Towards a comprehensive classification for procedural content generation techniques

Nathan Oliveira[1]* Rodrigo Duarte Seabra[2]

[1]UNIFEI – IESTI – Brazil
[2]UNIFEI – IMC – Brazil

ABSTRACT

The video game industry has relied on procedural content generation techniques for decades, be it as means of trading memory consumption for processing power or as a way of creating content either faster of with more variation than what can be handmade. Still, until recently, very little research has been done towards a unified taxonomy that could then be used to better understand these procedural techniques and help their users choose the best algorithms for their particular needs. We aim to provide a plausible taxonomy of procedural techniques, based on related work by Hendrikx et al. [5], Kelly and McCabe [6], and Togelius et al. [16]. To achieve this goal, we identify their taxonomies, group them in a single taxonomy and explain each method of the taxonomy. With this, we conclude that there is a need to discuss which the best way to classify the procedural techniques currently in use is, in order to help the research on further techniques.

Keywords: Procedural Modeling, Procedural Content Generation, Algorithm Analysis, Game Content Generation

1 INTRODUCTION

With the advance of computer graphics, we can render scenes and virtual worlds with ever increasing realism. Initially, these elements were created manually, through 3D editors. However, this process is becoming too expensive, with games such as *Star Wars: The Old Republic* and *Grand Theft Auto V* having costs of US$200,000,000 and £170,000,000, respectively [17], with *Grand Theft Auto V* having more than a thousand developers, and with complaints that some development teams worked twelve hours a day, six days a week for various months [14] to achieve the game production deadlines. Therefore, procedural techniques have been used to help and to speed-up the content creation.

There is currently a growing demand for realism in video games. This desire comes from players wanting more immersion in their games, and causes an increase in production costs that can lead to higher games prices or even the bankruptcy of game studios. These problems arise from the fact that the production of game content is a complex task, which requires skilled content creators. They create the game contents with the help of specialized software, but this process does not scale well, and the need for skilled people grow at a rate at which makes it hard to find the workers needed.

With these problems in mind, one good solution is the use of procedural generation for the game content. In the past, procedural content generation (PCG) was used to overcome limitations in storage space, by storing only a procedure that could create the necessary content during the execution of the game. Although some games still use procedural techniques for trading memory use for processing power – especially to reduce the bandwidth use between CPU and GPU, these techniques provide ways of creating content faster than by hand and with more variability than that obtained by creating the content beforehand.

PCG is a set of techniques that can produce content for creating terrains, maps, characters, sounds, and other content used in digital games. The research on these techniques began in the 1970s, with the works on shape grammars, by Stiny and Gips [15], and fractals, by Mandelbrot [9]. Shape grammars are types of formal grammars that can produce strings of geometrical shapes by using a defined set of production rules. Fractals are mathematical sets that present recursive patterns and can be used to represent content with infinite level of detail.

Even though for the games industry PCG has already raised great interest, scientific research on the subject is still in its infancy. This can because very few authors of published papers about PCG try to further research about it, with the vast majority of the publications focusing on the creation of commercial products, sometimes omitting vital parts of the techniques created, which prevents both the reproduction of the results and a further study on the proposed technique. Another problem of this limited scientific interest is that even after decades of active use and research of PCG techniques, we still do not have a definitive taxonomy of the main algorithms and procedures used.

In Brazil, the PCG field is just starting to draw attention, with little national production. Among these works, we can highlight the research by Miranda, Cordeiro and Chaimowicz [10], which proposes a technique for generating terrains using both CPU and GPU; Silva, França and Cabral [12], with a technique based on fuzzy systems for generating games soundtracks; and Leite and Lima [8], which proposes a technique for creating bidimensional maps of caves, dungeons and islands. Apart from these works, four masters' dissertations about PCG were found: Bevilacqua [2], with a technique to generate coastal terrains in real-time; Pereira [11], with a technique based on fuzzy systems to create tasks for an educational game focused on teaching reading and writing skills; Carli [3], with a technique for generating tridimensional canyons; and Duarte [4], with a technique to generate scenes based on objectives.

Recent proposed taxonomies include the works by Kelly and McCabe [6] that surveys techniques for city generation, Togelius et al. [16] with a survey on techniques based on search and optimization, and Hendrikx et al. [5] with a survey into algorithms for a wide range of content types. These taxonomies are shown in Section 2. Some other works proposes taxonomies for related techniques, such as Smelik et al. [13], that proposes a taxonomy for procedural modeling techniques. However, our work focuses on techniques that can generate content without user intervention. The work of Smelik et al. [13] does propose a family of procedural modeling that can be used for PCG, but we do not list it in this paper for brevity, since the families are very similar to what Hendrix et al. [5] proposed.

The main goal of our work in unifying and expanding these previous works is to create a taxonomy that can be used for any type of PCG technique, not being focused in a special type of content. The proposed taxonomy could then be used to create databases of implementations of PCG techniques that can be browsed and filtered

*e-mail: nathan@unifei.edu.br

to find PCG techniques that match the needs of both developers and researchers. This is because we still do not have a readily available database for code of procedural methods, and this forces anyone that wants to study or research PCG methods to implement them from scratch or spend a good amount of time searching for already made implementations on the internet. This database would help reduce the time needed to obtain, understand and modify common PCG techniques.

This paper is structured as follows: Section 2 presents a review of the taxonomies of Hendrikx et al. [5], Kelly and McCabe [6], and Togelius et al. [16]; and Section 3 describes our proposed taxonomy. We end the paper with our conclusion and possible future works in Section 4.

2 Related work

Kelly and McCabe [6] define seven criteria to judge the quality of results of the surveyed techniques: (i) realism, (ii) scale, (iii) variation, (iv) input, (v) efficiency, (vi) control, and (vii) real-time. Although proposed for analyzing city generation, these criteria can be used in different types of techniques. Based on their definitions, we can broaden each criterion: (i) *realism* – if the content generated looks real or not – this is highly subjective, and depends on the type of game art as some games adopt a cartoony look, e.g. *World of Warcraft*; (ii) *scale* – if it appears to be the correct size – the size of a virtual object depends on its representation in the game world and the graphics engine of the game must be taken in account for this analysis, e.g. a pebble and a boulder can be represented by the same content, just at a different scale; (iii) *variation* – whether the technique can generate outputs that cover the large range of variations encountered in nature or not – for example, a terrain generator can output all types of geographical features, or at least enough different types to cover the needs of the game; (iv) *input* – what is the necessary input data for the algorithm to produce an output, and what input can produce good results; (v) *efficiency* – both the time needed to create the output and the algorithm complexity; (vi) *control* – if the user can alter the results by changing some parameters, to what degree can the results be changed and whether the control is intuitive or not; and (vii) *real-time* – if the technique can be used to produce content when it is needed or has to create it beforehand – some typess of content take considerable time to be correctly generated and can only be used in places where it can be precomputed.

Togelius et al. [16] propose some distinctions for search-based PCG: (i) online versus offline, (ii) necessary versus optional content, (iii) random seeds versus parameter vectors, (iv) stochastic versus deterministic generation, and (v) constructive versus generate-and-test. They note though, that more distinctions are necessary to classify the PCG techniques. Some of their taxonomy can be used for techniques that do not use a search-based approach. Here, based on their definitions, we show how they can be applied to non-search-based PCG: (i) *online versus offline* – their definition is the same as that of Kelly and McCabe's *real-time* criterion; (ii) *necessary versus optional content* – a distinction of the type of content generated, where necessary means that the content is needed in order to progress in the game, and optional content can be skipped by the player – the necessary content must be correct or the game becomes unplayable, but the optional content can be generated incorrectly, since the player can ignore the wrong content; (iii) *random seeds versus parameter vectors* – this distinction regards the input data of the algorithms – some only receive a seed value for a random number generator, while others can receive more values that control the results – it is somewhat related to Kelly and McCabe's *control* criterion, in the sense that the more parameters available, the finer the control offered to the user and their *input* criterion that concerns the minimum necessary input; (iv) *stochastic versus deterministic generation* – whether the algorithm produces the same output for a given input or the output changes; and (v) *constructive versus generate-and-test* – this distinction regards the stop condition of the algorithm, namely if it creates a result and stops (constructive) or if it performs tests on the result to check for fitness and only returns when the result has the necessary properties.

Hendrikx et al. [5] create classes of game content that can be produced by PCG and introduce a taxonomy for PCG techniques. The classes of game content they made are as follows: *game bits*, which encompasses the following content types: textures, sound, vegetation, buildings, behavior, fire, water, stone and clouds – these elements are rarely displayed by themselves, being integrated to other elements to create a believable scene; *game space*, composed of indoor and outdoor maps, and bodies of water and other map features– the maps present the players with a space to perform their actions, and the water bodies and other map features are generally used as barriers to limit the playing area; *game systems* are behavior models that represent believable interaction between elements in the game, and are composed of ecosystems, road networks, urban environments and entity behavior; *game scenarios*, are ways of describing the world in which the player is, in the form of puzzles, storyboards, the story or the levels of the game; *game design*, which represents the rules and goals of the game, and is composed of the system design and of the world design; and *derived content*, generated by the players, consisting of news, broadcasts and leaderboards. For each of these content classes, they cite examples of successful uses and discuss their results. As for the techniques taxonomy, the authors introduce: (i) pseudo-random number generators, (ii) generative grammars, (iii) image filtering, (iv) spatial algorithms, (v) modeling and simulating complex systems, and (vi) artificial intelligence. These classes of methods represent the most common methods used to create more complex procedural content generators. The classes are explained as: (i) *pseudo-random number generators*, used to mimic the seemingly random patterns of nature; (ii) *generative grammars*, which can be used to generate content guaranteed to be correct by the use of sets of expansion rules – among which, Lindenmayer-systems, split grammars, wall grammars and shape grammars; (iii) *image filtering*, techniques to transform images by applying simple operations, and can be used to remove noise and to detect edges among other operations – the more common operation types are binary morphology and convolution filters; (iv) *spatial algorithms* are techniques to generate content based on geometric manipulations of space – the authors cite tiling and layering, grid subdivision, fractals and Voronoi diagrams as examples; (v) *modeling and simulation of complex systems* can be used to create content that is too complex to be generated by mathematical equations – some of these techniques are cellular automata, tensor fields, and agent-based simulation; and (vi) *artificial intelligence* that tries to model problems mimicking what animals do, with some techniques such as genetic algorithms and artificial neural networks used to find good results in some algorithms.

3 Algorithms taxonomy

Based on the taxonomies of Hendrikx et al. [5], Kelly and McCabe [6], and Togelius et al. [16], we propose a taxonomy to unify and to expand their works. With this, we expect to help the research of PCG to move towards a comprehensive taxonomy of its techniques, which can help both research and development in the field.

We propose that the taxonomy should discriminate each technique against the following classes: (1) type of content generated; (2) base technique(s) used; (3) generation time; (4) input; (5) realism; (6) scale; (7) variation; (8) control; (9) correctness of result; (10) determinism; (11) stop condition; (12) algorithmic complexity; (13) execution time; and (14) memory consumption. Note that these categories are not listed in order of importance, and only the users of the techniques can determine what is important for their use cases. We now explain what we intend to achieve with each

classification and how to classify the content in each one of them.

3.1 Type of content generated

The reason to classify on the type of content generated is to help the user find techniques to achieve the necessary goals.

This is a simple classification achieved by simply checking the proposed content classes against the original classification by Hendrikx et al. [5].

3.2 Base technique(s) used

The main reason to discriminate the used base technique(s) is to help researchers find them based on technical aspects. This can facilitate the study of algorithmically similar solutions. Another important reason is that the base algorithms used can give the user a rough idea of the behavior of the content generator.

Again, to obtain this classification is simply a matter of fitting the base technique used against the original classification by Hendrikx et al. [5].

3.3 Generation time

This category is based on Kelly and McCabe's *real-time* and Togelius et al. *online vs offline* classifications. This is an important distinction to be made, given that some techniques cannot be used to generate content during the execution of a game, since the player will not wait too long to be able to use the content.

The best way to classify this is by using the execution time metric (explained in Subsection 3.13). This can give an estimate of the possibility of use at run-time, but the final answer to this category depends on each use case, since the developers alone can determine if there is enough time to run the generator.

3.4 Input

The analysis of the inputs needed to generate the content can be used to choose the best generator for a given need. There are cases in which the user only needs some content and its results do not need to be controlled and others in which fine control is a must. This analysis is related to the control metric (explained in Subsection 3.8), in the sense that more input parameters may lead to better control of the outputs.

To obtain this classification, an analysis of all input parameters, their domains and boundaries must be made. This is a complex task, but one that can help guarantee the correctness of the outputs (cf. Subsection 3.9).

3.5 Realism

This is a highly subjective classification, as there is no final metric to define it other than examining the results and comparing them to the equivalents in nature. It is an important metric, though, given that realism is one of the most sought element in games nowadays.

With this classification, we should prevent the comparison of the rendering engine used instead of the content. To this end, we should use good rendering engines, and guarantee that no detail of the generated content is lost in the rendering process. A subpar rendering engine can cause the content to look unreal even if it could look better with proper rendering techniques.

3.6 Scale

Scale is a way of measuring the content in order to see if it has the same size as similar content in nature. It is a measure that can be used to prove that the content does not look real, although it is not sufficient to prove otherwise. Kelly and McCabe [6] define scale as a measure of the maximum size that the generated content can have. In their case, cities should have the "feel" of a city, covering large areas of the map with road networks and buildings, or else it looks like a small town or village. Outside of the context of cities, the concept of scale is still applicable. Some types of content, such as vegetation or height maps that can exhibit recurring, fractal-like patterns and these patterns can be used to generate contents of different sizes. In general, scale should be a measure of the proportions of the generated content in order to determine if it displays the same size properties as their natural counterparts.

To measure scale, we again depend on the rendering engine, as scaling operations can change the resulting look of the content. Because of this, the same rules of Subsection 3.5 should be used.

3.7 Variation

Kelly and McCabe [6] define the variation metric as the amount of heterogeneity possible in one instance of the generated content. This is an important metric in the generation of cities, but there are many types of content, such as noise, vegetation, clouds and indoor maps where, given one instance of generated content, little variation should occur. In these cases, variation is good only among different instances of content.

Given that one of the main goals of PCG is the ability to create large quantities of related content cheaply and fast, and that our goal is to propose a taxonomy that can be used for different types of content, we propose that variation should be a measure of the possible outputs by the method. The variation of the content is a good metric, since we sometimes need large differences between the content generated, and sometimes, the contents produced have to be similar to one another. As such, users can base their choice of generator based on the outcome desired.

The analysis of output variation depends on the class of content generated, with some classes, such as pseudo-random number generators lending themselves to simple statistical analysis, while others, such as cities generators, have to be analyzed manually.

3.8 Control

This category refers to the user's ability to influence the outcome of the generation process. It is desirable for *offline* generators to give users a great amount of control, since they are normally used to help artists create the game content. As for *online* generators, these controls have to be manipulated by the user, and should be simpler. In both cases, a measure of the intuitiveness of the controls is useful.

The degree of control must be measured after the input analysis is performed. This is because the input domains must be fully analyzed in order to understand what changes they make to the results. An analysis of intuitiveness should be performed from a usability point of view, although an experienced user can give feedback on the "feel" of the controls that can be used to achieve a rough estimate of control intuitiveness.

3.9 Correctness of result

Correctness of result is a guarantee that the output of the generator will always be right. It is important to define if the content can be used as necessary or only as optional. This classification is important to allow users to know beforehand if the generator meets the necessary guarantee of correctness. Note that not all content must be correct, as long as the player can choose to ignore it.

Again, this measure depends on the input analysis. This analysis is very complex, and normally needs to be performed mathematically, since we cannot generally test all the possible inputs and outputs of the generator.

3.10 Determinism

This class differentiates between techniques that, given the same input, always produce the same results and those that do not. It is important for generating content such as textures, in which the artists might want the result to be the same for all users. On the other hand, the randomness of non-deterministic algorithms is important for content – like maps, where it is interesting for the player to always have a new map to play with.

This measure also requires the analysis of the inputs and outputs of the generator, although it is far easier to prove than the correctness of result. Togelius et al. [16], in proposing this classification, exclude the random seeds for the pseudo-random number generators from the analysis, as taking it into account would mean that all techniques are deterministic.

3.11 Stop condition

The stop condition of a generator regards how the algorithm behaves at the end of the processing phase. It can simply output the generated content (*constructive*) or test the result for fitness and only output a result when the test conditions are met (*generate-and-test*). Clearly, from this behavior, we can see that the second method might take a long time, or never stop. This makes that class of algorithms only suitable for *offline* use.

This classification is simple to obtain by just reading the code and by identifying any generate and test loop.

3.12 Algorithmic complexity

Algorithmic complexity, with the big-*O* notation, can be a good reference of how much processing power the generator needs in the worst case scenario [7]. This helps the user understand the behavior of the algorithm when changing, e.g. the size of the map generated or the number of steps in a cellular automata technique. This metric, allied with the execution time, can be used to choose algorithms guaranteed to finish in a necessary time frame.

To determine the algorithmic complexity, we simply analyze the complexity of the algorithm.

3.13 Execution time

Especially when selecting an *online* generator, the execution time is of vital importance, as the player will not wait indefinitely for the game to process. This metric, allied with the algorithmic complexity, can provide developers with enough data to choose the best algorithm for their use cases.

In Figure 1, we show a simple way of measuring the execution time of a technique. C++ was chosen as the sample language as it is the *lingua franca* of the game industry.

```
using namespace std::chrono;
auto begin = high_resolution_clock::now();
// call the method to be measured...
auto end = high_resolution_clock::now();
auto duration =
        duration_cast<nanoseconds>(end - begin);
auto elapsed = duration.count();
```

Figure 1: C++ code for measuring the elapsed time of a piece of code.

3.14 Memory consumption

Memory consumption is not always a concern for PCG techniques, but they should be taken into account, especially for generators intended for use in mobile devices, in which memory is limited. This metric should be used particularly when we use PCG as a means of trading memory consumption for processing power, in order to see if the memory ratio really justifies the trade-off.

To obtain the memory consumption, a profiler can be used, or we can simply analyze the algorithm to determine its memory consumption in the worst case scenario.

4 Conclusion

With this paper, we expect to help the field of PCG move towards a more complete taxonomy for its techniques. We did not propose this taxonomy as a definitive one, but as a unification of already proposed topical taxonomy that, when unified, complement themselves into a more comprehensive one.

As future work, we see two actionable points deriving from this work. The first is the improvement of this taxonomy, towards a future definitive version. And the second is the application of this classification to the most commonly used techniques in PCG.

Should this taxonomy be accepted by the community, we believe that it would be useful to expand the Procedural Content Generation Wiki [1] with both known and future implementations of PCG techniques and its classification. This classification work could be useful in helping users of PCG algorithms to choose the best option for their use cases among the available options and in bringing better understanding of the particularities of the techniques. This would prove helpful to future works on expanding and adding PCG techniques for new types of content.

Acknowledgment

The authors would like to thank CAPES for the support to this work.

References

[1] Procedural Content Generation Wiki, 2016.

[2] F. Bevilacqua. Ferramenta para geração em tempo real de bordas de mapas virtuais pseudo-infinitos para jogos 3D, 2008.

[3] D. M. D. Carli. Geração procedural de cenários 3d de cânions com foco em jogos digitais, 2012.

[4] P. M. Duarte. Geração Procedural de Cenários Orientada a Objetivos, 2012.

[5] M. Hendrikx, S. Meijer, J. Van Der Velden, and A. Iosup. Procedural content generation for games: A survey. *ACM Trans. Multimedia Comput. Commun. Appl.*, 9(1):1:1–1:22, Feb. 2013.

[6] G. Kelly and H. McCabe. A survey of procedural techniques for city generation. *ITB Journal*, pages 87–130, 2006.

[7] D. E. Knuth. Big Omicron and Big Omega and Big Theta. *SIGACT News*, 8(2):18–24, Apr. 1976.

[8] G. d. O. B. Leite and E. S. D. Lima. Geração Procedural de Mapas para Jogos 2D. *Proceedings of SBGames 2015*, pages 244–247, 2015.

[9] B. B. Mandelbrot. *The Fractal Geometry of Nature*. W. H. Freeman and Company, 1982.

[10] F. M. Miranda, C. S. Cordeiro, and L. Chaimowicz. Um Sistema para Geração Procedural de Terrenos Pseudo-Infinitos em Tempo-Real Utilizando GPU e CPU. *VIII Brazilian Symposium on Games and Digital Entertainment*, pages 113–116, 2009.

[11] A. B. C. Pereira. Um sistema fuzzy para geração de tarefas de ensino de leitura e escrita em um jogo digital, 2012.

[12] M. C. Silva, F. M. G. França, and G. R. E. Cabral. Construindo Trilhas Sonoras Dinâmicas Em Jogos Utilizando Sistemas Fuzzy. *Proceedings of the SBGames 2014*, pages 974–977, 2014.

[13] R. M. Smelik, T. Tutenel, R. Bidarra, and B. Benes. A survey on procedural modelling for virtual worlds. *Computer Graphics Forum*, 33(6):31–50, 2014.

[14] R. Spouses. Wives of Rockstar San Diego employees have collected themselves, 07 2010.

[15] G. Stiny and J. Gips. Shape grammars and the generative specification of painting and sculpture. In *Segmentation of Buildings for 3DGeneralisation. In: Proceedings of the Workshop on generalisation and multiple representation , Leicester*, 1971.

[16] J. Togelius, G. Yannakakis, K. Stanley, and C. Browne. Search-based procedural content generation: A taxonomy and survey. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3(3):172–186, Sept 2011.

[17] L. Villapaz. 'GTA 5' costs $265 million to develop and market, making it the most expensive video game ever produced: Report, 2013.