Neuron Developmental Modeling and Structural Representation: An introduction to the N++ Language, an Open Stochastic L-System

Richard W. DeVaul
Bruce H. McCormick*
Technical Report
Scientific Visualization Laboratory
Department of Computer Science
Texas A&M University, College Station, TX, USA

December 11, 1996

Contents

*This material is based in part on work supported by the Texas Advanced Technnology Program under Grant No. 9999024 (McCormick)

List of Figures

List of Tables

1 Abstract

This is the first in a series of papers describing a set of tools for the developmental modeling, visualization, and statistical analysis of neuron populations. This paper discusses the use of an open parametric stochastic L-system for the developmental modeling and structural representation of neurons. An overview of the basic principles of L-systems and turtle geometry is presented, followed by an introduction to the concepts and grammar of the N++ langauge, an open stochastic L-system developed to model the growth and morphology of neurons as viewed at the limit of optical resolution.

2 Specifications

The N++ language[1], an open stochastic L-system[3], provides a representational framework for the description of neuron development and morphology. The N++ language produces well-formed sentences, each of which may be geometrically interpreted as a neuron, adult (if composed only of terminal symbols) or immature (composed of at least some non-terminal symbols). The physical growth process of developing neurons is mimicked in the N++ language grammar. This means that the modeled neurons presumably grow the same way real ones do, and that a sentence formed at any stage of the generation process is a "snapshot" of a developing (or developed) neuron.

3 Features

The N++ language was designed with three key features in mind:

1. Accurate growth modeling. The system should accurately model the growth and development of neurons within the confines of neural tissue.

2. Accurate structural representation. The syntax of the representation should reflect the morphology of neurons.

3. Ease of statistical modeling. The syntax should lend itself to the types of statistical analysis likely to be needed for stochastic modeling.

These requirements led to the development of an open stochastic Lindenmayer system for the developmental modeling and structural representation of neurons.

4 Significance

The N++ language was created to address four issues in neuronal development and morphology:

1. What structurally defines cell typology?

2. What constitutes the normal pattern of growth of a cell type?

[1]The symbol N++ (pronounced en-plus-plus) refers to an integrated collection of tools for the stochastic modeling, visualization, and statistical analysis of neurons, one of which is the N++ langauge.

3. What defines the normal morphology of a cell type?
4. When can the morphology of two cell populations be statistically differentiated?

In the N++ language cell typology is defined by the stochastic model. The normal pattern of growth is defined by the N++ language grammar in combination with the stochastic model. The normal morphology of a cell type is defined by the turtle geometry interpretation of the N++ language productions, in combination with the N++ language grammar and stochastic model. Statistical differentiation between two cell population is facilitated by the string representation of N++ neuron models.

5 Lindenmayer Systems

L-systems are a type of formal language first described by Lindenmayer in [1] and [2]. Since L-systems are not well known outside the procedural modeling community a quick tutorial introduction to the concepts and syntax of L-systems is presented here. For further information on L-systems, both as grammar and in application to the geometric modeling of branching structures, the reader is referred to [6].

5.1 Introduction

An L-system is essentially a parallel string rewriting system. Iteration of the system results in a production string being substituted for each non-terminal symbol in the current string, resulting in a new string. These substitutions occur in parallel, meaning that productions depend only on the predecessor string[2]. Further, some or all of the symbols in an L-system may have a graphical or geometric interpretation. This section focuses on the fundamentals of an L-system as a grammar. The geometric interpretation of an L-system is explained in Section 7.2.

An open L-system[3] is a recent development which allows two-way communication between an L-system and its environment. Open L-systems are an extension of environmentally sensitive L-systems, which in turn are an extension of parametric L-systems[5].

Traditional L-system notation describes a system in terms of a start string, or *axiom*, and a set of productions. The notation used here is somewhat different, and borrows from the description of context free grammars as well as classic L-system notation. The set of terminal and non-terminal symbols are explicitly stated along with the set of productions and the axiom. See Section 5.6 for a comparison of the notation used here and classical L-system notation.

5.2 Simple L-systems

The following is an example of a simple L-system. V is the set of non-terminal symbols, T is the set of terminal symbols, P is the set of productions.

[2]This is not strictly true in environmentally sensitive or open L-systems as the environment may effect productions, but the main point is that the *results* of productions do not influence each other

$$V = \{A, B\},\ T = \{\Omega\}$$

$$P = \{(A \rightarrow AA), (B \rightarrow BB)\}$$

If we take $\omega_0 = (A\Omega B\Omega)$ as our start string, then the following sequence occurs:

$$\begin{array}{ll}
\omega_0 & (A\Omega B\Omega) \\
\omega_1 & (AA\Omega BB\Omega) \\
\omega_2 & (AAAA\Omega BBBB\Omega) \\
\omega_3 & (AAAAAAAA\Omega BBBBBBBB\Omega) \\
\omega_4 & \ldots
\end{array}$$

Note that these substitutions occur in parallel. This is a simple example of exponential parallel growth using an L-system.

5.3 Stochastic L-systems

In the previous example the results are always the same for a given axiom. This is because there is exactly one production for each non-terminal symbol. When more than one production is specified for the same non-terminal it is necessary to have some means of deciding which production to use. One way of deciding between productions is to use a *stochastic* L-system. In a stochastic L-system a *weight* or probability is assigned to each production such that $0 \leq p \leq 1$, and that the sum of the weights for a given non-terminal equals 1.

The syntax used in this paper differs somewhat from that used in other treatments of stochastic L-systems. For the sake of simplicity the weight of a production will be placed above the arrow symbol in the production string, *e.g.* $(A \overset{1/2}{\rightarrow} B)$ indicates a weight of $1/2$ for the production of B from A. Where no weight is specified in a production it is assumed to be 1.

The following example is a linear stochastic growth model, in which the string either grows linearly or terminates.

$$V = \{g, S\},\ T = \{\Lambda\}$$

$$P = \{(g \overset{1/2}{\rightarrow} Sg), (g \overset{1/2}{\rightarrow} \Lambda)\}$$

If we take $\omega_0 = (g)$ as our start string, then the following sequences are possible:

$$\begin{array}{ll} \omega_1 & (Sg) \\ \omega_2 & (SSg) \\ \omega_3 & (SSSg) \\ \omega_4 & (SSS\Lambda) \end{array}$$

or

$$\begin{array}{ll} \omega_1 & (Sg) \\ \omega_2 & (S\Lambda) \end{array}$$

or

$$\begin{array}{ll} \omega_1 & (\Lambda) \end{array}$$

and an infinite number of other combinations which may be summarized as:

$$\omega_n \in \{(S^n g),\ (S^i \Lambda) \mid 0 \leq i < n\}$$

This example is more interesting from the point of view of neuron modeling because it suggests that a population of strings (or models) may be generated from a single stochastic grammar. Further, by adjusting the weights the statistical properties of the population may be changed. However, there is no way to deterministically control the final length of the string[3]. Deterministic control over the choice among multiple productions for a given non-terminal symbol may be achieved by using a parametric L-system.

5.4 Parametric L-systems

L-systems may be parametric, meaning that the choice of one production over another for a given non-terminal may depend on parameters associated with that symbol. Parameters are enclosed in parentheses immediately following the symbol they are associated with. In the syntax of this paper a conditional production is indicated by a vertical bar $|$ separating the production string and a boolean expression governing the production.

As an example of a simple parametric L-system, the previous grammar could be changed by adding a length parameter to g. Two conditional productions handle the cases when the parameter is zero, or greater than zero. This results in a simple deterministic linear growth system in which

[3]It is always possible to create deterministic results by specifying weights of 1 or 0, or by composing a start string of terminal symbols. This is not particularly useful because the first two cases are equivalent to a non-stochastic system and in the last case no rewriting takes place.

the final string length is easily controllable by specifying the value of the parameter for g in the start string.

Note that this example is not stochastic because the choice of one production over another depends only on a parameter.

$$P = \{(g(0) \rightarrow \Lambda), (g(i) \rightarrow Sg(i-1) \mid i \in N,\ i \neq 0)\}$$

Taking $\omega_0 = (g(3))$ as our start string results in the following productions:

$$\begin{array}{ll} \omega_0 & (g(3)) \\ \omega_1 & (Sg(2)) \\ \omega_2 & (SSg(1)) \\ \omega_3 & (SSSg(0)) \\ \omega_4 & (SSS\Lambda) \end{array}$$

Thus, the length of the resulting string will always be one plus the value of the parameter associated with g in the start string.

This type of parametric control is important in modeling neurons because it allows properties decided at one level of the grammar to be passed down to lower levels. For example, the length of a dendritic segment may be decided before the segment itself begins to develop.

5.5 Open L-systems

An open L-system is a type of L-system allowing bilateral communication between an L-system and its environment. Open L-systems are an extension of the idea of environmentally sensitive L-systems, which provide one-way, environment-to-L-system communication.

In an environmentally sensitive L-system, information is queried from the environment after each iteration using a query symbol $?X$, where X designates the type of information to be queried. The results of the query fill in the parameters of the query symbol, which may then be used in subsequent productions. Typically the information requested is the position or orientation of the turtle (Section 7.2).

Open L-systems extend this idea by providing a two-way communication mechanism. This is done through a special query symbol, $?E$, which acts as a “mailbox” for communications between the L-system and the environment. Parameters to $?E$ set by the L-system are passed to the environment, which interprets them and replaces the original parameters with a response message. The format of the messages is defined elsewhere in a special message format file.

Open (or at least environmentally sensitive) L-systems are needed to model neuron populations because the environment in which neurons grow is a dense space-filling medium through which dendritic and axonal segments must "feel" their way. As the segments grow, they themselves fill the space in the matrix and change the environment. Further, boundary-conforming growth is exhibited by neurons composing anatomical structures such as pyramidal cell modules in the striate cortex.

5.6 Notation Summary

In the notation used in this paper[4], an L-system is defined as a set of non-terminal symbols, V, a set of terminal symbols T, and a set of productions, P.

The notation used for productions can be summarized as follows:

$$id: \quad pred \ \overset{prob}{\longrightarrow} \ succ \mid cond \tag{1}$$

where *id* is a label, *pred* is the *strict predecessor* (the symbol for which the production is being specified), *cond* is a conditional governing the production, *succ* is the successor string (the string that is substituted for the predecessor in the resulting string), and *prob* is a probability governing a stochastic production. Productions are also written in string notation without an explicit label:

$$(pred \ \overset{prob}{\longrightarrow} \ succ \mid cond) \tag{2}$$

[4]The classical L-system notation is somewhat different. The set of symbols is not explicitly stated, and the notation for productions differs as well:

$$id : lc \ < \ pred \ > \ rc : cond \ \rightarrow \ succ : prob$$

The *lc* and *rc* symbols, which in the classical notation allow for context sensitive L-systems, are a construct not used in this paper.

Symbol	Description	Function
N	Neuron	Neuron generation start symbol.
O	Soma	Starts soma generation
A	Axonal arbor	Starts generation of axonal arbor.
D	Dendritic arbor	Starts generation of dendritic arbor.
J	Junction	Produces a junction marker and daughter segments.
S	Segment	Terminates or produces a growth cone.
g	Growth cone	Produces junction or a micro-segment plus growth cone.
u	Micro-segment	Produces a rotation plus a translation plus geometry.

Table 1: N++ non-terminal symbols

6 N++ Grammar

The intent of this section is to acquaint the reader with the organization of the N++ language. This section describes the overall structure of the N++ language and presents a simplified L-system grammar.

The N++ language is a stochastic open L-system. The grammar presented here is simplified for the sake of clarity. The full grammar is fundamentally the same but its syntax is more complex to allow (1) arbors of more than one type (*e.g.* basal and apical dendritic arbors in pyramidal cells), and (2) to allow neuron populations to interact with their environment.

6.1 Symbols

The N++ alphabet may be summarized as:

$$V = \{N, O, A, D, J, S, g, u\},\ T = \{[,], r, t, G, J', J'', H, u', s, r, t, \lambda\} \quad (3)$$

A brief description of the N++ language non-terminal symbols is to be found in Table 1. From the point of view of the grammar, non-terminal symbols are best described in terms of their productions (Section 6.3). Table 2 is a summary of the terminal N++ symbols, and is provided mostly for reference. Section 7 describes the function of the turtle-geometry related symbols ([,], r, and t). Section 8 discusses the interpretation of the other symbols as they relate to neuron morphology.

Symbol	Description	Function
[	Push	Pushes the state of the turtle onto the stack.
]	Pop	Pops the state of the turtle.
r	Rotation	Turtle rotation symbol
t	Translation	Turtle translation symbol
G	Soma geometry	Represents soma geometry.
J'	Junction marker	Marks a junction terminating a parent segment.
J''	Soma-arbor junction marker	Marks a junction of soma and arbor.
S'	Segment header	Marks initiation of a daughter segment.
u'	Micro-sphere	Represents micro-segment geometry.
s	Spine	Represents spine geometry.
λ	Termination	Growth termination symbol

Table 2: N++ terminal symbols

Label	Production		
p_N :	N	$\longrightarrow$	$[O[A]^+[D]^+]$
p_O :	O	$\longrightarrow$	$rtrG$
p_A :	A	$\longrightarrow$	$J''[rS]$
p_D :	D	$\longrightarrow$	$J''[rS]$
$p_{J_{00}}$:	J	$\xrightarrow{f_{J'_{00}}}$	$J'[rS][rS]$
$p_{J_{01}}$:		$\xrightarrow{f_{J'_{01}}}$	$J'[r[rS]S]$
$p_{J_{000}}$:		$\xrightarrow{f_{J'_{000}}}$	$J'[rS][rS][rS]$
$\vdots$	(See Equations 19, 20, and 21.)		
p_{S^+} :	S	$\xrightarrow{f^+_{non-term}}$	$S'g$
p_{S^-} :		$\xrightarrow{f^-_{non-term}}$	λ
p_{g^+} :	g	$\xrightarrow{f^+_J}$	J
p_{g^-} :		$\xrightarrow{f^-_J}$	ug
p_{u^+} :	u	$\xrightarrow{f^+_{spine}}$	$rtu's$
p_{u^-} :		$\xrightarrow{f^-_{spine}}$	rtu'

Table 3: Simplified N++grammar

6.2 Grammar

As previously stated the full N++ grammar is an open, stochastic L-system. For the sake of simplicity the grammar is presented here as a stochastic L-system in which production weights are represented symbolically. Parametric expressions and conditionals are not used. The notation f^+ and f^- indicate that $f^+ + f^- = 1$. It is assumed that the sum of symbolic weights for all productions associated with a given non-terminal equals one. The special symbol $+$ is used in the classic regular expression sense to denote one or more repetitions of a symbolic unit.

In the full N++ grammar the symbolic production weights of the simplified grammar are replaced by a more complicated system allowing growing neurons to interact with both the physical environment and a user-defined distribution function environment. In effect, each non-terminal symbol X of the simplified grammar is equivalent to $?E(X\ldots)$, where the ellipses represent symbolic parameters passed to the two environments. Likewise, each stochastic production of the simplified grammar becomes a conditional production which is a function of the information passed back.

6.3 Productions

The set of productions for the simplified N++ language grammar may be summarized as:

$$P = \{p_N, p_O, p_A, p_D, p_{J_{00}}, p_{J_{01}}, p_{J_{000}}, p_{S^+}, p_{S^-}, p_{g^+}, p_{g^-}, p_{u^+}, p_{u^-}\} \qquad (4)$$

The individual productions are described in Table 3.

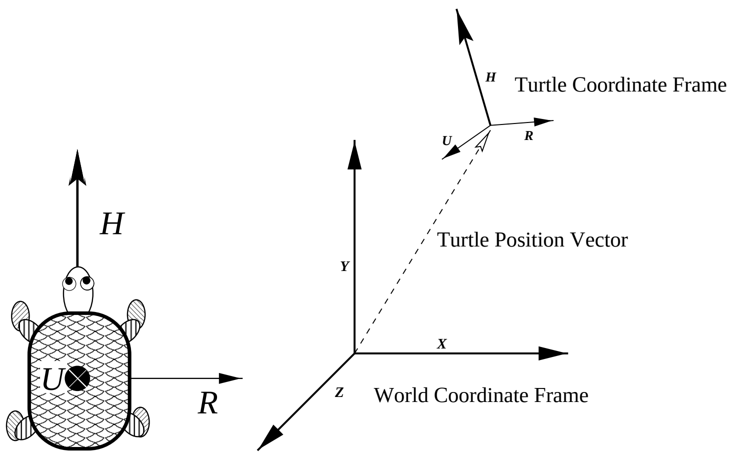

Figure 1: The turtle specifies a position and orientation in 3-space.

7 L-system Geometric Modeling

7.1 Overview

L-systems are useful in graphics because of their applications in procedural modeling. Symbols in an L-system string may be assigned a graphical or geometric interpretation, allowing an L-system grammar to generate two-dimensional images or three-dimensional models. This interpretation is performed in a step separate from the parallel rewriting grammar.

Traditionally, L-systems have been combined with a geometric construct called turtle geometry. This allows a simple geometric interpretation to be assigned to symbols in an L-system alphabet. By interpreting an L-system string symbol-by-symbol, a geometric model may be constructed. This string interpretation step is sequential and completely separate from the parallel operation of the grammar.

7.2 Turtle Geometry

The "turtle" in turtle geometry is a position in space combined with an orientation. (See Figure 1) The turtle may be rotated to produce a new orientation or translated along its primary H axis to a new position. An analogy may be made between the turtle and a vehicle which can either rotate about its center or move forwards, but not both simultaneously.

7.3 Turtle Movement Commands

In the syntax of the N++ there are four turtle geometry movement commands: *rotate*, *translate*, *push*, and *pop*. These are all terminal symbols. Rotate and translate are parametric, whereas push and pop are not.

rotate — $r(\ldots)$ Rotate the turtle relative to the turtle's current coordinate frame. The actual rotation is defined by the parameters to r and may be specified in a number of ways.

translate — $t(d)$ Translate d units in the direction of the turtle's primary axis, as defined by the H vector (See Figure 1.)

push — `[` Push the current orientation and position of the turtle (the turtle's state) on to the stack[5]. This does not change the state of the turtle. See *pop*.

pop — `]` Pop replaces the turtle's current state with the state on the top of the stack. The top state is then removed from (popped off) the stack. Push and pop work together to provide a simple mechanism to save and restore the state of the turtle.

In some applications of turtle geometry the turtle is analogous to a "pen" which draws as it moves. In N++, neurons are modeled as space-filing solid models. The turtle acts as a cursor which positions and orients geometric primitives in space. Any command which creates geometry does so with the turtle's current position and orientation.

7.3.1 Turtle Movement and Parallel Growth

The reader may observe that the construction of branching figures using L-system turtle geometry conventions is a *depth-first* process, meaning that a single branch is traced from its origin to its end before the turtle backs up to trace other branches. At first glance this may appear contradictory to the principle of parallel growth. However, the apparent contradiction is resolved when one observes that parallel growth is a property of the parallel rewriting grammar, and depth-first geometric construction is a property of the subsequent sequential interpretation and visualization of the string.

[5]A stack is a list of items which is accessed in LIFO, or last-in-first-out, order. An analogy may be made to a stack of books, in which only the book on top is accessible.

8 Representation of Neuron Components

8.1 Introduction

In this section the N++ representation of neuron morphology is explained on an element-by-element basis. This representation is based in large measure on the neuron morphology modeling work of Kishore Mulchandani[4]. Rather than present a complete morphology representation and dissect it into components, a top-down approach is utilized in which progressively more detail is added to a simple representation. The reader is encouraged to refer back to Section 6 to see how the representations discussed here fit into the N++ grammar, and to Section 7 for clarification of the geometric interpretation of N++ strings.

8.2 A Schematic Arbor

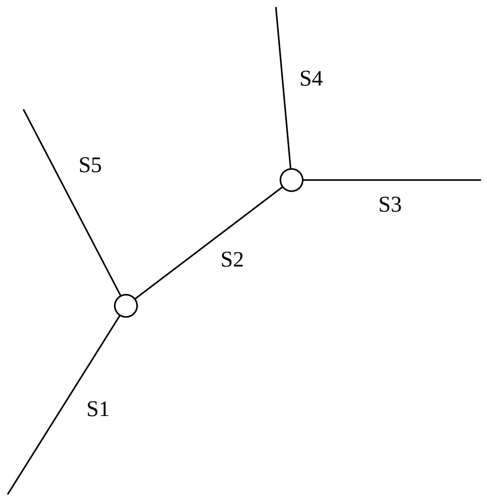

Figure 2: A schematic representation of dendrite branching

To begin the discussion of neuron morphology, we will start with the schematic dendritic arbor presented in Figure 2. While this figure is very simple it does exhibit the fundamental branching structure characteristic of neuron morphology. Figure 2 may be represented in a bracketed string notation as follows:

$$[S_1[S_2[S_3]S_4]S_5] \tag{5}$$

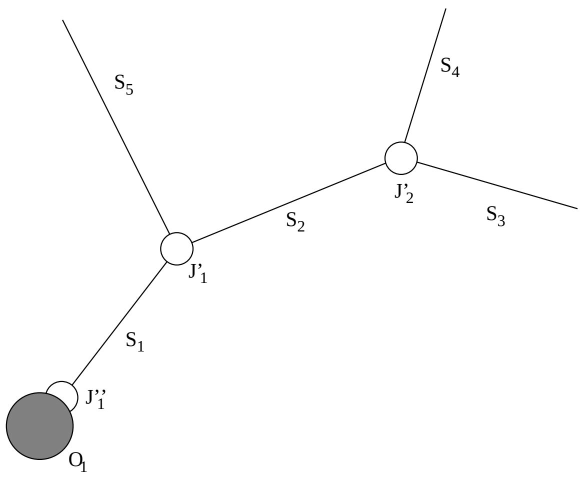

Figure 3: A schematic neuron

Everything has been eliminated from the representation but dendritic segments S_i $(1 \leq i \leq 5)$, [and]. Reading from left-to-right, this string is interpreted as:

> Save (push) the state of the turtle, draw segment one, push the state again, draw segment two, push the state, draw segment three, replace the turtle's current state with the one on the top of the stack and remove that state from the stack (pop the state), draw segment four, pop the state again, draw segment five, pop the state.[6]

Equation 5 shows how a simple branching figure may be represented using the L-system turtle geometry conventions, particularly the operation of the "push" and "pop" commands.

8.3 A Schematic Neuron

Figure 3 is a schematic representation of a neuron with a single dendritic arbor and no axon, and is the next step in elaboration on the basic branching

[6]The reader may note that this sequence begins with a push and ends with a pop, thus restoring the original state of the turtle. While not necessary in this example, strings representing segments, arbors, neurons, and larger-scale neural structures should always be bracketed in this way, allowing elements to be cleanly positioned and oriented.

structure represented in Figure 2. This cell could be described using an abbreviated N++ representation as follows:

$$[O_1 J''_1 [S_1 J'_1 [S_2 J'_2 [S_3] S_4] S_5]] \tag{6}$$

In this equation O_1 represents the soma, J''_1 represents the soma-arbor junction, The S_i represent segments and The J'_i represent junctions. Note that in this simplified representation there are no explicit references to rotations, translations, or geometric primitives, and the two non-terminal symbols O and S are used as though they were terminal. While this representation is deliberately incomplete, Equation 6 will form the basis for further elaboration.

8.4 Representation of the Soma

The first symbol in Equation 6 is O_1, representing the soma. The soma is the "root" of the dendritic and axonal arbors of a neuron, and may have a variety of different shapes. A soma has three basic properties: a position in space, an orientation, and a geometric form.

8.4.1 Soma positioning

The first step in describing a soma is to position it in space. Using turtle geometry conventions, this involves a rotation followed by a translation. The next step is to orient the soma, which is a rotation. This process is represented as a production in Equation 7.

$$O \rightarrow rtr \tag{7}$$

8.4.2 Soma geometry

Equation 7 is not complete. Once the Soma is positioned and oriented its geometry must be created to represent the soma as a three-dimensional solid model. In N++ this is done either by approximation with simple geometric primitives (*e.g.*, sphere, cone) or through a more complex reconstruction based on sampled contours. In either case the symbol G will be used to represent this geometry, with both the type and sampled data (if any) being associated with G as parameters.

$$O \rightarrow rtrG \tag{8}$$

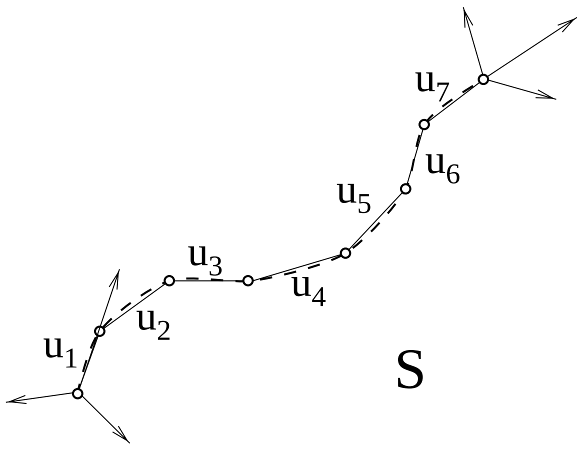

Figure 4: A micro-segment approximation for segment trajectory.

This results in Equation 8, which the reader may observe is equivalent to production p_O in Table 3.

8.5 Representation of the Arbor

The remainder of Equation 6 (everything but the first symbol) is a simplified arbor. In N++ an arbor (either axonal or dendritic) begins with a soma-arbor junction marker J''. This is a marker symbol used in the statistical analysis phase and has no geometric interpretation. Following the soma-arbor junction marker is a series of segments intersecting at junctions.

8.6 Representation of the Segment

8.6.1 Segment trajectory

An axonal or dendritic segment has two primary properties: a trajectory in space and a diameter that may vary along the length of the segment. In Equation 6 a neuron segment is represented with a single symbol S. The first step in refining this representation is to add more trajectory information.

The trajectory of a real neuron segment is a continuous path in space. This may be approximated by a series of straight trajectory micro-segments, as is illustrated in Figure 4. The symbol u is used to represent such a micro-segment. Thus, for a trajectory approximation with n micro-segments:

$$S \rightarrow u_1 u_2 \ldots u_n \tag{9}$$

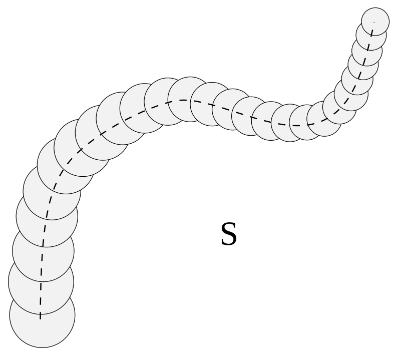

Figure 5: A series of spheres approximating segment geometry.

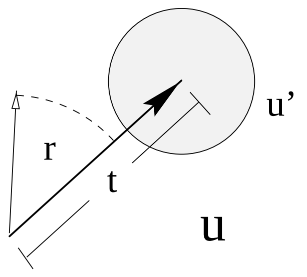

Figure 6: Micro-segment details.

As a trajectory approximation, each micro-segment is equivalent to a rotation followed by a translation. Thus:

$$u \to rt \tag{10}$$

The position resulting from the interpretation of a micro-segment may be thought of as a either a sampled data point or a knot point in a piecewise interpolating spline approximating the real trajectory of the neuron segment. With sufficient subdivision the use of micro-segments provides a good trajectory approximation.

8.6.2 Segment Geometry

In N++ a series of small spheres, or micro-spheres, is used to approximate the smooth tapered cylindrical geometry of the segment, as illustrated in Figure 5. The symbol u' represents a micro-sphere. This redefines the micro-segment as a displacement followed by a micro-sphere (See Figure 6):

$$u \rightarrow rtu' \tag{11}$$

8.6.3 Spines

One other feature of segments is visible at optical resolution. Small protrusions, called spines, may be seen stochastically distributed along the length of the segment. Where they exist spines are dealt with as a further elaboration of the micro-segment. A spine is represented by the symbol s.

$$u \rightarrow rtu's \tag{12}$$

Note that except for stochastic weights, Equation 11 and Equation 12 are equivalent to productions p_{u^-} and p_{u^+} of Table 3.

As a matter of practice, segments are represented as generalized cylinders, with spines added by applying a displacement map to the smooth surface of the cylinder.

8.7 Representation of the Junction

A junction is the intersection of a single segment and a soma (in the case of a soma-arbor junction) or the branching point where a single segment bifurcates or multifurcates. The first type of junction has been discussed in Section 8.5. The second type of junction, a bifurcating or multifurcating junction, is discussed here.

8.7.1 Bifurcation

Bifurcation is the process of a single parent segment dividing into two new daughter segments. This process may be represented as:

$$J \rightarrow [S_1][S_2] \tag{13}$$

To aid subsequent statistical analysis, a junction marker symbol J' is introduced. Like J'', this symbol has no geometric interpretation. Thus Equation 13 becomes:

$$J \rightarrow J'[S_1][S_2] \tag{14}$$

Because the two segments do not start with the same orientation, an initial junction rotation is needed to orient them. This is handled by introducing a junction rotation symbol H:

$$J \rightarrow J'[H_1S_1][H_2S_2] \tag{15}$$

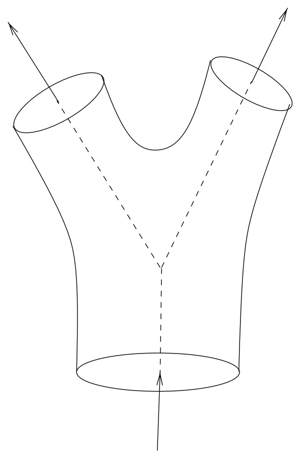

Figure 7: A bifurcation junction.

In Equation 15 the initial orientation of both segments depends only on the orientation of the junction, which is to say on the final orientation of the parent segment. It is also possible that the initial orientation of S_2 could depend on S_1 as well. Thus, there are two possibilities for a bifurcation:

$$J \xrightarrow{f_{J'_{00}}} J'[H_1 S_1][H_2 S_2] \tag{16}$$

$$J \xrightarrow{f_{J'_{01}}} J'[H_1[H_2 S_2]S_1] \tag{17}$$

In Equation 16 the initial orientation of the two segments are independent, and in Equation 17 the initial orientation of S_2 depends on H_1 as well as H_2. These two equations are equivalent to productions $p_{J_{00}}$ and $p_{J_{01}}$ of Table 3.

8.7.2 Multifurcation

The process of multifurcation is similar to bifurcation, except that a single segment divides into more than two segments. In N++ , it is assumed that all multifurcation are trifurcations. The syntax for multifurcation is the same as for bifurcation except that there are more possibilities for the interdependence of the initial orientation of segments. For the sake of simplicity Table 3 lists only the simple dependency case as production $p_{J_{000}}$. The full list of multifurcation dependency possibilities is as follows:

$$J \xrightarrow{f_{J'_{000}}} J'[HS][HS][HS] \tag{18}$$

$$J \xrightarrow{f_{J'_{010}}} J'[H[HS]S][HS] \tag{19}$$

$$J \xrightarrow{f_{J'_{011}}} J'[H[HS][HS]S] \tag{20}$$

$$J \xrightarrow{f_{J'_{012}}} J'[H[H[HS]S]S] \tag{21}$$

9 Summary

The N++ language is an open, stochastic L-system developed for the stochastic modeling of neuron morphology. The N++ language is both a parallel-rewriting grammar and a string-based model representation language. The N++ language grammar is designed to mirror the growth and development of neurons in the matrix of neural tissue. As a string-based model description language, the N++ language is designed to accurately represent the morphology of neurons as viewed at the limit of optical resolution.

In this paper a general introduction to L-systems, both as grammar and as modeling tool, is presented. The purpose of the N++ language and how it fits into the N++ system is discussed. A simplified N++ language grammar is presented along with a detailed description of how this grammar works to produce models of developing or mature neurons.

The next paper in this series, *The Statistical Model*, is an overview of the probability density functions governing the stochastic modeling of neuron populations.

10 Acknowledgments

The authors would like to thank Kishore Mulchandani for laying the groundwork for this project. The N++ language described here owes much of its structure to his work on neuron morphology modeling. The authors would also like to thank Sandeep Tewari for his contributions to modeling neuron development in space-filling structures, including the use of implicit function modeling for neuron morphology.

11 Literature Cited

References

[1] A. Lindenmayer. Mathematical models for cellular interaction in development, parts i and ii. *Journal of Theoretical Biology*, 18:280–315, 1968.

[2] A. Lindenmayer. Developmental systems without cellular interaction, their languages and grammars. *Journal of Theoretical Biology*, 30:455–484, 1971.

[3] R. Mech and P. Prusinkiewicz. Visual models of plants interacting with their environment. In *Computer Graphics Proceedings, Annual Conference Series, 1996*, pages 397–410, August 1996.

[4] K. Mulchandani. Morphological modeling of neurons. Master's thesis, Texas A&M University, College Station, TX, 1996.

[5] P. Prusinkiewicz, M. James, and R. Mech. Synthetic topiary. In *Computer Graphics Proceedings, Annual Conference Series, 1995*, pages 351–358, August 1995.

[6] P. Prusinkiewicz and A. Lindenmayer. *The Algorithmic Beauty of Plants.* Springer-Verlag, New York, 1990.