

School of Computer Science and Software Engineering  
Monash University



Bachelor of Software Engineering Honours Project

Literature Review – Semester 1, 2006

# **Generating Sinewy Networks using Environment-Sensitive Automata**

Anthony Gaarenstroom 18472869

Supervisor: Alan Dorin

# Contents

<b>LIST OF FIGURES.....</b>	<b>3</b>
<b>1. INTRODUCTION.....</b>	<b>4</b>
<b>2. GROWTH SYSTEMS .....</b>	<b>5</b>
2.1. L-SYSTEMS : STRUCTURE & RULES .....	5
2.2. CONTEXT-SENSITIVE L-SYSTEMS .....	6
<b>3. VOXEL SPACE AUTOMATA.....</b>	<b>8</b>
3.1. ENVIRONMENT-SENSITIVE AUTOMATA.....	8
3.2. VOXEL ENVIRONMENTS .....	9
3.3. TILING METHODS.....	10
3.4. INHIBITORS AND ENHANCERS.....	11
3.4.1. Modelling Light.....	12
3.4.2. Texture Mixing.....	13
<b>4. REFERENCES.....</b>	<b>15</b>

# List of Figures

1-1	An example of a sinewy network. (Image produced by an initial version of the software component of this project.)	4
2-1	A sample L-System, showing the initiator, a single generator, and a sequence of two iterations.	5
2-2	Examples of biological structures created using L-Systems. (Taken from [22], <i>The Algorithmic Beauty of Plants</i> )	6
2-3	The effects of pruning. (Taken from [21], <i>Synthetic Topiary</i> )	7
3-1	A model of a plant growing toward a light source. (Taken from [3], <i>Fast Estimation of Growth Direction in Virtual Plants Simulation</i> )	8
3-2	Choosing the path for a new segment by checking for obstacles using rays. This method was used by Arvo and Kirk [2].	9
3-3	A plant system grown using a VSA. This type of image is similar to what the project will hope to produce. (Taken from [7], <i>Voxel Space Automata: Modeling with Stochastic Growth Processes in Voxel Space</i> )	10
3-4	A simple 2D drawing made up of lines (left), and the voxel grid (right) produced when this drawing is tiled. (The voxel grid is 20x20 unit squares)	11
3-5	Mapping a line into a 2D voxel grid. The voxels are set in alphabetical order. (Taken from [1], <i>A Fast Voxel Traversal Algorithm for Ray Tracing</i> )	11
3-6	The affect of enhancers and inhibitors on a growth system. The intensity of each inhibitor and enhancer is shown in brackets. (Image produced by an initial version of the software component of this project.)	12
3-7	The appearance of a growth system given two sources, each with a texture of a single flat colour.	13
3-8	Two base textures, and the results when mixing them using the average and addition methods. (Taken from [5], <i>An Artist's Real-Time 3D Glossary - Texture Blending Examples</i> )	14

# 1. Introduction

This project aims to develop a procedure that will generate a 3D computer graphics model of a sinewy network, using a range of environmental factors. A sinewy network is a generic structure with an organic appearance, for example vines growing up the side of a house, or the tendons in a human arm. (An example is shown in Figure 1-1)



**Figure 1-1** An example of a sinewy network. (Image produced by an initial version of the software component of this project.)

This project is divided into two sections. The first is the growth system that models the sinewy network. (Section 2) This section will cover a number of different methods for modeling growth systems, and will go on to describe the Lindenmayer-System, or L-System, which will be used to model the growth system for this project. The second section is the simulated environment which influences the development of the growth system. (Section 3) This section will look at different types of Environment-Sensitive Automata, and will then cover the type of environment that will be used to influence the growth system in this project, the Voxel Space Automata. Methods for portraying light in a virtual environment, and their affects on the growth system, will also be discussed in this section.

## 2. Growth Systems

To create models of sinewy networks using computer graphics, there are many different methods. A number of these are covered by Prusinkiewicz [18]. These methods are divided into two categories: Space-Oriented, and Structure-Oriented. Space-Oriented methods typically focus on the interaction of elements over the entire system, while Structure-Oriented methods focus on a mathematically based function for specifying a growth system.

Space-Oriented methods include Reaction-Diffusion (R-D) systems, which allow patterns to be formed by modeling the interaction of two or more morphogens reacting with each other. The reaction is simulated by the use of virtual chemicals, and these morphogens decide what the final shape of the chemical distribution will be. This final distribution of chemicals forms the final pattern. While R-D systems have been used successfully for modeling some biological structures, such as the use R-D patterns used with sea shells [17], their use is limited by the way that they develop. The purpose of an R-D system is to create a pattern, which is excellent for a texture to be used in a biological structure, but not to model an actual, three dimensional sinewy network with its own growth structure. (This is mainly because an R-D system does not grow from a single point, like a tree does, and cannot be controlled very easily.)

Structure-Oriented methods include Lindenmayer-Systems [22], which are the chosen method of modeling sinewy networks for this project. Structure-Oriented methods are excellent for use in modeling sinewy networks because they allow the programmer to replicate the process that a real network uses to grow. For example, a basic plant will always start from the ground, and can only extend its growth from areas where it already exists. This plant development is very similar to an L-System, which replaces existing sections of itself with new sections for every iteration of the method. This allows an L-System to model not only the final structure, but also to model the process of how the network will grow, from start to finish. This is important because of the need to model how the structure interacts with its environment.

### 2.1. L-Systems : Structure & Rules

An L-System consists of a single initiator, which represents the original structure of the L-System, and one or more generators. The final structure is built up by repeatedly applying the generator rules to the existing structure. At every iteration the structure is examined, and if an element is found that fits the requirements for a generator, it will be replaced by the output of that generator. (See Figure 2-1) L-Systems are usually represented as a series of symbols, which can be interpreted by a path-drawing tool such as a turtle.

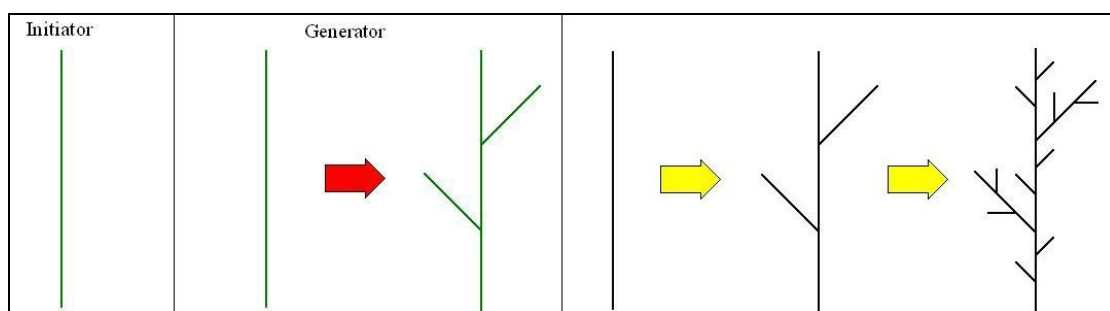
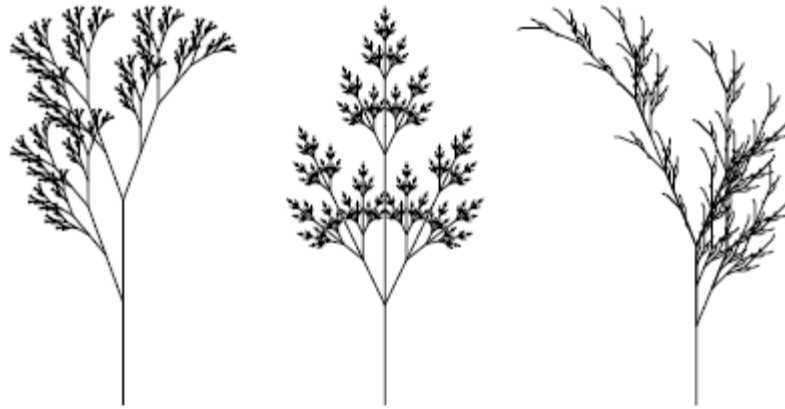


Figure 2-1 A sample L-System, showing the initiator, a single generator, and a sequence of two iterations.

Prusinkiewicz and Lindenmayer [18, 22] first introduced the idea of using L-Systems for modeling plant growth. They show that an L-System, when given the right parameters, will result in a structure that resembles a plant. (Figure 2-2) They also show various methods for modelling different types of plant structures, such as flowers, and leaves. Most importantly, however, is their specification of the rules that result in the creation of an L-System that resembles a sinewy network.



**Figure 2–2** Examples of biological structures created using L-Systems. (Taken from [22], *The Algorithmic Beauty of Plants*)

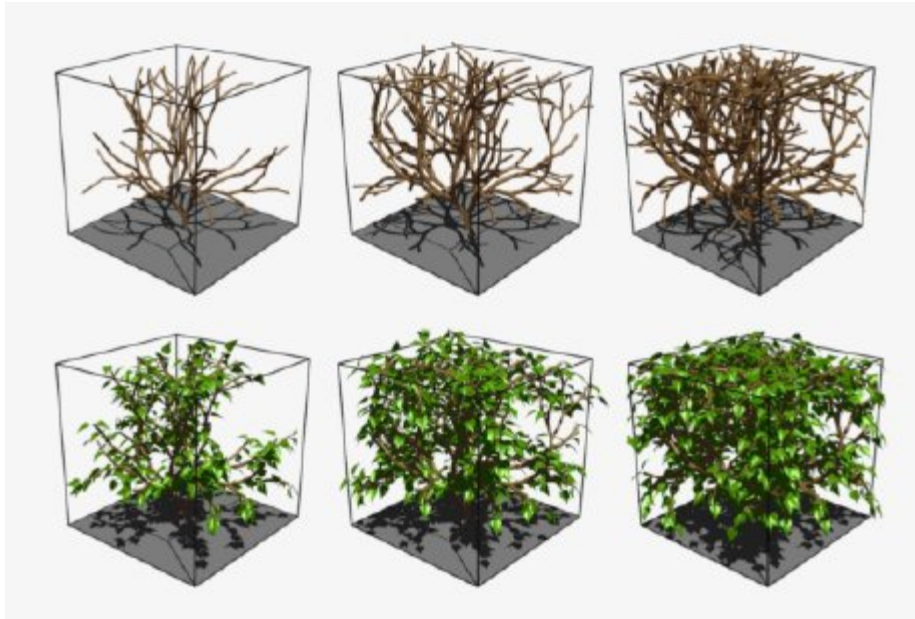
Měch and Prusinkiewicz [16] also introduce a possible application of the L-System, where the L-System can give information to the environment (such as the position of branch segments), and receive information from the environment (such as the intensity of simulated light sources). Their system introduces the concept of an Open L-System, which allows input from an external system, in this case the environment. One example used in their paper shows the growth of a 2D L-System over a surface with different intensity values at each section. The growth is mostly localized around areas with a higher intensity value, whereas areas with a low intensity value are virtually barren.

For this project, a regular L-System that does not allow inputs from an external system cannot be used. Instead a context-sensitive L-System will be used, in order to take advantage of the surrounding environment.

## ***2.2. Context-Sensitive L-Systems***

Context-sensitive L-Systems are different from normal ones in that the rules they contain do not always have a 100% chance of being applied. The decision to apply a certain rule may be different depending on what iteration the method is at, what part is being replaced, or in most cases, what the status of the L-System segments before and after the current segment are.

Prusinkiewicz, James and Měch described the idea of pruning an L-System [21]. Previous L-Systems simply kept growing until they were told to stop, and while this was effective in producing aesthetically pleasing structures, it was difficult to model the effects of obstructions on plant growth. In Section 5 of their paper, a method is described to “prune” the L-System. This means that if a segment grows outside a set boundary, it is automatically deleted. Examples of this method in action are given in Section 6 of their paper. (Figure 2-3)



**Figure 2-3** The effects of pruning. (Taken from [21], *Synthetic Topiary*)

While this method is useful in some circumstances, it does not model the way that a plant would grow under real conditions, but rather how a person would expect a plant to grow, were they pruning it. In real life, a plant does not simply stop growing when it reaches an obstacle, but rather grows around it. This is the approach that will be adopted by this project.

Prusinkiewicz, Hammel, Měch and Hanan described the idea of death in the L-System method [20]. This means that as well as new segments being introduced, old segments may also be removed. While this is similar to pruning, it is different in that the removal of a segment of an L-System may be done for any number of reasons, not just when the ends reach outside a set boundary. Reasons may include age, lack of energy, (when using an simulated environment with the effect of light, usually a model of the sun) and even an attack by an insect. (Described in Section 8 of their paper.)

Another idea described in this paper is that of “information flow”. (Section 8 of their paper.) This term denotes the way that the growth of an L-System behaves as a single unit, instead of a structure made up many small units. For example, information may be passed from the root nodes to the nodes at the end, telling them to grow in a particular direction. (In this way the overall growth of a plant can be directed to a specific point, even though the overall appearance of the plant is still seemingly stochastic.)

Although this method can be useful for creating structures that resemble a particular plant, this project will not use them. This is because the direction of growth of a node in an L-System should be decided by the effects placed on that node by the simulated environment, and not by the effects placed on the node by other nodes in the system, in order to create a more natural looking network.

In an advanced look at L-Systems, the idea of using genetic programming [13] to model the evolution of plants is described by Jacob [10]. He shows that by applying a genetic program to an existing L-System, (representing a plant) the plant can improve itself. An example of this is by maximizing the amount of light absorbed by the plant, which is dependent on the way the plant is structured, by changing the location of symbols in the L-System. While this process will not be used in this project, the work he has completed is a good example of the use of context-sensitive L-Systems.

As well as the resources listed above, I have also consulted a number of other references [8, 12, 19, 23, 24] on the topic of Growth Systems. However these references, while providing excellent background information, were not relevant enough to include in this paper.

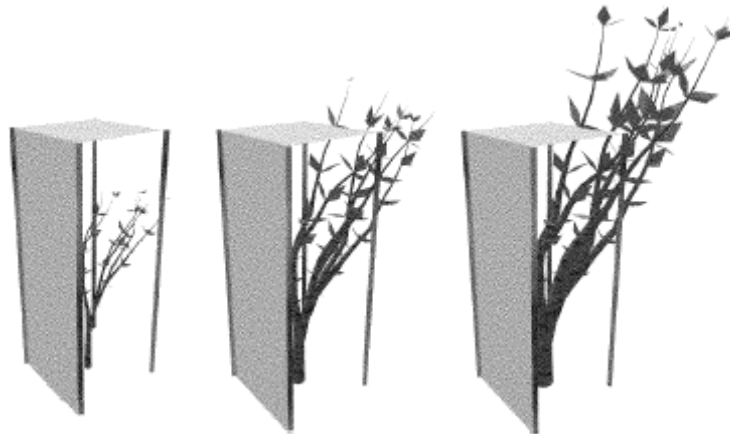
## 3. Voxel Space Automata

The concept of an L-System which interacts with a simulated environment, which may include properties such as light and obstacles, was introduced above, in Section 2.1. Měch and Prusinkiewicz [16] described various ways that the environment can affect the structure of an L-System. This L-System is then interpreted as an Environment-Sensitive Automata. (ESA) An Environment-Sensitive Automata is a method for growing structures that allows input from a simulated environment. An example of a typical ESA is a context-sensitive L-System that tests for obstacles (such as the boundaries of the environment) before allowing itself to grow. (See *Pruning* above, Section 2.2)

### 3.1. Environment-Sensitive Automata

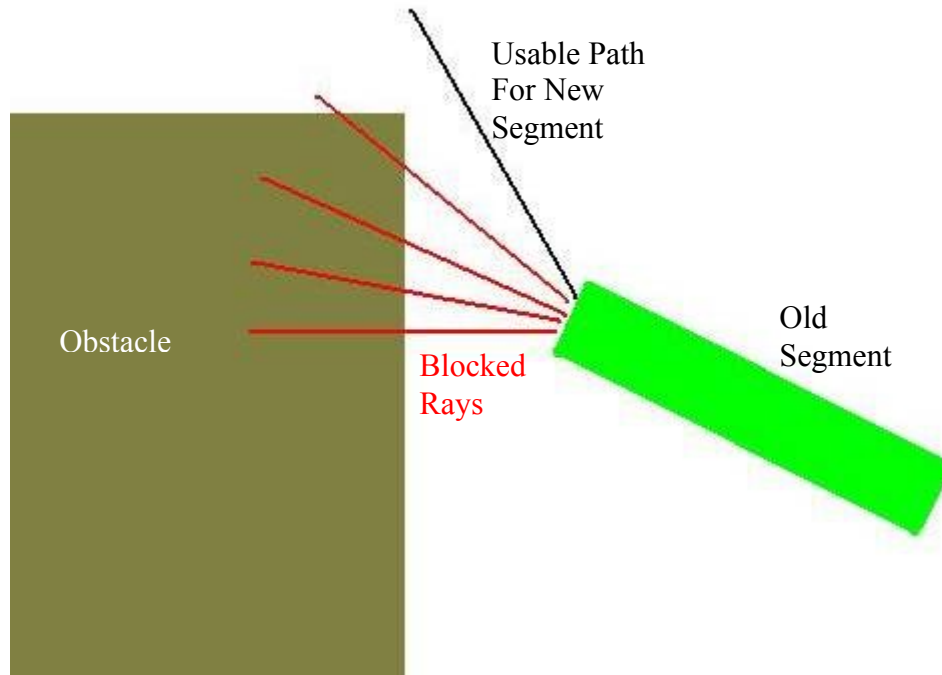
As described by Prusinkiewicz, Hammel, Měch and Hanan [20], there are many different elements that can effect the growth of an ESA. The two most recognized and used elements are light and obstacles, and these are the ones that will be covered by this project. Other elements such as gravity, temperature, humidity, and animal interference can have either a positive or negative effect on the growth of an ESA.

An example of a simple ESA has been completed by Beneš [3]. This ESA calculates the flow of energy from the sun onto a number of leaves, and uses that to determine which way the structure should grow. For example, if all the leaves on the left side of the plant received no light, while the ones on the right received full light, the plant as a whole would grow towards the right. This example of an ESA, while effective, lacks the complexity of a real plant. This is because the direction of growth of a segment should only be dependent only on the effects of light on that segment, rather than on the structure as a whole.



**Figure 3-1** A model of a plant growing toward a light source. (Taken from [3], *Fast Estimation of Growth Direction in Virtual Plants Simulation*)

Two of the earliest people to investigate the use of an ESA were Arvo and Kirk [2]. By using Ray-Tracing, they were able to simulate the way that a segment would detect surrounding obstacles. To do this, a ray was created from every new segment and shot out into the direction of desired growth. If the ray intersected any obstacles, be it a simulated wall or another part of the existing structure, then it would look for another path by sending out a ray in a different direction. (Figure 3-2)



**Figure 3–2** Choosing the path for a new segment by checking for obstacles using rays. This method was used by Arvo and Kirk [2].

Although this method is simple, it is also computationally expensive. There could be many obstacles in the environment, including the growth structure itself, and a new segment would have to test for intersection with all of these. As the structure itself grows, so does the number of tests, which results in the Ray-Tracing ESA being extremely slow. For this reason the project will be using a Voxel Space Automata, which sacrifices space to increase speed and reduce complexity.

As well as the resources listed above, I have also consulted another reference [6] on the topic of Environment-Sensitive Automata. However this reference, while providing excellent background information, was not relevant enough to include in this paper.

### **3.2. *Voxel Environments***

A voxel environment consists of “a three dimensional space partitioned into identical cubes (volume elements or voxels)” [7]. This space is used to compare the location of all the items in the environment. Every item in the environment is read into the voxel grid, and this grid is used when intersection has to be tested. (This process of reading in items is known as tiling, and is covered in more detail in Section 3.3) This saves a lot of time for intersection tests, as only the voxel grid has to be used for testing. However, the disadvantage is that the positions stored in the voxel grid are only approximations of the actual position. This is usually not a problem if the voxel grid has a high resolution, and most applications of Voxel Space Automata (VSA) accept a small amount of error when testing for intersections.

An example of the use of VSA is shown by Greene [7]. In his paper, Greene outlines a method for simulating the growth of a plant structure using Voxel Space to store both the obstacles in the environment, and the plant structure itself. Greene introduced a number of procedures for building plants, such as a fixed number of trials to determine the best path to follow, and reading in the light values into the Voxel Space, as opposed to calculating the light for each new segment as it was being grown. An example of Greene’s work is shown below. (Figure 3-3)



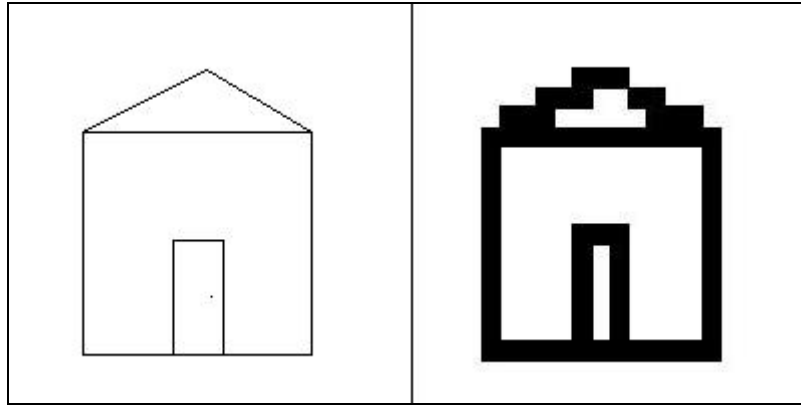
**Figure 3-3** A plant system grown using a VSA. This type of image is similar to what the project will hope to produce. (Taken from [7], *Voxel Space Automata: Modeling with Stochastic Growth Processes in Voxel Space*)

The most important feature of Greene's methods is that of the VSA, and its use for every part of the growth process. This feature means that the overall method is simpler and faster than those using Ray-Tracing, as it does not have to work with different algorithms to accomplish different tasks. (For example, testing for collision with a wall is exactly the same process as testing for collision with another part of the plant, and in most cases the new plant segment doesn't even need to know what it has collided with, only that it cannot grow in that direction.) Because of the simplicity of Greene's results, this project will adopt a slightly modified version of the methods that Greene used. These modifications are mostly to do with the way that light is simulated, and will be explained in Section 4.1 of this paper.

As well as the resources listed above, I have also consulted a number of other references [4, 11, 27] on the topic of Voxel Environments. However these references, while providing excellent background information, were not relevant enough to include in this paper.

### **3.3. *Tiling Methods***

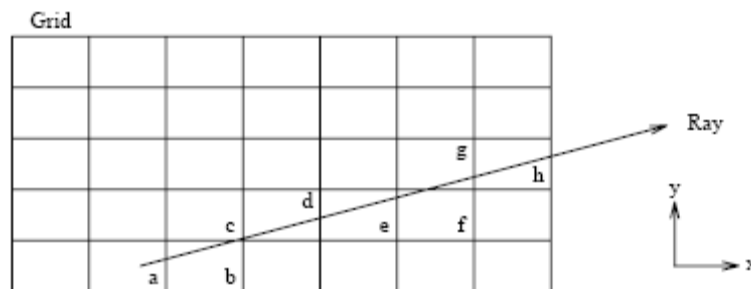
As mentioned above, tiling is the process of reading objects into Voxel Space. In most cases the objects are split into triangles, and then each triangle is separately tiled into the voxel grid. In the case of more complex shapes, such as polygons of more than 3 sides, or any 3D shape, the object is split up into triangles, which are then tiled into the voxel grid. An example of this process, in 2D, is shown below. (Figure 3-4) As this figure shows, the voxel space shows only an approximation of the original object.



**Figure 3-4** A simple 2D drawing made up of lines (left), and the voxel grid (right) produced when this drawing is tiled. (The voxel grid is 20x20 unit squares)

An overview of tiling methods for use with VSA is given by Huang, Yagel, Filippov and Kurzion [9]. In their paper, they describe various methods for tiling lines, planes, and polygonal meshes into a voxel grid. These methods are mostly maths based, involving the use of vectors and trigonometry to gain the exact positions of all the objects. These methods work by focusing on a single voxel, and testing to see what triangles intersect with that voxel

Amanatides [1], on the other hand, proposed a simple and fast method for ray-voxel intersection. For every line, that needs to be read in the first voxel is set, and then the voxels after this one are set, one by one, in the direction of the line. (Figure 3-5) While this method is undoubtedly faster and simpler, it has less uses, as it only applies to lines. (The procedure for tiling a simple triangle requires at least three of these runs, plus an algorithm to fill in the middle of the triangle.)



**Figure 3-5** Mapping a line into a 2D voxel grid. The voxels are set in alphabetical order. (Taken from [1], *A Fast Voxel Traversal Algorithm for Ray Tracing*)

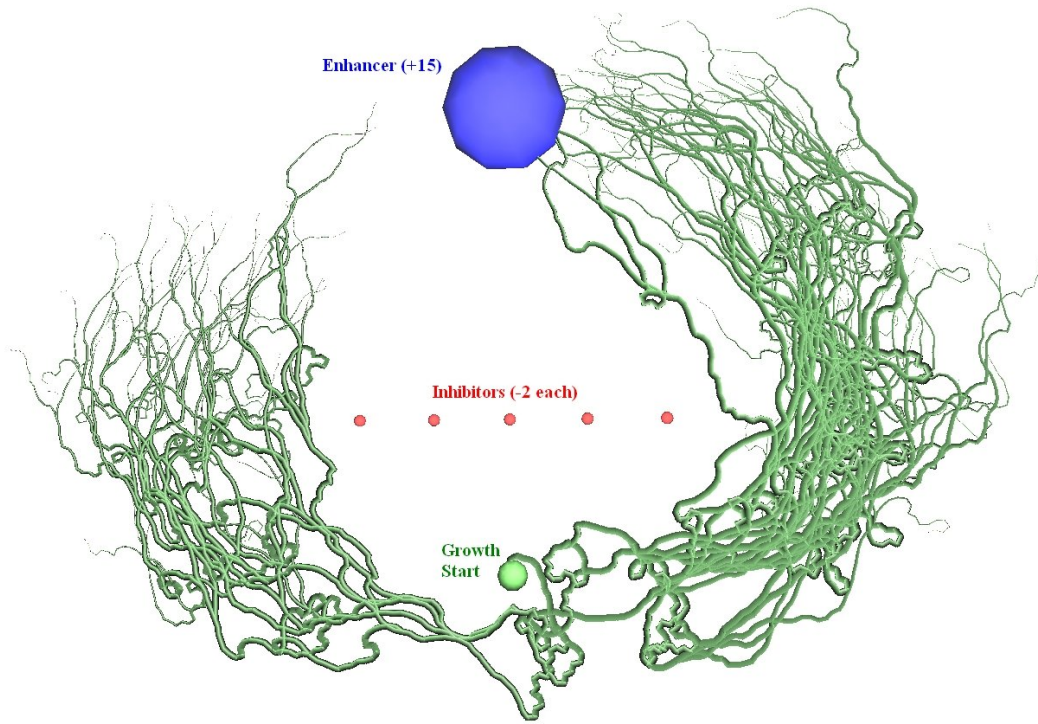
These methods are useful under different circumstances. The first is excellent for more advanced shapes, and where accuracy is preferred, while the second is excellent for quick, simple tests. Both will be used for this project, to cover different tiling operations. (The accurate method will be used to initialise the voxel space with the original obstacles, while the faster method will be used to tile the cylindrical segments.)

As well as the resources listed above, I have also consulted a number of other references [4, 14, 15] on the topic of Tiling Methods. However these references, while providing excellent background information, were not relevant enough to include in this paper.

### 3.4. *Inhibitors and Enhancers*

For the purpose of this project, inhibitors and enhancers will refer to simulated, static objects which influence growth systems in a virtual environment. (Another term used in this section is “source”, which can refer to either an inhibitor or an enhancer.) While they do not have a limit on their area of influence, the amount that a source can affect growth systems around themselves decreases proportionate to the distance away from the source. While inhibitors dissuade growth, enhancers encourage growth. (See Figure 3-6) A growth system would tend to grow in the direction of enhancers,

while leaving the areas near inhibitors empty. It is usually the case that the light affecting the growth of a new segment is defined by the sum total of the light values for all the voxels that the new segment will occupy.



**Figure 3-6** The affect of enhancers and inhibitors on a growth system. The intensity of each inhibitor and enhancer is shown in brackets. (Image produced by an initial version of the software component of this project.)

### 3.4.1. Modelling Light

The simplest method for integrating inhibitors and enhancers into a Voxel Space Automata (VSA) is to leave them as external entities of the system. Whenever the light value for a specific voxel needs to be calculated, it can be done quite simply by averaging the intensities of all the surrounding sources and factoring in the distances between the voxel and each source. The fact that this would have to be done for every new voxel that the segment would occupy makes it a very slow method.

Another method was introduced by Greene [7]. This method made use of the existing voxel grid by calculating the light intensity for every voxel as a preprocessing step. While this did increase the time taken to initialize the VSA, the actual simulation ran much faster than it would have using the method in the previous paragraph. This method of storing lights values in the voxel grid also allowed for a quick comparison between different possible segments. (This in turn produces a better final structure, as many different possibilities can be compared and the best chosen.) However, the disadvantage of this method is that it does not account for the possibility that the amount of light affecting a voxel will be diminished by shadows caused by the new segments of the plant structure. One possible solution to this is to recalculate the light intensity values for every voxel at every iteration; however this would negate the increase in speed that the method previously benefited from.

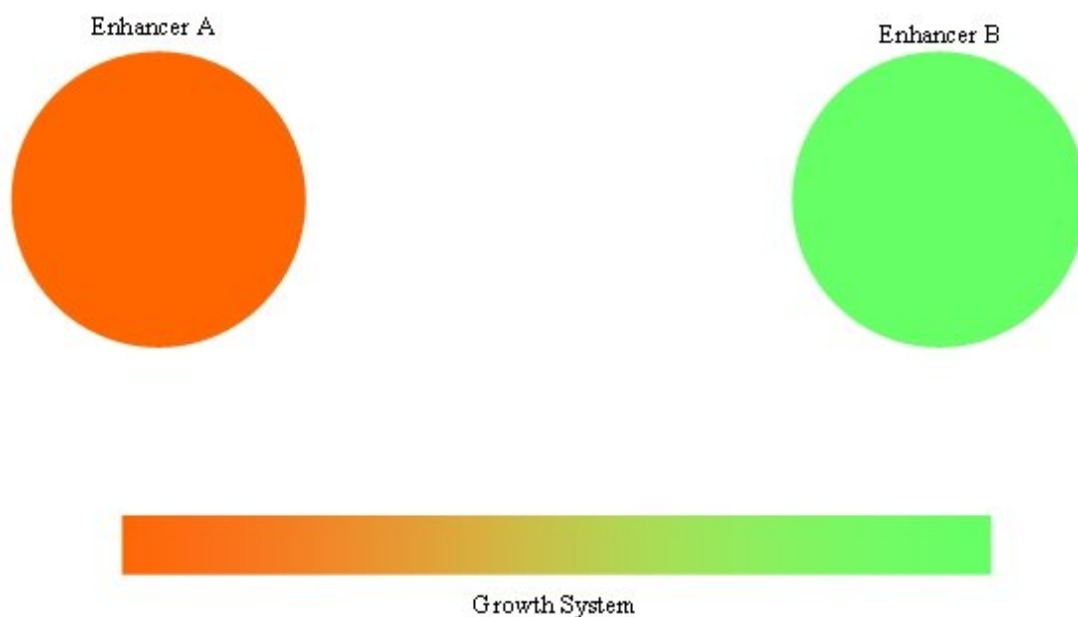
The main disadvantage of Greene's method is that he only takes two different light sources into account; the sun and the sky. The light from each is calculated by a process of shooting rays from each voxel to either the sun or the sky dome, (a hemispherical dome representing the sky, which covers the entire virtual environment) and calculating the final light intensity values based on the number of

rays that reach their target. (i.e. Rays that intersect with an obstacle before hitting their target indicate some amount of shade at that voxel.) While this implementation of Ray-Tracing to extract light intensity values is useful, the fact that only the sun and the sky are examined is a noticeable disadvantage. This project will contain inhibitors and enhancers that can be changed by the user. (i.e. The position and intensity can be modified.)

Another approach for determining the amount of light affecting a growth system is described by Beneš [3]. He uses a virtual camera to simulate a point in the sky, and calculates the exposure of the tree to that point in the sky based on the amount of pixels that are shown to the camera. The exposure from many different points in the virtual sky is used to determine which direction the individual segments should grow. This method is useful for making the growth system as a whole move towards some common point, and in particular is advantageous in the fact that it correctly models the way that light is absorbed by a real tree. However, it suffers the problem that it grows as a single object, as opposed to each part growing independently. (This independent growth is one of the main objectives of this project.)

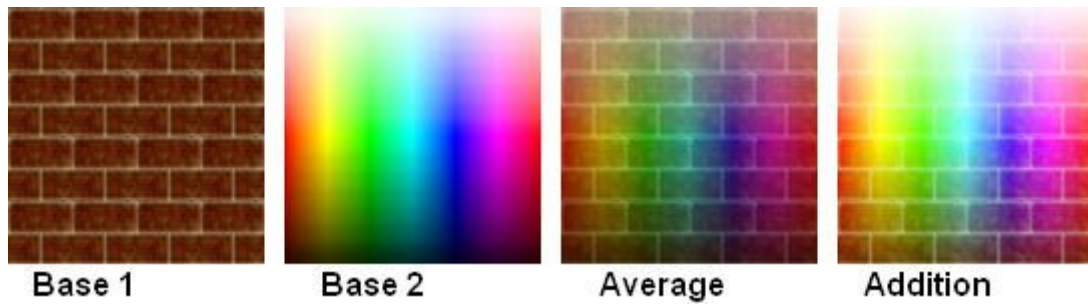
### 3.4.2. Texture Mixing

In addition to controlling the direction of growth, the sources will also control the appearance of every section. This will be done through a process of texture mixing. Each source will have a single cylindrical texture (A texture that can seamlessly wrap around a cylinder) associated with it, and will apply this texture to segments of the growth system. Textures from different sources will be mixed based on their proximity to the segment. (Figure 3-7)



**Figure 3-7** The appearance of a growth system given two sources, each with a texture of a single flat colour.

Techniques for mixing textures are shown by Eric Chadwick [5]. This site shows a number of different methods, an example of which is shown below. (Figure 3-8) These methods for texture mixing are completed by taking the RGB values (Red, Green and Blue, each with a range of 0 to 1) from a pixel in each image, and applying an operation to each to extract the RGB value for the pixel in the final texture. This is done for every pixel in the texture. (For example average texture mixing takes the average of the RGB values for each base texture, while addition texture mixing adds the two RGB values and truncates any values that exceed 1.)



**Figure 3-8** Two base textures, and the results when mixing them using the average and addition methods. (Taken from [5], *An Artist's Real-Time 3D Glossary - Texture Blending Examples*)

It is most likely that this project will use average texture mixing for two reasons. The first is that the amount of each texture needed will be variable, and averaging works with this very well. The second reason is that adding or subtracting a large number of different textures (say 6 or 7) will always give either white or black, except if the calculations are tightly controlled. (And while this is certainly possible, it is unknown whether or not this will give better results than the simpler, more flexible averaging method.)

As well as the resources listed above, I have also consulted a number of other references [16, 25, 26] on the topic of Inhibitors and Enhancers. However these references, while providing excellent background information, were not relevant enough to include in this paper.

## 4. References

- [1] Amanatides, J. (1987). A Fast Voxel Traversal Algorithm for Ray Tracing. In: *Computer Animation: CG87. Proceedings of the Conference held at Computer Graphics 87*, London, UK, Oct. 1987 (pp. 149-55). Pinner, UK: Online Publications
- [2] Arvo, J., Kirk, D. (1988). Modeling Plants with Environment-Sensitive Automata. In *Ausgraph 88 Proceedings*, Melbourne, Vic., Australia, July 4-8 1988 (pp. 27-33). Parkville, Vic., Australia: Australasian Comput. Graphics Assoc
- [3] Beneš, B. (1997). Fast Estimation of Growth Direction in Virtual Plants Simulation. In *WSCG'97. Fifth International Conference in Central Europe on Computer Graphics and Visualization '97*, Plzen, Czech Republic, Feb. 10-14 1997 (pp. 1-10 vol. 1). Plzen, Czech Republic, Univ. West Bohemia
- [4] Borro, D., García-Alonso, A., Matey, L. (2004). Approximation of Optimal Voxel Size for Collision Detection in Maintainability Simulations within Massive Virtual Environments. *Computer Graphics Forum*, 23(1), (pp. 13-23).
- [5] Chadwick, E. (1999). An Artist's Real-Time 3D Glossary - Texture Blending Examples. Retrieved 1 June 2006, from <http://www.brilliantdigital.com/developers/3dglossary/examples/blending.html>
- [6] Dyer, C. R. (2001). Volumetric Scene Reconstruction from Multiple Views. In: *Foundations of Image Understanding*. (pp. 469-89). Boston, USA: Kluwer
- [7] Greene, N. (1989). Voxel Space Automata: Modeling with stochastic growth processes in voxel space. In *Proceedings of SIGGRAPH '89*, Boston, MA, USA, July 31-August 4, 1989 (pp. 175-184). New York, NY, USA: ACM SIGGRAPH
- [8] Hart, J. C., Baker, B. (2003). Structural Simulation of Tree Growth and Response. In: *Proceedings International Conference on Shape Modeling and Applications*, Genova, Italy, May 7-11 2001 (pp. 151-63). Berlin, Germany: Springer-Verlag
- [9] Huang, J., Yagel, R., Filippov, V., Kurzion, Y. (1998). An Accurate Method for Voxelize Polygon Meshes. In: *IEEE Symposium on Volume Visualization*, Research Triangle Park, NC, USA, Oct. 19-20 1998 (pp. 119-126, 175). New York, NY, USA: IEEE
- [10] Jacob, C. (1996). Evolution Programs Evolved. In *Parallel Problem Solving from Nature – PPSN IV. International Conference on Evolutionary Computation. 5<sup>th</sup> Conference on Parallel Problem Solving from Nature*, Berlin, Germany, Sept. 22-26 1996 (pp. 42-51). Berlin, Germany: Springer-Verlag
- [11] Jones, M.W., Satherley, R.A. (2001). Shape Representation Using Space Filled Sub-Voxel Distance Fields. In *Proceedings International Conference on Shape Modeling and Applications*, Genova, Italy, May 7-11 2001 (pp. 316-325). Los Alamitos, CA, USA: IEEE Computer Society
- [12] Kaandorp, J. A. (1991). Modelling Growth Forms of the Sponge *Haliclona Oculata* (Porifera, Demospongiae) using Fractal Techniques. *Marine Biology*, 110, (pp. 203-215).
- [13] Koza, J. R. (2004). [www.genetic-programming.com](http://www.genetic-programming.com). Retrieved 3 June 2006, from <http://www.genetic-programming.com/>
- [14] Lawlor, O.S., Kalé, L.V. (2002). A Voxel-Based Parallel Collision Detection Algorithm. In *Conference Proceedings of the 2002 International Conference on SUPERCOMPUTING*, New York, NY, USA, June, 22-26, 2002 (pp. 285-293). New York, NY, USA: ACM

- [15] Marmitt, G., Kleer, A., Wald, I., Friedrich, H., Slusallek, P. (2004). Fast and Accurate Ray-Voxel Intersection Techniques for Iso-Surface Ray Tracing. In: *Proc. Vision, Modeling, and Visualization (VMV)*, Stanford, USA, November 16-18, 2004
- [16] Měch, R., Prusinkiewicz P. (1996). Visual Models of Plants Interacting with Their Environment. In: *Proceedings of 23rd International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'96)*, New Orleans, LA, USA, Aug. 4-9 1996 (pp. 397-410). New York, NY, USA: ACM
- [17] Meinhardt, H., Fowler, D. R., Prusinkiewicz, P. (1992), Modeling seashells, In: *Proceedings of the ACM SIGGRAPH Conference on Computer Graphics*, Chicago, Illinois, USA, July 26-31 1992 (pp. 379-87). New York, NY, USA: ACM
- [18] Prusinkiewicz, P. (1993). Modeling and Visualization of Biological Structures. In: *Proceedings Graphics Interface '93*, Toronto, Ont., Canada, May 19-21 1993 (pp. 128-137). Toronto, Ont., Canada: Canadian Inf. Process. Soc
- [19] Prusinkiewicz, P. Hammel, M., Měch, R. (1997). Visual Models of Morphogenesis. Retrieved 17 May 2006, from <http://algorithmicbotany.org/vmm-deluxe/TitlePage.html>
- [20] Prusinkiewicz, P., Hammel, M., Měch, R., Hanan, J. (1995). The Artificial Life of Plants. In: *Course Notes of SIGGRAPH '95*, 1, (pp. 1-38).
- [21] Prusinkiewicz, P., James, M., Měch, R. (1994). Synthetic Topiary. In: *Proceedings of 21st International SIGGRAPH Conference*, Orlando, FL, USA, July 24-29 1994 (pp. 351-8). New York, NY, USA: ACM
- [22] Prusinkiewicz, P., Lindenmayer, A. (1990). *The Algorithmic Beauty Of Plants*. New York, NY, USA: Springer-Verlag
- [23] Prusinkiewicz, P., Lindenmayer, A., Hanan, J. (1988). Developmental Models of Herbaceous Plants for Computer Imagery Purposes. In *SIGGRAPH 88 Conference*, Atlanta, GA, USA, Aug. 1-5 1988 (pp. 141-150). USA: ACM;IEEE
- [24] Sakaguchi, T., Ohya, J., (1999). Modeling and Animation of Botanical Trees for Interactive Virtual Environments. In *VRST'99. Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, London, UK, Dec. 20-22 1999 (pp. 139-146). New York, NY, USA: ACM
- [25] Soler, C., Sillion, F., Blaise, F., De Reffye, P. (2001). A Physiological Plant Growth Simulation Engine Based on Accurate Radiant Energy Transfer (Tech. Rep. No. 4116).
- [26] Van Haevre, W., Di Fiore, F., Bekaert, P., Van Reeth, F. (2004). A Ray Density Estimation Approach to Take into Account Environment Illumination in Plant Growth Simulation. In: *Proceedings of the 20<sup>th</sup> spring conference on Computer graphics*, Budmerice, Slovakia, April 22-24 2004 (pp. 121-131). New York, NY, USA: ACM
- [27] Zhang, H., Wyvill, B. (1997). Behavioural Simulation in Voxel Space. In *Proceedings. Computer Animation '97 (Cat. No.97TB100120)*, Geneva, Switzerland, June 5-6 1997 (pp. 119-126). Los Alamitos, CA, USA: IEEE Coput. Soc. Press