Simplified 3D terrain meshes using Poisson Disk Sampling and Perlin Noise on GPU

1st Rodrigo A. Cayro Cuadros
Institute of Mathematics and Computer Sciences (ICMC)
University of São Paulo (USP)
São Carlos, SP, Brazil
rodrigo.cayro@usp.br

4th Leonardo Tortoro Pereira
Institute of Geosciences and Exact Sciences (IGCE)
Sao Paulo State University (UNESP)
Rio Claro, SP, Brazil
leonardo.t.pereira@unesp.br

2nd Cláudio F. Motta Toledo
Institute of Mathematics and Computer Sciences (ICMC)
University of São Paulo (USP)
São Carlos, SP, Brazil
claudio@icmc.usp.br

3rd Juan C. Gutiérrez Cáceres
Computer Science (CCOMP)
Catholic University San Pablo (UCSP)
Arequipa, Peru
jcgutierrezc@ucsp.edu.pe

***Abstract*—The video game market has become one of the most lucrative sectors, generating $187.7 billion in 2023 and projected to reach $200 billion by 2026. Terrain generation, essential in video games and graphical applications, poses challenges in terms of time and resources. Automated methods, while efficient, often lack customization, leading to increased computational costs for larger terrains. Our proposal combines *Perlin Noise* and *Poisson-Disk Sampling*, allowing terrain complexity to be reduced to 4% of its complexity and size, using parallelization on the GPU, and comparing its performance against traditional techniques such as CellularAutomata and DiamondSquare.**

***Index Terms*—Terrain generation, Poisson-Disk Sampling, Perlin Noise, GPU, Noise generation**

Full/Regular Research Paper submission for the conference CSCI-RTCS: Applications of Computational Science

I. Introduction

Procedural Content Generation (PCG) refers to the automated process of creating virtual content for visual applications through algorithms that ensure consistency, uniformity, and reliability. This content can encompass a wide range of digital elements, including levels, maps, rules, textures, and graphics. Well-known games such as *Diablo*, *Minecraft*, *Spore*, *Starcraft*, *Terraria*, among others, have leveraged PCG techniques to dynamically generate their content [17].

In recent years, the digital gaming industry has grown into a vast global market, generating hundreds of billions of dollars in revenue [14]. As a result, the demand for advanced algorithms capable of efficiently handling large-scale content generation, particularly in terrain creation, has increased significantly. These algorithms must offer not only speed and efficiency but also scalability to meet the ever-growing requirements of modern game development.

The generation of artificial terrains finds its most prominent applications in video games, movies, and other graphical applications, such as animations and simulations, as highlighted in [15] and [20]. The creation of such applications involves significant costs, both in terms of development resources and financial investment, especially in the constantly growing video game industry [18].

In this context, we can identify three main approaches for generating artificial terrains, as described by [17]:

- **Artificial Intelligence (AI)-based methods:** These approaches generate environments dynamically and automatically using learning techniques trained on real data or images. Notable examples include Generative Adversarial Networks (GANs), as in the work of [16], or evolutionary methods. While these techniques can produce impressive results, they tend to be resource-intensive, both in terms of time and computational power. Moreover, many of these methods follow a sequential approach, which limits their scalability.
- **Geometric or graphic methods:** These techniques rely on mathematical calculations to create realistic and natural-looking terrains. One of the most popular methods is noise-based generation. For instance, Poisson-disk sampling is a technique that enables the creation of a random yet uniform distribution of points, which can be useful for generating smooth and naturally varying terrains [21]. These methods offer flexibility by allowing terrain generation to be parameterized and parallelized. However, they can face challenges when scaling to higher dimensions, often due to their sequential nature and increased computational complexity.
- **Rule-based methods:** This approach involves defining a set of states for each part of a terrain, along with transition rules that dictate how the states evolve. For example, a terrain may have states such as water, land, or mountains, with rules specifying that water transitions to land if surrounded by land, and land transitions to mountains if surrounded by mountains. While rule-based methods are effective in generating simple and coherent terrains, they are limited by the complexity of defining

these rules and the restricted number of possible states.

This work focuses on the challenge of generating 3D terrains through **geometric methods**. Terrain creation, while essential for developers, often demands significant resources. In this context, automatic generation offers a practical alternative to manual design. However, relying solely on automatic generation tends to produce repetitive, uninspired results with limited flexibility in design. As the scale of 3D terrains expands, so does the complexity, making it increasingly difficult to shape specific terrain features without a substantial increase in workload. Additionally, running complex algorithms on less advanced hardware can lead to reduced computational efficiency, which in turn diminishes the richness and variety of the in-game terrain, directly impacting the player's experience.

We propose a system that integrates Poisson-Disk Sampling and Perlin noise [11] to generate terrain samples and simplified terrain meshes, leveraging GPU acceleration for 3D terrain generation. This hybrid approach facilitates the creation of terrains in a simpler, faster manner while allowing for effective control over computational complexity, all while preserving an appropriate level of detail. The system begins by operating in 2D to generate sample points and subsequently applies Perlin noise in 3D, fully utilizing the capabilities of massively parallel GPU architectures.

A key advantage of our system is its flexibility in generating terrains of virtually any desired shape. The primary contribution of this work is the introduction of a novel method that simplifies terrain generation by controlling the quality. Additionally, we offer a framework for evaluating our terrain generation approach using various metrics, including processing time, complexity, spectral quality, and algorithmic complexity. These metrics are compared against existing methods such as Cellular Automata (CA) and the Diamond-Square algorithms. Furthermore, the generated terrains are reconstructed within a game engine, facilitating graphical applications and enabling future testing in video games.

In Section II, we are going to present the works related to our proposal, the state of the art. In the following section, we will present our proposal III. In Section IV, we will present our results with the preliminary experiments collected so far. Finally, our limitations are discussed in Section V, along with the conclusions.

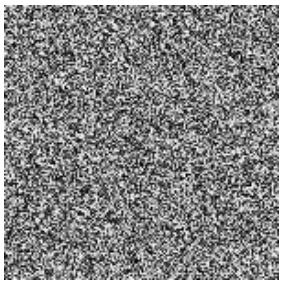
II. Related work and Techniques

In this section, we review key works that influenced our approach and discuss the techniques used for comparison. We will first explain Random Noise and Poisson Disk Sampling to better understand our proposal.

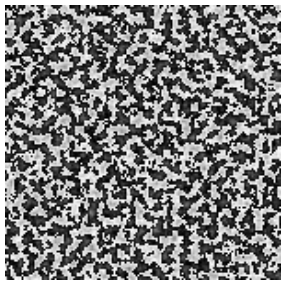
A. Random Noise and Poisson Disk Sampling

Random noise plays a crucial role in procedural content generation, enabling diverse variations in objects. For terrain generation, different seed types create unique environments, as outlined by [8]. However, [10] highlights that random numbers alone do not contribute meaningfully to terrain generation. Figure 1a illustrates an example of random noise.

To effectively leverage noise in terrain generation, Thanh Le [10] presents a 2D terrain model where random values range from 0 to 255. These values can represent terrain heights, with lower values indicating features such as rivers or plains, while higher values suggest the presence of mountains or the sky. This approach introduces meaningful variation to the generated terrain, as illustrated in Figure 1b.



(a) Random Noise, don't look like a terrain.



(b) Perlin Noise, have a terrain pattern.

Fig. 1: Comparison of Random Noise and Perlin Noise

For **Poisson-disk sampling** algorithms, the foundation is the Poisson distribution, illustrated in Figure 2. This distribution is a probabilistic model that shows the frequency of events occurring within a specific time period. In Poisson-disk sampling, it is used to randomly distribute points in space while ensuring they are adequately separated. In order to perform a sampling that represents the distribution space.

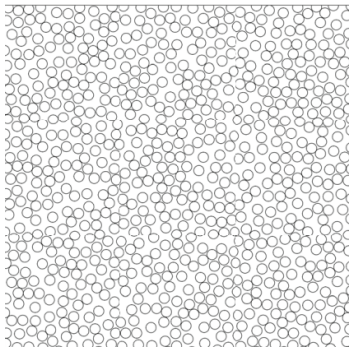


Fig. 2: Poisson Sampling using [22].

As discussed in Section I, understanding the theory behind **Poisson-disk sampling** is crucial for applying this method to terrain generation. Poisson-disk sampling has three key characteristics:

1) **Minimum Distance:** Each point is surrounded by a disk with a minimum radius, ensuring that the distance between any two points x_i and x_j in the set E is at least $\min(r_i, r_j)$, where r is the radius used for point distribution. The radius can be constant, as in [1], or adaptive, as in [3], allowing for variable distances between points for finer control over the distribution.
2) **Uniform Placement:** If S is the space where points are placed and n is the number of points, the probability of

placing a point in a region $R \subseteq S$ is proportional to the volume of R. Specifically, $P(R) = \frac{V(R)}{V(S)}$. This ensures that every point has an equal chance of being placed in any part of the space, independent of location, relying solely on the region's size.

3) **Maximal vs. Non-Maximal Distribution:** Maximal distribution aims to cover as much area as possible with minimal gaps, as seen in [13]. Non-maximal distribution does not prioritize covering empty spaces, leading to partial area coverage.

Given the explanations of key terms above, we will now explore two major approaches in state-of-the-art terrain generation for this research: AI-based methods and Geometric methods.

B. AI-Based Methods

AI-based methods for procedural content generation (PCG) include several notable approaches:

In [9] introduces a method for real-time cave generation using Cellular Automata (CA). This method divides the map into grids, with each cell assigned a type (rock or soil) and states based on neighboring cells.

The research of [5] presents a PCG approach for 2D dungeons using context-free grammar to define mission sequences for level completion. The map generation involves CA with both constant and non-uniform evolution. The space is divided into grids, with one randomly chosen as the starting point. Two rules guide the automaton cells: one for forming rooms based on mission symbols and another for placing connections between rooms using Manhattan distance as a metric.

the work of [16] describes the use of a Generative Adversarial Network (GAN) trained on points of interest, including their geodesic coordinates and altitude. This GAN can generate realistic terrain images from a 2D sketch or design, demonstrating its ability to create photorealistic terrain images.

In [21] explores PCG for 2D dungeons in video games. They use CA for terrain generation, complemented by Poisson Disk Sampling to create a playable environment. The game uses grids where each vertex represents a cell, forming simple, uniform connections. Connections are established using a Depth-First Search (DFS) algorithm, and Poisson Disk Sampling distributes content across the map. This method offers advantages such as a quick processing time of 72 milliseconds for map generation and adaptability to various mechanics or parameters. However, it has some drawbacks, including repetitive patterns due to limited terrain variation, the lack of parallel generation, and the need for smoothing to correct erroneous content distribution. Additionally, it does not explore 3D terrain generation.

C. Geometric Methods

Geometric methods use mathematical techniques and algorithms to shape terrain, considering factors like elevation, slope, and irregularity. These methods often employ noise functions, fractals, and similar techniques.

In [2] proposes a fractal-based approach optimized with geographical and physical considerations for large-scale terrain generation. This method uses fractals to generate terrain features like tectonic elevation and river erosion. Fractals are effective for non-fluvial terrains as they capture terrain repetition at different scales. However, for fluvial terrains, fractals alone are insufficient due to a lack of geological realism. To address this, [7] introduces a fractal enhancement algorithm for generating curves for ridges and rivers.

In the work of [19] describes a real-time 3D terrain generation method using fractals and GPU parallelization. The approach includes a minimalist terrain editor based on user-provided sketches, allowing for realistic and expressive terrain creation. The algorithm operates in two stages: bottom-up, where user-defined elevations guide fractal subdivisions, and top-down, where a subdivision algorithm refines the terrain based on the elevation map.

The works [12] presents a CPU-optimized terrain generation method using Perlin noise. This technique generates terrain with continuous randomness between 0 and 1, defining mountain heights within the terrain, as shown in Figure 1b. The process starts with determining the desired detail level for the mesh, generating 3D Perlin noise, and applying it to the mesh. This method offers detailed terrain maps and efficient CPU processing but lacks GPU support, and larger maps require more complex meshes.

Recent approach of [4] introduces a noise-based method for procedural terrain creation with elevation constraints, such as routes or points of interest. This method incorporates fixed values in the height map function and creates an equation system to meet these constraints, maintaining random noise while adhering to user-defined points. This approach enables large-scale terrain generation without losing resolution or increasing computational costs, preserving the procedural generation characteristics. As observed, geometric methods like Perlin noise align with our objectives for high-quality, parallel terrain generation on GPUs. Algorithms such as Diamond-Square and CA show partially effective results, useful for comparison with our proposal.

Regarding Poisson-Disk Sampling, effective for space distribution, is demonstrated in [22] to be highly applicable for graphical tasks, including parallelizable surface distribution on GPUs. This method is valuable for terrain reconstruction in various spaces and shapes.

In Section III, we will discuss our GPU parallel implementation of Perlin noise and analyze its performance. Unlike existing methods, our approach uses Poisson-Disk Sampling to distribute objects rather than map elements, and employs Perlin noise for sequential map design.

III. METHODOLOGY

In this section, we present our terrain generation system, designed to reduce terrain mesh complexity using 3D Perlin Noise and Poisson Disk Sampling. This method enables terrain generation with uniform distribution while optimizing space,

time, and processing speed through GPU parallelization. The process consists of three stages, as shown in Figure 3:

1) Distribution Phase: Random points are distributed using **Poisson Disk Sampling**, with adjustable radius for either high- or low-quality (increasing or decreasing the radius) terrain meshes.
2) Noise Generation: The points are converted into an elevation map using **Perlin Noise**.
3) Terrain Generation: We use the **Delaunay triangulation** to connects the points, forming a mesh where each triangle represents part of the terrain, ensuring detail, efficiency, and fast processing.

A. Distribution Phase

In the distribution phase, we employ the algorithm by [22], which leverages GPU parallelism for virtual terrain processing. The Poisson Disk Sampling algorithm generates a random, uniform distribution of points, enabling the creation of accurate triangle meshes. The terrain input, defined by its contour and size, forms a point cloud P. Then we have T, T represents the number of GPU threads, and r denotes the minimum distance between points (this value can be increased to produce a coarser or finer mesh). Poisson Disk Sampling reduces P into a Poisson Disk Sampling Pattern (PSDP), which serves as the basis for terrain generation. In each iteration, $T[i]$ activates random points and checks distances within r to other points. The *CheckStatus* function updates point states, helping to eliminate points within the radius of active points or resolving conflicts between points selected by different threads simultaneously by prioritizing the one with the higher priority value. Figure 4 illustrates this process.

Each point stores variables necessary for distance calculations and spatial distribution, including its state, a randomly assigned priority value, coordinates, and radius. The priority ensures unbiased distribution, as defined by Equation 1.

$$priority(p_i) = \frac{(rand() * T + i)}{RANDMAX * T} \tag{1}$$

Points are placed according to the terrain dimensions, with each space accommodating a single point. The *CheckStatus* function resolves conflicts by rejecting points that are too close to each other or verifying them against nearby active points. Once validated, the point's state is marked as accepted, preparing the points for the Terrain Generation Phase.

B. Noise Generation Phase

Once the Poisson point distribution is generated, we assign heights to each point to create a height map. This map serves as the foundation for generating smoother and more natural terrain. We adapt the parallel GPU idea proposed in the research of [11] for this purpose.

We start by using the terrain dimensions and point coordinates to create a grid where Perlin noise will be applied. The process begins by overlaying a grid on the terrain. We use an array S of positive integers (ranging from 1 to 256) to represent potential terrain heights. A gradient array G is then created by shuffling S, which introduces variations in frequency and amplitude, resulting in natural terrain features, as shown in Equation 2.

$$\begin{aligned} &G_i = S_i \\ &S_i \leftrightarrow S_j \quad \text{where} \quad j = \lfloor 1 + n \cdot \text{rand}()/\text{RAND\_MAX} \rfloor \end{aligned} \tag{2}$$

To calculate the height at each point using Perlin noise, we consider eight surrounding points that form a grid cell around the target point. These points are used to compute the gradients as shown in Equation 3. The frequency parameter (freq) controls the level of detail: higher frequencies result in more detailed and rough terrain, while lower frequencies produce smoother and less detailed terrain.

$$\begin{aligned} x &= [x^{'}] \cdot \text{freq} - \lfloor [x^{'} \cdot \text{freq}] \rfloor \\ y &= [y^{'}] \cdot \text{freq} - \lfloor [y^{'} \cdot \text{freq}] \rfloor \\ z &= [z^{'}] \cdot \text{freq} - \lfloor [z^{'} \cdot \text{freq}] \rfloor \end{aligned} \tag{3}$$

Random direction vectors are assigned to these eight points to compute gradients. For each of these eight corner points P_i, the gradient is calculated using $m = f(f(f(X) + Y) + Z)$, where X, Y, and Z are the coordinates relative to P_i, as described in Equation 4. The gradient values for these eight points are shown in Table I.

$$\begin{aligned} X &= [x^{'}]^{n-1} \\ Y &= [y^{'}]^{n-1} \\ Z &= [z^{'}]^{n-1} \\ f(i) &= G_i, \quad i = 1, 2, ..., 2n \end{aligned} \tag{4}$$

i	Point P_i	Gradient G_i
1	(x,y,z)	g_1
2	(x-1,y,z)	g_2
3	(x,y-1,z)	g_3
4	(x-1,y-1,z)	g_4
5	(x,y,z-1)	g_5
6	(x-1,y,z-1)	g_6
7	(x,y-1,z-1)	g_7
8	(x-1,y-1,z-1)	g_8

TABLE I: Gradient values for each point in the grid.

Linear interpolation between gradients can sometimes produce undesirable artifacts. To achieve a more natural terrain appearance, we use a smoother interpolation function (Equation 5), where u, v, and w are the smoothing coefficients.

$$\begin{aligned} u &= 6x^5 - 15x^4 + 10x^3 \\ v &= 6y^5 - 15y^4 + 10y^3 \\ w &= 6z^5 - 15z^4 + 10z^3 \end{aligned} \tag{5}$$

The final height L_z at each point is calculated using the interpolation results as shown in Equation 6.

$$\begin{aligned} L_{x1} &= g_1 + u \cdot (g_2 - g_1) \\ L_{x2} &= g_3 + u \cdot (g_4 - g_3) \\ L_{y1} &= L_{x1} + v \cdot (L_{x2} - L_{x1}) \\ L_z &= L_{y1} + w \cdot (L_{y2} - L_{y1}) \end{aligned} \tag{6}$$

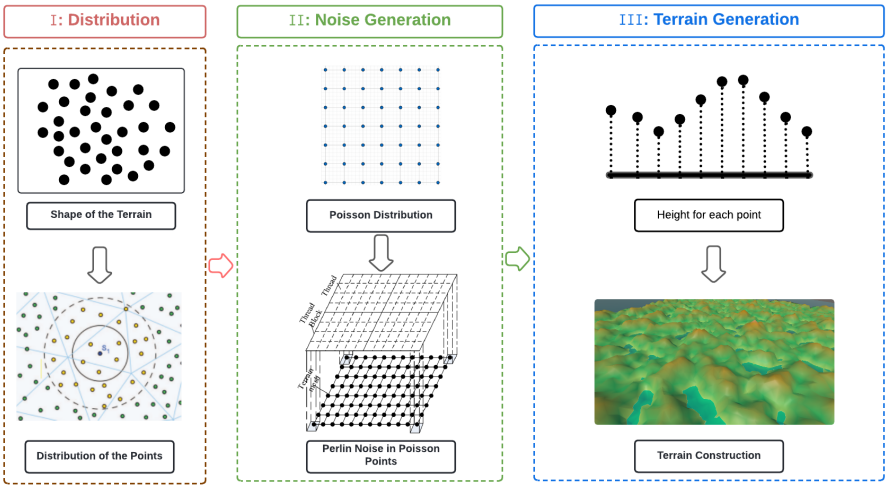


Fig. 3: Pipeline of the proposed method.

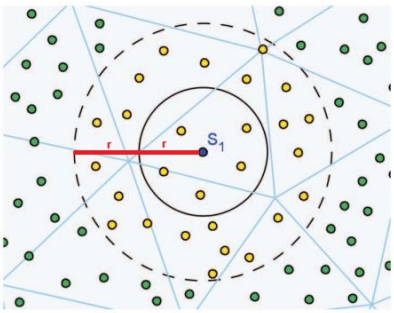


Fig. 4: Processing from [22], where S_i represents the selected point. Yellow points within its radius are eliminated as they are in the *inactive* state.

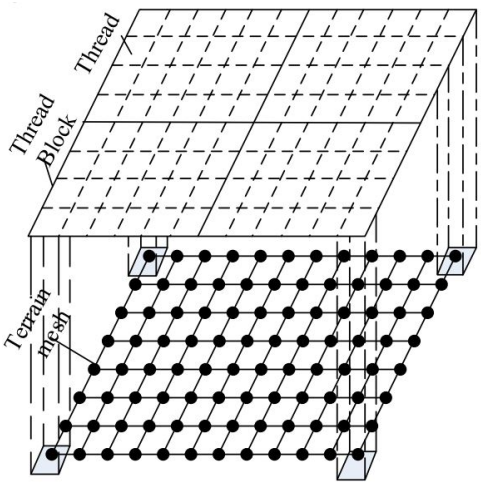


Fig. 5: GPU grid for parallel Perlin noise processing [11].

After computing the heights, the terrain grid is divided into octants. Perlin noise is applied to each section, and each thread processes a single point in space to calculate its height. This method is illustrated in Figure 5, which shows how the GPU grid is used for parallel Perlin noise processing. The resulting height map is then used to generate the final terrain mesh.

C. Terrain Generation Phase

In this phase, with the Poisson distribution and Perlin noise calculations completed, we can proceed to generate the terrain map using Delaunay triangulation.

In this case we use Unity game engine [6] to generate the Delaunay triangulation process. We provide Unity with a point cloud where each point includes its x, y, and z coordinates. The z-value represents the Perlin noise, resulting in a visual representation as shown in Figure 6.

Unity then connects the points to create the triangle mesh. The final terrain model is ready for use in various applications, offering efficient performance in both design and processing, thanks to parallel GPU computations.

IV. Results

In this section, we present the results and evaluations of our research, focusing on three key comparisons with state-of-the-art methods. First, we assess the quality of point distribution, specifically evaluating the Poisson distribution in space. Next, we apply Perlin noise to these distributions on fixed-size terrains. Our proposal aims to generate terrains of various shapes, and as such, the results will showcase different terrain shapes to demonstrate the effectiveness of our method. Additionally, we compare our approach to the Cellular Automata (CA) and Diamond-Square algorithms.

As mentioned earlier, we can define whether we want a high-resolution or low-resolution mesh by adjusting the point radius. In these experiments, we use a radius of 4, allowing us to reduce terrain complexity, which, when processed on a GPU, can be applied in real-time. For terrain rendering, we use Unity with a color gradient ranging from blue (representing low elevations), yellow, green, brown, to white (representing

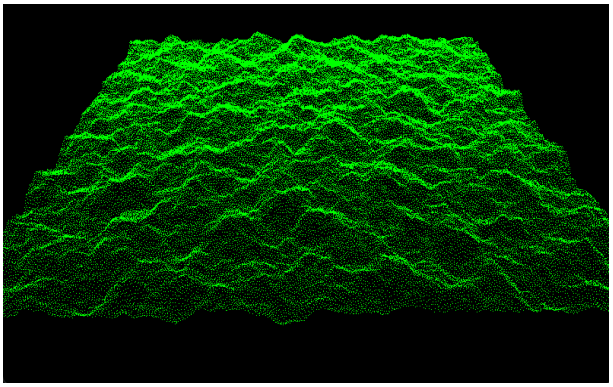


Fig. 6: Poisson distribution points with height after apply Perlin Noise.

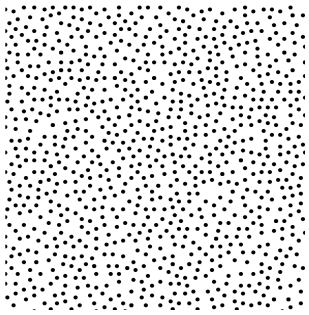
mountains at higher elevations).

First, we analyze the distribution of terrains with regular NxN shapes in Figures 7 and 8, specifically focusing on sizes 500x500 and 1000x1000. For the 500x500 space (Figure 7), the distribution is effective and conflict-free, as confirmed by the Power Spectrum analysis (Figure 7b), which reveals the typical expansion pattern observed in Poisson Disk applications [3]. This validates a high-quality distribution that serves as a solid foundation for terrain generation using Perlin Noise.

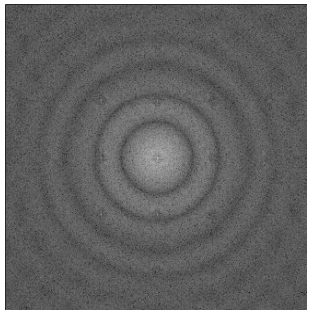
Initial verification (Figure 7c) shows a well-structured Poisson distribution exhibiting blue noise properties, progressing smoothly from low to high frequencies. By applying Perlin Noise to the points from this distribution, we generate a coherent terrain (Figure 7d), suitable for graphical applications. While increasing the radius tends to decrease the map's quality, reducing it leads to improvements. Nevertheless, the overall coherence of the terrain remains intact, making it adaptable for various devices and scenarios.

In the 1000x1000 space (Figure 8a), a similarly effective distribution is observed, with no point intersections, following Poisson characteristics. The corresponding Power Spectrum (Figure 8b) shows the typical Poisson noise pattern, though with fewer circles due to increased point and radius distances. The low-frequency spectrum again confirms blue noise properties (Figure 8c). As with the previous case (Figure 8d), combining Poisson with Perlin Noise proves efficient for generating terrains with simpler but coherent meshes.

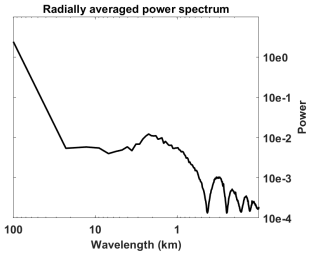
As mentioned earlier, a key advantage of our proposal is that Poisson distribution allows us to generate terrain based on almost any type of continuous or discontinuous shape. This ensures that the mesh simplification is successful regardless of the terrain shape, as demonstrated in Figures 9 and 10. In Figure 9, we see a complex geometric shape known as *Charizard*. Starting with the defined shape 9a, we apply Poisson Disk Sampling to generate the point distribution shown in 9b. Then, Perlin Noise is used to assign elevations to each point, resulting in the simplified triangular terrain mesh shown in 9c. Finally, we provide a close-up of the generated terrain details in 9d. Figure 10 presents a less complex, more continuous



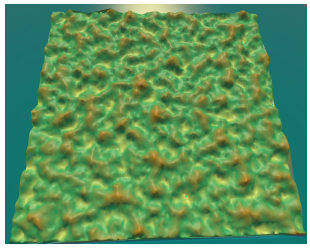
(a) Poisson distribution 500x500.



(b) *Power Spectrum.*

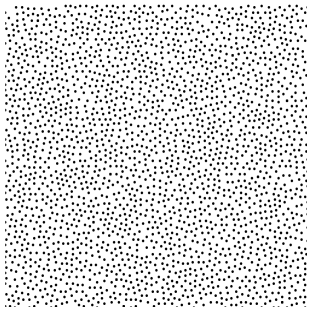


(c) *Radially averaged.*

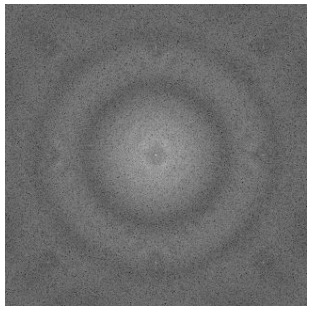


(d) Final Terrain.

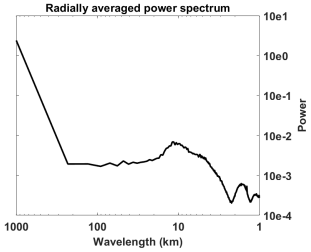
Fig. 7: Complete generation proposed 500x500.



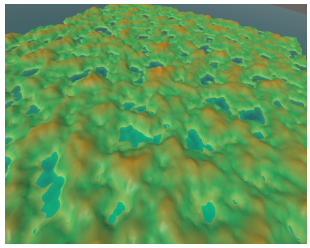
(a) Distribution in 1000x1000.



(b) *Power Spectrum.*



(c) *Radially averaged.*

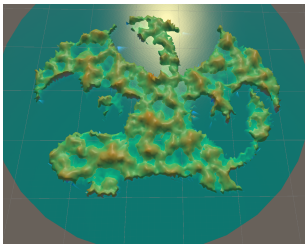


(d) Final Terrain.

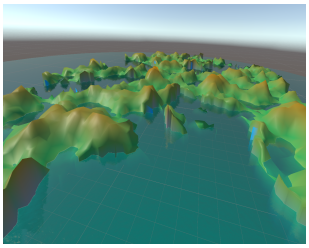
Fig. 8: Comparison of different noises.

(a) Charizard Shape.

(b) Poisson Distribution



(c) Terrain generated.

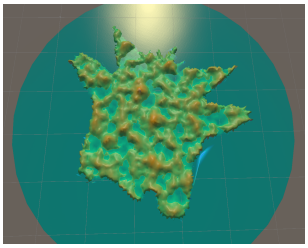


(d) Terrain zoom.

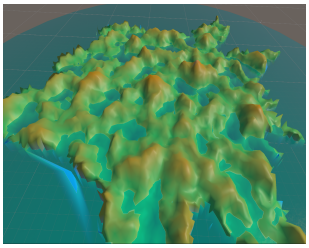
Fig. 9: Charizard terrain generation.

(a) Gengar Shape.

(b) Poisson Distribution



(c) Terrain generated.



(d) Terrain zoom.

Fig. 10: Gengar terrain generation.

shape known as *Gengar*. Following the same process, we first create the terrain distribution 10a, then generate the simplified mesh using Poisson Disk Sampling 10b, and apply Perlin Noise to render the final terrain in 10c, with a detailed view provided in 10d.

Given the visual results from both regular terrains and those based on geometric shapes, the generated terrain has shown coherence between heights and the final rendered scenario, as demonstrated in Figures 7a, 8a, 9, and 10. This approach effectively reduces mesh complexity while proving that the combination of these two methods produces customizable and high-quality terrains.

A. Comparison with Diamond-Square and Cellular Automata

We compared the processing times of our terrain generation method with those of Cellular Automata (CA) and the *Diamond-Square* algorithm. Our approach utilizes Perlin noise on the GPU following Poisson Disk Sampling, and we tested it on terrains of sizes 512x512, 1024x1024, and 2048x2048. Unlike the other methods, our approach adapts to non-predefined terrain shapes. As shown in Figure 11, CA exhibited the slowest processing times, followed by *Diamond-Square*, while our method proved to be the fastest. Table II summarizes the complexity of each method, highlighting that our implementation supports up to 400 threads, facilitating real-time execution through parallel processing. In terms of complexity for Poisson Disk Distribution, we consider $N = 50$, focusing on points within a radius of $2r = 4$ and utilizing Euclidean distances. This adjustment lowers the complexity from $O(n^2 \log n)$, as noted in the original work [22]. Despite the 400-thread limitation of our GPU, the overall complexity remains manageable. For Perlin noise, the number of threads is adjusted according to the size of the space and multiplied by 8 to account for iterations that measure terrain smoothness, where more iterations yield smoother terrain.

The *Diamond-Square* algorithm is effective for generating terrain in smaller spaces, producing consistent and visually appealing results. Its complexity is $O(n^2)$ and significantly increases with larger terrain sizes due to its sequential nature, with overall complexity influenced by the number of points and iterations used. For Cellular Automata, n represents the number of processed points, while m denotes the number of generations. Although increasing the number of generations improves terrain quality, it also escalates processing costs. Our approach leverages GPU processing, which reduces complexity through parallelism, enabling real-time applications and significantly lowering complexity compared to sequential methods.

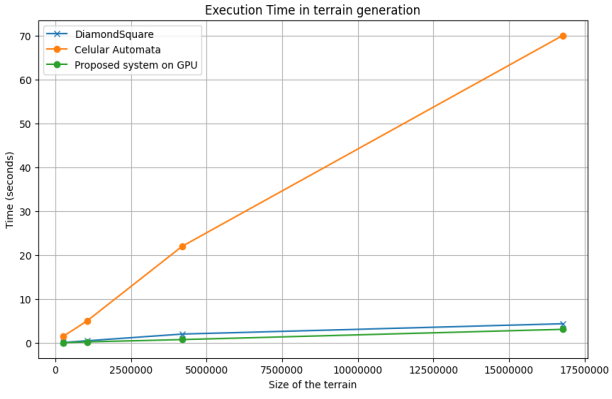


Fig. 11: Execution times for the best evaluated techniques in a logarithmic point chart.

Regarding the mesh for our **proposal**, its statistics can be observed in Table III. As shown in the second column, the number of vertices we have managed to optimize or simplify is

TABLE II: Comparison of the complexity of different terrain generation methods: Poisson GPU Proposal

Dimension (Points)	**Poisson GPU Proposal (L = Points, N = Points in the radius, Num. Threads = Dimension, n = Terrain size , m = iterations)**
Proposal	O(L + 400 x N=50) + O(Num. Threads x 8)
Diamond Square	$O(n^2)$
Cellular Automata	O(n x m)

approximately 4.13% of the total terrain size, and the number of triangles is just over double the number of vertices. This demonstrates that the proposal aims to reduce complexity in terrain generation. Increasing the radius of the points results in more simplified terrains, while decreasing the radius increases terrain definition but with greater complexity in the mesh. In contrast to other methods, where the meshes for terrain generation encompass 100% of the total terrain size, as seen in Table IV, the number of vertices equals the terrain size, and the number of triangles is double that figure. Although the terrains generated by these methods may yield good results, their meshes tend to be more complex, which can hinder optimization on low-performance computers.

TABLE III: Mesh Statistics in the Proposal

Dimension (Points)	**Number of Vertices**	**Number of Triangles**
262,144	12,185	24,368
1,048,576	46,074	92,146
4,194,304	173,200	346,398

TABLE IV: Mesh Statistics in Other Methods

Dimension (Points)	**Number of Vertices**	**Number of Triangles**
262,144	262,144	524,286
1,048,576	1,048,576	2,097,150
4,194,304	4,194,304	8,388,606

V. Conclusion

In this research, the primary objectives were successfully achieved. A simplified 3D terrain generation method was developed using *Poisson-Disk Sampling* and *Perlin* noise, both parallelized on the GPU. This enabled significantly faster spatial distribution and terrain generation, nearing real-time performance despite increasing thread complexity. The results show coherent, stochastic spatial distribution and customizable terrains, as seen in Figures 9, 10, 7d, and 8d. Compared to *Cellular Automata* (CA) and *Diamond Square*, our method reduces both processing time and complexity. It is adaptable to low-resource systems and scalable by adjusting the point radius, reducing complexity by our system up to 4%. However, the method is limited to handling a maximum of 400 threads. Future work will focus on addressing this limitation, exploring new noise generation techniques, and expanding the system for gaming applications while further reducing complexity.

References

[1] Cook, R.L.: Stochastic sampling in computer graphics. ACM Transactions on Graphics (TOG) **5**(1), 51–72 (1986)

[2] Cordonnier, G., Braun, J., Cani, M.P., Benes, B., Galin, E., Peytavie, A., Guérin, E.: Large scale terrain generation from tectonic uplift and fluvial erosion. In: Computer Graphics Forum. vol. 35, pp. 165–175. Wiley Online Library (2016)

[3] Ebeida, M.S., Awad, M.A., Ge, X., Mahmoud, A.H., Mitchell, S.A., Knupp, P.M., Wei, L.Y.: Improving spatial coverage while preserving the blue noise of point sets. Computer-Aided Design **46**, 25–36 (2014)

[4] Gasch, C., Chover, M., Remolar, I., Rebollo, C.: Procedural modelling of terrains with constraints. Multimedia Tools and Applications **79**(41), 31125–31146 (2020). https://doi.org/10.1007/s11042-020-09476-3, https://doi.org/10.1007/s11042-020-09476-3

[5] Gellel, A., Sweetser, P.: A hybrid approach to procedural generation of roguelike video game levels. In: International Conference on the Foundations of Digital Games. pp. 1–10 (2020)

[6] Haas, J.K.: A history of the unity game engine (2014)

[7] Hnaidi, H., Guérin, E., Akkouche, S., Peytavie, A., Galin, E.: Feature based terrain generation using diffusion equation. In: Computer Graphics Forum. vol. 29, pp. 2179–2186. Wiley Online Library (2010)

[8] Hyttinen, T., Mäkinen, E., Poranen, T.: Terrain synthesis using noise by examples. In: Proceedings of the 21st International Academic Mindtrek Conference. p. 17–25. AcademicMindtrek '17, Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3131085.3131099, https://doi.org/10.1145/3131085.3131099

[9] Johnson, L., Yannakakis, G.N., Togelius, J.: Cellular automata for real-time generation of infinite cave levels. In: Proceedings of the 2010 Workshop on Procedural Content Generation in Games. pp. 1–4 (2010)

[10] Le, T.: Procedural Terrain Generation Using Perlin Noise. Graduate project, California State Polytechnic University, Pomona (2023), http://localhost/files/r494vs85w, scholarWorks

[11] Li, H., Tuo, X., Liu, Y., Jiang, X.: A parallel algorithm using perlin noise superposition method for terrain generation based on cuda architecture. In: International Conference on Materials Engineering and Information Technology Applications (MEITA 2015). pp. 967–974. Atlantis Press (2015)

[12] Minarno, A.E., Agisna, A.R., Kusuma, W.A., Suharso, W., Wibowo, H.: Optimizing game performance with dynamic level of detail mesh terrain based on cpu usage. In: 2020 3rd International Conference on Intelligent Autonomous Systems (ICoIAS). pp. 93–98. IEEE (2020)

[13] Mitchell, D.P.: Generating antialiased images at low sampling densities. In: Proceedings of the 14th annual conference on Computer graphics and interactive techniques. pp. 65–72 (1987)

[14] Newzoo: 2023 newzoo free global games market report (2023), http://www.daelab.cn/wp-content/uploads/2023/09/2023$_N ewzoo_F$ $2024 - 08 - 03$

[15] Ong, T.J., Saunders, R., Keyser, J., Leggett, J.J.: Terrain generation using genetic algorithms. In: Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation. p. 1463–1470. GECCO '05, Association for Computing Machinery, New York, NY, USA (2005). https://doi.org/10.1145/1068009.1068241, https://doi.org/10.1145/1068009.1068241

[16] Panagiotou, E., Charou, E.: Procedural 3d terrain generation using generative adversarial networks (2020)

[17] Shaker, N., Togelius, J., Nelson, M.J.: Procedural content generation in games. Springer (2016)

[18] Statista: Number of games released on steam 2018 — statistic. https://www.statista.com/statistics/552623/number-games-rel (2018), acessado em: 15/05/2019

[19] Talgorn, F.X., Belhadj, F.: Real-time sketch-based terrain generation. In: Proceedings of Computer Graphics International 2018. p. 13–18. CGI 2018, Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3208159.3208184, https://doi.org/10.1145/3208159.3208184

[20] Wilson, J.P.: Environmental applications of digital terrain modeling. John Wiley & Sons (2018)

[21] Yahya, N.M.H.H., Fabroyir, H., Herumurti, D., Kuswardayan, I., Arifiani, S.: Dungeon's room generation using cellular automata and poisson disk sampling in roguelike game. In: 2021 13th International Conference on Information & Communication Technology and System (ICTS). pp. 29–34. IEEE (2021)

[22] Ying, X., Xin, S.Q., Sun, Q., He, Y.: An intrinsic algorithm for parallel poisson disk sampling on arbitrary surfaces. IEEE transactions on visualization and computer graphics **19**(9), 1425–1437 (2013)