

Modeling Surface and Structure Growth Using Lindenmayer Systems in a 3-D Environment

Markus K. Kangas
mkkangas@mit.edu

Advanced Undergraduate Project

May 16, 1999

Supervisors: Una-May O'Reilly – AI Lab

Professor Peter Testa - Department of Architecture

I. Introduction / Abstract	3
II. Lindenmayer Systems	4
III. Working in Three Dimensions	7
IV. Modularity	9
V. Replication	11
VI. The Environment	12
VII. Software	14
VIII. Structure and Surface Images	15
IX. Conclusions / Future Plans	18
X. Acknowledgements	20

I. Introduction / Abstract

Lindenmayer systems (L-systems) serve as a useful tool for modeling biological growth.

Incredibly complex designs can be represented by an L-system consisting of a language, grammar, and geometrical interpretations. Buildings and surfaces can similarly be created by such growth systems.

This project involves the design of a plug-in for Alias Wavefront which interprets such systems in an interactive environment. This environment adds controlled flexibility to the system which can emulate real-world constraints. These constraints include obstacles which prohibit growth such as a ceiling, and objects that can alter growth such as attractors and repellers. The goal is for this software be a tool for architects and designers to create new surface designs given spatial and structural requirements.

Genetic Algorithms will eventually be implemented into the software allowing for computer-aided design.

II. Lindenmayer Systems

Lindenmayer systems are a particular type of symbolic dynamical system with the added feature of a geometrical interpretation of the evolution of the system. They were invented in 1968 by Aristid Lindenmayer to model biological growth. The limiting geometry of even very simple systems can be

$$V = \{a, b\}$$

$$\omega = a$$

$$p_1 : a \longrightarrow ab$$

$$p_2 : b \longrightarrow ba$$

extraordinary fractals. The *Thue-Morse* System demonstrates biological growth:

Generation 0 - a

Generation 1 - ab

Generation 2 - abba

Generation 3 - abbabaab

Generation 4 - abbabaabbaababba

Geometric interpretations can then be applied to the production rules as follows on a 2-D plane:

Let '+' mean rotate right by ALPHA degrees

Let '-' mean rotate left by ALPHA degrees

Let 'F' mean move forward one unit and draw a line segment

Using this grammar with ALPHA = 90°:

$$W \Rightarrow L$$

$$L \Rightarrow LLR$$

$$R \Rightarrow R$$

We obtain the following output:

Generation 0 – L

Generation 1 - LLR

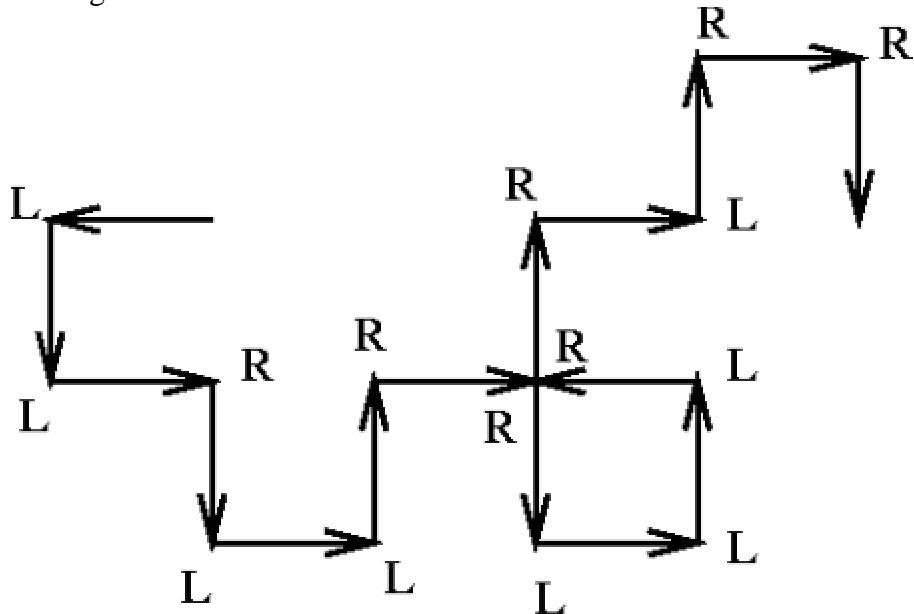
Generation 2 – LLRLLRR

Generation 3 – LLRLLRRLLRLLRRRR

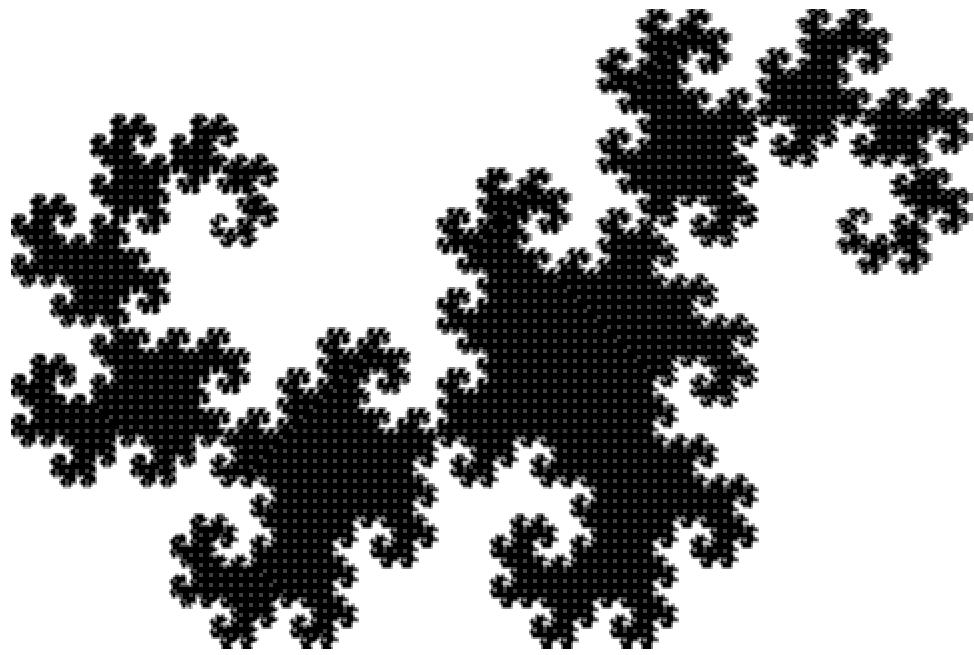
Generation 4 - LLRLLRRLLRLLRRRLRLLRLLRRRR

Replacing every ‘L’ with “F-“, and every ‘R’ with “F+”, we obtain the following image on the

fourth generation:



And on the tenth generation:



(Otherwise known as the Dragon Curve)

Clearly, a very powerful language can be built using very simple geometrical interpretations.

Since many biological forms are branched in appearance and growth, we can add this feature to our language through the symbols [and] to represent push and pop of the turtle state.

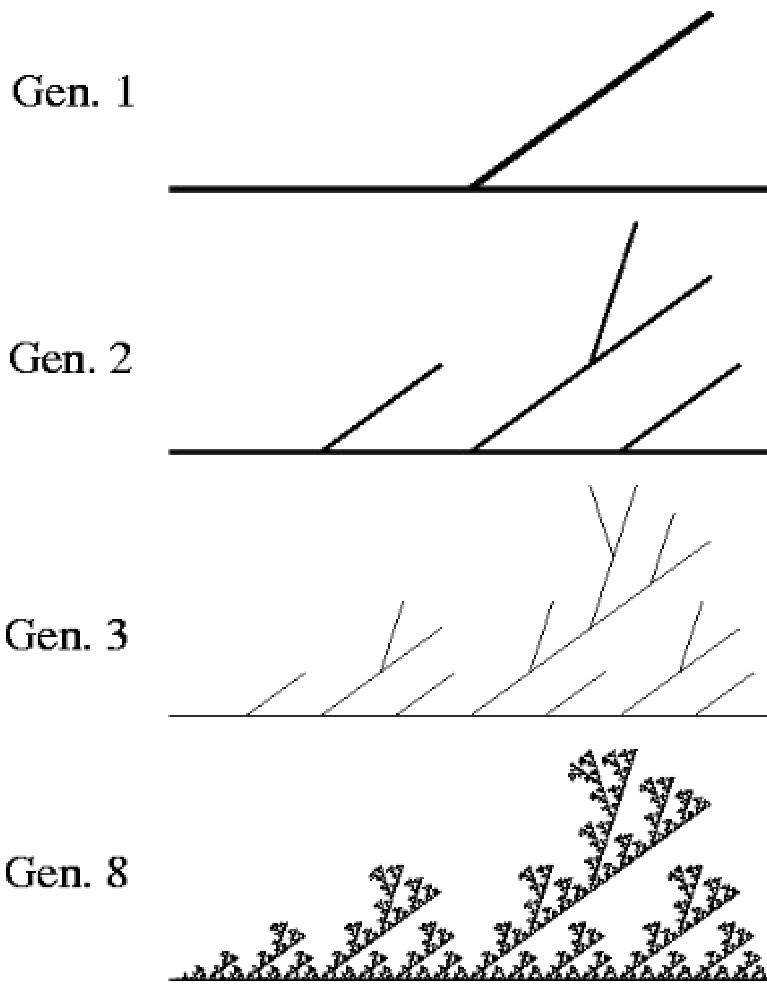
For example, using the following grammar:

ALPHA = 36°

W => F

F => F[F]F

We obtain the following output:



III. Working in Three Dimensions

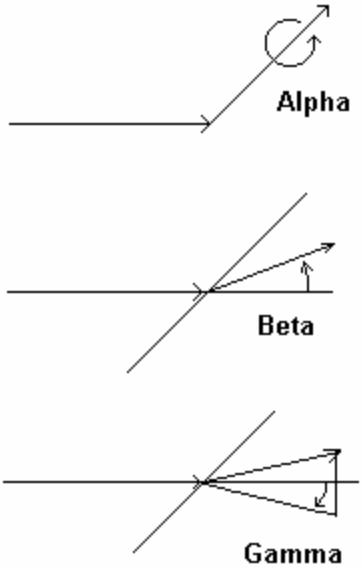
Three Dimensional L-systems require a more advanced geometrical interpretation system.

Although two angles are minimally required to create a vector in 3-D, using three can make the language easier to work with.

Let ALPHA represent the roll, modified by symbols + and -

Let BETA represent the pitch, modified by symbols ^ and &

Let GAMMA represent the yaw, modified by symbols < and >



Different ALPHA, BETA, and GAMMA values allow for vastly different base geometries. For example, 30 degrees would allow easy creation of triangles and squares by using the string “F>>>F>>>F” and “F>>F>>F>>F” respectively. Setting a base value of 1 degree would allow massive geometrical flexibility at the cost of lengthy grammar strings.

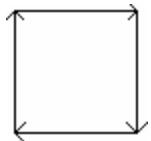
We also introduce the symbol ‘f’ here which causes the turtle to move one unit forward without drawing a line segment.

IV. Modularity

The following grammar defines a square surface, a cube, and a 2x2x2 block of cubes to illustrate the modularity of L-systems:

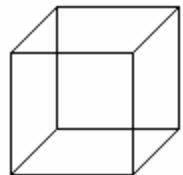
ALPHA = 90°, BETA = 90°, GAMMA = 90°

S => F > F > F > F



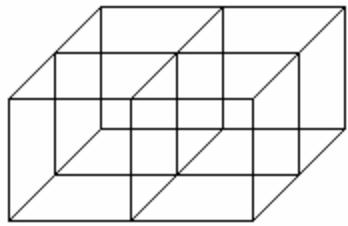
The square surface – rather straightforward

C => S f ^ S f ^ S f ^ S f ^ [< ^ S] f > f ^ S



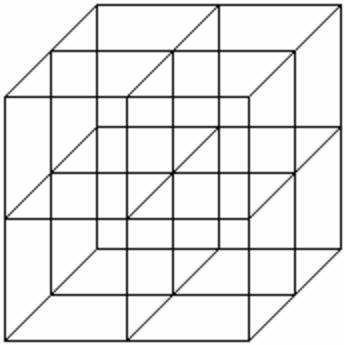
The cube, makes 6 calls to S to form each face. Initially 4 faces are formed by moving forward and pitching upwards making a face each time. Afterwards an extra pitch is performed to return the turtle to its initial position. Then a bracketed section is called to form the top of the box without changing the turtle position. Finally, the last surface is placed with a little tricky turtle movement.

$B2 \Rightarrow C f C > f C f > f C$



This sub-base object is created by placing 4 cube objects in a 2 by 2 square – somewhat similar to the definition of S.

$B \Rightarrow B2 \wedge f \& B2$

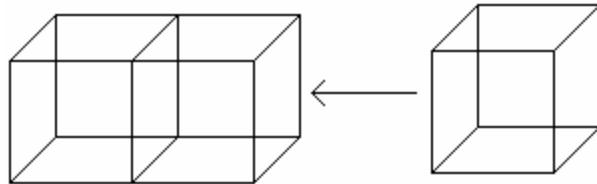


The Base is formed by placing one 2x2x1 sub-base object on top of another. After calling B2, the turtle is pitched up, moves forward one unit, then pitches down to a point directly above the origin.

V. Replication

To achieve replicating behavior, the grammar requires a simple modification:

$$W \Rightarrow C f W$$

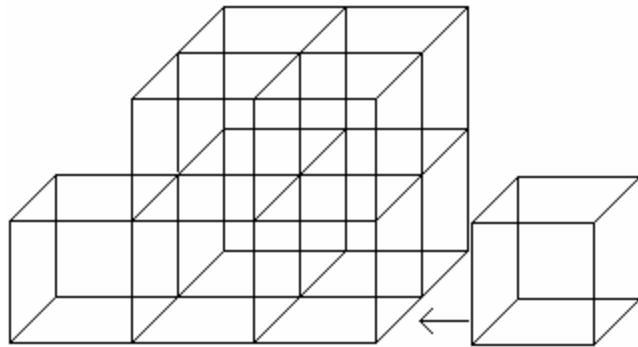


This produces an infinite sequence of cubes with a new cube added each successive generation.

Similarly, an infinite alternating set of cubes and blocks can be created:

$$W \Rightarrow C f V$$

$$V \Rightarrow B f f W \text{ (note two 'f' calls since a block is two units in length)}$$

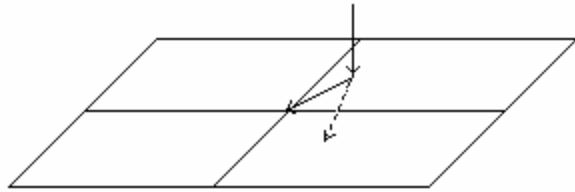


VI. The Environment

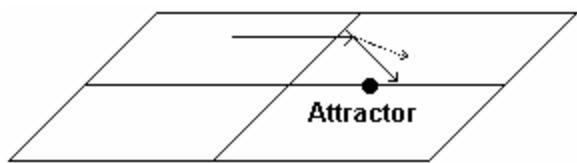
The environment in which an L-system operates will ultimately decide the final geometry of a surface or structure. A resource level or setting a generation limit can decide how large the system will grow. A ceiling and floor can force growth in lateral directions. Regions of space can be defined as impassible forcing the L-system to grow around it. In addition, attractors and repellers can be placed in space promoting or discouraging growth.

Setting a resource limit provides a handy way for creating objects with limited resources. Since material is generally proportional to cost and complexity in the real-world, a designer can theoretically find cheaper solutions by maintaining a lower limit. Setting a generation limit instead will limit growth from its origin radially. This can be useful when spatial distances are important.

A ceiling or floor is simply defined as a plane that prohibits growth through it. Modification of the segment placing routine within the simulator will force the would-be segment to be shifted to a legal position while retaining the same length. In the case that a new segment hits it orthogonally, a random new direction would be chosen upon contact with the plane.

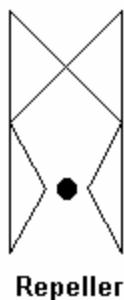


An attractor or repeller is simply defined as a point where growth is promoted or discouraged. The effect of an attractor is demonstrated below:



Any number of attractors or repellers can be added to the environment creating a very complex system. Each segment in space is affected by each and every attractor/repeller which can significantly alter generated geometry.

However, one important issue does arise regarding attractors/repellers and surfaces. Joints of surface plates which normally would join are sometimes bent away from each other leaving gaps. Although this models the effect properly, surfaces do become discontiguous. These gaps can be removed as we discuss later in section VIII. Below is a picture of an extreme case:



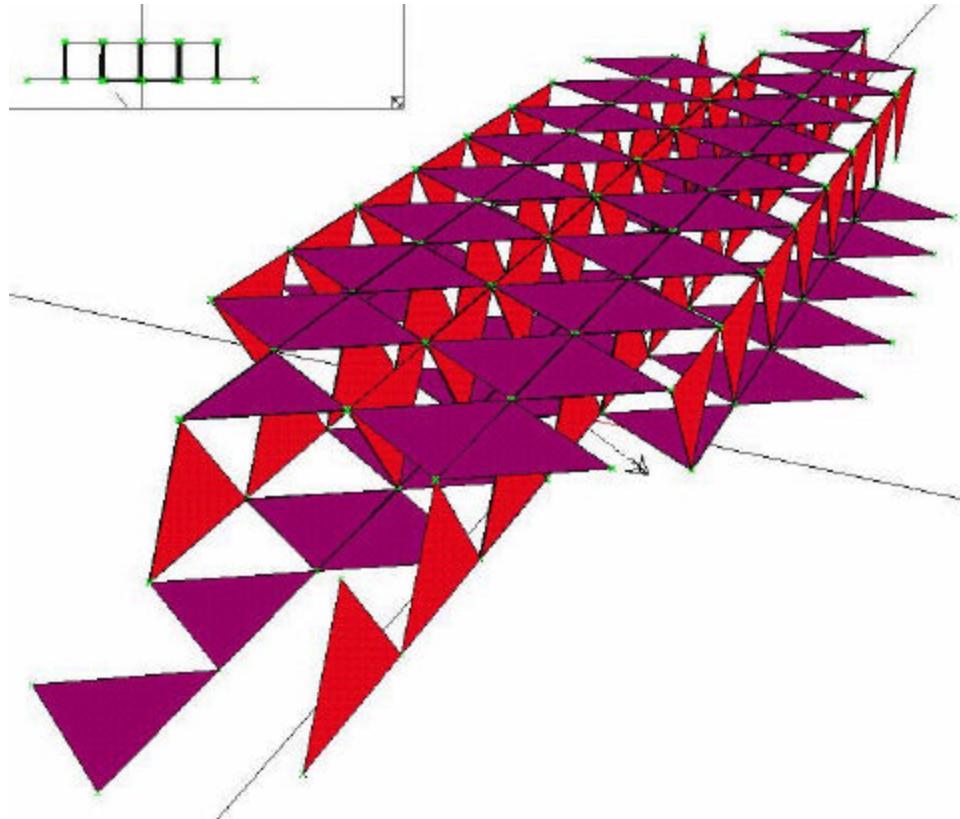
VII. The Software:

The software is programmed completely in C++ as a continuous OpenAlias plugin for Alias Wavefront (version 8.5). Wrappers allow the program to interact with the powerful 3-D environment that Alias provides. Detailed documentation on designing plugins for Alias can be obtained by emailing aliasdocs@aw.sgi.com.

A SGI hardware platform is required to compile and run the code. Since Alias is native to a 32-bit environment, running the L-system software on a 32-bit machine such as an Indy tends to be more stable than on a 64-bit O2. In addition, the installation of Alias requires the optional OpenModel / OpenAlias package for the C++ libraries.

Since no GUI currently exists, the source code (*tree.c++*) must be modified in order to change the grammars and environmental variables. The global struct *grammar* created in *grammar_init* is where new grammars can be entered in a key, value sequence. Similarly, the *repeller* object is initialized in *repeller_init*. All other environmental variables are defined as constants with text descriptions at the beginning of the code.

VIII. Structure and Surface Images



The Grammar for the above image:

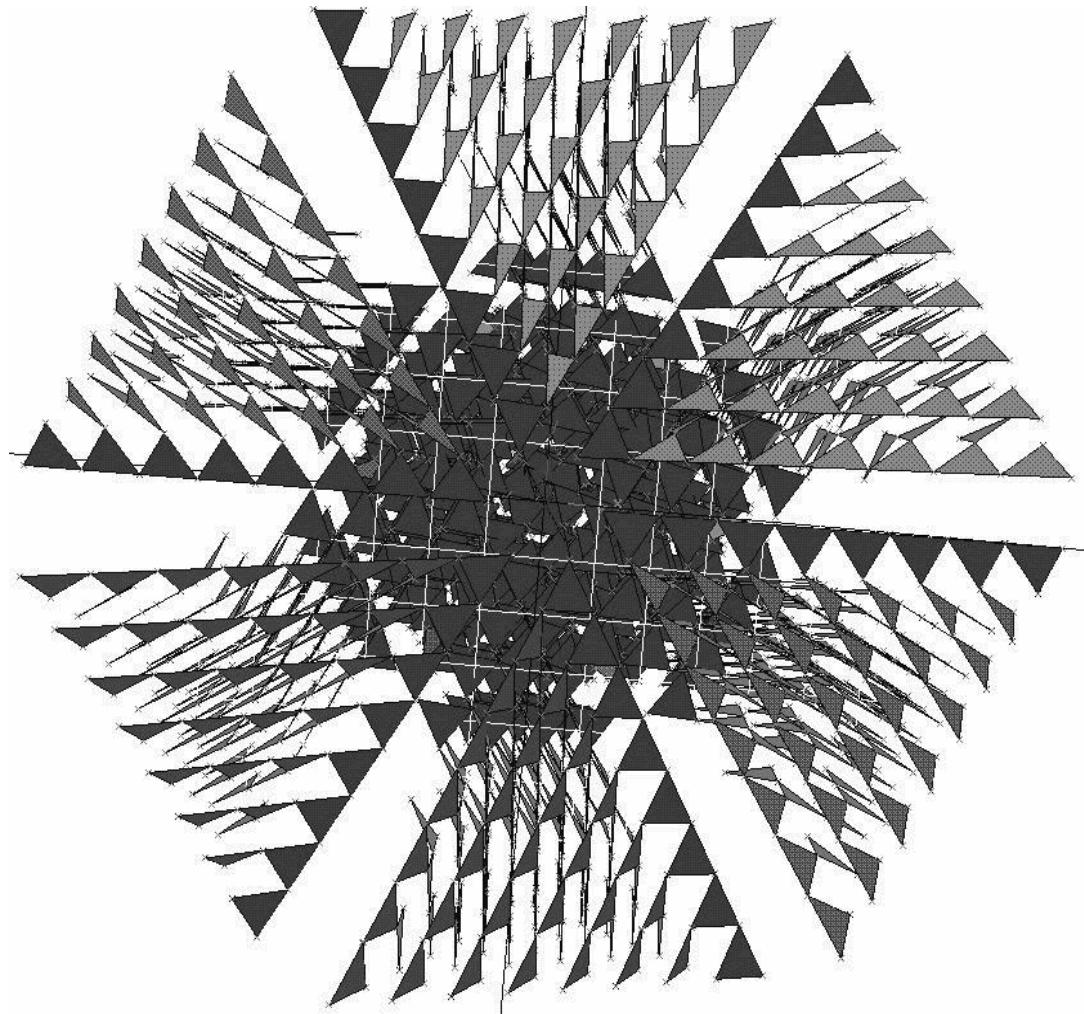
ALPHA = 30°

BETA = 30°

W => f & & & f S

S => F S & & & & & + + + F S + + + & & & & F + + + ^ ^ ^ S

Although this grammar appears complex at first, one notices the cubic nature of the geometry is indicated in the two sets of 3 pluses. In addition, the triangular surface shape is obvious in the 3 F calls



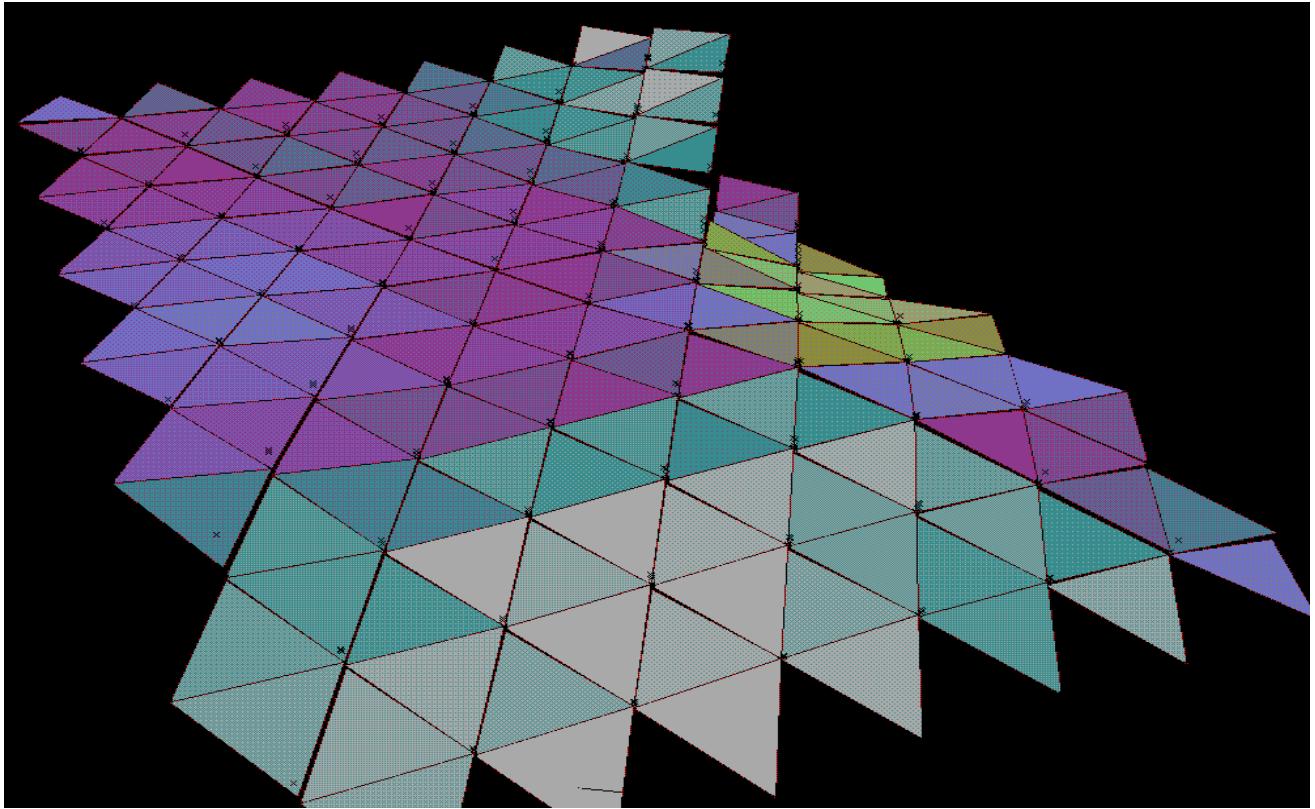
in S. The effect of W is also clear in the two layers of the structure.

ALPHA = 30°

BETA = 30°

W => f & & & [f + S] ++ [f + S]
S => F S & & & & & + + + F & & & & + + + F + S

This image represents the top view of the generated structure after nine generations. The underlying hexagonal geometry is indicated in the definition of W. From a closer inspection of S, one notices linear growth in the first call to S, but curved and deviated growth in the second call.



ALPHA = 30°, GAMMA = 30°

X => f & & & f >>>> f K <<<< f << K

K => Y [>> f << K] >>>>> Y

Y => S [>> S f << S] [<< S] f Y

S => f >>>>> f >>>> f

The output is shown after nine generations. The L-system defined here creates a parallelogram-shaped, triangle-based surface parallel to the x-y plane. The undulation is caused by the effect of a single attractor and a single repeller on the system. The continuous surface is maintained through the use of a

filter which connects slightly detached vertices by slightly elongating the edge lengths. Surface shading was also incorporated in this figure where red facets indicate stressed regions where edge lengths required more severe alterations, and blue facets indicate relatively unperturbed regions.

IX. Conclusions / Future Plans

The L-system plugin-in is a powerful tool to study structure and surface growth in a three dimensional environment. Arbitrary systems can be implemented with a wide spectrum of results.

However, the effect of environmental variables is dramatic and requires further study. A regular, continuous surface is bent by attractors/repellers leaving non-contiguous joints and openings in the surface. These spatial irregularities need to be smoothed in order to create a smooth, continuous output surface. Currently this is done by stretching facet edges. Other methods that have not yet been implemented include creating a wire-mesh around the outermost vertex points and building a spline network around the vertices. Facets which are altered significantly by such a process are colored differently to provide the user with some design feedback.

Eventually GA's (Genetic Algorithms) can be implemented into the simulator to allow for

computer-aided design of structures or surfaces. Crossovers and mutations of L-system grammars is relatively simple. However, a fitness measure is difficult to design since both aesthetic and physical results are important. The easiest means is for the user to tediously decide among the resulting objects, with obvious aberrations being removed ahead of time by a filter (such as physically impossible structures which can be detected by the simulator).

In addition, the GUI is almost non-existent and must be improved significantly before it can be used as a tool by designers. Since functionality remains the top-priority, work on this will need to be delayed further.

X. Acknowledgements

References

1. Przemyslaw Prusinkiewicz. Visual Models of Morphogenesis. *Artificial Life*.
2. Tatsuo Unemi, Takeshi Koike. Evolution of a Botanical Development System in 3D Euclidean Space. *Artificial Life*.
3. Chikara Furusawa, Kunihiko Kaneko. Emergence of Multicellular Organisms with Dynamic Differentiation and Spatial Pattern. *Artificial Life*.
4. Peter Eggenberger. Evolving Morphologies of Simulated 3D Organisms Based on Differential Gene Expression. *Fourth European Conference on Artificial Life*.
5. Brian Clarkson, Hyung-Jin Kim. *Gentically Recombinant Modelz*.
6. Rodney Brooks, Cynthia Breazeal. *Embodied Intelligence*.