Transactions on Computer Science and Intelligent Systems Research
ISSN: 2960-1800, eISSN: 2960-2238 | Volume 9
AIDML 2025

Analysis of Procedural Content Generation and Stylisation Techniques in Developing Video Games

Junkai Xie *

Entertainment Arts and Engineering, University of Utah, Salt Lake City, United States

* Corresponding Author Email: u1535130@utah.edu

Abstract. This paper explores the evolving role of Procedural Content Generation (PCG) technology in developing video games, highlighting its importance in enabling automatic generation of scalable, dynamic, and visually distinct gameplay experiences. This research aims to provide an overview of the current methods of procedural terrain generation and assess how emerging generative artificial intelligence (AI) technologies can address their limitations. Specifically, the study reviews methods, including Perlin noise, fractal noise, voxel systems, physical simulation, and generative adversarial networks (GANs). Exemplary applications of traditional and modern PCG systems, including game applications, are analyzed. Experimental observations reveal that while PCG effectively generates structural diversity, generative AI models significantly improve context awareness and content richness, offering potential solutions to historical limitations of PCG. These conclusions indicate future developments where intelligent PCG frameworks enhance artistic control and environmental narrative depth. Integrating generative AI into PCG processes elevates asset creation and introduces a new text-driven and context-aware content generation paradigm.

Keywords: Procedural Content Generation; Generative AI; Terrain Generation; Generative Adversarial Networks.

1. Introduction

Throughout the development of the videogame industry, the developmental practices have constantly evolved to drive the presentation of the interactive experiences to be more appealing and dynamic. Out of all methods that emerged from development, one is exceptionally fundamental in introducing interesting graphics and enabling new gameplay cornerstones: Procedural Content Generation (PCG). With the implementation of PCG, the game system can generate assets, terrains, and environments without requiring direct human input. Incorporating PCG technology into videogames opens up new potential for developers to experiment with novel artistic presentations, ensures greater replayability by producing unique environments with each iteration, and enhances productivity within the developmental pipeline. Within procedural generation, different methods and stylization techniques help shape the appeal and coherence of generated environments, aiding the visual storytelling and player immersion aspects. With this clarification, this paper seeks to review the role and implementation of procedural generation and its stylization application to videogames and point to possible future directions of these technologies.

Procedural generation has been practiced in game development since its earliest days. Early applications of PCG date back to games like Rogue, released in 1980, which featured a random dungeon generation system to create unique playthroughs. In more recent releases, PCG is commonly implemented using techniques such as cellular automata, Perlin noise, fractals, and generative adversarial networks (GANs) to generate terrains and environments [1]. These evolved methods of PCG allow the technology to automatically produce 2d and 3d terrains that mimic real geographic features. This technical fidelity in recent videogame development has allowed PCG technology to serve as the pillar of gameplay, with examples being No Man's Sky and Minecraft, released respectively in 2016 and 2009, in both of which the procedurally generated terrains offer an almost infinite in-game environment for the player to explore and interact with [2]. Outside of allowing for automated in-game world generation, procedural generation in recent years, with its continued

development, has been incorporated into the production pipeline of many games to create large-scale terrain assets without the need for costly refinements [3].

Even with the capabilities to automatically generate almost infinite terrains and structures, PCG technology in its application in the videogame industry has never been able to fully replace the role of manual modeling due to its noticeable limitations. In most applications of PCG technology in videogames, the extent of variation or content depth of the generated assets is limited due to the random nature of the algorithms behind it. However, these shortcomings may finally be addressed with the recent emergence of generative Artificial Intelligence (AI) and Large Language Models (LLMs). This paper aims to provide an overview of the technical foundations of PCG technology, its applications in the gaming industry, and potential future developmental trajectories of the technology.

2. Methodology

2.1. Introducing Procedural Content Generation (PCG)

To investigate the application of PCG technology (as shown in Fig. 1) in the games industry, this methods section will provide overviews of the standard algorithms of PCG and, along with their practical applications, review their weaknesses. Section 2.2 overviews of standard PCG techniques: noise map, fractals, voxels, and machine learning based methods. This section provides the technical background for the reader to understand the mechanisms behind game procedural generation. Section 2.3 covers the topic of PCG's application in games through exploring the use of the technology in well-known games, highlighting areas of its implementation such as terrain generation, level design, object variability, and gameplay structure. Sources such as academic studies, surveys, and review articles are used for the investigation.

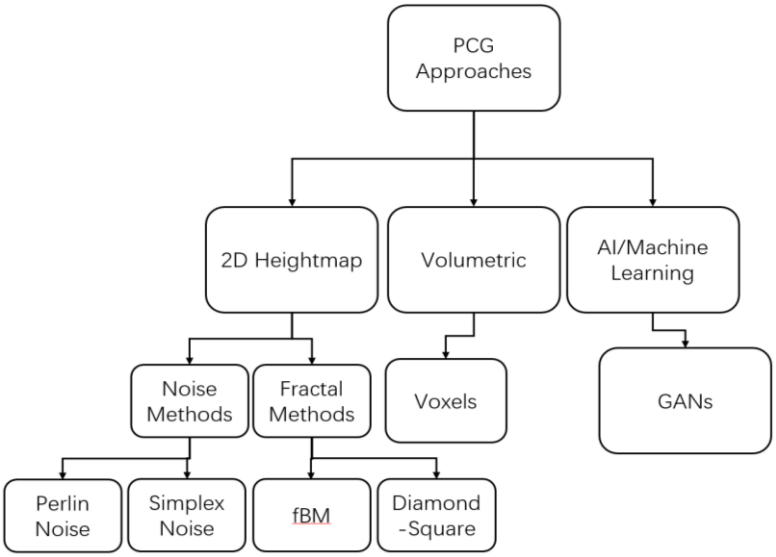

Figure 1. Approaches to PCG (Photo credit: Original)

2.2. Major Approaches to PCG

The technical basis of modern PCG in games is centred around a range of algorithms capable of generating patterns that produce planes and 3D meshes that simulate terrain, the most prolific of which are noise functions, which provide structured randomness to produce natural-looking terrain features (as shown in Fig. 2). Perlin and Simplex noise generate smooth, continuous heightmaps that simulate landscape elevation [4]. When translated to terrain height values, the smooth interpolation between height levels and the circular patterns in these noise maps produce geometries that resemble patches of smooth and short hills without abrupt changes in height. Simplex noise is a later improvement of Perlin noise, reducing computational complexity in higher dimensions and minimizing directional artefacts. These noise-based approaches create structures that simulate

geographic features such as mountains, plateaus, and valleys by mapping the terrain plane's heights according to noise maps generated [5].

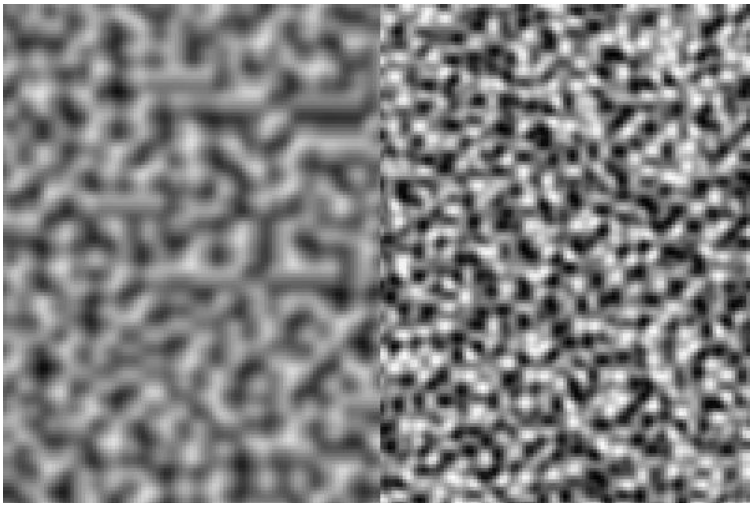

Figure 2. Map of Perlin Noise v. Simplex Noise [6].

Fractal-based methods and algorithms simulate natural terrain through recursive subdivision. The fractal method of Diamond-Square divides a grid into smaller subsections through iteration and applies randomness at each step, producing heightmaps with details that resemble naturally occurring variations. Another fractal method commonly used is Fractal Brownian Motion (fBM), whose results are shown in Fig. 3. fBM is a method that achieves terrain complexity without sacrificing performance, as it works by layering multiple octaves of noise, refining terrain details [7]. fBM introduces persistence and lacunarity parameters, which allow fine adjustments of the roughness and frequency distributions, allowing it to produce smooth and ragged terrains.

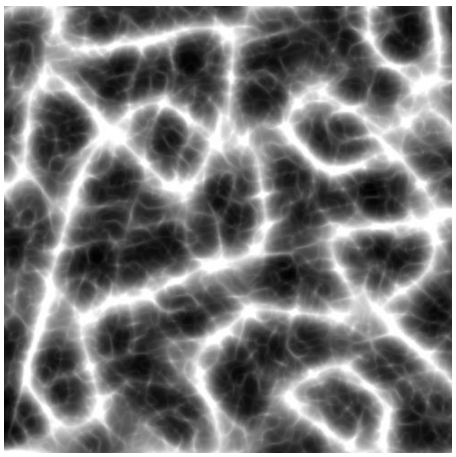

Figure 3. fBM Noise map with edited parameters to emphasize ridges [8].

Voxel-based terrain generation provides an enhanced approach, utilising a volumetric data structure rather than a heightmap to allow for complex 3d structures that require not only a singular alternating height map but volumetric variation, such as caves, overhangs, and tunnels [9]. This technique is particularly popular in games like Minecraft, released in 2009, in which the terrain is constructed from cubic elements, enabling further player exploration and modification of the environment. Using voxels permits fully destructible and editable worlds, which is not typically feasible with traditional heightmap-based systems. Outside the example of Minecraft, voxel-based PCG is also seen in games like No Man's Sky, in which the volumetric data applies to a terrain mesh, allowing for complex terrain structures but without the stylistic limitation to simple assets.

Deep learning models, particularly GANs, have been trained on real-world height data to generate realistic terrains alike while maintaining controllability (as shown in Fig. 4) [10]. Techniques such as conditional GANs or diffusion models can be used with terrain masks, elevation constraints, or biome

maps. These AI-based methods provide excellent detail and adaptability, allowing developers to generate unique landscapes based on intended visual style and gameplay.

Figure 4. A Volcanic Terrain Generated by a GANs Model with Ridges Modeled about Simple Line Strokes (Upper Left Corner) [10].

2.3. Analysis of the Relationships Between Players' Choices in Video Games and Reality

As the technology mainly revolves around the automatic generation of noise heightmaps, the most prominent applications of PCG in gaming have been terrain and environment generation. Games such as Minecraft and No Man's Sky use PCG to create almost an infinite number of different expansive worlds through random seeds. These games employ PCG methods to produce landscapes and 3d structures. In these cases, procedural systems determine the topography, biome distribution, resource placement, and environmental transitions.

Beyond terrain, the application of PCG can also be found in disciplines such as level design and dungeon generation. Looking over the history of videogames, the principles of PCG have been applied to titles as early as Rogue and Sid Meyer's Civilisation, and Minesweeper, released around 1980-1991. In these releases, the implementation of PCG is to automatically generate map tile patterns, allowing for dynamic game setups that randomize player experience on each game run. This first application of PCG sets the example of PCG as a cornerstone in game-level design, enhancing replayability by adding randomness to each generation. Behind this are the PCG algorithms like cellular automata and random walk algorithms, guiding the creation of room placement, connectivity, and pathfinding. Similar methods are also seen in more recent titles, such as Spelunky and Enter the Gungeon [11].

With the integration of machine learning models, such as GANs, procedural generation is evolving further, offering developers tools to generate content that aligns with stylistic and gameplay directions while maintaining fine visual fidelity. As games demand richer, larger-scale content, PCG remains a foundational method for scalable, dynamic design. In application, PCG can also be found in aspects of game development outside of 3d environment generation. Elements such as character design, quest structure, and item attributes also utilize procedural systems. Titles like Borderlands utilise procedural systems to dynamically generate weapon combinations, each with distinctions in attributes and appearance [12].

3. Discussion

3.1. Results Analysis

Figs. 5 and 6 present the in-game structures generated by PCG, with Fig. 5 representing a 2D plane tile dungeon and Fig. 6 displaying a whole terrain. The presented graphics show that PCG technology in its current application can construct complex structures utilizing its assembly of various prefab

assets, ensuring the automatic creation of dynamic game environments. With the proper in-engine rendering applied, the procedurally generated terrain of Fig. 6 can appear visually appealing with the appropriate details. However, one shortcoming of PCG applied here is the limitation of prop assets. As the algorithm of PCG is only responsible for constructing environmental layouts through various pattern-generation algorithms, the layout's assets are still dependent on manually created 3d models, prefabs, textures, etc. This limitation in application effectively limits the extent of content depth and specialization that a procedurally generated game world may have. It can make extended in-game exploration feel monotonous as the patterns of assets and terrain features appear repeatedly [9]. In the case of No Man's Sky, for example, it differentiates between generations of its different planet environments by giving each randomized preset of terrain styles, flora, fauna, structures, and one of its environment archetypes. In a short play, the environment may seem appealing, but after spending extended time with the game, the assets used to assemble the procedurally generated flora, fauna, and terrains will start to become noticeably repetitive, making the game world depicted lose its narrative credibility. So, while in the overview of PCG's application, thesis have explored its effects on in-game environmental generation, its weakness of being unable to generate environments with actual distinction and context-sensitive narrative makes it so that the application of it is limited to only basic terrain and prop layout generation and in disciplines lower in complexity [11].

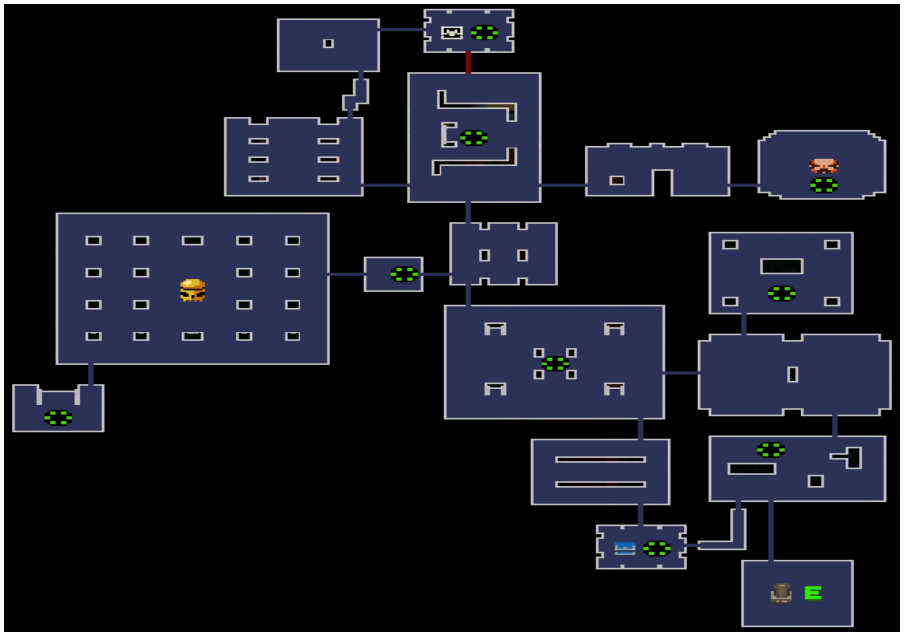

Figure 5. Enter the Gungeon map layout [6].

Figure 6. 3D Terrain Generated by the Game No Man's Sky [6].

4. Conclusion

The standard limitation in PCG is the lack of distinction and contextual awareness. In practice, classic PCG techniques have been unable to generate specific content with an understanding of thematic coherence or gameplay context. As a result, while structurally diverse, procedurally generated environments often lack meaningful differentiation, leading to visually repetitive, indistinctive outputs that negatively impact player engagement/immersion and environmental narrative depth.

Deep learning models, particularly GANs and diffusion models, may offer partial solutions. These methods have demonstrated the ability to learn complex data distributions and produce high-fidelity, diverse, and semantically meaningful outputs. When trained on datasets of various terrain types, architectural styles, or biomes, these models can interpret and replicate the surface appearance and the underlying logic and themes of the content. This enables the generation of environments that are not only visually distinct but also aligned with gameplay objectives, narrative beats, or artistic direction. Furthermore, integrating LLMs into PCG workflows introduces the potential for developing text-to-content generation, where developers or designers can describe intended outcomes in written language. This level of control enhances the control over input while the generative system preserves the ability to produce variations at scale. By connecting procedural structure and creative intent, generative AI supports a method enhanced in expressiveness and context awareness. This new development addresses the contextual ambiguity in traditional PCG applications and provides potential for more distinct, intentional, and narratively cohesive virtual worlds.

References

[1] POP Adrian, et al. Procedural Generation of landmasses and terrain. rochi.utcluj.ro, 2023, 1 - 7.

[2] VOULGARIS Georgios, IOANNIS Mademlis, and IOANNIS Pitas. Procedural terrain generation using generative adversarial networks. European Signal Processing Conference (EUSIPCO). IEEE, 2021: 686 - 690.

[3] PARK Gene. 'Starfield' makes good on its galactic promises. The Washington Post, 2023, 1 - 2.

[4] ETHERINGTON Thomas. Perlin noise as a hierarchical neutral landscape model. Web Ecology, 2022, 22 (1): 1 - 6.

[5] LAGAE Ares, et al. A Survey of Procedural Noise Functions. Computer Graphics Forum, 2010, 29 (8), 2579 - 2600.

[6] JAIN Aryamaan, AVINASH Sharma, and RAJAN. Adaptive & multi-resolution procedural infinite terrain generation with diffusion models and Perlin noise. Proceedings of the Thirteenth Indian Conference on Computer Vision, Graphics and Image Processing. 2022: 1 - 9.

[7] ONG Teong Joo, et al. Terrain generation using genetic algorithms. Proceedings of the annual conference on Genetic and evolutionary computation. 2005: 1463 - 1470.

[8] ALDA Álvaro. Introduction to Shaders. Beginner's Guide to Unity Shader Graph: Create Immersive Game Worlds Using Unity’s Shader Tool. Berkeley, CA: Apress, 2023. 1 - 21.

[9] VASILAKIS Andreas Alexandros, KONSTANTINOS Vardis, and GEORGIOS Papaioannou. A survey of multifragmenting rendering. Computer graphics forum. 2020, 39 (2): 623 - 642.

[10] GUERIN Éric, et al. Interactive example-based terrain authoring with conditional generative adversarial networks. ACM Transactions on Graphics, 2017, 36 (6).

[11] TAIT Emma, and INGRID Nelson. No scalability and generating digital outer space natures in No Man’s Sky. Environment and Planning E: Nature and Space, 2022, 5 (2): 694 - 718.

[12] LIMA Edirlei Soares, BRUNO Feijó, and ANTONIO Furtado. Procedural generation of branching quests for games. Entertainment Computing, 2022, 43: 100491.