Proceedings of CONF-MLA 2025 Symposium: Intelligent Systems and Automation: AI Models, IoT, and Robotic Algorithms
DOI: 10.54254/2755-2721/2025.LD28166

Procedural Noise in Open-World Survival Game Design: Applications and Insights

Chenhe Zhang

The International School of Kuala Lumpur, Kuala Lumpur, Malaysia
czhang28@iskl.edu.my

Abstract. Due to the popularity of open-world games and competitive survival games, map design has become increasingly important, with more realistic graphics and detailed terrain becoming a key factor in attracting and retaining players. This is why a lot of games currently running are using a concept called procedural terrain generation to create refreshing, unpredictable maps for virtually infinite replayability. Rather than relying solely on handcrafted worlds, procedural systems can provide variety, ensure that no two playthroughs feel exactly the same, and extend the lifespan of the game by reducing repetitiveness. This paper will delve into the situations where procedural generation is most beneficial, while also pointing out the contexts where handcrafted design remains superior. In particular, it will discuss how procedural terrain gen can be integrated with game progression to provide a balanced pace. To illustrate these points, this paper will dissect Minecraft's terrain generation system as a representative example.

Keywords: Procedural, Perlin, Terrain, Game Design, Generation

1. Introduction

Terrain Gen, or terrain generation [1-3], has been a staple in games for some time, especially in open-world games. They play a huge role in making each individual saving feel unique, unlike meticulously manually designed level maps. Terrain generation (Which this paper will refer to as terrain gen for the rest of this paper) is tricky to pick up because of the vast number of seeds, allowing for seemingly non-repeating save states, except for every play.

But when diving into the exact composition of terrain gen, people will notice that a lot of games with terrain gen tend to have almost wave-like formations (Minecraft, for example, uses this kind of generation), where if someone experiences a Minecraft world themselves, it feels like a natural earth-like environment filled with mountains and valleys, and caves. Still, if they zoom out or put the seed on a seed map, they'll quickly notice that the formation of biomes and the shape they create is much more even, like patches of sitchings on a worn-down pair of pants, they scatter with a fractal-like pattern, painted with bright colors.

© 2026 The Authors. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (https://creativecommons.org/licenses/by/4.0/).

DOI: 10.54254/2755-2721/2025.LD28166

2. Concept introduction

This is the root of most terrain gen: Perlin Noise [4-6]. Usually, when people think of terrain gen, they think of random values, seeds, and an algorithm that turns a seed into a blueprint for how to create the terrain (If they have the same seed, they can effectively recreate the same world multiple times). The exact process that this seed leads to a blueprint section is a bit more complex, and this research will get into it later, but now the simple way to explain 2D perlin noise(The kind of perlin noise that traditional terrain generators most commonly use) is that it's a bunch of gradient vectors generated on a 2d grid, the animated effect people usually see is most likely done by change the previously mentioned gradient vector at a constant rate, although not often used for terrain gen, this will produce the smooth almost water flow like motion people typically see with procedural liquid/fog material.

There are examples where games would need and prefer a generated world rather than a manually designed world, and there are times when vice versa is better fitted. The popular factory building tycoon game called "Satisfactory" uses a giant manually built map for its gameplay, which they have their reasons: Satisfactory is a really complex user free form creation game, where spline curves are used for rails and modular buildings can be placed virtually anywhere, which this leads to a problem: Progression, because of the extensive crafting and building, distribution of resources if game is crucial, with manually designed world terrain, it is easier to control which section the player spawns in and the resources located within the player's view. The same logic transfers to other games like Legend of Zelda: Breath of the Wild, where the extensive storyline and the need for story-driven world-building are really complicated to create with procedurally generated maps. Manually designed maps also enable more control over the game's pace and storyline.

There is the elephant in the room, the one game that is famous for having infinitely generated maps, each one unique, each one almost completely procedural except for structures (Even structors are driven by procedural room gen with prebuilt room templates.) This game is called "Minecraft", as one of the first examples the paper above. There is a reason that Minecraft chooses to use procedurally generated worlds instead of manually designed terrain, the biggest reason being: Size. Given the almost 60 million blocks wide overworld, it is nearly impossible to be designed manually; this is also the selling point of Minecraft, a virtually infinite world with infinite possibilities. Stripping Minecraft of its procedural generation is comparable to removing guns from Call of Duty; it'd be detrimental to the brand identity.

The second reason: It's low-res by design. A game like Minecraft, where they take advantage of the voxel-like, relatively easy-to-generate terrain to create a charming retro atmosphere, is not only done for brand identity, but it is also done to reduce the distribution that world generation glitch creates. Despite being an AAA game now, Minecraft's terrain generator isn't perfect. There are numerous community examples of terrain failures, such as desert temples placed in other biomes, or Water not being updated when structures like mineshafts and strongholds are generated and exposed to it. The point is, this kind of world error has become common enough to be a part of the game's charm, like the famous far lands; they create an impact on the community even if they are just bugs that will eventually be fixed.

Minecraft and Satisfactory are both crafting, building, and exploring games, but because of the targeted game design, they decided to use different methods to provide players with immersive worlds. So the way to decide if a developer wants to make procedural world gen a part of the game is to pinpoint the target genre and the brand identity of the game first. Now, let's deep dive into the inner workings of Minecraft's Terrain Generation's most important aspects.

DOI: 10.54254/2755-2721/2025.LD28166

3. Terrain generation system

Now, onto how Perlin noise is used to make procedural generation possible. Again, using Minecraft as an example, the common type of terrain gen people see uses 2D procedural noise(Most likely perlin), 2D noise creates a map composed of white and blacklack, one of them being the max y level and the other being the min y level, the gradient between is valley and mountain ranges, note that this is not cave generation, as those would require 3D noise. The 2D present in terrain generation is made through pseudo-random values(Values that are generated based on a "seed", hence this is why it is recreatable consistently), Once the noise is generated, the process of generating the whole game world requires more logic, as minecraft have different biomes, each having their ownn noise pattern, they need to be compatible with each other and connect to adjacent biomes: There are 7 noise variables that determine the characteristics of a biome, that being "Temperature", "Humidity"/"Vegatation", "Continentalness", "Erosion", "Depth", "Peaks and Valleys" and "Weirdness" [7] (In the order from left to right, the order display in the F3 menu in game). These values are ranges and thresholds used for computer decision making, so they might not correspond to how humans translate these variables into real-world terrain characteristics.

The temperature variable controls the biome's temperature to the real-life counterpart: They range in five sections from frozen to warm, with the frozen being T:0, T<-0.45, and the Warm being T:4, T>0.55 [7]. The biome naming in this section is relatively straightforward, giving an example with the ocean biome sets, they are split into 2 categories except for T:4(Warm), the two categories being Ocean and Deep Ocean, with a prefix in the order from 0 to 4(frozen ocean & deep frozen ocean, cold ocean and cold deep ocean, ocean and deep ocean, lukewarm ocean and deep lukewarm ocean and warm ocean). Note that oceanic biomes (non-inland biomes) like the examples given above are not affected by humidity, erosion, and weirdness at all.

The humidity variable controls where dry and wet biomes are placed; these range from plains to different types of forests. They are also divided into 5 sections. Giving badlands biomes as an example, H=0,1 can generate badlands shown in Figure 1 and eroded badlands [7], H=2 can generate Badlands, and H=3,4 can generate wooded badlands shown in Figure 2. Although both Oceans & Badlands count as two biomes, they have variants that generate based on conditions, which ensures the variety you see throughout the terrain is refreshing and geologically consistent with the surrounding terrain. Other variables like "erosion" work in a similar range system, where biomes and biome variants are ranked on a scale and appropriately taken from the scale during a time of need. For example, continentalness creates beaches, separating large areas of land, and erosion creates flat non-oceanic areas, and peaks & valleys create valley-like structures, mostly filled with water because valleys in Minecraft usually extend below the water level.

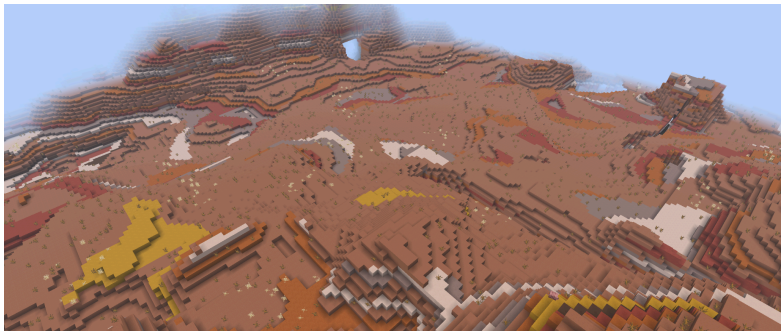

Figure 1. A visualization of Badlands.

DOI: 10.54254/2755-2721/2025.LD28166



Figure 2. A visualization of wooded Badlands.

4. Cave generation system

Along with surface biomes are underground cave systems shown in Figure 3. There are 3 underground cave system types [7], namely “cheese cave”, “spaghetti cave”, and “noodle cave” [7]. Cheese caves resemble large airholes in the terrain, spaghetti caves curve with minimal branches, while noodle caves are spaghetti caves but much more claustrophobic and harder to navigate in-game; they can be considered a subcategory.

The method that cheese caves use is by using perlin noise as a mask, taking the white section out, and using it as a mold to create the shape of the cave. This method creates big blob-type openings. There is a value called “Hollowness” that controls the openness of caves. With it being high, you can see different white sections blending to create larger, more connected cave structures.

Spaghetti caves are generated by tracing out where the black and white intersect on the noise map. Since the blobs are organically shaped, the curves that generate out of them feel like a long, stretchy, bendy tunnel. Noodle caves use the same algorithm to generate, except the range they generate is much smaller, sometimes not enough for an in-game player to navigate directly while walking. This adds variety to the system, mimicking the real-life cave structure’s cave diving process.

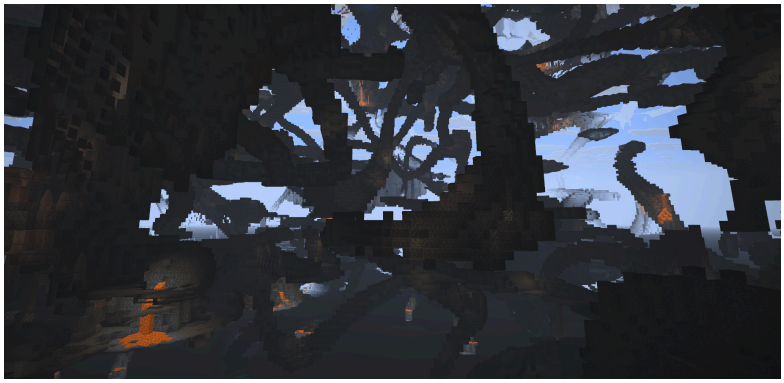

Figure 3. A visualisation of the cave system, with all cave types mixed in the final world.

5. Ore generation system

A key section of Minecraft is its mining system shown in Figure 4 and Figure 5. Ores integrate into the cave system to simplify progression, as caves provide an easy way to reach deeper depths. To make ore generation less uniform, Minecraftaft has an ore vein system. These ores are accessible at

DOI: 10.54254/2755-2721/2025.LD28166

some point from early game to late game, and they generate at different depths, making the ones in the late game harder to obtain without proper previous progression steps.

During 1.18(November 2021), they reworked where ores spawn and extended the underground's depth, making ores dynamically prefer certain Y coordinate values. Certain biomes also affect ore spawn, as badlands biomes generate more gold than typical [8]. These are all part of the world gen system.

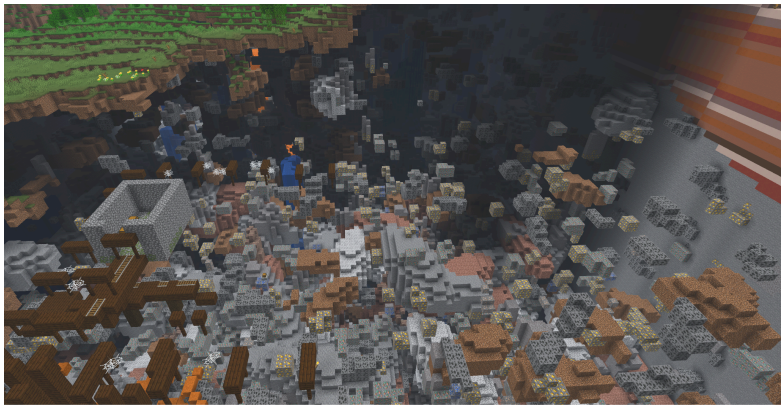

Figure 4. Visualization of gold generation in the Badlands biome

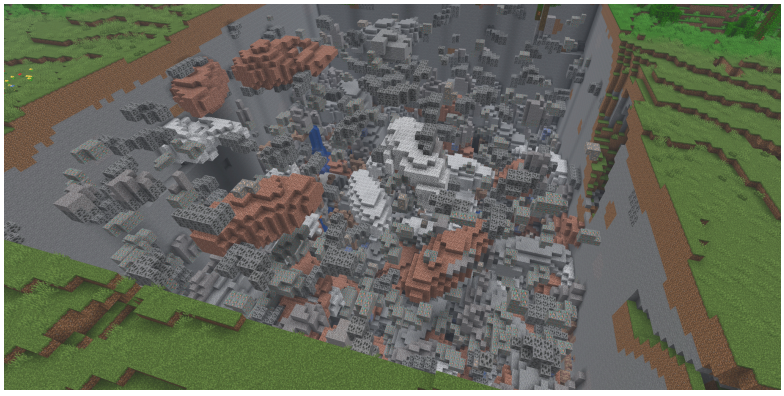

Figure 5. Visualization of gold generation in the Plains biome

A deeper level to this ore generation mechanic is that ore veins also appear in procedural blob or web-like patterns, making some discoveries more rewarding than others. There are 3 noises controlling ore vein generation: "Toggle", "ridge", and "gap" [9]. Toggle, also known as large ore veins, only exists for iron and copper (Other ores use the term blob/cluster, which are smaller in size but much easier to find); this is partially the reason why those two materials are seemingly the most abundant throughout the world. Toggle will attempt to generate Y=-60 to Y=51; outside this range, it is always the value of 0.

Ridge noise is for controlling which block is included in the vein [9]. If ridge>=0, this block will not be considered inside the vein; if ridge<=0, this block will be set either to the ore of the vein or the vein's filler (granite, tuff, which are for copper veins shown in Figure 6 and iron veins, respectively).

Gap noise controls the amount of actual useful ore blocks in that set vein [9]. This noise is applied on top of the noise used to create the vein in the first place. If this noise value is <-0.3, it's a filler block; otherwise, it takes the mentioned above toggle noise and maps it from [0.4,0.6] onto a [0.1,0.3] range [9]. Values outside are clamped to either 0.3 or 0.4; the system then generates a

DOI: 10.54254/2755-2721/2025.LD28166

random number [0,1]. If this rolled number is less than the clamped value, then an ore is placed with a 2% chance of being a raw variant of this ore; if not, then a filler block is placed.

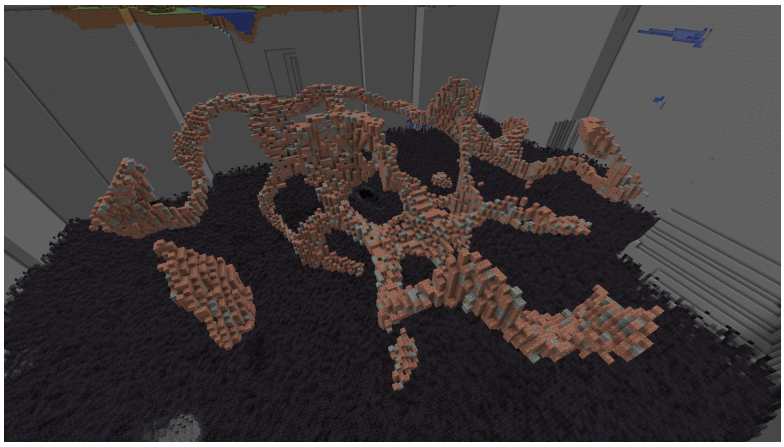

Figure 6. An extracted copper ore vein [9].

6. Conclusion

This is the summary of the most important world generation system in Minecraft. This system can also be transferred to other games, although for non-voxel games, the lack of convenience of having block grids will make the process slightly more complicated. With traditional prefab 3d models for ores, problems like depth buffering artifacts can appear.

These terrain/world generation algorithms can also be used to assist in manually mapping designed worlds, and the concept of noise extends far beyond world generation, procedural material, and just about most things procedural utilize layers of different formats of noise to get the desired outcome.

This paper aims to provide a simplified explanation of the inner workings of Minecraft's world generation system, as well as to clarify that not all genres of games require a procedurally generated world to feel alive and unique. In most scenarios, procedural generation is the icing on the cake, where it enhances the playing experience by making things more unpredictable.

In the future, this concept might be used in more than just survival open world; other genres that rely on unpredictability, such as stealth games, could also partially generate their rooms or terrain. This is only the beginning.

References

[1] Olsen, J. (2004). Realtime procedural terrain generation. mit.edu: 31.
[2] Panagiotou, E., & Charou, E. (2020). Procedural 3d terrain generation using generative adversarial networks. arXiv preprint arXiv: 2010.06411.
[3] Santana, P. P. F. (2020). 3D Terrain Generation using Neural Networks, Master's thesis, search.proquest.com.
[4] Jain, A., Sharma, A., & Rajan. (2022). Adaptive & multi-resolution procedural infinite terrain generation with diffusion models and Perlin noise. In Proceedings of the Thirteenth Indian Conference on Computer Vision, Graphics and Image Processing (pp. 1-9).
[5] Barton, H. (2025). Procedural terrain generation: noise functions, modern methods, and style transfer.
[6] Belda Calvo, L. (2021). Study of procedural terrain generation in plain and spherical surfaces.
[7] Minecraft.wiki (2025), world_generation. https: //minecraft.wiki/w/World_generation
[8] Minecraft.wiki (2025), Ore, https: //minecraft.wiki/w/Ore
[9] Minecraft.wiki (2025), Ore vein. https: //minecraft.wiki/w/Ore_vein