[[1]](#footnote-1)

Environmentally Affected L-System using Voxels

A. Leeuwenberg

University of Wales, Trinity Saint David

*Abstract*— Methods for creating computer generated trees have been developed since the 1960's. The most known technique is L-systems. Most methods do not simulate environmental effects on plant growth. This paper proposes a method for developing vegetation and having the growth affected by the environment. This is achieved by growing L-systems in three dimensional voxel space. The factors affecting growth are direct and ambient light and space availability. Direct and ambient light were calculated using a method of voxel ray casting. Plants would also detect and avoid objects and other branches. This was achieved by voxelising collision geometry at the start of the process and updating the voxel space as the plant grew.

A L-system generates a string which describes a plant. A novel method for interpreting the string was developed. This allowed for realistic growth to occur. With this method occlusion and collision of a trees’ own branches as well as its neighbours was possible.

**The proposed solution was able to generate the desired affects. Future work could focus on implementing a more advanced L-system that can generate realistic trees. Improving the methodology would increase computational times. This would allow for real-time feedback and adjustments so that the method could be used for production purposes.**

*Keywords*—L-System, fractal, voxel, trees, environment, competition

# Introduction

Trees form visually interesting and complex structures. Their shape is determined by their genetics and environment. It could be assumed that an algorithm for generating these structures would have to be equally complex and advanced. Tackling the problem of generating vegetation has received attention since the 1960's and today there exists a significant body of methods that can generate the complex shapes of trees. The problem with many of these methods is that the tree growth is not affected by their environment.

Computer generated forest are required for film and television. Manually creating individual trees would be time consuming. This paper aims to present an approach to procedurally generate tree structures and have environmental factors affect their growth. The approach is to generate L-systems structures and have their shape affected by their surrounding environment by growing them in voxel space.

The factors that will affect growth are light and space availability. The availability of these factors are affected by the environment, neighbouring plants and the plant itself.

# Background

## Fractals

The fundamental concept for generating plants is fractals, developed by Benoid Mandelbrot [1]. A fractal shape shows increased detail when magnified. If the detail remains similar, irrelevant of the magnification, then the object is said to be self-similar. This can be seen in plants where the structure of a tree’s smaller branches is similar to the larger ones. Plants cannot be said to be true fractal given that nature is finite however the concept is still valid: the reason being that if an algorithm can describe the shape of a stem then it can also generate the smaller branches.

## Tree growth methods

There exists a myriad of techniques that are able to generate vegetation [2]. One of the most well-known methods is L-systems, developed by Astrid Lindenmayer [3]. This technique uses the fractal concept of self-similarityto generate plants. This technique iteratively rewrites a string which afterwards can be used to describe biological structures. The initial string is known as the axiom, *ω*. It is rewritten over a defined number of generations, *g*. Rewriting is controlled by processors, *P,* where a predecessor is changed to a successor during each generation. The L-systemhas an alphabet, *V*, of characters with functions associated with them. The string is entirely rewritten every generation. If *ω* = F, *V* = Ff, *P0* = F-> Ff, *P1* = f ->F and *g* = 3.

*ω* F

*p1* Ff

*p2* FfF

*p3 FfFFf*

Turtle is the commonly used name for the agent that is used in order to generate the tree. When generating the shape from the string the turtle which contains a transformation matrix is manipulated. Different characters affect the turtle’s orientation and position. As the turtle is translated through space it plots out the shape. There are many different L-systems. The bracketed OL-system stores the turtle transformation matrix when an open bracket occurs in the string. The transformation matrix is then re-instated to the turtle when a closed bracket occurs. This enables branch structures to be generated. Prusinkiewics extended the L-system to simulate environmental effects with Open L-systems [4].

Voxel space is at term which describes the division of three dimensional space into equal sized cubes [5]. This is a computationally effective way of representing an environment. Each voxel can store any number of parameters.

Ned Greene used voxels space to affect the growth of plants. The growth was affected by luminosity, light direction and object collision.

Light intensity was calculated for each growth call. Diffuse light was determined by randomly scattering rays towards the sky hemisphere and traversing through the voxel space. Direct light was determined by sending rays towards an arc which represented the path of the sun through the sky. For both diffuse and direct light the fraction of rays that reached the sky without passing through an occupied voxel represented the luminosity of a point. This creates a luminosity value between 0 and 1.

## Hypothesis

H0 is that computer simulated plant development which is affected by their environment cannot be achieved by generating L-systems in a 3D voxel environment.

H1 is that computer simulated plant development which is affected by their environment can be achieved by generating L-systems in a 3D voxel environment.

# Methodology

The first stage is to voxilise the environment. To generate the tree or trees first the string rewriting process creates the L-system string, L-string. This is then used to plot the tree. Each time a branch grows the luminosity and space availability is calculated and the stem growth is adjusted. The individual processes will be discussed in detail in the following sections.

## Generating L-system string

The L-string needs to be generating before the plant can be grown. The processors are exported as a single string and separated once imported. The axiom and generation are also imported. The L-string is generated checking each character of the axiom and finding the corresponding predecessor. The successor associated with the processor is appended to a temporary string. Once all the characters in the axiom have been analysed the final string is complete. The process is repeated for each generation but instead in using the output string is used instead of the axiom. Once complete the final L-string can be used to generate the tree.

## L-system string interpretation

The L-string represents the plant structure. L-strings are interpreted linearly from left to right but this does not generate the structure in a manner consistent with natural development. Plants growth occurs from the region near the tip of a branch called the meristem. All active meristems in a plant develop simultaneously. When a branching event occurs the L-string develops the branch to its terminal point. Next it steps back, while constantly developing all subsequent branching points to their terminal status before returning to the initial branch point. This would create a bias when calculating self-collision and self-occlusion.

A method of interpreting a L-string was developed to be more consistent with natural plant growth. The main feature of the method is that multiple branches are able to grow simultaneously. This was achieved by analysing and restructuring the L-string into individual branch L-strings which describe a single branch. The L-strings are organised into a structure which contained vital data; L-string which describes the specific stem, branch identification number, when it should start growing and its parent branch. The child branch inherits its starting data from the parent branch. This information included its transformation matrix and growth rate.

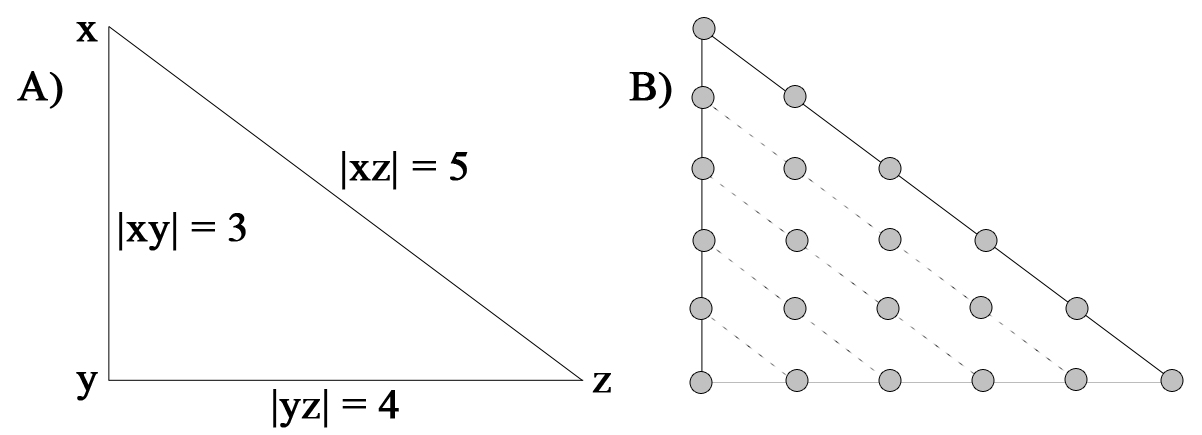
A second structure is generated to control which branches were currently active. This structure was generated by scanning the initial L-string. For each increment in the structure a character for all active branches is executed. A branch remains active until all its characters have been executed. The initial structure generated stored all the branch identification integers in numerical order. To prevent branch bias the order of the integers were randomised.

## Converting geometry to voxels

Calculating collision and light occlusion using a mesh is computationally complex and expensive. Representing the mesh in a less complex form decreases the computation times. Voxel space was used to increase computational speed. A method for determining which voxels a mesh occupies was devised. For the results to be correct it was assumed that the mesh was constructed purely from triangles. A number of scatter points were distributed on the surface. These determine which voxel scatter are occupied. The state of any occupied voxel was changed from empty to full. The status of each voxel was stored in a list. This is known as spacial enumeration [6].

The distribution of scatter points was determined by the size of a voxel. he length of the hypotenuse was calculated for each face. If the triangle was equilateral then an arbitrary edge was chosen. The subdivision was calculated using the formulae:

where |*m*| is the hypotenuse, *v* is the voxel width and *d* is the quotient. In Figure 1A |m| = 5. If *v* = 1.0 then *d* = 6. Next an arbitrary vertex, *a,* of the face was chosen. The vector, *v1* and *v2*, and magnitude, |*m1*| and |*m2*|, to the other two vertices from *a* were calculated. Starting at the other two vertices a vector, *q0*, is generated between the two edges associated with *a*. Evenly distributed points were generated along the vector, *q*. The number of points generated was *d - i* where *i* is the current loop iteration. The voxel associated with the position was changed from and empty to occupied status. A new vector, *q1*, was created between *v1* and *v2*. The fraction along *v1* and *v2* is determined by (1/ d) \* (*d - i*).This process repeats until *d - i* = 0. An example the scatter points generated is shown in Figure 1B.



*Figure 1 shows the sample positions on a face. a) The magnitude of each edge is calculated. b) Sample points generated based on the maximum magnitude.*

## Growing trees

Once the L-string is re-interpreted and the environment has been voxilised the trees can be generated. For a single tree the structure which stores the branch ids is read. The corresponding id is called from the structure contain the branch data. The current character of the L-string is called and the corresponding function is called. If there are further characters in the string the next character is prepared for when the branch is called again. The next branch id is then called from the structure. A single iteration is complete once all the branch ids have been called. The next iteration in the structure is then interpreted and the process is repeated when all the branches have been developed.

Due to the new method of interpreting the L-string only two types of commands exists: local-rotation and growth. Euler rotation alters the transformation matrix of a branch. Growth generates a point from the branches current position along the y-axis vector to an end point determined by a magnitude stored in the branch structure. The voxels which the branch growth traverses are changed to an occupied status.

In order to grow multiple trees a structure in the program is created for each tree. The structure contains pointers to the individual trees growth structures. One string command is executed for each of the trees. Once the end of this list has been reached the process is repeated until all the string commands have been executed.

## Collision detection and avoidance

Collision detection and avoidance ensures that a plant either grows around or is prevented from growing though an object or other branch. Before a branch is generated a function in the program checks whether it will grow through an occupied voxel. This is achieved by creating evenly distributed points along the branch vector. The number of points and the distance between them is based on the magnitude and voxel size. If the collision check is true then the branch will start generating vectors within a certain radius from its proposed growth point. If any of these branches do not grow through an occupied voxel then the stem is generated. If no permitted branch is generated after N attempts then the branch, and all its child branches, are aborted.

## Light intensity

Plants require light in order to photosynthesise and grow. The illumination intensity at each active meristem affects the direction of growth as the plant develops. In this model two techniques are implemented. The value from both is weighted and combined in order to calculate illumination at any given point. The first technique calculates direct light. This is light which travels unobstructed from its source to a position. The second technique determines ambient light. This is caused by light rays being scattered by the environment. Greenes' method was used to calculate the two values [5]. This is done by generating points around the growth position. The distribution of the points differs for direct and ambient light. A vector from the position to the point is generated. The points are 1 unit distance from the growth position so that the vectors are normalised.

### Direct light

In this method the source of direct light is the sun. The sun travels at a constant velocity in an arch through the sky. Equidistant points were created along an 180° arc. Points were incrementally generated along the vector of the ray. The step size is determined by the voxel size. A ray is aborted if it traverses a voxel which is not empty. The light intensity is calculated by dividing the number of aborted rays by the total number of rays.

### Ambient light

Ambient light represents light which is scattered in the environment. Unlike direct light the points need to be randomly generated on a semi-sphere. The first method attempted was to generate scatter points using N-rooks. This method divides a two-dimensional area into rows and columns. Points are scattered so that each row and column only contains one point [7]. Once these points had been generated they were normalized. The problem with the results generated was that the scatter points where not distributed over half a spherical surface. Instead they were distributed on over a square section of a sphere. The points were also biased towards the edges.

Scatter points were instead generated using Marsaglias’ spherical scattering method [8]. This method works by randomly generating values for *V1*and *V2*between -1 and 1. Next *S* is calculated where *S = V­12 + V22* < 1. A new value for *V1*and *V2* are generated if of *S* > 1. The x, y and z positions are calculated using the formulae:

The results create scatter points on a whole sphere. To create scatter points on half a sphere the range of *V1* was changed to be between 0 and 1. The diffuse light intensity is then calculated in the same way as direct light.

### Light requirement

Once the two values are calculated they are combined based on a weighting. The weighting between 0 and 1 is used to simulate pioneer plants and shade tolerant plant species. Pioneer species require direct light to grow whereas shade tolerant species are able to grow in lower luminosity. The weighting for both direct and ambient light equals 1. If the direct light weighting is increased towards 1 the plant will require more direct light. When grown in shade the size of the plant will be reduced.

### Direction of light

A tree grows from low to high luminosity. This behaviour is determined at all of the plants meristems. The methodology used to replicate this phenomenon is based on Greens technique [5]. The direction of greatest light intensity is determined by stepping through the longitudinal and latitudinal voxels from a growth point. The number of voxels in each direction can be controlled. If an occupied voxel is encountered the check is terminated for that direction. The average direct and ambient illumination is calculated for each direction. If this value is below a user defined threshold the plants growth is rotated towards the direction of greatest illumination. The magnitude of rotation is calculated using the formulae:

where *θmax* is the maximum angle of rotation, *imax* is the greatest illumination and *imin* is the lowest illumination. This is calculated for both latitude and longitude. The branch is then rotated *θmax* towards the greatest latitudinal and longitudinal luminosity.

# Results and Discussion

## Luminosity

The effect of voxel size and the number of rays on luminosity can be seen in figure 2. The mean (n=20) and standard deviation was calculated. The results were compared using two-tailed T-test. When 1,000, or 10,000 rays are used the results from decreasing voxel size are statistically different (P<0.05). When the number of rays was 200 or less the results were not statistically different (P>0.05) between voxel size of 0.1 and 0.25 as well as between 0.01 and 0.1. One anomaly was that there was a significant difference comparing the voxel size of 0.1 and 0.25 when 100 rays were sampled. Irrelevant of the number of rays sampled there was a statistical difference between the luminosity values when comparing voxel size 0.01 and 0.25.

*Figure 2 shows the relationship that the number of rays and voxel size have on luminosity. The results show the mean value (n=20) and the error bars show the standard deviation.*

Comparing the results between voxel size 0.01 and 0.25 shows that decreasing voxel size decreases the average luminosity results. The reason for this is that collision geometry is more accurately represented when the voxel size is reduced. A ray is therefore less likely to travel through an occupied voxel.

Increasing the number of rays sampled decreases the standard deviation. In order to achieve consistent results 1,000 or more rays should be sampled. Due to the computational time and because the luminosity needs to be calculated for each growth call only 50 rays are sampled for the ambient luminosity at the growth point and 30 samples for each voxel when determining the direction of greatest luminosity. This creates a large error associated with the result which sometimes affects the branches rotating towards the light.

## Stem structure

Even though the stems generated are modified by the environment the underlying structure remains the same. This is due to the L-system that is implemented being unable to create stochastic variance to the structure. Currently only a bracketed OL-system is implemented. There are more advanced L-systems which can generate realistic tree stems. Since the string rewriting process is separate from generating the structure all these methods would still function with the techniques implemented in this paper.

Another limitation is that stems have no width. A branch will occupy only the voxels which its center passes through. L-systems can have characters which control the stem width. Adding stem width would allow for the main stems to occupy a greater volume of space compared to smaller branches. This would allow for main stems to occlude more light than the small branches.

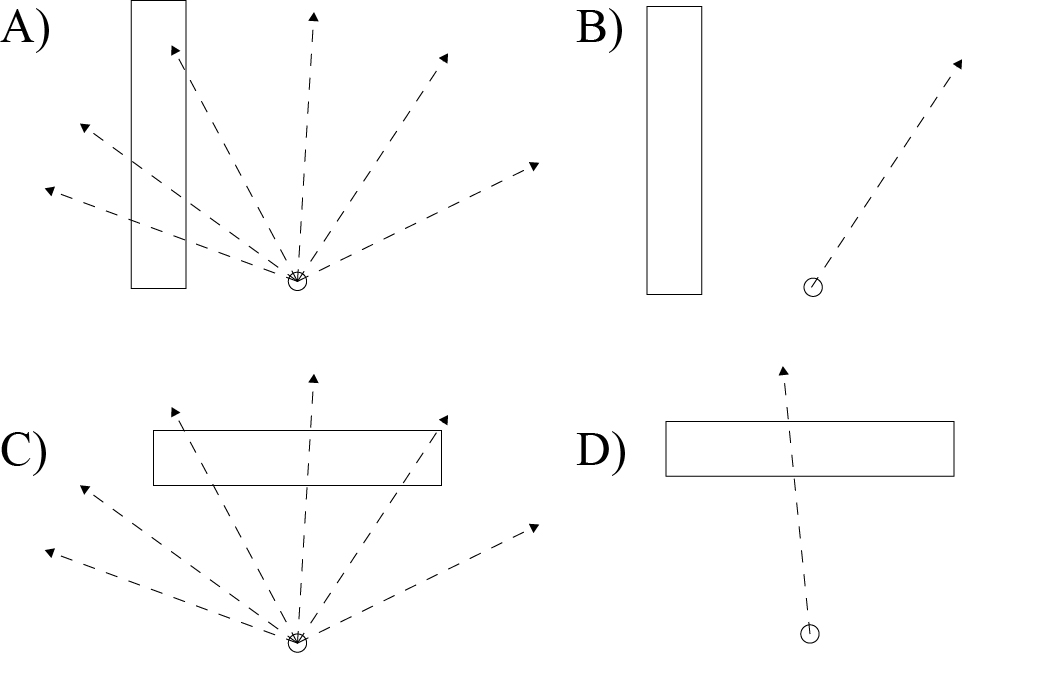
## Competition

Plant competition is passive [9]. Individuals affect their neighbours’ growth by reducing the availability of resources. In the implemented method the resources that are reduced are space and light. As the plant grows the voxel environment is updated which impacts other plants. In nature plants also compete for underground nutrients. A method for simulating root growth could be implemented and would allow for more accurate competition results.

## Rotations

Using Euler rotations limited the quality of the results. This was because Euler rotations do not easily allow interpolation between two orientations [10]. This caused problems when adjusting the branches growth from low to high luminosity. In the current implementation the branches transformation matrix is interpreted to determine the axis vector positions relative to each other. The appropriate rotation is then applied. This requires numerous checks to be made and the resultant rotation is not entirely correct. Using quaternions would eliminate this problem since this methods of rotation allows for interpolation between two orientations [10].

Quaternions would also make it easy to interpolate to any orientation, not just the x and z axis as is currently implemented. This would allow for a branch to grow towards the greatest luminosity. The direction of greatest luminosity could be calculated by averaging the ray cast vectors that do not hit an occupied voxel. Figure 3A shows the rays and figure 5B shows the average direction.



*Figure 3 shows the potential problem of averaging luminosity values.*

A problem could occur if the point is occluded by an object directly above it (figure 3C). The average vector would then be directly towards the occluding object (figure 3D). This problem also exists with the current implementation for calculating the direction of greatest luminosity.

## Real-time

Although the methodology does produce results that are close to the desired effect there are several problems with the implementation. For this method to have any production value it needs to allow the user to interact and alter the tree and get close to real-time feedback. With the current implementation this is not feasible. Factors affecting the computational speed include the number of generations for the *L-*system and voxilisation and illumination algorithms. Some suggested solutions on how to improve the speed and results have been outlined in the subsequent sections.

### Separating the processes

The current implementation with Autodesk Maya prevents real-time adjustments. This is due to the current implementation computing and returning the result. The data from the calculations is lost at this point. If any adjustments are made then the all the calculations need to be recalculated.

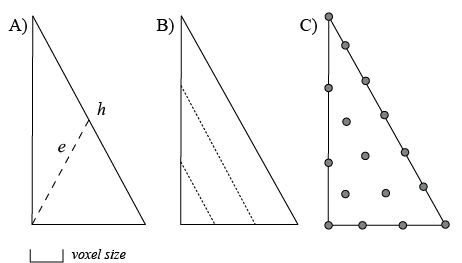
To allow for real-time adjustments in Maya the processes should be separated into self-contained nodes. This would include nodes that generate the L-string, voxelises an object or grows an individual tree. The nodes would have a dependency hierarchy. This allows for a tree to be translated without having to recalculate the L-string, voxel space or other trees. If the axiom or processors of the L-system was changed the voxel space would not change but the trees would be recalculated.

The method for growing multiple trees is functional but basic. The main limitation is that currently the ­L-string is duplicated to each individual tree. This means that only a single structure of tree is generated multiple times. Any variance in form is purely a product of the environmental effects on the plant. Using a node based structure would solve this problem by creating a separate L-string node for each tree.

The other problem is that all the trees are grown simultaneously. This is not desirable since in nature plants start growth at different times. A growth manager node could control when a plant would start growing. This would allow the growth of trees to be staggered.

### Voxels

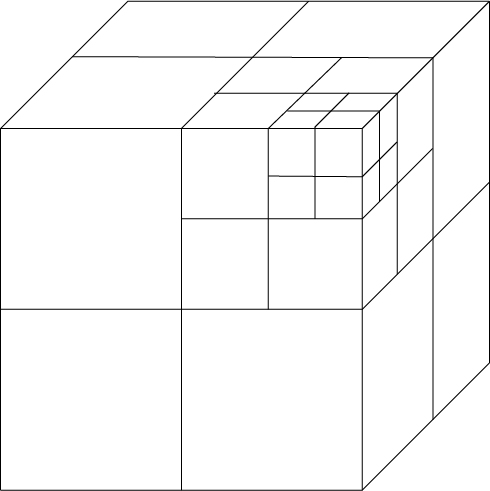
The method of voxelising the geometry is not fully optimised. The algorithm only estimates the number of scatter points based on the hypotenuse of a face. This estimation is appropriate if the triangle is equilateral. If one edge is significantly shorter than the other two or the angle opposite the hypotenuse is close to 180° then an excess of sample points are generated.



*Figure 4 shows a proposed method for evenly generating points on triangle.*

To solve this the distance between the opposite vertex and the centre of the hypotenuse, *e,* should be calculated (figure 3A). The length of *e* determines the number of lines between the adjacent and opposite edge (figure 4B). Points are generated on these lines based on the voxel size (figure 4C).

Another problem is that decreasing the voxel size increases processing times. This is due to the size of a voxel determining the number of sub steps along a ray when calculating light intensity. The distance between sub-steps remains constant even in large empty areas. An improved method would adjust the sample density along the ray based on the local density of collision objects. This could be achieved by using octrees [6]. Octrees store spatial enumeration lists in a hierarchy from low to high detail. At the root level the 3D space is represented by 8 sections. Moving up the hierarchy exponentially increases the number of integers that represent the same space. The size of each section is therefore exponentially decreased (figure 5).



*Figure 5 shows the hierarchical structure of octrees.*

To check if a point is in an occupied voxel the root level voxellist is queried. If it is empty then all its child sections are also empty. If it is occupied the next list in the hierarchy is queried. This continues until either a section is empty or the terminal node in the hierarchy is reached.

Using this method would decrease the time for calculating light. This is because in empty areas only the root voxel list needs to be queried therefore increases the step size along a ray. If the reduction in processing times were effective enough the number of ray casting samples could be increased. This would allow for more accurate luminosity values to be calculated.

# Future Work

As previously mentioned the initial focus for future work could be to decease the processing times in order to allow for real-time manipulation of the trees.

Extending the L-system to cater for more advanced processors would enable realistic trees to be generated. This would allow stochastic variation in the tree structure. Features such as growing flowers and leaves at the end of stems would also be possible. Once this has been achieved focus should be to generate tree geometry with textures.

# Conclusion

The hypothesis that computer simulated plant development which is affected by their environment can be achieved by generating L-systems in a 3D voxel environment is accepted. By re-interpreting the L-string the tree can be generated in a manner consistent with natural growth. This means that the plants growth can affected by itself and its neighbours.

There are many improvements that can be implemented to the methodology that will improve the final result quality and computational speed. These improvements improve either cosmetic or computation speeds of the results without impacting the technique.

# Bibliography

|  |  |
| --- | --- |
| [1] | P. E. Oppenheimer, "Real Time Design and Animation of Fractal Plants and Trees.," *Computer Graphics Lab,* vol. 20, no. 4, pp. 18-22, 1986. |
| [2] | A. Leeuwenberg, "Procedural Creation and Management of Large Scale Vegetation Environements," *Unpublished,* 2014. |
| [3] | P. Prusinkiewicz and A. Lindenmayer, TheAlgorithmic Beauty of Plants, New York: Springer-Verlag, 1996. |
| [4] | M. R and P. P, "Visual Models of PlantsInteracting with Their Environment," *SIGGRAPH,* pp. 397-400, 1996. |
| [5] | N. Greene, "Voxel space automata: Modeling with stochastic growth processes in voxel space," *Computer Graphics,* vol. 23, no. 3, pp. 175-184, 1989. |
| [6] | D. Meagher, "Geometric Modeling Using Octree Encoding," *Computer Graphics and Image Processing,* vol. 19, pp. 129 - 147, 1982. |
| [7] | P. Shirley, "Discrepancy as a Quality Measure for Sample Distributions," *Proceedings of Eurographics,* pp. 183-193, 1991. |
| [8] | G. Marsaglia, "Choosing a point from the surface of a sphere," *The Annuals of Mathematical Statistics,* vol. 43, no. 2, pp. 645-646, 1972. |
| [9] | A. Leeuwenberg, "The effect of nutrioent availability, intra and inter-specific competition on the growth of Ceiba pentandra and Chromolaena odorata," *unpublished,* 2009. |
| [10] | F. Dunn and I. Parberry, 3D Math Primer for Graphics and Game Development, Boca Raton: CRC Press, 2011. |

1. [↑](#footnote-ref-1)