University of Applied Sciences and Arts,
Lucerne

Bachelor Thesis

Supervisors:
Florian Krautkrämer & Manuela Maier-Hummel

Advancements in Procedural Generation

Simon Hischier
Telackerstrasse 5
3948 Oberems
+41 79 688 17 29
simon.hischier@gmail.com

Course of Studies: Digital Ideation, 6. semester
total number of characters: 34967

09 April 2019

Content

1 Abstract

This work discusses the relation between the past lack of procedural content generation in the game industry with the recent uptake of procedural content generation in emerging tools for game developers and artists and the use of procedural content generation in the game industry. How is procedural content generation as part of games and as a developing tool changing so that it is gaining relevance in the game industry and related sectors? In recent years, new tools and a more data-driven approach to procedural content generation have lead to a less noticeable but steady increase in usage across a number of disciplines and workfields and a shift in toolsets in the game industry. The paper observes and analyzes the trends and explains the newfound interest through analogy, literature review, market and workflow analysis. We highlight the deviating techniques these new tools use and where they differ from classic tools. Based on this we draw a conclusion why procedural content generation gains popularity.

2 Introduction

The cost to make games seems to be on an exponential curve as Raph Koster, an industry veteran and lead designer for *Ultima Online* (Origin Systems, 1997), states in 2018. He suggests that game developers should focus a lot more on algorithmic and procedural approaches to keep the cost of games under control[1]. On Quora, an online discussion forum, Steve Theodore who worked on *Half-Life* (Valve Corporation, 1998) and other games, explains in which fields artists spend most time and therefore most money. Theodore states, that most extra time is spent on texturing, shading and animation. He based his reasoning on a practical comparison between two models he worked on: A character model for the game *Half-Life* took about two working weeks and a model for a different game 10 years later took him seven working weeks[2].

Increased work times and a trend to build large, more realistic worlds in AAA games is a major issue. Studios developing AAA games have to spend a lot more money to create this content. In the paper "*The Role of Semantics in Games and Simulations*", Tim Tutenel and Farael Bidarra explain that studios have an ever-increasing pressure to create more models with more realistic looks and behaviour. The challenge the gaming and simulation industry faces is how the models for these worlds are made. Tim Tutenel et al. in 2008 point out that the models get more complex but the way they are made still resembles a high-tech variation on how handicrafts are made[3]. Worlds increase in size but the amount of time and money that a company can spend on game development is limited. Modern models are only improving in looks but not in semantics[3]. Ultimately new 3D models in games differ mostly in vertex count and how they are rendered. They find that enhancing new models with additional semantics has the potential to greatly increase the production value of models and to future proof artists work. Further, better tools and semantically enhanced models have the potential for world generation based on rules where virtually every object is interactable. For the word semantics Tim Tutenel et al. use the Wikipedia definition of semantic as related to meaning, significant and to signify, to indicate[4, 3]. Models with more semantic information can not only help during the development phase:

Games like *The Legend of Zelda: Breath of the Wild* (Nintendo Entertainment Planning & Development, 2017) show how a basic semantically enhanced world leads to an exponential rise in interactability and a more believable world. Samuel Rantaeskola in 2017 states that systems were less powerful in the past. The limiting factor was optimization. Nowadays the content and its increasing complexity is the big limitation[2]. More people working on a project increases overhead and the more work on a project the less impact there is when a single person is added to a team. Ever-increasing cost of development comes with financial side effects. A lot of big studios closed down or had massive layoffs over the past year [5, 6, 7, 8] and this seems to be a long term trend backed by older numbers[9]. The race to higher costs leave midsized studios in an awkward position. They are too big to survive by creating small games but not big enough to compete with AAA games. Steve Theodore recalls seeing a lot of midsized studios closing down in the Seattle area because they grew into big studios, went indie or died[2]. As more AAA games are produced, studios not only compete against each other but have to overshadow their own previous work. For artists to keep up with the ever increasing demand for more output in a set timeframe, industry software is incorporating Procedural Content Generation (PCG) into their products which seems to be a very promising route. We define PCG as a way to generate (game) content and visual effects through algorithms. It has the potential to greatly reduce the workload for artists. However the increased relevance of PCG cannot be pinpointed to a single development or enhancement. This paper highlights several advancements in various fields which are partly responsible. First in section 5 we take a look at how modern workflows for drafting content (as defined in section 4) and for pre production can look like. The section 6 highlights advancements in texturing based on procedural algorithms. The following section, middleware (section 7), describes previous and recent middleware advancements related to game development. We find the most important advancements are the uptake of scripting languages in section 8 because it leverages the potential of PCG and puts it in the hands of a large audience.

3 Problem/Research Question

The computer game industry is a rather collaborative and open industry as shown by the popularity of the Game Developers Conference[10] and Gamasutra[11]. Ideas, knowledge and creative techniques are shared and adapted in different studios. However, the industry is very competitive. Developing games is very expensive, they are produced over several years mostly without a revenue stream during the development period. The spent money needs to be reclaimed on game release putting the studios under great pressure. Games are widely different to appeal to various audiences but getting the development cost back is in no way guaranteed. The fact that in developed countries the game markets are close to saturation[1] does not help. For game studios this results in increased competition with each other. Games try to have unique features which can be marketed as unique selling points. Each game series has its strengths, long running series like *Assassins Creed* (Ubisoft) try to extend the gameplay and, if possible, keep their core gameplay formula untouched. To achieve this each studio is developing unique workflows, engines, management strategies etc. This leads to several problems: (i) The production pipeline in each studio is different. Every game engine needs data in different formats sometimes uniquely crafted just for one

engine. (ii) Acquiring new talents and introducing them to the workflow costs a lot of time and money. Every new employee must learn the engine requirements and how to produce content aimed to meet the engines needs. This difference in engines is one of many reasons why standards for various data formats are kept to the lowest common denominator. However, Unity and Unreal Engine have helped streamline the workflow for studios in some way: Some rendering pipelines like the physical based rendering pipeline has become widespread and is used on many devices. As an example there is a new 3D object data format called *glTF2.0* to exchange 3D data with textures for physical based rendering as a format requirement[12]. This helps keeping track of files and reduces checks to load new models. Most engines can use the same texture components (basecolor, metallic, roughness). (iii) But unique game aspects demand for unique data formats on top of these commonly shared data or need additional information. (iv) However the dataformats cannot help produce content more quickly. The demand for more content in the same timespan remains unsolved with or without common components. Standardized engines did not solve the problem of limited content output[1].

This paper gathers several advancements in various fields which are partly responsible why emerging tools differ from traditional solutions. With these tools more PCG is integrated in various steps of game development and the new tools are adapted into workflows slowly but steadily. These new tools yield potential for studios to accelerate workflows and content output. Besides directly influencing productivity, these standardized tools independent from game engines have the potential to immensely reduce the amount of time for new employees to be productive. We base this assumption on several reasons: Independent and widely used tools can be taught in schools. Tools used in different domains greatly reduce the barrier for new talents to enter the gaming industry. In return the developers of these tools profit from a big user base that will foster the community around those tools and can help improve the user experience. A big community increases productivity as more questions are answered and creative people can rely on an increasing knowledge database. With new tools and formats becoming standardized and tested, mainstream engines start to adapt these tools as proven by the adaption of the *glTF2.0* 3D format[13]. A more hidden facette of emerging tools such as *Houdini* (SideFX), *Substance Designer* (Allegorithmic) and *Blender* (Blender Foundation) is, that they all integrate some level of PCG as we will discuss in section 8. We specifically chose these 3 tools to cover a broad aspect of game development such as pre-production and content generation (defined in section 4), modeling and texturing. We chose *Substance Designer* because it is an emerging tool in the game industry. Alternatives like *NeoTextureEdit* (Holger Dammertz, GNU LGPL v. 3) and *MapZone* (Allegorithmic) are discontinued. *Blender* is a free alternative to *Cinema4D* (Maxon) and *Maya* (Autodesk) with a huge and supportive community. *Houdini* is relevant because of the procedural workflow and the integration in game engines. As pointed out in section 2, current 3D assets still lack semantics which can be a very powerful catalyst to increase content output. As PCG is a core part of this trend we ask the question: How is procedural content generation as part of games and as a developing tool changing so that it is gaining relevance in the game industry and related sectors? PCG lets designers iterate on models of previous games. Integrating semantics and enhancing the data becomes easier. Maturing and tested formats reduce the complexity of the internal engine logic, outsourcing the reading and writing of these formats to middleware converters.

4 Categories of Procedural Generation

We are using PCG categories based on the book *Procedural Generation in Game Design*[14, p. 3] where the following four categories are described. We chose these categories because it honors the broader game development pipeline. The categorization allows for a categorization of PCG based on where in the production pipeline it is used and does not categorize what kind of PCG or what technology is used.

4.1 Integral

The use of PCG must be planned and considered in the concept and game design from the get-go. These games rely heavily on a working PCG and even core gameplay can be affected. Games such as *Rogue* (A.I. Design, 1980) or a more modern game like *Dwarf Fortress* (Bay 12 Games, 2006) are using PCG extensively and would not work without it. They need it to be the type of game they are. Changes to the project planning have vast implications on the codebase. These games are built around central algorithms and project changes will result in redrafting these algorithms.

4.2 Drafting Content

From a game design perspective these games do not rely on PCG from the start. Game designers rely on PCG to generate initial drafts of game content such as the map or items. These drafts can be looked through by humans and are then handpicked. But before algorithmically generating these initial drafts of the final game content, artists can prototype game parts and handbuild parts to get a better understanding of what the algorithm should generate so they can start working on other parts of the game right away. Some games use this method to generate a world which is polished by hand afterwards. An example of generating and polishing is *Skyrim* (Bethesda Game Studios, 2011) and a more recent and sophisticated example is *Far Cry 5* (Ubisoft, 2018) where Carrier Étienne eplained that the world was modified by humans but vast parts were regenerated daily by procedural algorithms[15].

4.3 Modal

Some games are built with little or even without the use of PCG and it is added later on during development. Even after release, PCG can be added in the form of an “infinity mode” or as procedural maps, for example in *Rust* (Facepunch Studios Ltd, 2013). While this type can add a lot of replay value to a game it is mostly used just for enhancing replayability and does not add new or innovative content to the game.

4.4 Segmented

Segmented refers to a modular game where the gameplay, levels, music etc. can be split into independent parts. Some of these parts can include PCG. The development can continue with or without these procedural parts. An example might be *Left 4 Dead 2* (Valve Corporation, 2009) where the enemies and the weather changes to adapt to the

players performance. If the desired standard is not met by the algorithm, the team can simply revert to handcrafted content. It can include parts of the game such as procedural music, graphical effects or randomized elements. At all times, the game developers have the possibility to revert to hand-generated content.

5 Workflows

Anastasia Opara in 2017 built a swamp village generator with *Houdini*. The village is completely generated with PCG and can generate an endless number of houses. Her output could be used in all defined categories. She says the interesting part of this method is exploring what the algorithm creates based on the rules given. "One of the most satisfying part was, when generating the final lake houses, the network would give me unexpected, but very creative results [...]"[17]. Anastasia Opara sees art as something we can translate into computer language. However, she explains that the workflow was very labor intensive and quite challenging. The village generator was created in various steps with increasing granularity. As her motivation for this project she states: "My goal was to expand the limits of what is expected from a procedural content generation [...]"[18]. To give an other example of a procedural enhanced workflow we look at the 2018 Blender Conference in Amsterdam, where Andrew Price tried to predict changes to the 3D industry in the next 5 to 10 years. A big point besides machine learning assistance becoming a standard is the adaption of procedural workflows and them becoming mainstream[16]. Price states that procedural texturing made by *Substance Designer* in combination with *Substance Painter* (Allegorithmic) is saving big studios money in the range of hundreds of thousands of US Dollars. He elaborates on this by giving an example. His service *Poliigon* (CGFort Pty Ltd) started out photographing real materials but switched to a more labor intensive workflow. Materials are created virtually in *Substance Designer* which comes with a great increase in value: Using these generated materials allows the user to adjust color, add details, scale to infinity and enhance with more semantic information which then can be used for PCG or directly in the game engine. Already existing textures can be enhanced as well, although with limitations. The next section gives an example of how already existing textures can be reused.

6 Procedural Stochastic Textures

Traditionally textures are a big topic in game development. Textures have seen many enhancements and iterations because it is a big part of modern rendering pipelines and reaches back to the early days of modern computer graphics. Games in the 80's and 90's had hard limits on disk space[19] and designing games involved balancing storage space between program code, music and images. We give two examples on these limitations: Besides a single digital sample channel, sounds on home consoles like the Nintendo Entertainment System (NES) were limited to hardware generated tones[20]. Game developers had to generate most of the music via PCG. Another limitation was that consoles like the NES and Super Nintendo Entertainment System (SNES) had no chips to calculate complex 3D objects. The Picture Processing unit was used to

display only sprites. To fill a TV screen the NES and SNES used texture tiling, a technique for creating small texture samples which can be tiled to create a seamless pattern. This technique is still in use for modern games with more complex textures and increased texture size. Games using PCG as an integral part rely heavily on tileable textures or alternatively use generated procedural textures for example Perlin[21] or Worley[22] noise. Those approaches come with disadvantages: Tiles are very repetitive and Perlin noise needs very sophisticated algorithms to generate interesting levels. We see a great increase in production value by using another texture generating solution as a modern alternative: By-example noise algorithms. These algorithms take a stochastic example texture and generate larger versions out of it. Until now they were too slow for real time generation and therefore were only suitable for drafting content (section 4). A recent paper by Heitz and Neyret created an algorithm able to create stochastic textures on-the-fly[23]. This enables artists to build levels and getting direct feedback with final textures already applied during building, making it suitable for all defined PCG categories. This technique even allows for algorithmic level generation without using tile based textures. This is especially important because previously, textures which were not based on tiles lead to hard edges and repeating patterns as shown in Figure 1.

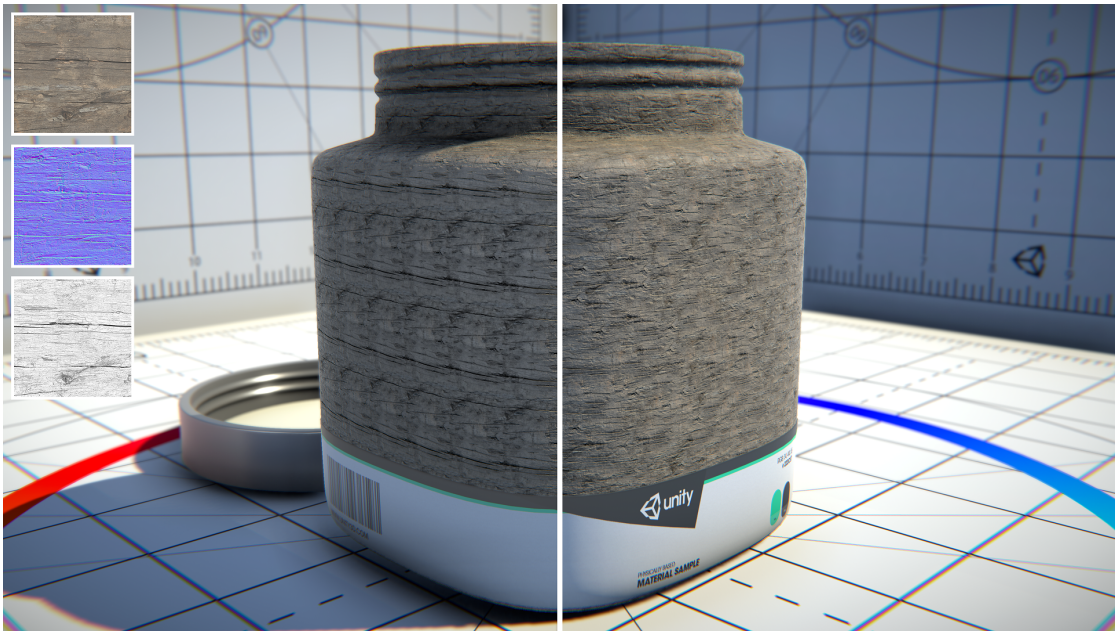

Figure 1: Wood detail maps with albedo, normal and occlusion textures. Retrieved 11:50, March 9, 2019 from https://blogs.unity3d.com/2019/02/14/procedural-stochastic-texturing-in-unity/

7 Middleware

A different area where PCG has had a great impact is middleware used by game developers. Creating completely new tools (for PCG) from scratch is a huge investment in time. PCG can save time if used at the appropriate places but Darren Grey writes, that using PCG is a bad idea when there are time restrictions[14, p. 6]. It is even worse if PCG is used as an integral part of the game as described in section 4. The game stands and falls with the algorithm, every part of the game is influenced by the generation. As an alternative to this high risks we see procedural middleware or usable parts already developed by other companies. An example of a widely used PCG algorithm middleware for game studios is *SpeedTree*[24]. The software generates trees for games in all forms and sizes. For a game studio this is a very small investment compared to writing their own tree generator (or even worse, handcrafting) just to fill the background of a scene. However, these kind of software middlewares are scarce. Writing a tool in house comes with a lot of additional work. The software should not only generate content, it should be easily understandable, controllable and should be fast to allow the artists to experiment with it and put the tool to good use. The commercial lack of use of PCG algorithms correlates with the lack of their control as Roland van der Linden assumes[25]. Easy to use tools and algorithms encourage experimentation and leads to faster adaptation of PCG techniques as Adam M. Smith shows[26]. The algorithm called *WaveFunctionCollapse* described by Alexei A. Efros and Thomas K. Leung is very easy to use without programming knowledge[27]. The algorithm needs an input image and can generate infinitely large output textures. The underlying algorithm works out of the box and does not need tweaking to get great results. This data driven algorithm can be adapted to 3 dimensional space. Oskar Stalberg explains how the islands in the game *Bad North* (Plausible Concept, 2018) are generated by using *WaveFunctionCollapse*. A more recent tech demo shows infinite city generation with interesting buildings made by using only around 100 building blocks[28].

8 Rise of Scripting languages

John K. Ousterhout wrote a paper[29] in 1998 where he analyzed the difference in use of system languages (C, C++, Java) and scripting languages (Python, Perl, Javascript). His analysis is important because we find that visual node system (NS)s (a part of the dataflow programming paradigm[30]) are comparable to scripting languages. He pointed out, that scripting languages are much better suited for casual programmers based on several reasons. One reason is the size of the programs. Scripting languages need less lines of code to get the same work done. Higher-level languages need several months of learning to master where as scripting languages lead to results in hours. This is of great importance, it allows a team to hire technical artists who are quickly ready to work on projects and it allows for an easier integration of PCG in various parts of the game development. However both languages are required, they are complementary.

> "Scripting languages assume that a collection of useful components already exists in other languages." [29, p. 2]

The limiting factor in larger adaptations are the power of computers. The paper finds that, the faster computers get the larger the applications built with scripting languages will be. Current developments on various applications indicate that computers are powerful enough for meaningful applications to be made with scripting languages. Workflows are shifting

towards programs with integrated scripting languages, Visual programming language (VPL) or visual NSs (as a category of VPL). VPL today is known from languages focused on teaching programming like Scratch (https://scratch.mit.edu/). Software built around easy to use VPL provide a simple way to introduce children to programming. Learners don't have to memorize sophisticated semantics and every logic unit is visualized, thus greatly reducing the cognitive load during programming. This is great for integrated PCG to be used by users as well. However this educational focus by VPL is a more recent development. The use of VPL predates this focus and modeling languages such as Unified Modeling Language (UML) started to be invented and used by programmers as a standardized way to document programs. During the late 80s and early 90s VPL was evaluated and while met with great skepsis showed convincing results[32]. A noteworthy Environment based on a VPL which started in 1986 is Laboratory Virtual Instrument Engineering Workbench (LabVIEW) which was successfully tested and observed in an industry-based study by the Measurement Technology Center[33]. The performance of LabVIEW convinced the Measurement Technology Center of the viability of programs based on VPL. Today VPLs exist on system-level and as scripting languages. We will use visual NS examples in both categories (system-level and Higher-level) to present current trending developments. Visual NSs combine features from VPL like accessibility, ease-of-use etc. with features from scripting languages like needing less lines of code (here nodes). On top of that, NSs make it easier to reuse code and have great potential to easily include PCG into various steps of the game production. In general NSs allow users to drag and drop logic or program blocks and have strictly defined inputs and outputs which can be connected in between blocks. To elaborate on the potential of NSs we present 3 systems for various parts of the game development.

8.1 Houdini (Side Effects Software Inc.)

Houdini started out as PRISM (Omnibus) which was bought in 1987 by SideFX. The core of this software has always been its procedural workflow. This strong focus on PCG in combination with VPL in the form of a NS has been the core strength of the software[34]. In our procedural categorization section 4, the *Houdini* software in game development is mainly used in drafting content. SideFX not only targets AAA studios, the pricing model has an option for Indie developers as well. SideFX CEO Kim Davidson estimated that in 2012 around 20% of the sales including commercial and games came from lighting and procedural modeling. *Houdini* was originally used for tv production and later expanded to various 3D sectors. To meet requirements of various 3D pipelines, game studios started to adapt *Houdini* even in AAA studios as stated by Artists at Ubisoft[15]. The *Houdini* engine is already integrated into Unity and Unreal[35].

8.2 Unreal Engine Blueprint (Epic Games)

Unreal is one of the most popular game engines to date and the first official release was in 1998. The License model changed drastically with new releases, it started out with license prices up to 350'000$ and with up to 7% royalties[36]. The newest Version has no upfront cost anymore making it more attractive for developers. The engine developer supported Unrealscripting, a classical scripting language, until Unreal

Engine 4. With Unreal Engine 3 Epic Games Introduced Kismet which is a VPL that started replacing UnrealScripting. The current iteration (Unreal Engine 4) replaced Kismet with the Blueprint system, a new VPL version making it possible to create games only with the VPL. The decision to kill UnrealScripting in favour of a fully VPL was made by Tim Sweeny because the VPL directly used the underlying C++ functions[37]. With the Blueprint system the line between handcrafted and procedural elements are blurred. Procedural logic can be easily integrated into a game for example with the object position node and allows developers to build elaborate PCG systems. This allows for PCG to easily be used in the categorie Segmented all the way to the categorie Integral as described in section 4. Blueprint allows the developers for rapid tweaking, which is important not just for PCG. The NS is great for engine optimization as Alan Willard, technical artist at Unreal, explains. The Blueprint system cuts iteration times from compilation to testing from 15 minutes to 30 seconds allowing developers to spend more time on the game development[38]. There is extensive cost savings with this system. The Blueprint system even allows for cross domain work as Willard explains: "I could say: I'm going to convert this pillar into a blueprint and add some sort of trap to it. It means I can really go in and start enhancing my world with interaction that just would not have been possible without a technical artist, a designer and a programmer and now any one of those three can do all of it, provided they have the assets handy."[38].

8.3 Unity Shader Graph (Unity)

Unity released the Shader Graph software with the release of Unity version 2018.1. Shader Graph is a new node based shader editor to create new custom shaders without having to write a line of code. The Graph handles all the code in the background and displays results in real time, which is especially helpful for PCG workflows. A great advancement in pushing the PCG field forward is the Subgraph system and how the Graph is integrated. Many shaders can be build with only a screenshot of the Shader Graph. This means artists can start sharing shaders like they share images on the internet without providing additional code and instructions. The shared Graphs can then be implemented by artists new to shaders, bringing the entry barrier down. At all times the system allows the user to only connect nodes that have the matching input and output. This fail-safe and easy to use system allows for trial and error. We see this as great potential to create new procedural shaders for example dynamic snow or procedural effects. Every node can display the intermediate step of what the output looks like which is great for debugging. Unity provides an API to write custom nodes. This is a great way for artists and programmers to work together. The programmer provides custom nodes which the artists then can use to create new effects. Artists can create complex systems which they can pack into a subgraph, hiding complexity and making /glspcg effects reusable across projects.

9 Conclusion and Future Work

In this paper we highlighted problems the game industry faces. We also show advancements and trends in several fields connected at least partially to PCG. We think that VPL and NS can greatly help reducing the complexity of the content

pipeline and empower individuals in the company. With the help of NS a single person can take ownership over more of the production pipeline again. This keeps the output closer to the artists vision and it allows for more complex experimentation with PCG. Additionally taking over more of the pipeline does reduce the management overhead again. Time saved not having to arrange and discuss how data has to be prepared for the next step in the pipeline is time freed to discuss design choices. In crafting procedural material pipelines and not creating single use materials, studios are future proofing and speeding up their content pipelines. Still more research is needed on the integration of semantics to create a more robust asset pipeline and for future proofing produced content. We think that the steady integration of PCG is of great value here for example to introduce semantics into the workflow. Materials can be reused more easily and enhancing the data in future games is less work than creating materials from scratch for each game. With new tools new services are established for high quality materials. Studios can spend less time creating their own libraries of textures. Bigger parts of the laborious work can be offloaded, recycled or created during pre production. Students can learn the discussed emerging new tools in school and do not have to learn significant parts of the workflow after getting hired, reducing the time needed to teach employees. Modern procedural algorithms take images or models as an input and rules don't have to be written programmatically, enabling for faster iteration times and earlier adoptions of PCG in game development. We think recent research on data-driven PCG like *WaveFunctionCollapse* as described in parts of section 7 lead to great tools and tempt artists to experiment with PCG more. We think research on new data-driven algorithms would greatly benefit multiple industry sectors, pushing PCG to be a standard part of prototyping and brainstorming. Using PCG is a first step in augmenting game development with semantics ultimately leading to more interconnected and believable game worlds as described by Tim Tutenel et al. This paper shows that PCG in a more subtle form found its way into many emerging tools already. Most advancements in PCG can be found in the categorie drafting content (section 4). Procedural and hand crafted content start to intertwine and classic tools for textures are slowly replaced by modern alternatives such as *Substance Designer* which integrate more procedural functions.

References

[1] Raph Koster. Gamasutra: Raph Koster's Blog - The cost of games. https://www.gamasutra.com/blogs/RaphKoster/20180117/313211/The_cost_of_games.php (accessed on 19 February 2019), 2018.

[2] Mike Prinke, Eric Backeberg, Samuel Rantaeskola, Steve Theodore, Matti Porkka, Eric Chung, Bill Hitchens, and Rowley Gregory. Why have video game budgets skyrocketed in recent years? - Quora. https://www.quora.com/Why-have-video-game-budgets-skyrocketed-in-recent-years (accessed on 25 February 2019), 2017.

[3] Tim Tutenel, Rafael Bidarra, Ruben M. Smelik, and Klaas Jan De Kraker. The role of semantics in games and simulations. *Computers in Entertainment*, 6(4):1, dec 2008.

[4] Wikipedia contributors. Semantics. https://en.wikipedia.org/w/index.php?title=Semantics&oldid=888248320 (accessed on 09 March 2019), 2019.

[5] Evan Lahti. Here are all the major game studios that have closed in the past year — PC Gamer. https://www.pcgamer.com/here-are-all-the-major-game-studios-that-have-closed-in-the-past-year/ (accessed on 25 February 2019), 2018.

[6] Jason Schreier. Activision-Blizzard Employees Brace For Massive Layoffs. https://kotaku.com/activision-blizzard-employees-brace-for-massive-layoffs-1832488999 (accessed on 25 February 2019), 2019.

[7] Alex Walker. EA's Australian Studio Hit By Massive Layoffs — Kotaku Australia. https://www.kotaku.com.au/2019/02/eas-australian-studio-hit-by-massive-layoffs/ (accessed on 25 February 2019), 2019.

[8] Jason Schreier. Guild Wars 2 Developer ArenaNet Plans For Mass Layoffs — Kotaku Australia. https://www.kotaku.com.au/2019/02/guild-wars-2developer-arenanet-plans-for-mass-layoffs/ (accessed on 25 February 2019), 2019.

[9] Luke Plunkett. Every Game Studio That's Closed Down Since 2006. https://kotaku.com/every-game-studio-thats-closed-down-since-2006-5876693 (accessed on 25 February 2019), 2012.

[10] GDC. GDC — Game Developers Conference. https://www.gdconf.com/ (accessed on 08 April 2019), 2019.

[11] Gamasutra. Gamasutra - The Art & Business of Making Games. https://www.gamasutra.com/ (accessed on 08 April 2019), 2019.

[12] The Khronos Group. glTF Overview - The Khronos Group Inc. https://www.khronos.org/gltf/ (accessed on 22 March 2019).

[13] Khronos Group. glTF Momentum Accelerates with New Support from Facebook, Epic, Unity, and Adobe - The Khronos Group Inc. https://www.khronos.org/blog/gltf-momentum-new-support-facebook-epic-unity-adobe (accessed on 10 March 2019), 2018.

[14] Tanya X Short and Tarn Adams. *Procedural Generation in Game Design*. A. K. Peters, Ltd., Natick, MA, USA, 1st edition, 2017.

[15] Étienne Carrier. Procedural World Generation of Ubisoft's Far Cry 5 — Etienne Carrier — Houdini HIVE Utrecht - YouTube. https://www.youtube.com/watch?v=NfizT369g60 (accessed on 04 March 2019), 2018.

[16] Andrew Price. The Next Leap: How A.I. will change the 3D industry - Andrew Price - YouTube. https://www.youtube.com/watch?v=FlgLxSLsYWQ (accessed on 25 February 2019), 2018.

[17] Anastasia Opara. anastasiaopara — LAKEVILLAGE. https://www.anastasiaopara.com/lakevillage (accessed on 01 April 2019), 2016.

[18] Anastasia Opara. How to make a Procedural Environment for Games. https://blog.therookies.co/2017/05/23/make-procedural-environment-games/ (accessed on 02 April 2019), 2017.

[19] Mark Ferrari. 8 Bit & '8 Bitish' Graphics-Outside the Box - YouTube. https://www.youtube.com/watch?v=aMcJ1Jvtef0 (accessed on 09 March 2019), 2016.

[20] Wikipedia contributors. "Nintendo Entertainment System" Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Nintendo_Entertainment_System&oldid=885685790 (accessed on 09 March 2019), 2019.

[21] Ken Perlin. An image synthesizer. *ACM Siggraph Computer Graphics*, 19(3):287–296, 1985.

[22] Steven Worley. A cellular texture basis function. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 291–294. ACM, 1996.

[23] Eric Heitz and Fabrice Neyret. High-Performance By-Example Noise using a Histogram-Preserving Blending Operator. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 1(2):1–25, aug 2018.

[24] SpeedTree. SpeedTree Vegetation Modeling. https://store.speedtree.com/ (accessed on 11 May 2018).

[25] Roland Van Der Linden, Ricardo Lopes, and Rafael Bidarra. Procedural generation of dungeons. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(1):78–89, mar 2014.

[26] Isaac Karth and Adam M. Smith. WaveFunctionCollapse is constraint solving in the wild. In *Proceedings of the International Conference on the Foundations of Digital Games - FDG '17*, pages 1–10, New York, New York, USA, 2017. ACM Press.

[27] A.A. Efros and T.K. Leung. Texture synthesis by non-parametric sampling. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, pages 1033–1038 vol.2. IEEE, 1999.

[28] Marian. Infinite procedurally generated city with the Wave Function Collapse algorithm — Marian's Blog. https://marian42.de/article/wfc/ (accessed on 20 March 2019), 2019.

[29] J.K. Ousterhout. Scripting: higher level programming for the 21st Century. *Computer*, 31(3):23–30, mar 1998.

[30] Wikipedia contributors. "Dataflow programming" Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Dataflow_programming&oldid=884393969 (accessed on 04 March 2019), 2019.

[31] Visual Studio Docs Contributors. Generate code from UML class diagrams - Visual Studio 2015 — Microsoft Docs. https://docs.microsoft.com/en-us/visualstudio/modeling/generate-code-from-uml-class-diagrams?view=vs-2015 (accessed on 10 March 2019), 2016.

[32] K.N. WHITLEY. Visual Programming Languages and the Empirical Evidence For and Against. *Journal of Visual Languages & Computing*, 8(1):109–142, feb 1997.

[33] R. Jamal and L. Wenzel. The applicability of the visual programming language LabVIEW to large real-world applications. In *Proceedings of Symposium on Visual Languages*, pages 99–106. IEEE Comput. Soc. Press.

[34] Mike Seymour. Side Effects Software – 25 years on — fxguide. https://www.fxguide.com/featured/side-effects-software-25-years-on/ (accessed on 11 March 2019), 2012.

[35] SideFX. Game Development — SideFX. https://www.sidefx.com/industries/games/ (accessed on 11 March 2019).

[36] Jessie Cameron Herz. GAME THEORY; For Game Maker, There's Gold in the Code - The New York Times. https://www.nytimes.com/1999/12/02/technology/game-theory-for-game-maker-there-s-gold-in-the-code.html (accessed on 15 March 2019), 1999.

[37] Alex Wawro. Gamasutra - For Tim Sweeney, advancing Epic means racing into AR and VR. http://www.gamasutra.com/view/news/292573/For_Tim_Sweeney_advancing_Epic_means_racing_into_AR_and_V (accessed on 15 March 2019), 2017.

[38] Stephen Totilo. How Unreal Engine 4 Will Change The Next Games You Play. https://kotaku.com/how-unreal-engine-4-will-change-the-next-games-you-play-5916859 (accessed on 11 March 2019), 2012.

List of Figures