Fachhochschule Luzern HSLU

Studiengang Digital Ideation, Bachelor
6. Semester

TBD

Simon Hischier

March 24, 2019

Inhalt

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 games industry and related sectors? New tools and a more data-driven approach to procedural content generation in recent years has 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 what these new tools and approaches provide and what previous tools were missing and the conclusion why procedural content generation gains popularity based on these changes.

2 Introduction

The cost to make games seems to be on an exponential curve as Raph Koster states. He suggests among other things 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 and other games, gives an example on the time needed for assets in games. A character model for the game *Half-Life* (Valve Corporation, 1998) took about 2 working weeks and a model 10 years later for a different game took him seven working weeks. Theodore states, that most extra time is spent on texturing, shading and animation[2]. Those are fields where Procedural Content Generation (PCG) has seen huge improvements.

Increased worktimes and a trend to bigger, more realistic worlds in AAA games is a big problem. Studios developing AAA games have to spend a lot more money to create these resources. 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 big problem the gaming and simulation industry is facing is how the models for these worlds are made. Tim Tutenel et al. point out that the models get more complex but how they are made still resembles a high-tech variation on how handicrafts are made[3]. Worlds increase 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. Ultimately new 3D models in games differ mostly in vertex count and how they are rendered. Enhancing new models with additional semantics has the potential to greatly increase the production value of models. We use the Wikipedia definition of semantic as related to meaning, significant and to signify, to indicate[4]. Better tools and semantically enhanced models have the potential for rule based world generation where virtually every object is interactable. 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 increase in interactability and a more believable world. Samuel Rantaeskola[2] states that in the past systems were less powerful and therefore the limiting factor was optimiza-

tion. Nowadays the content and increasing complexity of content is the big limitation. 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 does come 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 midsize 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 midsize studios closing down in the seattle area because they grew into big studios, went indie or died[2]. As more AAA games get produced, studios not only compete against each other and have to overshadow their own previous work. A very promising route seems to be PCG. An increasing number of industry software is incorporating PCG into their products and it allows the artists to keep up with ever increasing demand for more output in the same time. We define PCG as a way to generate (game) content through algorithms. It's a way to automatically generate content and can greatly reduce the workload of artists.

3 Problem/Research Question

The computer, games industry is a rather collaborative and open industry. Knowledge and creative techniques are quickly shared and adapted in different studios. However, the industry is very competitive. Games cost a lot of money and are produced over several years without earning a dollar during the development period. The spent money needs to be reclaimed on game release putting the studios under great pressure. Games are widely different and 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. As part of the entertainment industry, games try to have unique features which can be marketed as unique selling points. Each gameseries has its strengths, long running series try to vary the gameplay and, if possible, keeping their core gameplay formula untouched. To achieve this each studio is developing unique workflows, engines, management strategies etc. allowing the studio to stand apart from other studios. This leads to several problems: Acquiring new talents costs a lot of money. The production pipeline in each studio is different. Every game engine needs data in different formats sometimes uniquely crafted just for the engine. 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 is minimal. Although Unity and Unreal Engine have helped streamline the workflow for studios in some ways: Some rendering pipelines like the physical based rendering pipeline become widespread and 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 requirement[10]. This helps keeping track of files and reduces checks to load new models. Most engines can use the same texture components (basecolor, metallic, roughness). But unique game aspects demand for unique data formats on top of these commonly shared data or need additional information. However the dataformats can not help produce content quicker. 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 on various fields who are partly responsible why emerging tools differ from traditional

solutions. With this tools more PCG is integrated in the 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 time for new employees to be productive. We base this assumption on several reasons: Independent and widely used tools can be teached in schools. Tools used in different domains greatly reduce the barrier for new talents to enter the gaming industry. In return the developers for this tools profit from a big user base that will grow the community around those tools and can help improve the user experience. A big community is increasing 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[11]. 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 pointed out in section 2, current 3D assets still lack semantics which can be a very powerful katalysator to increase content output. With PCG becoming commonplace in tools, the additional work to integrate semantics into content production is greatly reduced. 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 games industry and related sectors? PCG lets designers iterate on models for previous games and enhancing the data becomes easier. Maturing and tested formats reduce the complexity of the internal engine logic, outsourcing the read 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"[12, p. 3] where the following four categories are described.

4.1 Integral

The use of PCG is part of the game design from the start. This 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* (Dwarf Fortress, 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 build around central algorithms and project changes will result in redrafting these algorithms.

4.2 Drafting Content

From a game design perspective this 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 this 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 and to start working on other parts of the game right away. Some games use this method to generate a world which is then polished by humans. 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 explains how the world was modified by humans but regenerated daily by his team[13].

4.3 Modal

Some games are build with little or even without the use for PCG and it gets 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 or replay value to a game it is mostly used just for that and does not add innovative content to the game.

4.4 Segmented

A game build with segmented content including closed off areas where PCG is used. The development can continue with or without these areas if the desired standard is not met by the algorithm. The term "areas" doesn't need to be restricted to levels. It can include parts of the game such as procedural music, graphical effects or randomized elements. The game developers have at all times the possibility to revert to hand-generated content.

5 Tools for Procedural Content Generation

At the Blender Conference 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 becoming mainstream[14]. 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 could then be used for PCG or directly in the game engine.

6 Procedural Stochastic Textures

Textures is traditionally a big topic in game development. A lot of research went into textures because it's 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[15] and designing games involved balancing storage space between program code, music and images. We list two examples on these limitation: Besides a single digital sample channel sounds on home consoles like the Nintendo Entertainment Sys-

tem (NES) were limited to hardware generated tones[16]. 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 increased 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[17] or Worley[18] 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 these were too slow for real time generation and therefore were only suitable for drafting content. A recent paper by Heitz and Neyret created an algorithm able to create stochastic textures on-the-fly[19]. This enables artists to build levels and getting direct feedback with final textures already applied during building. This technique even allows for algorithmic level generation without using tile based textures. This is especially important because previously, textures not based on tiles lead to hard edges and repeating patterns shown in Figure 1.

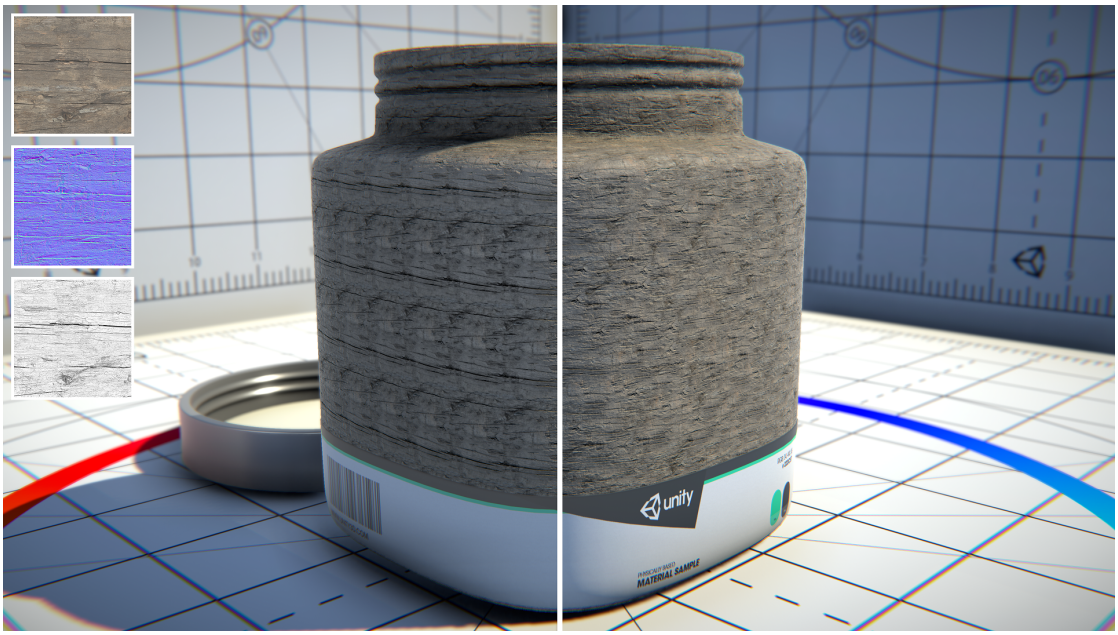

Figure 1: Wood detail maps with albedo, normal and occlusion textures. Retrieved 11:50, March 9, 2019 from https://blogs.unity3d.com/2019/02/14/procedural-stochastic-texturing-in-unity/

7 WaveFunctionCollapse

Creating completely new tools for PCG is a huge time investment. Darren Grey writes, that using PCG is a bad idea when there is a time restriction[12, p. 6]. It is even worse if PCG is used as an integral part of the game. 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[20]. 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 scares. 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 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[21]. Easy to use tools and algorithms encourage experimentation and leads to faster adaptation of PCG techniques as Adam M. Smith shows[22]. The algorithm called "WaveFunctionCollapse" described by Alexei A. Efros and Thomas K. Leung is very easy to use without programming knowledge[23]. 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 only using around 100 building blocks[24].

8 Rise of Scripting languages

John K. Ousterhout wrote a paper[25] 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 in this paper we look at visual node system (NS)s (a part of the dataflow programming paradigm[26]) as 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 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. However both languages are required, they are complementary.

> "Scripting languages assume that a collection of useful components already exists in other languages."[25, 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 build 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. 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. While UML is a modeling language, it can be used in various ways such as to generate source code[27] in various granularity depending on the model used. As such UML can be viewed as a VPL. During the late 80s and early 90s VPL were evaluated and while met with great skepsis showed convincing results[28]. A noteworthy development environment that started in 1986 based on a VPL is Laboratory Virtual Instrument Engineering Workbench (LabVIEW) which was successfully tested and observed on an industry-based study by the Measurement Technology Center[29]. The performance of LabVIEW convinced the Measurement Technology Center of the viability of programs based on VPL. Today VPLs exist on the 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 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 Side Effects Software and later used as the basis for *Houdini*. 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[30]. 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 aswell. SideFX CEO Kim Davidson estimated that in 2012 around 20% of the sales including commercial and games comes from lighting and procedural modeling. Houdini originally was used for tv production and later expanded to various 3D sectors. With this expansion to meet requirements of various 3D pipelines, game studios started to adapt Houdini even in AAA studios as stated by Artists at Ubisoft[13]. The Houdini engine is already integrated into Unity and Unreal[31].

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 costing up to 350'000$ in license cost and up to 7% royalties[32]. The newest Version has no upfront cost anymore making it more attractive for developers. The engine developer supported Unearlscripting, 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[33]. With the Blueprint system procedural elements can be easily integrated into a game for example with the object position node or allows developers to build elaborate PCG systems. This allows for PCG to easily be used in the categorie Seg-

mented 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. Alan Willard explains the Blueprint system cuts iteration times from 15 minutes to 30 seconds allowing developers to spend more time making a good game and not waiting for it to compile[34]. 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."[34].

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 very important not just 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 barrier to entry down. At all times the system only allows the user to connect nodes that have the matching input and output. This failsafe and easy to use system allows for trial and error. Shader Graph is a great tool that puts the right options into the right peoples hands. 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 providing custom nodes which the artists then can use to create new effects. Artists can create complex systems which they can then pack into a subgraph, hiding complexity and making effects reusable across projects.

9 Conclusion and Future Work

In this paper we highlighted problems the games industry is facing in the not so distant future. 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 allows for complexer experimentation and in the end keeps the output closer to the artists vision. Taking over more of the pipeline reduces 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 material pipelines and not creating single use materials, studios are future proofing their content pipelines. Materials can be reused more easily and enhancing the data in future games is less work then 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 for textures. Bigger parts of the laborious work can be offloaded, recycled or created during pre production. Students can learn these 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. Making PCG a standard part of prototyping and brainstorming. Using PCG is the first step in augmenting game development with semantics ultimately leading to more interconnected and believable game worlds. Potentially opening and connecting game production pipelines and other 3D pipelines such as movies.

10 References and acronyms

Acronyms

LabVIEW Laboratory Virtual Instrument Engineering Workbench. 8

NES Nintendo Entertainment System. 5, 6

NS node system. 7, 8

PCG Procedural Content Generation. 2–9

SNES Super Nintendo Entertainment System. 6

UML Unified Modeling Language. 8

VPL Visual programming language. 7, 8

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, 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, 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, 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/, 2018.

[6] Jason Schreier. Activision-Blizzard Employees Brace For Massive Layoffs. https://kotaku.com/activision-blizzard-employees-brace-for-massive-layoffs-1832488999, 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/, 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/, 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, 2012.

[10] The Khronos Group. glTF Overview - The Khronos Group Inc. https://www.khronos.org/gltf/.

[11] 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, 2018.

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

[13] Étienne Carrier. Procedural World Generation of Ubisoft's Far Cry 5 — Etienne Carrier — Houdini HIVE Utrecht - YouTube. https://www.youtube.com/watch?v=NfizT369g60, 2018.

[14] Andrew Price. The Next Leap: How A.I. will change the 3D industry - Andrew Price - YouTube. https://www.youtube.com/watch?v=FlgLxSLsYWQ, 2018.

[15] Mark Ferrari. 8 Bit & '8 Bitish' Graphics-Outside the Box - YouTube. https://www.youtube.com/watch?v=aMcJ1Jvtef0, 2016.

[16] Wikipedia contributors. "Nintendo Entertainment System" Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Nintendo_Entertainment_System&oldid=885685790, 2019.

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

[18] 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.

[19] 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.

[20] SpeedTree. SpeedTree Vegetation Modeling. https://store.speedtree.com/.

[21] 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.

[22] 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.

[23] 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.

[24] Marian. Infinite procedurally generated city with the Wave Function Collapse algorithm — Marian's Blog. https://marian42.de/article/wfc/, 2019.

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

[26] Wikipedia contributors. "Dataflow programming" Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Dataflow_programming&oldid=884393969, 2019.

[27] 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, 2016.

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

[29] 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.

[30] Mike Seymour. Side Effects Software – 25 years on — fxguide. https://www.fxguide.com/featured/side-effects-software-25-years-on/, 2012.

[31] SideFX. Game Development — SideFX. https://www.sidefx.com/industries/games/.

[32] 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, 1999.

[33] 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_VR.php, 2017.

[34] 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, 2012.

List of Figures