New Noise Function Discovery

I explored two different types of noise generative methods. The first of which was a modification of Worley (or Voronoi) noise with Bezier curves while the second dealt with the exploration of prime number patterns. Additionally, I explored the impacts of warping domains.

Modified Worley Noise

Worley noise generates noise by placing random cells on the screen and calculating the distance between a given point and each cell. There are 4 types of distances that were observed:

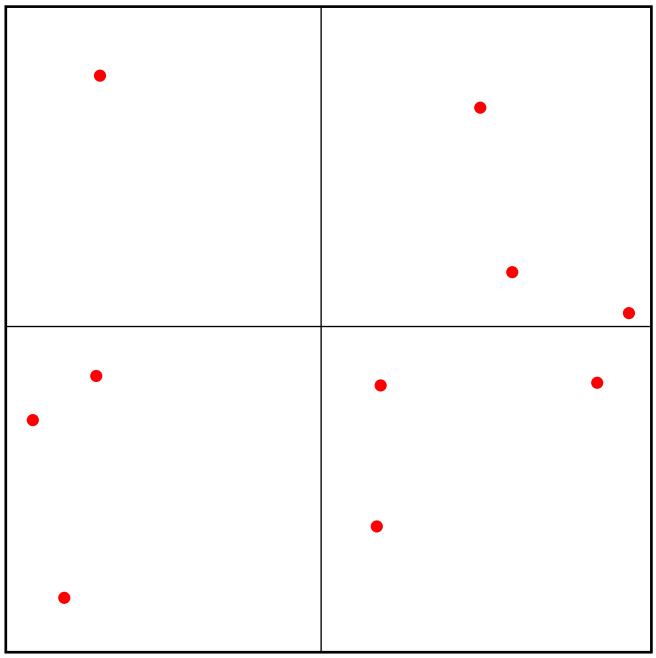
1. Euclidean Distance
   $ED = ((x_2 - x_1)^2 + (y_2 - y_1)^2)^{1/2}$
2. Manhattan Distance
   $MD = |(x_2 - x_1)| + |(y_2 - y_1)|$
3. Chebyshev Distance
   $CD = \max(|(x_2 - x_1)|, |(y_2 - y_1)|)$
4. Mahalanobis Distance
   $MHD = ((vec3_2 - vec3_1)^T S^{-1} (vec3_2 - vec3_1))^{1/2}$
   S = Covariate matrix

With each of the types of distances, output was observed with respect to using the following 5 measures:

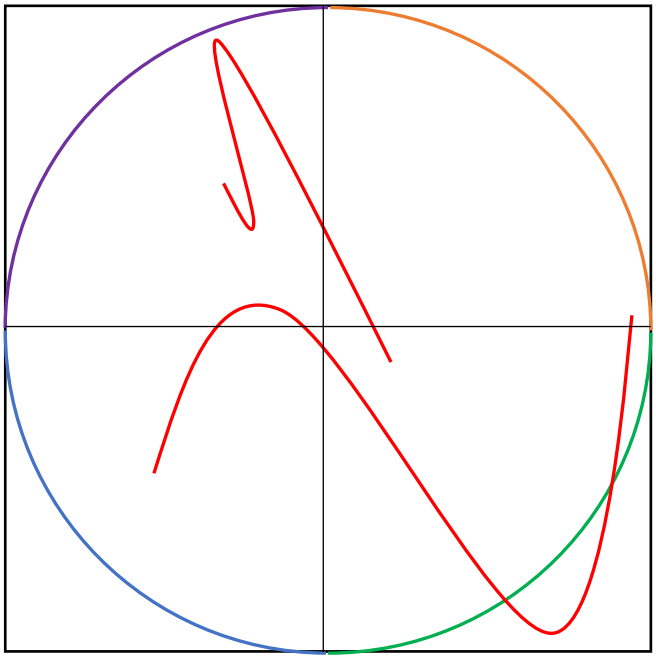
1. Minimum distance
2. Second minimum distance
3. Second minimum distance minus first minimum distance
4. Third minimum distance
5. Average of all first, second and third minimum distances

These measures can be utilized in conjunction with Bezier curves (thus replacing cells). However, given the nature in which points on a Bezier curve are calculated, the cost to calculate drastically increases with the precision in which you walk along the Bezier curve.

Traditional Worley with 10 Cells

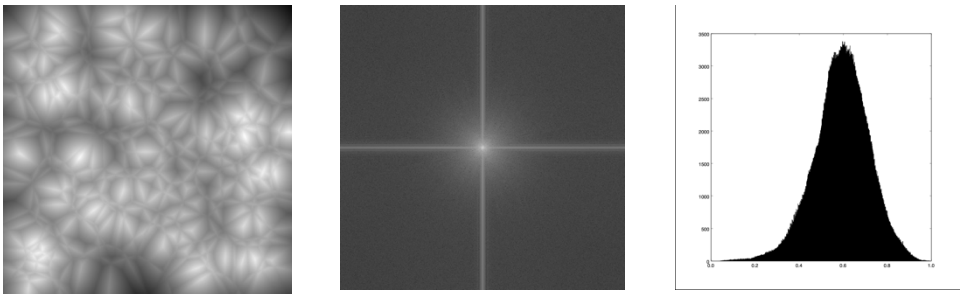


Bezier Worley with 6 Curves

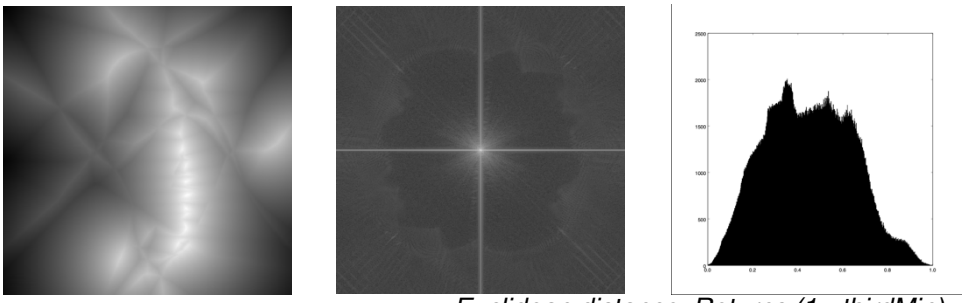


For each point, we generate the minimum distance to each Bezier curve, tracking the three smallest distances. For example, taking the average of the two minimum distances with yield patches of high and low concentration. Sparse area's will output high noise values while dense area's output the opposite. By generating pseudorandom points, much alike traditional Worley noise, and connecting these (in groups of 4) to form Bezier curves, this helps generate more gradually changing noise values. Take the below examples:

Worley Noise
(Traditional)
160 Cells

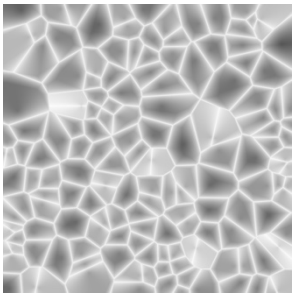
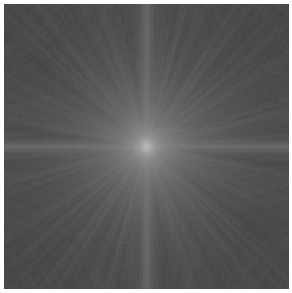
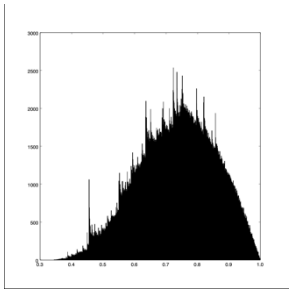


Worley Noise
(Bezier)
8 Curves
0.05 Increment
160 Total Cells

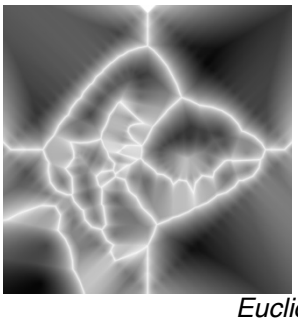
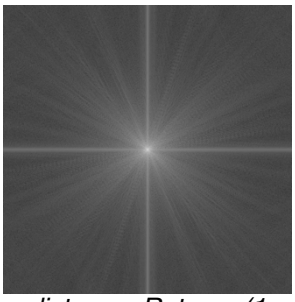
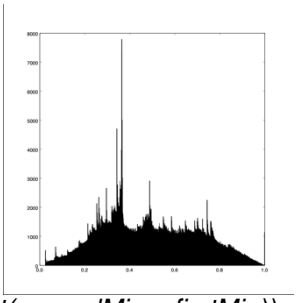


Euclidean distance. Returns (1 - thirdMin)

Worley Noise
(Traditional)
160 Cells

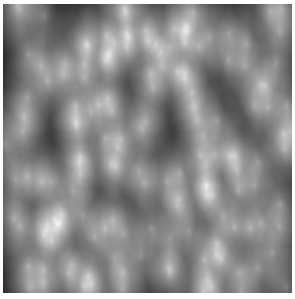
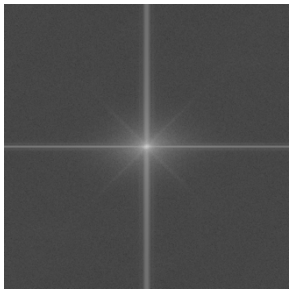
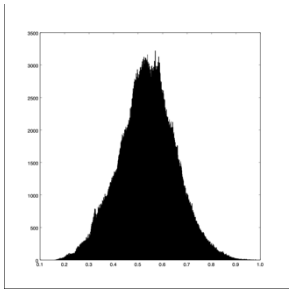




Worley Noise
(Bezier)
8 Curves
0.05 Increment
160 Total Cells

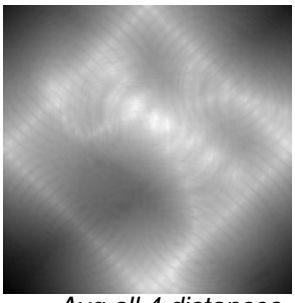
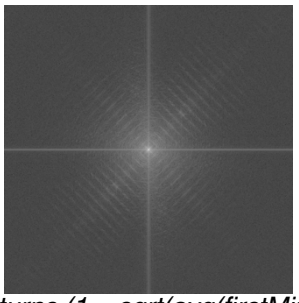
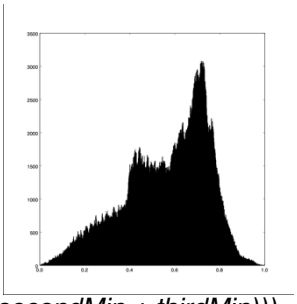




Euclidean distance. Returns (1 – sqrt(secondMin – firstMin))

Worley Noise
(Traditional)
160 Cells

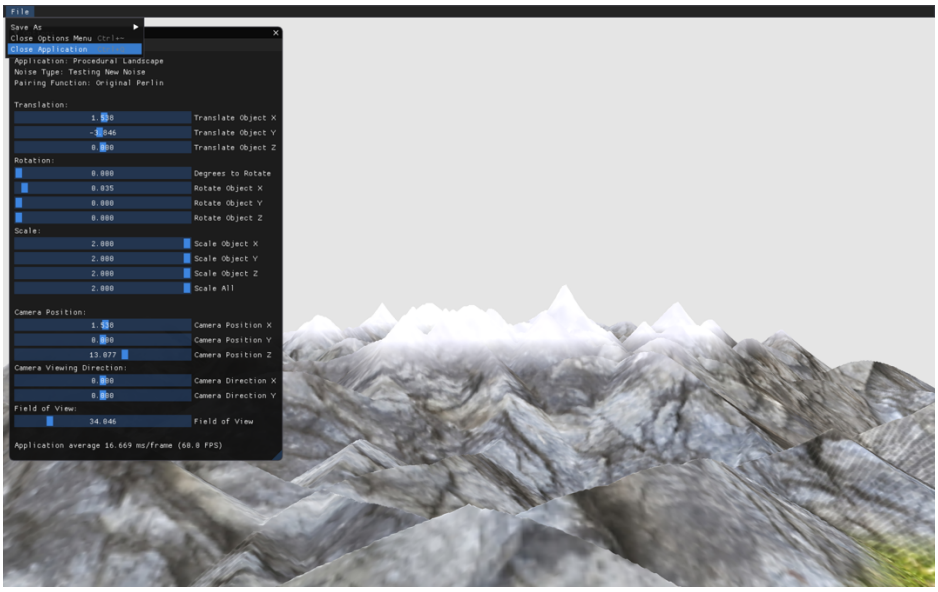




Worley Noise
(Bezier)
8 Curves
0.05 Increment
160 Total Cells

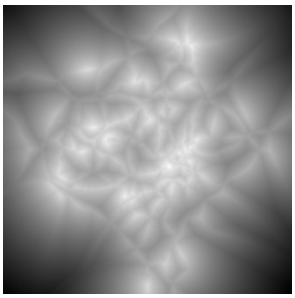




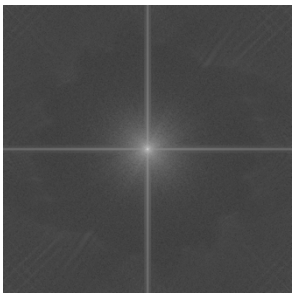
Avg all 4 distances. Returns (1 – sqrt(avg(firstMin +secondMin + thirdMin)))

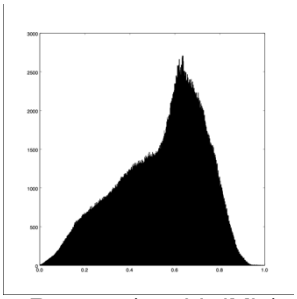
As you can see from the above tests, a modified Worley noise with Bezier curve may produce desired results where more continuous and associated ranges are emitted in the output, such as a mountain range.



Worley Noise
(Bezier)
16 Curves
0.05 Increment
320 Total Cells



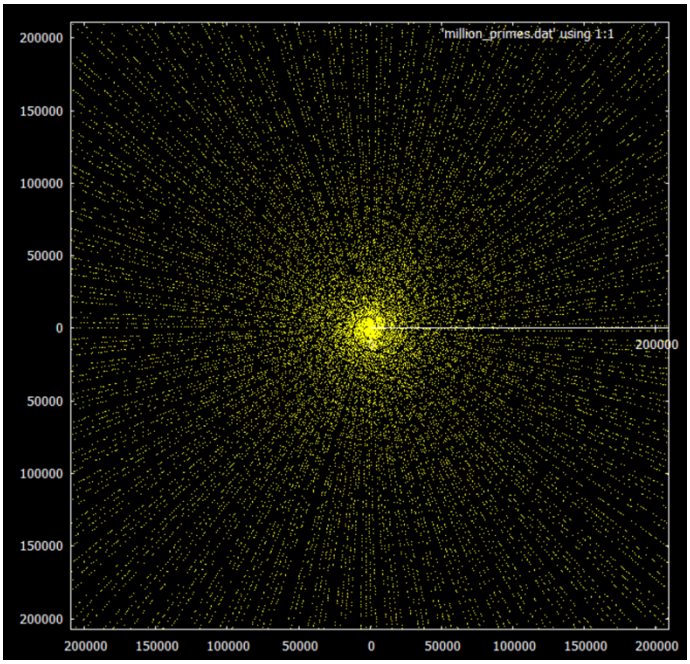




Euclidean distance. Returns (1 - thirdMin)

Prime Numbers & Noise Generation

Through inspection of the distribution of prime numbers, such that the polar coordinates (r, θ) = (p, p), where p is prime, we can see that when mapped to a 2D plane a pattern is formed that roughly resembles that of the targeted Fourier analysis we desire.



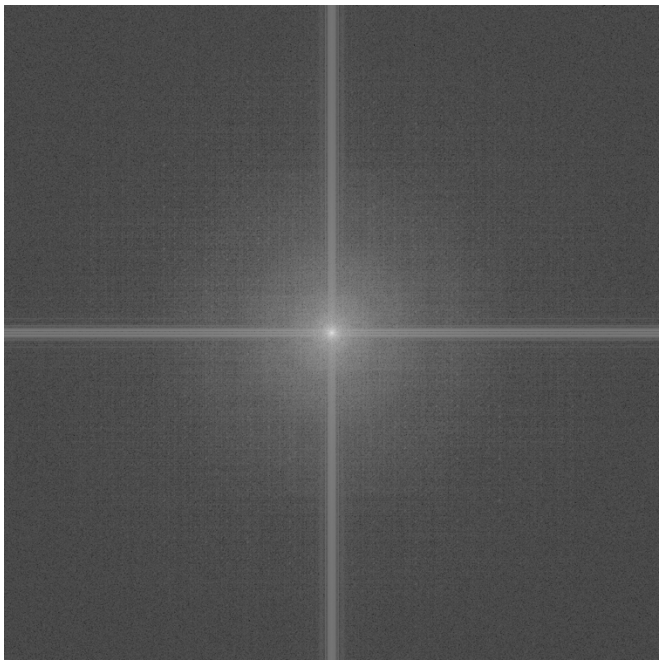


Image on the left was obtained from StackOverflow:
https://math.stackexchange.com/questions/885879/meaning-of-rays-in-polar-plot-of-prime-numbers

Idea presented came from the YouTube series 3Blue1Brown:
https://www.youtube.com/watch?v=EK32jo7i5LQ

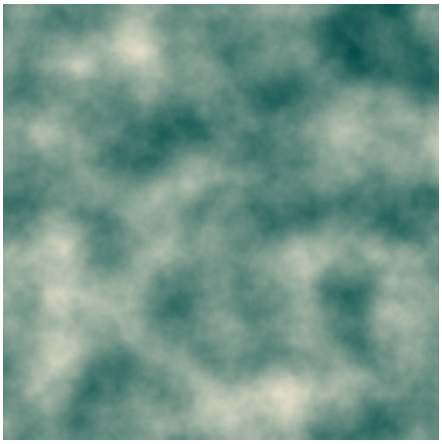
More to come. Still in progress.

Domain Warping

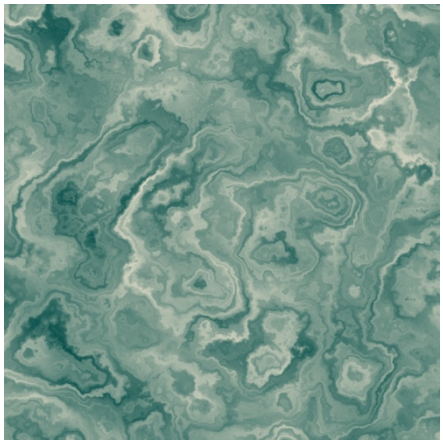
This process deals with running the fractional Brownian motion multiple times at each pixel. The steps are as follows:

Let noise = f(x, y, z)
First Level Domain Warp: f(noise + f(noise))
Second Level Domain Warp: f(noise + f(noise + f(noise)))
Third Level Domain Warp: f(noise + f(noise + f(noise + f(noise))))
Fourth Level Domain Warp: f(noise + f(noise + f(noise + f(noise + f(noise)))))

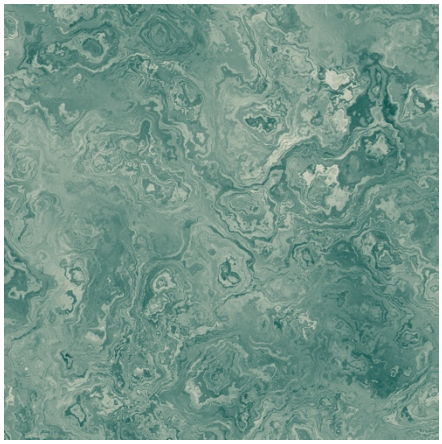
With general Perlin noise generation, we can visualize the domain being warped as the number of times the function is sent into the noise function increases:



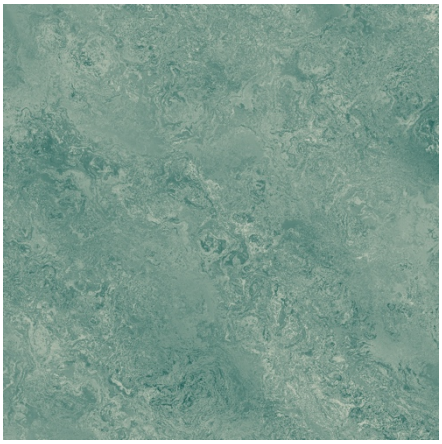
Perlin Noise



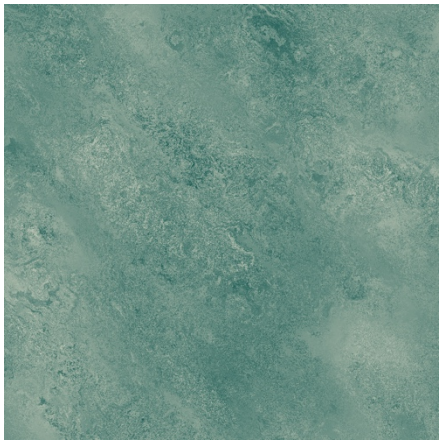
First Level Domain Warp



Second Level Domain Warp



Third Level Domain Warp



Fourth Level Domain Warp

This could prove useful in generating movement of textures such as water.