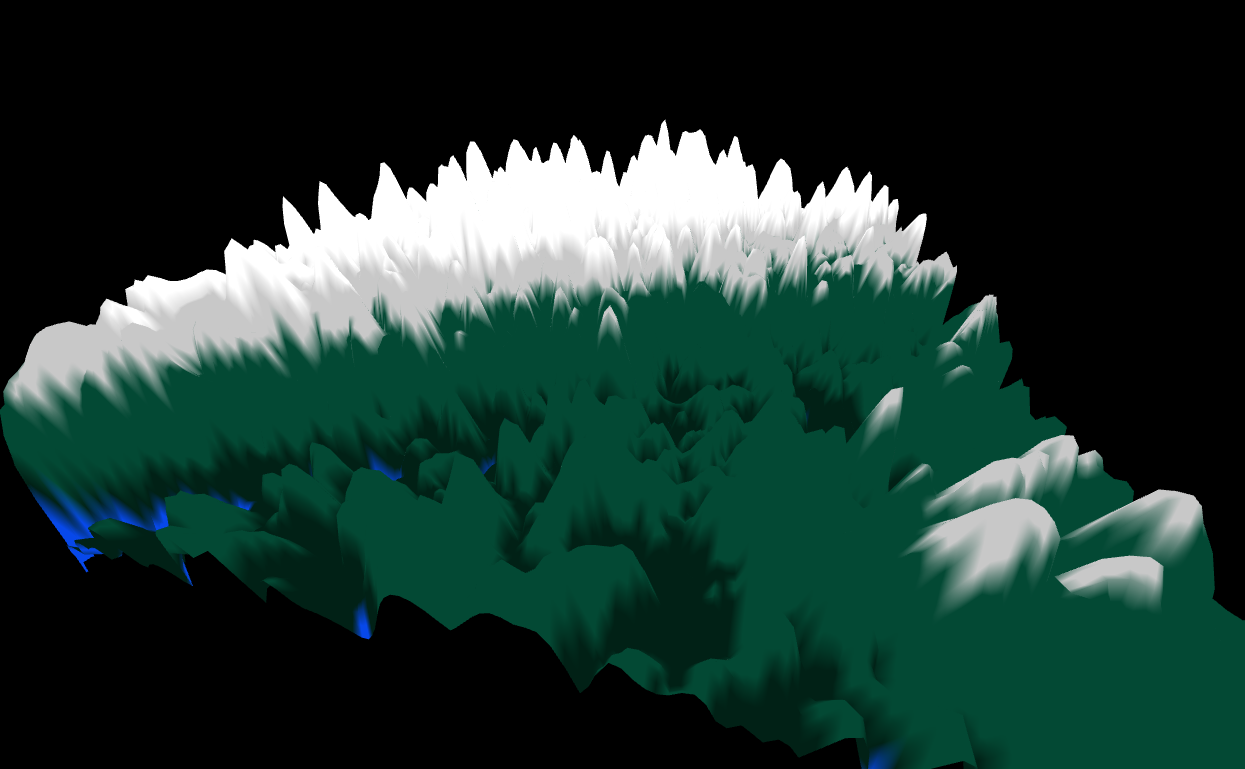
image.data[pxi] = image.data[pxi+1] = image.data[pxi+2] = 255;

}

image.data[pxi+3] = 255;

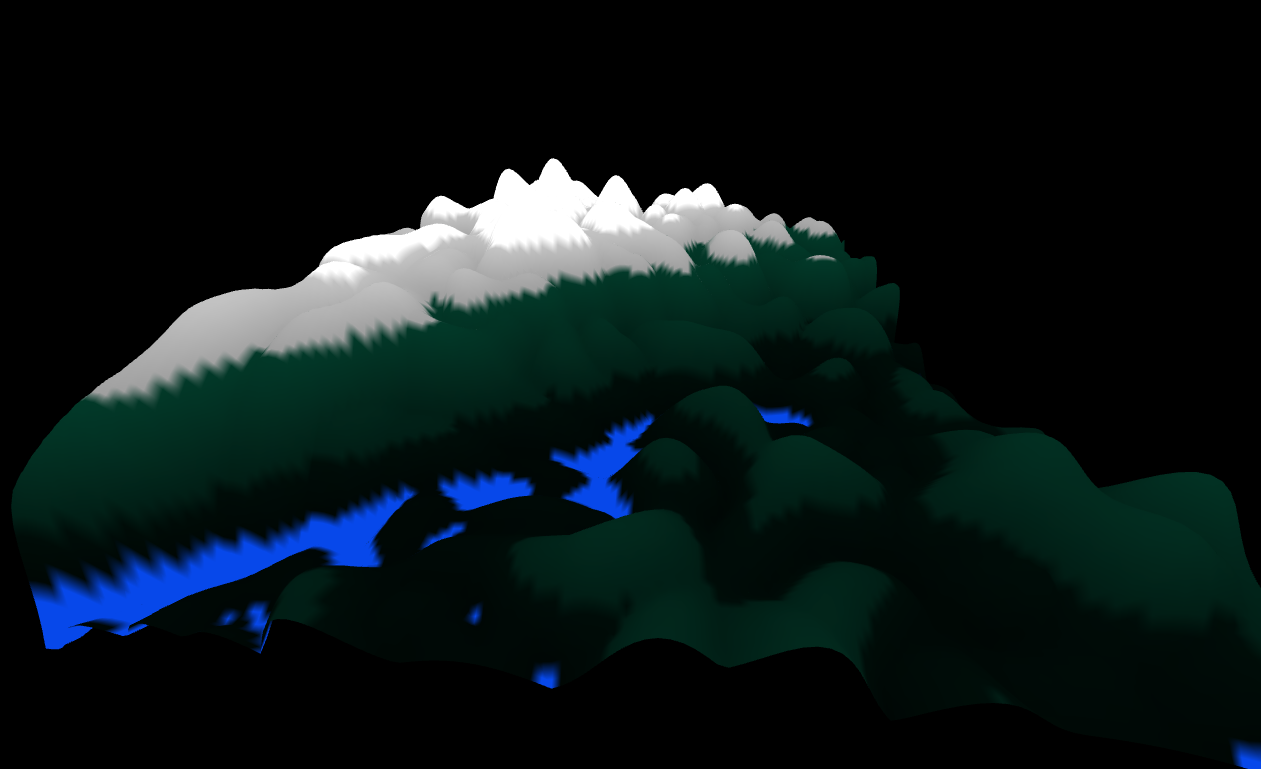
}

Here is the output of doing that:



Now we can add a gradient factor by simply multiplying by the passed “c” parameter (height). Notice that the height can be used to serve several abstract concepts, like in this case, gradient. The value must be clamped before use. By multiplying by “c”, which will be called “gradient” from this point, will linearly darken or lighten the color for each threshold.

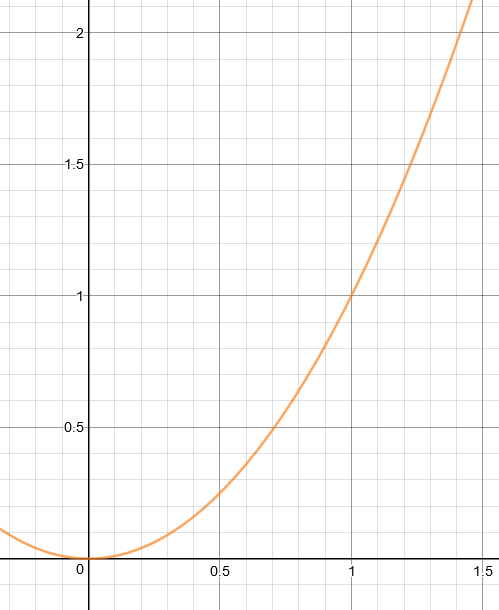
Output for this is:



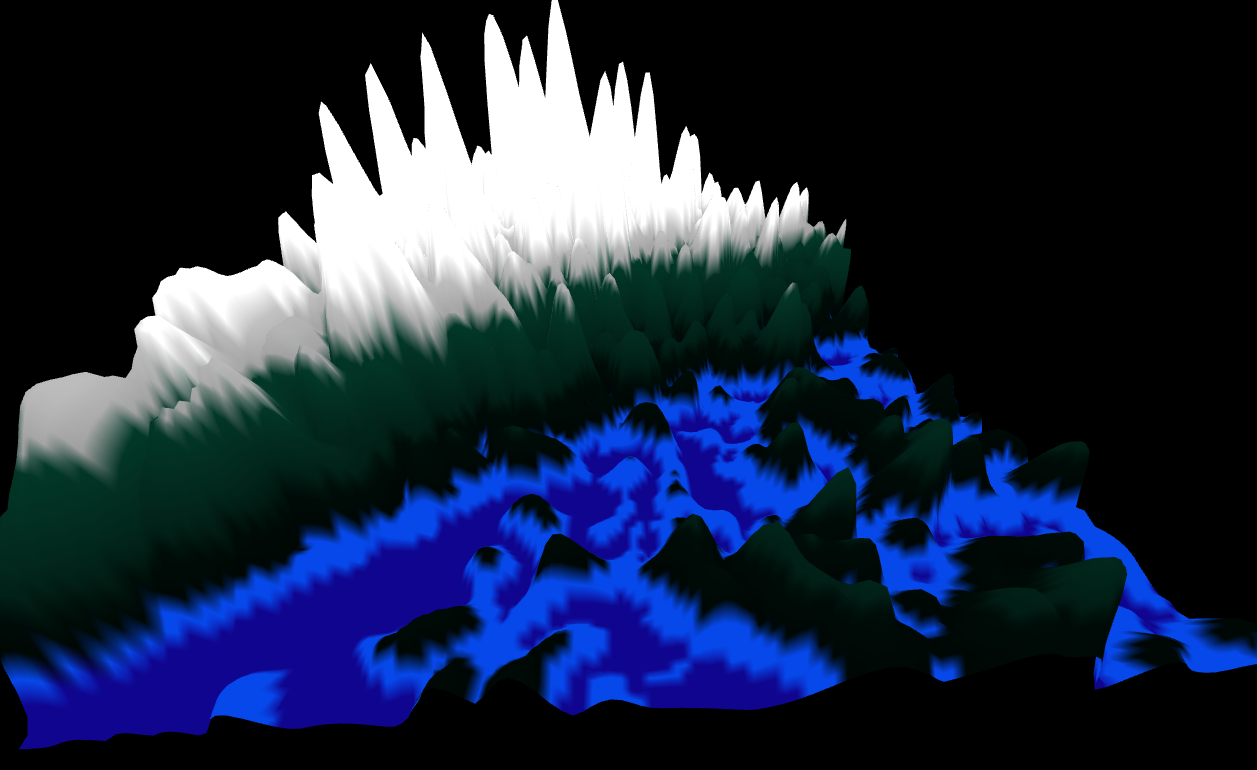
**(Optional Enhancements)**

**Using various functions to control mountains**

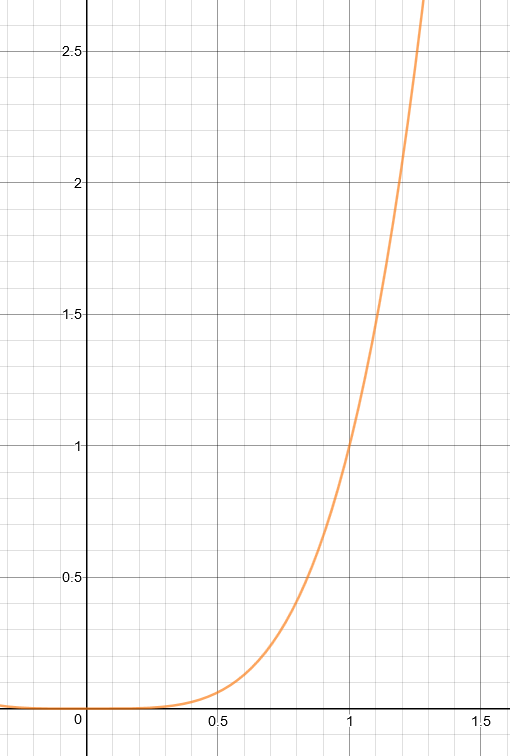
Exponential: Notice an exponential curve? It affects the probability and shape of the mountainous landscape.

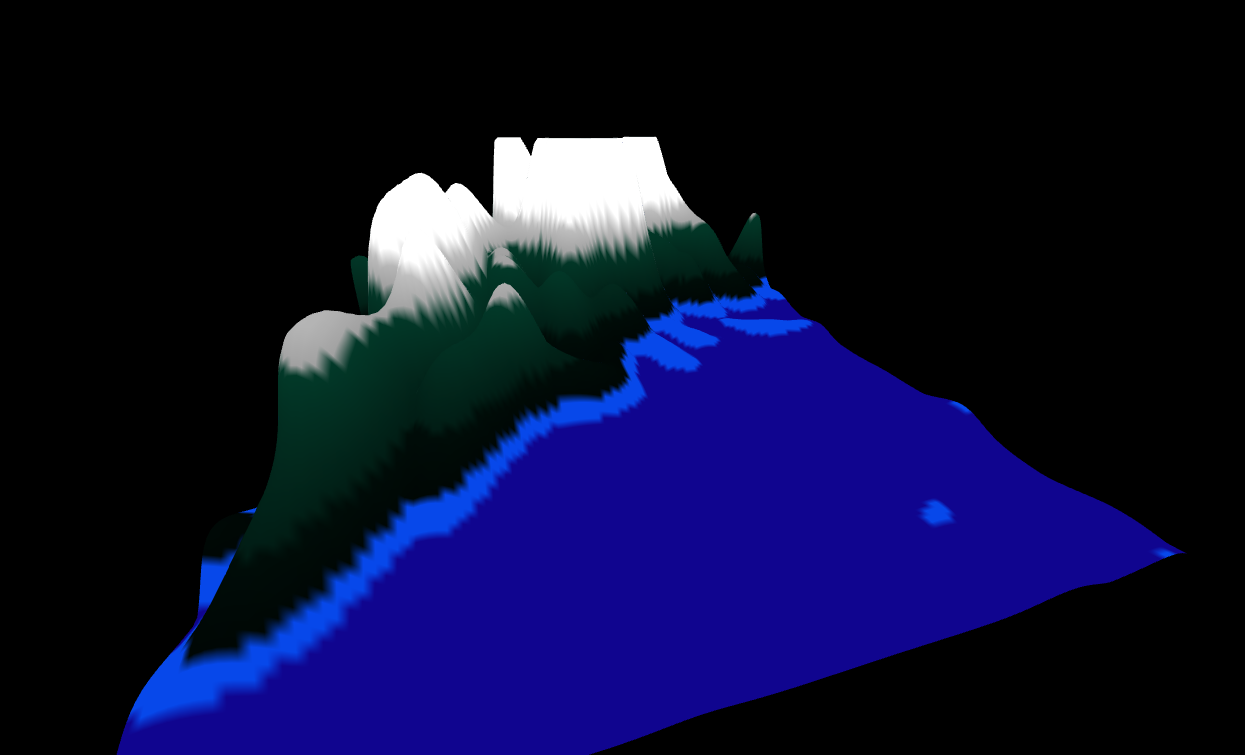


Now applying this the height of each vertice by switching x with y 🡪 using pow(y,2) to get the new height. Additionally, I added a clamp to 2.5 to avoid very high mountains and to have a flat mountain top if above.



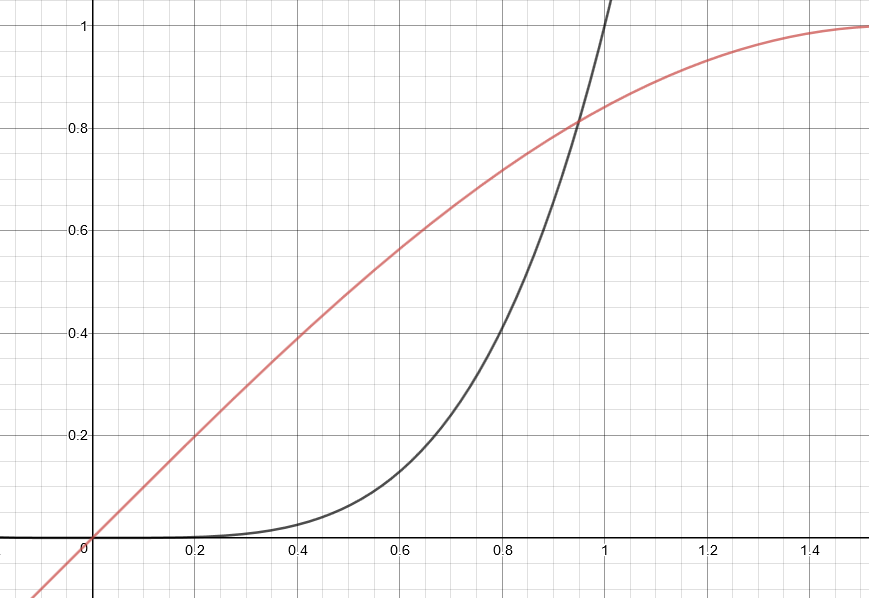
Using pow(y,4):





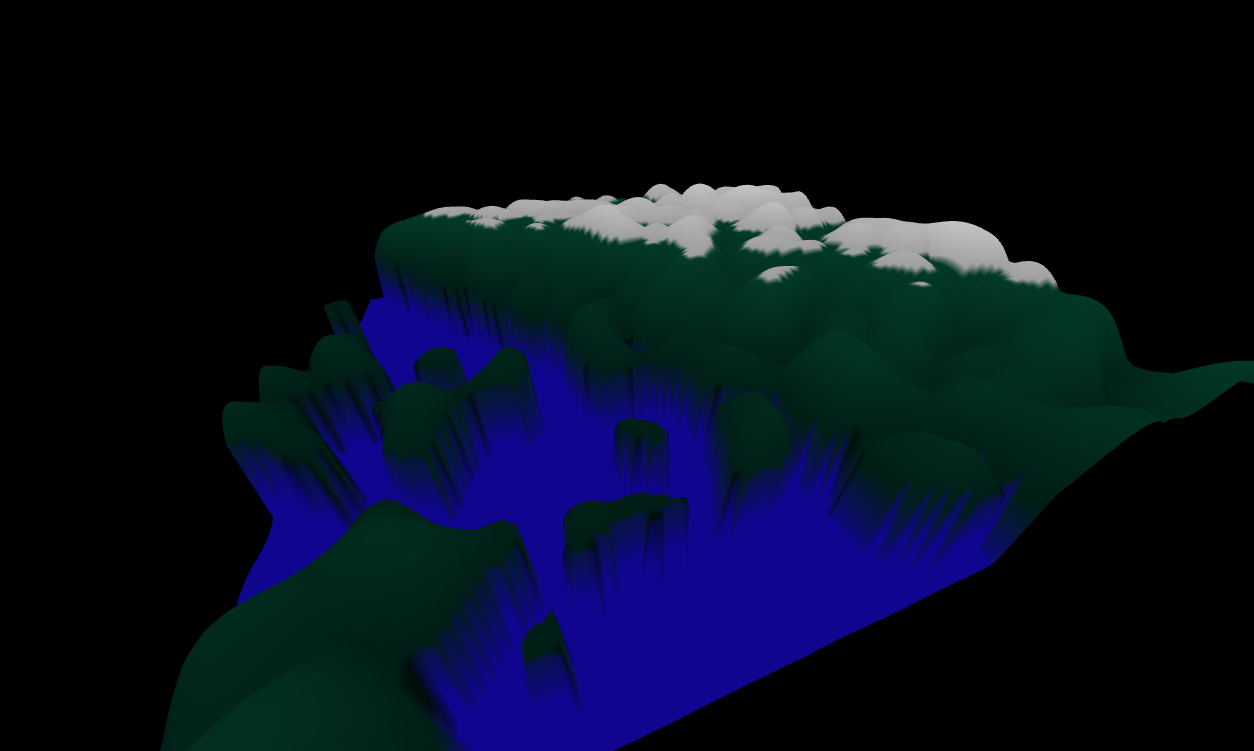
To get the best output, we can introduce piecewise functions!

Using the piece wise function:



Pow(y,4) if y <= 0.4

Sin(y) if y > 0.4



<http://lodev.org/cgtutor/randomnoise.html>

<https://www.redblobgames.com/maps/terrain-from-noise/>

<https://github.com/josephg/noisejs> for perlin.js