3D Terrain Generation Using Noise

Mitchell Talyat, Greg Carter

What is procedural terrain generation?

- Generate terrain pseudo-randomly
- Emulate the real world without modeling
- Infinite replayability
- Endless possibilities

What & Why

What is noise?

- Data generated by a stochastic (random) process
- Colors of noise (due to average shape of frequency)

What is (Perlin) Noise?

- A random sequence producing “a more natural succession of numbers”.
- Basically a smoother gradient of random numbers

Why do we use it?

- Modifying other data values to different degrees
- Using normal random function can generate points that are too far apart or too close while being the exact opposite in a different dimension
- Smoother numbers can make for more natural looking generation and transition between areas

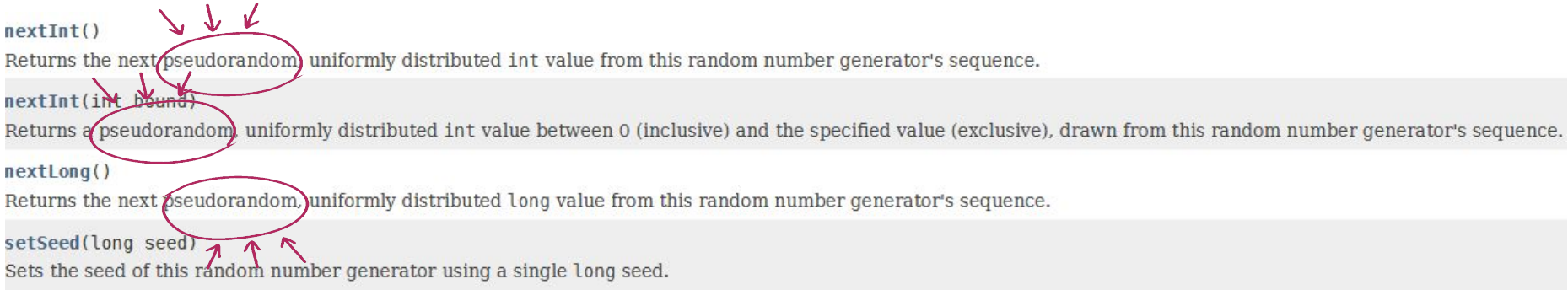
Noise in Generation

Seeds

- Allows for reproducible random number generation
- Good for video games, especially for testing

Random Function (Stochastic Process)

- A series of random values generated over time
- Most programs use a “Pseudo-random” function that is based on the seed



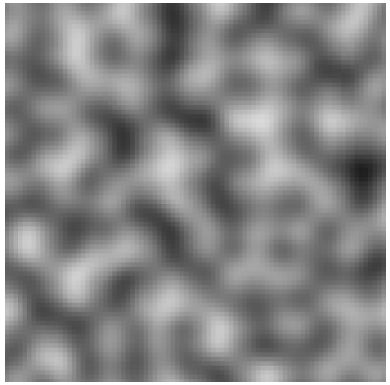
nextInt()
Returns the next pseudorandom, uniformly distributed int value from this random number generator's sequence.

nextInt(int bound)
Returns a pseudorandom, uniformly distributed int value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.

nextLong()
Returns the next pseudorandom, uniformly distributed long value from this random number generator's sequence.

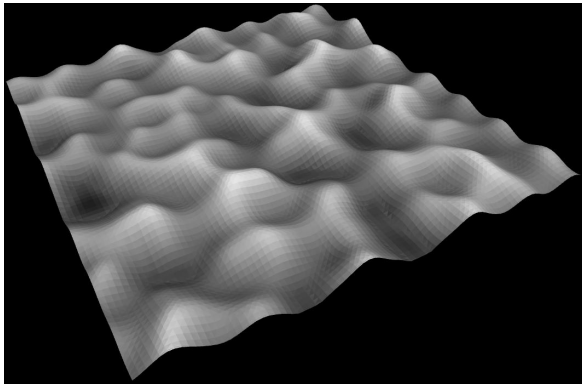
setSeed(long seed)
Sets the seed of this random number generator using a single long seed.

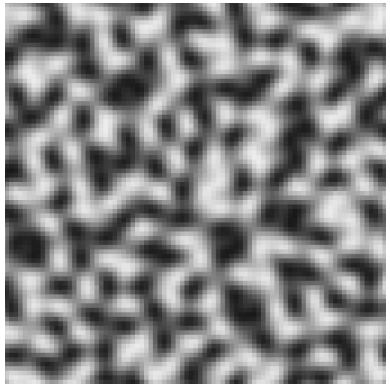
Common Noise Types



Perlin

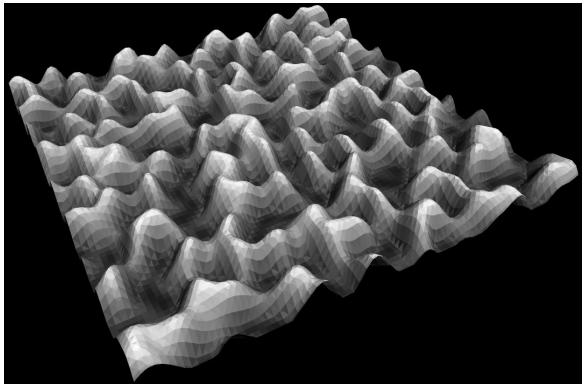
- “Classic” noise implementation
- Smooth & wavy



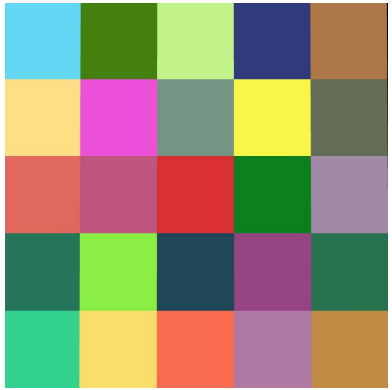


Open Simplex 2

- Faster
- Better runtime
- Sharper hills

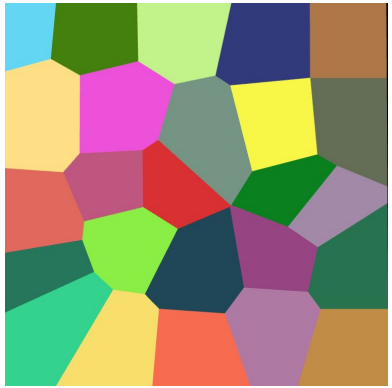


Common Sampling Methods



Uniform Grid

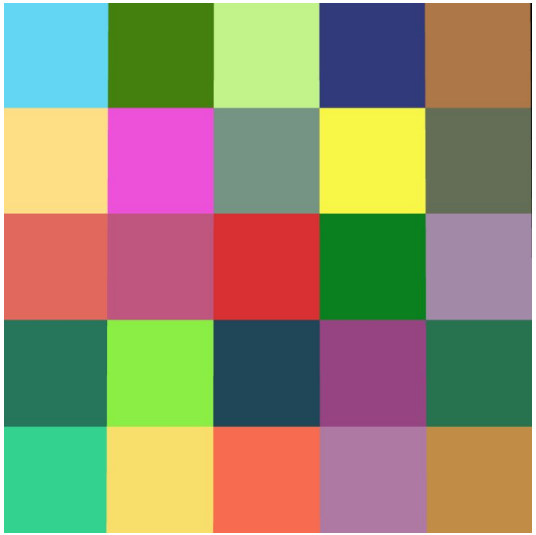
- Classic implementation
- Easy to do
- Directly based on noise at uniform intervals



Voronoi Polygons

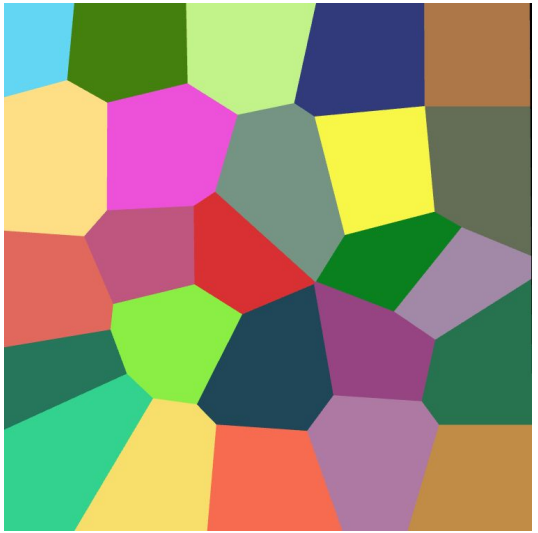
- Based on random points
- Uniform with variation

Polygon Sampling



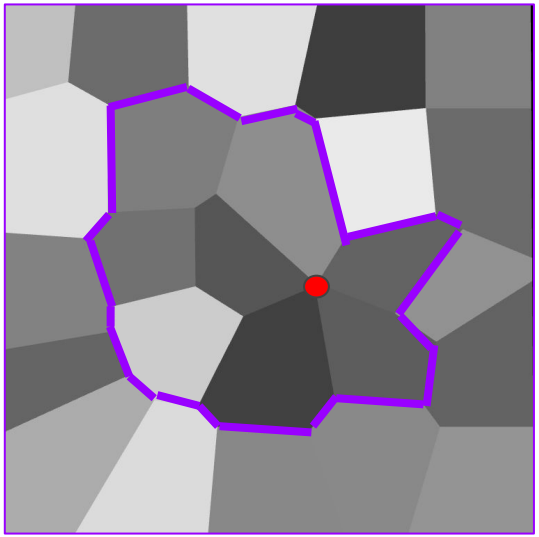
Uniform Grid

Random Voronoi Polygons

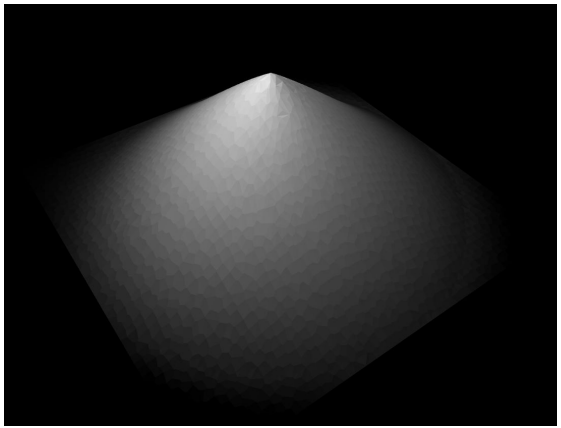


Random Voronoi Polygons with Lloyd's Algorithm

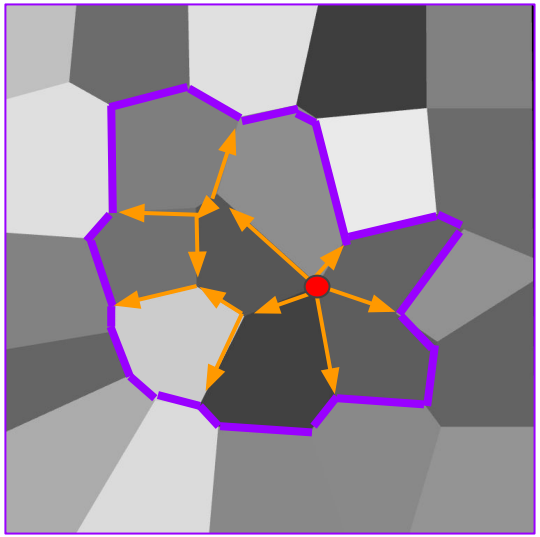
Basic Polygon Island Generation



Define the bounds of the shape



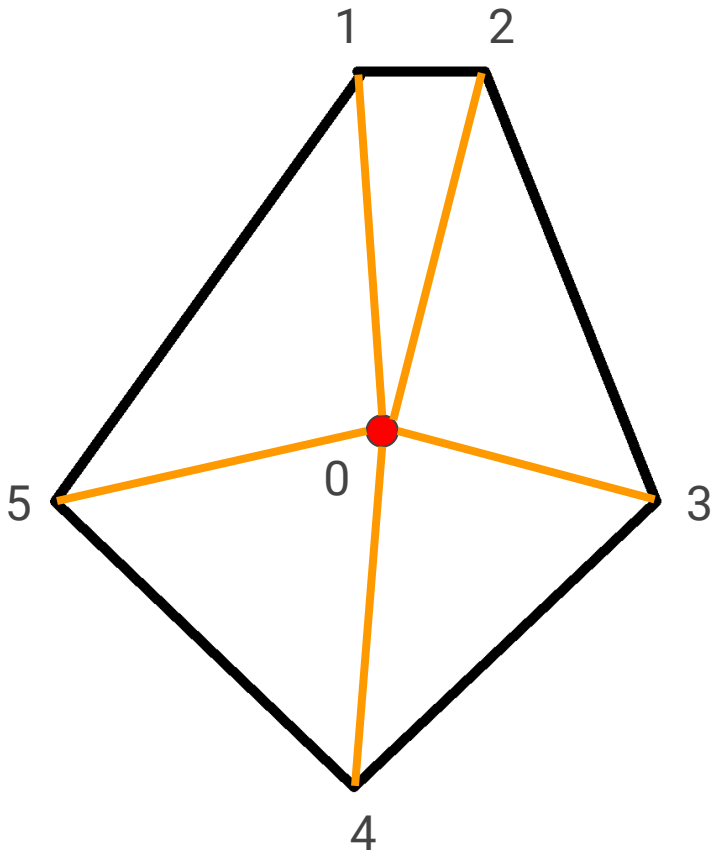
Create Height Map & choose highest point (red)



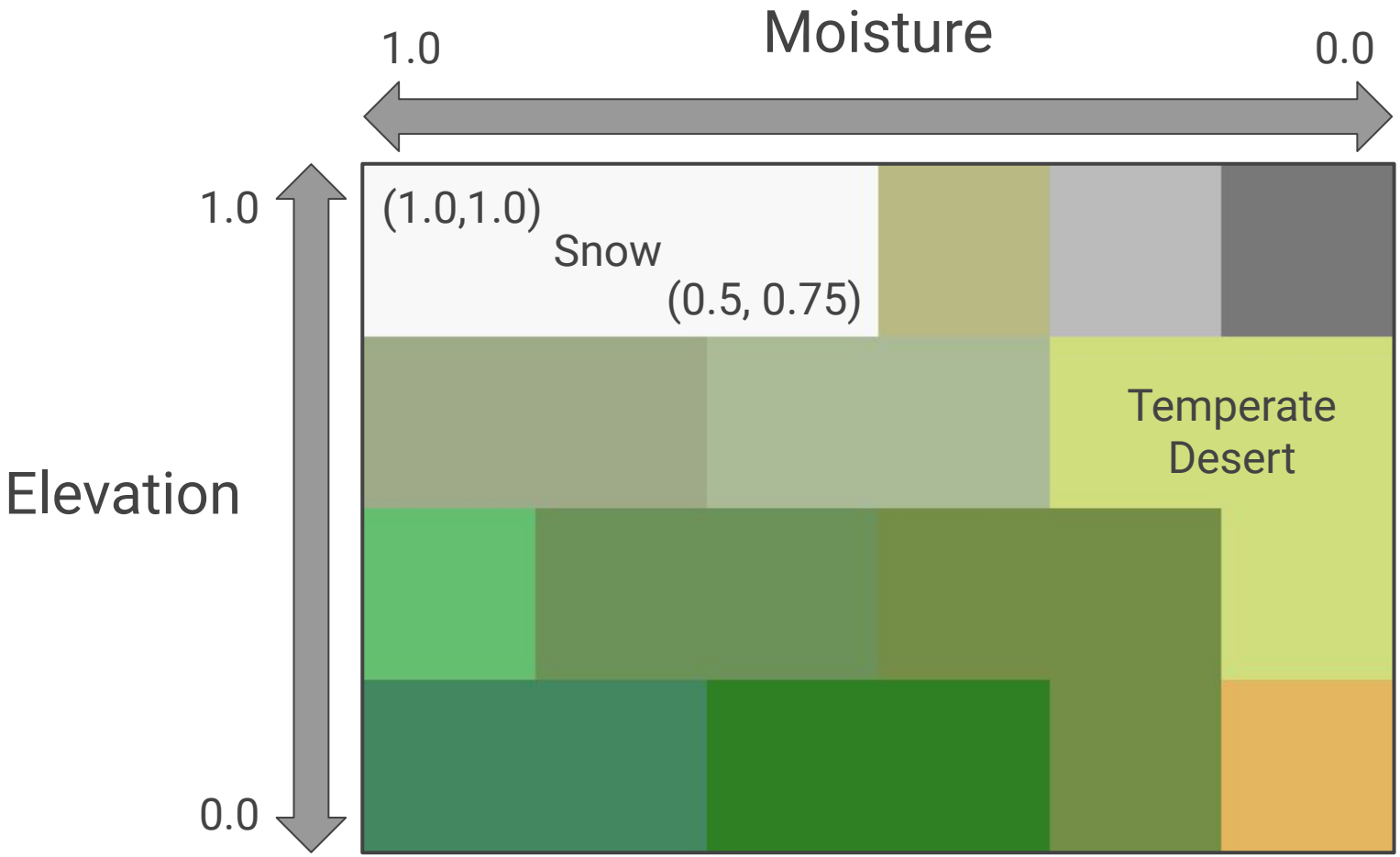
Use height map to generate downslope connections and heights of other points

Polygon Meshes

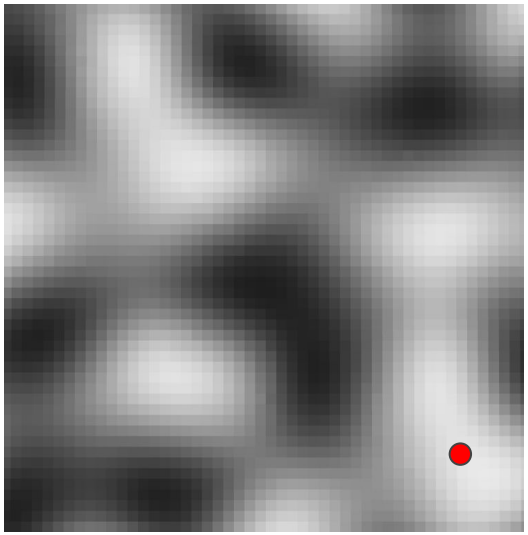
- Just need a list of points for each polygon
- Find center by averaging them
- Index in order, reusing center - known as a triangle fan
- Create triangle for each point, connected to center
- 0, 1, 2, 0, 2, 3, 0, 3, 4...
- Triangulation!



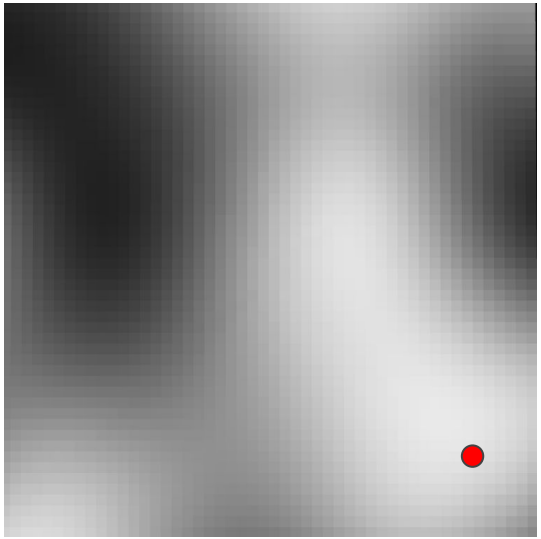
Biomes


1.0
Moisture
0.0
1.0
(1.0,1.0)
Snow
(0.5, 0.75)
Temperate
Desert
Elevation
0.0

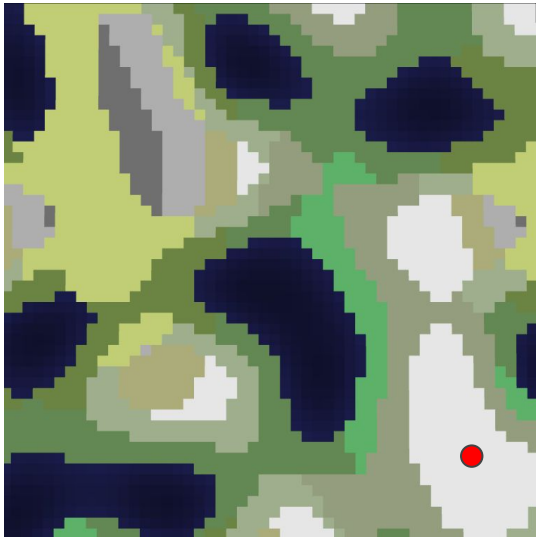
Biome Sampling



Height Map

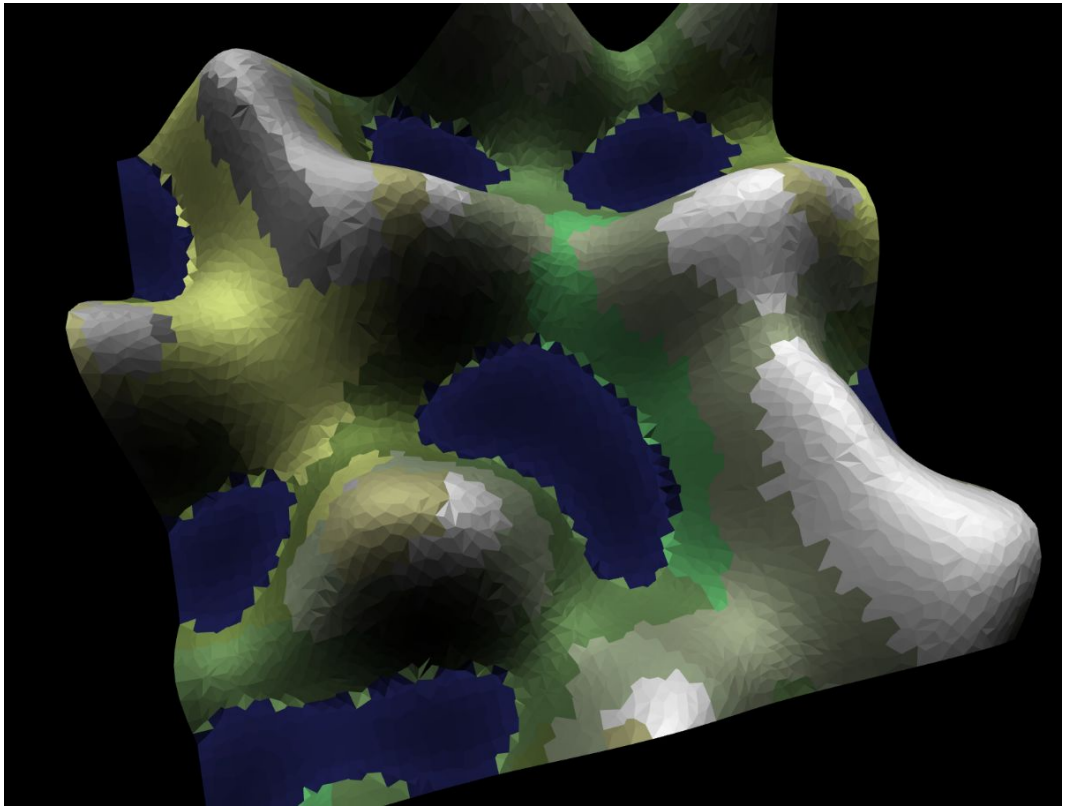


Moisture Map



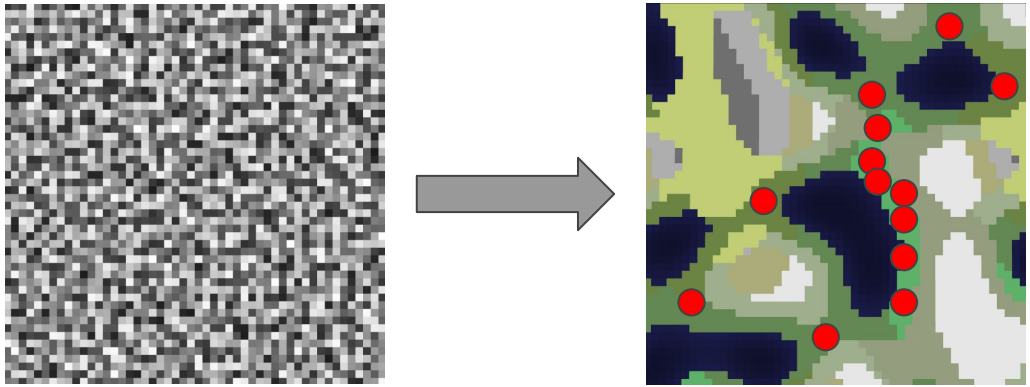
Biome Map

Biome Sampling - Result



Object (Tree) Spawning

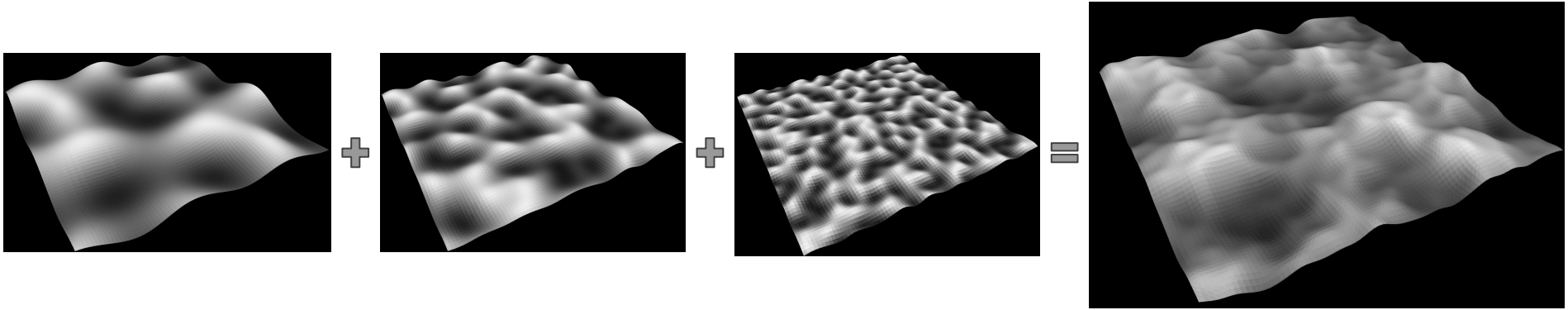
- Sample noise at a high frequency
- Check on a grid for values over some threshold
- Place object if true
- Different per biome



Fractal Brownian motion (fBm)

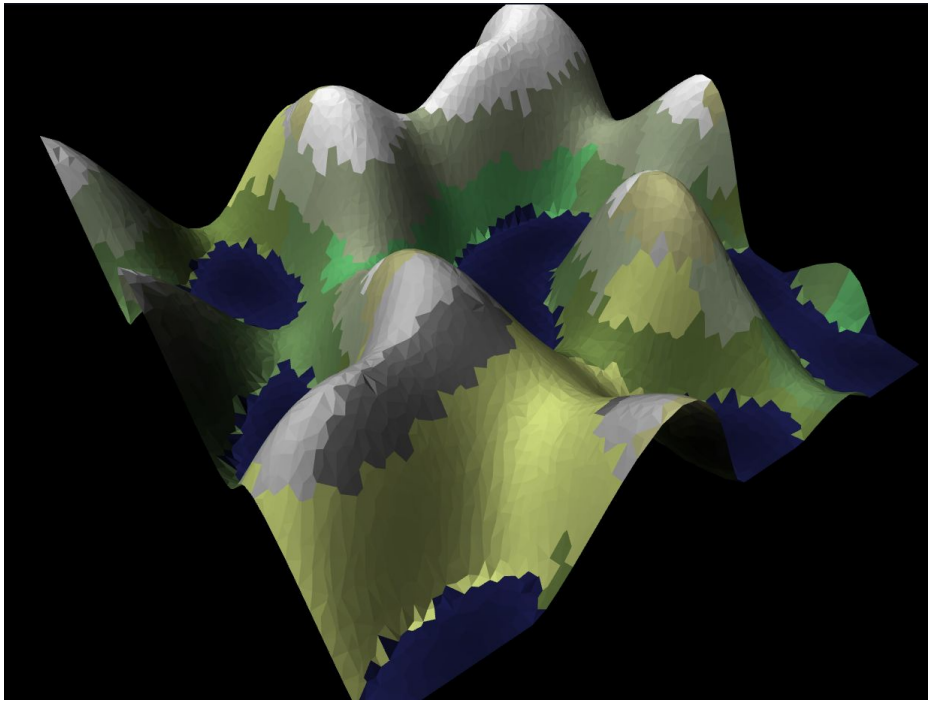
- Makes terrain look more natural
- Most terrain isn't smooth like noise is

- fBm applies multiple layers of noise on top of each other
- More octaves is more expensive, but more detailed results



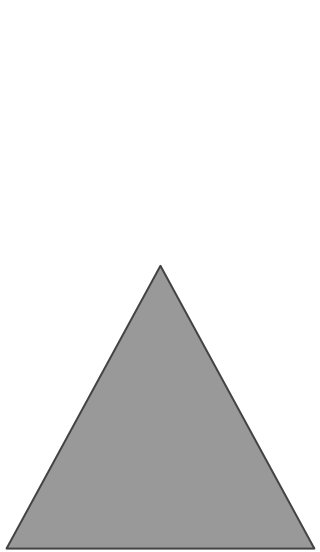
Noise Adjustments

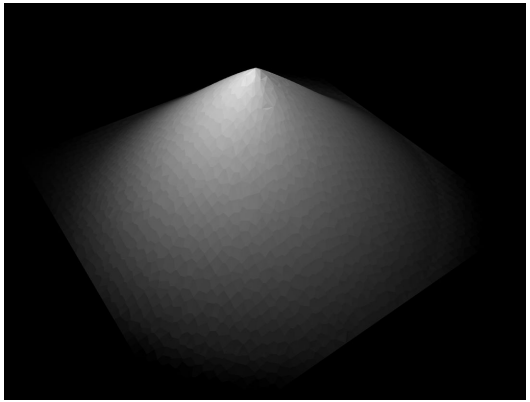
- Noise can be manipulated to create terrain for unique circumstances
  - Normal
  - Island shape
  - Wrapping X and/or Y axes
  - Etc.

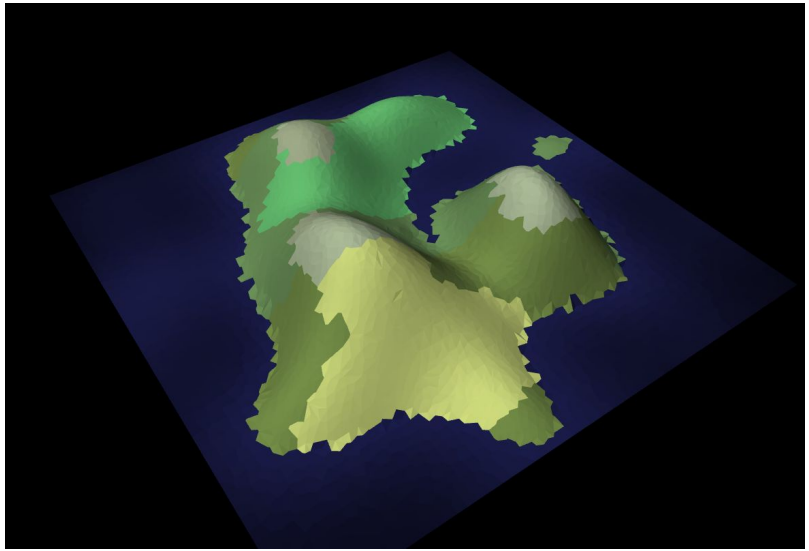


(Unaltered terrain)

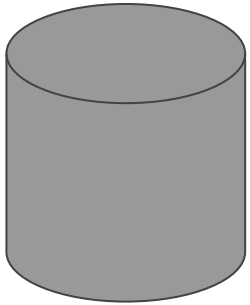
Noise Adjustments - Island



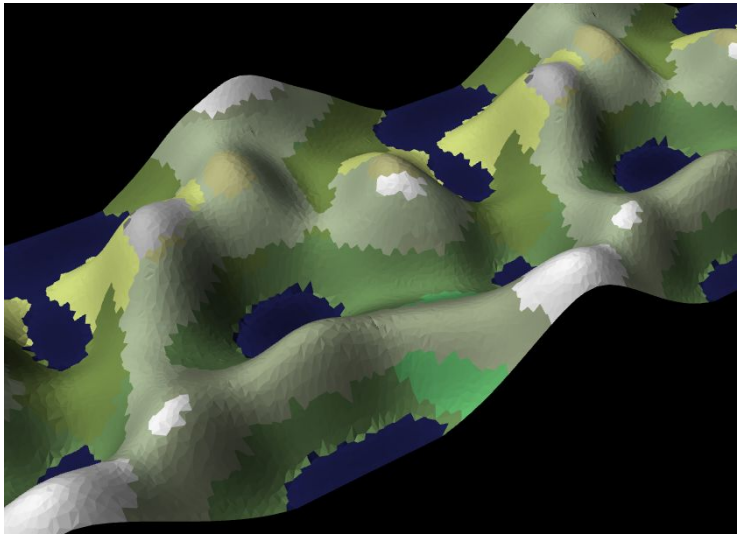




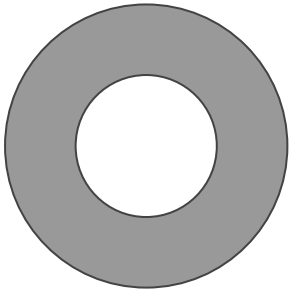
Noise Adjustments - Wrap Single Axis

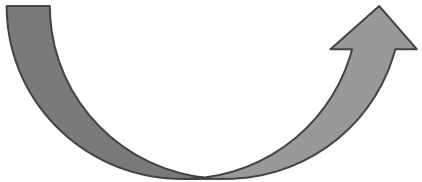


```
// cylinder wrap on X axis
function getNoiseWrapX(noise, x, y) {
    const pi2 = Math.PI * 2;
    const angle = pi2 * x;
    return noise.GetNoise(
        Math.cos(angle) / pi2,
        Math.sin(angle) / pi2, y)
        * 0.5 + 0.5;
}
```

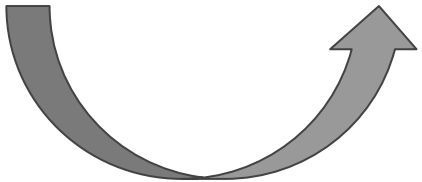


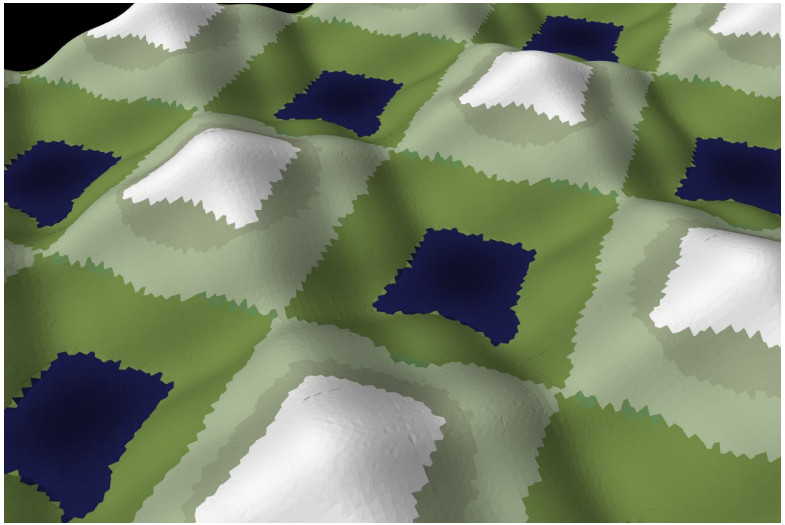
Noise Adjustments - Wrap Both Axis





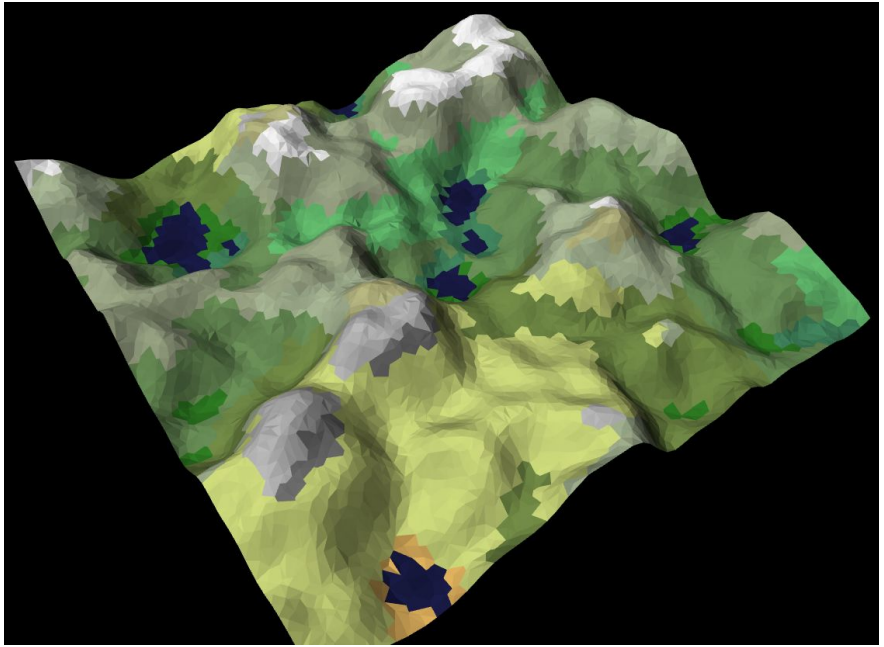
```
// torus wrapping (both X and Y)
function getNoiseWrap(noise, x, y) {
    const pi2 = Math.PI * 2;
    const angleX = pi2 * x;
    const angleY = pi2 * y;
    return getNoise4D(noise,
        Math.cos(angleX) / pi2,
        Math.sin(angleX) / pi2,
        Math.cos(angleY) / pi2,
        Math.sin(angleY) / pi2)
        * 0.5 + 0.5;
}
```



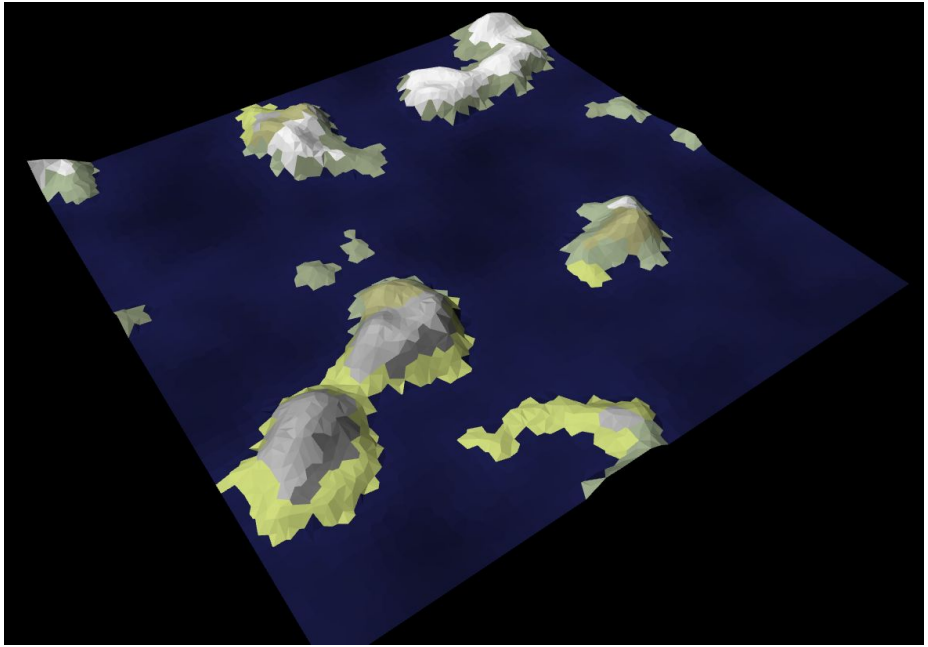


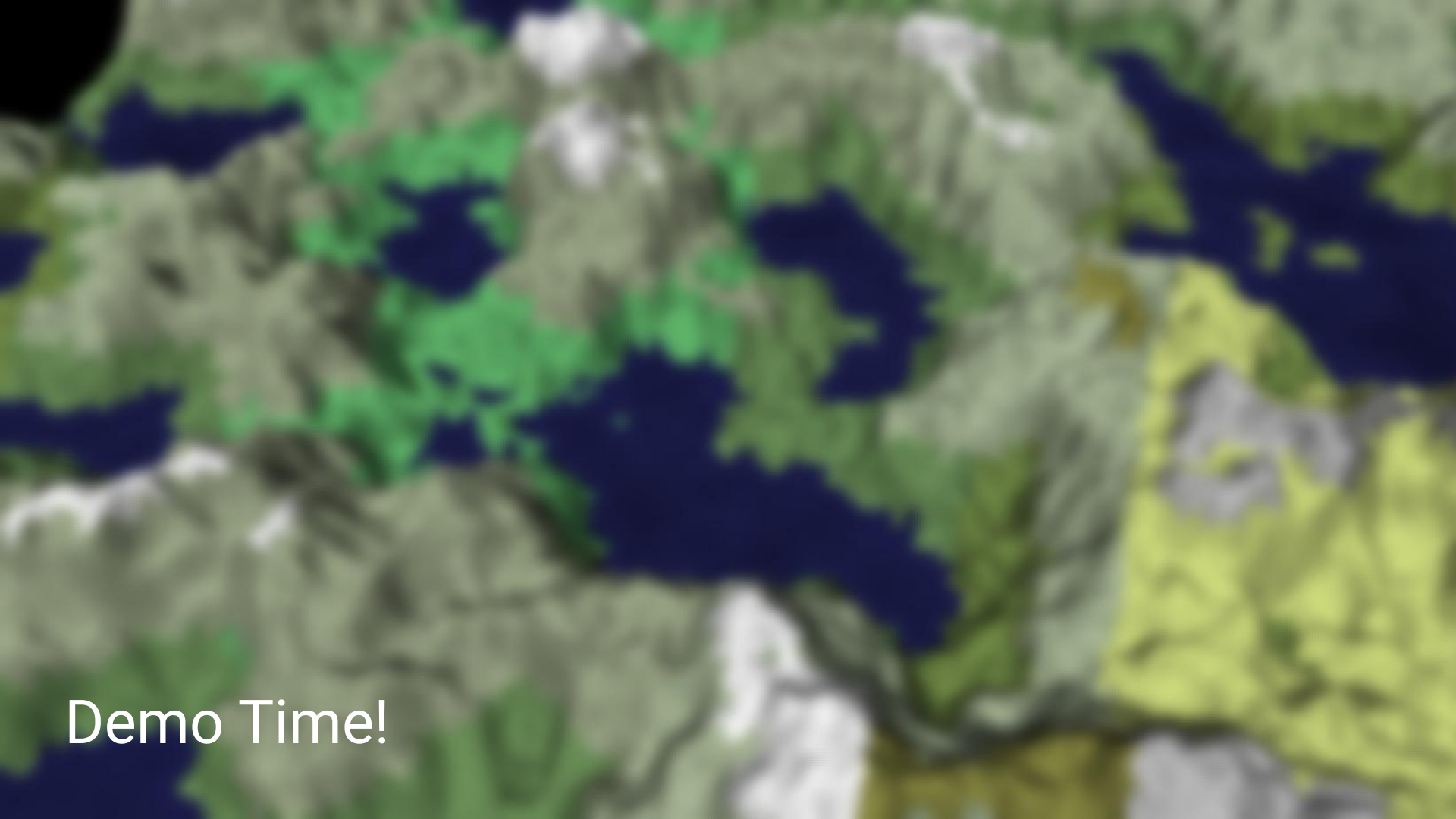
Sea Level

Lower = Less Water



Higher = More Water/Islands




Demo Time!

References

Images:

https://www.nintendo.com/en-gb/Games/Nintendo-Switch-download-software/Terraria-1424601.html

https://medium.com/the-indie-system/realm-of-the-mad-god-3-10-fc653249b739

https://www.imdb.com/title/tt2011970/mediaviewer/rm3864135680

https://wccftech.com/valheim-newcomers-heres-a-handful-of-tips-and-tricks-from/

Information:

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

http://www-cs-students.stanford.edu/~amitp/game-programming/polygon-map-generation/