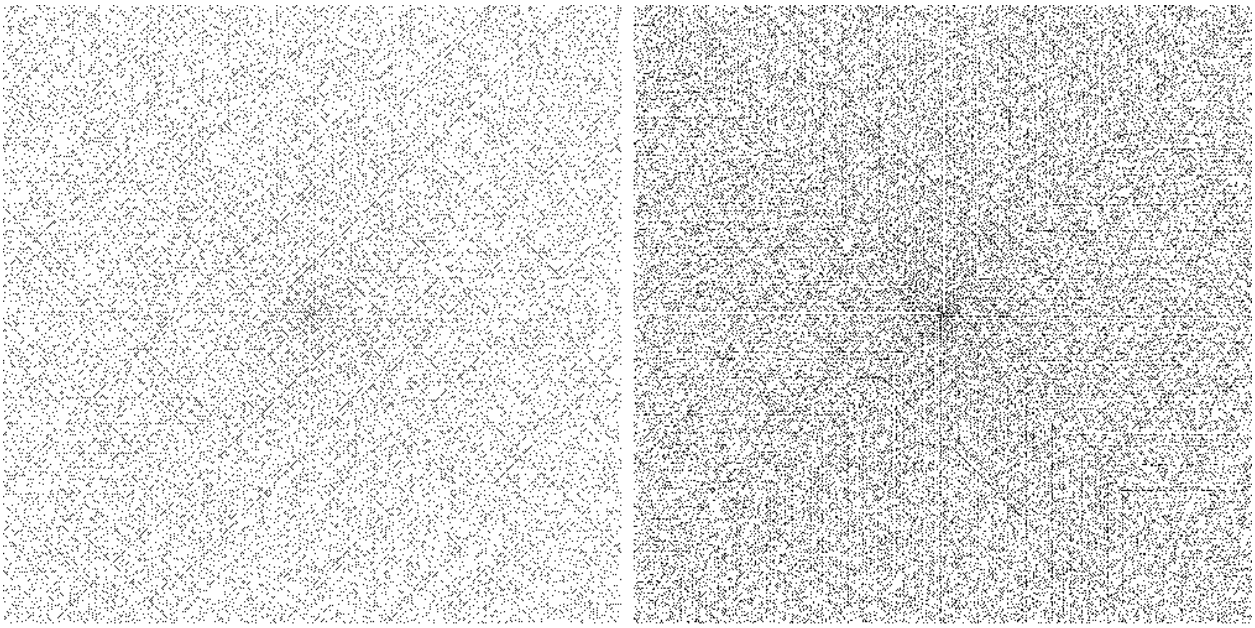
A Program for Generating a Generalized Ulam Spiral

Analysis of the Beurling Tree

Devin Platt

July 12, 2015

The Ulam spiral provides a visualization of the prime numbers. Here we provide a version for the Beurling primes.



The Natural Numbers

The Odd Numbers

In my first article on the Beurling primes I gave some images of generalized Ulam spirals. The code for generating such spirals can be found on Github (https://github.com/devinplatt/Generalized-Ulam-Spiral). The problem can be reduced into two parts: generating the spiral and representing a generalized prime system to determine the spiral. Since the first part has been done many times I will mainly explain the second. One way of generating the primes is to use a sieve. The basic algorithm is:

```
Algorithm Sieve(integer N)
    mark 1 as composite;
    mark every other number up to N as prime;
    foreach x in the list of integers do
        if x is marked prime then
            foreach multiple y of x below N do
                mark y as composite;
            end
        end
    end
```

There's nothing that says that we have to use normal multiplication for this. That's why in the program I provide a way to use generalized multiplication. It turns out that it is fairly easy to calculate the generalized multiplication function for generalized integer systems formed by removing a few numbers from the normal set of primes.

Just removing primes though is fairly limited. In fact, all systems generated this way will be denser than the normal integers, generating darker spirals. To generate less dense systems I've implemented functionality for any set of primes. The unfortunate trade-off is that this runs quite slowly.

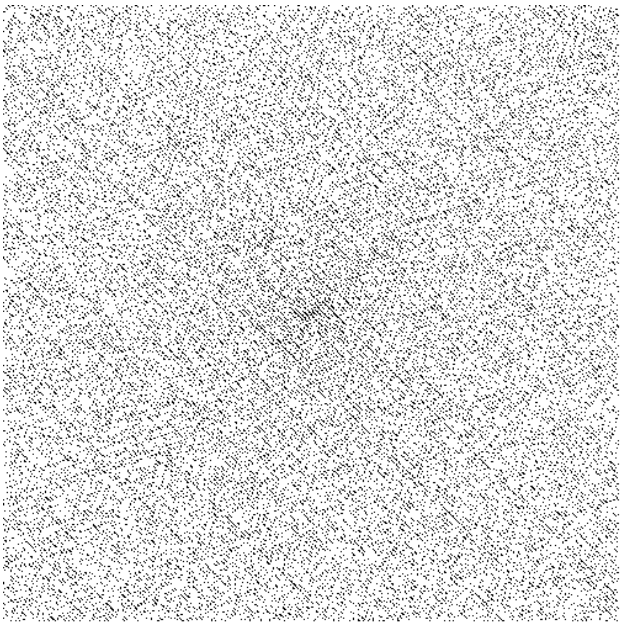
Given a list of primes $\{p_n\} = p_1, p_2, \ldots p_n$, the problem is to generate all of the integers up to p_n. The solution is to keep a list of generalized integers less than or equal to p_n. We start with the list $\{1\}$ and then iteratively for each prime add to the list multiples the prime with items in the list (including those just generated). While the other algorthms will run in $n * \mathcal{O}(multiply)$ time, to be honest I'm not even sure what the best possible runtime would be for this general solution.

1 Limitations

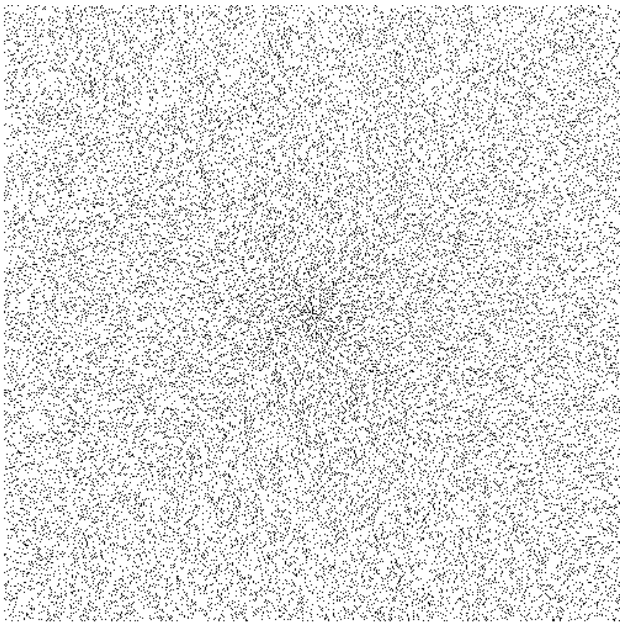
Ulam Sprials are great in that they provide a visualization of prime distributions, since we are lacking in such visualizations. However, there are two major limitations to Ulam Spirals:

- Ulam spirals lack information about limiting behavior of a distribution. This is a MAJOR flaw (admittedly it takes more creativity to figure out how to encode such information).
- For Beurling generalized primes, ANY ulam spiral is possible, with the exception that 1 is never prime and 2 is always prime. That is to say, that if we consider 500x500 pixel pictures, we are free to color in 500*500-2 pixels however we choose, and there are g-integer systems corresponding to that picture.

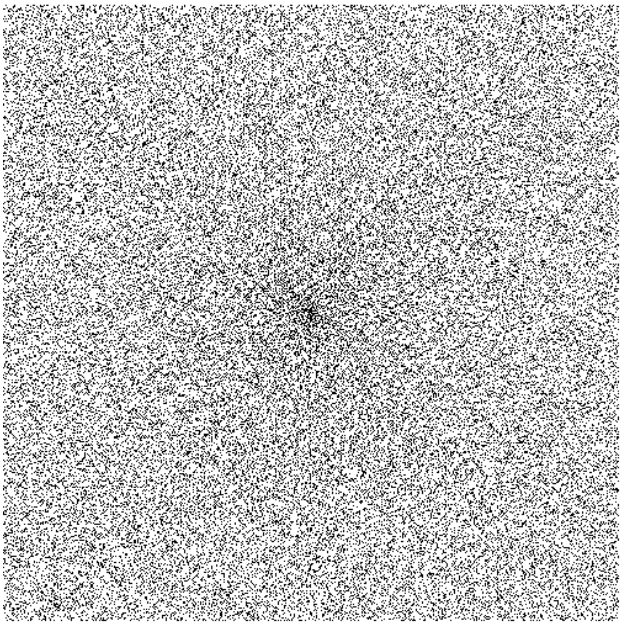
One possible workaround to the second limitation would be to use multiple shades rather than a binary black-white scheme. A number could be colored in based on how many prime factors it has. The picture could be normalized based on the maximum number of prime factors any number has in the picture. This would encode information about composites, and therefore address limitation 2 described above. I have yet to implement this type of spiral, but perhaps I can in the future.



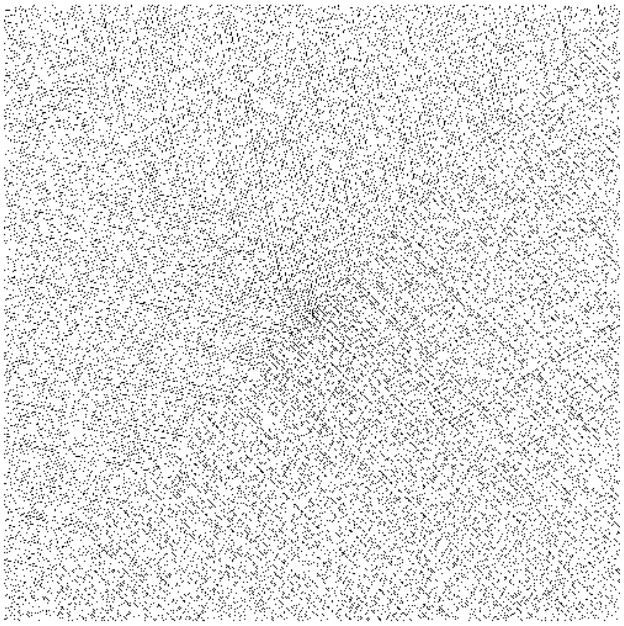
Sieving by 3



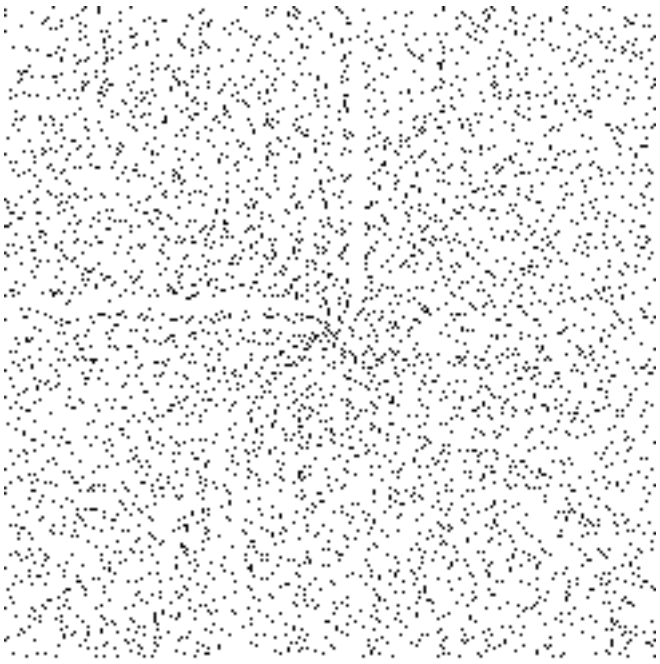
Sieving by 7, 11, and 31



Sieving by 2 and 3



Sieving by 5



Sieve by 2, add 1.5