Everling Noise: A Linear-Time Noise Algorithm for Multi-Dimensional Procedural Terrain Generation

Cassio Dalla Barba Everling
Undergraduate Computer Science Student
University of South Florida
dallabarbaeverling@usf.edu

Abstract - Procedural terrain generation often relies on noise algorithms to simulate natural randomness and visual complexity. This paper introduces Everling Noise, a novel method that reverses the traditional noise design paradigm: instead of building smooth transitions between predefined altitude points, it integrates Gaussian-distributed altitude variations to reconstruct terrain. The algorithm exploits the Central Limit Theorem to ensure natural-looking transitions and supports parameter tuning through mean and standard deviation modulation. Unlike Perlin or Simplex noise, which suffer from exponential time complexity in higher dimensions, Everling Noise achieves O(n) time complexity per dimension, making it highly scalable. Experimental results demonstrate a wide range of visual outcomes - from clustered archipelagos to fractal mountains - achieved by varying integration strategies and traversal methods. The technique's efficiency and flexibility make it especially suitable for indie games and applications requiring stylized, yet geographically plausible, procedural worlds.

Keywords - procedural generation, noise maps, terrain generation, Gaussian noise, Central Limit Theorem, real-time graphics, multi-dimensional algorithms, Everling Noise

I. INTRODUCTION

Many video-games have their concepts built around giving the player a new experience every time a new play thought is started. Rogue-like games are a good example of this principle, every single time the player starts the game, a new dungeon is created, similar but always different. Meanwhile open world games such as Minecraft and Terraria give birth to an entire new world that keeps its consistency without making use of pre-build sections. Both are use case examples of implementing controlled randomness algorithms, to reassemble real features, however in two they work in two completely different ways.

In dungeon games, it would feel unnatural for the player if a certain level would have two rooms the first time he entered it and hundreds the next. Similarly, adjacent rooms' difficulty can not vary drastically without disrupting the game's inner logic. Yet, if they were to always be too similar, they would also seem artificial. The need for such balance highlights a key property of natural systems: they tend to follow normal (Gaussian) distributions. Procedural dungeon generation algorithms therefore rely on normally distributed random variables to create a world variable and coherent [1].

In contrast, most noise does not deal with the actual feature distributions, focussing, rather, on the smoothness of gradients created, allowing for the generation of complex terrains and features without a predefined rule-set. These maps provide a spatial relation on a changing variable, which can later be processed to acquire different meanings - altitude, vegetation density, biome - based purely on mathematical noise functions. Efficient and multi-purposed, noise algorithms are found in various areas outside of game development, such as procedural animations and simulations. However, despite their popularity, they still fail to properly represent reality.

This paper introduces a novel category of noise map generation algorithms that merges statistical modeling with terrain traversal techniques. The proposed approach is designed to offer both efficient scalability to higher dimensions and flexible parameter tuning - two features often lacking or computationally expensive in traditional noise generation methods.

II. NOISE MAPS

Noise maps are grids filled with values that follow some distribution, often used to simulate natural variation. Instead of designing every feature by hand, noise allows the computer to generate complexity from simple rules. Whether for landscapes, textures, or even behaviors, noise maps are a fundamental building block in procedural content generation.

A. Application on Terrain Generation

In terrain generation, each value in the noise map can be interpreted as a height. That alone is enough to produce rolling hills or jagged cliffs depending on how the map was built. The output doesn't need to be realistic in every case—it just needs to look organic. With some coloring and smoothing, even a raw noise function can define entire worlds.

B. General Uses in Procedural Generation

Besides terrain, noise is used to drive randomness in a controllable way. It can decide where to place trees, how a cloud changes shape, or the density of an enemy spawn zone. What makes it powerful is the balance between unpredictability and structure - noise can make every instance different, while still obeying the same rules.

C. Perlin Noise

In 1983, another software developer was faced with a similar challenge. Ken Perlin approached the issue with the following perspective: instead of using completely random values, we could assign gradients and interpolate between them. This gave rise to Perlin Noise, a coherent noise function that looks natural at multiple scales. It became widely adopted in graphics due to its controlled randomness. [2][3]

D. Other Noise Maps

Other approaches tried different trade-offs. Value Noise uses random values at grid points and smooths them out. Simplex Noise, also by Perlin, improved performance in higher dimensions [4]. Worley Noise builds distance maps and is often used for cellular textures or erosion-like effects. Depending on what you're building—mountains, cracks, clouds—each type has a best fit [5].

E. Gaussian Noise

Gaussian noise takes a different route. Instead of structure, it leans into randomness. Its values follow a bell curve, which makes it useful when modeling natural variation statistically. In procedural generation, Gaussian noise can add subtle displacements or be used to break symmetry, even if it's not the best tool for building coherent shapes on its own [6]..

III. EVERLING NOISE

Although the presented noise functions have proven themself efficient along the decades and up to this day, my approach to the presented issue hardly agrees with his. Meanwhile the focus of existing algorithms is defining peaks and valleys first and then implementing various mathematical methods to achieve a smooth gradient in between them, the problem of terrain generation can also be understood as valleys and peaks being the fruit of a smooth terrain, rather than its cause.

Despite not essential for the development of regular software, an implementation of noise maps based on a new interpretation of how to represent ground fluctuations can give us improvements worth pursuing.

Since every noise function creates a different kind of texture pattern - such nuances are evident when Perlin Noise is put side to side with Pink Noise (Figure X) - new noise generators are usually found out to be perfectly tailored for uses in which others require loads of post processing. Considering that is true for algorithms that essentially implement the same background concept, when writing it in a fundamentally different way, the results are likely to be even more disruptive.

On top of that, noise algorithms are known for their poor efficiency. The widely used Perlin Noise has a time complexity of $O(n2^n)$ for n dimensions, making it hardly viable to be used in anything more than a 3D noise. Even though some solutions - for example, the simplex noise - have been accomplished to reduce the time complexity to $O(n^2)$, there. The proposed algorithm aims for an increased time complexity, aching a linear growth to higher dimensions given a constant amount of numbers generated.

A. Normal Distributions

Most features in reality are modeled after a normal distribution. This raises the possibility of describing terrain in terms of gaussian numbers, just as used on many procedural dungeon generation algorithms. However, there is no clear indication that altitudes actually do follow such a pattern, especially when increasing the map dimensions and attempting to represent multiple geological structures, they may do but also may not. Without a solid scientific foundation any algorithm that aims for a realistic terrain generation must not make any assumption regarding that, because if it is not universally true, the creation of some kinds of land may be harshened and even made impossible.

But despite the nature of the original distribution - in our case the altitudes distributions - the mean of the means of a feature across all samples in a population are normally distributed, as stated by the statistical Central Limit Theorem [7]. Therefore, if the height means across an area should be normally distributed, however this is still not a useful conclusion, since altitudes are not randomly distributed across space, jumping from a high mountain to a deep ocean is a rare event in nature, yet it would be a frequent outcome in a generative algorithm that models altitude values directly using a random distribution. On the other hand, modeling the mean change in altitude offers a more consistent and natural representation: in large areas, this change tends to vary smoothly, and when abrupt transitions are needed (such as a cliff or a plateau), they emerge from local variations rather than random height jumps. In contrast, applying randomness directly to altitude could result in isolated and unnatural terrain spikes.

In a single dimension, getting the mean change of altitude is rather simple. Since only one data point represents each location, the mean change of altitude is no different than the absolute value of the difference of those two points. Increasing dimensions, using the mean instead of the actual

changes becomes relevant. The mean becomes a multi-dimensional mean.

$$\Delta h_{mean} = \frac{\frac{h_{x_{-1}} - h_{x_0}}{2} + \frac{h_{x_1} - h_{x_0}}{2} + \frac{h_{y_{-1}} - h_{y_0}}{2} + \frac{h_{y_1} - h_{y_0}}{2}}{4}$$

Since h_{x0} and h_{y0} refers to the same data point, it can be rewritten as

$$\Delta h_{mean} = \frac{\frac{h_{x_{-1}} - h_0}{2} + \frac{h_{x_1} - h_0}{2} + \frac{h_{y_{-1}} - h_0}{2} + \frac{h_{y_1} - h_0}{2}}{4}$$

Writing h_0 in terms of the change in altitude:

$$h_0 = \frac{h_{x_{-1}} + h_{x_1} + h_{y_{-1}} + h_{y_1}}{4} - 2\Delta h_{mean}$$

Due to the natural inconsistency of terrain variation across the world, no single uniformly smooth texture could possibly be used to generate a diverse world, such as the one we live in. Creating a mountain range, then, requires completely different parameter tuning than generating plains, beaches and oceans.

However, a solution can still exist. Usually, when populating a procedurally generated world with diversified geological structures, a biome is set for the chunk before its terrain starts being processed based on a biome-specific tuned noise map. This process requires intermediary structures in between the main biomes to achieve a smooth transition. But, if the terrain's altitude variation is sourced by a normal distribution, providing an auxiliary function that describes how the mean and standard deviation change across space would offer an even more natural biome transition then the one seen in games such as Minecraft and Terraria.

Such possibilities further highlight why developing a sound algorithm based on average altitude variation instead of gradients can result in a solution perfectly fit for certain procedural generation scenarios.

Summing up, it does not matter if the final terrain altitude or height changes are normally distributed or not, most likely they won't be - such as natural terrains, with huge shifts and unpredictable corners. The fundamental concept is: in order to create a noise map that represents a terrain, this geographic structure has to be divided in same sized lines, squares or cubes, which can be infinitely smaller, but never just a dot. This allows for the mean difference in altitude to be calculated, and if that is done on the whole map, all means will, according to the Central Limit Theorem follow a normal distribution. This is the key point that allows us to revers-se engineer this process and go from normally distributed random numbers into a naturally looking terrain, independent of the nature of the distribution of its altitudes.

The mean difference of altitude to all surrounding points should be normally distributed, despite even the nature of the distribution of height variations. When generating a one dimensional noise, this mean will just be the variation. But when increasing dimensions, the mean of all sides must be considered.

B. Mountain Profile - 1D

The simplest way to test the concept behind Everling Noise is by gradually scaling it up, starting with a one-dimensional implementation. Working in a single dimension allows for a simplified version of the algorithm to be tested without the added complexity of handling multiple unknown averages or multi-dimensional surfaces.

This process is analogous to drawing a mountain on a sheet of paper. Starting by one point and moving sideways, vertical movements will establish movement change points. Provided that such points are evenly spaced, they can be used to compute the mean difference of altitude between every two consecutive points.

Using only one adjacent point to determine the new value can seem odd at first, since in our mathematical derivation, the two adjacent values should be considered. But in one dimension, by starting from a single data point and one axis of movement, at any given point, one side (the past point) has a known value, while the other (the future) is yet unknown, which can not be computed into the average, resulting in the same outcome as using only one point. This allows for a more optimized implementation of the same algorithm.

A simple coding implementation can break down the process into two main components. First, a Normal Noise of length *n* is created, i. e., an array of size *n* is created and populated with normally distributed numbers, which will so on be called *Step Map*. Then, the noise is cumulatively summed to generate a terrain profile — a 1D elevation map. Simply defining a random index of the array to have a predefined value and then adding up its value to the adjacent noise should result in the desired outcome.

In a calculus-based interpretation, the whole process is no different than building a function. Assuming that there exists a continuous function *f(x)* that describes the altitude variation of a surface, there is also a function *f'(x)* which perfectly informs how the altitude changes. It is important to notice, at this point, that when integrating over a derivative, we get the original function up to an unknown *C*, which in the case of a surface level function corresponds to the original height. Therefore, the one-dimensional implementation of the

Everling Noise consists in generating a Gaussian distributed white noise - *step map* - to represent the surface derivative, integrating this noise to reconstruct the surface profile, and finally adding the missing constant - original altitude.

1. Algorithm Outcomes

The integration process is clear on images _ and _ built by, in the left, terrain variation and, in the right, the final landscape. Despite being built from a similar standpoint, they drastically differ on altitude and smoothness. This is a clear consequence of changes in the step map stand deviation and the original altitude. This simple changes allows for the creation of plains and mountain chains.

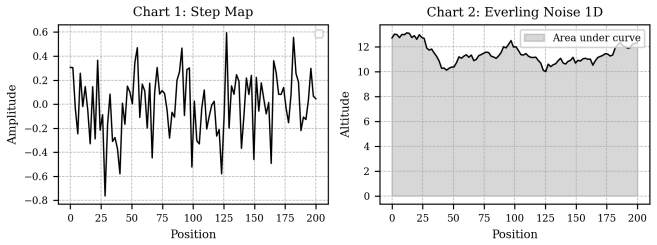

Image 1: On the left, DFS based Everling Noise considering diagonals.

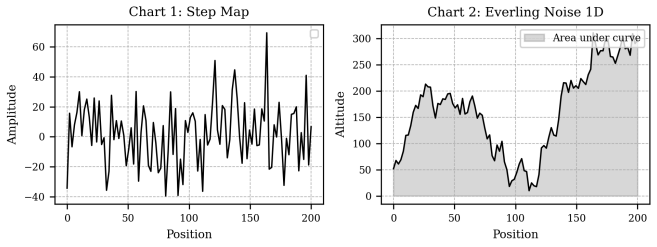

Image 2: On the left, DFS based Everling Noise considering diagonals.

Testing passing functions into the mean instead of a single value. This kind of wraps the noise around another function, but is useful for generating .It is likely to be a feature more useful when creating 1D noise maps. Nevertheless, it does provide interesting possibilities and allows for the creation of more complex geological structures. In the following examples,

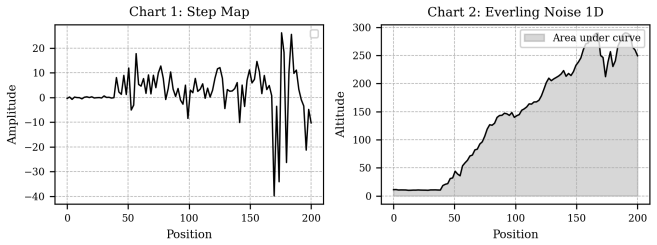

Image 3: On the left, DFS based Everling Noise considering diagonals.

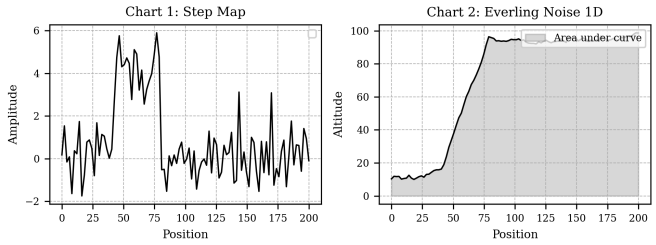

Image 4: On the left, DFS based Everling Noise considering diagonals.

2. Time Complexity

For analysing the performance of the algorithm, we begin by an asymptotic analysis. When generating a sequence of n numbers, the base implementation of the Everling Noise visits every position of the array twice, first for generating the base Gaussean Noise - performing a constant number c_1 operations every time - and second when adding up to the adjacent position - performing c_2 operations. Adding those numbers, the algorithm will execute $c_1 + c_2$ operations for every n index in the list. Therefore the estimated time complexity of the algorithm is $O(n)$ for an 1 dimensional map.

This theoretical conclusion is supported by graph _, that plots the time usage for a Python implementation of the unidimensional Everling Noise with constant mean, run 10000 times with increasing array size. The resulting correlation index of — indicates a high linear correlation.

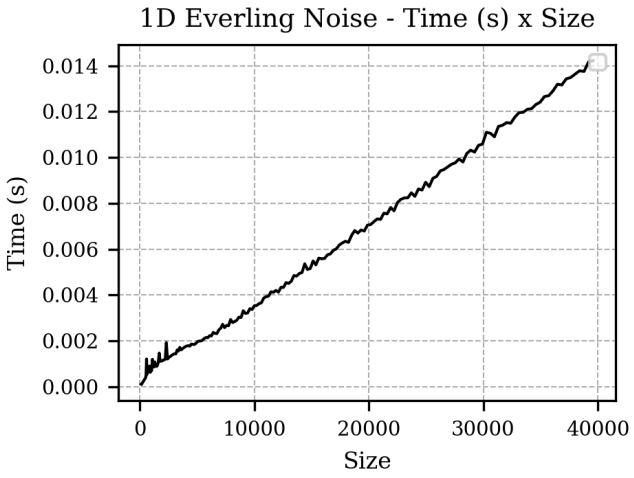

Image 5: Time performance of 200 1D Everling Noise samples of increasing size from 10 to 200[2]

C. Map Generation - 2D

Similar to the one in one dimension, the approach used for generating terrain fluctuations over a two dimensional plane, consists in walking through a plane of gaussian-distributed

altitude variations. The key difference is that with one extra dimension, each cell receives one extra degree of freedom and influence, i. e., the altitude difference calculated is not only relative to one point, rather, it should be taken into account its surrounding values in both axes and once the altitude is computed no single course of action is defined, since there can be more than one adjacent cell with unknown value.

Supported on the statistical background established, the value of a given node can be calculated as the average height of the nodes that share a square side with it plus the noise variation. When considering a square, the mean height is calculated by the sum of, which can be stated as the average of that two x-axis adjacent cells averages with the average of the mean of the two adjacent y-axis adjacent cells. This means that, in order to have a well balanced terrain, there is no need for computing diagonals in the calculation.

Furthermore, there is mathematical support for not including diagonal values into the average computation. Since every point suffers direct influence over its value by surrounding cells, adding them to the mean calculation would have no impact other than increasing the influence from distant data - in the case that they were computed before the side connecting nodes - or adding its normal variation to the mean calculation which would be redundant, since the all normal values surround equality the same mean. Such similarity is supported by Images 6 and 7, showcasing DFS implementation of the *Everling Noise* run with the same seed.

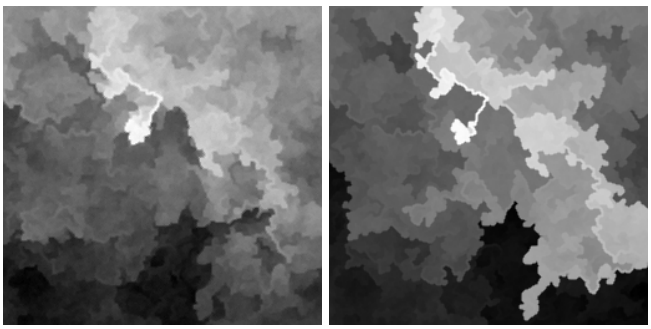

Image 6: On the left, DFS based Everling Noise considering diagonals. On the right, DFS based Everling Noise not considering diagonals.

In the left side of image 6, the adjacent diagonals are considered when averaging the values, but not offered as possible paths for the DFS. On the right side, on the other hand, diagonals were considered as not adjacent throughout the whole code. Results show similar over all patterns on both, but when diagonals are considered, it is as if an after smoothing filter was applied. Since this difference comes at the cost of an increased runtime duration and has similar effects to an average smoothing function, it will be treated as such.

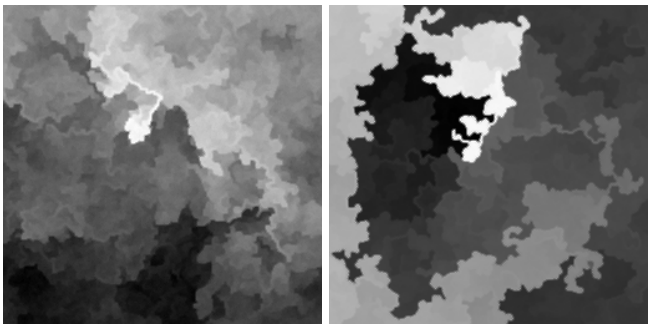

Image 7: On the left, DFS based Everling Noise considering diagonals. On the right, DFS based Everling Noise considering and moving on diagonals.

Image 7 compares the same map generated with averaging diagonal used on Image 6 to one with the same random seed but diagonals were also considered as possible paths for the DFS. The outcome could not be more clear, the unique smoothing effect achieved on the lefthand side is completely lost on the right-hand side, producing patterns that can not be clearly distinguished then the ones in Image 6, where diagonals were not considered at all.

Another important nuance that comes with scaling the algorithm to two dimensions is defining in which order cells should be submitted to the integration process. Since this refers to how cells are accessed in the matrix, this process will be referred to as *access methods*. *Access methods* should both be fast and controlledly random.

An option that fulfills the first requirements is implementing a depth first search (DFS) or breath first search (BSF) looking through the entire array. Both of them, although producing different styles of image - radial for BFS, expanding the data influence on circles, and spiral for DFS, looking though expanding on a line of influence -, are too neat for a noise image, giving it an unnatural look. That could, however, be solved if instead of appending adjacent nodes in a standard form, they were shuffled first. But this comes at the expense of performing extra 4 operations per node, drastically impacting performance, despite also running into the problem of checking a node more than once. The solution for that: first checking if the position is already loaded on queue or stack through a separate hashset or boolean matrix[1] before shuffling and appending, this way, every node will be shuffled only once instead of 4 times.

An alternative is to use random access on an array of future positions[2]. This would result in a less controlled

[1] Since the final size of both structures has to contain all the matrix positions, picking a matrix has no disadvantages and avoids performance leaks due to changing the hash set size.

[2] Unlike dynamic containers such as lists or queues, arrays allow constant-time random access. Although resizing and deletion can be costly,

outcome, that would however tend to have a radial image around the origin point, since that there is a higher probability of a cluster of unknowns to form around that. Another option, is to use gaussian numbers instead of completely random ones, this gives extra control over the chosen array index while keeping an extra layer of unpredictability when compared to DSF and BFS. If the peak of the gaussian curve is set to be in the most recent numbers, it would resemble the results of the DFS, and, in the beginning, the BSF without any meaningful change in performance, since the random number use on shuffling would just be reused for picking an index.

A mix of all of the discussed techniques is possible. By controlling the probability of different algorithms to be applied, the flexibility of the Everling Noise highly increases, allowing for special tunings depending on what the situation demands.

1. Algorithm Outcomes

Due to the many ways in which the starting *step map* can be generated and the different walk through process that can be performed in the Evelring Noise, different parameter tunings can imply drastic changes to the outcome. In this section, comparisons will be drawn between different combinations of walking techniques for later analysing the impact of variations on gaussian means and standard deviations functions. In this process

For analysing the impact of different, direct comparisons will be drawn from stating from the same point in the same gaussian plane, performed with the same random seed. This assures that all nuances in the outcome are exclusively consequences of the paths taken on integrating the *step map*. Also, the use of topological maps will be applied in order to provide a clearer understanding of the maps' behaviours.

For the first comparison, the algorithm was divided into two parts: generating the step map and performing the integration walk. In Image X, the step map displayed in the top left has dimensions of 128×128, a mean of 0, and a standard deviation of 0.3. The three different path systems were applied to this same map. This allows for a fair comparison of the patterns that emerge in images generated by different tunings of *Everling Noise*.

these are avoided by preallocating the maximum size and using a control variable to track the effective length. Deletion is implemented by swapping the removed element with the last active element and decrementing the control variable.

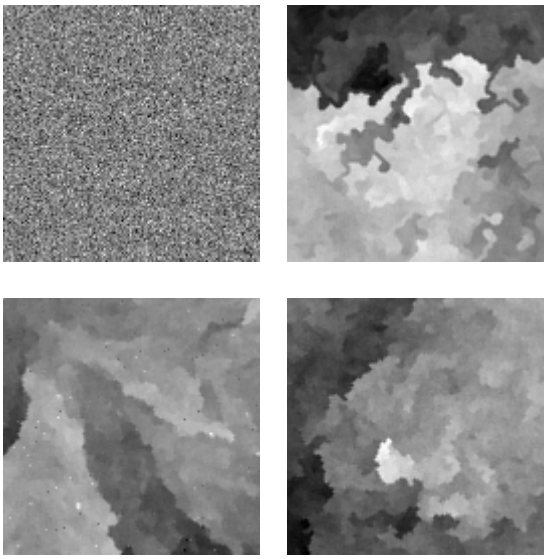

Image 8: Step Map (1), Stack Access (2), Random Access (3) and Center Gaussian Access - sd 0.3 - (4) all with Gausean Map with mean 0 and standard deviation 7, python random seed 500, dimensions 128x128.

As expected, the outcomes are visually distinct (Image X). The stack access variant (top right) produces a highly fractal pattern, with vein-like structures that emphasize elevation transitions. In contrast, random access (bottom left) yields a smoother, more radial image, resulting from its circular integration walk behavior. Finally, random access with the mean centered on the lookup array (bottom right) exhibits a much smoother and more clustered structure, while still preserving fractal characteristics in its cloudy, organic appearance. Each of those patterns represent a terrain that is different in nature and ultimately, their combination can give birth to even more complex and realistic environments.

The height differences within the result of the random access variation produces visuals with strong erosion patterns, but there is lack of continuity and strong radial look, which only gets more visible as map size increases. Meanwhile, the stack access offers quite the opposite, with a strong continuity and fractal nature, spawning almost vain-like structures that tend to repeat himself disregarding how far it is from the origin point. Mixing both so that for every cell there are equal chances that each method will be chosen for picking the next should mix both results, keeping the continuity of the *DFS Access* but also giving it some erosion patterns. Image 8 represents this combination producing a cloudy outcome that somewhat resembles the *Normal Access*, way more sparse peaks.

To better represent the use of the *Everling Noise* in terrain generation, the noise map was converted into a topological map - shown on the right -, where every color represents a set altitude range. When this simple filter is applied to the noise map, some characteristics of the noise

generated get more evident. In image 8, the topology has more visible similarities with its two integration components than the noise itself. The islands of the archipelago generated keep the fractal and curvy aspect of the *DFS Access*, as well as its continuity, seen majorly in the second diagonal of the map, while also having erosion patterns, breaking long chains of similar height into lots of small islands with clear water erosion.

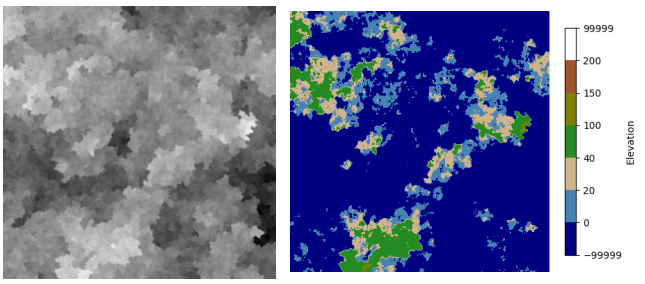

Image 9: 256 x 256 Noise Map (50% Random Access + 50% DFS Access) mean: 0; sd: 3

Similar erosion patterns could also be applied to the *Gaussian Access* integration method if mixed with *Random Access*. Since, unlike *DFS Access,* it does not have a strong continuity, the main radial pattern persists. As shown on Image 9, the noise resembles more of a radially spaced *Gaussian Access* or even an eroded version of the *Random Access* noise then what was intended. This impacts that the bigger the map with these settings, the more triangular shaped the pieces of land are. However, if the balance is changed to favor the least strongly spatially distributed, the scenario changes.

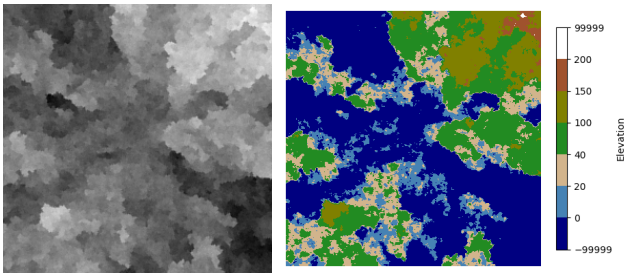

Image 10: 256 x 256 Noise Map (0.5 normal access, 0.5 random access) mean: 0; sd: 3

On image 10, *Normal Access* is four times more likely to be chosen over its completely random counterpart. This version of combined access nails the initial goal of producing a version of the *Normal Access* original cloudy, clustered and evenly distributed visual affected by erosion patterns. In the gray scale noise map, the radial nature of the *Random Access* is no longer distinguishable, different from Image 9. Yet, it is not the same as purely *Normal Access Everling Noise*, its borders are more rusty and altitude shifts are less smooth.

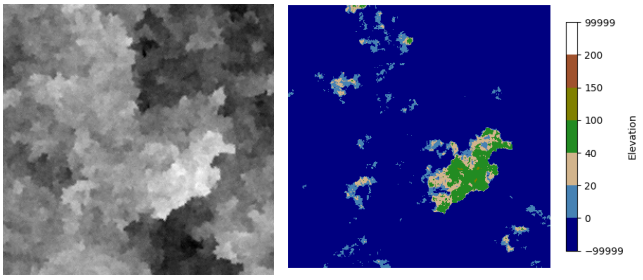

Image 11 : 256 x 256 Noise Map (0.8 normal access, 0.2 random access) mean: 0; sd: 3

Those are amazing outcomes for archipelagos and mountain chains, however they do not profoundly match with transitions from plains to hills because none of those has suffered meaningful recent wind or water erosion. What is needed for such landscapes is a mix of smooth and marked continuous transitions, which are offered, respectively by *Normal Access* and *DFS Access* variations of the *Everling Noise.* Their even mix is showcased on Image 10, where the cloudy noise is organized in a more conspicuous and curvilinear pattern, highly marked by smooth altitude changes into progressively more visible and aligned fractal patterns. When converted into a topological map, this results in mountains usually smoother on at least one side but sometimes drastic on another, resembling various hill formations commonly seen near the Atlantic border of the American Continent, in the Appalachians of the U.S. and on the Serra do Mar Region of Brazil.

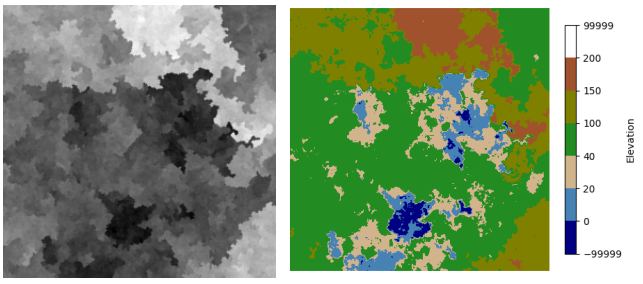

Image 12: 256 x 256 Noise Map (0.5 normal access and 0.5 stack access) mean: 0; sd: 1

Still, a combination of the three methods is likely to produce the most balanced, universal and varied terrains. By adding some amount of erosion patterns to the hills produced on Image 10, kanyon structures can be given birth. At the same time, a slightly more clustered aspect on the noise map of Image 9 could produce an archipelago composed of major and satellite islands, common in active tectonic regions such as the Pacific Fire Ring. These combinations, when tuned,

can produce controlled random noise that properly describe certain geographic regions, but also when in harmony in larger maps, have high probability of producing parts that follow each of these patterns[3], leading to a geographically rich world without huge development complexity.

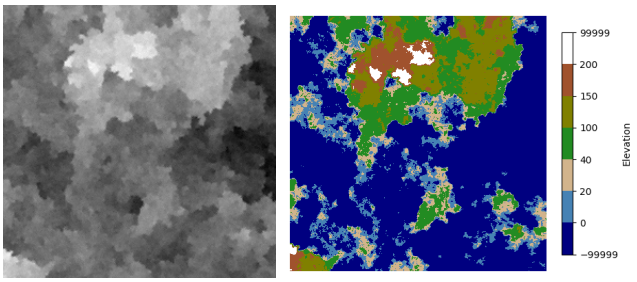

Image 13: 256 x 256 Noise Map (⅓ Normal Access, ⅓ DFS Access and ⅓ Random Access) mean: 0; sd: 3

An even distribution of the three methods is available on image 11, built using an integration access controller with a third chance that each is selected. The gray scale noise map clearly shares aspects with its three components. When looking at the bottom left corner, the *Random Access* rays moving sideways and towards the top center are noticeable. Meanwhile, radial curved corners are present in the entire map, more visible in the central vertical axis of the image. And, elevation clusters, a common characteristic of the *Gaussian Access*, manifests in distinguishable lighter and darker parts.

When processing the topology filter, it results in the most geographically diverse map produced so far. In the same 256 by 256 grid, mountains, plains, canyons, hills, sparse and clustered islands and multi-shaped coast appear. This variety further highlights how an even distribution can shine in small scale maps, specially for creating countries, continents or world maps.

The discussed combinations can surely produce widely distinct noise and topological maps, but they were still tuned by one of the meaningful parameters of the *Everling Noise*. Just as in one dimension[4], changes in mean and standard deviation when producing the step-map can vastly impact the final noise map. Inputting different values and functions as mean rules the overall tendency of the map as it spreads out of the origin, while higher standard deviations make the terrain more uneven.

In image 12, the generation algorithm is the same through the image, despite a change in standard deviation between left and right sides of the map. The simple change from 1 (left-hand side) to 7 (right-hand side) in standard deviation is enough to turn what originally looked like a plain into a mountain chain.

The mean also plays a huge role. It can be used to set general tendencies of the map. With the same standard deviation, it can be the difference between producing a single island (mean < 0) and a valley (mean > 0) around the origin. This is clearly shown when using a full *DFS Access*, which has a circular tendency, together with a negative mean for creating a lonely island within image _.

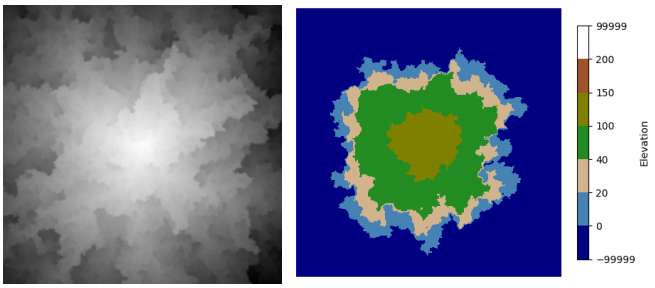

Image 14: 256 x 256 Noise Map (⅓ Normal Access, ⅓ DFS Access and ⅓ Random Access) mean: -1; sd: 0.1

The ratio between the mean and the standard deviation is also relevant and can produce two different scenarios. If the standard deviation is greater than the mean, it is possible for it to counteract the general tendency in certain points. On image _, for example, a greater standard deviation could produce an extra amount of satellite islands. But when the absolute value mean is greater or equal to the standard deviation, the whole map will obey its rule.

Other kinds of tendencies can also be created by implementing the mean and the standard deviations as functions inputting which coordinates of the step map they are generating. It can be used to determine points of change in the behaviour of the landscape or even smooth transitions. Image 15 has constant standard deviation, but mean described by $(x + y) * (x - y)$, leading to a progressively higher inclination on the corners where the difference between x and y are bigger, results in a valley that can be the exact kind of map intended. If the map was expanded, however, the tendency would continue, creating higher and higher mountains, something completely out of reality. Function means should, therefore, be used carefully and may require extra work when compared to constant means. But if there is a clear goal and the developer has a good understanding of

[3] Given a large enough map, all of the described structures are likely to appear because of the random nature of the method choice. If a coin is flipped a few times, the odds of a sequence of *n* heads is hardly going to happen, however if the sample size increases, it becomes progressively more probable. The same happens with picking access choices.

[4] See section III.B.1 for a more detailed discussion, specially in regards to implementing mean and standard deviation as multi variable functions. This is not deeply analysed in this section to avoid redundancy and because of the increased difficulty of visualization of those behaviours in three dimensions.

how the derivative of the terrain behaves and of the integration walk access methods, this can be a powerful tool.

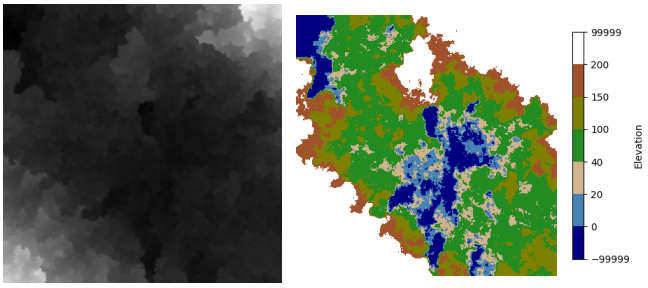

Image 15: 256 x 256 Noise Map (0.5 normal access and 0.5 stack access) mean: $(x + y) * (x - y)$**; sd: 3**

Overall, it is safe to argue that the *Everling Noise* is capable of producing almost any kind of landscape with high fidelity. Further experimentation with mean and access methods can lead to even more stunning outcomes and variations. The use of functions to control behaviour and distribution of biomes around a procedural generation around the world, as well as multi-layer approaches and extra integration techniques - achieved even by changing the parameters of the *Normal Access* - should also be tested[5].

2. Time Complexity Analysis

As discussed in the previous sections, when increasing from one to two dimensions, the number of operations performed on each node remain unaffected. Plotting the 2D *Everling Noise* on square matrices of increasing side size, will result in a quadratic curve. This however is not due to an increasing amount of computational power required, but because the amount of cells in a matrix of size *n x n* is then in a matrix *n x 1*, .

Graph 3, obtained by running a Python version[6] of the *Everling Noise* in a MacBook Air with processor M2 for square matrices of side length from 10 to 200, measuring the run time and plotting it in terms of matrix side size and number of cells.

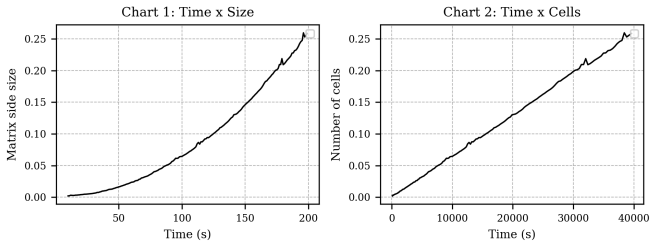

Image 16: exponential correlation between time and size and linear correlation between size and amount of cells

D. Higher dimensions analysis

The previous analysis indicates that both one dimensional and two dimensional implementations of the presented algorithm present a linear correlation between time consumption and the amount of matrix cells generated. Since there is no reason to believe that such a tendency would not be kept when running to higher dimensions, it is safe to state that for any *n*-dimensional square matrix of constant side size *c*, one of the components of its time complexity of our noise function is O(*1*). The behaviour of the coefficient of linearity as the dimensions of the noise increase, however, is still uncertain.

When running, the algorithm will generate a random value and calculate the neighbors average for every cell in the noise map. This means that, no matter how many dimensions are being considered, the number of major tasks will remain the same. But, despite the amount of times the average function is called being constant, its performance increases. This is due to the increase in adjacent cells computed in the mean calculation. In one dimension, there will be only two values, in two there are four, in three there are six and so on, as represented on Table I. Based on that data, the theoretical time complexity of the Everling Noise scales linearly as the number of dimensions increases.

Number of Dimensions	Number of Neighbor Cells
1	*2*
2	*4*
3	*6*
4	*8*

Table I: Number of Neighbor cells for each matrix cell in Everling Noise according to the number of dimensions

Aiming to discover if this prediction is reflected in real implementations, the Everling noise has been run in 1D, 2D and 3D for every perfect square up to 1000 (Figure 4). The nature of these charts allows ascensions to be made regarding the time complexity of the algorithm in higher dimensions as well as about its dimensional scalability. About the first, all the graphs present a strong linear correlation, showing that inside a given dimension, the performance will always be proportional to the amount of numbers generated. Similarly, the linearity of the fourth chart - comparing the average time per number generated in the three different algorithms confirms the prediction that the time complexity of the Everling Noise in regards to dimensions is O(2n).

[5] Examples in https://preview--everling-noise-scapes.lovable.app/
[6] Code available in: https://github.com/CasEverling/EverlingNoise

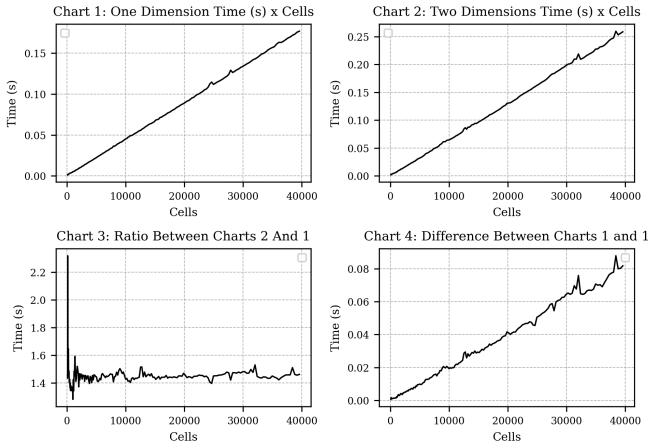

Image 17: exponential correlation between time and size and linear correlation between size and amount of cells

This is unusual for noise maps developed in the traditional way, that scale exponentially. Even the simplex noise, developed focussed on performing well in higher spaces grows quadratically [4]. The Everling Noise has, then, a high performance in high dimensional spaces. This relative gain in performance makes it specially suited for dealing with 2D texture animations - such as weather, fire and smoke -, 3D generation - caves, clouds and fogs - and 3D animations - for example, procedural smoke animation.

E. Perlin Noise Comparison

The spark that led to the development of the Everling Noise algorithm was looking for a way to achieve the results of the Perlin Noise based on a different conceptual understanding of the process to achieve them. Therefore, in order to understand whether it serves its purpose, a direct comparison is required. Moreover, since the Perlin Noise is the industry standard, most noise map algorithms have been widely compared to it in all of its features. Therefore, by analysing the comparison between the Everling Noise and the Perlin Noise, further assessments can be made about how it compares to other procedural generation algorithms allowing for a more precise understanding of how it fits developers' needs.

In this section a basic Python implementation of the Perlin Noise, based on the description of the algorithm given on its original presentation [1], written by Chat GPT, will be compared to a Python implementation of the Everling Noise. Both were implemented using only the core features of Python and Numpy v1.2.3. It is important to disclose, though, that these tests may not reflect either of these algorithms on its best performing versions, therefore time efficiency comparisons should be taken with caution and focussed on how these two may fit better in different coding situations.

1. Time complexity

As argued on sections III.B, III.C and III.D, the *Everling Noise*'s time complexity is estimated to be linearly related to the amount of cells generated, independent of its number of dimensions. It is, the time complexity increases linearly in regards to the number of dimensions considered. This contrasts with the exponential nature of the *Perlin Noise*.

But in one and two dimensions, the two should have the same time complexity, since $2^1 = 2 * 1$ and $2^2 = 2 * 2$, making a direct comparison relevant. Image _ was achieved by running the optimized one dimension and the two dimension versions of the *Everling Noise* approach and the one and two dimensional implementations of the *Perlin Noise* for a number of cells equal to all perfect squares from 10 to 200 and plotting them according to their dimensions.

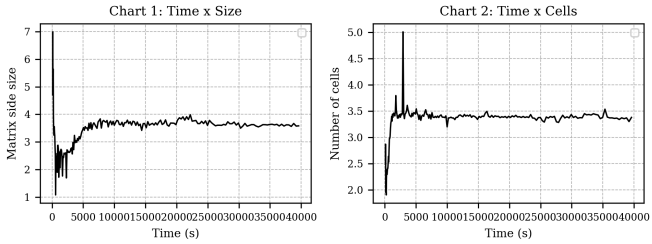

Image 18: Ratio between runtime time between Perlin in Everling Noise.

This direct comparison shows that the *Everling Noise* has, even in low dimensions, the upper hand against the theoretical implementation of the *Perlin Noise*. Although implementation dependent, this result shows, in a pessimistic analysis, a similar performance.

2. Noise Maps

Independent of the impact of performance, it is relevant to consider that both have meaningfully different outcomes. Independent of which is the integration algorithm of choice, Everling Noise has a way more clustered, fitting use cases where more spatially concentrated features are desired, like picking spawn points for asteroids in outer space, population per area, etc; but fails in producing continuous lines.

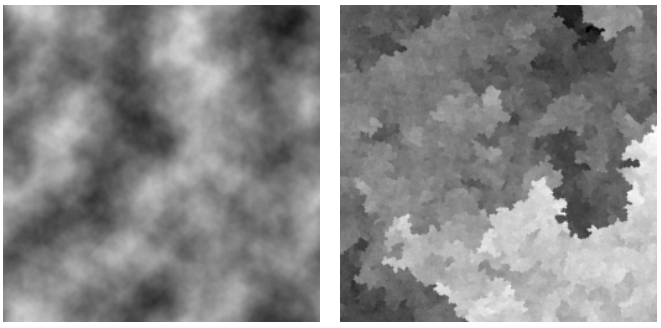

Image 19: Gray scale *Perlin Noise* in the left and gray scale *Everling Noise* in the right.

Another relevant contrast is the palette of patterns created by the Everling Noise. Restraining the possibilities to the ones mentioned in the present text, there are at least hundreds of permutations of the four traversing methods, each one resulting in a different style of image - shown on section III.C.2 -. Although the Perlin Noise does have the octaves parameter, which controls the number of layers used on the map, it does not alter the nature of the final image other than by increasing its resolution.

Everling Noise also produces patterns more suited for natural terrain generation. Different from Perlin Noise, the Everling Noise, from its core concept is focussed on terrain generation. Having a background statistical distribution applied specifically for terrain generation results in more naturally distributed geological structures, allowing for the creation of less repetitive and more unique scenarios.

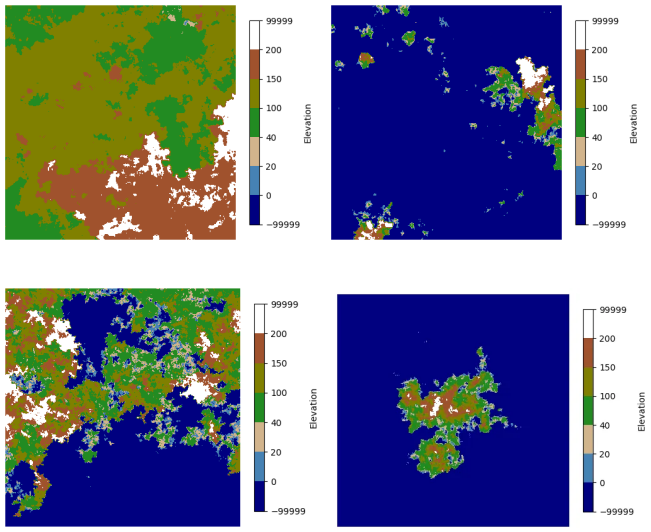

Image 20: Colored *Everling* noise map for map generation

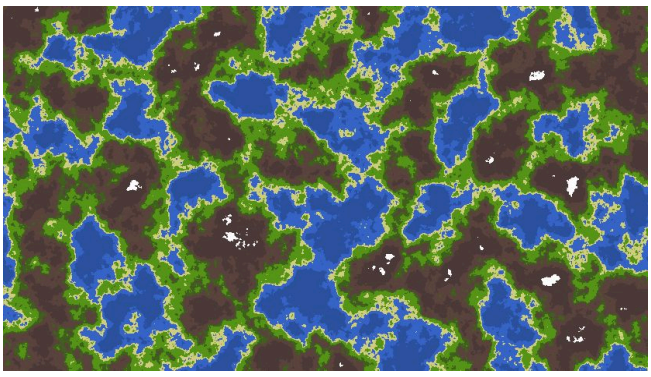

Image 21: Map generated based on perlin noise

This key difference is visible when comparing images _ - representing maps created by the *Everling Noise* - and _, a typical *Perlin Noise* based map. But this is not an absolute strength nor weakness, different algorithms are more efficient in solving certain kinds of problems. While the Perlin Noise struggles more in the conception of large differing biomes, some kinds of textures, such as water, better fits with it.

Yet, it is evident that for most procedural terrain generation use-cases, especially when it comes to indie games, the Everling Noise has meaningful advantages. Mainly due to ease of customization allowing the same code to be reused for vastly different types of land and geographic characteristics, and effortless translation from noise to actual terrain. Furthermore, since Perlin and Simplex noises are broadly applied for terrain generation, the patterns of the Everling Noise would probably stand out as a more unique alternative.

IV. CONCLUSION

Noise maps are the most well established tool for procedural world generation, being used on various titles such as Minecraft, Terraria and Factorio. However, noise algorithms tend to work by defining high and lows points in a grid and using noise functions to smooth the surface gradient. By switching the process from focussing on the peaks and valleys, to the ground fluctuation itself, a new kind of noise-map algorithm is proposed.

Populating a grid with gaussian generated values for altitude variance and defining a set height point, the computer can walk through the matrix converting it into a height map, one of the core versions of a noise map. Such an approach, despite being uncommon, is backed by the statistical principle of Central Limit Theorem [7], which states that despite the distribution of the original population - changes in altitude per square meter - its means are normally distributed.

The implementation of this algorithm has a time complexity of $O(n^d)$ for generating a d-dimensional matrix with sides of length n. This approach is more efficient than the Perlin Noise when generating fully rendered noise maps. In other words, for creating a 2D map, for, for example, a 2D top-to-down RPG, the presented algorithm scales linearly rather than exponentially.

This, however, is not achieved without some caveats, the increased performance on fully rendered images is counterpart by the incapability of performing single point operations. In such a scenario, Perlin Noise could perform all operations in constant time, while our code would keep its original time and space consumption.

Moreover, due to the linearity of the conversion from height variation into a height map, converting the code so that it runs on the GPU instead of the CPU may be quite challenging. Still, this opens up space for different techniques for walking though the variation matrix which may be way

more efficient, both on the GPU as well as in the CPU. Possibilities such as weighting the altitude change in terms of the surrounding changes or, starting by multiple points at once or performing a breath first search (BFS) in parallel have to be further analysed before coming to a final conclusion.

Despite performance, however, it is clear that the developed algorithm manages to generate patterns that differ from those of the most widely used noises. Two scenarios can arrive from this. First, the new noise textures will be able to perform the same kind of generation as the current, depending only on a different post processing. Second, their difference is so huge in nature that this kind of noise map will be relegated to a niche, maintaining the standard approaches for most of the tasks, but working as an extra tool for developers facing specific challenges.

Which of the possibilities will manifest itself in reality can only be known as time passes and programmers explore the viability of both use cases. Nevertheless, the variety of results obtained from a single normally distributed function depending on the processing method chosen, is certainly going to play a huge role in the discussion, as well as the use of assisting functions when setting the parameters for the gaussian-number generator.

If you are a fellow game developer aiming to give a shot to a new technology, and a more stylised world generation, please feel free to use and explore the concepts of this paper in your code. A full implementation of 1-dimensional and 2-dimensional version of the code in C# and Python can be accessed on: https://github.com/CasEverling/EverlingNoise.

V. REFERENCES

[1] R. van der Linden, R. Lopes and R. Bidarra, "Procedural Generation of Dungeons," in IEEE Transactions on Computational Intelligence and AI in Games, vol. 6, no. 1, pp. 78-89, March 2014, doi: 10.1109/TCIAIG.2013.2290371.

[2] K. Perlin, “An image synthesizer,” *ACM SIGGRAPH Computer Graphics*, vol. 19, no. 3, pp. 287–296, Jul. 1985. DOI: 10.1145/325165.325247

[3] K. Perlin, “Improving noise,” *ACM Transactions on Graphics (TOG)*, vol. 21, no. 3, pp. 681–682, Jul. 2002. DOI: 10.1145/566654.566636

[4] K. Perlin, “Noise hardware,” in *Proceedings of the ACM SIGGRAPH 2001 Conference on Computer Graphics*, ACM, 2001, pp. 377–386. DOI: 10.1145/383259.383327

[5] S. Worley, “A cellular texture basis function,” in *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH ’96)*, ACM, 1996, pp. 291–294. DOI: 10.1145/237170.237267

[6] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed. Pearson, 2018.

[7] S. M. Ross, *Introduction to Probability and Statistics for Engineers and Scientists*, 6th ed., Academic Press, 2020.