Application of Procedural Generation in Modeling the Power Grid

Sean He

Abstract

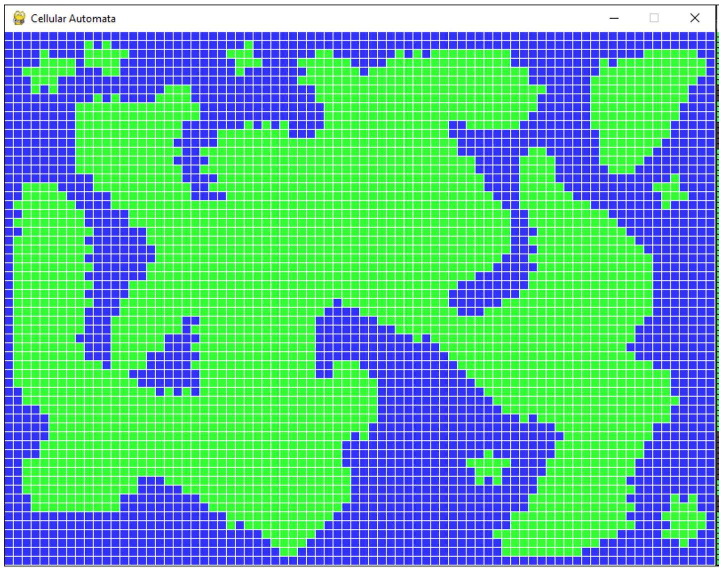
Power infrastructure security is a major topic of concern, as a failure of the power grid could lead to catastrophic outcomes, impacting large amounts of people. A major focus of the Cybersecurity research at Pacific Northwest National Laboratory is to evaluate and develop technologies to protect critical infrastructure. Procedural generation is a technology for algorithmically creating content, used primarily in the fields of computer graphics and the gaming industry. In this project, we use procedural generation and explore its application in power grid security. We developed a system to procedurally generate a realistic environment and power grid and then analyze the resilience of the generated network. This paper outlines the use of different procedural generation techniques, such as cellular automata and Perlin noise to generate terrain and a node-edge system, and how a series of programs and tools were created in Python to build and test a model power grid. The program was designed to be dynamic and realistic, for example, being able to create a realistic power grid which serves different island type environments. This system allows us to simulate and examine the resilience of a realistic power system without necessitating the need for real world mapping and data collection.

Introduction

The power grid is a critical infrastructure network, transmitting electrical power to millions of households across the nation. Assessing the vulnerabilities and protecting the power grid and other critical infrastructure is thus a center of focus. Previous attempts in modelling a power grid for analysis use systems heavily reliant on the underlying input data. A major problem which impacts even the best energy system models is the issue of availability of grid data. Generally, grid data is not publicly available due to non-disclosure policies rooted in security, which means models cannot be validated or evaluated [1]. This is the main motivation in looking to utilize procedural generation in power grid modelling. Procedural generation is a method which uses an algorithm to create data, incorporating computer randomness with human created assets. The main applications of procedural generation are in visual media, particularly in video games. Different procedural generation techniques are used to create the levels and worlds of various games, such as Rogue (1980) or Minecraft (2011). We look to explore if it is viable to use procedural generation to create a realistic sample power system, which can be used in simulations and limit testing. As a shorter-term goal, we look to create a proof of concept by generating a small system, such as an island type environment. If it proves to be possible to create a procedurally generated model, large and realistic power systems can be simulated and tested, bypassing the issues regarding grid data unavailability.

Progress

First, we wanted to determine if such a project was even possible, so a preliminary search of existing resources was conducted. The search was focused on finding and exploring power systems analysis tools and applications of procedural generation. In this exploratory stage of the project, both academic research and miscellaneous online resources were considered. Nearly all previous research on procedural generation was in its application in content creation for video games, such as procedurally creating cities to be used as a map in video games. An unexpectedly valuable resource was a talk given by a Minecraft developer, describing in detail how Minecraft uses Perlin noise maps to create its worlds. After discussion with my mentor team, I started the process of programming a few procedural generation algorithms to compare the viability of each for use for mapping the power grid. I settled upon using Python and started off by learning how to use the Pygame library to help visualize the maps I would be creating. Cellular automaton is a modelling algorithm in which there is a grid of discrete cells, and according to a predefined ruleset, will change state as a function of time. In our case, a grid is randomly populated with values, either which represents “land” or “water”, visualized as green and blue respectively. The ruleset dictates that if there are more than four water cells surrounding any cell, it will change into a water, otherwise being set to a “land” cell. As we progress through multiple iterations, clumps of land cells start forming and consolidating into distinct masses as seen in Figure 1a. Next, I created a proof of concept using Perlin noise. Perlin noise is a type of continuous random noise which is characterized for having smooth gradients of change. The noise was normalized to be between 0 and 1. However, Perlin noise is characterized by its smooth gradients of change, which in its unmodified form, does not emulate real terrain height very well. To modify this, we added additional octaves of Perlin noise. An octave of noise is defined as noise with twice the



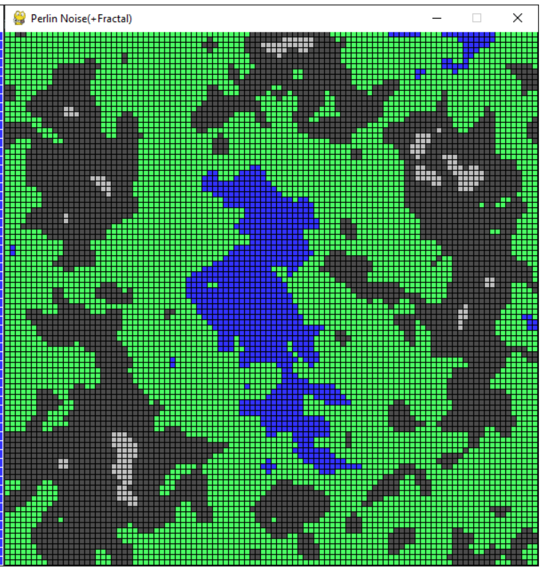


Figure 1a - Cellular Automaton

Figure 1b – Four Octave Fractal Perlin Noise

frequency, hence the name paralleling acoustics, and half the amplitude. This adds smaller scale details to the noise, akin to adding smaller hills on top of a rolling landscape. Four octaves of Perlin noise were then summed to each other, creating a ‘fractal’ noise map. Arbitrary boundaries between 0 and 1 were set for the different colors for visualization purposes as shown in Figure 1b. Blue was meant to represent water, green as the land, and the grey and light grey as high-altitude mountains.

The next step was to generate and distribute nodes across the terrain maps. However, this approach using a video game creation library as the visualization tool proved to be computationally inefficient and unscalable. We decided on continuing using the Perlin noise maps, as it was the most scalable and versatile. Instead of using Pygame, matplotlib’s pyplot library could be used to show images, on top of which a power network could be generated and displayed using a scatter plot.

A priority was to keep the distribution of nodes realistic. The map starts off as a grid of fractalized Perlin noise, normalized to between 0 and 1 (Figure 2a). This is used as the terrain

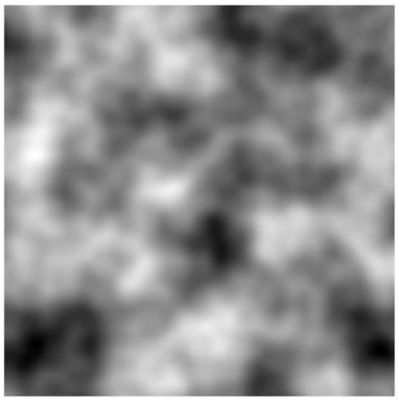


Figure 2a – Fractal Noise map

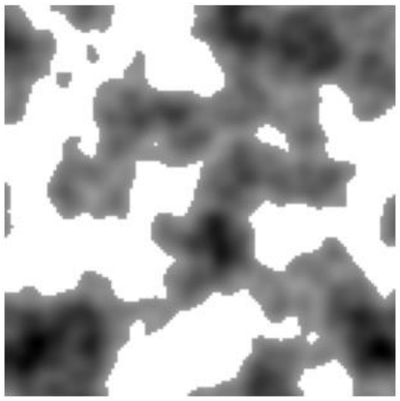


Figure 2b – Map with cutoffs

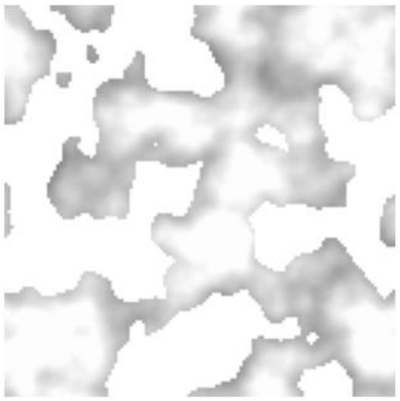


Figure 2c – Final Probability Map

map, on top of which nodes will also be procedurally generated. Next, a second map for the nodes would be generated. This second map would represent the probability that a node would be created at any given location. The node noise map starts off as the terrain map, where a cutoff is established on the map, removing areas that would be water (Figure 2b). Then, the probabilities were inverted, making it so that the proximity to a body of water positively correlated to the probability of a node forming. This helps contribute to realism, as civilization in the real world tends to form around bodies of water. This produces a gradient of nodes, with the majority forming by coasts. The next step was to create another fractalized noise map, and then sum it to the existing map to introduce randomness to the node generation. Finally, a spline function was applied to the probabilities to produce the final probability map of a node generating (Figure 2c). A spline function is a piecewise function which serves to de-linearize and scale the probabilities. Finally, using the probability map, a list of nodes was generated. The nodes were represented using an object-oriented approach, where each node was represented by its coordinates, node type, and connections. Figure 3 shows a generated map will all the features available currently, a terrain map with generated nodes, which are still without properties or connections.

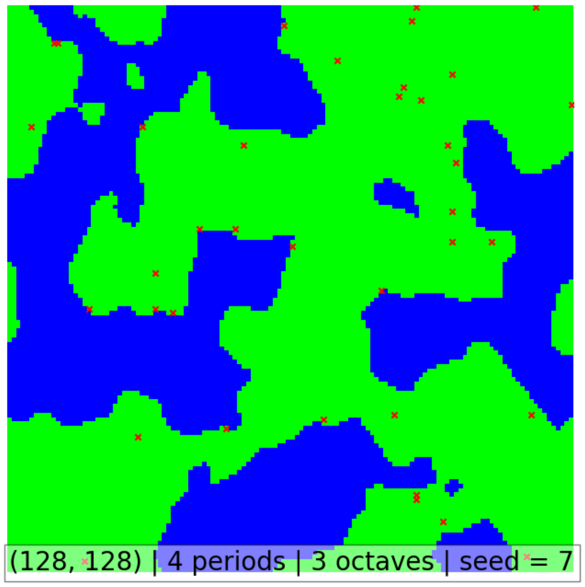


Figure 3 – A Procedurally generated map and nodes

Future Work

We will continue to explore this path and experiment with different methods to specialize and connect these independent nodes into a coherent network, as well as bridge the gap to a simulation toolkit. A simple power grid consists of generation, a substation, transmission, and distribution. In order to model a power grid, the nodes must have an assigned type, and connect in a realistic manner. The nearest neighbor algorithm is potentially the simplest approach, where a random node is selected, and then the network is built from there, connecting to the nearest node and assigning the corresponding properties as the network grows. We will also explore different pathfinding algorithms, such as Dijkstra's algorithm, which is an algorithm in graph theory where the most efficient path to take through a graph to reach from point A to B is found. We will also consider different approaches, perhaps generating a power grid as a vertex-edge graph first, then distributing it geographically. Using this alternate approach would prevent the complicated step of creating a network from a list of disconnected nodes. A possible method is to use a Lindenmayer system. L-Systems are a grammar system which can be used to generate strings which represent a network. There has been previous work using a L-System to procedurally model a city. The L-System was used to create a network of streets and intersections, creating a complex city map [2]. The work above serves as a proof of concept, and we will look to scale the project up, to perhaps using a game engine such as Unity or Unreal

Engine to simulate and model to a more granular level as well as at a larger scale. These models could then be used in load simulations, to see the impacts of different events on the grid.

Impact on Laboratory or National Missions

As part of the National Security Directorate and the Cyber Risk Reduction team at Pacific Northwest National Laboratory, a major priority is to ensure the security of critical infrastructure at every point in the supply chain. Additionally, the Department of Energy has a special interest in protecting its critical systems and addressing emerging threats. Being able to model and simulate a power grid would greatly benefit research in risk reduction and impact. This project would allow for generation and simulation of critical infrastructure like the power grid and may be a jumping off point for mapping other critical infrastructure systems.

Conclusions

Procedural Generation is a method of using an algorithm to generate content, previously primarily used in visual media. Through this project, we demonstrated its viability as a tool to map critical infrastructure, in this case, procedurally creating a representation of a power grid. As we continue to build upon and refine the process, we look to create a realistic power grid on which deeper analysis can be conducted. Through the processes explored in this paper, we will continue to work towards a small-scale island type environment as a proof of concept, and then look to scale up and expand to a more sophisticated model.

References

[1] Medjroubi, Philipp Muller, Scharf, Matke, Kleinhans. 2016. “Open Data in Power Grid Modelling: New Approaches Towards Transparent Grid Models”. Energy Reports, Volume 3: 14-21. [DOI: https://doi.org/10.1016/j.egyr.2016.12.001]

[2] Parish, Müller. 2001. “Procedural modeling of cities”. In Proceedings of the 28th annual conference on Computer graphics and interactive techniques (SIGGRAPH '01): 301–308. [DOI: https://doi.org/10.1145/383259.383292]

Appendix

Participants

Name	Institution	Project Role
Jessica Smith	Pacific Northwest National Laboratory	Cyber Security Researcher, Project PI; Team Lead and Mentor
Animesh Pattanayak	Pacific Northwest National Laboratory	Cyber Security Researcher, Mentor; oversees programming tasks
Matthew Kirkland	Pacific Northwest National Laboratory	Cyber Security Researcher, Mentor
Darien Littlewolf	Pacific Northwest National Laboratory	SULI Intern; resource collection and literature research
Sean He	Pacific Northwest National Laboratory	CCI Intern; resource collection, literature research, main programmer