Dynamic Procedural Generation of Terrain with Hydrologic Systems

Advanced Computer Graphics, Project Report

Jay Franklin
Rensselaer Polytechnic Institute

Abstract

Keywords: Template, formats, instructions, length, conference publications

Introduction

Creating the vast areas of terrain needed for games and other simulations is a time-intensive process when performed manually by an artist. With procedural content generation (PCG), terrain is instead algorithmically generated without input from a user beyond specification of initial parameters. Many popular video games, including the highly popular game Minecraft, use PCG to create an infinite world for the player to explore—using dynamic world generation methods, new terrain is continuously generated around the player and the world appears to never end. In contrast, world-building methods use PCG to pre-generate a world of fixed size, typically iterating over this world multiple times to simulate real-world environmental processes [1]. Although world-building methods require an initial period of time to generate the world, they are able to be used with a greater variety of algorithms than dynamic PCG methods since the entire world state can be accessed during the generation process.

Therefore, dynamic world generation methods are somewhat limited in the realism of terrain they can produce in comparison to world-building methods [2]. In particular, elements of hydrologic systems such as rivers—an extremely common feature of terrain and terrain generation algorithms—are difficult to generate and simulate realistically with conventional dynamic approaches. For example, Figure 1 depicts a river which abides by two simple requirements: firstly, the river must flow from a point of high elevation to a point of low elevation, and secondly, a river which intersects with another river should join it, rather than cross it. Simple rivers matching these requirements are easily possible to achieve via world-building methods, as the path of a river can be traced iteratively from its source. However, this approach cannot work without modification if the entire world has not yet been generated. If the location at point B is generated before point A, then point B will not contain the river until point A is generated. Rivers generated via dynamic terrain generation are therefore frequently unrealistic, such as rivers in Minecraft worlds which do not flow from high to low points, and instead remain at sea level [3].

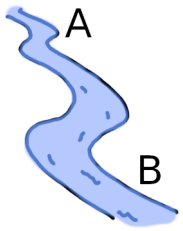

Figure 1. A river, flowing from high elevation at point A to low elevation at point B.

In this project, we introduce a dynamic terrain generation method that overcomes this issue via incorporating elements of both world-building and dynamic PCG in the terrain generation process. This hybrid method creates terrain with both geologic elements (such as mountains and valleys) and a system of interconnected hydrologic elements (rivers, lakes, and seas) to generate world of unlimited size at runtime with rivers that match the basic properties outlined above. We demonstrate our results via generation of 2D maps, where the color at a given (x,y) point represents either elevation or the presence of water, and analyze the results.

Related Work

Procedural terrain generation is a highly active area of research, and there has been a huge investment into researching the best approaches to use for generating a game world, including approaches to generate systems of rivers. According to a survey of the field by Smelik et al. [4], river generation approaches typically operate over an entire pre-existing heightmap, and are therefore categorized as world-building approaches (as opposed to dynamic world generation). In general, most river network generation approaches can be divided into two categories: those using teleological methods, and those using ontogenetic methods [5]. Ontogenetic approaches eschew simulating in-between steps and instead try to algorithmically approximate the end result of the terrain process directly: a good example of this is a height map generated with Perlin noise [6]——Perlin noise has no relation to the real-world processes that determine elevation, but it can stand in for it well enough to be used to procedurally generate terrain. Ontogenetic approaches to river network generation include an early method by Kelley et al [4], which uses fractal subdivision of linear components, or the more advanced fractal method by Prusinkiewicz et al. [7] which creates rivers and mountains simultaneously. The latter technique could theoretically be adapted for dynamic generation of an infinite world, but the approach has several issues which cause its rivers to lack realism, and the highly mathematical nature of the approach results in low explainability and a prohibitively high difficulty in its modification. However, we do use a similar fractal approach to algorithmic generation of geological features, as fractal methods can be used to dynamically expand from a lower level of detail to a higher level.

Other approaches to procedural generation are teleological. Teleological approaches generate a world by simulating the actual physical processes that create terrain in real life (e.g. erosion, continental drift, and so on) to ensure that the terrain appears realistic when compared to real-world terrain. An example of this is the seminal 2013 SIGGRAPH paper by Genevaux et al. [8] which simulates the locations of watersheds and trajectories of rivers according to a hierarchical drainage network, created by expanding a graph of river paths inward from coastal outlets. Unlike many other teleological river-terrain generation methods, such as the similar methods described by Peytavie et al. [9], the Genevaux methods include elements of dynamic terrain generation. For this reason, along with the simplicity of the algorithm and the highly realistic results of the approach, we have chosen to adapt parts of the Genevaux methods to be used with our infinite-world terrain generation framework.

Methods

Features

The basic principle behind our approach is to dynamically generate a connected graph of features—metadata that describes some geological or hydrological aspect of the terrain—and to combine these features with world-building techniques to ensure terrain remains consistent and realistic. We chose this feature-based method as it naturally lends itself to a high level of explainability, can be extended with the addition of more types of features, and is intuitive to understand. If a mountain is not some emergent aspect of a random noise function, but instead a feature known within the world ahead of time, then one does not need to waste computation on finding the nearest local optimum—the location of nearby mountains are already known. Similarly, this feature system lends itself well to customizability and adaptability—the manifestation of a feature can be adjusted to use whatever algorithm makes the most sense for the use case, and can even include multiple sub-features nested within it (such as a mountain range feature containing multiple mountains).

Feature can have the following properties:

- **Origin**: The (base) position of the feature, in (x,y) coordinates.
- **Influence**: A function accepting an (x,y) value as input and returning a value representing the level of influence the feature should have over the given coordinates. A value of one is a normal level of influence, a value of one-half indicates the feature's influence is reduced by half, and a value of zero indicates the feature does not affect the coordinates in any way.
- **Elevation**: A function accepting an (x,y) value as input and returning a value representing the elevation that the feature adds or subtracts for the given coordinates. This function is used for **geological**

features—features that describe the basic lay of the land.
- **Moisture**: A function accepting an (*x*,*y*) value as input and returning a value representing how much surface water is present at the given coordinates. At present, any nonzero value is rendered the same way (as indicating water is present), but this value also be used to indicate different levels of water-flow.
- **Moisture**: A function accepting an (*x*,*y*) value as input and returning a value representing how much surface water is present at the given coordinates. At present, any nonzero value is rendered the same way (as indicating water is present), but this value also be used to indicate different levels of water-flow.
- **Sub-features**: A list of additional, nested features that are contained within the parent feature.

Using these properties, we can create a variety of features which lend themselves well to creating a realistic-looking world. In particular, we can use the influence mechanic to ensure features never extend too far, beyond the space where features have been computed; most features have an internal *radius* parameter where a feature is at 100% influence when the distance from their origin is at 75% of their radius, to 0% influence at 100% of the radius or more.

The list of features we have included is as follows:

- **Continent**: A continent describes the top-level landmass that defines the basic shape of an area of land, including its origin (a position roughly corresponding to the center of the continent), and baseline dimensions in *j*-*k*-*l* space (hexagonal coordinates, as described in *World Structure* below). It contains many additional features as sub-features, and uses these to compute elevation and moisture for a given coordinate.
- **Mountain**: A geological feature indicating that an area of land is elevated. Mountains are at present defined according to an input size, which determines its dimensions in dimensions in *j*-*k*-*l* space and its maximum elevation. The dimensions and maximum elevation are used to create the mountain's elevation function, which resembles a cosine curve with its maximum value at the mountain's origin (its summit) and reaching zero at the extent of its dimensions (its base). Mountains that are placed in water will generally create islands.
- **Mountain range**: A mountain range describes a series of mountains which are placed, one after the other in a polyline. Mountains are smaller at the ends of the line, and larger at the center.
- **Valley**: The inverse of a mountain, a valley has a negative elevation and therefore reduces land in a given area. Valleys that reduce an area below sea level will generally create gulfs, peninsulas, or even inland seas.
- **River**: Rivers start at their origin—a river source, frequently placed on the snow line of a mountaintop—and descend to an area of lower elevation. The river returns a positive moisture value for areas with surface water, and a non-positive moisture value for areas without surface water.
- **Lake**: Lakes are places whenever a river runs into a local minimum—an area where it can no longer descend downward. Like a river, a lake will also return a positive moisture value for the areas it encompasses which are directly covered by water. Lakes will generally have one or more rivers branching out from them.

World Structure

The world itself is subdivided into different zones. with a maximum of one continent having its origin in each zone. The zones are a tessellation of hexagons, as rectangular zones might create too much regularity, and if a continent is to be placed in a zone the location of that continent is jittered (Figure 2).

A continent can extend to multiple zones, up to a maximum distance of the radius of one zone. Due to this, there may be multiple continents overlapping a specific coordinate, and a coordinate may be affected only by continents outside of its containing zone. For this reason, we also define a zone neighborhood: for a given coordinate, its neighborhood of zones is 1) its containing zone, and 2) the six zones bordering that containing zone. This is illustrated in Figure 2: the hexagonal area labeled "A" and the bordering zones labeled "B" make up the zone neighborhood of any location within "A."

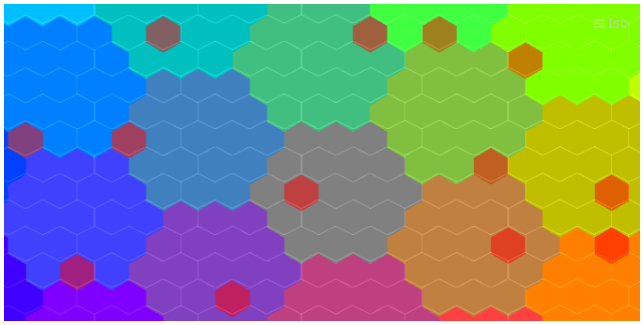

Figure 2. The zones used to divide the world. Each red hexagon represents a jittered location where a continent could spawn; each group of hexagons in one color represents one zone.

To allow maximum flexibility within the world structure, we used a (*i*,*j*,*k*) coordinate system in addition to the basic (*x*,*y*) system. The (*i*,*j*,*k*) coordinate system is structured as described by Amit Patel [10], such that each coordinate corresponds to a pair of sides of a given hexagon, and moving in one dimension (such as the *i* dimension) means that dimension stays static, one of the other dimensions is increased and the other is decreased (so *j* could increase and *k* could decrease, as shown in figure 3). The advantages of using this coordinate system are that it gives greater variation to functions operating over a 2-D space: the elevation of an (*x*,*y*) coordinate within the influence of a mountain is computed by first converting these coordinates to the (*i*,*j*,*k*) system, and then applying a separate elevation function along each of the (*i*,*j*,*k*) axes.

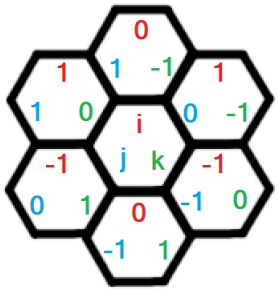

Figure 3. The (*i*,*j*,*k*) coordinate system for hexagons. The central hexagon is at (0,0,0). Movement through one of the three axes results in two values changing, such that all coordinates for a given hexagon add up to zero.

Additionally, the world itself stores one additional value beyond its zones' continents (and their nested features): the world seed. The world seed is combined with the (*i*,*j*,*k*) coordinates of each zone to create a new seed for said zone, and this seed is used to determine whether a continent exists in a zone, what its location is, and what additional features and parameters it has (and the additional features and parameters of those nested features, and so on).

Dynamic World Generation

When elevation or moisture for a specific (*x*,*y*) coordinate needs to be generated, the dynamic world generation algorithm is run to ensure all required features are generated, and then these features are queried for their elevation, moisture, and influence values. The algorithm is as follows

1. **Continent retrieval**: In the first step, the appropriate continent(s) are retrieved, corresponding to all of the continent(s) in the zone neighborhood enclosing (*x*,*y*). If a zone has not yet been assigned a continent at this point, the continent is generated now, and cached within the world—although like all features, the subfeatures of this continent are not populated with features until they are needed to be retrieved. Until then, only the continents which (*x*,*y*) is within the influence of are retrieved and (if they haven't already) populated during the following step.
2. **Elevation**: In the next step, elevation for the (*x*,*y*) coordinate is determined according to the sum of the elevation at (*x*,*y*) of each continent retrieved during step **1**. If they haven't already, the geological features of this continent are created according to the seed of their enclosing continent, assigned any additional subfeatures, and used to determine the elevation. There is a random mix of mountains and valleys of varying sizes and shapes created, and possibly a few mountain ranges. Most of these features

are clustered within the coastline of the continent itself, but some mountains or mountain ranges may be outside of it (islands and island chains).
After all of the features have contributed their elevation, the sum elevation value is multiplied by the influence function of (**x**,**y**) for the enclosing continent, ensuring there is no hard edge between areas of influence. Finally, three layers of Perlin Noise are applied to the final value: the first has a small amplitude and higher variation, the second has medium amplitude and medium variation, and the third has a high amplitude and a high variation. Although the noise serves to further vary the continent, it is not the primary source of variation and only serves to enhance the existing variation and ensure hard lines do not remain — the noise layer is not large compared to the features to outweigh them.

3. **Moisture**: In the final step, the moisture map is computed by adding the two hydrological features: rivers and lakes. River sources are added to mountains which reach a predefined snow level, as well as sprinkled around the rest of the area. When a river is added, a recursive function is started which finds areas adjacent to the river that are lower than it in elevation, chooses one at random, and recursively continues finding locations from that point on. If it cannot find a lower area, it adds a lake, and begins searching the area surrounding the lake for outlets to the river. If it encounters other sources of water—such as another river, or the sea—the process stops. In this way, an entire drainage network is iteratively added over the whole continent.

Results

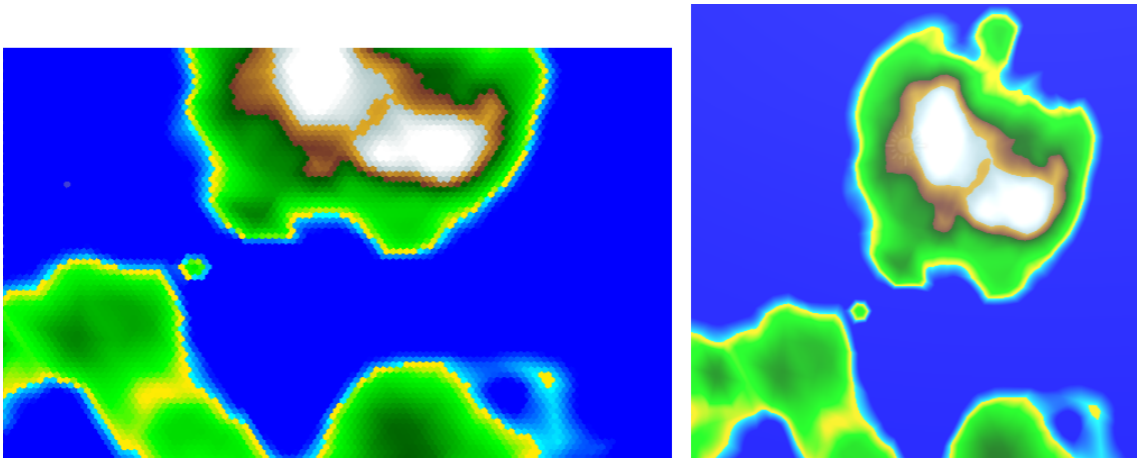

Figure 4. *Left*: the hexagonal render (note it is somewhat stretched in the horizontal dimension). *Right*: the full render.

We tested the algorithm using two renders: the hexagonal render, which is rendered quickly but only fills in each hexagonal tile on the map with the elevation or moisture value at the center of that hexagon, and the full render, which takes longer to produce but colors each pixel with a unique value. We tested with a 100-by-80-hexagon image, rendering in about five or ten seconds depending on the complexity of the terrain, and a 1000-by-1000-pixel full render (of varying scales), which took about two or three minutes (again depending on terrain complexity). See Figure 4 for a side-by-side comparison.

Although we did not have enough time to create a height-map, we used the 2D maps to represent both elevation and water for a given point. If a point is light blue to dark blue, it is either below sea level or covered by a river or lake. If a point is yellow, green, dark green, brown, gray, or white, in that order, the point is representing an area of lower or higher elevation, respectively. Additionally, gray or white points represent areas which are above the snow line, and therefore are likely to be originating rivers. Brown points are of high enough elevation that they typically indicate a mountain feature.

We evaluate the efficacy of the method according to the following requirements, which were initially outlined in the project proposal:

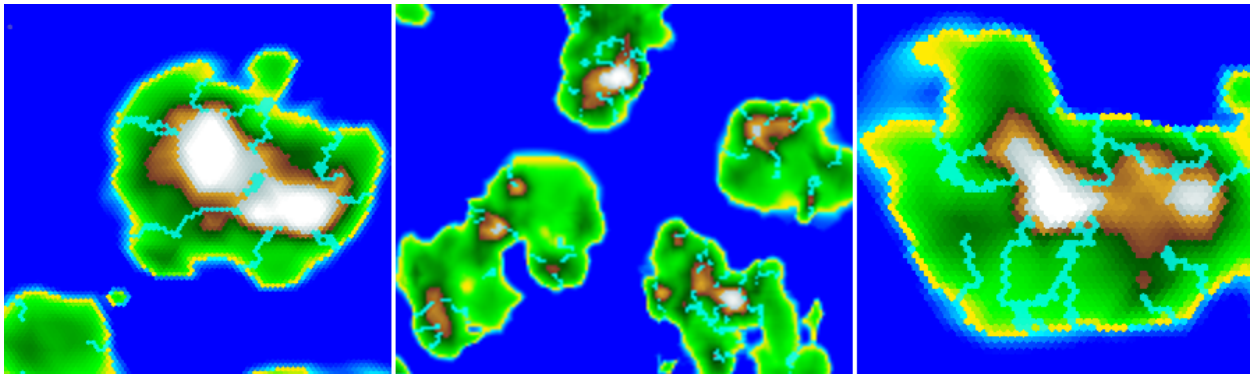

Figure 5. *Left*: A hexagonal render with rivers. *Center*: A full render with rivers. *Right*: Another hexagonal render with rivers, this time showing how multiple rivers can merge into one river or a lake.

1. Terrain is generated within the bounds of a continent, which has realistic-looking coastlines. Specifically, the coastlines are jagged/natural-seeming, and has areas with gulfs, peninsulas, etcetera.

Figure 7 depicts some of the varying coastlines generated using our methods. As the methods overlap the "basic" coastline of the initial continent feature with additional mountain and valley features, we can see terrain resembling gulfs, peninsulas, and so on. Additionally, the coastlines are jagged in small areas, and appear smoother in others, resembling real-world geography.

2. Within the continent, the elevation is effected by geological features such as mountains and valleys.

This is again easily visible in Figure 7, as the placement of mountains and valleys creates interesting topology. Note that lighter green depicts lower elevation, and darker green depicts higher elevation, with brown and gray/white at the highest elevation. There are a few mountain ranges visible in these images as well—areas of multiple mountains in a row—but these will require some more tweaking to be truly noticeable.

3. Rivers flow realistically from higher elevation (e.g. mountaintop) to lower elevation (e.g. coastline).

We consider this to be mostly a success. As seen in Figure 5 and Figure 8, rivers do flow from areas of high elevation to areas of low elevation, and meander in a somewhat realistic way. However, we believe that with a more complex river-generation algorithm applied over the existing terrain model we have developed, we can continue to improve the hydrology to make more realistic-seeming river paths.

4. Rivers interact realistically with one another––two rivers will not cross one another, but merge into one river.

This is again seen in Figure 5, where rivers do not zig-zag across one another but instead merge into one river. However, we think that this could be made more realistic by taking into account variables such as river flow and channel capacity, such that two rivers will merge into a larger river.

Discussion

Challenges

We faced a number of challenges during the development process. Although the algorithm was mostly sound, implementation was difficult, as the multiple interacting components needed to be implemented

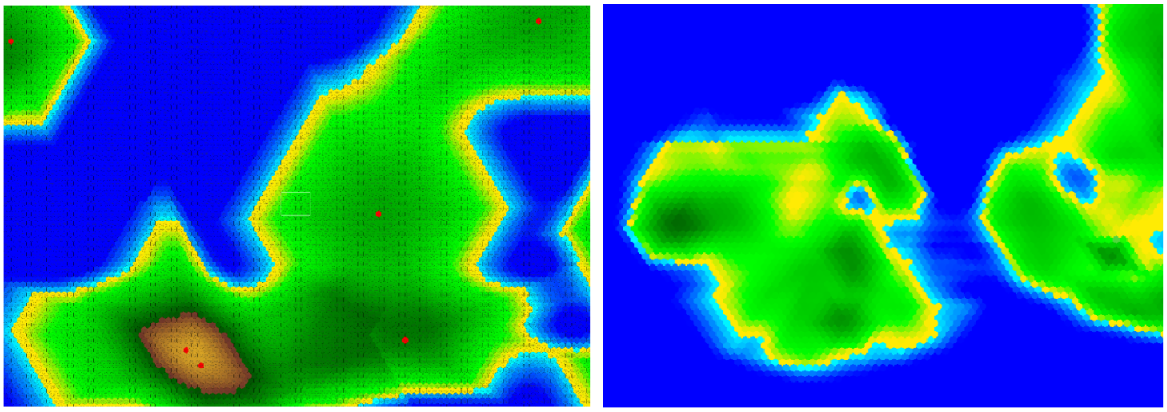

Figure 6. *Left*: A blooper image from early in the development process. *Right*: A blooper image from more recent in development (hexagonal render).

exactly right in order to not produce artifacts. If a miscalculation occurred, a hard edge would be seen between different features, as seen in Figure 6. The reason for these varied, but typically what would occur is either 1) the correct feature would not be retrieved for a given coordinate, or 2) a feature would not gradually diminish in influence, but instead keep going until it reached the boundary of the area where it would be retrieved. In total, development took about fifty or sixty hours over more than a month, including a great deal of time simply spent sketching out and revising algorithms as well as their implementation, revision, and debugging.

Additionally, deciding on the best approach for rivers was difficult. We intially were planning to implement a more complex river generation algorithm, based on [8], but due to time constraints we were limited to implementation of the simpler algorithm only. We hope to revise the project in the future and continue to add additionally complexity over the existing framework.

Conclusion

Overall, we consider the project to be a success. It could still be improved in quite a few ways, but at the same time, the finished product meets all of our initial requirements set out in the project proposal. Moreover, the framework we have created is incredibly extensible, as more features can be added with ease, as well as explainable—as all of the features are stored with the terrain as metadata, tweaking the environment and adjusting parameters to produce different terrain is a far simpler process. Parameters include values such as the number of features per continent, the size of continents, probabilities of each feature, sizes of each feature, the size of the dividing zones, and so on—all easily connected to a core component of the algorithmic process, such that revising terrain to create the kind of terrain desired by a user for a program is a far simpler process than tweaking black-box variables. As a whole, we consider creating this project to be a valuable research experience, and will continue to add on to the project in the future.

References

1. Doull A. World Building — Procedural Content Generation Wiki; 2010. [Online; accessed 3-May-2021]. Available from: `http://pcg.wikidot.com/pcg-algorithm:world-building`.
2. Doull A. The death of the level designer; 2008. Available from: `http://pcg.wikidot.com/the-death-of-the-level-designer`.
3. Wiki contributors. River — Official Minecraft Wiki; 2021. [Online; accessed 3-May-2021]. Available from: `https://minecraft.fandom.com/wiki/River`.
4. Kelley AD, Malin MC, Nielson GM. Terrain simulation using a model of stream erosion. In: Proceedings of the 15th annual conference on Computer graphics and interactive techniques; 1988. p. 263–268.
5. Doull A. Teleological vs. Ontogenetic — Procedural Content Generation Wiki; 2010. [On-

line; accessed 3-May-2021]. Available from: http://pcg.wikidot.com/pcg-algorithm:teleological-vs-ontogenetic.
6. Marinescu A. OPTIMIZATIONS IN PERLIN NOISE-GENERATED PROCEDURAL TERRAIN. Studia Universitatis Babes-Bolyai, Informatica. 2012;57(2).
7. Prusinkiewicz P, Hammel M. A fractal model of mountains and rivers. In: Graphics Interface. vol. 93. Canadian Information Processing Society; 1993. p. 174–180.
8. Génevaux JD, Galin E, Guérin E, Peytavie A, Benes B. Terrain generation using procedural models based on hydrology. ACM Transactions on Graphics (TOG). 2013;32(4):1–13.
9. Peytavie A, Dupont T, Guérin E, Cortial Y, Benes B, Gain J, et al. Procedural Riverscapes. In: Computer Graphics Forum. vol. 38. Wiley Online Library; 2019. p. 35–46.
10. Patel A. Hexagonal Grids; 2013. [Online; accessed 3-May-2021]. Available from: https://www.redblobgames.com/grids/hexagons.

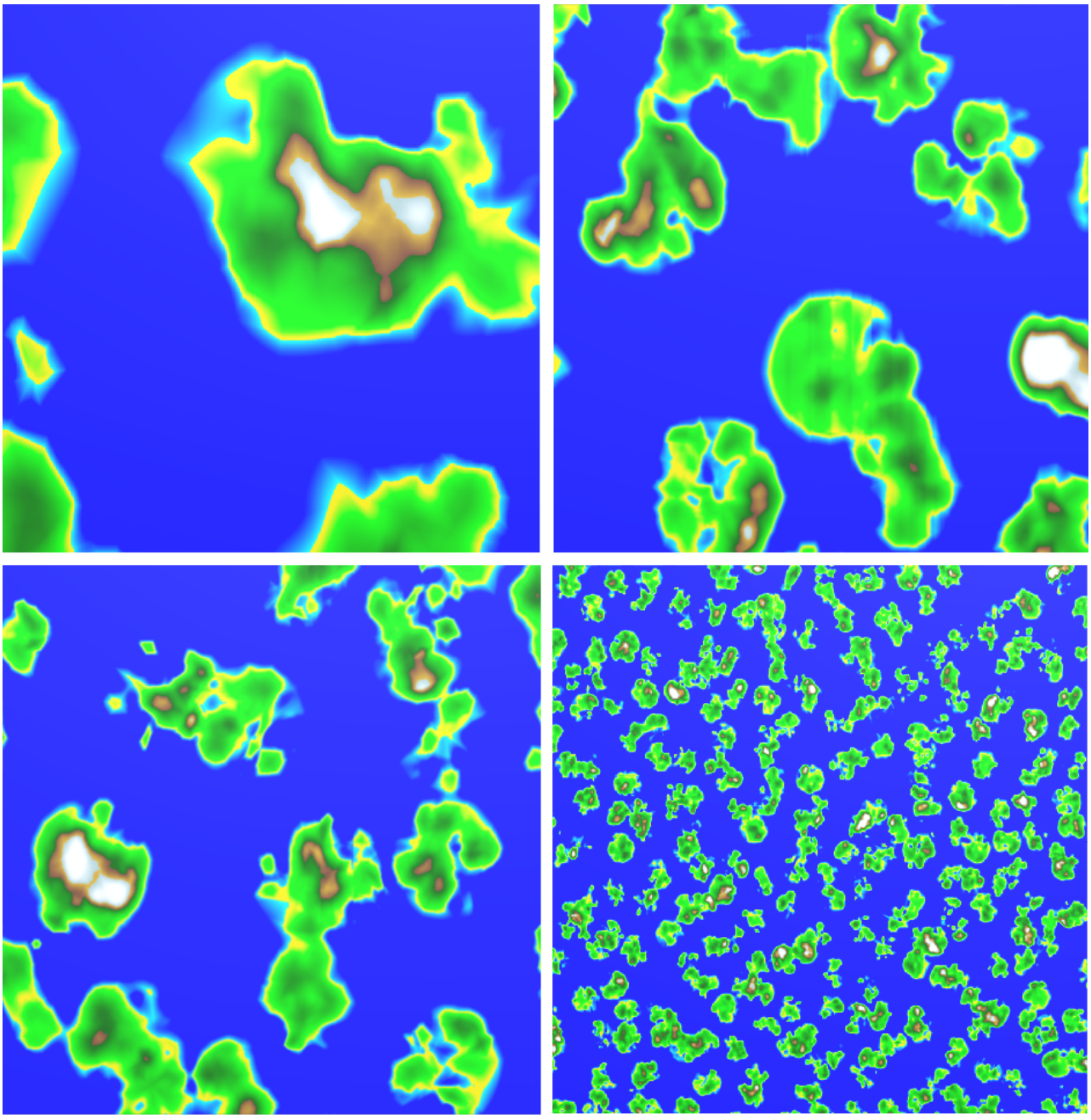

Figure 7. Four full renders of different worlds and at different scales, showing the various mountain and coastline patterns which appear with some of the basic parameters. *Left to right, top to bottom*: a ten-samples-per-hexagon render, a five-sample scale render, a four-sample render, and a one-sample render.

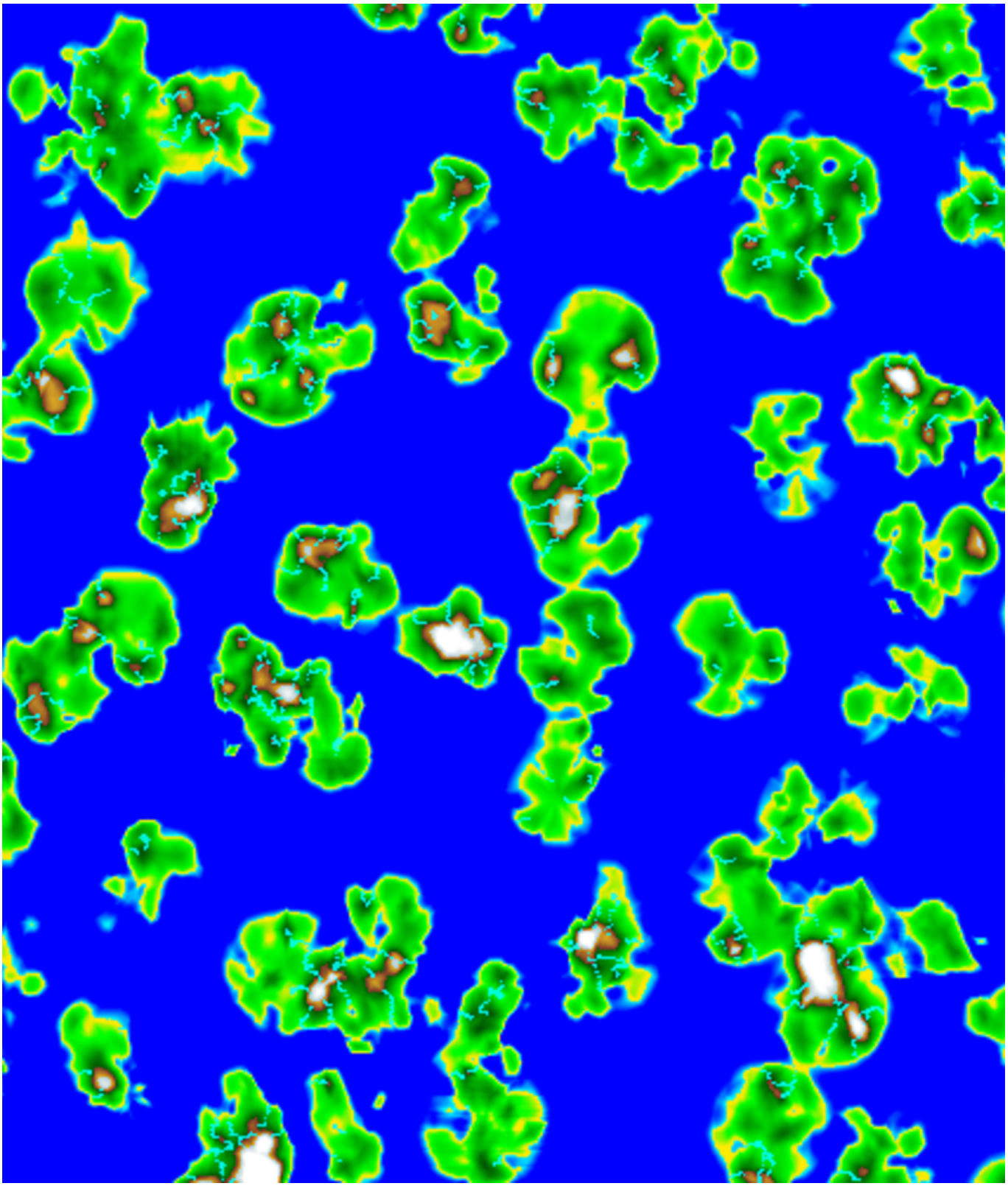

Figure 8. A full render with rivers at a scale of one sample per hexagon per pixel.