Procedural Generation of Trees Using a Space Colonization Algorithm

Paulius Bieksa
40216180@live.napier.ac.uk
Edinburgh Napier University - Advanced Games Engineering (SET10110)

Abstract

This project attempts to develop a proof of concept application capable of generating realistic looking three-dimensional trees procedurally, using the space colonization algorithm. The algorithm has a small number of control parameters that can change the resulting tree structure. The project is successful and with future work could potentially be developed into a useful tool for content generation in projects that require large amounts of 3D models.

Keywords – Procedural, PCG, Space colonization, tree

1 Introduction

Project aims This aim of this project is to develop an application capable of procedurally generating realistic looking trees. The goal of this is to delve into a field not explored by other modules. For this project the field in question is procedural generation (PCG). Large quantities of 3d models of foliage are often required when making certain types of games (e.g. open world), and trees are one of the most common type of models required. This project focuses on generating trees by using a few control parameters to change the output.

Space Colonization Algorithm The core of the application developed, is the space colonization algorithm. The algorithm treats competition for space as the main factor determining branching and the structure of the tree. To start the algorithm an initial input of an N number of attraction points and a root for the tree structure is required. The algorithm then operates in an iterative process. In each iteration the algorithm starts by going through all the attraction points and finding the closest node on the tree. If the distance between the attraction point and the node is smaller then the radius of influence r_i, a normalized attraction vector pointing towards the attraction point is added to a list of attraction vectors $S(v)$ for that node. The next step is to average all of these lists of vectors for each node to get a single normalized vector v_{att} determining the direction of growth for the next branch at that node. A new node is then placed at a predefined distance d from the node along the v_{att} vector. After all the new nodes have been added, all of the attraction points that are closer to any node than the kill distance d_k are removed. The algorithm iterates until there are no more attraction points left or no attraction point is within the radius of influence e_i of any node in the tree. A visual representation of a single iteration of the algorithm can be seen in **figure 1**.

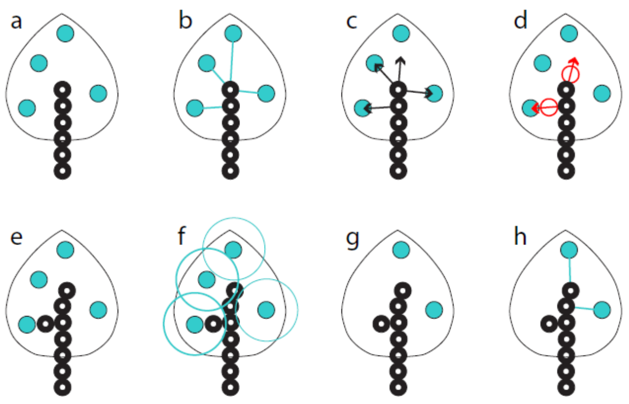

Figure 1: **Space colonization algorithm** - an example of space colonization algorithm used for leaf vein generation.

2 Implementation

Dependencies The focus of this project was the implementation of the space colonization algorithm, therefore developing an engine to display the results would be wasted time. For this reason a graphics engine from module SET08116 was used. There were several reasons for picking this particular framework. Firstly the framework is cross-platform. The framework is also familiar due to being used in said module. The final reason is that set up to be used with source control, therefore the source files for the application only contain the code for the application itself and the framework is downloaded as a git submodule when building the application.

Algorithm implementation In this application, the attraction points are generated using a uniform distribution of randomly generated points within a specified volume. The volume is defined by a 2D curve that is rotated around the y axis of 3D space to create an envelope. Any point outside of the area defined by this curve has its position re-generated and checked again. Once a set number of points is generated inside the envelope, the algorithm can be executed. To represent the tree structure an n-tree is used. This data structure is implemented using a C++ struct.

Control parameters The program uses 4 control parameters to influence resulting tree. Those control parameters are:
r_i - radius of influence (*ri* in the application),
d - node placement distance (*dp,* in the application),
d_k - attraction point kill distance (*dk* in the application),
Envelope defining the crown.
The first three of these are only changeable in the code of the application. They are defined as constants and are easily accessible for a programmer with means of compiling a C++ application, however at this point unavailable for people with access only to the compiled application. The envelope, however, can be manipulated from within the application (**Figure 2**), to let the user define the shape of the crown of the tree the way they want.

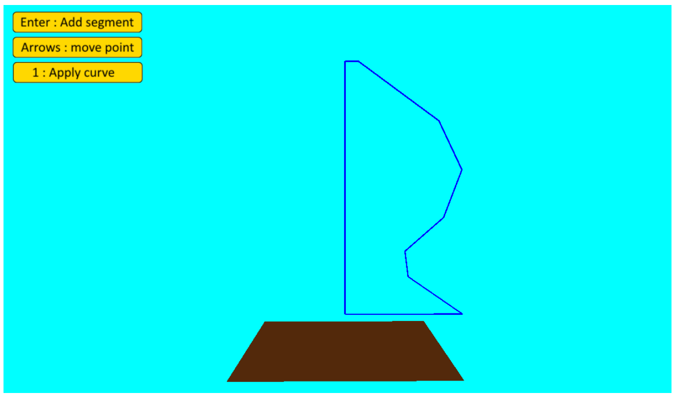

Figure 2: **Defining the Envelope** - example of defining the tree crown envelope within the application.

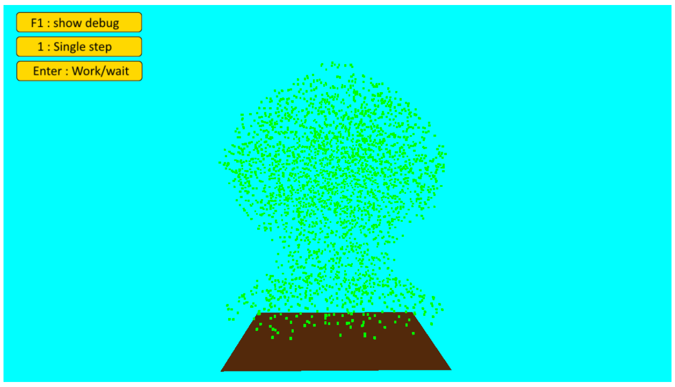

Figure 3: **Attraction points** - attraction points generated using the envelope in figure 2.

Representing the tree While the space colonization algorithm can output a structure of the tree, the tree itself needs to be represented in a 3D environment. To do this, the girth of every branch has to be considered. An algorithm for calculating this was developed based on formula:

$$r^k_{parent} = r^k_{child1} + r^k_{child2}$$

The coefficient *k* used in the program is 2.3. This gives the formula realistic proportions. The algorithm traverses the tree form tips to root and calculates the radius for each branch. The tree is then represented by a series of cylinders that have a radius calculated using this algorithm. To reduce any unnecessary complexity in the tree structure and reduce the load on system resources, non-branching segments of the tree that are representing a straight line are combined to a single node.

3 Future Work

The project can provide a basis for future work in developing a tool that can be used to generate content for applications that require large amounts of foliage. To bring the application to that point some improvements are required. Firstly the simplistic 3D representation of the tree, while fine for a proof of concept, would need to be improved. Some sort of polygonization would be required to represent the tree as a single mesh. An ability to export the resulting mesh would also be a feature of high priority for most use-cases. The Structure of the tree could be slightly improved by some sort of tree skeleton post-processing. Lastly adding leaves, flowers, fruit etc. would increase the fidelity of the tree by a large margin and would be required in most uses of the resulting software.

4 Conclusion

The project was successful in fulfilling it's aims. The trees that the application generates are definitely tree-like. The control parameters allow for a wide range of results. This project has provided a great opportunity for learning about PCG techniques.

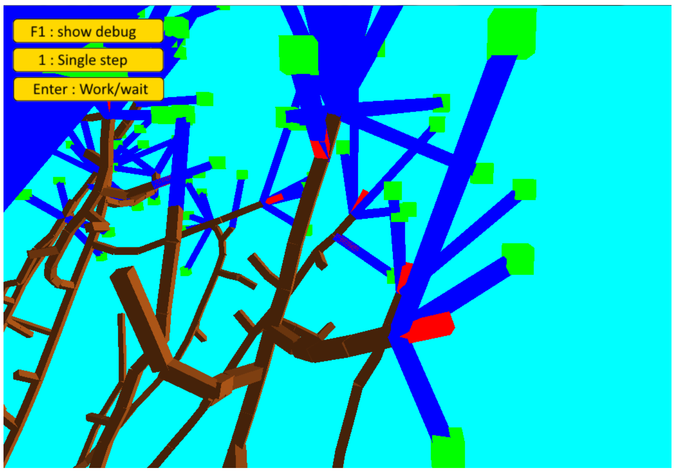

Figure 4: **Space colonization algorithm in the application.** Brown represents the tree structure, blue - attraction vectors before normalization, green - attraction points, red - position for the next branch placement.

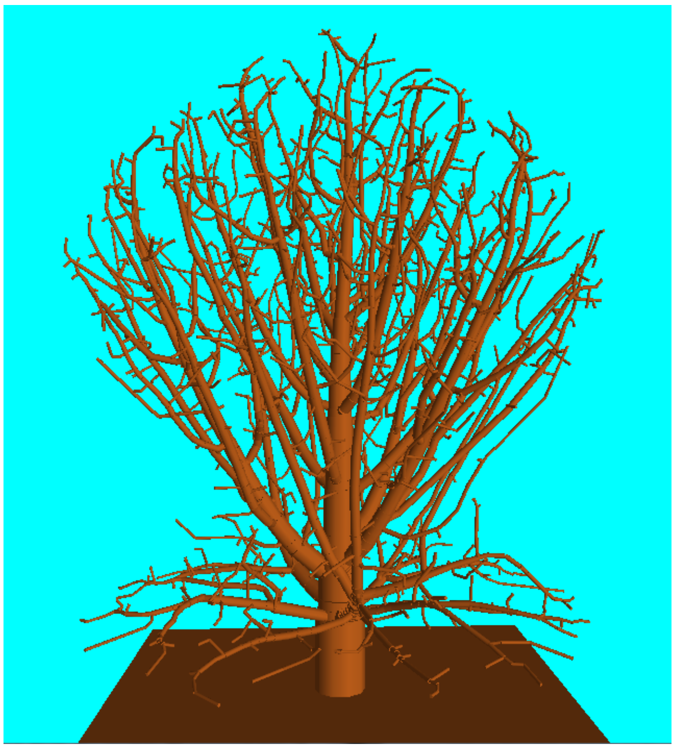

Figure 5: **Tree generated using attraction points in Figure 3**

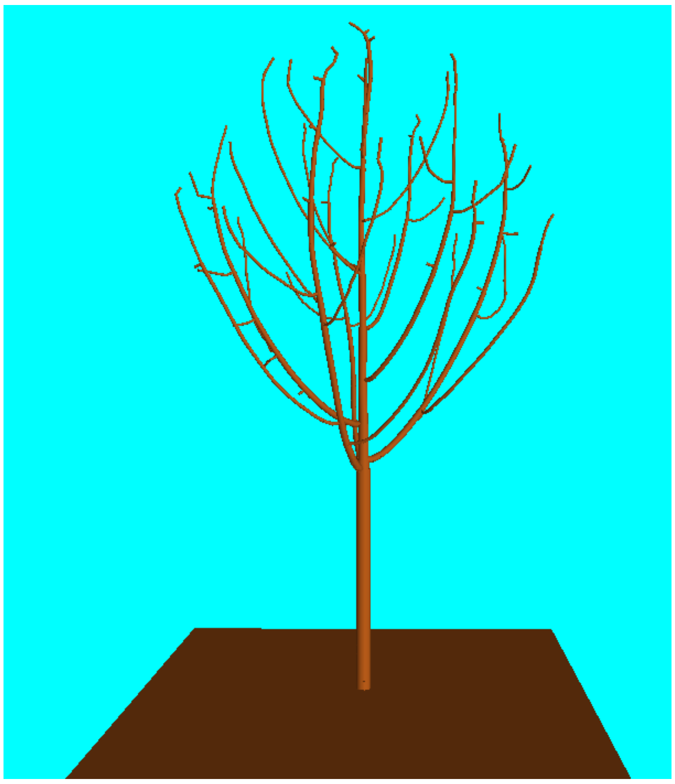

Figure 7: **Effect of increasing kill distance**

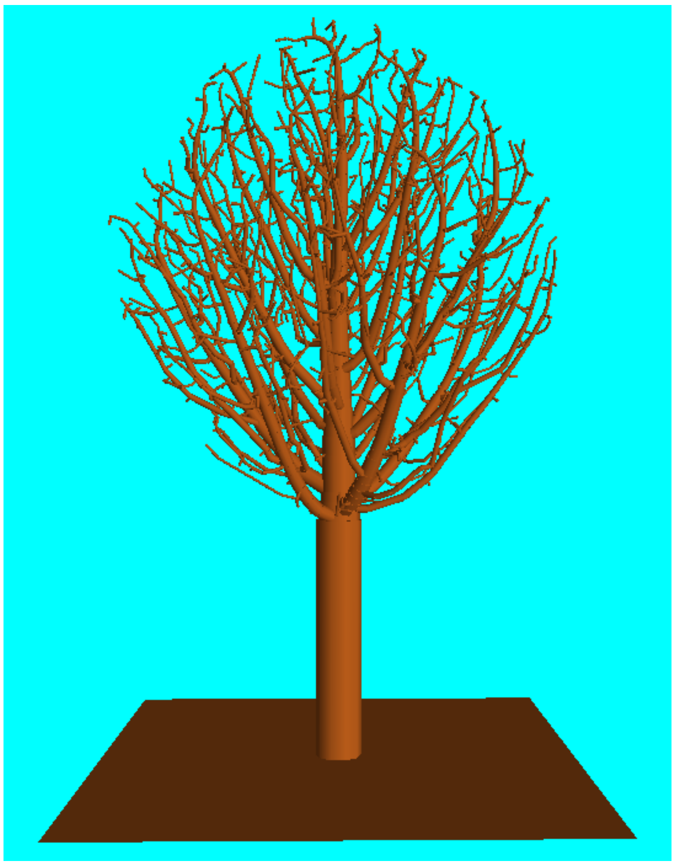

Figure 6: **Tree generated using default parameters**

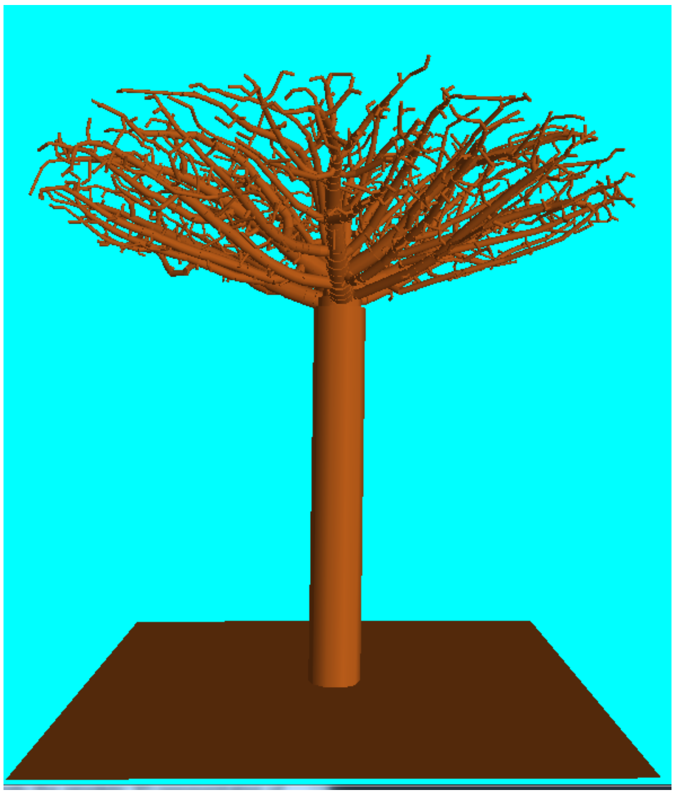

Figure 8: **Tree generated using a small envelope with reduced radius of influence and kill distance**