***Abstract –* The purpose of this paper is to provide an overview and critical review of the various techniques for the procedural generation of game content both proposed in research and examples in commercial games. This topic has seen a large amount of academic interest over the last number of years thus a fully exhaustive review is beyond the scope of this review. Instead it will explore the strengths and weaknesses of the four main categories of content generation and analyse choice examples for each before concluding with ...**

**!// Figure out how to end this thing !!**

1. Introduction

The development of video games has become increasingly more expensive and time consuming as technology has improved with developers spending millions of dollars and taking years to produce more and more detailed games.[1] This cost is largely caused by both improvements in and the AAA[[1]](#footnote-2) market's focus on graphical technology requiring more and more assets to be created by the artists.

Similarly designing games has proved more challenging due to the wide range of play styles and expectations, even within a single game, as shown by Drachen *et al.* Study of Tomb Raider Underworld.[2] Accommodating all of these players and crafting an enjoyable experience for as many play styles as possible is a daunting task.

One approach which can be applied to both of these problems and has seen large scale adoption within the game's industry is the procedural generation of content. The most obvious benefit to generation is the ability for smaller and less costly teams to produce content rich games. It can also be used as a creative aid to augment a designer's imagination and has also be a core mechanic in Minecraft, one of the most successful games of all time. [5]

Procedural Content Generation(PCG) is the collective term for a large range of techniques to automatically create content with minimum human input. In this context, *'content'* can be defined as any non agent (Non playable Character) behaviour systems within the game i.e. textures, terrain, items, or even entire plot lines [3].

Togelius *et al.* in their taxonomy of the subject offer a further distinction of whether the content is *'necessary or optional'*[4] where necessary content is aspects of the game the player must interact with in order to progress while optional is content which can be entirely ignored by players with no great impact to their experience.

Togelius *et al.* also define several features to aid in the comparison of PCG approaches; *'Online or Offline'* refers to whether the content is generated at the runtime of the game or during development of the game. *'Random Seed or Parameter Vectors'* a detail of the algorithm itself, this property describes the number of parameters, and thus amount of control, the algorithm allows the developer or whether these parameters are 'seeded' using random numbers. Finally *'Stochastic or Deterministic'* explains the amount of randomness within the generation, i.e. a purely deterministic algorithm with generate identical content if repeatedly called with the same inputs.

Due to these sometimes conflicting properties a large array of algorithms for PCG have been developed. Despite this breadth most can be grouped into broad categories; Constructive, Generate and Test and Search Based. These will be discussed in depth in the follow sections along with a separate section for Narrative generation which comes with it's own unique challenges and approaches.

2. Content Generation Techniques

2.1 Constructive

Constructive PCG algorithms will run only once to generate content of an acceptable quality. Meaning that these algorithms must be guaranteed to produce usable content.

The earliest examples of constructive PCG were purely deterministic and used as a form of memory optimisation. As a work around of the hardware constraints of early computers content was saved in a compressed format which could then be expanded out at runtime. A prime example of this was *Elite* [6] which 8 galaxies of 256 planets which were completely generated (name, position, items etc.) from a seed number. A more modern example is *.kkrieger*[7] which compresses an entire first person shooter into a 97kb file. Advances is hardware and the large impact on load times has made the use of PCG for data compression almost entirely abandoned by the industry.

The most prevalent use of constructive algorithms in games is the generation of terrain. A variety of methods exist to produce fractal like landscapes by stochastically manipulating vertex elevation to produce a hightmap representation of the terrain. One of the earliest and more popular implementations is Fournier *et al.* Diamond-Square Algorithm.[9] Introduced in 1982, the algorithm takes a square 2D array with a width of 2^n + 1 and recursively subdivides into diamonds and squares setting the midpoint to the average of each corner plus a random number until all elements are set. At each iteration the magnitude of the random number is reduced to produce gradual slopes. While the Diamond-Square algorithm can produce good looking and realistic terrain in a reasonably efficient way, a number of flaws have been pointed out. Miller [10] criticised the algorithm due to the noticeable creases produced as the more significant changes in height occur in a rectangular grid. Diamond-Square results can improved to remove these artifacts by a generalised stochastic algorithm to calculating the midpoint using a linear estimation rather than averaging the neighbours as described in [11]. Miller also proposed the Square-Square algorithm, outlined in [10], which aimed to avoid the crease like artifacts.

A key flaw with most sub-division techniques is that they offer little control over the generation process beyond providing a roughness constant which scales the random displacement of the midpoint at each iteration to control the steepness of the displacement and the initial seed values. Another disadvantage to sub-division is that it is limited to one of two dimensions meaning that overhangs and cave structures cannot be generated.

An alternate approach to creating fractal terrain to generate noise for elevation data. By far the most popular noise generation is Perlin Noise [12] and its variants [13]-[15]. First introduced in 1985 as an attempt to produce more natural looking textures using gradient noise. The original Perlin algorithm defines an n-dimensional grid with each node assigned a random unit vector. For any given point the dot product of the surround nodes gradient vector and distance to the point are calculated and a fade function applied to interpolate across these values. This leads to very poor scaling due to the complexity O(2n) though subsequent versions resolve this. [13]

The ability to use n-dimensions allows for the generation of overhang and cave structures, unlike sub-division, by using three or more dimensional noise. Minecraft[16] demonstrates this use of Perlin[17].

Ultimately the choice between sub-division or Perlin Noise is largely a person choice. While Diamond-Square is simpler to implement and faster[18] it can lack the variance in terrain that Perlin produces and more easily suffers from linear artifacts. Both algorithms can produce interesting terrain but struggle to produce near photo realistic terrain on their own. Similarly, Howard Zhou *et al.*[19]highlight the lack of control both provide. Users are unable to specify the type or placement of terrain features; i.e. the ability to specify a river must exist in the terrain.

It has been noted [21][22] that due to fractal terrains isotropic nature it cannot accurately describe several aspects of nature. Musgrave *et al.* [22]were the first to suggest modelling physical erosion to produce more natural terrain. They proposed two separate algorithms to model hydraulic and thermal erosion. The hydraulic algorithm works by storing the current height, water and sediment values for each point, over a number of time steps a fraction of the water and sediment is distributed to neighbours with a lower elevation. If the water value reaches 0 the sediment value is added to the height value of that point. The thermal algorithm is a simpler model that smooths large changes in height. At each timestep the difference in elevation between a point and its neighbour is compared to a global talus angel, if the greater than the angel a fixed percentage of the point is transferred to the neighbour. Various improvements have been suggested [23][24][25] for original hydraulic algorithm seeking to produce more physically realistic simulations. However, Musgrave's original was already computationally expensive thus making their approaches less for online generation. Benes explains in [26] that they can produce oscillations and provide little controllability. He broke Musgrave's algorithm into 4 separate simplified steps, water placement, evaporation, deposition and water transport. While not physically accurate Benes approach creates a reasonably approximation in a faster time. In [27] Olsen provides an optimisation and analysis of both Benes and Musgrave's work in the context of their use in video games. While the optimised versions of both the hydraulic algorithm drastically reduced in runtime while maintaining similar erosion scores (standard deviation of the maps slopes divided by the mean value[22]) it were still too slow for runtime generation with 500 iterations taking an average 25s (Not including base terrain generation). Olsen solution was to change Musgrave's thermal algorithm to ignore slopes steeper than the talus angle and flatten the rest. The proposed algorithm created higher scoring terrain in less iterations, achieving 2.15 in 50 iterations compared to the 450 hydraulic required. It also resulted in linear time scaling with 500 iterations taken an average of 10s. The rendered terrain while less realistic looking is likely more usable within games due to the larger flat areas and smoother height transitions.

Overall, erosion techniques can produce more realistic, smoother terrain than a pure fractal approach though at a large computational expense and often provide little control making them less suitable for run-time use.

Attempting to solve the issue of controllability while maintaining the desirable stochastic nature of PCG Doran *et al.* investigated using software agents to model the terrain in [20]. Their system uses a number of asynchronous, autonomous agents that change the evaluation of points within the heightmap according to their predefined goal. To demonstrates the concept Doran *et al.* created 5 agent types; (1.) coastline, creates the initial landmass. (2.) smoothing, take random walks lowering elevation. (3.) beach, flatten areas near coastline. (4.) mountain, raise mountain chains. (5.) river, erode a random path from high to low elevation. Each agent is relatively simplistic but the interaction between them produce unique results. The system is easily extendible by introducing agents with new goals. The large degree of oversight for designers on the generated terrain is achieved by specifying the number of agents of specific types i.e. selecting 3 river agents results in 3 rivers being generated. However this increased control comes at the price of slower generation speeds. A 512x512 heightmap reportedly took 20 seconds on a 3.2GHz pentium, though it should be noted the paper was published nearly 10 years ago at the time of writing and thus performance has likely improved on modern hardware.

While this section has focused on constructive terrain approaches as this is a major focus of both the industry and academia, terrain is not the only content that can be generated.

**!// Expand the speedTree section, or cut entirely**

One of the most prolific constructive PCG tools is SpeedTree [8] which offers both online and offline tools for the generation of foliage. The tool also generates the texture atlases for the foliage.

2.2 Generate and Test

Generate and Test algorithms perform some for of check on the content generated and may perform a number of iterations until content of an acceptable quality has been generated.[4] Unlike constructive algorithms generate and test is not limited to content that is guaranteed to work potentially leader to more varied results. This comes at the expense of runtime as the algorithm could continually fail to create content of an acceptable degree. An example of this approach in action is the world generation in Dwarf Fortress[27] which continually checks and rejects candidate worlds during generation.[28]

2.3 Search Based

A variation on generate and test that instead of a simple pass or fail check evaluated the candidate and assigned a fitness value. Subsequent candidate generation is contingent on the fitness of previous instances with the aim of producing higher scoring content.[4] In other words rather than continually generating from scratch, the algorithm searches for the best solution. This naturally lends itself, but is not limited to evolutionary algorithms (EA); though a significant percentage of the examples discussed do make use of evolutionary techniques.

EA and the variation Genetic Algorithms (GA) work by maintaining a population of potential solutions where the more fit candidates are combined and or “mutated” to form new generations until a solution of sufficient fitness is found or a maximum number of iterations completed. [29] provides an introduction to the topic. The examples discussed in this section can be further distinguished based on how they evaluate the fitness of the generated content.

Designing a function to correctly evaluate something as subjective as game content is non-trivial, especially when developers and players may value different aspects. The most direct solution is to allow users to provide direct feedback to drive the search. This is known as interactive evaluation. This also allows players to tailor their gaming experience to their own preferences. Both [30] and [31] using player feedback to evaluate generated maps.

In [30] Liapis *et al.* evolve real time strategy maps represented as 74 value arrays containing the coordinates of player bases, resource nodes and impassible areas. Each map is evaluated on 3 spacial navigation, 3 resource distribution and 4 aesthetic algorithms with the overall fitness given as a weighted sum. The weighting of this sum was adjusted interactively using two methods; players select their favourite map from a pool of candidates, or players rank the pool based on their preference. Experiments using AI to model player preferences showed the algorithm did generate maps tailored to the players preference, with the rank-based method adapting much faster. The rank-based evaluation was also much faster to adapt to changes in player preferences. However both performed poorly compared to the control, non interactive algorithm which optimised in less iterations, thus creating objectively “better” maps, by always selected the map with the highest fitness. Raffe *et al.* [31] investigated a somewhat similar approach but introduce a recommender-system(RS) based player model to evaluate the content. When a new map is to be generated 7 potential map geometries based on the last map are created using pre-build tiles. The player chosen geometry is then passed to a Compositional pattern-producing network neuroevolution of augmenting topologies (CPPN-NEAT)[32] to evolve CPPNs which calculate the position and density of game content such as weapons, enemies etc. The RS player model evaluates the candidate CPPNs and assigns the probably of being enjoyed. After 10,000 iterations or a perfect candidate being found the geometry and CPPN are combined, once the player has beaten the generated map they rate it between 1 and 5. This rating then updates the RS model for the next iteration. While play testing showed players generally preferred the evolved content over randomly generated maps it was only by a slight margin. Raffe *et al.* were unclear whether this was caused by a poor player model or players self-validating their map choice. Similarly, 75% of participants gave the maps an average rating which Raffe *et al.* suggest might be caused by the CPPN-NEAT reaching a local optimum and repeating content or by the games limited assets.

**//Discuss indirect collection of player preferences stuff**

While allowing players to personalise game content to their taste is an appealing idea and appears to improve the experience for players [33][34] there are problems with the approach. Directly gathering player information interrupts the gameplay and breaks immersion while indirect can easily produce inaccurate or noisy data. The approaches discussed above generally only provide the player a single screenshot on which to evaluate the candidates which not only doesn't give enough information for the player but can prevent optimisation by selecting conflicting preferences.

2.4 Narrative

3. Conclusion

References

1. A. Travers. "GTA 5' Costs $265 Million To Develop And Market, Making It The Most Expensive Video Game Ever Produced: Report". February 2013. GamesIndustry.biz, Available : <http://www.ibtimes.com/gta-5-costs-265-million-develop-market-making-it-most-expensive-video-game-ever-produced-report> Retrieved: 28/10/16
2. A. Drachen, A. Canossa, and G. N. Yannakakis, “Player modeling using self-organization in tomb raider: Underworld.”. September, 2009. Proceedings of the 2009 IEEE Symposium on Computational Intelligence and Games, CIG 2009, Milano, Italy.
3. B. Kybartas, C. Verbrugge. “Analysis of ReGEN as a Graph-Rewriting System for Quest Generation”. November 2013. IEEE Transactions on Computational Intelligence and AI in Games (Volume: 6, Issue: 2)
4. J. Togelius, G. N. Yannakakis, K. O. Stanley, C. Browne. “Search-Based Procedural Content Generation: A Taxonomy and Survey”. September 2011. IEEE Transactions on Computational Intelligence and AI in Games (Volume: 3, Issue: 3)
5. T. Warren. "Minecraft sales top 100 million". June 2016. The Verge. Available: <http://www.theverge.com/2016/6/2/11838036/minecraft-sales-100-million> Retrieved 28/10/16.
6. Elite, Acornsoft. 1984.
7. .kkrieger, Farbrausch. 2004.
8. SpeedTree. Interactive Data Visualization, Inc. Available: <http://www.speedtree.com/>
9. Fournier, Alain; Fussell, Don; Carpenter, Loren. 1982. "Computer rendering of stochastic models". Communications of the ACM.
10. Miller, G.S.P. “The definition and rendering of terrain maps”. 1986. Proceedings of the 13th annual conference on Computer graphics and interactive techniques. SIGGRAPH
11. Lewis, J. P. "Generalized stochastic subdivision". 1987 ACM Transactions on Graphics.
12. Perlin, Ken. "An Image Synthesizer". 1985. Proceedings of the 12th annual conference on Computer graphics and interactive techniques. SIGGRAPH
13. Ken Perlin. “Noise hardware”. 2001. Real-Time Shading SIGGRAPH Course Notes.
14. Ken Perlin. “Improving Noise”. 2002. Proceedings of the 29th annual conference on Computer graphics and interactive techniques. SIGGRAPH
15. Kurt Spencer. “Noise!”. Available : <http://uniblock.tumblr.com/> Retrieved 04/11/16
16. Minecraft. Mojang. 2011
17. Markus Persson. "Terrain generation, Part 1” March 2011. Available: <http://notch.tumblr.com/post/3746989361/terrain-generation-part-1> Retrieved: 04/11/16
18. Travis Archer. “Procedurally Generating Terrain” 2011. Morningside College.
19. Howard Zhou, Jie Sun, Greg Turk, James M. Rehg. “Terrain Synthesis from Digital Elevation Models”. 2007. IEEE Transactions On Visualization and Computer Graphics. Vol.13. No.4
20. Jonathon Doran, Ian Parberry “Controlled Procedural Terrain Generation Using Software Agents”. 2010. IEEE Transactions On Computational Intelligence and AI in Games Vol.2 No. 2
21. LEWIS, J. P. 1990. Is the Fractal Model Appropriate for Terrain? Disney Secret Lab.
22. F. Kenton Musgrave, Craig E. Kolb and Robert S. Mace. “The Synthesis and Rendering of Eroded Fractal Terrains.” 1989. In Proceedings of the 16th annual conference on Computer graphics and interactive techniques.
23. Kenji Nagashima “Computer generation of eroded valley and mountain terrains.” 1997. The Visual Computer, p.g.456-464.
24. D. D’Ambrosio, S. Di Gregorio, S. Gabriele, R. Gaudio “A Cellular Automata Model for Soil Erosion by Water.”. Department of Mathematics, University of Calabria.
25. N. Chiba, K. Muraoka and K. Fujita “An Erosion Model Based on Velocity Fields for the Visual Simulation of Mountain Scenery.” 1998. The Journal of Visualization and Computer Animation, Vol.9
26. Bedrich Benes. “Visual simulation of hydraulic erosion”. 2002. Journal of WSCG 10.
27. Dwarf Fortress. Bay 12 Games. 2006
28. “World Generation” Dwarf Fortress Wiki. [Online] Available: <http://dwarffortresswiki.org/index.php/DF2014:World_generation> Retrieved:09/11/16
29. David E. Goldberg. “Genetic Algorithms in Search, Optimization, and Machine Learning”. 1989. ISBN: 0201157675
30. Antonios Liapis, Héctor P. Martínez, Julian Togelius, Georgios N. Yannakakis. “Adaptive game level creation through rank-based interactive evolution”. 2013. IEEE Conference on Computational Intelligence in Games.
31. William L. Raffe, Fabio Zambetta, Xiaodong Li, Kenneth O. Stanley. “Integrated Approach to Personalized Procedural Map Generation Using Evolutionary Algorithms”. 2014. IEEE Transactions on Computational Intelligence and AI in Games Vol.7 No.2
32. Kenneth O. Stanley, Risto Miikkulainen. “Evolving Neural Networks Through Augmenting Topologies”. 2002. Evolutionary Computation Vol.10
33. E. J. Hastings, R. K. Guha, and K. O. Stanley, “Evolving content in the

Galactic Arms Race video game,” 2009. IEEE Transactions on Computational Intelligence and AI in Games. Vol.1 No.4

1. Sebastian Risi, Joel Lehman, David B. D'Ambrosio, Ryan Hall, Kenneth O. Stanley. “Combining Search-Based Procedural Content Generation and Social Gaming in the Petalz Video Game”. 2012. Eighth Artificial Intelligence and Interactive Digital Entertainment Conference.

1. AAA (pronounced triple A) is a classification of games with the highest development and marketing budgets and usually form the years bestsellers. [↑](#footnote-ref-2)