Procedural Generation of Realistic Trees

Lauren Jefferson Pugh[1]

School of Computing Science, Newcastle University, UK

Abstract

In recent times, media projects are designing worlds with beautiful landscapes filled with nature. Trees play an important role in making these landscapes look realistic, however modelling them by hand can take a lot of time and effort. In this project, we look to investigate techniques and algorithm for procedurally generating realistic trees and discuss the method of implementing the space colonization algorithm into a program for tree generation.

Keywords: Tree, Procedural Generation, Space Colonization, L-System

1. Introduction

1.1 Motivation & Rationale

1.1.1 Context

Technology has dramatically improved in power and performance over recent years, and with that, the demand for realistic visuals in video games has also increased. Many games have answered the demand for realistic visuals designing beautiful landscapes filled with nature.

Trees play a very important part in making these landscapes looks realistic; if all trees are the exact same model, it won't look as realistic or immersive. Therefore, modelling trees for these games is a very time-consuming process.

1.1.2 Problem

There are a lot of tools available to assist with modelling trees for media such as video games and movies. However, even with those tools, it is still a process which is

[1] Email: l.pugh@ncl.ac.uk

extremely time-consuming. Algorithms can be used to procedural generate trees and reduce the time needed for modelling trees manually. However, there are multiple algorithms available for this, and each have their own advantages disadvantages. Therefore, in this paper, the pros and cons of the most common tree generating algorithms will be evaluated and a program will be developed to generate trees procedurally using the algorithm that is determined to be the most suitable.

1.2 Aims & Objectives

The main aim of this project is the develop a program which can assist in reducing the amount of time needed to model realistic trees to be used in media like video games and movies.

To achieve this goal, the aim is divided into the following objectives:

1. Identify algorithms for procedurally generating trees.

   This objective is to investigate existing algorithms for generating realistic trees, in order to find the most efficient algorithms (in terms of processing power and generation time) while also producing a realistic result. This objective will be achieved once an algorithm that is most suitable for the program has been identified.

2. Identify suitable engines for creating a program to procedurally generate trees.

   This objective includes researching many different engines (e.g., Unity) to identify an engine that will be most suitable for generating realistic trees procedurally. This objective will be achieved once existing engines have been analysed and a choice has been made for which engine will be used for the development of the program, based on the analysis.

3. Develop a program that will reduce the time needed for modelling trees by procedurally generating them, using the algorithm selected in objective 2.

   Using the engine identified in objective 2 develop a program which procedurally generates realistic trees using the algorithm that was identified as the most suitable in objective 1 and can be evaluated to compare the processing power and generation time.

4. Summarise and evaluate the performance of the program.

The final objective is to assess whether the final program was successful. This will be achieved by evaluating the generation time, the run-time performance, and the realism of the generated trees.

Following and completing these objectives should allow for the development of a program which can generate trees in a much quicker time frame than modelling the trees manually, while also being as realistic as a tree that was manually modelled.

2. Background Research

2.1 Tree Generation Algorithms

2.1.1 Space Colonization

> *"The space colonization algorithm treats competition for space as the key factor determining the branching structure of trees."* [1]

Space colonization is an algorithm that is used for iteratively growing networks of branching lines based on the placement of organic materials, like leaves and trees. This algorithm uses attractors and nodes to model the iterative growth of organic branching structures.

- Attractors – These are the resources which promote natural growth and is different for each branching system that is modelled. An example of an attractor would be sunlight, when modelling tree branches.
- Nodes – These are the points in which lines are drawn to render the branches of the model.

[2]

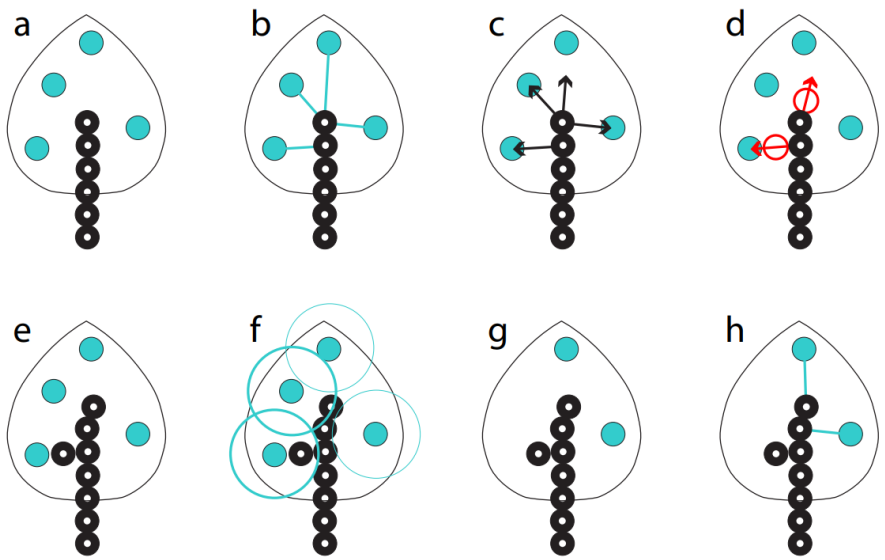

Figure 1: Diagram of the space colonization process [1]

Figure 1 shows a diagram of the steps of the space colonization process. The steps start with placing attractors around several nodes which make up the base of the model. The attractors influence specific nodes and, for each node that will be influenced, an average direction calculation is performed towards all of the attractors that are influencing it. Positions of the new nodes are calculated by normalising the

average direction to a unit vector, and then scaling it by a pre-defined segment length. Once the new nodes are placed, there is then a check to see if any of them are inside attractors' kill zones. If a node is in an attractor's kill zone, that attractor is removed. The process is then repeated from step *(b)*. [2]

In the space colonization process shown in figure 1, there are parameters which can be configured to achieve different effects:

- Attraction distance – only nodes within this radius around an attractor can be influenced by that attractor. A larger attraction distances means smoother and more subtle branch curves but will be costly to performance.
- Kill distance – this is the distance around an attractor which allows an attractor to be removed when at least one node is within it.
- Segment length – this is the distance between the nodes, as the network grows. A larger segment length will produce better performance, but the branches will not be as smooth.

[2]

Open and Closed Venation

There are two different growth types that can be used with the space colonization algorithm: open venation and closed venation. The 'open' and 'closed' refers to adjacent branches forming loops with each other. [2]

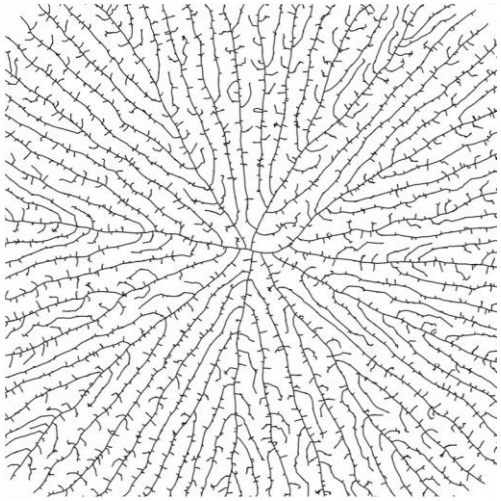

Figure 2: Open venation [2]

Open venation is the least computationally demanding growth type. Figure 1 shows open venation, as attractors are removed as soon as any node enters its kill distance. Figure 2 shows the results of using open venation to generate branches.

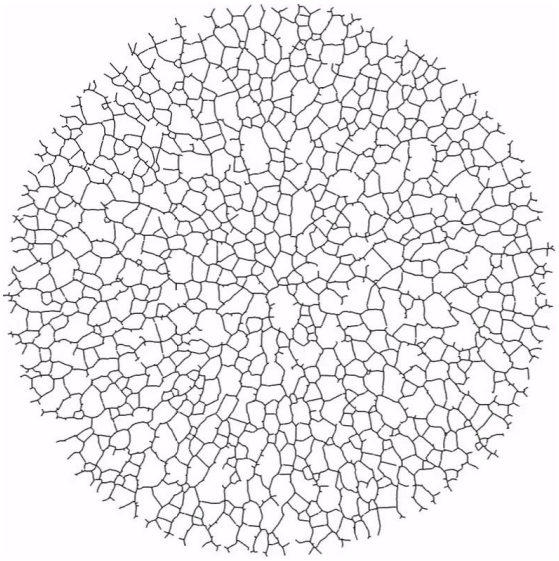

Figure 3: Closed venation [2]

For closed venation, each attractor is linked to all nodes within its attraction distance. Attractors are only removed when all of the nodes linked to it are within the kill distance of that attractor. This causes branches to connect and form loops, as shown in Figure 3.

2.1.2 L-Systems

Lindenmayer systems (L-systems) are a string-rewriting framework which were created as a way of formalising patterns of bacteria growth. Essentially, an L-system is a set of rules which describe how to transform a series of symbols. Each rule, known as a production, describes the transformation of one symbol. On each iteration of the algorithm, the productions are applied to each symbol simultaneously, resulting in a new series of symbols. Today, L-systems are most commonly used in computer graphics when visualising and simulating organic growth. [3]

Productions can be described as "before" and "after" states. For example, the production $\boldsymbol{a \rightarrow ab}$ represents that symbol $\boldsymbol{a}$ transforms into the symbols $\boldsymbol{ab}$ on every iteration. The number of iterations is represented by $\boldsymbol{n}$. [3]

Example of string rewriting with productions $\boldsymbol{a \rightarrow ab}$ and $\boldsymbol{b \rightarrow a}$ from $\boldsymbol{n}$ = 0 to $\boldsymbol{n}$ = 4:

a
ab
aba
abaab
abaababa

Formalising L-systems

L-systems are formalised as a tuple with the following definition:

$$G=\langle V,w,P\rangle$$

- V – the potential symbols in the string.
- w – the starting word, also known as the axiom, which is made up of symbols from V.
- P – a series of productions describing the transformations.

Example of a formalised L-system with productions $a \rightarrow ab$ and $b \rightarrow a$:

$w : a$
$P_1 : b \rightarrow a$
$P_2 : a \rightarrow ab$

[3]

Graphically Representing L-systems

Turtle graphics can be used to visually represent L-systems. Turtle graphics are vector graphics which are created by passing commands to a 'turtle' which will follow the commands and draw a line behind it as it follows the command. The type of commands that are passed to the turtle in turtle graphics include "move forward 5 spaces" and "turn left 90 degrees".

Symbols in L-systems each represent a command. After getting the result of an L-system, using its production rules, the strings can then be parsed from left to right and used to control the turtle in turtle graphics. [3]

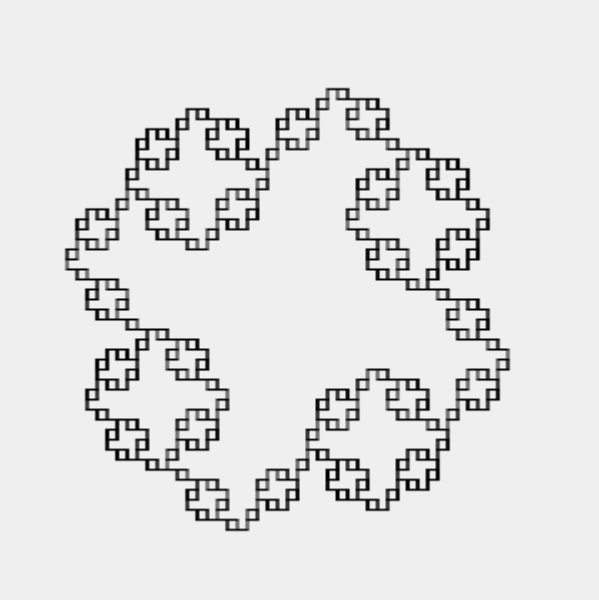

Figure 4: L-system represented with turtle graphics [4]

Figure 4 shows the result of 4 iterations an L-system. The formalised L-system of this example is:

w : A
P_1 : A → flflflf
P_2: f → fflflflflflfrf

The commands for the turtle in the example in Figure 4 are:

- ***r*** - turtle turns right by 90 degrees.
- ***l*** - turtle turns left by 90 degrees.
- ***f*** - turtle moves in the direction it's facing 5 spaces, drawing a line behind it.

2.1.3 Comparison of Space Colonization and L-Systems

The main advantage of the L-system algorithm is the speed of which it is able to generate trees and branches. This is very useful for programs which need to generate trees in real time. The reason for the algorithm's speed is that it does not care about branches that have already been created. Another advantage of the L-system algorithm is that it is easy to implement additional rules for trees and branches to follow. [5]

The main disadvantage of L-systems is that they recursively generate branches regardless of previous data. This causes a problem of intersecting branches. [6] Another problem with the L-system algorithm is that the models generated are often quite similar and would look very repetitive in a scene with a lot of trees. [7]

The main advantage of the space colonization algorithm is that it is good at simulating types of trees from specific climates, such as trees from temperate deciduous forests, which other modelling algorithms find difficult. Models generated with the space colonization algorithm are visually believable both with and without leaves. The space colonization algorithm is biologically justifiable. This is because, in nature, competition for space is mediated by quantity and quality of light. This competition has a significant impact on a plant's shape. The competition for space plays the dominant role in determining the branches structure of trees and shrubs. [1]

The main disadvantage of the space colonization algorithm is the computation time required to generate a tree. It takes between a few seconds and a few minutes to generate a tree on a 3.0 GHz processor (from 2007). The time depends strongly on the parameters used when generating the model. [1]

3. Design and Implementation

3.1 Design Choices

After gaining knowledge and understanding of different techniques for procedurally generating trees, the designing of the program was able to begin.

3.1.1 Algorithm

The decision was made to implement the space colonization algorithm in the program. The reason for this decision was that the research that was conducted showed that the space colonization algorithm produces the most realistic trees. Although the space colonization algorithm is not as quick or efficient as generating trees compared to L-Systems, the main aim of this project was to produce realistic trees so therefore that takes priority over the generation speed. The time it takes to generate realistic trees using the space colonization algorithm would still be quicker than manually modelling trees, therefore the aim to reduce time taken for modelling trees would still be met. The online resource 'Generating a 3D growing tree using a space colonization algorithm'[8] was used to assist with the development of the space colonization algorithm.

3.1.2 Engine

The engine that will be used to develop the application will the Unity. Unity is a powerful game engine which was developed by Unity Technologies in 2005. It supports over 25 platforms and can be used to render real-world graphics with minimum processing power. It is also extremely user friendly, well-documented, and contains a lot of tools which will be useful for developing this application. [9]

3.1.3 Scripting Language

C# was used as the scripting language, as it is the standard language for Unity. C# is a high-level, object-oriented, component-orientated programming language. It is an extension of the C programming language and was originally titled COOL, an acronym that stood for "C-like Object-Oriented Language". As it is an object-orientated language, it allows for code to be reused, saving time and effort.[10]

3.1.4 IDE

The IDE used for the development of this project was Visual Studio Code. Visual Studio Code is a lightweight editor that supports multiple programming languages. It uses Intelli-Sense to detect syntax errors and also gives suggestions on how to improve code snippets. [11]

3.2 Implementation

3.2.1 Distributing the leaves

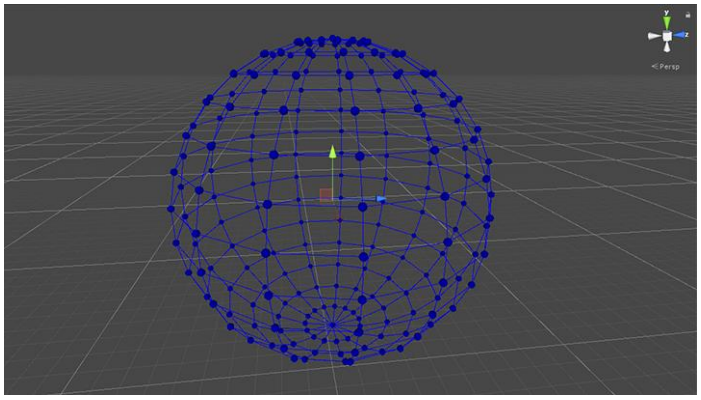

Figure 5: The spherical container which the leaves were generated within [8]

A sphere was used for the container of the leaves to enable easier distribution. Random positions for the leaves within the sphere were generated by randomising a direction vector and its magnitude.

$$\vec{D} = \begin{bmatrix} cos(\theta) \cdot sin(\alpha) \\ sin(\theta) \cdot sin(\alpha) \\ cos(\alpha) \end{bmatrix}$$

Figure 6: Direction vector generation [8]

The direction vector was computed by generating two random angles and passing them into the equation, shown in figure 6. The magnitude used was a random value generated between 0 and the radius of the spherical container used. The position of the leave was then determined by multiplying the direction vector by the magnitude.

Figure 7: Generation parameters for 'radius' of the spherical volume and 'Nb Attractors' (leaves)

A shown in figure 7, the number of leaves and the radius of the spherical volume containing the leaves is able to be adjusted to produce different results.

3.2.2 Branches

```
public class Branch {
    public Vector3 _start;
    public Vector3 _end;
    public Vector3 _direction;
    public Branch _parent;
    public float _size;
    public float _lastSize;
    public List<Branch> _children = new List<Branch>();
    public List<Vector3> _attractors = new List<Vector3>();
    public int _verticesId; // the index of the vertices within the vertices array
    public int _distanceFromRoot = 0;
    public bool _grown = false;

    public Branch(Vector3 start, Vector3 end, Vector3 direction, Branch parent = null) {
        _start = start;
        _end = end;
        _direction = direction;
        _parent = parent;
    }
}
```

Figure 8: Branch class

Figure 8 shows the branch class used in the implementation of the algorithm. Each branch stores its parent and its children to allow for navigation through the branches. It also stores its direction so that when it is needed it won't need to be computed again, therefore saving time and computational power. When the branches are growing, they are attracted by some leaves, therefore the leaves it is attracted to are also stored in a list.

On each iteration of the algorithm, branches are added on the furthest points of the existing branches within the attraction distance of one or more leaves. All of the branches of the tree are stored in a list. The initial branch is added underneath the spherical volume, to act as the trunk of the tree, and is used as the initial branch for the space colonization algorithm.

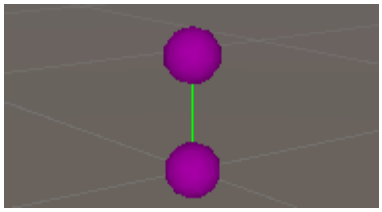

Figure 9: The initial branch

Before the initialisation, the start position and length of branch parameters were set. For this test the starting position was set to (0,0,0) and the length of the branch set to 0.2. Figure 9 shows the results of these parameters.

3.2.3 Growing Branches

When growing the branches, they need to be attracted by the leaves in which they are in the attraction distance of. An attraction range of 2.0 was set as the initial value. On each iteration, for each branch, the distance between the end point of the branch and the leaf was calculated. If the furthest point of the branch is within the attraction distance of a leaf, the leaf is added as an attraction point to the branch. To enable the tree branches to split in multiple directions, multiple attraction points are checked to see if the furthest point of the branch is within their attraction distance. For each attraction point that the branch is within the attraction distance of, a child branch will be created in the direction to the attraction point. To make the trees look more realistic, a randomness factor was added to stop the branches growing in completely straight lines. To do this, the branch direction vector is added to a normalised random vector multiplied by the randomness factor constant.

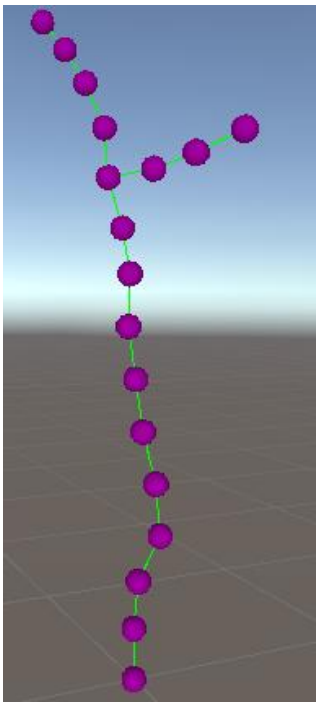

Figure 10: Result of multiple child branches and a randomness factor of 0.147

To stop the branches constantly growing towards attraction points infinitely, kill zones needed to be added. When a branch is within the kill distance of an attractor, the attractor will be removed from the list of attraction points. The kill distance was set to be lower than the attraction range but longer than the branch length.

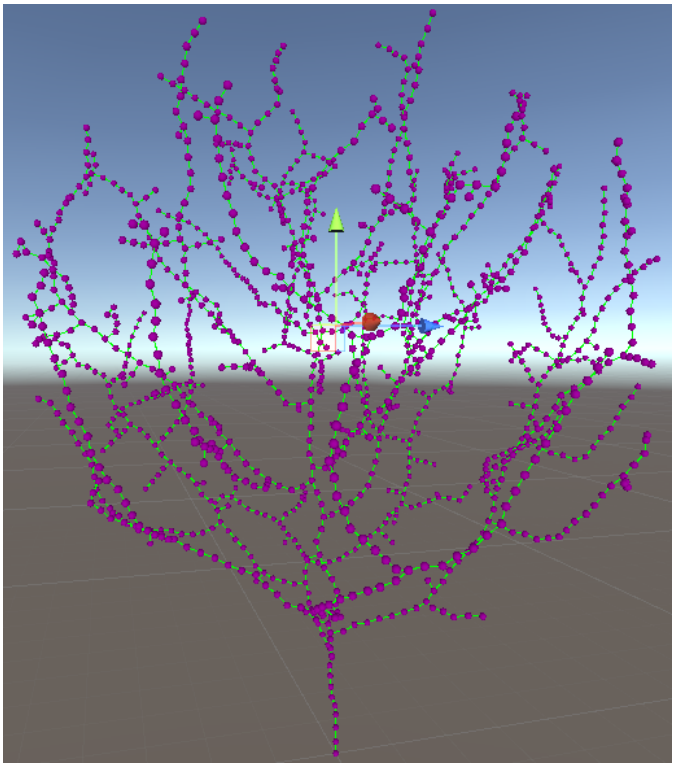

Figure 11: Branch graph generation using kill zones

3.2.4 Branch Mesh

To be able to use textures and shaders on the tree graph, a mesh was needed. For each branch, *S* vertices are created around the furthest point of each branch and along the branch direction. The distance of the vertices from the branch graph was determined using a radius constant. Faces are created using the vertices from that branch to its parent's vertices. The number of vertices (N_v) and number of faces (N_f) are shown in the following equations, where the number of branches is N_b.

$$N_v = (N_b + 1) \cdot S$$

$$N_f = N_b \cdot S \cdot 2$$

Figure 12: Equations for number of faces and number of vertices [8]

[8]

Figure 13 shows the results of this mesh generation.

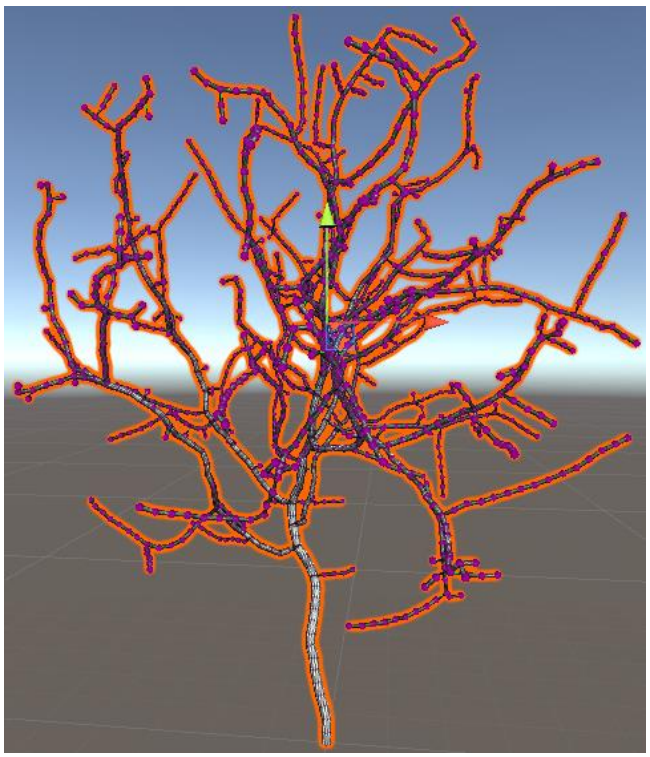

Figure 13: Branch mesh generation

3.2.5 Different Thickness Branches

To make the trees look more realistic, branches of different thicknesses were needed. The further away the branches from the starting position, the thinner they should be, and the closer to the starting position the branches are the thicker they should be. This would mean that the trunk of the tree would be the thickest branch. To do this, the list of branches was used to traverse from the end to the beginning, calculating the thickness of the branches using an invert growth. This size of the branch (s_b) is calculated using the following equations where r_e is the size of the furthest point of a branch.

If a branch has no children, it is a furthest point, therefore:

$$s_b = r_e$$

If a branch does have child branches, the size of the branch is calculated; where NB_{ch} is the number of children that the branch has, *children[]* is the array of children for that branch, and I_g is the invert growth factor:

$$s_b = (\sum_{n=1}^{Nb_{ch}} children[n].size^{I_g})^{\frac{1}{I_g}} \quad [8]$$

Figure 14 shows the results of this, using an invert growth factor of 2.41.

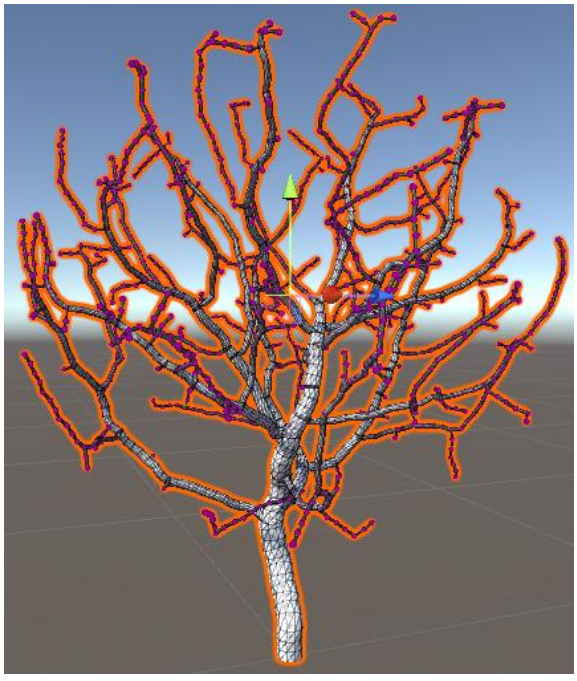

Figure 14: Branch mesh generation with an invert growth factor of 2.41

4. Evaluation

4.1 Parameter Tweaking

This section discusses the results when using different parameters.

4.1.1 Nb Attractors

Reducing the Nb Attractors parameter produces fewer desirable results, if too low it will cause branches to grow infinitely in strange directions.

Figure 15: Generated tree when Nb Attractors = 100

Increasing the Nb Attractors parameter gives more desirable results, however the more it is increased the longer the generation time.

Figure 16: Generated tree when Nb Attractors = 1000

4.1.2 Sphere Volume Radius

Reducing the radius of the sphere volume for the attractors too much causes undesirable results where the branches only frow at the very top of the tree, as shown in figure 17 below.

Figure 17: Generated tree when radius=0

Increasing the radius of the sphere volume too many causes the branches to grow infinitely, as shown in figure 18 below.

Figure 18: Generated tree when radius = 10

For desirable results, the radius should be kept in the range of 4 to 5.

4.1.3 Branch Length

Reducing the branch length parameter generates desirable looking trees, however reducing it too much causes slower generation times as the program has to perform more iterations before the branches enter the kill zones of the attractors. Increasing the branch length causes more unnatural results, however the generation time is must faster.

Figure 19: Generated tree when branch length = 0.05

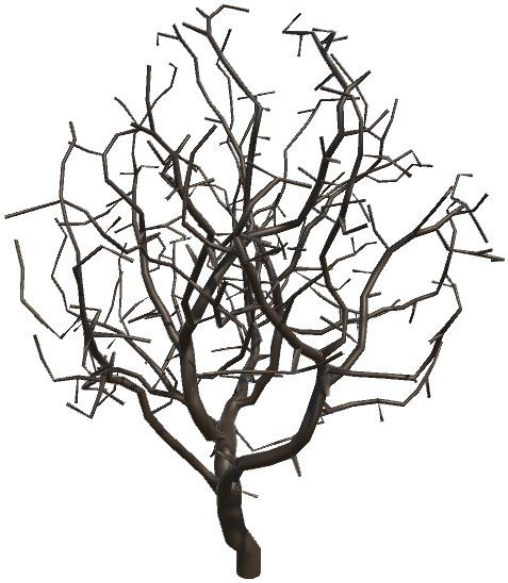

Figure 20: Generated Tree when branch length = 0.5

4.1.4 Attraction Range

If the attraction range is set to be too low, it will cause one long branch to grow, producing an undesirable result.

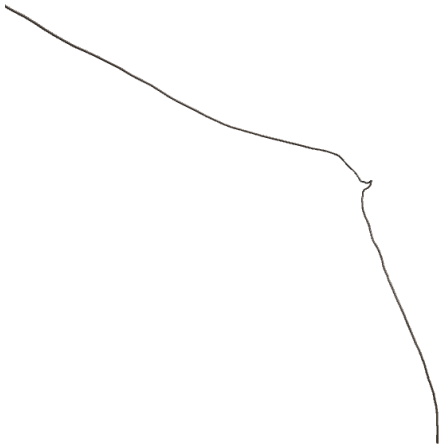

Figure 21: Generated tree when attraction range = 0.5

The higher the attraction range, the more branches that will be affected by it. Increasing the attraction range creates much more desirable results. As shown by Figure 22 below.

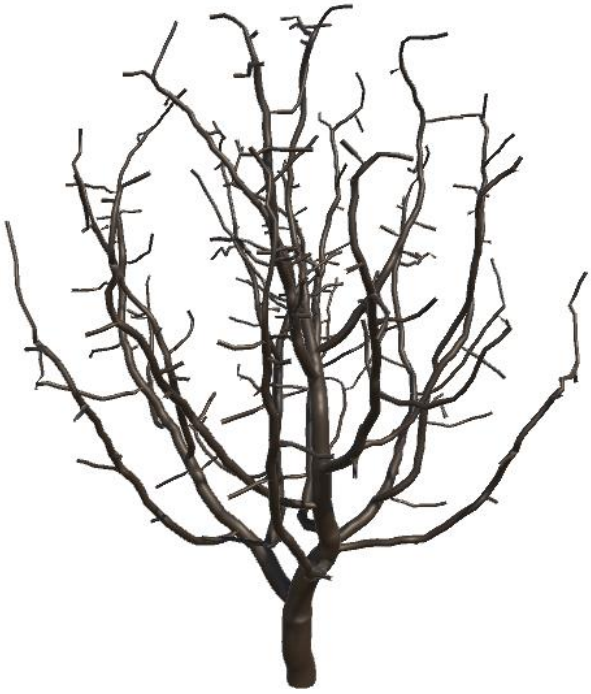

Figure 22: Generated tree when attraction range = 3

4.1.5 Kill Range

Setting the kill range parameter too high will cause one branch to grow infinitely, as all the attractors will be removed immediately.

Figure 23: Generated tree when kill range = 2

Setting the kill range too low will also cause undesirable results. It will cause the branches to continuously grow around the attractors as shown by figure 24 below.

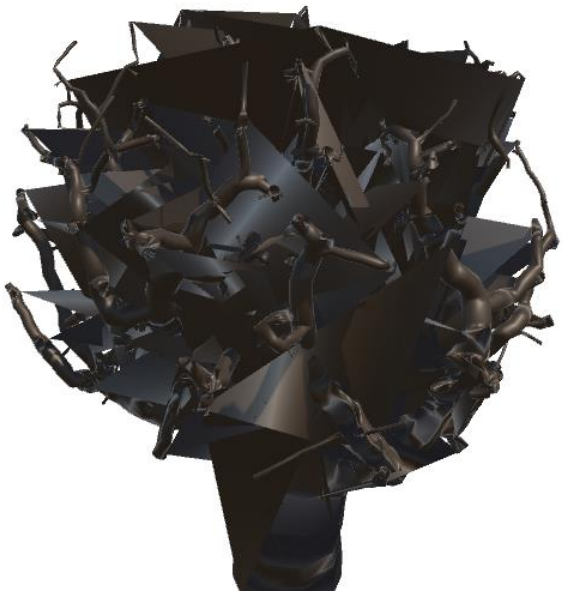

Figure 24: Generated tree when kill range = 0.05

The kill range should be lower than the attraction range and larger than the branch length for desirable results.

4.1.6 Invert Growth

Setting the invert growth parameter too high will cause the generated tree branches to all be similar thicknesses. This is shown in figure 25.

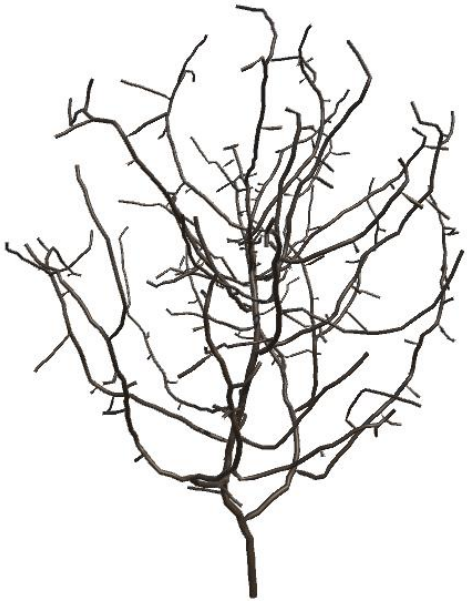

Figure 25: Generated tree when invert growth = 5

Setting the invert growth parameter too low will cause extremely undesirable results, as the trunk of the tree will become extremely wide. This is shown in figure 26.

Figure 26: Generated tree when invert growth = 1

For desirable results, the invert growth parameter should be greater than 2.

4.2 Data Analysis

Not only was the realism of the trees that were generated important, but the performance of the program was also important. To analyse the performance of the program, a range of data was collected. The average tree generation time was analysed within a given number of attractors. To ensure accurate results, the underlying framework of the application was kept the same; the only thing changed was the number of attractors. Figure 27 shows the parameters that were kept the same during the analysis.

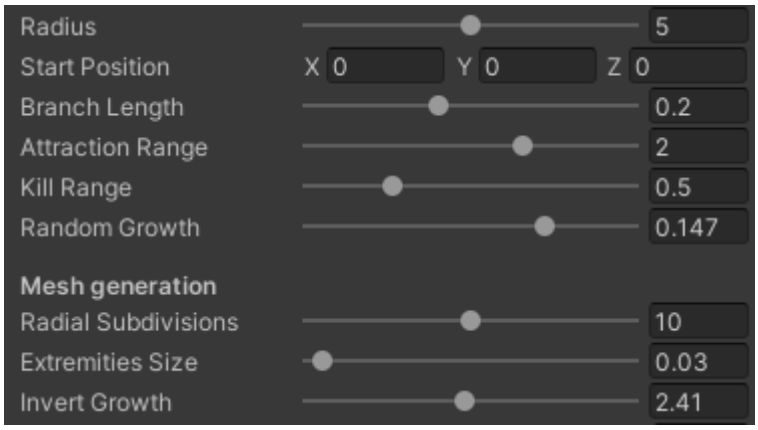

Figure 27: Parameters used for the analysis

The results are presented in the form of tables and graphs, with figures of the generated trees.

	Generation Time (s)					
Attractors	Run 1	Run 2	Run 3	Run 4	Run 5	Average
300	2.306117	2.168764	2.358751	2.187482	2.216715	**2.24757**
600	2.413493	2.410944	2.426583	2.442411	2.508196	**2.44033**
900	2.787472	2.791797	2.89171	2.791065	2.679782	**2.78837**
1200	3.160788	3.099884	3.274865	3.269638	3.181691	**3.19737**
1500	3.523603	3.473064	3.693347	3.500559	3.617037	**3.56152**
1800	3.990574	4.067004	3.982015	3.913623	4.018132	**3.99427**
2100	4.425501	4.499617	4.346182	4.421669	4.396477	**4.41789**
2400	4.840498	4.83825	4.986872	4.944571	4.981865	**4.91841**
2700	5.304573	5.375334	5.360233	5.431627	5.291136	**5.35258**
3000	5.77221	5.900846	5.800119	5.888273	5.997758	**5.87184**

Figure 28: Table to show the change in generation time when increasing the number of attractors

Figure 28 shows the results of the performance analysis. The tests were ran 5 times and an average was taken to ensure that the results were as accurate as possible, and to tule out any possible outliers. Figure 29 shows the graph that was created using the averages and the number of attractors.

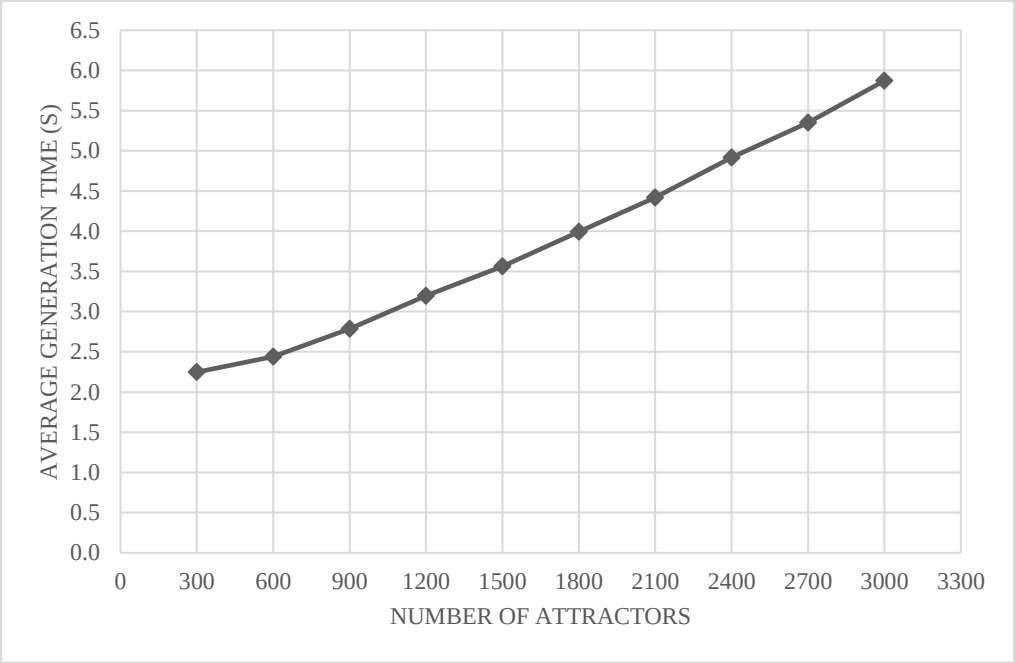

Figure 29: Graph to show the average generation time for different numbers of attractors

The graph in figure 29 shows a linear increase in time, as the number of attractors increase. Therefore, the trees have a faster generation time when less attractors are used in the generation. While the program has its limitations for generating trees using more attractors, the visual difference after 1500 attractors is not that much different, as shown in figure 30. If the user wanted to use 3000 attractors for generating their trees, an average of 5.9 seconds is still much faster than the amount of time it would take for them to model the tree manually.

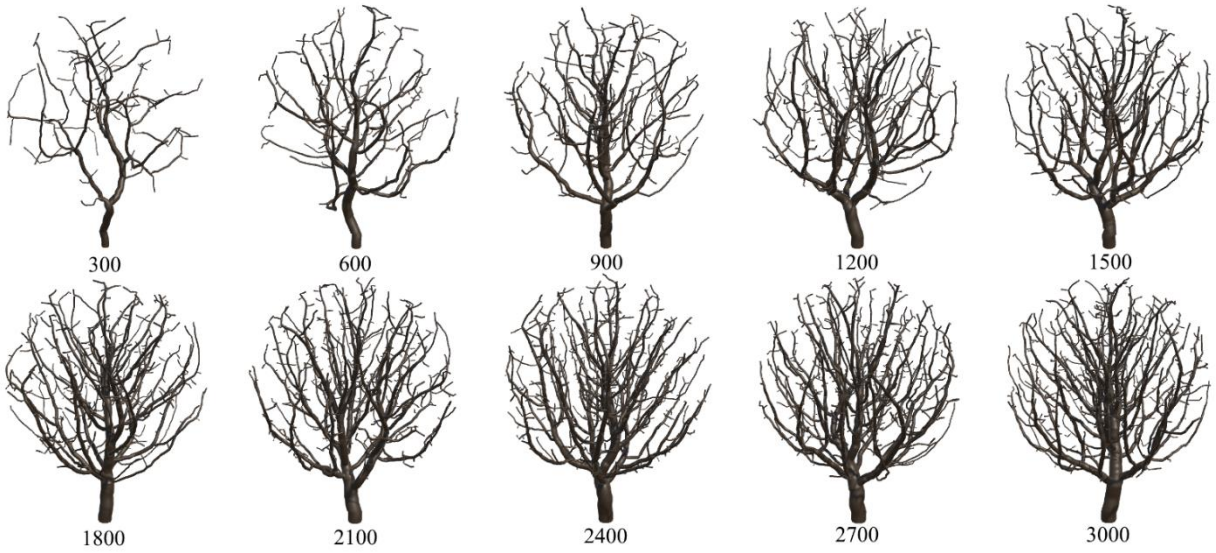

Figure 30: Performance analysis generated trees

Figure 30 shows the generated trees for each number of attractors that were tested. From the results it is clear that reducing the number of attractors during the generation of the trees produces trees with less branches, which could be undesirable to the user. Increasing the number of attractors results in more branches and more realistic looking trees, however it also results in a longer generation time.

5. Conclusions

5.1 Satisfaction of the Aims and Objectives

The overall aim of this project was to develop a program to reduce the time needed to model realistic trees, by procedurally generating them. The aim was broken down into objectives that needed to be met, so that the overall aim could be met.

1. Identify algorithms for procedurally generating trees.

   Given the research completed in Section 2 on tree generation algorithms and the decision made in Section 3.1.1, it is clear that this objective has been met.

2. Identify suitable engines for creating a program to procedurally generate trees.

   In Section 3.1.2 a decision was made to use the Unity engine to procedurally generate the trees, therefore this objective has been met.

3. Develop a program that will reduce the time needed for modelling trees by procedurally generating them, using the algorithm selected in objective 2.

   Section 3.2 shows the implementation of the space colonization algorithm, that was selected in objective 2, into Unity scripts which are used to procedurally generate trees; therefore, reducing the time needed to model trees as discussed in Section 4.2. For this reason, this objective has been met.

4. Summarise and evaluate the performance of the program.

   A performance analysis was undertaken on the program, in Section 4.2. This looked at the generation time of trees, as the number of attractors increased. Therefore, this objective was fully met and, subsequently, the overall aim of the dissertation was met.

5.2 Recommendations for Procedural Tree Generation in the Games Industry

If, in the games industry, online generation is used for the procedural generation of terrain and trees etc., a smaller number of attractors would need to be used (or a different tree generation method like L-Systems). This is because, in online generation, the procedural generation is performed in real-time. This is much more taxing on the hardware and requires a lot of computational power. If a large number of attractors were used it would increase the generation time and computational power needed for generating the world.

If offline generation was used, this method of tree generation would be more suitable. This is because, in offline generation, the map is procedurally generated before running the game. An example of this is 'No Man's Sky', which uses offline generation, and all players explore the same map. The generation time would be increased by using this method of tree generation; however, it would not impact the performance as the map would already be generated before the user enters the game.

This method of tree generation is much faster than modelling trees by hand, so would work best when assisting level designers to create the world. Instead of modelling trees by hand they can use the program developed in this project to generate trees and place them in the world. This would also provide much more variation for trees in the world, rather than reusing models, making it more immersive and realistic.

5.3 Reflections

During this project, a program was developed which uses the space colonization algorithm to procedurally generate realistic trees. The properties of the program can be adjusted to produce different results for different types of trees to be generated. Using this program would definitely reduce the time needed to model a realistic tree manually and therefore has successfully met the aim that was set for the project. When analysing the performance of the program the tests were fair and comparable, as all parameters except from the one being tested were kept the same. The algorithm chosen to procedurally generate the trees ensured that the trees being produced were realistic, as it is shown that nature works in a similar way.

The program could have been improved by adding the ability to export the generated meshes so that they could be used in different projects. Currently the tree cannot be saved and only exists whilst the program is running. This is a problem because the user cannot user the model that was generated and should be added in a future project.

The program could also have been improved by adding the ability to add textures to the tree. Currently the trees generated can only be a solid colour. This makes the tree generated less realistic, which is a problem as this project aims to procedurally generate realistic trees. Additional steps could be taken to allow for this feature to be added to the project in the future.

Another improvement for this project would be that it cannot generate leaves on the trees. This severely limits the types of trees that can be generated with this program. In a future project it would be good to research this feature so that it can be added to the program meaning it could be used in a wider range of projects.

A final improvement for this project would be in regards to the shape of the trees that are generated using the program. Currently all trees generated with this program are a similar shape to each other and this is because of the spherical volume which the attractors are positioned in. In a future project, more options of shapes could be added to allow for different shaped trees to be generated as this would allow for the program to be used in many different scenarios.

6 References

[1] Runions, A., Lane, B., and Prusinkiewicz, P. 2007. Modeling Trees with a Space Colonization Algorithm. (pp. 63-70).

[2] Webb, J. (2021). Modeling organic branching structures with the space colonization algorithm and JavaScript. [online] Medium. Available at: https://medium.com/@jason.webb/space-colonization-algorithm-in-javascript-6f683b743dc5 [Accessed 29 Apr. 2022].

[3] Santell, J. (2019). L-systems. [online] jsantell.com. Available at: https://jsantell.com/l-systems/ [Accessed 4 May 2022].

[4] Twitter: @nice_byte. (2018). Generating Trees and Other Interesting Shapes with L-Systems. gpfault.net. Available at: https://gpfault.net/posts/generating-trees.txt.html [Accessed 5 May 2022].

[5] Nuić, H., and Mihajlović, Ž. 2019. Algorithms for procedural generation and display of trees. In 2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO) (pp. 230-235).

[6] Knutzen, J. (2009). Generating Climbing Plants Using L-Systems.

[7] Prusinkiewicz, P. 1999. A look at the visual modeling of plants using L-systems. Agronomie, EDP. Sciences, 19 (3-4), pp.211-224. ffhal-00885925f

[8] Crespy, B. (2019). Generating a 3D growing tree using a space colonization algorithm – ciphrd. [online] ciphrd. Available at: https://ciphrd.com/2019/09/11/generating-a-3d-growing-tree-using-a-space-colonization-algorithm/ [Accessed 09 May 2022].

[9] Juego Studio. (2020). 9 Major Advantages of Unity 3D Game Engine: An Ultimate Game Developing Tool. [online] Available at: https://www.juegostudio.com/blog/advantage-of-unity-3d-game-development/. [Accessed 09 May 2022]

[10] Pluralsight.com. (2019). Everything you need to know about C# | Pluralsight. [online] Available at: https://www.pluralsight.com/blog/software-development/everything-you-need-to-know-about-c-. [Accessed 14 05 2022].

[11] Pedamkar, P. (2019). What is Visual Studio Code? | Features And Advantages | Scope & Career. [online] EDUCBA. Available at: https://www.educba.com/what-is-visual-studio-code/ [Accessed 15 May 2022].

7 Appendix

Code can be found at the following GitHub link:
https://github.com/LaurenJeffersonPugh/Stage-4-Dissertation
Note that the code is not mine and is largely identical to ‘Generating a 3D growing tree using a space colonization algorithm’[8] as this algorithm was chosen as a basis for the program.