Ryan Vera
Qitong Liu

CS 6366
5/4/2025

Procedural Terrain Generation Final Report

1 Summary

This project explores procedural terrain generation by implementing a Perlin noise generator to create lifelike and diverse landscapes, in addition to implementations of the Diamond-Square and Voronoi Biome algorithms. These techniques are widely used in video games, simulations, and virtual environments for generating dynamic terrains without manual modeling. The project aims to analyze and compare the algorithms in terms of realism, scalability, and computational efficiency, providing valuable insights for applications in world-building and real-time rendering. With all three algorithms implemented, this project will also comparatively evaluate the results of the generated terrains.

2 Description

For each terrain generation algorithm, the implementation followed a structured approach. Initially, a basic version of the algorithm was developed and plotted to visualize its raw output. Next, smoothing functions were implemented and applied to refine the generated terrain and enhance its continuity. To further evaluate the algorithms, experiments were conducted using a range of parameter values, allowing for a comparative analysis of different settings. The most effective parameter configurations were identified and documented for analysis, alongside example images showcasing the optimized results. Finally, a terrain-like color pattern was incorporated into the plots to improve visual realism, making the generated landscapes more representative of natural environments.

3 Results

3.1 Perlin Noise

Perlin noise is a gradient-based procedural algorithm widely used for generating natural-looking textures and terrain. It creates smooth, continuous variations by blending gradients across a grid, producing organic patterns that resemble rolling hills or fluid-like surfaces

3.1.1 True Perlin Noise

This version of Perlin Noise was created using a Python noise library. The results produced by this served as a comparative baseline for the implementation that was conducted in this project. The images shown below have been smoothed to show realistic results using the noise library,

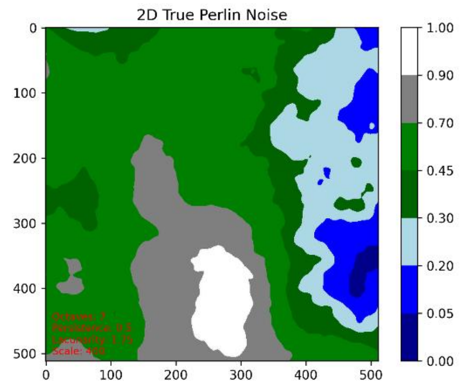

Figure 1) 2D plot of Perlin Noise produced by noise library

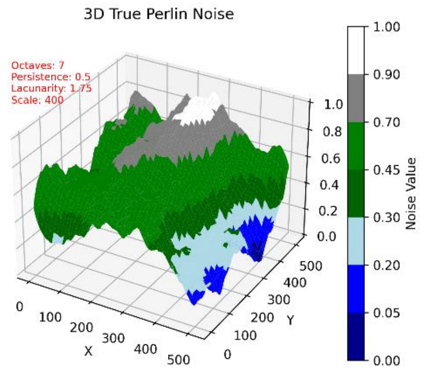

Figure 2) 3D plot of Perlin Noise produced by noise library

3.1.2 Perlin Noise

This version of Perlin Noise was implemented for this project. It features no smoothing.

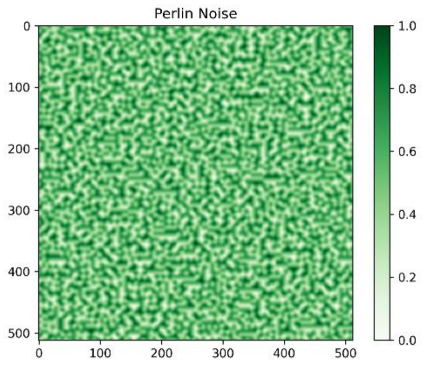

Figure 3) 2D plot of Perlin Noise

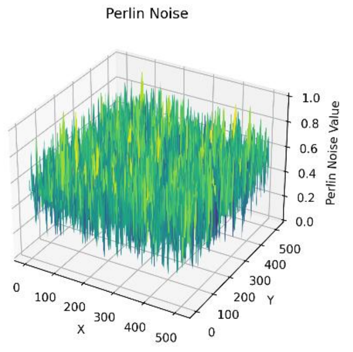

Figure 4) 3D plot of Perlin Noise

3.1.3 Fractal Smoothing

This version of Perlin Noise has been smoothed by the fractal smoothing technique. Essentially, this technique combines multiple iterations of regular Perlin Noise to create a smooth surface.

The fractal smoothing technique for Perlin noise relies on several key parameters that influence the final terrain appearance. The **frequency** determines the scale of terrain features, with higher values producing finer details and lower values creating broader undulations. The **amplitude** controls the height variations, affecting the contrast between peaks and valleys. The **number of octaves** defines how many layers of Perlin noise are combined—higher octave counts introduce more complexity and finer details, while fewer octaves result in smoother terrain. Adjusting these parameters allows for a balance between rugged, realistic landscapes and more subdued, gradual elevation changes, making fractal smoothing highly versatile for different terrain styles.

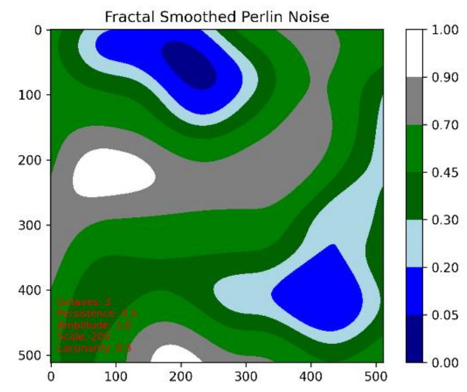

Figure 5) 2D plot of fractal smoothed Perlin Noise

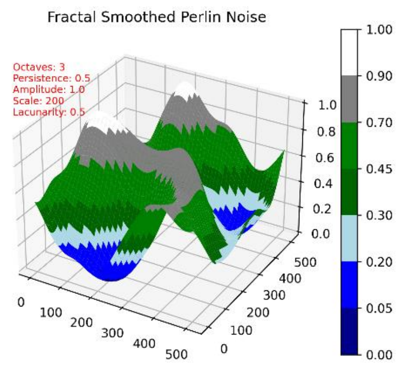

Figure 6) 3D plot of fractal smoothed Perlin Noise

3.1.4 Gaussian *Smoothing*

This version of Perlin Noise was smoothed using the Gaussian smoothing technique. The use of this technique refines the terrain by reducing sudden changes in elevation. The smoothed noise output created softer transitions between peaks and valleys, enhancing realism while maintaining the procedural generation's characteristic variability.

The main parameter in this algorithm is **sigma**. It's responsible for standard deviation of the gaussian function that is used to smooth the noise.

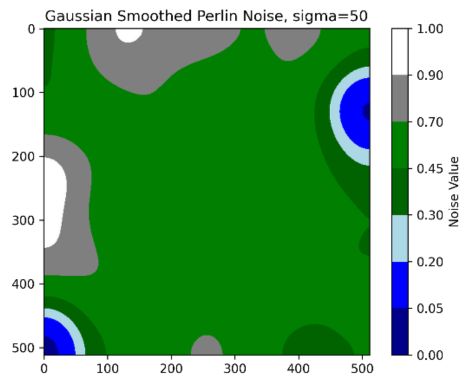

Figure 7) 2D plot of Gaussian smoothed Perlin Noise, sigma=50

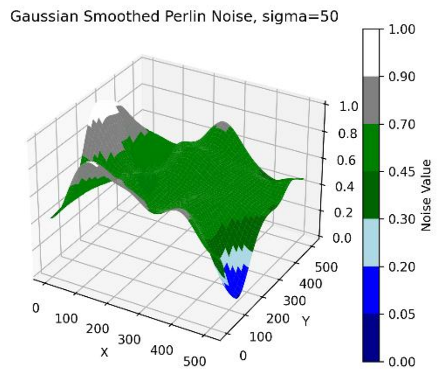

Figure 8) 3D plot of Gaussian smoothed Perlin Noise, sigma=50

3.2 Diamond-Square

Diamond-Square is a recursive fractal algorithm that generates terrain through two steps – a Diamond step, where the corners of squares are averaged and the center point is displaced, and the Square step, where surrounding points of an edge are averaged and the midpoint is displaced. These displacements are applied in every step but decrease in magnitude as the steps are repeated.

3.2.1 Original Diamond-Square

This version of Diamond-Square was implemented as a baseline fractal terrain generation algorithm. Random values were assigned to the heights of the four corners of the map, then the values inside were filled in according to the Diamond and Square steps. Gaussian smoothing was also applied to smooth out the created maps.

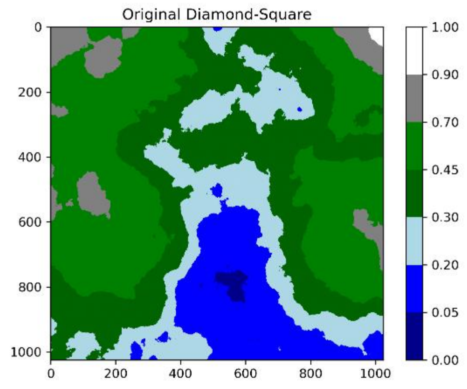

Figure 9) 2D plot of Original Diamond-Square

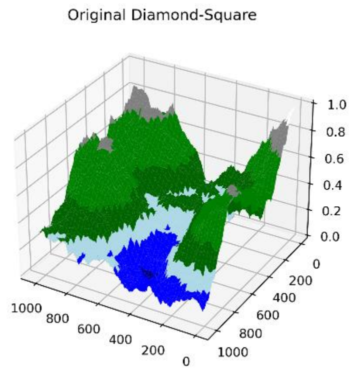

Figure 10) 3D plot of Original Diamond-Square

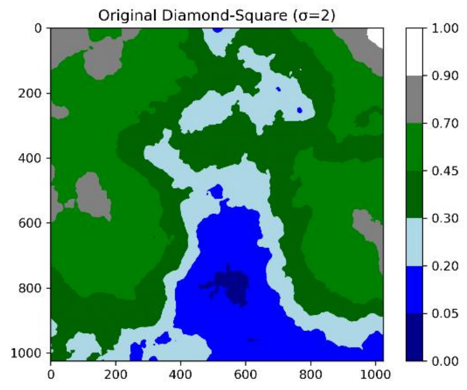

Figure 11) 2D plot of Gaussian Smoothed Original Diamond-Square with sigma = 2

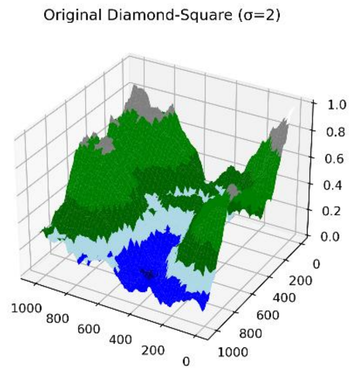

Figure 12) 3D plot of Gaussian Smoothed Original Diamond-Square with sigma = 2

3.2.2 *Wrapped Diamond-Square*

The wrapped version of the Diamond-Square algorithm is altered to create a pattern that can be repeated seamlessly. By using modulo indexing and assigning all four corners to the same value, the terrain wraps around the edges, which allows for the map to be repeated without any sudden changes in altitude.

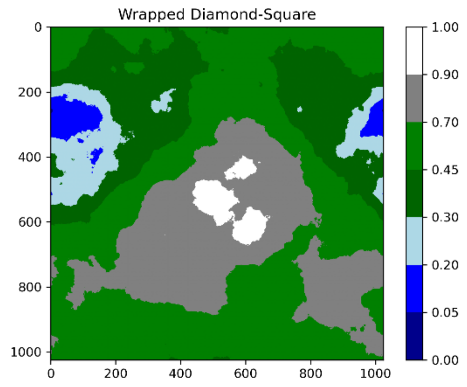

Figure 13) 2D plot of Wrapped Diamond-Square

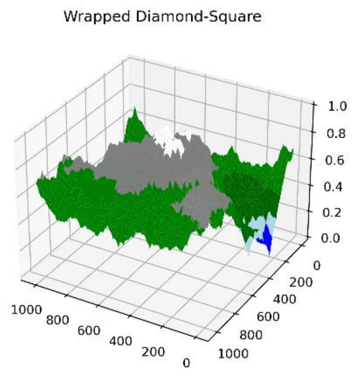

Figure 14) 3D plot of Wrapped Diamond-Square

3.2.3 *Radial Falloff Diamond Square*

This variant of the Diamond-Square algorithm applies a radial falloff mask, which multiplies the heights of the generated terrain by a value that decreases as the distance from the center increases. This creates a terrain that is ideal for island-like structures that may be centered around a volcano or mountain.

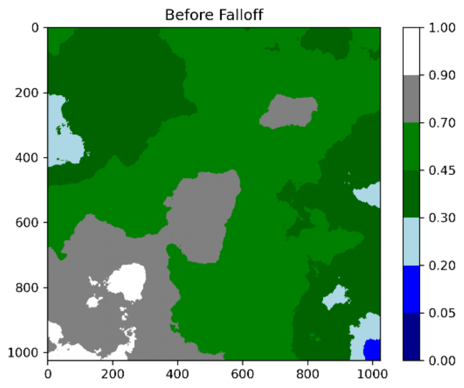

Figure 14) 2D plot of Diamond-Square before Falloff

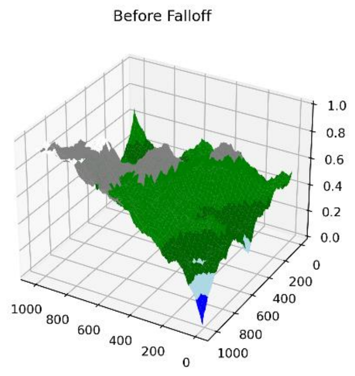

Figure 15) 3D plot of Diamond-Square before Falloff

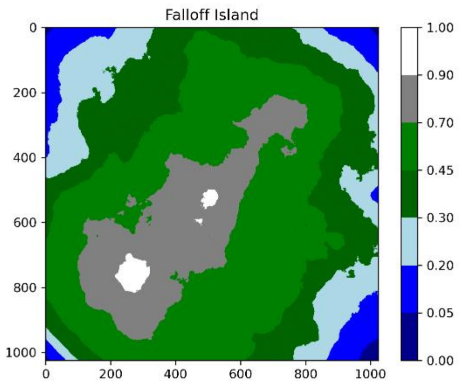

Figure 16) 2D plot of Diamond-Square with Radial Falloff

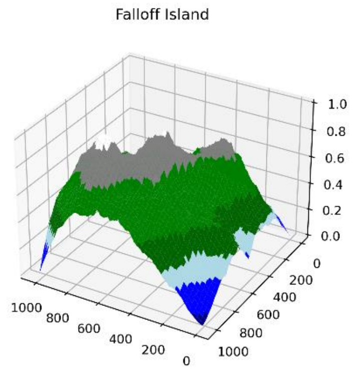

Figure 17) 3D plot of Diamond-Square with Radial Falloff

3.3 Voronoi Biome

Voronoi Biome generation splits a map into different regions based on distances to randomized seed points. Within this project, a Two-Tier Voronoi Biome algorithm was implemented. Major seed locations were first randomly selected, with each one receiving a biome classification. For this project, each biome was also guaranteed to be selected at least once. Afterwards, minor seed locations were then randomly selected. Minor seeds were assigned biomes depending on what major seed they were closest to but implemented an "in-between" value if there was a second major seed close enough to the minor seed. This implementation creates a smoother, more organic transition between regions rather than harsh cutoffs. A severe limitation of this algorithm, however, is that it does not generate any heights.

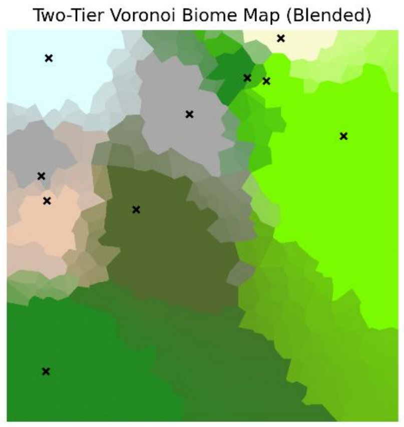

Figure 18) 2D plot of Two-Tier Voronoi Biome

3.4 Comparative Analysis

Perlin Noise, Diamond Square, and Voronoi Biome are distinct procedural terrain generation algorithms, each contributing unique characteristics to landscape formation. Perlin Noise produces smooth, continuous elevation variations, making it well-suited for natural terrain with gradual slopes. Diamond Square, utilizing recursive subdivision, creates rougher, more rugged landscapes, often resembling mountainous regions. Voronoi Biome forms distinct, segmented terrain features based on proximity-based partitioning, effectively simulating biome separations or geological formations. While each algorithm has individual strengths, they can be combined to generate complex and varied terrains. Integrating Perlin Noise for base elevation, Diamond Square for added ruggedness, and Voronoi Biome for biome distribution enables a balanced approach, resulting in diverse and realistic procedural environments tailored to specific applications

4 Analysis of Work

This project successfully met its original objectives, implementing Perlin Noise, Diamond Square, and Voronoi Biome algorithms along with their respective smoothing functions. Each algorithm was developed, tested, and refined, demonstrating diverse terrain generation capabilities. Experiments with different parameter values provided insights into terrain realism and adaptability, while the addition of terrain-like color patterns enhanced visualization. All planned goals were achieved, ensuring technically sound and visually representative outputs. No significant setbacks were encountered, as the structured approach—beginning with basic implementations, followed by smoothing, experimentation, and visualization—allowed for a comprehensive exploration of procedural terrain generation.

5 Repository

A link to the GitHub repository for this project can be found here.