Fachhochschule Luzern HSLU

Studiengang Digital Ideation, Bachelor
6. Semester

TBD

Simon Hischier

March 11, 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 tries to explain the newfound interest through analogy, literature review, market and workflow analysis. We want to highlight what these new tools and approaches provide and what previous tools were missing and the conclusion why procedural content generation gains popularity again based on these changes.

2 Introduction

The cost to make games seems to be on an exponential curve as Raph Koster states[1]. 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. On Quora, an online discussion forum, Steve Theodore gives an example on the time needed for assets in games. A model for the game Half-Life 1 took about 2 working weeks and a model 10 years later 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.

Studios developing AAA games spend a lot more money on creating 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. Bigger, more realistic worlds is a trend in the AAA market. 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]. 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. Worlds increase but the amount of time and money that a company can spend on game development is limited. Better tools and semantically enhanced models have the potential for rule based world generation where virtually every object is interactable. 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 optimization. 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 side effects. A lot of big studios closed down or had massive layoffs over the past year [4, 5, 6, 7] and this seems to be a long term trend backed by older numbers[8]. 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].

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 the 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 regained on release putting the studios under great pressure. Games are widely different and getting the development cost back is in no way guaranteed (reference needed). The fact that in developed countries 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 and long running series try to vary the gameplay while keeping their core gameplay formula untouched if possible. Each studio is therefore 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 how to produce content which meets the engines requirements. stitch together Standards for data is minimal because each engine needs unique data. Although Unity and Unreal Engine have streamlined the workflow for studios in some ways: Some rendering methods like physical based rendering become industry accepted (ref?). Most engines can use the same texture components like bump map, reflection map etc. But unique game aspects demand for unique data formats on top of these commonly shared data or need additional information. The demand for more content in the same timespan remains unsolved with or without common components too. Standardized engines did not solve the problem of limited content output[1].

Here we look at how emerging tools differ from traditional solutions. New tools are adapted into workflows slowly but steadily. As new third party tools get integrated, studios can massively profit from these new tools. 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. These tools can be teached in schools. Cross domain tools greatly reduce the barrier for new talents to enter the gaming industry (REF?). The tool developers profit from a wider user base not just because of higher sales figures but more users will grow the community around those tools. A big community is vital to be productivity (REF?) as more questions are answered and creative people can rely on an increasing knowledge. With new tools and formats becoming standardized and tested, mainstream engines start to adapt these tools as proven by the adaption of the glTF 3D format[9]. Emerging tools such as Houdini, Substance Designer and Blender all integrate some level of PCG. As pointed out in section 2, current 3D assets lack semantics. With PCG becoming commonplace in tools, the work to

include more information 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 Literature Review

4.1 Categories of Procedural Generation

We are using PCG categories based on the book "Procedural Generation in Game Design"[10, p. 3] where the following four categories are described.

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 the algorithms.

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. 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[11].

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 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.

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.

4.2 Rise of Scripting languages

John K. Ousterhout wrote a paper[12] 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 systems (a part of the dataflow programming paradigm[13]) as comparable to scripting languages in the sense that they are more easily understandable. 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."[12, 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 node systems (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 the programming. 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[14] in various granularity depending on the model used. As such UML can be viewed as VPL. During the late 80s and early 90s VPL were evaluated and while met with great skepsis showed convincing results[15]. A noteworthy development environment that started in 1986 based on VPL is Laboratory Virtual Instrument Engineering Workbench (LabVIEW) which was successfully tested and observed on an industry-based study by the Measurement Technology Center[16]. The performance of LabVIEW convinced the Measurement Technology Center of the viability of programs based on VPL.

While VPL can be used on the system-level, we are not specifically focused on system-level programming nor are we focused on scripting languages. We will use visual node system examples in both categories (system-level and Higher-level) to present current trending developments. Visual node systems 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, node systems make it easier to reuse code and have great potential to easily include PCG into various steps of the game production. In general node systems 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 node systems we present 3 systems for various parts of the game development.

4.2.1 Houdini (Side Effects Software Inc, https://www.sidefx.com/)

Houdini started out as PRISM (Omnibus) which was bought in 1987 by Side Effects Software and later used as the basis for Hou-

dini. The core of this software has always been its procedural workflow. This strong focus on PCG in combination with VPL in form of a node system has been the core strength of the software[17]. In our procedural categorization subsection 4.1, Houdini is a software mainly used in drafting content. 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[11]. The Houdini engine is already integrated into Unity and Unreal[18]. The pricing model has an option for Indie developers which indicates that the company's target game audience is not only the AAA studios.

5 Methodology and Results

5.1 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[19] and designing games involved balancing storage space between program code, music and images. As an example on the limitation: Besides a single digital sample channel sounds on home consoles like the Nintendo Entertainment System (NES) were limited to hardware generated tones[20]. 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[21] or Worley[22] noise. Those approaches come with disadvantages: Tiles are very repetitive and Perlin noise need very sophisticated algorithms to generate interesting levels. Another texture generating solution is using 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[23]. 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 while not using tile based textures. This is especially important because textures not based on tiles lead to hard edges and repeating patterns shown in Figure 1.

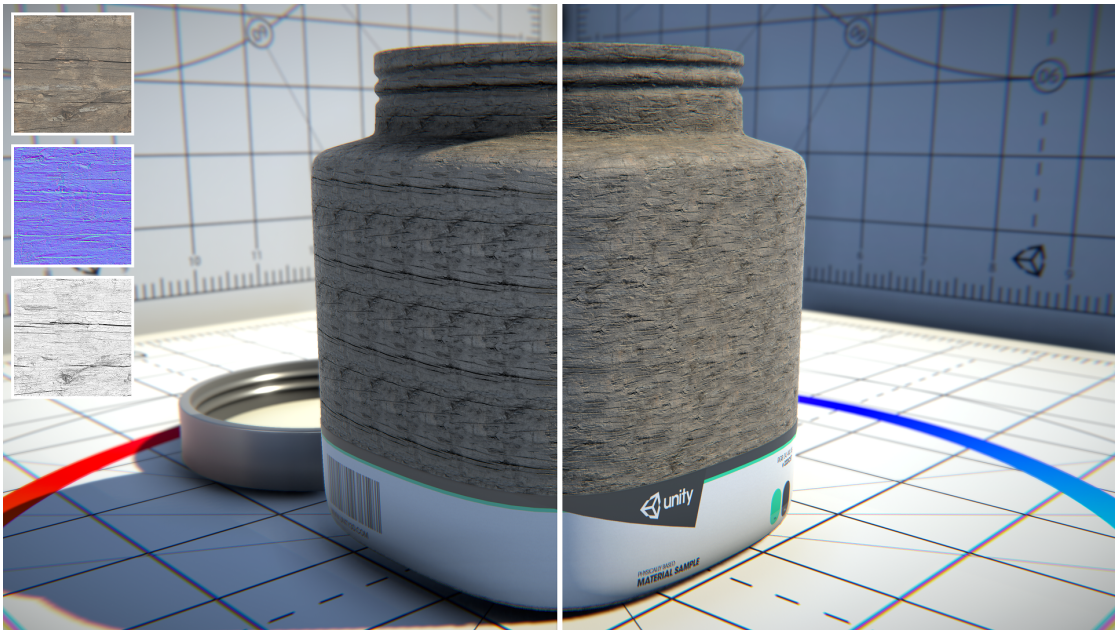

Figure 1: Wood detail maps with albedo, normal and occlusion textures. Retrieved 11:50, March 9, 2019 from https://blogs.unity3d.com/2019/02/14/procedural-stochastic-texturing-in-unity/

5.2 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[24]. Price states that procedural texturing made by Substance Designer in combination with Substance Painter is saving big studios money in the range of hundreds of thousands of US Dollars.

6 Conclusion and Future Work

todo

7 References and acronyms

Acronyms

LabVIEW Laboratory Virtual Instrument Engineering Workbench. 5

NES Nintendo Entertainment System. 6

PCG Procedural Content Generation. 2–6

SNES Super Nintendo Entertainment System. 6

UML Unified Modeling Language. 5

VPL Visual programming language. 5, 6

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] 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.

[5] Jason Schreier. Activision-Blizzard Employees Brace For Massive Layoffs. https://kotaku.com/activision-blizzard-employees-brace-for-massive-layoffs-1832488999, 2019.

[6] 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.

[7] 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.

[8] Luke Plunkett. Every Game Studio That's Closed Down Since 2006. https://kotaku.com/every-game-studio-thats-closed-down-since-2006-5876693, 2012.

[9] 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.

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

[11] 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.

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

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

[14] 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.

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

[16] 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.

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

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

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

[20] Wikipedia contributors. "Nintendo Entertainment System" Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Nintendo_Entertainment_System&oldid=885685790, 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] Andrew Price. The Next Leap: How A.I. will change the 3D industry - Andrew Price - YouTube. https://www.youtube.com/watch?v=FlgLxSLsYWQ, 2018.

List of Figures

List of Tables