A Very Short History of Dynamic and Procedural Content Generation

Chapter · April 2017

DOI: 10.1007/978-3-319-53088-8_1

CITATION
1

READS
399

2 authors:

Michael Blatz
Offenburg University of Applied Sciences
4 PUBLICATIONS **7** CITATIONS
SEE PROFILE

Oliver Korn
Offenburg University of Applied Sciences
61 PUBLICATIONS **469** CITATIONS
SEE PROFILE

A Very Short History of Dynamic and Procedural Content Generation

Michael Blatz and Oliver Korn

Abstract This chapter portrays the historical and mathematical background of dynamic and procedural content generation (PCG). We portray and compare various PCG methods and analyze which mathematical approach is suited for typical applications in game design. In the next step, a structural overview of games applying PCG as well as types of PCG is presented. As abundant PCG content can be overwhelming, we discuss context-aware adaptation as a way to adapt the challenge to individual players' requirements. Finally, we take a brief look at the future of PCG.

1 Introduction and Motivation

Game contents like terrain, characters, items and even story elements can be developed in two radically different ways: either by manual content creation: Artists develop graphics, writers create stories, etc., or by procedural content generation (PCG). With PCG, algorithms generate the content procedurally, either before a level starts or even continuously during runtime.

However, procedural structures have been there long before the invention of computers and even long before humankind. Many elemental and natural structures both on very large scales (galaxies, island distribution, coastlines, etc.) and on smaller scales (reef structures, some vegetables, bacteria growth, etc.) show that the

M. Blatz
KORION GmbH, Ludwigsburg, Germany
e-mail: michael.blatz@korion.de

O. Korn (✉)
Offenburg University, Offenburg, Germany
e-mail: oliver.korn@acm.org

O. Korn and N. Lee (eds.), *Game Dynamics*,
DOI 10.1007/978-3-319-53088-8_1

Fig. 1 Romanesco is a good example of how iterative processes can create aesthetically pleasing self-similar structures

structures generated by the iteration of simple rules are in fact universal. Even a common vegetable like Romanesco shows these "self-similar" structures, and thus the graphical effects of PCG at work (Fig. 1).

While PCG has always played a role in game development, it is especially influential today. Platforms like Steam, App Store or Google Play allow an easier distribution of games without publishers and distribution media. In consequence, the video game sector, which was already very industrialized, is reliving its early days: Independent developers with small teams create thousands of creative applications. These development teams typically have neither the number of skilled employees nor the budget to create manually substantial amounts of game content. Thus, either the "indies" create very small games—or they delve into PCG. Applying PCG usually results in saving artists' capacity by modifying or even creating assets with the use of algorithms, typically using a variety of randomized parameters. On the graphical level, this results in different looks of rocks, trees, weapons or even whole buildings—eventually, the whole game world could be generated procedurally (Hosking 2013).

A large scale of potential game design patterns accompanies these possibilities. Only a few years back, it was primarily the technical possibilities, which limited how games looked and felt like. Today the simple "pixel look" of *Minecraft* (Persson 2011) is used deliberately as a stylistic device and exists independently besides highly detailed, almost photo-realistic graphics. The current spectrum in computer game designs reaches from pixel graphics as in the era of the Commodore 64 (Jones 2013) to high-definition, 3D graphics. The spectrum of stories reaches from simple jump and run ("rescue the princess") to role-playing games offering

fascinating and complex experiences even for players familiar with top-notch movies and series.

However, even fascinating stories or excellent graphics are no guarantee for appreciation. A good example is the *Call of Duty* series, which undoubtedly is produced with great financial and artistic effort. Nevertheless, in player forums the games are often criticized for their tube-like linear level maps, which leave players little freedom of choice. Such restrictions are characteristic flaws of manually generated stories and levels, making *Call of Duty* a typical representative of state-of-the-art manual content generation. At the same time the critics should realize that the richness of detail and the cinematic quality of some of the "epic" moments in this series can only be achieved through a high level of control, inevitably at the cost of the players' freedom (Yannakakis and Togelius 2011). When aiming at such cinematic experiences, much budget is required for the development of high-quality assets. Therefore, the interest in PCG grows even in the traditional game industry.

In Sect. 2 of this chapter, we portray the historical and mathematical background of generating content dynamically and procedurally and compare various methods. We mainly focus on graphical aspects, the area where PCG originated. In Sect. 3, we present a structural overview of games applying PCG as well as different types of PCG. We discuss a method to adapt the challenge to individual players' requirements. Finally, we take a brief look at the future of PCG.

2 PCG Methods

For a better understanding of how PCG can be utilized in games, we portray the most influential methods to create procedural content. Algorithms for procedural content are often applied in computer games, yet there are others mainly focusing on artificial intelligence (e.g., AI Director) or changes of state (e.g., Markov chains). Therefore, the particular use case defines the implementation of specific PCG methods.

The most common use of PCG is probably generating terrain or landscapes (Shaker et al. 2016). However, the underlying algorithms can be adapted for other use cases. Typically, noise functions are used to create the content, but principally other solutions like fractal-based algorithms could also generate the required self-similar structures. To provide an overview of the algorithmic potentials, we briefly describe common PCG methods and their historical background.

2.1 Brown Fractals and Mandelbrot Sets

The Brownian motion is an important starting point in the history of procedural algorithms. Its discovery goes back to the botanist Robert Brown (1773–1858), who

observed the random motion of pollen grain particles within water. This motion is the result of non-visible molecules in liquids and gasses that move with random velocities, in random directions, colliding with the pollen grains. As a natural phenomenon, it is one of the simplest observable stochastic processes.

Almost a century later, the mathematician Norbert Wiener (1894–1964) was inspired by this phenomenon and together with Godfrey Harold "G. H." Hardy tried to map it in an algorithm. Later the French physicist and Nobel-prize winner Jean Perrin (1870–1942) had the idea of describing the Brownian motion as continuous-time stochastic process by using non-differentiable curves (Mandelbrot 1982)—a method called "Wiener process" (visualized in Fig. 2) in honor of Wiener.

Benoit Mandelbrot (1924–2010) later frequently described the Wiener process as one of the keystones of his work. Mandelbrot used the Brownian motion to develop what he calls a "self-avoiding random walk" (Fig. 3). Clearly, this implementation is well suited to generate forms that are similar to land masses or jagged coastlines.

Another algorithmic approach is fractals, defined as a mathematical set that exhibits a repeating pattern that displays at every scale (Boeing 2016). From a geometrical point of view, fractals are self-similar structures (Fig. 4) based on recursive functions with potentially infinite depth. The very common "Julia sets" or "Mandelbrot sets" are such fractal structures, the latter one named after the discoverer Mandelbrot. It is visually striking, how several plants (Fig. 1) but also macroscopic structures like coastal lines of reefs incorporate structures similar to fractals. This phenomenon indicates that such natural structures can be algorithmically described using fractals. Mandelbrot even claimed that all forms of nature like trees, mountains, coastlines or clouds can only adequately be described using fractals (Ziegler 2013).

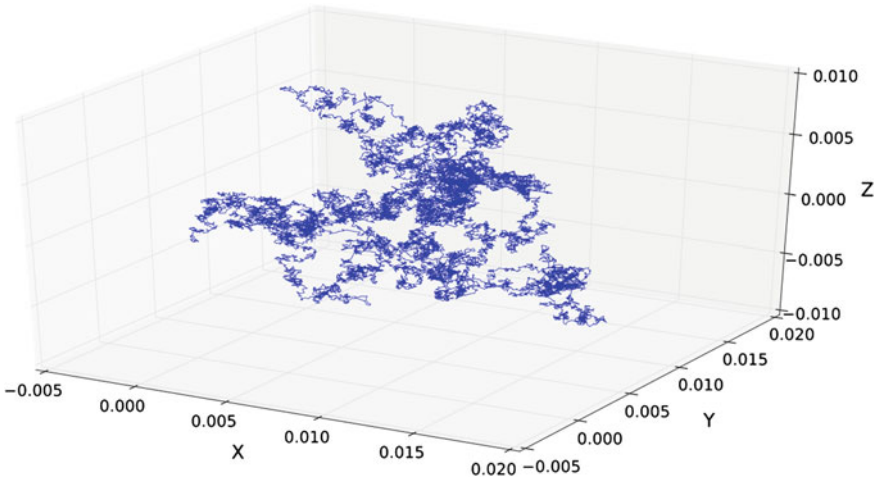


Fig. 2 Realization of a single Wiener process in 3D. Created by Shiyu Ji, CC BY-SA 4.0

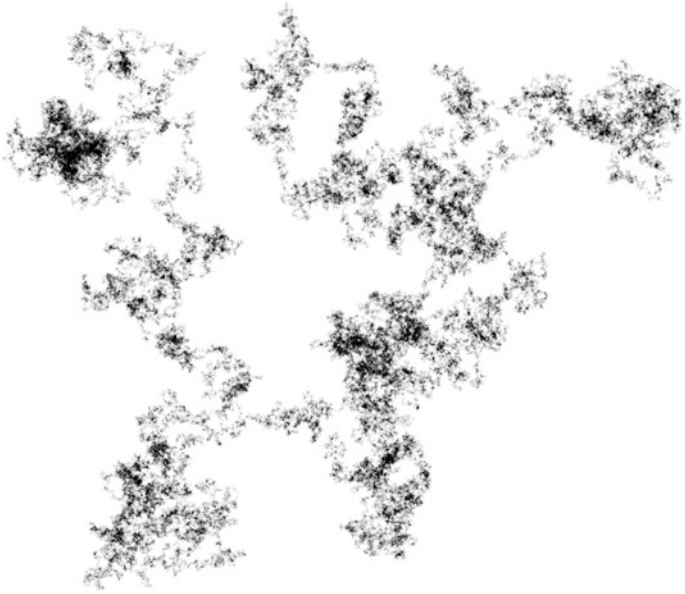


Fig. 3 Self-avoiding random walk in two dimensions. The *darker* the region, the more it has been visited. Created with MATLAB by Purpy Pupple, CC BY-SA 3.0

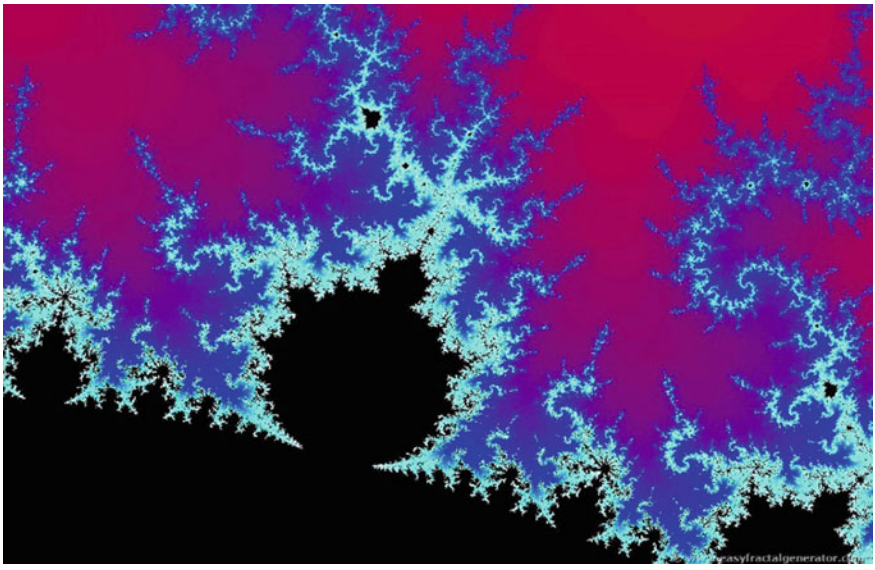


Fig. 4 Mandelbrot set probably is the most popular geometrical fractal. Its self-similar iterative structure is found in both small and large structures, e.g., vegetables and reefs. The image was generated with the easy fractal generator

The only technical restriction in generating fractal structures is processing time. Consequently, with increasing recursion depth, the processing duration grows quickly. This makes a real-time implementation of fractals for games very difficult.

2.2 Perlin Noise

Due to their recursive structure, the algorithms described above require considerable computing power. For practical purposes like game development (especially for mobile devices), alternatives that can describe natural forms with less computing effort are required. Gradient noise algorithms not based on recursions are such an alternative. A noise function is a mapping, which assigns a random value to every natural number. However, how can noise images be used to describe natural forms?

Ken Perlin addressed this question in the eighties. As he was frustrated with the pixelated look of graphics at that time, he searched for an "image synthesizer," generating "naturalistic visual complexity" (Perlin 1985). In 1997, he received the Academy Award for Technical Achievement for discovering that algorithm. Perlin defined noise as "a texturing primitive you can use to create a very wide variety of natural looking textures" and explained that "combining noise into various mathematical expressions produces procedural texture" (Perlin 1999). This texture can be applied to any object, so Perlin noise can be used in 2D as well as in 3D (Fig. 5).

Another advantage of gradient noise functions is that, even if the distribution is randomly generated, every configuration can be saved and reapplied so that the same conditions will always produce the same results. Perlin used frequency and amplitude to describe the function. The texture is a single-channel image (usually gray scale) where pixel values represent frequencies between −1 and +1. A high frequency (up to 1) means smaller details, resulting in many small points in a specific area, while a low frequency results in mostly large points. The noise function's amplitude describes the color values' "height" between completely black (0) and completely white (255). The implementation is described in a four-step process (Perlin 1999):

1. Given an input point
2. For each of its neighboring grid points: pick a "pseudo-random" gradient vector.
3. Compute linear function (dot product)
4. Take weighted sum, using ease curves.

Later a revised version called "Simplex Noise" was developed (Olano et al. 2003), as the first implementation only performed well in up to three dimensions. Simplex noise does not use a grid based on quads, but a grid with equal-sided triangles, reducing the number of neighboring points and therefore computing time. Contrary to Perlin noise with the complexity of $O(2n)$, simplex noise only requires $2(n^2)$, using only $n + 1$ vertices instead of $2n$. This is most noticeable when working in dimensions with $n > 4$.

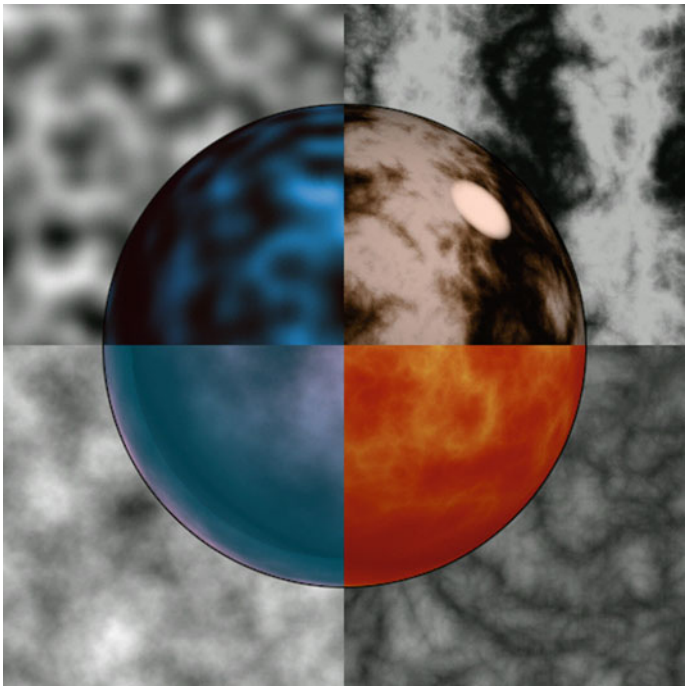


Fig. 5 Different versions of noise as described by Perlin

3 State-of-the-Art

We already introduced PCG as a versatile algorithmic tool to enrich computer games. In this section, we present how procedural contents have previously been used games. We also examine how the potentially abundant variety generated by PCG can be dealt with to meet individual user requirements.

3.1 Categorizing PCG in Games

In most cases, not the complete game, but only specific parts of it are created procedurally. While terrain (Génevaux et al. 2013; Togelius et al. 2011) and assets like vegetation are most common examples, almost all elements in a game can potentially be generated procedurally (Hosking 2013): characters, artificial intelligence or stories and quests. Even music can be generated procedurally. The game *Audiosurf* is a good example: It can potentially generate as many levels as there are music tracks in the world, as every level is generated based on a piece of music. Tunes and rhythm are used to not only generate the outlines of a level, but also to

place items according to the beats, creating a game with only a minimum of graphical effort.

How can this vast field of appliances be categorized and ordered? Hendrikx et al. (2013) proposed the following structure to differentiate between types and levels of procedurally generated content:

- Game Bits: e.g., textures, sound
- Game Space: the game world
- Game Systems: complex relations between game objects
- Game Scenarios: e.g., Story.

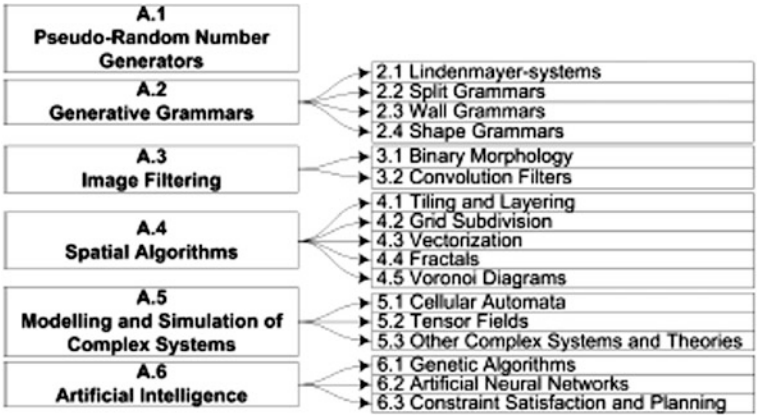
Using this differentiation, Hendrikx et al. looked at the application of PCG in 22 of the more influential video games of the last 35 years (Table 1).

We believe that this system is an approach well suited to describe PCG use in game. Especially, if it the content-oriented perspective is complemented by a

Table 1 Structural overview on the use of PCG in Games (Hendrikx et al. 2013)

Games	Release	Game bits	Game space	Game systems	Game scenarios
Borderlands	2009	X			
Diablo I	2000		X		
Diablo II	2008		X		X
Dwarf Fortress	2006		X	X	X
Elder Scrolls IV: Oblivion	2007	X			
Elder Scrolls V: Skyrim	2011				X
Elite	1984		X	X	X
EVE Online	2003	X	X		X
Façade	2005				X
FreeCiv and Civilization IV	2004		X		
Fuel	2009		X		
Gears of War 2	2008	X			
Left4Dead	2008				X
.kkrieger	2004	X			
Minecraft	2009		X	X	
Noctis	2002		X		
RoboBlitz	2006	X			
Realm of the Mad God	2010	X			
Rogue	1980		X		X
Spelunky	2008	X	X		
Torchlight	2009		X		
X-Com: UFO Defense	1994		X		

Table 2 Categorization of technical complexity of PCG use (Hendrikx et al. 2013)



technical perspective. Indeed, in a second step, Hendrikx et al. categorize PCG use according to different levels of technical complexity, beginning with pseudo-random number generators up to artificial intelligence (Table 2).

It would be interesting to investigate which technical PCG approaches were used in each game, e.g., in the games presented in Table 1. However, implementation details on games often remain unpublished, as the scientific documentation of algorithms and methods plays a minor role in the game industry. This situation is starting to change, as several of the other chapters in this book show.

3.2 Adjusting the Challenge When Using PCG

In the previous subsection, we showed a way to structure the potential abundance of procedural content. However, how can game designers prevent that this abundance overwhelms the players? How can the potentially endless variety of PCG be controlled and adjusted to the rest of the game world and the player requirements? In the field of artificial intelligence, Valve has drawn some attention with the series *Left 4 Dead*. Using the tool *AI Director*, they changed the dynamic creation of the game world depending on the course of the game. This was done in a covert way, e.g., by enabling passages that were previously blocked. In contrast to many competitors, they not only changed the behavior of the enemies, but also the game map according to the flow of the game, a method described by Thue and Bulitko (2012). Additionally, according to the calculated stress level of the player, enemy waves were adapted either to challenge the player, or to give him or her time to

relax. This context-aware adaptation can prevent that the potential richness of PCG overwhelms the players.

This approach is mirrored in the gamification of other domains: New assistive systems in production environments analyze a worker's behavior, trying to determine the flow state and adjust the challenge level (Korn et al. 2014, 2015a). For exergames (games for physiological training or sports exercises), similar methods of adjusting the challenge level have been described (Brach et al. 2012). Just like *Left 4 Dead*, these gamified real-world applications raise the challenge level to prevent boredom or lower it to prevent overextension.

In these real-world domains, sensors are required to evaluate the status of the user. In games, the properties of the user's avatar like position and health are completely transparent, so challenge adjustment mechanisms can be implemented with comparatively little effort. Ideally, this game-immanent information is complemented by physiological user data like the heart rate, which clearly indicates a player's level of arousal. By analyzing head movements and color fluctuations in the face, pulse can even be measured without physiological sensors (Balakrishnan et al. 2013)—which leads us to the future of PCG.

4 The Future of PCG

While PCG can be applied in various content types, it also benefits from new technological platforms: For mobile games, it is especially rewarding to use procedural and dynamic approaches. Mobile or web-based games are often developed in small-scale productions (Korn et al. 2015b), so the benefits of using PCG are highest. Furthermore, many successful mobile games like *Doodle Jump* or *Tiny Wings* incorporate both a simple gameplay and a high replay value—features well suited for the use of PCG.

As the very successful game series *Borderlands* exemplifies, PCG can even be used in AAA productions to generate diversity in the gameplay. In *Borderlands,* a countless number of different weapons are generated procedurally. This creates the illusion of almost unlimited technical variety, enriching the game experience. *Minecraft* takes a step beyond the procedural generation of game elements and creates the complete game world procedurally. The resulting freedom to constantly discover new areas of the world in combination with the expensive ways to manipulate the game world generates a very high replay value (Belinkie 2010). Thus, the procedural approach is corresponding well with a central paradigm of game development: "the game enables the experience, but it is not the experience" (Schell 2015).

In spite of these two highly successful examples, there still are only a few studios relying on PCG, especially with bigger productions. When evaluating the consequences of PCG use, productions with a budget of several millions dollars usually still demand a higher level of reliability that counteracts innovation, flexibility and the "openness" resulting from procedural approaches. We think that AAA

productions involving PCG will only become more frequent if the experimental factor concerning the rules of the game remains manageable. *Borderlands* has already cleared this hurdle: It neither changed the ground rules of the shooter genre nor let the procedural elements significantly alter the course and the rules of the game. Alternatively, the context-aware adaption mechanisms described in the previous subsection show effective ways to control the potential abundance of PCG in larger projects.

Nevertheless, the reluctance of larger studios regarding PCG currently puts independent (Indie) developers in the leading role of innovation in the game industry. Several of these smaller studios have shown the potential to create fascinating game worlds using PCG. In our view, PCG has not yet arrived at the peak of its potential. However, its importance is growing as the fascination of players for "endless" worlds and high replay values grow. Thus, the future of PCG in game design is mainly forged by the way it is applied: as a mere tool to create a variety of assets—or as a gameplay paradigm with a deep impact on user experience and replay value.

5 Conclusion

In this chapter, we portrayed the historical and mathematical background of procedural algorithms like fractals, Mandelbrot-sets and Perlin noise. A structural overview of games applying PCG as well as types of PCG was presented. We discussed ways to adapt the challenge to prevent that the potential richness and complexity of procedurally generated assets overwhelm individual players. Finally, we took a brief look at the future of PCG and explained why currently rather Indie developers than established game studios shape this future.

References

Balakrishnan, G., Durand, F., & Guttag, J. (2013). Detecting Pulse from Head Motions in Video (pp. 3430–3437). IEEE. 10.1109/CVPR.2013.440

Belinkie, M. (2010, November 11). What Makes Minecraft So Addictive? Retrieved from http://www.overthinkingit.com/2010/11/11/minecraft-vs-second-life/

Boeing, G. (2016). Visual Analysis of Nonlinear Dynamical Systems: Chaos, Fractals, Self-Similarity and the Limits of Prediction. *Systems*, *4*(4), 37. 10.3390/systems4040037

Brach, M., Hauer, K., Rotter, L., Werres, C., Korn, O., Konrad, R., & Göbel, S. (2012). Modern principles of training in exergames for sedentary seniors: requirements and approaches for sport and exercise sciences. *International Journal of Computer Science in Sport*, *11*, 86–99.

Génevaux, J.-D., Galin, É., Guérin, E., Peytavie, A., & Beneš, B. (2013). Terrain Generation Using Procedural Models Based on Hydrology. *ACM Trans. Graph.*, *32*(4), 143:1–143:13. 10.1145/2461912.2461996

Hendrikx, M., Meijer, S., Van Der Velden, J., & Iosup, A. (2013). Procedural Content Generation for Games: A Survey. *ACM Trans. Multimedia Comput. Commun. Appl.*, *9*(1), 1:1–1:22. 10.1145/2422956.2422957

Hosking, C. (2013, October 12). Stop dwelling on graphics and embrace procedural generation. Retrieved from http://www.polygon.com/2013/12/10/5192058/opinion-stop-dwelling-on-graphics-and-embrace-procedural-generation

Jones, O. (2013, September 1). Polygons to Pixels: The Resurgence of Pixel Games. Retrieved from http://gamingillustrated.com/polygons-to-pixels-the-resurgence-of-pixel-games/

Korn, O., Funk, M., Abele, S., Hörz, T., & Schmidt, A. (2014). Context-aware Assistive Systems at the Workplace: Analyzing the Effects of Projection and Gamification. In *PETRA'14 Proceedings of the 7th International Conference on PErvasive Technologies Related to Assistive Environments* (p. 38:1–38:8). New York, NY, USA: ACM. 10.1145/2674396.2674406

Korn, O., Funk, M., & Schmidt, A. (2015). Assistive Systems for the Workplace: Towards Context-Aware Assistance. In L. B. Theng (Ed.), *Assistive Technologies for Physical and Cognitive Disabilities* (pp. 121–133). IGI Global. Retrieved from http://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/978-1-4666-7373-1

Korn, O., Rees, A., & Schulz, U. (2015). Small-Scale Cross Media Productions: A Case Study of a Documentary Game. In *Proceedings of the ACM International Conference on Interactive Experiences for TV and Online Video* (pp. 149–154). New York, NY, USA: ACM. 10.1145/2745197.2755516

Mandelbrot, B. B. (1982). *The fractal geometry of nature*. San Francisco: W.H. Freeman.

Olano, M., Hart, J. C., Heidrich, W., Mark, B., & Perlin, K. (2003, March 1). Real-time shading languages. Retrieved from http://www.csee.umbc.edu/~olano/s2002c36/ch02.pdf

Perlin, K. (1985). An Image Synthesizer. In *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques* (pp. 287–296). New York, NY, USA: ACM. 10.1145/325334.325247

Perlin, K. (1999, September). *Making Noise*. Presented at the GDCHardCore.

Persson, M. (2011, March 19). Terrain generation, Part 1. Retrieved from http://notch.tumblr.com/post/3746989361/terrain-generation-part-1

Schell, J. (2015). *The Art of Game Design: A Book of Lenses* (Second edition). Boca Raton: CRC Press.

Shaker, N., Togelius, J., & Nelson, M. J. (2016). Fractals, noise and agents with applications to landscapes. In *Procedural Content Generation in Games* (pp. 57–72). Springer International Publishing. 10.1007/978-3-319-42716-4_4

Thue, D., & Bulitko, V. (2012). Procedural game adaptation: Framing experience management as changing an mdp. In *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference*. Retrieved from http://musicweb.ucsd.edu/~sdubnov/Mu270d/AIIDE12/03/WS12-14-012.pdf

Togelius, J., Kastbjerg, E., Schedl, D., & Yannakakis, G. N. (2011). What is Procedural Content Generation?: Mario on the Borderline. In *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games* (p. 3:1–3:6). New York, NY, USA: ACM. 10.1145/2000919.2000922

Yannakakis, G. N., & Togelius, J. (2011). Experience-Driven Procedural Content Generation. *IEEE Transactions on Affective Computing*, *2*(3), 147–161. 10.1109/T-AFFC.2011.6

Ziegler, G. M. (2013). *Mathematik - Das ist doch keine Kunst*. Munich, Germany: Knaus.

Author Biographies

Michael Blatz is Chief Executive Officer (CEO) of KORION GmbH, a software company focusing on games and simulations based in southern Germany.

Blatz studied Virtual Reality and Game Development at the SRH Heidelberg University. He developed games for several platforms, especially mobile and PC. In 2013, he joined the Fraunhofer spin-off KORION. He participated in two national research projects and gained experience in several industrial game development projects. In 2016, he became CEO of KORION.

Blatz is a game enthusiast and an excellent connoisseur of game development strategies and methods on all platforms. His passion for platformers led him to procedural algorithms, e.g. for dynamic level generation. He applies procedural methods in most of KORION's projects—most recently in the Science Fiction strategy game Space Battle Core, where players explore procedurally generated solar systems.

Oliver Korn is a full professor for Human–Computer Interaction (HCI) at Offenburg University in Germany. He also is a certified project manager (German Chamber of Commerce, DIHK), professional member of the ACM and the IEEE, and an evaluator for the European Commission.

After completing his master's focusing on computational linguistics, he worked at the Fraunhofer Institute for Industrial Engineering (IAO) and the Stuttgart Media University (HdM). He focused on HCI, especially simulations, gaming, and gamification. In 2003, he co-founded KORION, a Fraunhofer spin-off developing simulations and games. As CEO, he was in charge of several national research projects and gained experience in countless industrial software development projects, both in the area of entertainment and business intelligence. In 2014, he received his Ph.D. in computer science at the SimTech Excellence Cluster of the University of Stuttgart. His research is published in numerous international publications.

He has been a game enthusiast since the times of the ZX81 and loves games with dynamic and procedural contents like *Civilization* or *Master of Magic*. Today, he works on the convergence of digital technology and real life, focusing on affective computing, gamification, augmented work, and learning. Korn's overall vision is integrating gameful design in education, health and work processes, augmenting and enriching everyday activities. In the area of affective computing, he aims to assess emotional states to improve computational context awareness.