Procedural Generation of Road Networks

**McIntyre, Lewis**

lewismcintyre@hotmail.com

# 1. Project Proposal

Procedural Content Generation (PCG) is the automatic creation of assets or terrain through use of algorithms combined with pre-set data. Generally, a successfully implementation of PCG has as little manual input as possible for the best diverse set achievable. Through its use in games like ‘Hades’, ‘Minecraft’ and ‘No Man’s Sky’, it provides a close to bottomless amount of content that would have been near impossible to implement by hand. These games also feature upmost quality and are idiosyncratic across all variations. In the case of ‘No Man’s Sky’, PCG allowed the creation of ’eighteen quintillion planets’ with only one designer (Murray, 2016). From a business perspective, the time and capital saved from recruiting fewer employees exemplifies the uses of PCG as budget can be a hugely limiting factor in development for most teams.

In this report, we will focus on using L-Systems to procedurally generate road networks of varying types, ranging from cities to highways connecting places of interest. The reason for undertaking this project is to produce high quality assets that could not easily be generated by hand and that could be used in a game setting. The network will be generated before the scene is loaded and will use a Stochastic model using parameterized vectors to give the designer better control of the final output. These vectors will allow the change of density and size of the road network, as well as the type of network the designer wishes to generate. It should be created optimally with as little time taken as possible. We will measure the efficiency by using the time the build started subtracted by the time when the final iteration has complete.

# 2.1 Road Generation Techniques

Techniques to procedurally generate road networks in virtual environments vary in both functionality and features. In this paper, we will briefly discuss a few of these techniques and look at the practical uses, afterwards moving on to the chosen approach for the implementation.

As stated in Procedural Content Generation through Quality Diversity (2019), procedurally generating content has minimum requirements on both the unpredictability and the quality of the output. If one of these are missing at any iteration, the quality of the final product is severely reduced. Many struggle to find the middle ground of increasing the changeability of the algorithm whilst keeping output natural. This paper suggests the use of ‘Quality Diversity’, which is an algorithm that has a set of criteria that must be hit, whilst also ensuring the branching of possibilities. The paper uses behavioural space to allow divergence whilst still sticking to a ruleset, the algorithm achieves this by taking into consideration the figurative ‘distance’ between each action and the type of behaviour it is. This paper shows the great uses, especially when generating levels for side-scrollers and bullet hells. It can be incredibly versatile towards most procedural generation cases. However, measuring the quality of each generation for a natural asset would be arduous, due to the need for multiple variables to be appended on top of each other. However, this does not take away from its uses in the procedural content generation sphere.

Another strategy in creating seemingly natural road networks includes the use of tensor fields. (Interactive Procedural Street Modeling, 2008) Guoning Chen suggests the similarity between tensor fields of varying shape (See figure 1.) and road networks. The paper suggests by combining tensor fields with global and local algorithms, that many of the general issues in generating a road network are solved. For example, the designer can edit the tensor field at any point of its generation, allowing it to change, delete and regenerate any single part at any time. It also has the capability to freely draw new roads that are then easily added to the network. Through the use of Perlin Noise, the field can be randomized to create much more unique networks. Road direction and position are created using eigenvectors and hyper-streamlines. By drawing along the densest part of the eigenvector, we get a rough idea of how the roads will be placed, then, using hyper-streamlines and road generation rules, the network is generated. This variation of generating a road network is incredibly versatile and designer-friendly however due to the time restraints of this project, it would be incredibly difficult to produce the results required.

A picture containing background pattern

Description automatically generated

(Figure )

Notmilad. 2008. Tensor Star. [Online Image] [Viewed 16/12/2022] Available from Wikimedia: https://commons.wikimedia.org/wiki/File:TensorStar.png

A final algorithm used to produce road networks is L-Systems, a grammar rewriting system that extends from the ‘Formal Grammar’ language theory. This algorithm uses an axiom, a set alphabet and grammar rules. For example,

Axiom – X

Alphabet – X, Y, Z

Rules – X=YZ, Y=XZ

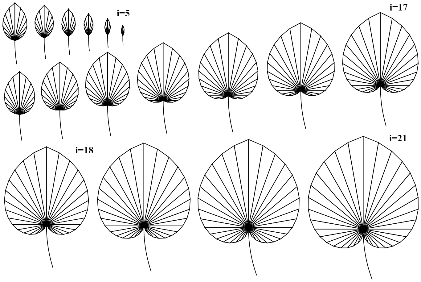
For each generation, the axiom would evolve using the rules provided. (See figure 2.)

|  |  |
| --- | --- |
| Generation (N) | Output |
| 0 | X |
| 1 | YZ |
| 2 | XZZ |
| 3 | YZZZ |
| 4 | XZZZZ |
| … | … |

(Figure )

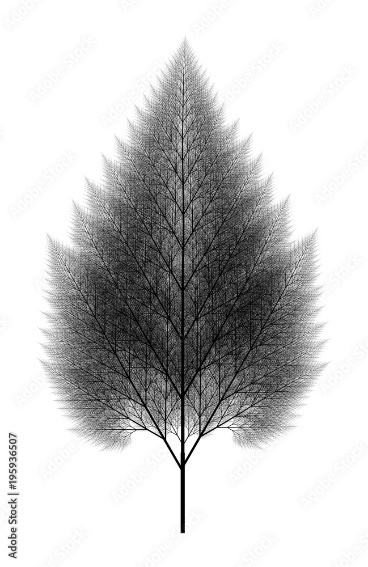
Table with example of L-System functionality

L-Systems or Lindenmayer systems are incredibly powerful in creating structured content and are incredibly versatile for the creation of assets. Generally, L-Systems seem to be the preferred route in terms of creating assets modelling the real world that are both realistic and natural, (See Figures 3, 4 and 5). However, a commonality between multiple papers involving L-Systems is the issue of control. By design L-Systems are not necessarily easy to describe or implement. Many create variations of L-Systems, such as Jormedals ‘Parametric L-System’ or Parish & Müllers ‘Extended L-System’. Allowing the use of more attributes that generalize effects on output can create a significantly more designer-friendly program. Through this we take that more control may increase development time, but is worth the improved output.



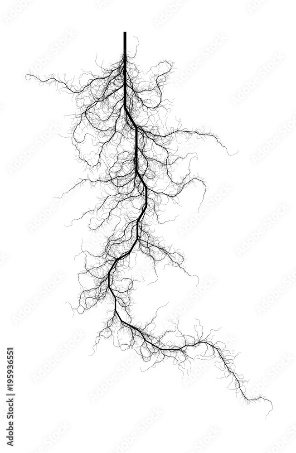
(Figure )

NightElfikk. 2011. L-System Leaf Grow.[Online Image] [Viewed 16/12/2022] Available from Wikimedia: https://commons.wikimedia.org/wiki/File:L-system-Leaf-grow.svg



(Figure )

kseniyaomega. 2022a. Tree Flat Computer Generated Self-Similar L-System. [Online Image] [Viewed 16th December 2022] Available from Adobe Stock: <https://stock.adobe.com/uk/images/flat-computer-generated-self-similar-l-system-Fbranching-tree-fractal-generative-art/195936507>



(Figure )

kseniyaomega. 2022b. Tree Flat Computer Generated Self-Similar L-System. [Online Image] [Viewed 16th December 2022] Available from Adobe Stock: <https://stock.adobe.com/images/flat-computer-generated-self-similar-l-system-branching-root-fractal-generative-art/195936551>

# 2.2 Types of L-System

Geospatial L-Systems have an incredible advantage in creating natural assets. As stated by A. Coelo et al in Expeditious Modelling of Virtual Urban Environments with Geospatial L-Systems’ (Coelho, 2007), these types of L-Systems associate geographic data to a selection of tuned parameters. Its advantage comes from both its reusability to take in any type of geographic data, but also due to its versatility in any situation. In their implementation, they could easily create pavements, buildings, and other geometric structures. A major issue in this implementation comes under the large bracket of overabundance of functionality. Geospatial L-Systems in their base form are incredibly vague leaving a lot of the functionailtiy left to each geographic data and its relative variables. For the purposes of this project, we will avoid using Geospatial L-Systems, due to both time factors and the lack of need for such a sophisticated implementation.

One approach in generating a realistic road network takes multiple iterations of a single system to include separate networks. According to the UKs Department for Transport (Transport, 2021), roads can be generalized into A-Roads, B-Roads, and C-Roads; as statistics these are 11%, 7% and 81% respectively. A-Roads account for major roads which cover vast distances linking important routes between cities. B-Roads on the other hand are used to micromanage traffic and intend to feed traffic between A-Roads and C-Roads. Finally, C-Roads are described as minor roads and consist of the roads connecting small towns and villages. These classifications are important in understanding the creation of natural road networks that also have useful functionality. A similar approach was used in Jormedals paper on ‘Procedural generation of road networks using L-Systems’ (Jormedal, 2013) in which the result had multiple iterations of varying types of networks to create a realistic scene. For example, Jormedal generated both a road network and a railway system in their report. He states that although this gave realistic looking results, it was quite tedious to append multiple generations on top of each other as it would be incredibly difficult to find the cause of any issues.

A final approach for realism can be achieved using a density map; in the case of Parish and Müller (Parish & Müller, 2019) they intended to model a city based on the combination of population, elevation, vegetation, and zone maps. This model is incredibly scalable as more maps can be added to achieve greater realism

# 3.1 L-System

In this paper we will use L-Systems for procedurally generating a road network. We use five separate actions, these are:

|  |  |
| --- | --- |
| ‘F’ | Draw; Moves forward and creates a new node. |
| ‘[‘ -. | Save; Pushes the current node to the queue |
| ‘]’ | Load; Pops the top node from the queue. |
| ‘+’ | Turn Right; a rotation defined by the angle given (90 degrees) |
| ‘-‘ | Turn Left; a rotation defined by the angle given (-90 degrees) |

L-Systems will generally include a lowercase ‘f’ representing move forward without drawing, however in the case of a road network, a space in between roads would look unnatural and therefore will be avoided.

The designer has controls over the length of each line, the angle of each rotation, the number of iterations, the rules, and the initial axiom. The length will be shortened after each line is placed. This is because we want to achieve a system similar to a real-world model, where motorways hold less area in total, but are longer in length (See Figure 11). All combined, this gives the designer freedom and control over the algorithm.

Figures 6 and 7 represent an example case described by Jormedal as ‘The Turtle System’, which uses the idea a turtle is following a set of instructions with a pen attached underneath it, a line is drawn everywhere the turtle walks (This is represented by Unity’s built in Line Renderer)

|  |  |
| --- | --- |
| Axiom | [F][-F][+F] |
| Rules | F -> F[+F]--F |
| Length | 5 |
| Iterations | 2 |
| Angle | 90 |

(Figure )

Table showing example of L-System combined with ‘The Turtle System’

Shape, arrow

Description automatically generated(Figure )

Visualization screenshot of Figure 6’s chart

We then work on top of this to stick to our stochastic model by adding randomized chance for each branch to be ignored. Using the LineRenderer tool is efficient, this build took <4ms. However, we would like to increase the complexity of the system by adding separate road types depending on a certain circumstance, for example when four roads collide from separate sides a four-way road should be generated.

We achieve this by instantiating an outline of the set of roads. After the entire network has been generated, we can loop through every single road that we have generated and place the corresponding road type in its place. We do this by finding the neighbours of the road to be fixed, then we can have separate cases for each type of road. A three-way road will have neighbours in three separate directions with its rotation defined by each direction, while with a case of four neighbours, it will always be the four-way road with no rotation. (See Figure 8)

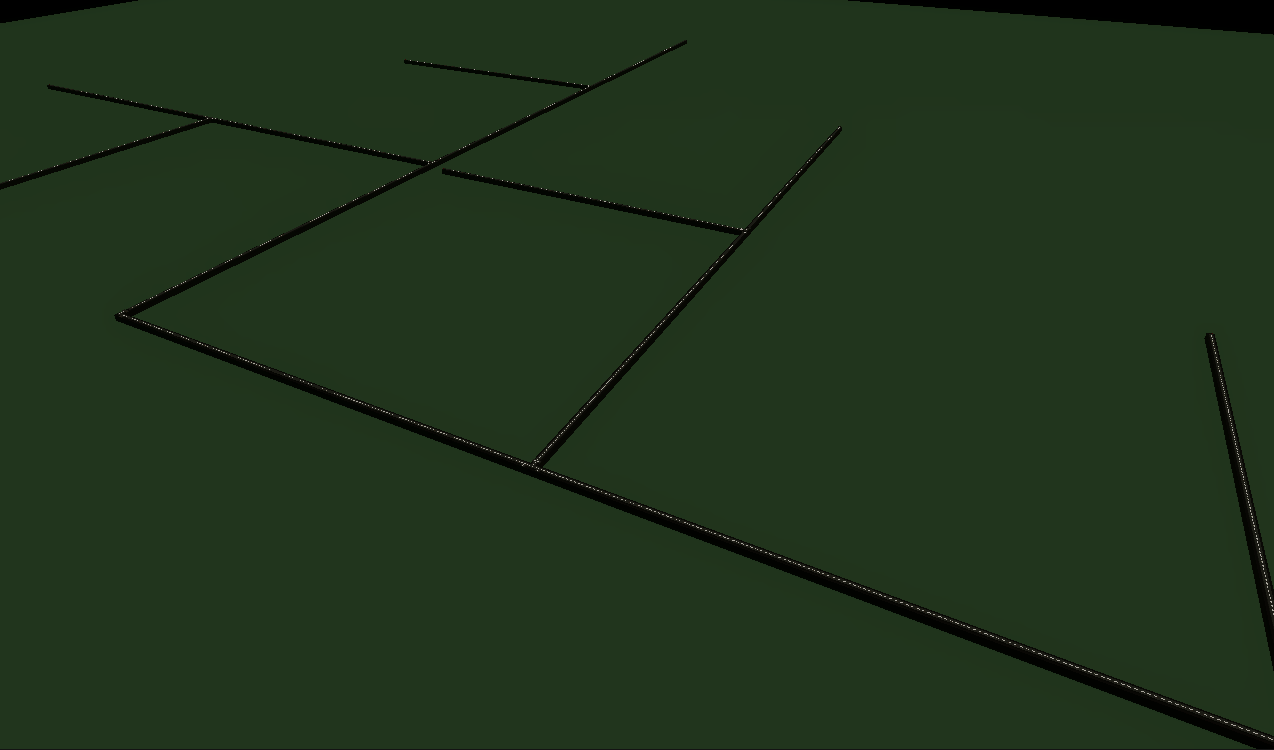
A close-up of a ferris wheel

Description automatically generated with medium confidence

(Figure )

Example of L-System Functionality

Depending on the implementation of L-System, it can have various levels of control due to the vast amount of content you can create with different rulesets. A ruleset with low iterations and high length can result in similar results as that of a highway whereas a ruleset with high iterations and low length can result in city-like landscapes. (See Figure 9 & 10) This can be expanded on further with angle and separate axioms to create almost any type of network.

Diagram, schematic

Description automatically generated

Figure Figure

Figures 9 & 10 both show in game screenshots of the different types of generators used,

# 3.2.1 Imposing Rules

With a basic form of L-Systems implemented with road prefabs, we can now begin to work on imposing rules to format the road network into a generator that is both manageable and follows the rules of a road. We can lay them out as follows.

1. Overlap in a road network comes in the form of separate intersections.
2. According to the UKs Department of Transport (Transport, 2021), 11% of roads should be classified as A-Roads (Motorways or carriageways), 7% should be classified as B-Roads (Connection between A and C-Roads) and finally 81% should be classified as C-Roads (Minor roads)
3. Roads by design fit a purpose, therefore an implementation which is accurate to a real-world planning process would be optimal.
4. Financial cost can be compared to Usage cost of each network generated, therefore efficiency is a top priority.
5. Roads with greater traffic tend to have larger surface areas, for example the M6 which covers 236 miles from Leicestershire to Dumfriesshire. (RAC Officials, 2018)

Diagram

Description automatically generated

(Figure )

Google. 2022. Google Maps - M6 Motorway. [Online Image] [Viewed 16/12/2022] Available from Google: https://www.google.com/maps

We use a HashMap (Dictionary in C#) to store the position as the key and tie its associated road piece. This implementation allows validation of each new piece added to the network. Each Road Prefab has a length and width of one, this creates a grid-like structure allowing the Dictionary to be searched using a Vector3Int. We will only ever instantiate a new road prefab if the Dictionary does not contain the current position. We will also look for an edge case where a road will be generated directly beside another whilst, facing the same direction. After all roads have been instantiated, we will loop through all roads placed and find all neighbour roads. We can then extrapolate the type of road required using neighbour count and type.

This workflow allows us to generate separate types of networks that combine into a grid like structure. We can create prefabs for City type generators and Highway type generators, then, with manual editing on top it will look similar to Figure 12

Diagram, schematic

Description automatically generated

(Figure )

Example of final product of generator with manual edits.

To setup a scene using these scripts, create a new scene and drag in the prefabs you would like to use. The starting position for each generator is shown by a plane that is removed at the start of runtime. You can adjust all values in the inspector as shown in Figure 13, or you can edit the rules which are stored as Scriptable Objects under the Rules folder. You are also able to create new rules by going to Create, Road Generator and then Rule. Each L-System Generator can store multiple rule sets that will be applied in order of each element in the array. You are also able to edit each type of roads prefab, all of which are available under the prefabs folder.

Graphical user interface

Description automatically generated

(Figure )

In-game view of the Inspector used to control the L-System.

# 4. Evaluation

Overall, this project was not a success. We set out to achieve a diverse set of realistic road networks and, although we can generate a large quantity of networks, we sacrifice way too much quality for it to be of value. The failure was due to many factors, mainly being time constraint however the scope of the project from the beginning was a huge concern. There was an overlap in the requirements for this project, certain rules set out in both the proposal and the literature review were unnecessary, mainly the need for such a large amount of control which ended up reducing the quality of each generation

The network as seen in Figure 12 took approximately 849ms to generate with an AMD Ryzen 6900HS processor. This is inefficient due to multiple factors, however most of these should have been foreseen and solved earlier on in the implementation. However, a decrease of 10000ms was seen after keeping only the base functionality for each road type. Since this project does not involve the use of a playable character, box colliders and rigid bodies were removed and only the base attributes for each road were stored

On the contrary, the designer is given an incredible amount of control with this system. We have stuck to the Stochastic model from the proposal by introducing a ‘Random Ignore Branch’ chance, however the parameterized vectors still provide designers control over the entire system. An issue of too little control has been seen in papers such as Procedural Modeling of Cities(Parish & Müller, 2019), however in this papers case, a major concern is the vast amount of control the designer has. Procedural Content Generation at its basic level should have as little designer input for the best diverse set of results achievable.

# 4.1 L-Systems

There were many difficulties in using L-Systems for road generation. With the goal of making the system as designer friendly as possible, the workload added up and the deadline increased the struggle to improve road efficiency. It can be said that L-Systems are successful tools used to generate road networks, as seen in Jormedals paper and their use of graph-based L-Systems. The use of ‘the turtle system’ appears simple at a low level however, when generating networks, it becomes vastly complex with numerous edge cases to account for.

# 4.2 Improvements

The first fix to aim for is efficiency in the regeneration of the network. Currently we’re storing every new road object in a dictionary, then afterwards looping back through it to validate road types. It would be interesting to find ways of reducing the number of roads replaced. This could be implemented by creating a new list of positions containing only roads to be fixed, then we can add the positions depending on certain characteristics of each piece placed. For example, we can assume that at the end of each road there will be a need to change its object to a Road End piece. We would repeat the same process with objects that have not been placed due to overlap.

Another improvement that could be made is to the assets used. Each road prefab consists of four or more unity cubes. This causes unseen parts of each prefab to be rendered without any extra value added to the network. A mesh for each type of road would significantly increase load times and would likely lessen RAM usage.

5. Reflection

In this paper I’ve learned a large amount of information on numerous topics. I’ve struggled with this project, mainly due to the lost ambition I felt after reading the latest on L-Systems and re-creating my whole project. I’ve enjoyed the hand-on side more than anything else, I tend to do large amounts of work at one time which I find is tricky to follow through with while also researching. I found myself recreating the same code multiple times hoping the result would be any better.

Demotivation became a big factor in my final performance. I wish I had researched more background knowledge of newer studies tackling road generation before starting my implementation. After reading a paper on tensor fields, I got distracted and spent way too much time looking into them to realize that it was way too late in the project to change my implementation. I’d like to say I’m passionate about road generation, but when I’ve found a better way of creating something I find it difficult to stay on course.

I enjoy all aspects of artificial intelligence for games, specifically asset creation. I have a passion for creating highly complex structures that not only look interesting but have a deeper functionality. I’ve always enjoyed sketching and painting, but I’ve never been talented enough to make anything of note. I would like to use the skills I have currently in programming to create aesthetic environments and assets.

I’ve learnt a lot about myself through this project, I’ve noticed that I’m a first-hand learner. I spent a lot of time reading every paper I could find on L-Systems, however I never got anywhere until I ended up starting my own implementation. I also spent a lot of time on the aesthetic of the game, the types of roads changed multiple times until I finally landed on a super simple design using mainly Unity cubes.

In the future I will take these three main points.

1. I should spend most of my time researching without implementation, as Abraham Lincoln said, “If I only had an hour to chop down a tree, I would spend the first 45 minutes sharpening my axe.”.
2. My second point is to not settle for a topic that you’re not motivated towards. Much of my best work has come from topics I’ve been incredibly passionate about, for instance this year in Games Preproduction I ended up choosing to spend my free time on development for the game, simply because I found it enjoyable. After the first two weeks on L-Systems my passion dropped and I wanted a new challenge, however I ended up sticking with it.
3. Finally, I’m going to improve my note-taking skills. Over this course, I’ve had five or six separate documents containing useful points I had for the entire project. None of the notes were organized and I ended up scrapping a lot of my ideas, simply because I had already written in its place without even seeing the notes.

# 6. References

Acknowledgements:

I’d like to give a special thank you to Hamid Homatash, throughout this course he’s been an incredible help and has always been available either in class or by email.

I’d also like to mention a lot of the code in this project was revised from Sunny Valley Studio, I ended up following the tutorial, then remaking the entire project on my own. I’ve worked on top of it by focusing more on the road generation and by changing many of the rules, adding hard-coded rules to stop multiple roads spawning next to each other and efficiency changes to the format such as using a HashSet instead of a HashMap(Dictionary) for the roads to be fixed.(Sunny Valley Studio, 2020)

Coelho, A. B. (2007). Expeditious Modelling of Virtual Urban Environments with Geospatial L-systems. *Computer Graphics Forum Volume 26, Issue 4*, 725-877.

Interactive Procedural Street Modeling. (2008). In T. a. SIGGRAPH, *ACM SIGGRAPH 2008 Papers* (pp. 1-10). New York: Association for Computing Machinery.

Jormedal, M. (2013). *Procedural generation of road networks using.* Linköping: Institutionen för systemteknik, Department of Electrical Engineering.

Murray, S. (2016, March 4). No Man's Sky - Interview Sean Murray. (A. Mai, Interviewer)

Notmilad. (2008, March 17). *Tensor Fields*. Retrieved from Wikimedia: https://commons.wikimedia.org/wiki/File:TensorStar.png

Parish, Y. I., & Müller, P. (2019). *Procedural Modeling of Cities.* Zürich: ETH Zürich University.

*Procedural Content Generation through Quality Diversity*. (2019, August 20). Retrieved from IEEE Xplore: https://ieeexplore.ieee.org/abstract/document/8848053

RAC Officials. (2018, August 28th). *Where are the longest roads in the UK?* Retrieved from RAC: https://www.rac.co.uk/drive/advice/know-how/longest-roads-in-the-uk/

Sunny Valley Studio. (2020, April 18). *Unity Procedural Town Tutorial*. Retrieved from YouTube: https://www.youtube.com/@SunnyValleyStudio/videos

Transport, D. f. (2021, March 10). *Road lengths in Great Britain* . Retrieved from gov.uk: https://www.gov.uk/government/statistics/road-lengths-in-great-britain-2021/road-lengths-in-great-britain-2021