Algorithms and Approaches for Procedural Terrain Generation

A brief review of current techniques

Thomas J. Rose
School of Science and Technology,
Computing and Immersive Technologies Department,
University of Northampton,
St. Georges Avenue, Northampton, NN2 6JB, UK
Email: Thomas.Rose3@northampton.ac.uk

Anastasios G. Bakaoukas
School of Science and Technology,
Computing and Immersive Technologies Department,
University of Northampton,
St. Georges Avenue, Northampton, NN2 6JB, UK
Email: Anastasios.Bakaoukas@northampton.ac.uk

***Abstract*—This paper aims to discuss existing approaches to procedural terrain generation for games. This will include both the many functions that are used to generate 'noise' (something that has proved exceptionally useful in procedural terrain and texture synthesis) as well as some advanced procedural content generation techniques. The paper concludes with a summary of the discussed material while attempting to highlight areas for future research.**

***Index Terms*—Pseudo-random, fractal, noise function, terrain.**

I. Introduction

One of the most integral aspects of the majority of video games is the terrain that the player will traverse. Producing a virtual game world along with appropriate textures and scenarios for a player to explore requires a lot of both time and resources which can significantly increase development costs.

Procedural Terrain Generation (PTG) can streamline much of this work through the use of pseudo-random algorithms. The term PTG refers to the creation of content by an automated system, rather than being produced manually. Terrain and texture generation can be performed by utilising algorithms to produce noise; while this noise looks completely random and unique, it is actually often the result of a pseudo-random set of procedures known as a noise function. The noise produced by these functions is useful as it has structure, rather than being totally random.

In addition to noise functions, other novel approaches have been taken towards procedural terrain generation such as the use of software agents, evolutionary algorithms and erosion modelling techniques.

II. Background

Procedurally generating game terrain can be difficult because of the unusual shapes and patterns found in the natural world. One of the key challenges of terrain generation is mimicking the way in which natural looking terrain is not constructed from Euclid shapes such as squares and triangles etc. Instead, terrain is typically described as fractal in nature. The term fractal was originally coined by Benoit Mandelbrot to help explain some of the irregular patterns that are found in aspects of the natural world such as coastlines, clouds and tree bark [1]. Fractals have two fundamental traits: they are both self-similar and chaotic. Self-similarity refers to the manner in which a fractal can be subdivided into smaller versions of itself. Additionally, fractals are also frequently described as chaotic due to their infinite complexity; fractal patterns are products of recursion and, as a result, can be viewed at an infinite number of scales. With computers, it is possible to produce seemingly random fractal terrain and textures through the use of noise functions; a term which refers to a set of instructions that can be used to generate pseudo-random noise.

The terrain data produced by a noise function can both be rendered directly and/or used to produce a heightmap. A heightmap is an image that can used to store the elevation value of each point of the terrain using one or more colour channels. For example, a grayscale heightmap might interpret the elevation data as luminosity resulting in white being the highest possible point and black being the lowest.

It is important to note that many noise functions are not inherently fractal. Some algorithms such as Value Noise and Perlin Noise are instead used in conjunction with fractional Brownian motion (fBm) to produce fractal images. Also known as 1/f noise, fBm is achieved by summing together multiple octaves of noise, each with an increased frequency and smaller amplitude; this results in finer and more detailed noise.

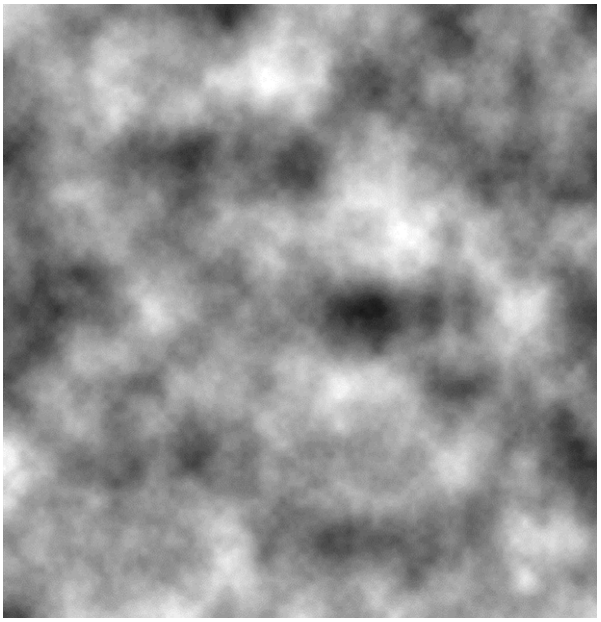

Figure 1. Seven octaves of Value Noise rendered as a heightmap.

978-1-5090-2722-4/16/$31.00 ©2016 IEEE

III. Noise Functions

Some of the core noise functions that have been previously utilised in procedural game terrain generation include: the Diamond-Square Algorithm [2], Value Noise [3], Perlin Noise [4], Simplex Noise [5] and Worley Noise [6]. The subsequent table [Table I] shows a comparison between these popular noise functions based on their speed, memory requirements and the quality of the noise they produce.

TABLE I. Noise Functions

Algorithm	Speed	Quality	Memory Requirements
Diamond-Square Algorithm	Very Fast	Moderate	High
Value Noise	Slow - Fast*	Low - Moderate*	Very Low
Perlin Noise	Moderate	High	Low
Simplex Noise	Moderate**	Very High	Low
Worley Noise	Variable	Unique	Variable

*Depends on what interpolation function is used
**Scales better into the higher dimensions than Perlin Noise

Aside from the Diamond-Square Algorithm all of the noise functions mentioned here are not natively fractal and typically used in conjunction with fBm to create fractal images.

The most suitable noise function will vary between terrain generation systems. If speed is essential, the Diamond-Square Algorithm is a great option as it is much faster than all the other algorithms and its fractal nature means that multiple octaves of noise are not required to produce fractal images. Value Noise is a great alternative if memory is in short supply and is also easily customisable due to its simple nature. Worley Noise, meanwhile, provides noise that is much more visually unique than the other noise functions and may be desirable in certain scenarios. For overall quality, however, Simplex Noise is the ideal choice although classic Perlin Noise will likely be sufficient in the vast majority of circumstances.

IV. Other Approaches

Software agents provide an interesting alternative to noise functions for generating terrain. One of the great advantages they yield is that they are much controllable and can be used in a PTG system that works with designer-chosen parameters, rather than abstract random number seeds [7].

Evolutionary algorithms are somewhat more difficult to control but provide a valuable way of generating a broad spectrum of game content types. The initial content population and fitness function are the best ways an evolutionary algorithm can be regulated along with reducing the amount of mutation that takes place each generation. This issue with reducing the mutation factor, however, is that this is where the algorithms random-element comes from and doing so makes the population more likely to stagnate [8].

Physically-based erosion models are a useful way of giving game terrain a more worn and natural appearance. Although hydraulic erosion is notably slower than thermal erosion and requires approximately three times the amount of memory, it produces significantly better results [3]. Simulating hydraulic erosion on the GPU can aid in overcoming the algorithm's general speed limitations [9].

V. Conclusion

There are a good variety of effective techniques currently that can be applied to procedural game content generation. The discussed noise functions are exceptionally useful for generating initial terrain and each one possesses its own useful set characteristics. Despite this, they can be difficult to influence in meaningful ways as their outputs are simply the product of the computation of pseudo-random numbers.

Although not as easy to manipulate as software agents, evolutionary algorithms provide a method of generating evolved game content by producing an initial content pool and improving it with successive generations by evaluating its members using a fitness function; which can be adjusted to the needs of the system and offers a degree of controllability.

Physically-based simulations are not necessarily well suited to producing procedural terrain on their own but can offer a way to further shape a virtual world and add a further layer of realism to it.

Doran and Parberry's software agents perhaps constitute the most significant step towards generating terrain using user-friendly parameters relating to terrain features rather than abstract numerical inputs. Despite this, adaptive and controllable content generation for games on the whole is certainly an area that could be explored further.

A final notion to be considered going forward is the way in which procedurally generated content can be evaluated. Content produced by a PTG system can be difficult to appraise as what makes game terrain "good" is a largely subjective matter. Furthermore, it can be challenging to compare the quality of different terrain-generating algorithms and techniques as the final output from a PTG system is heavily affected by the rendering techniques that are used.

References

[1] B. Mandelbrot, "The Fractal Geometry of Nature." W. H. Freeman and Co, 1982.

[2] G. S. P. Miller, "The definition and rendering of terrain maps," *ACM SIGGRAPH Comput. Graph.*, vol. 20, no. 4, pp. 39–48, 1986.

[3] T. Archer, "Procedurally Generating Terrain," *3rd Int. Conf. Comput. Support. Educ.*, 2011.

[4] K. Perlin, "Noise Machine." [Online]. Available: http://www.noisemachine.com/talk1/index.html. [Accessed: 29-Apr-2016].

[5] S. Gustavson, "Simplex noise demystified," *Linköping Univ. Schweden*, p. 17, 2005.

[6] S. Worley, "A cellular texture basis function," *Proc. 23rd Annu. Conf. Comput. Graph. Interact. Tech. - SIGGRAPH '96*, pp. 291–294, 1996.

[7] J. Doran and I. Parberry, "Controlled Procedural Terrain Generation Using Software Agents," *IEEE Trans. Comput. Intell. AI Games*, vol. 2, no. 2, pp. 111–119, 2010.

[8] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-Based Procedural Content Generation : A Taxonomy and Survey," *IEEE Trans. Comput. Intell. AI Games*, vol. 3, no. 3, pp. 172–186, 2011.

[9] X. Mei and P. Decaudin, "Fast Hydraulic Erosion Simulation and Visualization on GPU," *Pacific Graph.*, 2007.