Creating 3D Procedural Content with Cellular Automata

Thomas Parker

Abstract

Procedural Content Generation using Cellular Automata has been researched and employed in a limited scope, mainly for 2D video games. The purpose of this paper is to extend that scope to 3D content generation, and showcase exciting possibilities for the future. Specifically, this paper will investigate potential methods for bringing Cellular Automata into this new scope and compare the results. First, two unique methods for generating terrain will be discussed and compared on a technical level. Next, the two methods will be visualized and analyzed from a usability/practicality standpoint. All methods for this work will be developed for the Unity game engine. The code for both generation and visualization will be included for reference.

Introduction

This section will provide a basic introduction to the relevant topics of this work, starting with fundamental concepts and ending with research methods. This will give the current research context and shed light on complexities readers may not be familiar with.

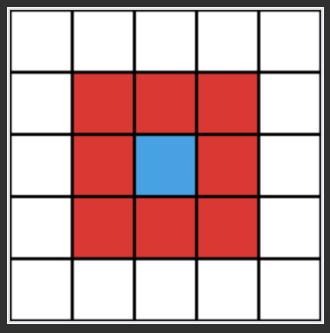
Cellular Automata

"This self-organizing structure consists of a grid of cells in any finite number of dimensions. Each cell has a reference to a set of cells that make up its neighborhood, and an initial state at zero time (t = 0). To calculate the state of a cell in the next generation (t + 1), a rule set is applied to the current state of the cell and the neighboring ones. After multiple generations, patterns may form in the grid, which are greatly dependent on the used rules and cell states" (van der Linden, Roland, et al., 2014, p79-80).

A Cellular Automaton, in simple terms, is a grid of cells (1D, 2D, 3D, etc). Each cell has a state, typically “Living” or “Dead”. The grid can updated any number of times by applying update rules to each cell. The rules for each cell determine its next state according to the state of its neighbors.

The Grid and Neighborhood

CA Grid (Wikipedia)



Moore Neighborhood (Wikipedia)

The right-most figure is known as a Moore Neighborhood, which considers a neighbor to be any cell directly adjacent or diagonal to the current cell. All proposed methods in this work will employ the Moore Neighborhood.

Rules

How a cell’s neighbors affect its state is determined by a set of simple rules. These rules often consider the number of living neighbors as the sole criteria. For example, a simple rule set would take the following form:

- Any dead cell will become living if it has X or more living neighbors
- Any living cell will become dead if it has less than Y or more than Z living neighbors

This format becomes complicated, however, when the possible states are expanded upon. Several Cellular Automata consider cells with many possible states. One proposed method in this work, for example, assigns each cell a real number between 0 and 1 as its state, which means an infinite set of possibilities. The rules for such cases are necessarily more complex.

The power of Cellular Automata comes from its unpredictable and complicated behavior, given simple rules and starting conditions.

"Complex systems such as CA produce global behaviour based on the interactions of simple units" (Burraston and Edmonds, 2005, p2).

Procedural Content Generation (PCG)

"Procedural content generation (PCG) is a term commonly used to describe the process of generating media content algorithmically rather than manually. Not only can PCG be used to alleviate the burden of the design process, but can also increase a game's replay value by generating new content to present to the player during gameplay" (Pech, Andrew, et al., 2015, p1).

PCG has been around in video games since the 1980's, and allows for a volume of content that is simply impossible to create by-hand. Popular video game Minecraft, for example, features a procedurally generated world on a scale much larger than Earth.

Within the realm of PCG, there are two important types. Online PCG refers to content generated on the user's machine in real time. As the game is being played, content is continuously being created. Offline PCG occurs on the developer-side (Shaker, et al., 2016, p7). The content is generated, reviewed, and added before the game is released. Online PCG allows for a unique player experience every time, whereas Offline PCG produces fixed assets that are constant. However, this allows for slower algorithms to be used, which would be impractical in real-time play.

Cellular Automata in PCG

The pairing between Cellular Automata's complex and interesting results and PCG have been exploited for quite some time. Most implementations so far have been applied to 2D cave generation, which produces realistic and usable cave levels very quickly. Before going further, it is important to note the drawbacks of using Cellular Automata. Although the complex behavior gives this method it's strength, it can act as a double-edged sword.

> "for both designers and programmers, it is not easy to fully understand the impact that a single parameter has on the generation process, since each parameter affects multiple features of the generated maps. It is not possible to create a map that has specific requirements, such as a given number of rooms with a certain connectivity. Therefore, gameplay features are somewhat disjoint from these control parameters. Any link between this generation method and gameplay features would have to be created through a process of trial and error" (Shaker, et al., 2016, p44).

Despite the challenge of translating Cellular Automata into desirable content, several successful cases exist.

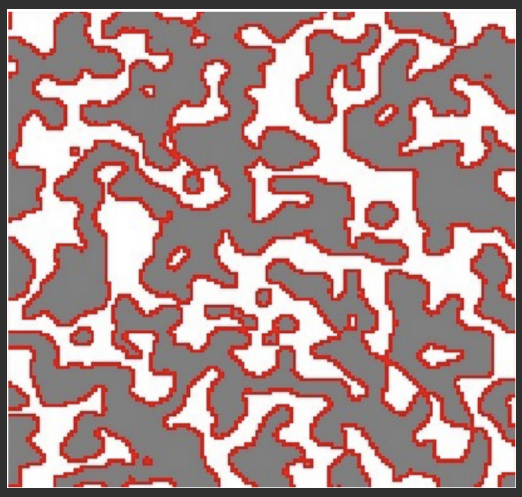
2D Dungeons with Cellular Automata

A substantial amount of research regarding Cellular Automata PCG for cave generation already exists. This section will briefly summarize the insights and results of that research.

In "Cellular automata for real-time generation of infinite cave levels", a method is described to generate a square 2D cave. The base cave is then expanded by generating square caves to the left, right, top, and bottom. The process is summarized as follows:

1. Each cell can have state “floor” or “rock”
2. Initialize a base grid of size 50x50
3. Initialize the cells with P probability of being in state “rock”, otherwise “floor”
4. Update the grid N times, according to the Cellular Automata rules
5. Every “rock” connected to at least one “floor” is considered a wall for the resulting dungeon
6. Apply cleaning algorithms that translate the wall states to an analogous 2D cave

The starting parameters can be chosen by the developer or computer, and influence the resulting cave structure. An example cave is shown below::



(Johnston, et al., 2010, p3)

Instances of Cellular Automata for 3D Content

The use of Cellular Automata for 3D content has seen little exploration so far. There, however, a few examples that are very relevant to this scope.

“Voxel Space Automata: Modeling with Stochastic Growth Processes in Voxel Space” presents a process for simulating 3D plant growth on objects. The results are very realistic, and provide an exciting example of the possibilities. Using a Voxel Space Automata, growth is simulated as well as lighting.

(Greene, 1989, p183)

This method is not strictly a form of cellular automata, however.

> "While both approaches are rule-based and operate on a matrix of cells, contrary to the spirit of cellular automata the implementation presented here is driven by geometry, and voxel representation functions primarily to simplify geometric operations. Nevertheless, some of the methods described are formally developed in the cellular automata literature, for example rules based on inspection of neighborhoods" (Greene, 1989, 177).

Although this method differs from Cellular Automata in meaningful ways, it is similar enough to prove the feasibility of such methods. It also bears some striking similarities to the 3D Cellular Automata method proposed in this work. Namely, the concept of voxels.

In "Cellular Automata for Real-time Generation of Infinite Cave Levels", two ideas for 3D cave levels are mentioned. The first is to take the generated 2D cave and add a third dimension with an additional process. Also proposed is the possibility of using 3D Cellular Automata directly, which will be explored in this work.

“The obvious next step is to generate 3D maps based on the 2D cave terrains created by the algorithm proposed here. In order to create a 3D world, one can project the tunnels generated along the y axis. Mid-point displacement algorithms such as diamond square or scatter noise could be then used to generate the walls of the tunnels. Alternatively, a designer could investigate the efficiency of 3D cellular automata for creating 3D maps directly, relying on cubic Moore neighborhoods” (Johnson, Lawrence, et al., 2010, p3).

Content Generation Methods Employed in This Work

“We conclude that the most promising challenges lie in the generation of complete dungeons for 3D games, preferably with extensive gameplay-based control. As discussed in Section VIII, current achievements show that these research goals are possible, although there are still quite some open challenges ahead” (van der Linden, et al., 2014, p88).

For 2D games, the translation from a 2D grid of cells to actual content is trivial. An issue arises, however, when we attempt to use 2D Cellular Automata in a 3D context. An approach for using 2D Cellular Automata in such a way will be covered, in the form of heightmap generation.

Though Cellular Automata are typically 2D, there is an analogous 3D representation. When we move to 3D Cellular Automata, there exists a much more intuitive mapping from data to 3D content. This comes at a price, however, as 3D Cellular Automata are more complex and computationally intensive.

Visualization Methods Employed in This Work

Because the purpose of these methods is to create visual results, it is necessary to discuss how visualizations are created. Among the several methods for drawing 3D surfaces to a computer screen, three have been chosen: heightmap terrains, voxels, and mesh generation.

Heightmaps are very straightforward to depict. Unity has built-in functionality for generating a 3D terrain from a 2D heightmap. Because heightmaps are 2D, they are highly compatible with 2D Cellular Automata.

For a 3D grid of cells, which may have caves and overhangs, more unique methods are required. The easiest method is to draw a cube on the screen for every living cell. The result is a terrain composed entirely of small cubes. This style has been made popular by games such as Minecraft.

A more sophisticated approach would be to create a 3D mesh, treating each living cell as a vertex for the rendered shape. For this approach, the well-known Marching Cubes Algorithm will be employed, as described by Paul Bourke in "Polygonising a scalar field"(Bourke, 1994, p1).

Methods For Using Cellular Automata in 3D

This section will provide an in-depth explanation of the PCG methods employed. In addition, each method will be reviewed in terms of results and computational cost.

Heightmaps

A heightmap is a 2D grid of numbers. Each number represents the height of the terrain at that point. For example, a heightmap of 0's represents a perfectly flat terrain at the lowest elevation. There are almost no well-established methods for heightmap generation using Cellular Automata. Therefore, a simple process has been designed for this work.

The method works like any regular Cellular Automata, except that instead of being alive or dead, each cell has a real number ranging from 0 to 1. (For example, a cell's state may be 0.0, 0.543, etc.) Each cell is updated based on the values of its neighbors. Below is a formal description of the process.

Generating an N x N heightmap with 2D Cellular Automata:

1. Initialize each cell with a random value between 0 and 1
2. To update the grid, perform the following process for each cell:
    a. Measure the value of all 8 neighboring cells, finding the maximum and minimum values
    b. If the current cell's value is not the maximum or the minimum:

$$newHeight = (currentHeight+maximum) / 2$$

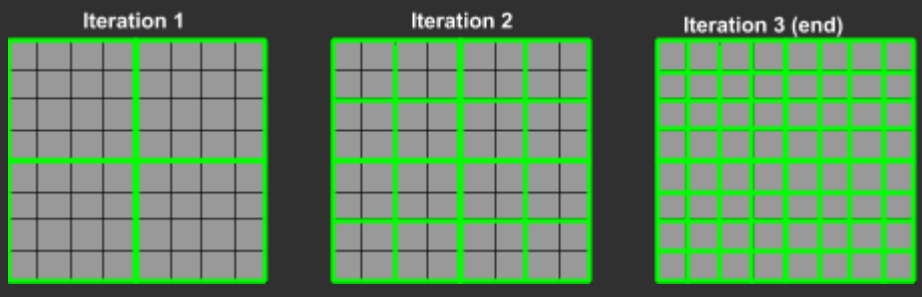
The idea behind this rule was to generate noisy, random points on the heightmap. Then, any relative minima or maxima would be fixed, and the intermediate values would be stretched between them to smooth the results. What follows are chaotic, terrain-like shapes. The terrains that emerge, however, are relatively flat. To add more interesting features such as large mountains and valleys to the terrain, the original algorithm is adapted.

To create large groups of cells that are higher or lower than others, we partition our grid and run the Cellular Automation on each quadrant as if it were a cell. We then repeat the process for smaller and smaller subdivisions of the grid. This concept was inspired by a similar method:

"The drawing area is then partitioned into quadrants and the automata recursively calls itself in each quadrant, making the appropriate transition for that quadrant. Each chain of downward transitions is separate, maintaining its own local values for both height and the accumulated decay parameter" (Ashlock and McGuiness, 2013, p2).

Updated Method With Partitioning:

1. Initialize each cell with a random value between 0 and 1
2. To update the grid, repeat the following process until the grid partitions are single cells.
    a. Divide the grid into subsections in the following manner:



    b. For each subdivision calculate the average height of all contained cells
    c. Run the original update algorithm treating each subdivision as a single cell (whose height is the computed average of it's contained cells)

d. Each partition will now have an updated height value.
    i. For each cell within the subdivision, add this height value to the cell's current height
e. Repeat process on next set of smaller subdivisions

The result of this more complicated algorithm is that large groups of cells are shifted at once. Smaller and smaller groups of cells are repeatedly updated. This introduces variations in height across larger areas, producing mountains and valleys. As the partitions get smaller, the jagged drop-offs between the initial quadrants are smoothed. An important note is that this subdivision process, from quadrants to single cells, is considered one grid update.

The heighmap method is the most efficient in terms of computation, as it only requires a single 2D grid. In addition to this, it arrives at desirable results after very few iterations. After numerous tweaking and experimenting, 4 grid updates was found to produce exceptional results, which will be shown later. The obvious trade-off for this speed is the inability to produce caves and overhangs. With optimizations in place, this method is feasible for Online PCG.

3D Cellular Automata

3D Cellular Automata operate on (and produce) a 3D grid of cells. The translation from this 3D grid to a terrain is straightforward: living cells form the shape of the terrain, dead cells are simply empty space.

Traditional 3D Cellular Automata work just like the 2D counterpart, except that cells now have several neighbors above and below them. The rules used for this method follow the form of the 2D version: Each cell is updated based on the number

of living neighbors. There is, however, one added feature. A cell's "height" (position on the z-axis) also affects how it is updated. The method is described below:

Terrain Generation with 3D Cellular Automata:

1. Start with an W x W x H grid of cells. Each cell has a probability P of being alive
2. Update the grid N times, applying the following rules to each cell:
    a. Measure cell height (position on z-axis)
        i. If height < 2, cell is "living"
        ii. If height = H-1 , cell is "dead"
        iii. If height > 10
            1. Generate random number between 0 and height+15
            2. If generated number > H, cell is "dead"
    b. Otherwise, compute nCount (number of living neighbors)
        i. If 4 < nCount < 8, cell is "living"
        ii. Otherwise, cell is "dead"

The parameters W , H , P, and N can be chosen by the developer for desired results. After numerous experiments, the parameters W=64, H=32, N=15, and P=50% were found to produce the best results. These results will be shown later.

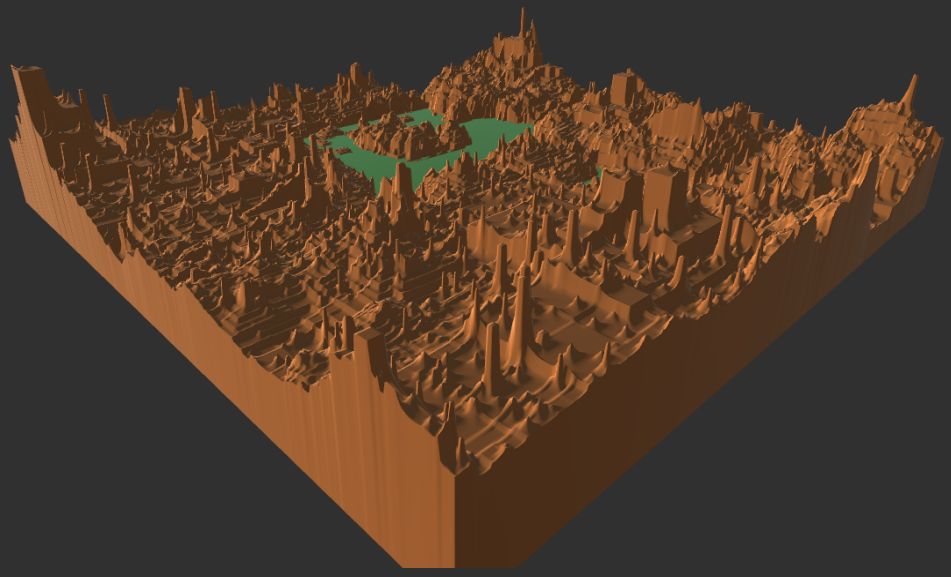
In terms of efficiency, the process takes much longer than heightmap generation, as updates are required across many more cells. The tradeoff for speed, however, is the possibility of caves and overhangs in the generated terrain.

One way to speed up this computation would be precompute a rule table (Pech, Andrew, et al. , 2015, p114). A rule table would be a mapping from every possible neighborhood to the corresponding cell state. This relieves much computation during the actual updates of the grid.

Visualizing the Terrain Data

Heightmaps

A heightmap was generated according to the method proposed earlier in this work. Unity then converted the heightmap into the following terrain, and a plane of water was added manually.



(128 x 128 cell grid, N=4 updates)

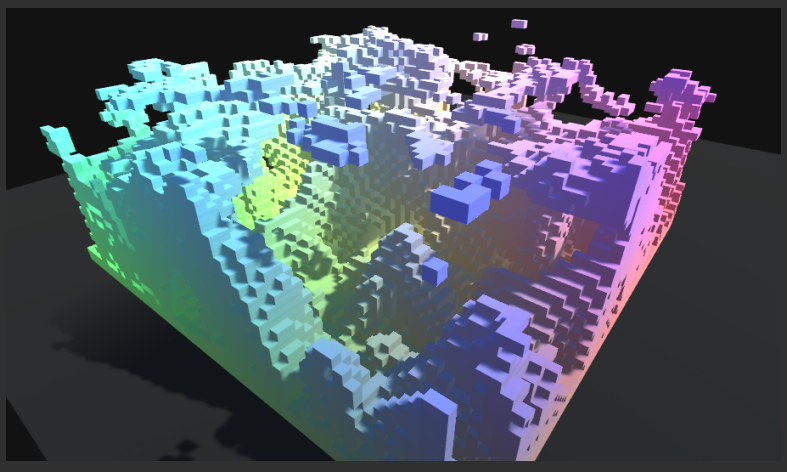
Heightmap generation is a very promising method for procedural content. Although it cannot produce caves or overhangs, it does produce very interesting results. It is by far the fastest of the methods, as it requires only a single 2D grid and achieves desirable results in only 4 updates. The results seen can likely be smoothed out with some minor adjustments to the method described.

3D Cellular Automata

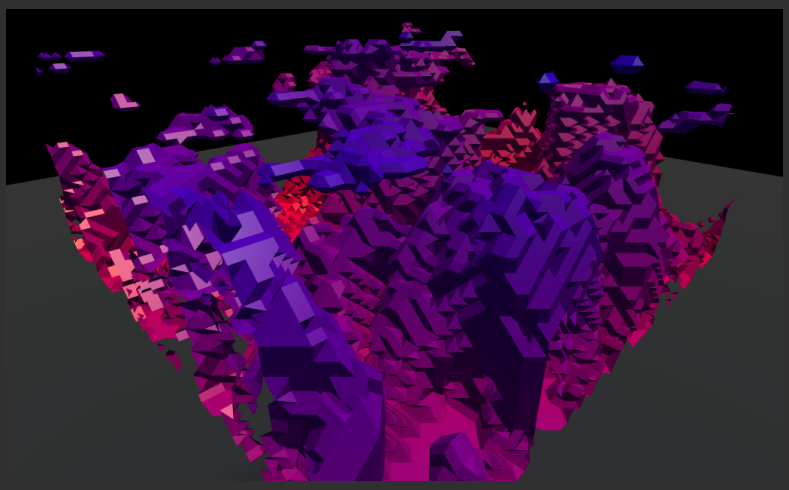
Below are two different example terrains generated from the proposed rules, one shown using voxels and the other with a mesh.

(W=64, H=32, N=15, P=50%)

Voxels



Marching Cubes Mesh



These results show promise for using 3D Cellular Automata. One important note is that several floating features are created, which are likely undesirable. A cleaning algorithm could be implemented to remove these features and further refine the terrain.

Conclusion

The possibilities for Cellular Automata in 3D PCG are exciting. There exist numerous ideas for extending and improving the proposed methods. Here are some possible methods not covered in this paper:

3D Accretor Cellular Automata

3D Accretor Cellular Automata are slightly more complex, and were specifically designed to generate art. The main differences are:

- Starting state is chosen, not random
- Physically impossible features are not generated
- Several states often employed

Although the starting state is not random for this form of Automata, randomness could be introduced. Introducing a random mutation of the starting state would be one approach. One desirable quality of this form is that no floating features are generated. This is achieved in the following manner:

"If the empty cell does not share at least 1 face with a neighbor stop processing the cell and go onto the next one. A simple if FaceCount<1 then continue check is all that is needed. This ensures the resulting structure is fully connected without cells "joined" at their diagonal corners. It also ensures that the resulting structure can be 3D printed" (Softology, 2018).

Additional states for 3D Cellular Automata

Currently, the proposed method uses two states. One represents filled space, and the other represents empty space. A possible expansion of this would be to have several states. Some representing emptiness, and others representing different forms of matter.

“The use of six states, only one representing empty space, was a choice made to give the algorithm greater expressive power. If a level map requiring only one type of rock is needed then the solid states can simply be merged into a single color. In many cases the obstructions of different color could represent different types of obstructions: rock, water features, pits, or even pools of lava” (Ashlock, 2015, p310).

Stacking a 2D Cellular Automaton

1D Cellular Automata have been employed to create 2D results in the past. This was achieved by layering each update under the previous one. This creates a 2D image that shows the starting state at the top, followed by it’s progression.

The same concept in 2D would create a 3D stack of shapes, each being one timestep in the automata’s evolution.

“The automata used in this study are one-dimensional with the complex images arising as a time history. While requiring a substantial increase in the amount of computation required it would be simple to generalize the system to three dimensions in which the image is a solid rendering of the time history of a two dimensional automata” (Ashlock, Tsang, 2009, p3344).

Layers of different 2D Cellular Automata

This method would employ several 2D Automata, layered on top of eachother. Each could be updated independently, according to a unique set of rules. This would introduce a high degree of nuance, at the cost of much more rules to manage.

“Another key area for further work is the CA data structure itself. At this point the data structure defines only one layer. An extension to n layers for each different set of cells would be an interesting research topic. The attribution of different layer behaviours, such as advanced corrosion, plastic and ceramic cracks, or paint peeling,would exponentially extend the palette choices for objects” (Gobron and Chiba, 1999, p156).

Additional Automata for Other Features

After the initial terrain is generated, subsequent automata could generate additional features. This could be trees, water, animals, etc. Even if the terrain is generated with 3D Cellular Automata, 2D Cellular Automata could be feasibly used for this step.

“Automata-layers are the main concept by which we propose improving the systems proposed so far. Their functioning is as simple as having additional cellular automata generating other features of the level such as vegetation, objects, roads, among other structures that could be modeled” (Yuri and Chaimowicz, 2017, p501).

Works Cited

Softology ***Accretor Cellular Automata | Softology's Blog.*** *https://softologyblog.wordpress.com/2018/01/12/accretor-cellular-automata/. 2018*, Accessed 1 Dec. 2019.

Arata, H., et al. ***"Free-Form Shape Modeling by 3D Cellular Automata."*** *Proceedings Shape Modeling International '99. International Conference on Shape Modeling and Applications*, 1999, pp. 242–47. *IEEE Xplore*, doi:*10.1109/SMA.1999.749346.*

Ashlock, Daniel. ***"Evolvable Fashion-Based Cellular Automata for Generating Cavern Systems."*** *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, 2015, pp. 306–13. *IEEE Xplore*, *doi:10.1109/CIG.2015.7317958.*

Ashlock, Daniel, and Cameron McGuinness. ***"Landscape Automata for Search Based Procedural Content Generation."*** *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, 2013, pp. 1–8. *IEEE Xplore*, doi:*10.1109/CIG.2013.6633619.*

Ashlock, Daniel, and Jeffrey Tsang. ***"Evolved Art via Control of Cellular Automata."*** *2009 IEEE Congress on Evolutionary Computation*, 2009, pp. 3338–44. *IEEE Xplore*, doi:*10.1109/CEC.2009.4983368.*

Burraston, Dave, and Ernest Edmonds. ***Cellular Automata in Generative Electronic Music and Sonic Art: A Historical and Technical Review. Digital Creativity.*** 2005.

Gobron, Stéphane, and Norishige Chiba. ***"3D Surface Cellular Automata and Their Applications."*** *The Journal of Visualization and Computer Animation*, vol. 10, no. 3, 1999, pp. 143–58. *Wiley Online Library*, doi:*10.1002/(SICI)1099-1778(199907/09)10:3<143::AID-VIS204>3.0.CO;2-W.*

Greene, N. ***"Voxel Space Automata: Modeling with Stochastic Growth Processes in Voxel Space."*** *Proceedings of the 16th Annual Conference on Computer Graphics*

and Interactive Techniques, ACM, 1989, pp. 175–184. *ACM Digital Library*, doi:*10.1145/74333.74351.*

Johnson, Lawrence, et al. ***"Cellular Automata for Real-Time Generation of Infinite Cave Levels."*** *Proceedings of the 2010 Workshop on Procedural Content Generation in Games - PCGames '10*, ACM Press, 2010, pp. 1–4. *DOI.org (Crossref)*, doi:*10.1145/1814256.1814266.*

Pech, Andrew, et al. ***"Evolving Cellular Automata for Maze Generation."*** *Artificial Life and Computational Intelligence*, edited by Stephan K. Chalup et al., Springer International Publishing, 2015, pp. 112–24. *Springer Link*, doi:*10.1007/978-3-319-14803-8_9.*

Pessoa Avelar Macedo, Yuri, and Luiz Chaimowicz. ***"Improving Procedural 2D Map Generation Based on Multi-Layered Cellular Automata and Hilbert Curves."*** *2017 16th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, IEEE, 2017, pp. 116–25. *DOI.org (Crossref)*, doi:*10.1109/SBGames.2017.00021.*

Bourke, Paul. ***Polygonising a Scalar Field (Marching Cubes).*** *http://paulbourke.net/geometry/polygonise/*. Accessed 1 Dec. 2019.

Shaker, Noor, et al. ***Procedural Content Generation in Games.*** Springer International Publishing, 2016. *DOI.org (Crossref)*, doi:*10.1007/978-3-319-42716-4.*

van der Linden, Roland, et al. ***"Procedural Generation of Dungeons."*** *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 1, Mar. 2014, pp. 78–89. *IEEE Xplore*, doi:*10.1109/TCIAIG.2013.2290371.*