University of Hull

Final Report

Procedural Cave Modelling

Submitted for BSc Computer Science (Software Engineering)

Student:
Maaz Selia

Supervisor:
Dr. Qingde Li

August 17, 2020

Word count: *x*

ABSTRACT

Procedural content generation is a craft receiving increasing attention for research, as a means of algorithmically generating virtual content. This is driven in large by the increasing demand for virtual content from the gaming industry as well as film and computer graphics. Procedural content generation is used as a means of avoiding the typical costs associated with manual generation of content. A common drawback of procedural content generation techniques is a lack of control over the generated output. This report focuses on the procedural generation of caves. Procedural cave generation has not received much attention to date. Terrain generation, however, has received much attention and can be applied in the generation of caves. This report explores existing methods and approaches for procedural cave (and terrain) generation. And follows to present a unique approach for procedural cave generation inspired by the Marching cubes algorithm. With a focus on control of generated output; inherently through surface level manipulation and intuitive/effective parameterisation. Allowing for the generation of diverse, user controlled, cave structures.

CONTENTS

INTRODUCTION

Procedural content generation has been receiving increasing attention from both academic and industrial audiences. The focus of this attention has been the algorithmic generation of content for virtual environments, movie production and most of all games. Games that range from recreational video games to simulators, as well as games used for training by military and first response personnel to mention a few.

The motivation for this has come from simple size of demand. Both in quantity and quality. 3D worlds have gone from primitive to complex and advanced (Smellik, 2011) and the biggest consumer of these worlds, the gaming industry now worth £36.9 billion and growing by 4.8% year on year (newzoo.com, 2020). And with it, a proportional demand for more content. And also advances in hardware which allow for the runtime procedural generation. (Fletcher et al, 2010)

The task of building these worlds has typically been a manual affair. However, with the increasing demand. PCG can be used to avoid the prohibitive costs associated with manual creation and if this continues the laboriousness and repetitiveness is likely to become unbearable.

PCG can also be looked at as not just an answer to demand but also an opportunity for the conception of an entire new genre of media. A genre of never ending media. A game that you can play forever. A movie that never ends. It can also be used as an opportunity to increase the scope of projects.

However, PCG is not without its flaws. Most PCG approaches struggle from the lack of control the user may exert over the generated output. Often they come with numerous parameters which are hard to manipulate effectively without in-depth knowledge of the underlying algorithms. And often a small change in the input parameters leads to larger undesired change to the output. As observed with fractal terrain generation (Fournier et al, 1982) Another documented drawback is the lack of availability of integrated solutions. Solutions which focus on the integration, unification, and adjustment of existing approaches into a complete world. However, this has been receiving some much needed attention through the developments of frameworks. (Smelik et al, 2002, Yanakakis and Togelius, 2011)

There has been much work done on the procedural generation of various forms of content such as terrain, plants and vegetation distributions , buildings and cities, roads and even rivers (Genevaux, 2013).

As such, although caves are a common feature of virtual worlds. They have not received as much attention as procedural generation of terrain. This can be due to caves being featured in

mostly entertainment industries outside of their use in the training of search-and-rescue personnel.

Terrain generation has become a relatively well established area with approaches for the generation of terrain based on fractals (Fournier et al, 1982), noise (Perlin, 1985), physics-based simulation (Musgrave, 1989. Olsen, 2004) as well as cellular automata (Johnson, 2010). And as caves can be seen simply as specialised configurations of terrain. This report aims to explore various published approaches for procedural cave generation. As well as procedural terrain and content generation.

This report then presents a unique approach to the generation of caves. The method adopted was built upon the marching cubes algorithm. The marching cubes algorithm allows for the generation of a polygonal mesh of a surface based on a discrete scalar field. (Lorensen and Cline, 1987). And the procedural definition of this scalar field is what allows this method for its generation of caves.

Section Literature Review explores existing methods of procedural content, terrain, and cave generation. By way of Fractals, noise-based, image-processing, physics-based simulation, search and solver based, and machine learning.
The following section focuses on the development and implementation of the proposed approach, accompanied by project management.

LITERATURE REVIEW

Procedural content generation techniques explored in this report can be categorised into three main categories. Procedural, Physics-based simulation, and Artificial Intelligence. The Procedural category explores approaches which are noise (Perlin, 1985. Boggus et al. Wollongong), image-processing (Dachsbacher and Stamminger, 2011), cellular automata (Johnson, 2010), and declarative (Smelik, 2011) based. The Physics-based simulation category explores approaches which are based on thermal, hydraulic erosion (Musgrave, 1989. Olsen, 2004), and hydrology (Genevaux, 2013). The Artificial Intelligence category looks at search (Togelius et al, 2011), solver (Smith et al, 2011) and machine learning (Summerville et al, 2018) approaches.

What makes a PCG technique good?

A successful procedural content generation technique would have a few universal characteristics. The most prominent of which is the ability for a user to have control over the generated output, often by means of meaningful and intuitive parameterisation (Smelik et al, 2011). And allows for constant iteration on the generated output. (Smith et al, 2011) For runtime approaches, performance of the technique is critical. Nonetheless, performance is often gained at the cost of correctness (Olsen 2004). As such, for the generation of terrain the capacity to produce output at varying levels of details is essential. This allows for a gain in performance without loss to correctness. See (Dachsbacher and Stamminger, 2011) for an excellent implementation in the form of camera-view dependent level of detail. Lastly, as (Smelik, 2011) derived, a procedural-manual hybrid would be the ideal.

Procedural

The techniques and approaches within this category, though some share similarities, vary greatly in comparison to the specialised focus of the following categories. All approaches within this category pertain specifically to either cave or terrain generation.

Noise & Image Processing

In procedural generation of terrain. A classical approach is the use of heightmaps (Miller 1986 “The definition and rendering of...”). Heightmaps are 2D images where each pixel stores surface elevation data. (Insert pic of heightmap)
The oldest algorithms for the generation of heightmaps are the subdivision methods. This is where a heightmap is divided iteratively, with controlled randomness added at each iteration for detail. (Smelik et al, 2011) This controlled randomness is often based on Fractal noise generators (Fournier et al 1982), such as Perlin noise (Perlin 1985). Which is generated by interpolating and sampling points within a grid of random vectors. A 2D image representation of terrain means common image-processing techniques can be applied to further enhance the

generated terrain. Cellular automata can be effectively applied to run erosion simulation on the terrain (Olsen, 2004). Or use image-processing techniques to alter level of detail based on requirements (Dachsbacher and Stamminger, 2011).

An inherent shortcoming of using heightmaps for terrain generation is the lack of terrain features such as overhangs and caves, as each pixel only supports one surface elevation value. (Boggus et al) overcame this by simply reflecting the height map to create ‘floors’ and ‘ceilings’ in their approach for procedural generation of solution cave models. Although this allows for the generation of caves via heightmaps, the resulting caves are uninteresting and the absolute reflection detracts from the realism. Although, using different heightmaps for ‘floors’ and ‘ceilings’, rather than a reflection, could prove fruitful.

(Dachsbacher and Stamminger, 2011) alter the paradigm of heightmap terrain generation, with the use of geometry images in place of heightmaps. Geometry images are the capture of surface geometry as a 2D array, with colours and normals also stored in similar 2D arrays using implicit surface parameterisation. This means any image operation performed on the geometry image also transfers to its geometry. Allowing for the down-filtering and upscaling of the terrain mesh. This is excellently complimented by their novel implementation of view-dependant level of detail rendering.

(Cui et al, 2011) present an approach for the procedural generation of caves. Their approach is centered around the creation of a base shape (e.g. sphere, cylinder), the surface of this shape is then perturbed by perlin noise to produce a visually believable cave structure. Their approach allows for various base shapes, which can be combined to form an entire cave network. However, this has been left for future work. With any noise based approach, the presence of “floating islands” and “air pockets” can be observed. Cui et al present a novel solution for the removal of these artifacts. They suggest using the common graphics algorithms scan-line for the removal of floating islands and flood-fill for air pockets.

Cellular Automata

Cellular Automata is often used in procedural content generation. Common uses include modelling of heat and fire, pressure and explosions, rain and fluid flow. (Johnson, 2010). Although, not as often used in the generation of terrain. It has been used in the generation of 2D caves (Johnson, 2010) and the self organising properties of Cellular Automata work well for erosion simulation (Olsen, 2004)

Declarative Approach

The declarative approach proposed by (Smelik, 2011) aimed to provide the lacking unifying modelling framework for procedural world generation. Their aim was to create a software which combined the strengths of semantic based modelling and procedural approaches, with integration of varying techniques into a complete and consistent world. The

target audience being designers with no in-depth knowledge of underlying procedural algorithms. Letting designers focus on “what they want” and to express it at a higher level of abstraction.

With the majority of current research looking at realism and performance of techniques. This was a needed step towards integrated solutions. Current procedural techniques require in-depth understanding of the underlying algorithm to effectively manipulate parameters, without resorting to a trial-and-error approach. To avoid this, the software should support a small number of intuitive parameters. It should allow iteration; fine-grained control and editing (Smelik, Framework). The proposed workflow includes the user sketching a coarse expression of their intention. With which the software generates a complete world, which may be edited further. The world is built of layers; urban, road, vegetation, water, earth. Which are all procedurally generated using existing methods and the interaction between these layers is handled by the software. During the entire generation process, the software performs “consistency maintenance”. This maintenance works to both prevent undesirable features, such as a river flowing through a building, and to ensure the characteristics of the users coarse sketch are preserved.

This approach supports experimentation and incremental improvement. However, most algorithms have several parameters which heavily influence the generated output. Finding well behaving values are difficult, and would be even more so when multiple procedural methods are combined. However, this is an important step for the shift from a *constructive* paradigm of procedural generation to a more accessible *declarative* paradigm.

Physics-based Simulations

Physics-based simulation procedural techniques allow for the generation of terrain which have been exposed to various morphological agents. Agents such as water and temperature. An anticipated consequence of which is the generation of highly realistic terrain, especially as a fractal terrain base is used in most approaches. However, simulation approaches are known to be notoriously slow. Often requiring hundreds to thousands of iterations to complete generation. Limiting their application to offline generation. Although, recent research shows promising results in the shifting of these algorithms to the GPU. (Mei et al 2007, Vanek et al 2011, Ahn et al 2007, St’ava et al, 2008)

Erosion

Erosion-based modelling is addressed by the seminal paper by (Musgrave et al). There are many forms of naturally occurring phenomena which can cause various changes to a landscape. However, these are difficult to describe mathematically and the complex math involved would not be feasibly computable in the generation of terrain. Thus, erosion was narrowed down to two types which could be easily represented, mathematically. This paper introduces a simple model for erosion-deposition algorithms for thermal and hydraulic erosion.

Thermal erosion is defined as material breaking loose and sliding down slopes to collect below. Hydraulic erosion is defined as transportation and deposition of dissolved material from one location to another. The primary focus of this paper was applicability of these algorithms, with physical correctness being secondary. This has been further developed with different algorithms presented in papers such as (Nagashima 1998, Chiba et al 1998, Benes and Forsbach 2002). In 2004, Olsen looked at the optimisation of the algorithms presented in (Musgrave et al). The result of which was a new algorithm. A speed optimised thermal erosion algorithm but modified to alter the terrain in a different way. This paper demonstrates well the concept of performance at the cost of geological correctness. And the inherent drawback of being limited size and detail of terrain remains, due to the exponential computation costs as these increase. (Olsen 2004)

Hydrology

The approach presented by (Genevaux 2013) is based on the observation that therrain in the real world is often centered around a river drainage network. And these networks subdivide the terrain into distinct sections (Rosgen 1994). Their aim was to design an algorithm which allows for the quick generation of controllable terrain whilst maintaining geological correctness.

Although this is also a physics-based simulation approach based on principles of hydrology, the computation overhead is avoided by not relying on complex mathematics. Instead, representing the river drainage network as a geometric graph.

The terrain is generated with the user optionally sketching the most important rivers on the terrain. As well as, terrain contours, river mouths, and “some input parameters”. From this, the algorithm generates a complete river drainage system. Using building blocks such as junctions, springs, and river trajectories. These blocks are inspired by the Rosgen classification (Rosgen 1994). Once the network is defined, the terrain is decomposed into a set of patches. The final output is controlled by the *river slope map* which controls the dendritic shape of the rivers and their type. And a *terrain slope map* which determines the locations of mountains and flatlands. Both of these maps can be user-defined or procedurally generated.

This approach generates, quickly, large terrain with distinct river networks which is difficult to achieve using traditional fractal and erosion-based approaches and guarantees hydrographic correctness. As an impression of performance, this approach can generate “several hundred square kilometers in a few seconds”. And through its large number of parameters, allows for a high level of control. However, incorrect usage of the parameters can lead to the generation of unnatural terrain. Although, this can be used in the generation of terrain in the production of fantasy/sci-fi games and movies.

Artificial Intelligence

Artificial Intelligence is a field which has made many, previously thought impossible, achievements in the recent years and the limits of this technology are unknown. Areas of application for this technology is only beginning to be understood. With the question often being asked; will Artificial Intelligence surpass human creativity? Some answer this question with; it is only a matter of time. (K Grace “when will AI exceed human”)

Search based

Search based procedural techniques apply evolutionary and other metaheuristic search algorithms in their approach for content generation. See (Togelius et al 2011) for a taxonomy and survey of search-based techniques used in procedural content generation. They work by ‘searching’ an entire design space until the answer, or a good enough answer, is found. Most Artificial Intelligence based techniques use a generate-and-test approach. This is where content is generated by the algorithm and is tested by some evaluation function assessing its *fitness*. If the *fitness* value is below a threshold; the generated content, or a part of, is either discarded or regenerated. This process repeats until a solution is found. This evaluation function is an integral part of a search based approach.

The evaluation functions can be in various forms. Two of which are direct and simulation based. Direct functions work by extracting features directly from the content and match them to a *fitness* value. This matching can be based on the designers intuition or some theory of player experience. Or feedback received directly from human players. Whereas, simulation based functions work by having an artificial agent experience the content and evaluate accordingly. These agents can be hand-coded or based on learned player behavioural models.

As evaluation functions are such an integral part of search-based approaches. And they often feature simulation, the computation required can be substantial. This means search-based approaches provide no guarantee of completion time. In addition to no guarantee that the generated output will be of good enough quality. This limits its application to areas of content generation where heavy constraints are in place. Although, placing heavy constraints leads to a reduction in diversity of content generation. They can help consistently inform output completion time and quality.

Answer set

(Smith et al, 2011) present an approach which proposes a shift in procedural content generation paradigm. Current procedural techniques can be seen as implicit in their declaration of the design space, (Smith et al, 2011) suggest we move to explicit declarations of design space. Allowing for the capture of the design space as an answer set program. Thus, allowing for “rapid definition and expressive sculpting of new generators”.

They achieve this through the coupling of declarative specification of the design space with powerful solvers in ASP. See (Smith et al, 2011) for full tutorial. They provide a widely applicable

framework for creating content generators, an aspect lacking in many procedural content generation areas. (Smith et al, 2011) believe what is lacking between their framework and a true "*generative space programming paradigm*" is an integrated way to apply procedural techniques where applicable. This is a common concern in this field but being addressed increasingly by papers such as (Smelik, Framework. Togelius et al 2011). They suggest future PCG research should look towards the integration of declarative, solver-based generators with other procedural generators in a way that preserves desirable aspects.

Machine learning

This is defined as Artificial Intelligence models trained on existing content (J, Montgomery 2016). Machine learning has recently seen an explosive resurgence, in the form of deep learning due to its capability and application for training models from existing data. (J.Schmidhuber, deep learning in neural networks).

Machine learning is a unique paradigm for procedural content generation, dissimilar to the *constructive* paradigm of procedural, simulation, and even search and solver approaches. Machine learning extracts data from existing content in order to create additional content. Whereas, the previously mentioned approaches, all algorithmically generate (Lorensen et al 1987) new content. (Summerville et al 2018)

The possibility of an Artificial Intelligence model that has learned from the data of real caves being used to generate hyper-realistic caves is intriguing. Unfortunately, high-fidelity scans of real caves are incredibly costly (Boggus et al) and with the amount of scans that would be needed to train the model. As well as, computation costs and engineers to manage and guide the model. (Lex GPT-3) Hence, such a model is unlikely, and rightfully so, to be developed.

An interesting application of machine learning is the paradigm of *player experience driven* approach proposed by (Yannakakis et al, 2011). Which proposes the use of machine learning to record player behaviour and response to generated content. And to use this data to generate content which can elicit the desired effect (Yannakakis et al, 2011). An action game where difficulty scales to each player's skills and habits, or a horror game that derives and faces you against your worst nightmare. However, many ethical considerations will need to be made. From a privacy perspective, as well as, preservation of mental health in the case of a horror game.

Summary

As procedural content generation develops further, performance and interactivity is expected to improve. And as more frameworks, methodologies, and data structures are formalised. It will allow for specialised and structured development.

The next major step should be the formalisation of a framework which integrates all existing techniques to develop complete products. Complemented by intuitive controls and non-intrusive

conflict management between techniques and maintains manual changes. And allows for manual creation and editing when required.

The future of procedural techniques seem to be simulation based techniques. Which, although lacking control, allow for the generation of realistic content. Given that the algorithms are implemented on the GPU, allowing for runtime simulation capability.

Procedural content generation is not widely used in the generation of content. This is likely due to the immaturity of the field. Once, the above mentioned goals are met, it is likely to see widespread adoption in content generation.

SOFTWARE

Requirements

The primary aim of this project was the procedural generation of caves. With a secondary aim of ensuring realism. The initial goal of the project was to experiment and develop several prototypes using different procedural approaches. And to pick one of these approaches and create a final cave network.

However, a major reduction in scope was required as a consequence of poor project management. Hence, the scope was minimised to the development of a single procedural approach. As well as a reduction from an entire cave network to the fragments of caves.

New requirements, informed by the literature review, were also considered. The generator should:

- Allow for user control of the generated output.
- Allow for iteration on the generated output.
- Allow for manual editing, if required.
- The algorithm should be reasonably performant.

Product requirements

What requirements have I decided to implement and why those requirements?

Upon consideration of the requirements informed by the literature review. User control of generated output and iteration on the generated output were chosen to be implemented. User control of the generated output is integral to avoid dependency on random seeds and allow for expression of intention. And iteration allows for continuous experimentation and development of the cave, experimentation is crucial in procedural generation in order to find well behaving values. Manual editing was ignored. As manual editing was desired by professional 3D modellers who felt restricted by procedural generation as compared to manual creation (Sketchaworld). And performance was chosen to be implemented by ensuring parallel processing possibility of the underlying algorithm.

Design

Software Design

Procedural technique

For the procedural generation technique to be developed. The Marching cubes algorithm was selected (Lorensen, Marching cubes). The other techniques under consideration were heightmap terrain generation (Boggus et al), Cellular Automata (Johnson, 2010) and erosion of fractal terrain (Musgrave, 1989. Olsen, 2004).

Heightmap terrain generation has an inherent lack of terrain features such as caves and overhangs. Although, this can be overcome through mirroring of heightmaps to create ‘floors’ and ‘ceilings’ (Boggus et al). The created terrain is uninteresting and the absolute reflection detracts from realism.

(Johnson, 2010) presented an intriguing method of 2D cave generation. The caves generated were interesting and performant due to the nature of Cellular Automata. However, the 3D application of Cellular Automata would likely result in sponge-like caves. Whilst these caves would be interesting, they would also be highly unrealistic.

As for erosion of fractal terrain (Musgrave, 1989. Olsen, 2004). This approach would create highly realistic terrain. However, implementing a cave system based on this approach is beyond the scope of this project.

The marching cubes algorithm allows for the generation of a polygonal mesh of a surface based on a scalar field. (Lorensen, Marching cubes). The values of the scalar field can be procedurally generated. And can be manipulated in order to create cave structures. A surface value is defined and each point is queried as to whether it is above or below the surface. As the name suggests, one cube at a time is considered. And a polygonal mesh is generated based on which points in that cube are above or below the surface level. This means there are 256 (2^8) possible configurations. However, only 14 unique triangle configurations, and symmetries of those.

For the output of the function it was determined by a combination of perlin noise (Perlin, 1985), distance from the center of the cave and decreasing global elevation the deeper the cave travelled.

Implementation

At the start of the project. The Marching cubes algorithm was implemented. The triangulation table for the algorithm was borrowed from (Lorensen, Marching cubes).
PC
Following this, flat surface levels were added to test the correctness of the marching cubes algorithm.

```
result = vector.y;
```

This was followed by the implementation of 2D noise into the function. This perturbed the terrain but featured repeating patterns.

```
result = vector.y + Mathf.PerlinNoise(vector, noiseConstant) * noiseAmplifier;
```

Then 3D noise was applied. The implementation of 3D noise, and major functions can be found in the Appendix.

```
result = perlin3D(pVector, noiseConstant + 0.1f) * noiseAmplifier;
```

And to create cave structures instead of terrains, the surface value was multiplied by -1. This essentially turned the terrain inside-out, creating a cave structure.

Following this, distance from the center of the cave passage was incorporated into the function. And a boost to surface value was given to points closer to the center.

```
result = distFromCenter(pVector) + (perlin3D(pVector, noiseConstant + 0.1f) * noiseAmplifier * -1);
```

Finally, decreasing global elevation based on cave depth was implemented. This was done by scaling the result from the earlier functions based on the distance of a point from either the center of passage (for gravity taper) or from center of the cave entrance (for erosion taper).

Testing

As the adjusted aim of the generator was to create a user controlled cave generator with intuitive parameters, that allows for the generation of diverse cave structures. Opening the generator up to testers and receiving feedback in the form of quantitative questionnaires on the above mentioned categories would give a real world assessment on the success of the generator.

Evaluation

It is clear that the project was met with setbacks resulting in drastic reductions to aims and objectives of the project. The ongoing COVID-19 pandemic contributed majorly to this setback. The project was initiated with well defined aims and objectives; and a concrete plan laid out. Which included thought out deliverables, milestones with a schedule in place. An iterative approach was chosen as the preferred method of development. It is believed that the project, although rough, produced an adequate procedural genevcave generator. On the basis of being a modular cave generator, the generated parts of which could later be combined to create entire cave networks. The aim of the generator was to allow for user control via intuitive

controls and support iterative development. Although the intuitiveness of the parameters remains to be seen. It has undoubtedly met its aim of allowing for user control and iterative development. And the marching cubes algorithm inherently allows for parallelisation, which would be of highest priority in future work.

The surface function, however, does require further experimentation and development in order to produce more realistic results. And an addition of flood-fill and scan-line algorithms for the removal of “air pockets” and “floating islands” (Cui et al, 2011).

REFERENCES

An image synthesizer | ACM SIGGRAPH Computer Graphics [WWW Document], n.d. URL https://dl.acm.org/doi/abs/10.1145/325165.325247 (accessed 8.17.20).

Boggus, M., Crawfis, R., Laboratories, D., Avenue, N., n.d. PROCEDURAL CREATION OF 3D SOLUTION CAVE MODELS 6.

Computer rendering of stochastic models | Communications of the ACM [WWW Document], n.d. URL https://dl.acm.org/doi/abs/10.1145/358523.358553 (accessed 8.17.20).

Cui, J., Chow, Y.-W., Zhang, M., 2011. A voxel-based octree construction approach for procedural cave generation 11.

Dachsbacher, C., Stamminger, M., n.d. Rendering Procedural Terrain by Geometry Image Warping 8.

Fletcher, D., Yue, Y., Kader, M.A., 2010. Challenges and Perspectives of Procedural Modelling and Effects, in: 2010 14th International Conference Information Visualisation. Presented at the 2010 14th International Conference Information Visualisation, pp. 543–550. https://doi.org/10.1109/IV.2010.80

Freedman, D.S., Mei, Z., Srinivasan, S.R., Berenson, G.S., Dietz, W.H., 2007. Cardiovascular Risk Factors and Excess Adiposity Among Overweight Children and Adolescents: The Bogalusa Heart Study. The Journal of Pediatrics 150, 12-17.e2. https://doi.org/10.1016/j.jpeds.2006.08.042Survey

Génevaux, J.-D., Galin, É., Guérin, E., Peytavie, A., Benes, B., 2013. Terrain generation using procedural models based on hydrology. ACM Trans. Graph. 32, 143:1–143:13. https://doi.org/10.1145/2461912.2461996

GPT-3 vs Human Brain, n.d.

Johnson, L., Yannakakis, G.N., Togelius, J., 2010. Cellular automata for real-time generation of infinite cave levels, in: Proceedings of the 2010 Workshop on Procedural Content Generation in Games, PCGames '10. Association for Computing Machinery, Monterey, California, pp. 1–4. https://doi.org/10.1145/1814256.1814266

Lorensen, W.E., Cline, H.E., 1987. Marching cubes: A high resolution 3D surface construction algorithm. SIGGRAPH Comput. Graph. 21, 163–169. https://doi.org/10.1145/37402.37422

Olsen, J., n.d. Realtime Procedural Terrain Generation 20.

Polygonising a scalar field (Marching Cubes) [WWW Document], n.d. URL http://paulbourke.net/geometry/polygonise/ (accessed 4.28.20).

Smelik, Ruben M, de Kraker, K.J., Groenewegen, S.A., Tutenel, T., Bidarra, R., n.d. A Survey of Procedural Methods for Terrain Modelling 10.

Smelik, R.M., Tutenel, T., de Kraker, K.J., Bidarra, R., 2011. A declarative approach to procedural modeling of virtual worlds. Computers & Graphics, Virtual Reality in Brazil 35, 352–363. https://doi.org/10.1016/j.cag.2010.11.011

Smelik, R M, Tutenel, T., de Kraker, K.J., Bidarra, R., n.d. A Proposal for a Procedural Terrain Modelling Framework 4.

Smith, A.M., Mateas, M., 2011. Answer Set Programming for Procedural Content Generation: A Design Space Approach. IEEE Transactions on Computational Intelligence and AI in Games 3, 187–200. https://doi.org/10.1109/TCIAIG.2011.2158545

Summerville, A., Snodgrass, S., Guzdial, M., Holmgård, C., Hoover, A.K., Isaksen, A., Nealen, A., Togelius, J., 2018. Procedural Content Generation via Machine Learning (PCGML). IEEE Transactions on Games 10, 257–270. https://doi.org/10.1109/TG.2018.2846639

The definition and rendering of terrain maps | Proceedings of the 13th annual conference on Computer graphics and interactive techniques [WWW Document], n.d. URL https://dl.acm.org/doi/abs/10.1145/15922.15890 (accessed 8.17.20).

The synthesis and rendering of eroded fractal terrains | ACM SIGGRAPH Computer Graphics [WWW Document], n.d. URL https://dl.acm.org/doi/abs/10.1145/74334.74337 (accessed 8.17.20).

Togelius, J., Shaker, N., Dormans, J., 2016. Grammars and L-systems with applications to vegetation and levels, in: Shaker, N., Togelius, J., Nelson, M.J. (Eds.), Procedural Content Generation in Games, Computational Synthesis and Creative Systems. Springer International Publishing, Cham, pp. 73–98. https://doi.org/10.1007/978-3-319-42716-4_5

Togelius, J., Yannakakis, G.N., Stanley, K.O., Browne, C., 2011. Search-Based Procedural Content Generation: A Taxonomy and Survey. IEEE Transactions on Computational Intelligence and AI in Games 3, 172–186. https://doi.org/10.1109/TCIAIG.2011.2148116

Yannakakis, G.N., Togelius, J., 2011. Experience-Driven Procedural Content Generation. IEEE Transactions on Affective Computing 2, 147–161. https://doi.org/10.1109/T-AFFC.2011.6

APPENDIX

Distance from center, 3D perlin noise, Main function

```
2 references
private float dist2points(UnityEngine.Vector3 a)
{
    // Simple tapering as you get further into the cave
    // if noise constant is less than 0.2f, then you get really smooth top halves
    float result = 0;

    if (!erosionTaper)
    {
        if (gravityTaper)
        {
            float centerY = center.y * (1.5f - (a.z / (zWidth*2)));
            result = (float)Math.Sqrt(Math.Pow(center.x - a.x, 2) + Math.Pow(centerY - a.y, 2));
            result *= distanceConstant + ((distanceConstant) * (a.z / zWidth));
        }
        else
        {
            result = (float)Math.Sqrt(Math.Pow(center.x - a.x, 2) + Math.Pow(center.y - a.y, 2));
            result *= distanceConstant + ((distanceConstant) * (a.z / zWidth));
        }
    }
    // erosion based distance (based on distance from cave entrance)
    else if (erosionTaper)
    {
        result = (float)Math.Sqrt(Math.Pow(center.x - a.x, 2) + Math.Pow(center.y - a.y, 2) + Math.Pow(center.z - a.z / (zWidth / zConstant), 2));
        result *= distanceConstant;
    }

    return result;
}
```

```
0 references
float Perlin3D(UnityEngine.Vector3 v, float noiseCon)
{
    // https://www.youtube.com/watch?v=Aga0TBJkchM&list=WL&index=36&t=0s
    float AB = Mathf.PerlinNoise(noiseCon * v.x, noiseCon * v.y);
    float BC = Mathf.PerlinNoise(noiseCon * v.y, noiseCon * v.z);
    float AC = Mathf.PerlinNoise(noiseCon * v.x, noiseCon * v.z);

    float BA = Mathf.PerlinNoise(noiseCon * v.y, noiseCon * v.x);
    float CB = Mathf.PerlinNoise(noiseCon * v.z, noiseCon * v.y);
    float CA = Mathf.PerlinNoise(noiseCon * v.z, noiseCon * v.x);

    float ABC = AB + BC + AC + BA + CB + CA;
    return ABC / 6f;
}
```

```
1 reference
private float myFunction(UnityEngine.Vector3 pVector)
{
    float result = 0;
    int yCeil = yWidth - 2;
    int xCeil = xWidth - 2;
    if ((pVector.y > yCeil || pVector.y < yWidth - yCeil || pVector.x > xCeil || pVector.x < xWidth - xCeil))
    {
        result = -1;
        return result;
    }
    else if (pVector.y < floor)
    {
        result = 50 - (dist2points(pVector) + Perlin3D(pVector, noiseConstant + 0.1f) * noiseAmplifier * -1);
        return result;
    }
    else
    {
        result = 50 - (dist2points(pVector) + Perlin3D(pVector, noiseConstant - (0.1f * (pVector.y/yWidth))) * noiseAmplifier * -1);
        return result;
    }
}
```