Procedural Content Generation for Game Props? A Study on the Effects on User Experience

OLIVER KORN, Offenburg University
MICHAEL BLATZ, ADRIAN REES, and JAKOB SCHAAL, KORION GmbH
VALENTIN SCHWIND, University of Stuttgart
DANIEL GÖRLICH, SRH Hochschule Heidelberg

This work demonstrates the potentials of procedural content generation (PCG) for games, focusing on the generation of specific graphic props (reefs) in an explorer game.

We briefly portray the state-of-the-art of PCG and compare various methods to create random patterns at runtime. Taking a step towards the game industry, we describe an actual game production and provide a detailed pseudocode implementation showing how Perlin or Simplex noise can be used efficiently.

In a comparative study, we investigate two alternative implementations of a decisive game prop: once created traditionally by artists and once generated by procedural algorithms. 41 test subjects played both implementations. The analysis shows that PCG can create a user experience that is significantly more realistic and at the same time perceived as more aesthetically pleasing. In addition, the ever-changing nature of the procedurally generated environments is preferred with high significance, especially by players aged 45 and above.

Categories and Subject Descriptors: I.6.8 [**Types of Simulation**]: Gaming; Animation; K.8.0 [**General**]: Games; D.2.9 [**Management**]: Productivity

General Terms: Design, Human Factors, Performance, Management

Additional Key Words and Phrases: Procedural content generation, game design, algorithms

ACM Reference Format:
Oliver Korn, Michael Blatz, Adrian Rees, Jakob Schaal, Valentin Schwind, and Daniel Görlich. 2017. Procedural content generation for game props? A study on the effects on user experience. Comput. Entertain. 15, 2, Article 1 (February 2017), 15 pages.
DOI: http://dx.doi.org/10.1145/2974026

INTRODUCTION AND MOTIVATION

In video games, graphics can be generated in different ways: on the one hand, there is the traditional creation through skilled artists; on the other hand, graphics can be generated by algorithms, i.e., procedurally. Thus, the programmers of the algorithm partially substitute the work previously done by artists. This procedural content generation (PCG) is getting more and more attention from the computer games industry. Especially when many varied iterations of similar elements (e.g., trees, walls, swords) are required, PCG is an interesting solution.

Authors' addresses: O. Korn, Offenburg University, Badstr. 24, 77652 Offenburg, Germany; email: oliver.korn@acm.org; M. Blatz, A. Rees, and J. Schaal, KORION GmbH, Moempelgardstr. 16, 71640 Ludwigsburg, Germany; emails: {michael.blatz, adrian.rees, info}@korion.de; V. Schwind, University of Stuttgart, VIS, Pfaffenwaldring 5a, 70569 Stuttgart, Germany; email: valentin.schwind@vis.uni-stuttgart.de; D. Goerlich, SRH Hochschule Heidelberg, Ludwig-Guttmann-Str. 6, 69123 Heidelberg, Germany; email: daniel.goerlich@hochschule-heidelberg.de.

DOI: http://dx.doi.org/10.1145/2974026

However, does PCG reach the quality of the artists? To address this question, we need to look at the development of computer graphics. Just a few years back, the look of games was mainly limited by technical possibilities. Today games like *Minecraft* use a simple "pixel look" deliberately as a stylistic device [Persson 2011]. Therefore, the current spectrum in game graphics reaches from pixel graphics from the era of the Commodore 64 to high-definition 3D graphics [Jones 2013].

However, even the best graphics are no guarantee for appreciation. The *Call of Duty* series undoubtedly is a flagship of the game industry, produced with great financial and artistic effort. Still it is notoriously criticized for tube-like level maps with little freedom of choice. Such restrictions are typical flaws of manually generated levels, making this series a representative of state-of-the-art manual content generation. However, the richness of detail and the cinematic quality of some dramatic moments in the narration can only be achieved through a high level of control, at the cost of the players' freedom [Yannakakis and Togelius 2011].

When aiming at such cinematic experiences, more and more cost and effort are required for the development of high-quality assets. A way to save resources is not using the artists' capacity for the production of similar assets. PCG, graphics can be modified with a variety of randomized parameters at runtime, resulting in completely different looks of generic assets like rocks or trees. As a logical next step, whole buildings and eventually whole game worlds can be generated procedurally [Hosking 2013].

The successful game series *Borderlands* shows that PCG can not only be used to generate diversity in graphics, but also in gameplay: a countless number of different weapons, grenades and shields is randomly generated, creating the illusion of technical variety und fueling the players' passion for loot, resulting in a vivid game experience.

A holistic procedural approach is taken by *Minecraft*, which not only generates game elements but the whole game world procedurally [Persson 2011]. Generation takes place at runtime, making it possible to constantly discover new areas of the world. In combination with the many ways to manipulate the game world, this generates a very high replay value [Belinkie 2010]. Thus, the procedural approach is corresponding well to a paradigm in game development: "the game enables the experience, but it is not the experience" [Schell 2015].

Due to the many possibilities of PCG, it is difficult to create a universal formula for the frequency and type of its implementation. Most games are either created with PCG elements or without them—thus it usually is impossible to tell how the game experience would have been different if another type of content generation had been used. In order to create valid findings on the effects of PCG, a control condition is required—a game that is identical in all aspects but one: using or not using PCG.

In the case study presented in this work, we used the production of the documentary game *Conquest of the Seven Seas* [Korn et al. 2015b] by the game studio KORION to do just that: one implementation is based on PCG while the other one is using the traditional artist approach (Figure 1). Both alternatives were played and judged by 41 players. Based on the study, we examine the acceptance of PCG in different age groups and validate if claims regarding the positive effect of PCG on the replay value are substantiated.

BACKGROUND

For a better understanding of how PCG can be utilized in games, we provide a short overview of the most influential methods to create procedural content.

Algorithms for generating procedural graphics are frequent in computer games; yet there are others focusing on artificial intelligence (e.g., AI Director) or changes of state (e.g., Markov chains). Thus, the implementation of specific PCG methods strongly depends on the particular use case.

Choosing the appropriate PCG method

The version without PCG featured nine manually generated reefs. Artists designed them based on pictures of the Australian Great Barrier Reef. The corresponding PCG implementation had to meet two requirements: respect the hardware limitations of mobile devices and produce realistic results when compared to photographs of actual reefs.

This scenario combined the typical limitations of PCG: shortage on resources (see Introduction) and achieving the highest possible graphical quality (see State of the Art). As the overall game had a realistic look consisting of very detailed sprites, the reefs had to be designed at an equal level of detail to match the appearance. With these demands in mind, the selection could be narrowed down to two methods of PCG: self-avoiding fractals and Perlin noise. Both methods are suited to produce structures similar to reefs. Finally, Perlin noise was selected for two reasons:

—The two-dimensional Perlin noise demands less computing effort.
—The generated noise image already offers a suitable structure making it possible to only draw the desired area and blending the remaining greyscales with the surroundings through tone addition. With self-avoiding fractals, the final structure would have to be calculated separately.

The developers from KORION implemented the procedural generation in *GameMaker Studio* by YoYo Games. This IDE allows fast prototyping and offers basic drawing operations as well as an HTML 5 export.

The pseudocode in Figure 7 shows an implementation of Perlin noise for the reef structures. The algorithm consists of two parts: initialization and calculation. First, the variables are set up to define the datatype, as the flooring of integer value calculations is required. The frequency of supporting discrete gradient points in both directions is doubled with each octave; the amplitude is multiplied with the persistence value. Typically, five octaves are iterated.

At the start of each octave, the discrete gradients are randomly chosen between -amplitude and +amplitude. Then every sample value is calculated in three steps:

(1) Calculate weight for each neighboring discrete gradient point;
(2) Bilinear interpolate these values (w00. . .w11) to the final result (for this sample at this octave) by the Blend Function;
(3) Each interpolation is added to the final Perlin noise.

Due to the grey values being adjustable in Perlin noise, it was possible to specify exactly, which values should be drawn, so, all values that were black or in dark grey were assigned an alpha value of 0, excluding them from being drawn. The resulting noise is normalized and cut off below 0.2. Removing this data resulted in the sharp outline around the remaining values, generating the reef's distinctive structure. This structure is a result of the remaining values between 0.2 and 1.0, which are mapped to grayscale color. Although Perlin noise typically is used for height maps with very soft edges, this straightforward approach matches the natural appearance in the contours of reefs very well.

A parametrized example for the function call would be: `perlin2D(5, 32, 32, 1, 0.5, 1024, 1024);`

We will shortly explain the parameters in this example to ease re-implementations based on the algorithm presented in Figure 7:

—*octaves [3-8]:* The number of octaves. A higher number produces a higher quality noise at the cost of performance. The amount of discrete gradients doubles with each octave, so performance costs grow exponentially. Normally, a value of 5 is sufficient;

values above 8 do not produce noticeable differences but cost a lot more processing time. Values below 3 produce very low quality noises.
—*startFrequencyX/Y [5-64]:* Higher numbers produce higher fluctuations in the resulting noise.
—*startAmplitude [0.1-1.0]:* This parameter's effect depends on the use of the noise. Higher values result in higher values in the generated noise and vice-versa. If the noise is normalized after generation, the value has no effect as long as it is above or equal to 0.1.
—*persistence [0.1-0.9]:* This parameter changes the amplitude of the octaves >1. A value of 0.5 is a good starting value. Values below 0.1 give octaves >1 too less value while values above 0.9 make them nearly as strong as the first octave.
—*sampleWidth/Height [64-4k_or_more]:* The width and height of the produced image/array/noise. Sets the required size of the output.

Note that these parameters are interrelated. For example, a higher amount of octaves means that the persistence has to be increased accordingly—otherwise higher octaves would become invisible.

A timer called at the beginning of the generation, showed that level generation (result in Figure 6, bottom) usually took less than one second. This shows that Perlin noise indeed allows a resource saving implementation.

As described in State of the Art, the degree to which PCG is applied in games can easily exceed the implementation shown in this case study. For example, even in the game presented here, clouds still consist of a number of handmade sprites, while they could have easily been procedurally generated with methods like Perlin noise. However, for the purpose of the study, the procedural implementation of just a single game element was essential.

STUDY

In this section, we describe a comparative study, laid out to determine the effects of the two alternative reef implementations (with and without PCG) on the players' experience.

Expected Results

As explained above, PCG is not limited to just affecting the look of a game—it can also affect the gameplay. While visual variety already reduces the frustration generated by redundant designs, PCG that affects the gameplay potentially increases replay value. In the selected game, the reefs differ in every map, constantly demanding tactical planning and flexible ship maneuvers. Thus, the reefs have a strong impact on the gameplay. However, while the manually generated reefs differ only in number, appearance, and position, the procedurally generated ones additionally differ in shape.

Through this constant variation of the map, players were to be motivated to continuously risk new battles; even lost battles should increase motivation as there is an element of chance—similar to gambling, but less intense. The prospect of more advantageous wind conditions and reef positions in the next battle are well suited to increase the replay value. Our hypothesis was that players would recognize and favor the greater variation of the PCG approach.

Population and Setup

Forty-one test subjects participated in the study. Twenty-six were aged 18–24, eight were aged 25–29, and seven were aged 45 and above. To determine if PCG has a different impact on middle-aged persons, we intentionally included test subjects older than 45. For the answers to remain as unbiased as possible, we did not indicate that

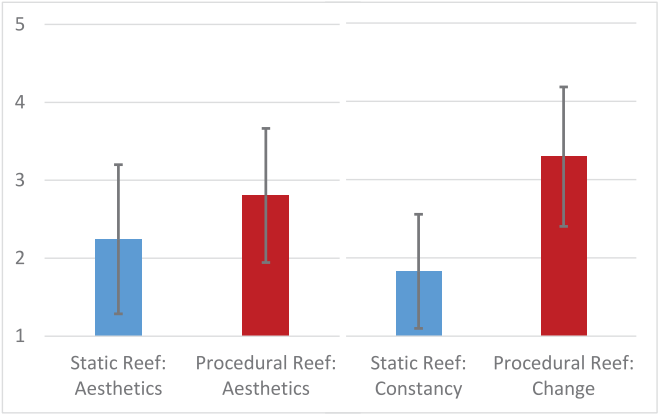

Fig. 9. Perceived aesthetic quality (columns 1 and 2) and preference with the explicit mentioning of constancy and change (columns 3 and 4) with standard deviations.

with $\bar{x} = 1.8$ (SD $= 0.7$) while the procedural reefs' appeal with respect to a changing look was assessed with $\bar{x} = 3.3$ (SD $= 0.9$). This difference is highly significant with $p < 0.01$ and even $p < 0.0000001$.

Discussion

It is interesting that 85% of the users favor PCG in the last question, as the differences when analyzing perceived realism and aesthetics were significant but less obvious: although about 60% of the players preferred the PCG-reefs, 40% of them still preferred the manually generated ones. We have two possible explanations for this: first, the changing look might be the most attractive element of the PCG-reefs—so when directly addressed, this advantage is more prominent; second, the words "constancy" versus "change" might create certain semantic priming effects, for example, a "modern" person might be more inclined to embrace change. Thus, the strong effect might also be attributable to a bias in the users' self-perception.

However, if such a bias existed, our data indicate that it is restricted to gamers aged 45 and older: there is both a negative correlation between age and the preference for static reefs ($r = -0.32$) and a positive correlation between age and the preference for procedurally generated reefs ($r = 0.28$). When comparing the affinity for a game experience with more variety by procedural reefs (Figure 10), the 26 players aged 18-25 embraced this change more ($\bar{x} = 3.3$, SD $= 0.8$) than the eight players aged 25-44 ($\bar{x} = 2.8$, SD $= 1.1$) – but the seven players aged 45 and above appreciated change most ($\bar{x} = 4.0$, SD $= 0.0$).

Even when comparing the players aged below and above 45 directly, this difference is highly significant ($p < 0.00001$). The stereotype intuition ("PCG creates experiences for young gamers") is not supported. It is the gamers aged 45 and above, who enjoy the effects of PCG most. This may be due to their advantage in life experience: they might prefer games mirroring the complexity of life—or they simply enjoy being surprised.

CONCLUSION

In this work, we first summarized the historical background of procedural algorithms in games. Then we described their use in various games and presented established methods to categorize and structure the use of PCG in games.

Based on the history of PCG, there are numerous ways to incorporate procedural algorithms in computer games. We showed that Perlin noise is an especially effective

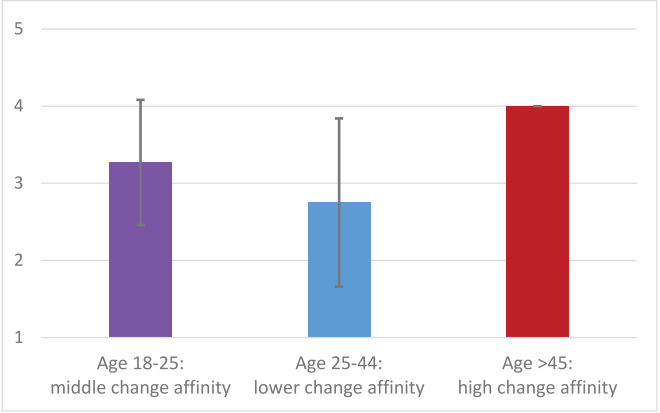

Fig. 10. Age groups and their affinity to change with standard deviations.

method when processing resources are limited. Within an ongoing game production, we developed an algorithm for generating reefs procedurally. We provided a detailed pseudocode implementation showing how Perlin or Simplex noise can be used efficiently.

Based on this procedural implementation and the traditional approach with manual designs by artists, we evaluated the effects of PCG on the user experience. The study shows that in comparable structures (here, manually versus procedurally generated reefs in exactly the same game) the use of PCG can be much more than a way to save money in production: it can create an experience that users perceive as both significantly more realistic and significantly more aesthetically pleasing.

Furthermore, users perceive the changing nature of the game environments as a great advantage. Changing environments are (highly significantly) preferred to traditional static environments. Interestingly, players aged above 45 enjoy this increased variation even more than the younger ones.

FUTURE WORK

The work presented here is a comparative study on the effects of PCG. To validate the findings, similar studies are required. These could be similar in scope, i.e., focusing on the effects of graphical elements on the gameplay. Ultimately, the scope should be broadened to examine the use of procedurally generated quests, characters or even artificial intelligence behaviors. However, such studies are difficult to implement: if actual games are to be used, the evaluation can only take place in the late phase of a game production, where the developers can still create alternative versions (at least for playable sub-aspects of the overall game design).

REFERENCES

Matthew Belinkie. 2010. What makes Minecraft so addictive? *Overthinking It* (November 2010).

Geoff Boeing. 2016. Visual analysis of nonlinear dynamical systems: Chaos, fractals, self-similarity and the limits of prediction. *Systems* 4, 4 (November 2016), 37. DOI: https://doi.org/10.3390/systems4040037

Mihály Csíkszentmihályi. 1975. *Beyond Boredom and Anxiety*, San Francisco, USA: Jossey-Bass Publishers.

Mihály Csíkszentmihályi, Sami Abuhamdeh, and Jeanne Nakamura. 2005. Flow. In *Handbook of Competence and Motivation.* New York, NY, USA: Guilford Press, 598–608.

Markus Funk, Oliver Korn, and Albrecht Schmidt. 2014. An augmented workplace for enabling user-defined tangibles. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI'14)*. ACM, New York, NY, USA. DOI: https://doi.org/10.1145/2559206.2581142