A Survey of Procedural Content Generation of Natural Objects in Games

Tianhan Gao
Software College
Northeastern University
Shenyang, China
gaoth@mail.neu.edu.cn

Jiahui Zhu
Software College
Northeastern University
Shenyang, China
2071361@stu.neu.edu.cn

***Abstract*—The natural environment is one of the important research fields in game design and development. A good game environment is a key factor in the process of game development and player experience. Procedural Content Generation (PCG) is currently a wide range of fully automatic game environment generation technology. This paper introduces some algorithms and experimental results of PCG for natural objects (such as vegetation, river, and terrain), with special attention to the applicability and aesthetic visualization. It is found that the appearance of PCG greatly reduces the time needed to design large-scale natural landscape for game levels. Furthermore, PCG is able to adjust the natural landscape in real time to improve the overall game development efficiency.**

Keywords—game development, procedural content generation, natural objects, procedural modeling methods

I. INTRODUCTION

With the rise of mobile games, video games have become the most profitable parts in the entertainment industry. According to the "2020 China Game Industry Report", the actual sales revenue in China's game market has reached 278.687 billion yuan in 2020. The game industry has developed for more than ten years, and the technology of games and modeling is also constantly changed. From the early arcade video games to the present 3D virtual world, the automatic creation of content has a huge attraction for game production. For most designers, creating a game world still requires a lot of repetitive labor. In order to cope with such heavy work, designers have introduced a novel procedural content generation method that enables designers to create a complete 3D world within a short time. Procedural Content Generation (PCG) is a game design technology which uses automation generation technology instead of traditional manual creation. This automatic production mode greatly improves the production speed and effectively reduces the cost, and the error rates are low. *Elite* (1984) and its sequel *Frontier* (1993) are typical examples, putting an endless universe into a small floppy disk, dynamically generated galaxy system, dynamically generated checkpoints, and a whole set of features is all generated by programs. *Kkrieger* (2004) is a first-person shooting game in which everything is created dynamically while the program is running, including level, map, model, animation, and sound effects. But it is incredible that the entire game is only 96kb. According to the traditional game storage method, such a beautiful game may take a few hundred megabytes of space. Procedural content generation rules are not only applicable to video games. Desktop games such as *Catan: World Explorers* (1995) [1] requires players to create colonies using randomly assigned natural resources, and the player who gets ten points first through action is WINNER. In each game, randomly distributed resources will bring a new experience for players, and procedural content generation has brought great success for this game.

The natural environment is becoming more and more important to the game world. Hand-built virtual worlds are very strict, and once built, they cannot be easily modified. In particular, the modification of some large scenes may make designers have to start over. Procedural content generation (PCG) can perform high quality treatment for specific plant models, animal models, rivers, terrain, and buildings.

Obviously, this technology of PCG can mitigate the burden of content creation. In addition to entertainment [2], PCG can also be used for simulation, training, education and decision-making in other sectors of society, Such as military [3] training peacekeeping missions and simulating tactical decision-making scenarios、 rescue troops [4] need to train to rescue trapped people in buildings、 the road network generated by PCG can help driving school students learn more conveniently, and PCG can provide the required scenes from all walks of life in society to education and training in schools.

The paper first introduces the definition and classification of PCG, then some existing procedural content generation methods of natural environment such as vegetation, river, and terrain has been summarized. And in the last section, the combination of PCG and other fields in the future is discussed.

II. DEFINITION AND CLASSIFICATION OF PCG

After understanding the source of procedural content generation, this chapter will detail the concept of procedural content generation. Julian Togelius [5] and others believe that PCG refers to "All aspects of the game that affect gameplay other than nonplayer character (NPC) behavior and the game engine itself", including "terrain, maps, levels, stories, dialogue, quests, characters, rulesets, dynamics, and weapons". Hendrikx [6] and others believe that the idea behind procedural content generation technology is that the game content is not produced manually by human designers, but is generated by a well-defined process executed by computers. It is another choice to make complex game world in a limited time, and will not bring heavy burden to game designers.

Freiknecht and Effelsberg [7] give a more detailed explanation of the concept of PCG: Procedural content generation is to automatically create digital assets of games, simulations or movies based on predefined algorithms and modes, requiring minimal user input.

With the growing game world, PCG technology is now more mature. Pixar Studio has been named the company that uses process content generation in RenderMan, which shows

that the automated content generation has long been accepted by the movie industry.

The following figure shows the objects with the most frequent procedural content generation in the game world. Taking the game world as the background, the procedural content generation technology is used to create the required objects. In the following chapters, this paper will introduce the generation methods of natural objects such as plants, rivers and terrain.

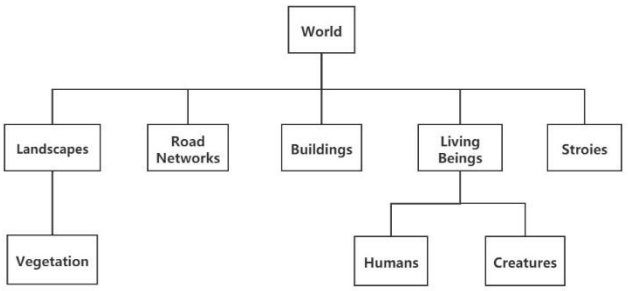

Fig. 1. Type classification of generated content

III. PCG OF NATURAL OBJECTS

A. Plant

Thousands of years of biological evolution have led to a complex ecological environment, which play a crucial role in various species and ecosystems. Object generation in nature, such as vegetation and terrain, has received much attention in the field of procedural content generation, and its development has reached quite mature so far.

1) Fractal of plants

Some of the plants in nature have inevitable self-similarities, and Mandelbort [8] is called fractal. Fractal [9] is a roughly divided geometry, which is divided into several parts of the original geometry, each of which is only different to size of the original whole. Fractal is defined as a geometric branch of describing the geometric patterns contained in nature. Fractal has global determinism and local randomness. Fractal structure [10] has higher stability and fault tolerance than Euclidean geometry with stronger certainty, this is why fractals are so common to nature, from trunks and branches to complex leaf vein structures.

Fractal objects have infinite details, have similar self-structure at different magnification levels, and the details of fractal objects are not directly visible, so they can gradually display after being magnified. This means that the higher the magnification, the more detail you get. Tree [11] has the greatest degree of self-similarity among fractal plants. Although tree is a very complex structure, it is well defined. Fractal geometry can be generated using a variety of methods, the most important and the most mature are L-System. In L-system, fractal can be used to model a two-dimensional tree according to its self-similarity and recursion.

2) L-System

As early as 1969, biologist Aristid Lindenmayer [12] used Lindenmayer (L-system) system to simulate the growth process of complex organisms such as algae and fungi, and later extended it to simulate higher plant species and complex branching systems.

System [13] is a context-free syntax. Each rule generated is only applicable to one symbol in geometry, and other symbols are not affected by the rules, starting from the initial structure. By replacing some parts of the syntax notation to form an object, and iteratively applying some rules for the string symbols to create a branch structure, each recursive iteration will increase the growth level of the string, and finally the string can represent the branch structure of the growing tree.

Define L-system G as a tuple $G = (V, \omega, P)$.

V : V (alphabet) or letter is a set up to V and formal symbols.

ω : The initial state of the system is defined, which is called axiom. Axiom is a string composed of V symbols, The string set of V is denoted as V^*.

P : A set of production rules a symbol, map $a \in V$ to string, $\omega \in V^*$ is written as $P: a \to \omega$, Variables can be replaced by a combination of constants and other variables, Predecessor or successor.

Rewriting——The basic idea of L-system. The rewriting rule defines that the left side of the generation can be replaced by the right side, and it can be replaced repeatedly as needed. For example, given two symbols A and B, the results obtained according to the rewriting rules are as follows:

$$a \to ab,$$

$$b \to a.$$

This principle was originally used by Chomsky [7] in describing programming languages. However, unlike Chomsky's language, L-Systems requires every rewriting rule to be applied once in each round, on the grounds that plant growth is based on cell division and occurs in parallel for all cells. If 'a' in the above example is used as the initial string, the process can be expressed as shown in Figure 2.

a
ab
aba
abaab
abaababa
abaababaabaab
.....

Fig. 2. String tree generated by rewrite rule

3) Visualization of L-system

The recursion of L-system leads to self-similarity, so it is easier to obtain fractal and branching forms. Increasing recursion will lead to the "growth" of models and generate more complex self-similar structures, which can be represented by symbols and graphics. Assuming that the length is h and the rotation angle is δ, then the symbolic commands of L-system used to describe tree visualization in the paper are as follows.

- F: Move one unit forward and draw a line.
- +: Turn right δ degrees (clockwise).
- -: Turn left δ degrees (counterclockwise).

- [: Save the current position and move according to the next command.
-]: back to the original position stored by the symbol "[".

The problems with the previous syntax explanation can be described by the steps arranged by the system. Then, with the help of the general algorithm of fractal object interpretation on L-system, the syntax can be interpreted as a graph through the following steps:

- Enter the number of rewriting rules(n), inclination angle (δ), and segment length (h).
- Determine the starting angle a_0, enter the value of a_0 to get the starting point F. Then, enter F_0 in the production rule formula p to obtain P_0.
- After each iteration, the next angle a_n and the next point F_n will be obtained. Then, enter F_n on P_{n-1} in the production rule to obtain P_n.
- Some line segments are obtained from axioms and production rules.

For graphical representation, Use the *turtle* models to reconstruct the tree graph, and the following are the experimental results from several generation rules used in figures 3 to 5.

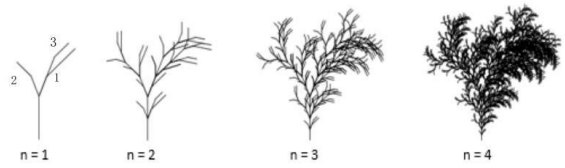

Fig. 3. Fractal tree generated by L-system, δ=25°, Axiom P: F:ω: *FF[+F+FF][-F-F][+FF+F]*

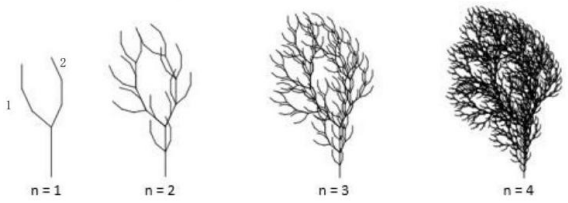

Fig. 4. Fractal tree generated by L-system, δ=25°, Axiom P: F:ω: *FF-[-F+F+F]+[+F-F-F]*

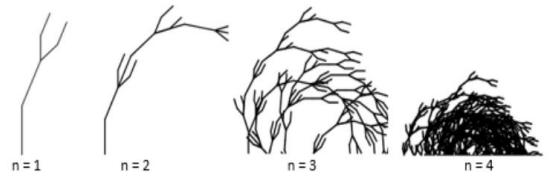

Fig. 5. Fractal tree generated by L-system, δ=25°, Axiom P: F:ω: *FF+FF[+F-F][-F+F]*

Figures 3 through 5 show the results of the spanning tree model. It can be seen that the tree model is different to different production rules. Figures 3 and 4 are close to the visualization of the tree and are more visually realistic. Figure 5 [11] successfully constructs the syntax, but fails to visualize the tree model.

Applying L-System to 3D plants is also very easy. Just add "tilt left", "tilt right", "tilt forward", "tilt backward" [7] at each decision point, replace the initial "turn left" and "turn right" operations with these operations, these short rules can be used to generate different 3D plants. Similarly, L-system is also suitable for generating shrubs and other types of plants. Up to now, the application of L-system is the most extensive method in plant procedural content generation. The latest research also confirms that the application of L-system has a high maturity.

B. Rivers

Water always plays a vital role in games. To make the natural environment in the game as detailed as real life, and excellent water resources are indispensable if the natural environment in the game are to be meticulous as in real life. On the contrary, it will make the players who were there instantly to break away.

Although water resources are always the same as a single element, the formation of rivers, lakes, oceans, and waterfalls is very different in many ways. Under the calm sea, there are sometimes more turbulent undercurrents; Sometimes the lake is calm like a mirror. Rivers generally keep flowing, and the creation of rivers [7] is usually carried out in two ways: When the terrain is created; Or place the landscape in a separate step later. Several authors have proposed algorithms specifically for producing rivers:

Kelley et al. [7] was the first person to propose a river network by program. They started from a single river path, formed a river network by recursive subdivision, and then filled the river network with scatter data interpolation function. Next Prusinkiewicz and Hammel [17] put forward a fully automatic method, which combines L-system with topographic erosion, and combines the generation of curved rivers with height map subdivision scheme. In the starting triangle of the river, one edge is marked as the entry and the other edge is marked as the exit. In the subdivision process, triangles are decomposed into smaller triangles. The elevation of the triangle containing the river is set as the sum of all negative displacements on all recursive levels of rivers. While other triangles are treated with mid-point displacements. After 8 or more iterations, the river will be quite natural [15].

In the design and implementation of checkpoints, creating natural phenomena requires changing the height of terrain. Huijser [16] introduced the concept of "cross-section" to express the formation of rivers, and used shape features to overcome the problem of asymmetric cross-section when rivers bend. Shape characteristics describe the local width, curvature, and slope of the shape, and create a richer river landscape according to the shape features of the river.

In order to increase the authenticity of river landscape generated, A. Peytavie and T. Dupont [14] and others put forward a novel program framework to create River Landscape: taking bare-earth as input, deducing river network trajectories affected by water flow, carving riverbeds in terrain, and then automatically generating corresponding blend-flow tree for the water surface. The width, depth, and shape of the riverbed is derived from topography and river type. The water surface is defined by a time-varying continuous function encoded as a blend-flow tree, in which leaves are parameterized procedural flow

primitives. The resulting framework can produce various of river forms, ranging from delaying winding rivers to the torrent of surging currents. These models also include surface effects, such as foam and leaves flowing down the river.

C. Terrain

Automatic terrain generation is one of the main topics of procedural content generation. It starts from natural phenomena such as plant growth and terrain elevation, and has been extended to the automatic generation of the urban environment. In previous experiments, the terrain is generally represented by height map [18], also known as Digital Terrain Model (DTM), which is a set of approximate elevation levels of a group of discrete points in the grid. The height of these points is the vertical distances between the terrain points and the reference surface. Usually, height map is composed of a gray bitmap, where elevation is represented by the gray shadow of the bitmap pixels. Then, use polygon mesh to visualize in 3D space. The whiter the pixel, the higher the elevation point. Figure 6 shows Mount Everest and other surrounding mountains represented by greyscale [7]. The higher areas are represented by lighter pixels and the lower regions are represented by darker pixels.

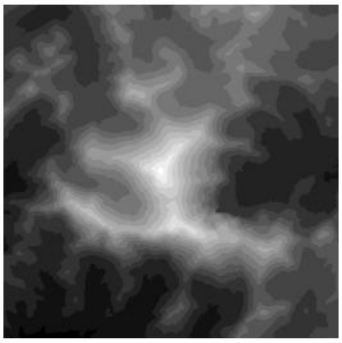

Fig. 6. Height map of mountains obtained in greyscale

The early height maps generation algorithms are based on subdivision methods, which refines the rough elevation map by iteration, and some random parameters are added in each iteration to change the elevation details. The first subdivision algorithm is called the mid-point displacement method: a rough height map is subdivided iteratively, and controllable randomness is used to add details in each iteration. In this method, Miller [18] set the elevation of a new point as the average of a triangle or diamond plus a random offset. According to the roughness of the generated height map, the range of offset at each iteration is reduced. The terrain generated by this method is fractal Brownian motioned (FBM) surface.

The generation of height map is usually based on fractal noise generator. The 2D Perlin noise map created by fractal Brownian Motion [20] is a series of fluctuation data onto smoothness and predictability (see Fig. 7). It generates noise by sampling and interpolating points in a random vector grid, and scales and accumulates several noises with increasing frequency into an elevation map, which is suitable for creating a landscape with mountains and valleys.

Perlin noise algorithm has excellent performance, because each grid point can be calculated independently of the values of neighbouring points, it is very suitable for parallel processing. However, the terrain generated by noise algorithm is generally uniform without the change of surface details. Therefore, if you want to add detail features of the smooth terrain, you can further to modify the terrain using algorithms based on physical phenomena, such as erosion.

Fig. 7. Output of simple Perlin noise on terrain grid

Musgrave [20] et al. realize the physical erosion process of local or overall erosion characteristics in height field by simply simulating the natural erosion process. The terrain generated by this method has the characteristics of fractal tree and arbitrary local control of cross dimension, which is not available in previous methods. They also proposed a global simulation to simulate what is called thermal weathering, Hydraulic erosion forms valleys and drainage networks, thermal weathering wears steep slopes and forms pluvial rocks at their feet. Compared with hydraulic erosion simulation, thermal weathering simulation can get more real results from a shorter time.

Although these erosion algorithms greatly improve the authenticity of mountain terrain, they need to run hundreds to thousands of iterations. Another method based on natural phenomena is Voronoi Tessellation [19] method, which usually occurs to the valley where the mountains meet, that is, the collection of all points in the Voronoi Tessellation region closest to the center of the region. By occupying the terrain mesh, the author raises up the mountains and the surrounding terrain at the same time, so each of the adjacent vertices is lifted. From the top view, the Voronoi Tessellation can be finally obtained. This method greatly saves the time of generating mountain terrain by programming content.

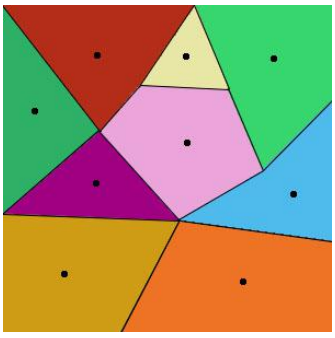

Fig. 8. Ordinary plane Voronoi diagram

IV. CONCLUSION

Procedural content generation is becoming more and more comfortable in creating the natural objects and some complex aspects of the game. However, these technologies need intuitive parameter control, powerful editing, and result visualization, for real-time operation. This paper introduces the procedural content generation methods and examples of plants, rivers, and terrain in games.

According to the statistics in [21], the algorithm families generated by application in the papers on PCG in the past ten years are shown in Figure 9. The overall proportion of each algorithm is relatively low, but the selection of algorithms is usually related to the programmer's personal preferences. Grammar has faced a downward trend since it became popular in early 2010-2013. Declarative and constructive methods also seem to be used smoothly in this decade. In addition, artificial intelligence (AI) is rising with a steady trend, which has become a popular way in game development.

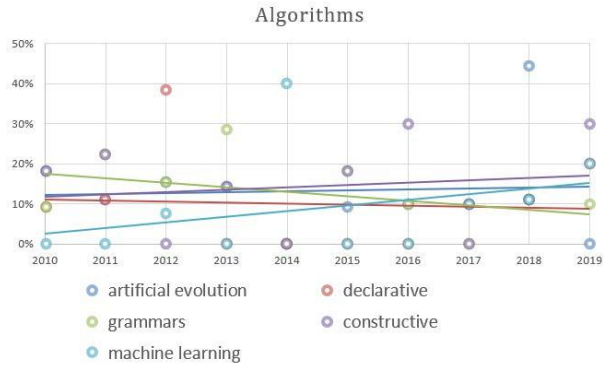

Fig. 9. Algorithm families applied in PCG papers in the past decade

Nowadays, AI has become an interdisciplinary field involving computer science, medicine, agronomy, mathematics, philosophy, and other disciplines. In terms of algorithm, AI can also be combined with PCG. Through machine learning, the concept of PCG is more extensive, which promotes the research of PCG seminars. Therefore, it is predicted that the combination of PCG and machine learning will continue to show an upward trend. In the era of abstract visualization of network traffic, PCG is a very important link in game design. PCG is able to stimulate designers' creativity. Designers can make full use of PCG to show us a new world they have never seen before.

ACKNOWLEDGMENTS

This paper is supported by the Fundamental Research Funds for the Central Universities under Grant Number: N2017003.

REFERENCES

[1] M. Wang, "Java Settlers Intelligente agentenbasierte Spielsysteme für intuitive Multi-Touch-Umgebungen," Ph.D. dissertation, Dept. Comput. Eng., Free Univ., Berlin, Germany, 2008.

[2] G. N. Yannakakis, and J. Togelius, "Experience-driven procedural content generation," *IEEE Tans On Affect Comput*, vol. 2, no. 3, pp. 147-161, 2011.

[3] K. Stanley, B. Bryant, and R. Miikkulainen. (2005, April). Real-Time Evolution in the NERO Video Game. [Online]. Available: https://ac.scmor.com/

[4] D. D. Djordjevich et al., "Preparing for the aftermath: Using emotional agents in game-based training for disaster response," *IEEE Sym On Comput Intell and Games*. 2008, pp. 266-275.

[5] J. Togelius, G. N. Yannakakis, et al., "Search-based procedural content generation: A taxonomy and survey," *IEEE Trans On Comput Intell and AI in Games*, vol. 3, no. 3, pp.172-186, 2011.

[6] M. Hendrikx, S. Meijer, et al., "Procedural content generation for games: A survey," *TOMM*, vol. 9, no.1, pp. 1-22, 2013.

[7] J. Freiknecht, and W. Effelsberg, "A survey on the procedural generation of virtual worlds," *Multimodal Technol and Interact*, vol. 4, no. 4, 2017.

[8] B. B. Mandelbrot, "The fractal geometry of nature," in *WH freeman*, vol. 1, New York, 1982.

[9] M. F. Barnsle, "Fractals everywhere," Academic press, 2014.

[10] P. H. Carr, "Does God play dice? Insights from the fractal geometry of nature," *Zygon®*, vol. 39, no. 4, pp. 933-940, 2004.

[11] E. W. Hidayat, I. Putra, A. D. Giriantari, et al., "Visualization of a two-dimensional tree modeling using fractal based on L-system," in *IOP Conf Series: Mater Sci and Eng*, 2019.

[12] G. Ochoa, "An introduction to lindenmayer systems," 1998. [Online]. Availabe: *http://www. cogs. susx. ac. uk/users/gabro/lsys/lsys. html*.

[13] M. H. Tanveer, A. Thomas, X. Wu, et al., "Simulate forest trees by integrating l-system and 3d cad files," *IEEE 3rd ICICT*, pp. 91-95, 2020.

[14] A. Peytavie, T. Dupont, E. Guérin, et al., "Procedural Riverscapes," *Comput Graph Forum*, vol. 38, no. 7, pp. 35-46, 2019.

[15] R. M. Smelik, T. Tutenel, R. Bidarra, et al., "A Survey on Procedural Modelling for Virtual Worlds," *Comput Graph Forum*, vol. 33, no. 6, pp. 31-50, 2014.

[16] R. Huijser, J. Dobbe, W. F. Bronsvoort, et al., "Procedural Natural Systems for Game Level Design," *IEEE Brazilian Symposium on Games and Digital Entertainment*, pp. 189-198, 2010.

[17] P. Prusinkiewicz, M. Hammel, "A fractal model of mountains with rivers," *InProceedings of Graphics Interface* , vol. 93, no. 4, pp. 174-180, 1993.

[18] R. M. Smelik et al., "A Survey of Procedural Methods for Terrain Modelling," *3AMIGAS*, vol. 2009, pp. 25-34, June, 2009.

[19] K. S. Emmanuel, C. Mathuram, A.R. Priyadarshi, et al., "A Beginners Guide to Procedural Terrain Modelling Techniques," *IEEE 2nd ICSPC*, pp. 212-217, March, 2019.

[20] F. K. Musgrave, C. E. Kolb, R.S. Mace, "The synthesis and rendering of eroded fractal terrains," *ACM Siggraph Comput Graph*, vol. 23, no. 3, pp. 41-50, 1989.

[21] A. Liapis, "10 Years of the PCG workshop: Past and Future Trends," in *International Conference on the Foundations of Digital Games*, pp.1-10, 2020.