On Lindenmayer Systems: Background and Applications

David Hofferber[1]

Abstract— **This paper covers the background of Lindenmayer Systems (L-systems) and their various applications with respect to geometric modeling. We discuss the practical implications of L-systems as a system of geometric modeling, and discuss potential limitations that such systems may hold.**

I. INTRODUCTION

Lindenmayer systems (L-systems) were originally composed by Aristid Lindenmayer in 1968 as a method of describing the behavior of organic models, most notably systems such as plant cells and plant growth models [4]. L-systems are expressed as a parallel rewriting system, and a type of formal grammar, leading to work which focused on their ability to be extended into more complex languages, notable variant including context sensitive and stochastic grammars. From their interesting properties, L-systems have been used most notably in the generation of fractals and the realistic modeling of complex structures.

A very central concept to the functionality of L-systems is their rewriting system, which allows for one to define complicated objects using a set of production rules. One desirable property is that this rewriting can be carried out recursively. As opposed to Chomsky grammars, in which productions are applied sequentially, L-systems reflect the biological motivation of their existence by utilizing its production rules in parallel. Similar to cellular organisms, which may have numerous cell divisions occurring at once, L-systems allow multiple production rules to be applied at once.

This work was done for credit in CS 396/496: Computational Geometry, taught by Huck Bennett.

H. Bennett is currently a postdoctoral fellow with the Department of Computer Science, Northwestern University. hbennett@eecs.northwestern.edu

[1]D. Hofferber is currently an undergraduate student in the Department of Computer Science, Northwestern University. dch@u.northwestern.edu

II. CLASSES OF L-SYSTEMS

As discussed by Prusinkiewicz et al, L-systems exhibit a number of desirable properties that have allowed them to be successfully applied towards complex modelling tasks. As mentioned in [7], we see that its success comes most notably from the following facts:

1) L-systems specify the development of structures in terms of temporally and spatially local declarative rewriting rules, allowing for both context-free and context sensitive productions. This allows L-systems to express most developmental processes in a very compact and intuitive manner.

2) L-systems describe the growth of structures in terms of local topological relationships between internal components. Such relationships are automatically maintained regardless of how components are added or removed from the structure, allowing for the application of production rules to be evaluated in any given context.

3) L-systems refer to components in a model by their type, state, and contextual information. As opposed to common programming paradigms, which require the specification of unique variable names and indices, L-systems allow for the development of complex structures via rules which define the relationship between a structures internal components.

Given these properties, we now discuss several different classes of L-systems, and how their extended properties allow for a more robust set of objects to be defined.

A. DOL-systems

In this section, we introduce the simplest class of L-systems, termed *DOL-systems*, standing for Deterministic and Context Free systems. An example of such an L-system with axiom ω and production rules p_1, p_2 is the following:

$$\begin{aligned} \omega: &\quad a \\ p_1: &\quad a \rightarrow b \\ p_2: &\quad b \rightarrow ab \end{aligned} \tag{1}$$

Our initial axiom, ω, notes that our initial state is composed of a. Our production rule p_1 states that any a in our system is turned into b, and our production rule p_2 states that any b is turned into ab. Following several iterations of this, we have the following:

$$\begin{aligned} 1: &\quad a \\ 2: &\quad b \\ 3: &\quad ab \\ 4: &\quad bab \\ 5: &\quad abbab \end{aligned} \tag{2}$$

The sequence may look familiar, as the string length of this L-system represents the Fibonacci sequence. As noted, the understanding of DOL-systems is relatively straightforward, with the production rules being evaluated against a given state in parallel.

B. Stochastic DOL-Systems

An extension of the previous system, we can avoid regularity in a given L-system by specifying different productions the same symbol and choosing between them based on a given probability distribution. For example, the previous rule p_2 could be expanded into the following:

$$\begin{aligned} p_{2i}: &\quad b \xrightarrow{0.7} ac \\ p_{2j}: &\quad b \xrightarrow{0.3} ab \end{aligned} \tag{3}$$

Where p_{2i} is applied with probability 0.7, and p_{2j} is applied with probability 0.3. In this case, due to the fact that they involve the same rules, only one of the production rules would be applied at once.

C. Bracketed DOL-Systems

With the addition of brackets to a language, we have the following capabilities given our two new characters, [and]:

1. [Pushes the current state of the system onto a stack. The information saved contains the current position and orientation, and possibly other contextual information commonly associated with given L-systems (e.g. color, or line width).

2.] Pops a state from the stack and makes it the current state of the system.

As an example, consider a common example of a bracketed L-system utilizing turtle graphics to be drawn in Figure 1, given the bracketed DOL-system described below:

$$\begin{aligned} \delta: &\quad 25.7^\circ \\ \omega: &\quad X \\ p_1: &\quad X \rightarrow F[+X]F[-X]+X \\ p_2: &\quad F \rightarrow FF \end{aligned} \tag{4}$$

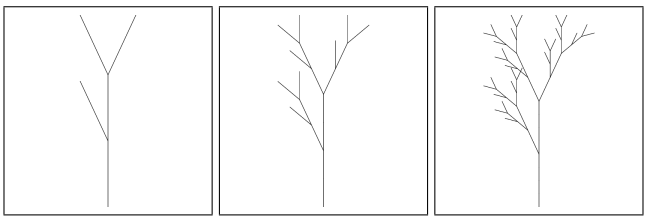

Fig. 1: A sequence of iterations on the example bracketed DOL-system, for iterations 1, 2, and 3.

Note that for visualizing the L-system, we specify an angle δ which describes the angle at which things should turn, with + representing a right turn by δ degrees and - representing a left turn by δ degrees.

D. Context-sensitive L-systems

As described previously, we see that productions for DOL-systems are context-free, meaning that production rules are applicable regardless of the context in which the predecessor(s) appear. A variety of context-sensitive extensions of L-systems have been proposed, with a commonly

used extension being that of 2L-systems. A 2L-system may have a production rule in the form:

$$p_1: \quad X_l < X > X_r \rightarrow Y \tag{5}$$

Which states that the letter X, known as the strict predecessor, can produce the letter Y if and only if X is preceded by letter X_l and followed by letter X_r. Thus, letters X_l and X_r form the left and right context respectively. We note that there are a variety of other extensions with similar properties. 1L-systems, for example, have a one-sided context only. In general, these systems fall under a wider class of IL-systems, also known as (k,l)-systems, which generalizes this concept to state that the left context is a word of length k and the right context is a word of length l.

Generally, we find that the specifications of IL-systems have been abstracted by allowing productions with different context lengths to exist within a system L-system. This allows less complicated definitions of L-systems, as generally we see that context-sensitive productions are assumed to have precedence over context-free productions which hold the same strict predecessor. If no production applies, then we assume that the letter is replaced by itself.

As an example, consider the following 1L-system which transfers a given symbol from one side of an L-system to another:

$$\begin{aligned} \omega: &\quad Xyyyy \\ p_1: &\quad X < y \rightarrow X \\ p_2: &\quad X \rightarrow y \end{aligned} \tag{6}$$

Iterating through our initial axiom using these production rules, we obtain the following:

$$\begin{aligned} 1: &\quad Xyyyy \\ 2: &\quad yXyyy \\ 3: &\quad yyXyy \\ 4: &\quad yyyXy \\ 5: &\quad yyyyX \end{aligned} \tag{7}$$

As seen above, context-sensitive L-systems can be utilized to interact with a given L-system in an interesting manner, allowing us to modify components based on the contextual information of the system given other parts of its topological information.

As discussed in Chapter 1 of [6], regarding the topic of context sensitive L-systems, we see that the introduction of context to bracketed L-systems is much more difficult than in L-systems without brackets due to the fact that bracketed strings do not preserve the topological neighborhood of its context. Due to this, we find that context matching in bracketed systems may find it necessary to skip over symbols that represent branches. As described in the book, we find that a production with the predecessor $BC < S > G[H]M$ can be applied to symbol S in the following system:

$$ABC[DE][SG[HI[JK]L]MNO]$$

Note that this is only possible by skipping over the symbols $[DE]$ in the search for the left context, and $I[JK]L$ in the search for the right context.

E. Other L-Systems

For the purposes of this paper, we will not explore other extensions of L-systems. However, it may be useful to the reader to note that there are a variety of other extensions on L-systems which exhibit interesting properties. Some examples of these include parametric L-systems, including the natural extension of parametric OL, 1L, and 2L systems. Other interesting extensions include the introduction of bi-directional grammars.

III. Application towards Geometric Modeling

A. B-splines through the Chaikin Algorithm

As discussed in [7], we find that a simple example of the application of L-systems to geometric modeling comes in the form of an L-system specification of the Chaikin algorithm. Given a closed polygon $P(v_0)P(v_1)\ldots P(v_m)$, this algorithm will produce a smooth curve by iteratively cutting the

corners of the control polygon and its descendants. This process can be described through the following context-sensitive L-system:

$$\begin{aligned} \omega: &\quad P(v_0)P(v_1)\dots P(v_m) \\ p: &\quad P(v_l) < P(v) > P(v_r) \\ &\quad \to P\left(\frac{1}{4}v_l + \frac{3}{4}v\right) P\left(\frac{3}{4}v + \frac{1}{4}v_r\right) \end{aligned} \tag{8}$$

Where $P(\alpha_1 v_1 + \alpha_2 v_2)$ represents the affine combination of points v_1 and v_2.

In the figure below, we see the application of Chaikin's algorithm to the user defined polygon (the square to the left). Note that at each iteration, the algorithm removes the corners of the polygon at each state (i.e. the red corners), and replaces each vertex with a pair of new vertices. The final step shows an approximately smooth curve obtained after 4 derivation steps.

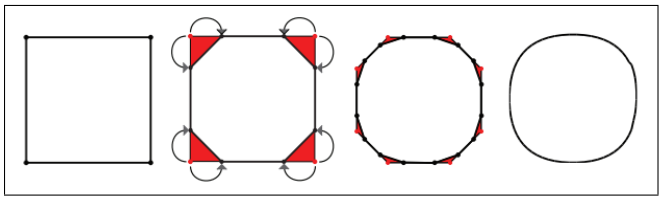

Fig. 2: A sequence of iterations of Chaikin's algorithm on a square.

With some slight modifications, this algorithm defined by our context sensitive L-system can generate B-spline curves of degree $n+1$ after n applications. This is a prime example of how useful L-systems can be in geometric modeling, as B-splines are widely used in the field due to their well understood geometric properties and the relative ease in which they can define various curves by interactively placing control points.

B. The de Casteljau Algorithm

We now consider how L-systems can be utilized to implement an incredibly useful algorithm in the field of geometric modeling, namely the de Casteljau algorithm. Given a control polygon $P(v_1)P(V_2)\dots P(v_n)$ with $n \geq 2$ vertices P located at the point v_i, the algorithm constructs a polygon $P(w_1)P(w_2)\dots P(w_{n-1})$ such that each point $P(w_i)$ divides the line segment $\overline{P(v_i)P(v_{i+1})}$ in proportion $t : 1-t$, where $t \in [0,1]$. This process is iterated $n-1$ times, ending with a single point $P(z_1)$. This point is called the locus, and its value for all t is the Bézier curve of degree $n-1$ defined by the control points of our inputted polygon.

Bézier curves can be seen as a specific form of B-splines, and have similar applications due to their properties of being able to easily change their shape by manipulating the control points. We see that the main operation of the de Casteljau algorithm can be defined as the following system:

$$\begin{aligned} \omega: &\quad P(v_1)EP(v_2)E\dots EP(v_n) \\ p_1: &\quad P(v_l) < E > P(v_r) \to P((1-t)v_l + tv_r) \\ p_2: &\quad E < P(v) > E \to E \\ p_3: &\quad P(v) \to \varepsilon \end{aligned}$$

Where the input is a series of points $P(v)$ and edges E, and where our production rule p_3 erases the given vertex $P(v)$.

We note that this L-system can be extended to complete the de Casteljau algorithm by subdivision. This is done by iterating on the results of the the above L-system, allowing us to approach the Bézier curve to arbitrary precision as we increase the number of subdivisions we utilize. The final L-system is described explicitly in [7] on page 8.

IV. Tensor Product Surfaces

Another application of L-systems towards geometric modeling comes from modeling subdivision surfaces. In the previous section, we saw that we could generate L-systems which approximate curves in the form of B-splines and Bézier curves. We now will look at a more complex case, namely that of utilizing L-systems for describing the subdivision of tensor product surfaces, as seen in [3].

We begin by noting that the Bézier bicubic tensor product surface is given by sixteen control vertices, represented as a 4x4 matrix P. From the

matrix P, another matrix can be computed using the de Casteljau scheme. We describe this as the production rule of the form:

$$P_i \rightarrow P_{i+1}$$

As we iterate through this process, we obtain the limit of our system which is the surface itself. While we will not go into large detail into this system, we find that this more complicated L-system naturally extends the process we saw previously into 3D surfaces, as seen in the figure below:

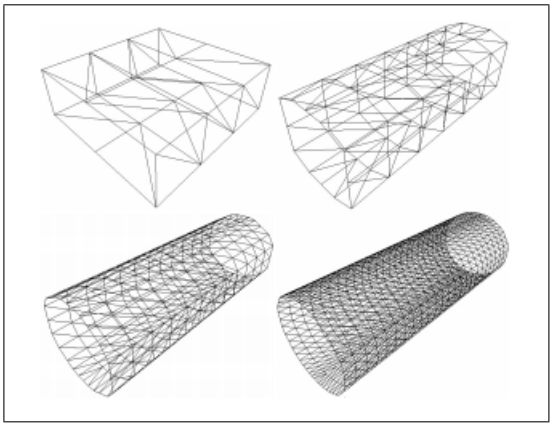

Fig. 3: A rational Bézier bicubic surface representing a cylinder as the result of the L-system seen in [3].

V. Procedural City Generation

In [5], we see a practical example of how L-systems can be used for modeling other complex systems. In many cases, one may want to procedurally model or generate complex environments, such as cities. Given various information, such as street patterns, population density, land/water/park boundaries, and elevation, we find that the L-system proposed in [5] does well at modeling the realistic generation of city roads.

As noted in the paper, most real cities display various road patterns due to historical growth and phases associated with the creation of the city. They adopted several patterns to describe trends that have been seen internationally in such city development:

1) The Basic Rule - This is the simplest rule they made, which has no superimposed pattern and with all roads following the population density. Typically older cities show this pattern, as this is simply the natural growth of a transportation network as a city grows larger.

2) The New York Rule - Given a global or local angle, and the maximal width and length of a single block, we find a frequent street pattern encountered in urban areas. This pattern describes the natural formation of rectangular blocks in such cities.

3) The Paris Rule - The highways follow radial tracks around a given center point.

4) The San Francicsco Rule - This pattern shows how streets follow the route of the least elevation. Roads on different height levels are connected by smaller streets, which follow the steepest elevation and tend to be much shorter than main roads. This pattern is usually observed in areas with large differences in ground elevation.

In Figure 4, we see how the iterative rewriting of the self-sensitive L-system described in [5] performs when attempting to model the street system of Manhattan, as compared to a real map of Manhattan streets. The algorithm performs quite well, especially for the purpose of general street generation. As noted in the conclusion of their paper, we find that more realistic city generations could be found by additional rules for global road creation, addition of stochastic systems showing the simulation and analysis of growth of the city, and better mechanisms for how one would realistically generate buildings on top of allotted segments.

VI. Computation of Fractal Dimensions

For the purposes of this paper, we focused mainly on applications towards geometric modeling and other similar practical applications. Due to this, we did not explore on how easily L-systems are able to represent fractal curves.

Fig. 4: Street creation as seen during various iterations of the L-system in [5]. The top row shows iterations after 28 and 142 steps. The middle shows the final roadmap, while the bottom shows a real map of the streets of Manhattan.

Perhaps one of the most well researched topics regarding L-systems is their ability to easily and compactly represent fractal geometries with dimensions between 1 and 2, as well as their ability to represent similar geometries such as space-filling curves.

As described by Mandelbrot, many people until around 1975 believed that dimensions could only have integer dimensions. With his relatively famous ball of yarn story, he brought up the following points:

1) From very far away, a ball of yarn just looks like a point, i.e. dimension $d = 0$.

2) From slightly closer, the ball starts to look like a circular disk, i.e. $d = 2$.

3) From relatively close to the ball, the ball appears to be 3-Dimensional, with its various curves and bumps now appearing to be spherical, i.e. $d = 3$.

Mandelbrot noted that the dimension of objects seem to depend on the distance from which the ball of yarn is viewed. Utilizing what is known as the Hausdorff-Dimension, or the Minkowski Dimension, one can compute the real-numbered fractal dimension of a given object. However, both of these definitions are somewhat complicated.

L-systems once again show an interesting property, namely that their axioms and production rules can be used to compute the fractal dimension of a given fractal object that an L-system approximates.

A relatively well known fractal curve, the Koch curve, can be represented as the following L-system:

$$\begin{aligned} \delta &: \quad 60^\circ \\ \omega &: \quad F \\ p_1 &: \quad F \rightarrow F+F--F+F \end{aligned} \tag{9}$$

The Koch curve is known to have a fractal dimension of the following form:

$$\frac{\log(4)}{\log(3)} = 1.262\ldots$$

As suggested in [1], the fractal dimension of a curve that is represented by a given L-system can be shown to be equal to the following:

$$D = \frac{\log(N)}{\log(d)} \tag{10}$$

Where D is the fractal dimension, N is the length of the visible walk of the production rule, and N is equal to the number of draw symbols in the generator (so long as the production rule does not overlap). The value d is the distance in a straight line from the start to the endpoint of the walk.

For the Koch curve, we see that the number of draw symbols is 4, and the distance between a the starting and end point is 3 here. Therefore the dimension of this curve is exactly equivalent to the dimension we saw previously.

There are some flaws with this definition. For example, the distance d in the denominator can be 0. However, such systems can be shown to not be fractals, meaning we can exclude such cases.

Additionally, the distance d could also be equal to 1, which would give an infinite fractal dimension. This arises due to the fact that after each iteration the geometry would expand, meaning that are not restricted to a limited space, and are thus not considered fractals.

Finally, the length N of the visible walk may not be equal to the number of draw-symbols in the string (i.e. some parts of the drawn symbols overlap). For these cases, we discover that one can compute the effective walking length via an algorithm described by Ortega and Alfonseca. With this, we conclude that we can adequately find the fractal dimension of an L-system that properly describes a given fractal.

VII. CONCLUSION

In this paper, we explored the background of L-systems and its various applications. We began by noting the formal definition of L-systems, namely as a formal grammar which is expressed in terms of an initial axiom and a set of production rules which are utilized as a rewriting system. We described how they are similar to Chomsky grammars, but with production rules that are ran in parallel rather than sequentially.

We then explored various classes of L-systems, and documented the various extensions that are commonly used and built upon. After describing the more complex classes of L-systems, we proceeded to show how these complex classes of L-systems can be used for the implementation of both theoretical and practical algorithms. We noted how their desirable properties resulted in easily implementing curves commonly used in geometric modeling, such as B-splines and Bézier curves, and showed how this could be extended into higher dimensions in the form of rational Bézier bicubic surfaces.

Finally, we concluded that though our main focus for this paper was to explore the application of L-systems towards geometric modeling, L-systems have desirable properties that are useful for other areas of interest. One notable property is the fact that one can effectively compute the fractal dimension of a given fractal so long as one has an L-system which properly describes the fractal geometry.

REFERENCES

[1] Manuel Alfonseca and Alfonso Ortega. 2001. Determination of fractal dimensions from equivalent L systems. IBM Journal of Research and Development 45, 6 (2001), 797805.

[2] Manuel Alfonseca and Alfonso Ortega. 1996. Representation of fractal curves by means of L systems. In Proceedings of the conference on Designing the future (APL '96), Michael Kent (Ed.). ACM, New York, NY, USA, 13-21.

[3] Ivana Kolingerov, Petr Mrz, and Bedrich Bene. 2006. Tensor product surfaces as rewriting process. In Proceedings of the 22nd Spring Conference on Computer Graphics (SCCG '06). ACM, New York, NY, USA, 107-112.

[4] Aristid Lindenmayer. 1968. Mathematical models for cellular interaction in development, Parts I and II. Journal of Theoretical Biology 18, pp. 280315.

[5] Yoav I. H. Parish and Pascal Mller. 2001. Procedural modeling of cities. In Proceedings of the 28th annual conference on Computer graphics and interactive techniques (SIGGRAPH '01). ACM, New York, NY, USA, 301-308.

[6] Przemyslaw Prusinkiewicz, Aristid Lindenmayer. 2004. The Algorithmic Beauty of Plants.

[7] Przemyslaw Prusinkiewicz, Mitra Shirmohammaddi, and Faramarz Samavati. 2010. L-systems in Geometric Modeling. In Proceedings of the 12th International Workshop on Descriptional Complexity of Formal Systems (DCFS 2010). EPTCS 31, 2010, pp. 3-14.

[8] Przemyslaw Prusinkiewicz, Faramarz Samavati, Colin Smith, and Radoslaw Karwowski. 2003. L-system description of subdivision curves. International Journal of Shape Modeling 9(1), pp. 4159

[9] Sara C. Schvartzman and Miguel A. Otaduy. 2014. Fracture animation based on high-dimensional Voronoi diagrams. In Proceedings of the 18th meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D '14). ACM, New York, NY, USA, 15-22.

[10] Megumi Takato and Yohei Nishidate. 2018. Generating Crack Patterns on Planar Geometry by L-system. In Proceedings of the 3rd International Conference on Applications in Information Technology (ICAIT'2018), Natalia Bogach, Evgeny Pyshkin, and Vitaly Klyuev (Eds.). ACM, New York, NY, USA, 58-62.