# Structural Tree Language (STL): L-System Plant Implementation with Structural Syntax

Yao-ming Yeh,
Chia-lin Chien,
Kuan-Sheng Lee
Dept. of Information And Computer Education,
National Taiwan Normal University, Taiwan.

## *Abstract*

**This paper presents a new method for modeling plant forms. Built on L-systems, Structural Tree Language, or STL, is introduced to describe tree shapes in language-based graphics model. Besides traditional L-system primitives (namely Drawing Components in STL), we propose a novel Structural Syntax to enhance the ability of STL, which can be used to describe the outline shapes as well as the exact branching behaviors of trees. In STL, we provide two methods to describe a tree, namely Outline Description and Branch Expansion Description. Outline Description can represent the branch area of a tree in a higher-level mechanism, which specifies only simple rules and shape definitions for trees; whereas Branch Expansion Description provides a lower-lever mechanism, which describes the detailed branch behaviors of trees. We also developed two modules for the user to build his or her trees interactively using these two methods respectively.**

***Key Words: Computer Graphics, Language-based Graphics Model, L-System***

## I. Introduction

It is always an ambition of computer scientists to simulate the appearances of or even the interaction among real-world objects and scenes using graphic models. Among those models, language-based graphics model such as L-system is often discussed when considering self-resemblance objects and scenes in the real world, especially plants. L-system was originally proposed by Lindenmayer to simulate the development of multicelluar organisms [1]. While it was originally proposed for biological purposes, computer graphics researchers have found that L-system can be used to render attractive and sophisticated graphics objects based upon L-system formalisms. Under these formalisms, specific symbols such as F, L, and R are defined as drawing primitives, and strings comprised by those symbols are built as production rules to represent the appearances of plants.

Beyond the basic idea, many additional parameters are introduced to enhance the modeling ability of the system. Chen [3] demonstrates an effortless transformation from the 2D L-system to a 3D one. Prusinkiewicz [4] introduces the environmental parameters into L-system. The Wildwood project presented by Mock [5] can generate a new plant by genetic algorithm. Chen and Jing [6] use context-sensitive grammar, named $L^+$-System, instead of conventional context-free grammar to simulate plant hybridization using string substitutions. Also, in order to overcome the deterministic characteristics of context-free grammars, Samal, Peterson, and Holliday [7] devise the Stochastic L-system in which probability calculations were added to simulate the diversity for plants of the same kind.

### I.1 Turtle Graphic System

The rendering of plants for L-system can be implemented by a Turtle graphic system [2]. In Turtle graphic system, a turtle is rambling about on the canvas with a pen fixed under its belly. The trail of the turtle would then be recorded on the canvas. Some text symbols are defined as commands to mandate the turtle's behavior. The followings are typical command symbols and their respective meanings:

F: Move the turtle forward (along its original direction), and leave a line segment.

R (or -): Turn the turtle right by a predefined degree.

L (or +): Turn the turtle left by a predefined degree.

[: Save the status of the turtle (i.e., push its position and direction onto stack).

]: Restore the status of the turtle previously saved (i.e., pop from stack).

Assuming the starting direction of the turtle is the direction of positive X axis, with the above definitions and a predefined turning degree as 60 degree, the string FLFLFLFLFLFLF bears the following graph (Figure 1):

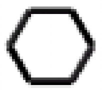


*Figure 1. Graph of string FLFLFLFLFLFLF in Turtle Graphics*

According to the concept of Fractal, we can define a starting symbol F for the Turtle Graphics, and the above string thus becomes a production rule:
F→FLFLFLFLFLFLF.

Figure 1 becomes the graph of the rule string

F→FLFRRFLF with a seed (starting symbol) F when the system applies the rule substitution once.

As another example, consider the production rule: F→FLFRRFLF with a seed F, we will obtain Figure 2 when the system applies the substitution once. If we apply the production rule 3 times, we will derive Figure 3, which is the famous Koch curve [1].

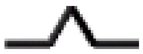
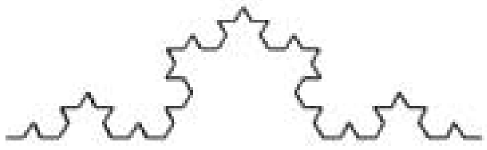


*Figure 2. Koch curve*

*Figure 3. Koch curve with triple substitution of the*



*production rule*

## I.2 L-System Basics

Although the appearances of plants in the real world seem to be arbitrary and unpredictable, with fine inspection, however, we can find that most plants possess the characteristics of self-resemblance. Take a tree as an example. Every single branch of a tree is just the miniature of the whole tree. Therefore, we can define some simple but adequate strings and string substitution rules, called production rules, to model the whole tree. This is the major concept of the L-system.

The shape of a graph generated by a Turtle-graphics system depends on the seed and the production rules provided by the user. In the L-system, often, for simplicity, the seed is a single symbol representing the command "forward" or is a single symbol without any meaning except for replacement purposes. We may carefully select suitable production rules to make our figures resemble real-world plants. And, of course, the starting orientation is upward. For example, consider the seed F and the rule F→F[RF]F[LF]F with the rotation angle predefined to be 30 degree. When applying the substitution once, we'll obtain the tree in the left of Figure 4. It resembles a portion of a plant.

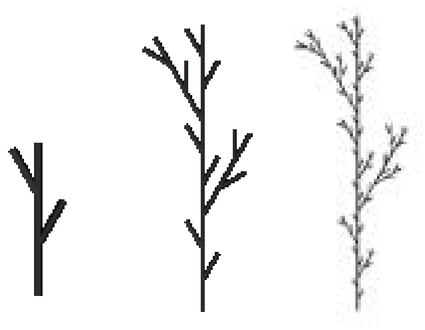


*Figure 4. L-system plant with rule string substitution 1 times (left), 2 times (middle) and 3 times (right)*

With closer examination, one will find that the plant with two-time substitution actually resembles the plant with one substitution as a whole, and it looks like being assembled by the one-substitution plant piece by piece. We call this self-resemblance. Thus, the one-substitution plant could be considered as the growth rule of the plant in the real world. Figure 4 also presents a plant using this growth rule, which 3-time substitutions.

Based on the above discussion, there are four fundamental components in an L-system:

1. A set of instruction symbols: Each symbol represents a command for the plant growth.
2. Starting symbol or seed: This is where the substitution begins. Often, the seed is a single instruction symbol that represents the meaning of "drawing a line forward".
3. Production rule: One or more text strings comprised of command symbols for symbol substitution.
4. Substitution counts or iteration counts.

Formally, a plant P modeled by the L-system is described as a 3-tuple P={S, P, I}, where S is the seed, P is the set of production rules, and I stands for the substitution count. The following abbreviations are used for convenience: Seed(P), and Product(P), are used here to represent the seed and the production rule set of plant T.

## I.3 Growth Models

Prusinkiewicz [4] creates virtual plants by simulating the growth of plants with L-systems. By defining complicated parameters and computations, the growth and therefore the appearances of stems and leaves of a plant would vary according to the factors of its surrounding environment, especially the intensity of light. Often, an axiom (seed) is parameterized, and several production rules (string substitution rules) are presented and chosen by predefined probability when substituting rule strings.

The following gives an example of the mechanism:

Axiom W:A(1)B(3)A(5)

Production p1: A(x) → A(x+1) : 0.4

Production p2: A(x) B(x-1) :0.6

Production p3: A(x) < B(y) > A(z) : y< 4→ b(x+z) [ a(y)]

The Axiom above can be expanded as: A(1)B(3)A(5) → A(2)B(6)[A(3)]B(4), in which p1 is applied in A(1), p2 is applied in A(5), and p3 is used in B(3).

With carefully designed parameters and productions, rather vivid pictures of plants can be generated through the growing model, but it also results in involving complex computations and adding difficulty in understanding the graphic language. Also, it is difficult and even laborious to find a good rule of computing parameters. A try-and-error method for a user to find good or even adequate rules is almost inevitable.

## II. Structural Tree Language (STL)

In Structural Tree Language (STL), we shall narrow our study from the whole range of plants to only the species of trees. There are two major parts in STL: Drawing Components and Structural Syntax, both of which are described in the following.

### II.1 Drawing Components in STL

The Drawing Components of STL are just those command symbols in a Turtle System. They are used here to specify the drawing actions of the Turtle to render the appearances of the trees. See the description of I.1 Turyle Graphic Systems for the definitions of command symbols in Drawing Components of STL.

### II.2 Structural Syntax in STL

Drawing Components described above represent common drawing primitives, therefore, in effect, they are adequate for all kinds of graphics. In contrast, "Structural Syntax" is developed specifically for tree modeling. In a sense, Structural Syntax describes "what" a tree looks like, rather than "how" to draw it. Therefore, we may term Drawing Components as "low-level" constructs and then Structural Syntax as "high- level" ones. Our Structural Syntax in STL includes Growing Rules (R), Self-Resemblance Count (I), Stem Parameters (B 、L), Branching Syntax, and Layout Description.

### II.2.1 Growth Rules ( R )

The Growth Rules deal with how a tree grows. Trees of the same species may have the same growth rules according their genes. Using the concept of macro, respective Growth Rules are grouped and are given respective names to enhance the readability.

For example, consider the seed of a tree T:

Seed(T) = [FFF][F[+F][-F]F][F[+F][-F]F]

And

Seed(T)' = R1R2R2

R1=[FFF]

R2=[F[+F][-F]F]

Seed(T) and Seed(T)' are syntactically equivalent. However, Seed(T)' are more readable since it specifies clearly how the seed of tree T branches. Therefore, STL would adopt the later style of description.

### II.2.2 Self-Resemblance Count ( I )

The self-resemblance property of a tree is modeled by recursively substituting the strings of production rules in L-system. In typical L-systems, an individual plant has a single rule and thus a single substitution count. This is not the case in STL, in which a tree may have multiple rules, each rule for each branch, and may have different substitution counts for each rules. The following syntax is used to describe the property:

[ I <count> <rule>],

Where <count> is a integer indicating the times of recursive substitution, and <rule> is a string of Drawing Components

### II.2.3 Stem Parameter (B, L)

In order to allow for more reality and flexibility, two parameters L and B are incorporated in STL. B defines the broadness and L defines the length of the stems. The syntax of the two parameters are listed below:

B = <default broadness> (<broadness decrement>)

L = <default length> (<length decrement>)

With the optional <broadness decrement> defined, the stems would become slimmer and slimmer while continuing branching. The latter the stem is materialized, the slimmer it is. The same case is applicable on the stem length described by L parameters.

### II.2.4 Branching Syntax

For a L-system tree, new stems grow from the end of all of its stems. The number of stems will grow at some exponential order. Obviously it is not the case in natural world [4], in which the number of branches is far much smaller. Prusinkiewicz and other researchers proposed many hypothetical mechanisms to simulate the branching constraint of real-world trees to avoid the exponential excess of stem numbers [13]. Here, we take a simplified approach to model the branching constraints. Our assumption is that a stem can have new branches on top of it only if there is enough free space for those new branches to occupy. Take Figure 5 as an example, the branch at the left side is failed because some existed stems already occupy the space it needs while the branch at the other side is succeeded because the space it needs is still free.

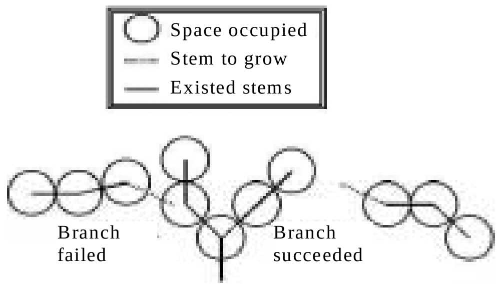


*Figure 5. A Example of Branching Constraints*

In STL, we define the following syntax to express the branching behaviors after those branching constraints are applied:

R {[R1], [R2], . . . , [Rm]}.

Note: anything in [ ] denotes it is optional.

The previous expression specifies that R has m branches. The nth symbol stands for the nth branch of R. If the nth symbol is Rn, the nth branch is succeeded and Rn is applied at the end of the branch; otherwise, the nth symbol is omitted representing that the nth branch is failed. For example, we have

R1 = [F][+F][-F]

R2 = [++F][-F]

The expression Rn = R1{R1, R2, R1} means the three branches of R1 are all succeeded, and after their successes, R1, R2, and R1 are applied to each of the extended branches from left to right. The first branch of Rn is [F], the second is [+F], and the last is [-F]. However, the above expression Rn is not complete, as we do not explicitly declare the branching rules for R1, R2, and R1 inside the {}. In the case that there is no further branching for those rules inside the {}, the expression should be rewritten as
Rn = {R1{, ,}, R2{,}, R1{, ,}}.

Substituting R1 and R2 with their right hand side in Rn, we'll obtain Rn in Drawing Components:

Rn = [F[F][+F][-F]] [+F[++F][-F]] [-F[F][+F][-F]].

Figure 6 demonstrates the visual representation of Rn. In this figure, The middle and right branches come from the first and the third branching rules of R1 (i.e., still R1), and the left branch comes from the second branching rules of R1 (i.e., R2).

As another example, The branching rules embraced in { } can each be expanded to include their respective branching rules such as

Rm = R1{R1{R2{,}, R2{,}, R2{,}}, R2{R2{,}, }, R1{R2{,}, R2{,}, R2{,}}},

Which, after expanded into Drawing Components, equals to

Rm = [+F [++F[++F][-F]][-F]] [-F[F[++F][-F]][+F[++F][-F]][-F[++F][-F]]]

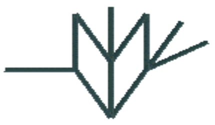


*Figure 6. Visual appearances of Rn applying branching rules*

As a summary, branching syntax provides a mechanism to describe the successive branches of all branches of a tree. When designing or appreciating a tree in STL using the branching syntax, the user can consider solely about how the branches would behave instead of fumbling about all those low-level Drawing Components. The branching rule expressions can be put inside Seed(T) or Product(T) for a given tree T.

### IV.1.5 Layout Description

Besides using the branching syntax to describe the branching behavior of a tree, STL provides "Layout Descriptions" to describe the growth (layout) range of a tree. Basically, Layout Descriptions are names of typical geometric shapes such as rectangles, ellipse, and polygons, or the combinations of those typical geometric shapes. In STL syntax, Layout Descriptions are defined as the following:

```
LAYOUT
    (<color>) <shape <shape parameters> >
    (<color>) <shape <shape parameters> >
...
ENDLAYOUT
```

The Layout Descriptions should always be enclosed by LAYOUT, ENDLAYOUT pairs. Each line defines a layout shape with its color, shape name (command) and shape parameters.

All shapes and their parameters usable in STL Layout Descriptions are listed below:

| Shape | Shape Parameters |
|---|---|
| Rectangle | Starting Point, Width, Height |
| Rounded Rectangle | Starting Point, Width, Height, Angle of Rounded Corner |
| Ellipse | Starting Point, Width, Height |
| Polygon | Point1, Point2, ..., Pointn |

*Table 1. Table of Layout Description Shapes and Parameters*

Shape color uses the color names in Delphi, such as clWhite, clRed... etc.

For example, the following defines the Layout shape in Figure 7.

```
LAYOUT
    (clWhite) Rectangle (0,0), 200, 100
ENDLAYOUT
```

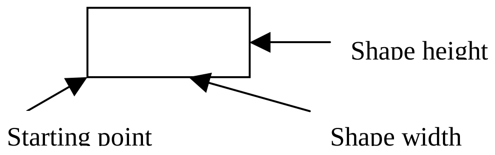


*Figure 7. Layout Description example*

### III. Outline Description (OD) vs. Branch-Expansion Description (BED)

Incorporating those Drawing Components and Structural Syntax, we propose two ways to describe a tree in STL, namely, Outline Description, and Branch-Expansion Description.

In Outline Description, a tree is described as the following:

```
<tree name>
<root coordinate>
<stem parameters>
RULE
<Production Rule 1>
<Production Rule 2>
. . .
<Production Rule n>
[ SEED
<Seed>
  ENDSEED ]
LAYOUT
<Layout Description 1>
<Layout Description 2>
. . .
<Layout Description n>
ENDLAYOUT
```

That is, in Outline Description, we only specify the seed and the unexpanded rules of the tree along with the area for the tree to grow. The seed segment may be omitted if the seed is only a single F, as it is common in most conventional L-systems.

In Branch-Expansion Description, the seed of a tree is expanded to incorporate the Growth Rules and other components in Structural Syntax to describe how each stem of the tree branches. A tree is described as the following:

```
<tree name>
<root coordinate>
<stem parameters>
RULE
<Production Rule 1>
<Production Rule 2>
. . .
<Production Rule n>
SEED
<Seed symbol string>
ENDSEED
```

The user describes her tree either by the Outline Description or by the Branch-Expansion Description. With the first description method, the user can specify some simple seed and production rules for recursion along with the area for the tree to grow. The responsibility to transform the original description to the Branch-Expansion from is left to the L-system program. Also, at times, the user may want to specify by herself all the branch rules for her tree. Under this circumstance, the second method is applied.

To give a concrete example, consider the following description for a tree named 2Dtree:

```
2DTree
(76,150)
L=10(0)
B=1(0)
RULE
    R1: [+F][-F]
LAYOUT
(clGreen)< RoundRect (0,0,180,192,7,7)>
(clWhite)< RoundRect (55,30,40,15,4,4)>
(clWhite)< RoundRect (35,125,76,30,5,5)>
ENDLAYOUT
```

The area specified by the Layout Description is shown in Figure 8 (left). The description above would then transformed into equivalent Branch-Expanded form listed below:

```
2DTree
(76,150)
L=10(0)
B=1(0)
RULE
R1: [+F][-F]
SEED
FR1{R1{R1{R1{R1{R1{,R1{,R1{,}}},R1{R1{,R
1{R1{,},R1{,R1{,}}}},R1{R1{,R1{,R1{,}}},R1{,}}}},
R1{R1{,R1{R1{,R1{,R1{,R1{,}}}},R1{R1{,R1{,R1{R
1{,R1{,}},}}},R1{R1{,R1{R1{,R1{R1{,R1{R1{,R1{,R
1{R1{,R1{,}},R1{,}}}},R1{R1{,R1{R1{,R1{,}},R1{,}
}},R1{,}}}},R1{R1{,R1{R1{,R1{R1{,R1{,}},R1{,}}},
R1{,}}},R1{,}}}},R1{R1{,R1{R1{,R1{R1{,R1{R1{,R1
{,}},R1{,}}},R1{,}}},R1{,}}},R1{,}}}},R1{R1{,R1{R
1{,R1{R1{,R1{R1{,R1{R1{,R1{,}},R1{R1{,R1{,R1{,}
}},R1{R1{,},}}}},R1{R1{,R1{,R1{R1{,},}}},R1{R1{,
R1{,R1{,}}},}}}},R1{R1{,},}}},R1{R1{,},R1{R1{,R1
{,}},}}}},R1{R1{,R1{,R1{R1{,R1{,}},}}},R1{R1{,R1
{R1{,},R1{,}}},R1{R1{,},}}}}}}}},}},},},R1{,R1{,R1
{,R1{R1{R1{R1{R1{R1{R1{,R1{R1{R1{R1{,},R1{,R
1{,}}},R1{R1{,R1{,}},R1{,}}},R1{R1{,R1{R1{,R1{R
1{R1{,R1{,R1{R1{,},}}},R1{R1{,R1{,}},}},}},}},}}},
R1{R1{,R1{R1{,R1{R1{,R1{R1{,R1{R1{,},R1{,R1{,}
}}},R1{R1{,R1{,R1{,}}},R1{,}}}},R1{R1{,R1{R1{,R
```

1{,R1{,}}},R1{,}}},R1{,}}}},R1{R1{,R1{R1{,R1{R1{
,R1{,}},R1{,}}},R1{,}}},R1{,}}}},R1{R1{,R1{R1{,R1
{R1{,R1{R1{,},}},R1{,}}},R1{,}}},R1{,}}}},R1{R1{,
R1{R1{,R1{R1{,},}},}},R1{,}}},R1{R1{,},}},R1{R1{,
},R1{,}}},R1{R1{,},R1{,}}},R1{R1{,},}}}}}}
ENDSEED

Figure 8 shows the visual appearance of the tree described above.

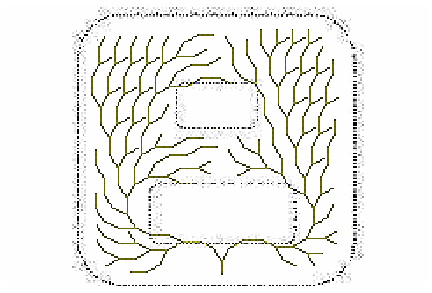


*Figure 8. Tree Outline and the Rendered Result*

### IV.2 Growth Strategies: Depth-First Growth vs. Breadth-First Growth:

In convention, a tree in a L-system gets its branches grown by 'Breadth-Firstly' substituting strings in its production rules. We name it 'Breadth-First Growth' as stems of the same level have their branches grown together. From the discussion of IV.1.4, it appears that the order of branching of stems can effectively alter the shape of the branches. In general, those stems that branch earlier would dominate the following growth. That is, earlier formed branches would occupy the space and block or confine the growth of latter branches. As a consequence, a tree applying a different growth strategy, i.e., Depth- First Growth, would have a greatly different shape from the original one. In Depth-First Growth, branch continues to grow until it cannot branch anymore because the boundary of the tree or the self-resemblance count of the particular rule has been reached. We discuss the two growth strategies here:

IV.2.1 Breadth-First Growth

Most L-systems utilize the strategy of Breadth-First Growth. Under this strategy, the growth of the whole tree may divide into many levels. After all recursive symbols at the same level are expanded with corresponding production rules, those at the next level are then expanded. That is, in a tree, all stems in level N are branched before stems in N+1 are branched, for all N greater than zero and not greater than user specified constraints. The order of branching in a single level is determined via the order of branching rules specified.

Take the tree T in Figure 9, in which Seed(T) = FR1 and R1 = [-F][+F], as an example. The branching rule for Level 1 is F, while the rule for level 2 and thereafter is R1. Those stems in level 2 are arranged according to R1. In a single stem when applying R1, The right-side branch has a higher priority to grow than the left-side branch, as in R1, a [-F], which means a right-side branching, is preceding a [+F], a left-side branching. As each level has its own branches grown before its next level does, all stems at the same level have almost equal probability to branch successfully. Therefore, the branches in Figure 9 seem equally distributed in left and right sides.

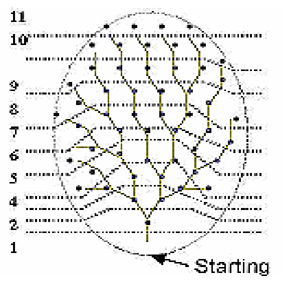


*Figure 9. Breadth-First Growth Example.*

### IV.2.2 Depth-First Growth

In Depth-First Growth, there is no the concept of levels. Each branch continues to branch and branch further until it cannot grow further anymore. When this happened, a next branch, which does not complete its growth, starts to grow until it cannot go any further. And again, a next branch starts to grow. Under this strategy, the branches with higher priorities will dominate the shape of the tree. Those branches with lower priorities struggle to survive because less free space is left for them. In other words, the lower priority a branch has, the less probability the branch will succeed in branching.

Figure 10 presents such an example in Depth-First Growth, in which the tree has the same seed and production rule with the tree in Figure 8. Using Depth-First Growth strategy, the tree has a complete different appearance from the previous tree. Observe that the tree has rather biased branches, and it looks some vine plants or floral decorations.

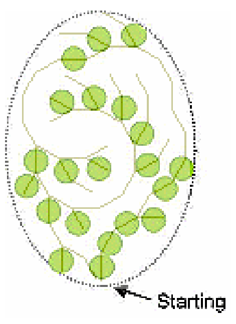


*Figure 10. Depth-First Growth Example.*

We summarize the comparisons of the two strategies in the table in Table 2 below.

| **Strategy** | **Breadth-First Growth** | **Depth-First Growth** |
|---|---|---|
| Description | All stems at the same level branches before those stems at | A single stem branches and branches further until some |

| | | |
|---|---|---|
| | the next level do. | constraints stop it. When this condition happened, one next unstopped branch start to grow. |
| Differences / Characteristics | 1. Stems in the same level shares almost the same probability to branch successfully.<br>2. Branches are balanced in each single level.<br>3. Branching Order in production rules has little impact on the shape of the tree. | 1. Stems in the same level have greatly varied probabilities to branch successfully.<br>2. Branches are biased in each single level.<br>3. Branching order in production rules dominates heavily on the shape of the tree. |
| Example Applicable Domains | Trees | Vines or floral ornament |

*Table 2. Comparisons of BFG and DFG.*

## V. System Description

Here is a concise description of the software -- Structural Tree Language Graphics System (abbreviated as STL Graphics System) -- we developed for implementing the proposed Structural Tree Language (STL).

There are two major components in STL Graphics System: Tree Maker is developed for the user to create tree graphics interactively. The user may use Outline Description (OD) to describe her tree and draw the shape for her tree to grow. Also, she may select the growth strategy, i.e., BFG or DFG, to apply on her tree. Tree Maker would then transform the tree in OD to the representations of Branch-Expansion Description (BED), render them onto the screen, and save those graphics in disks as either BMP format binary files or BED-formatted text files. The other is Tree Renderer. It is used mainly to read BED-formatted text files in and to re-render the trees represented in BED onto the screen. The user can also apply BED syntax directly to create her graphics in Tree Render. Tree Maker was designed using Borland Delphi 3, while Tree Renderer was created using Symantec Visual Café pro. The two both run upon the platform of Microsoft Windows system.

Figure 11 shows the block diagram of STL Graphics System components, in which the upper division represents the Tree Maker and the lower one stands for the Tree Renderer. The descriptions of those blocks are listed below:

### V.1 Tree Maker:

Main Frame: Main Program of Tree Maker. It initializes system parameters, maintains the interactions with the user, and invokes other components when necessary.

Language Analyzer: It translates tree descriptions written in Structural Syntax to descriptions formed solely by Drawing Components for 'Graphics L-system' to render the tree.

Graphics L-system: It is essentially a Turtle System, which takes descriptions formed by Drawing Components and renders the corresponding figure onto the canvas.

Graphics Tool: The user uses this component to define a shape for her tree to grow. The way to define a shape is like to draw a typical geometric object in most common painting software.

Graphics Analyzer: It is the preprocessor of the 'Graphics Developer'. Its main task is to analyze the shape drawn by the user in 'Graphics Tool'.

6. Graphics Developer: It transforms descriptions written in Outline Description (OD) format to corresponding descriptions in Branch-Expansion Description (BED) format based on the branching strategy specified.

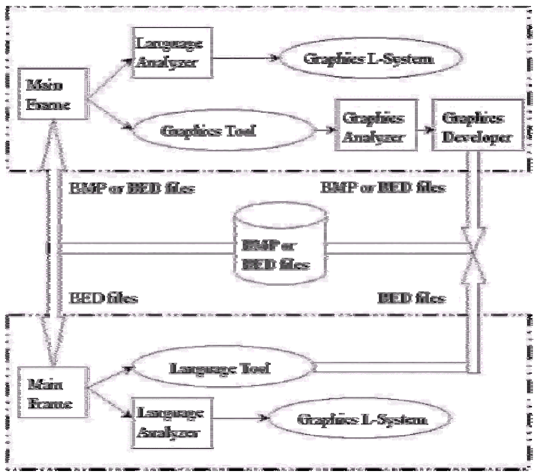


*Figure 11. System Organization Black Diagram of Structural Tree Language Graphics System (STL GRAPHICS SYSTEM)*

### V.2 Tree Renderer:

Main Frame: Main Program of Tree Renderer. It initializes system parameters, maintains the interactions with the user, and invokes other components when necessary.

Language Tool: It provides the interface for the user to add her own BED descriptions.

Language Analyzer: The component is the same as the identically named Component in Tree Maker.

Graphics L-system: The component is the same as the identically named Component in Tree Maker.

VI. Conclusion and Discussion

In this paper, we defined Structural Tree Language (STL), a language built upon L-systems to describe the appearances of trees. There are two major parts in STL, namely, Drawing Components, which are command symbols of drawing primitives in a typical Turtle system, and Structural Syntax, which is our proposed scheme to describe the spatial constraints and branching behaviors of a tree. Structural Syntax comprises of five parts: Growth Rules, Self-Resemblance Count, Stem Parameters, Branch Syntax, and Layout Description. Incorporating the above five parts, a tree may have a complex seed, several different production rules each has its own repetition count, and a confined range of space to grow. Also, a production rule may have multiple different rules applied in each of its branches.

Utilizing Drawing Components and Structural Syntax in STL, two ways to describe a tree are possible. Outline Description is adequate to define a tree by only specifying the production rules of a tree and a shape (outline) for it to grow, while Branch-Expansion Description is used to clearly specify all branching details of the tree.

We then developed a software system named STL Graphics System. In the system, the user may create her own tree interactively using Outline Description and, in a rare case, Branch-Expansion Description. The program would then transform statements in Outline Description into equivalent statements in Branch-Expansion Description. Statements in Branch-Expansion Description are thereafter used for rendering or for saving.

Structural Tree Language (STL) has the following advantages over the traditional L-systems.

Because of environmental and other effects, although trees have the attribute of self-resemblance, it is not natural that every branch has exactly the same appearances as other branches do in a single tree. With STL, the branches of each branch may be respectively described in detail, thus to simulate the outside effects and to confine the branch growth.

STL is more readable than traditional L-system in that the user may obtain the branching information of a tree from reading its Structural Syntax. Also, in the opposite sense, STL is more controllable because the user may use Structural Syntax to describe her desired branching in a tree.

As for the further work, to extend our STL to be a three-dimensional one should be straightforward. Also, as graphics described by a certain language possess a tremendous compression ratio against those stored as bitmapped files, incorporating STL into the web should result in saving lot of graphics transmission time and lot of bandwidth. It should be a good practice to re-define our STL in XML tags and to render our trees in Java applets.

## References:

1. Lindenmayer A., “Mathematical Models for Cellular Interaction in Development”, Parts II & I. J. Theoretical Biology 18, pp. 280~315, 1968.
2. Edward Angel, “Interactive Computer Graphics, A top-down approach with OpenGL”, Addison-Wesley , 1998
3. Hung-Wen Chen, “L-system plant geometry generator”, HTTP DOC, January 1995. 《 http://www.tc.cornell.edu/Visualization/contrib/cs490-94to95/hwchen/》
4. Radomir Mech and Przemyslaw Prusinkiewicz, “Visual models of plants interacting with their environment”, In Proceedings of SIGGRAPH’ 96, pp.397-410, 1996.
5. Kenrick J. Mock, “Wildwood: The Evolution of L-system Plants for Virtual Environments”, IEEE, pp. 476~480, 1998.
6. Chua Mei Chen and Hsu Wen Jing, "A Simulation Study of Plant Hybridization Using L+-System", IEEE, pp. 123~127.
7. Ashok Samal, Brian Peterson and David J. Holliday, "Recognizing Plants Using Stochastic L-Systems"
8. Mark Green and Hanqiu Sun, “A language and system for procedural modeling and motion”, IEEE Computer Graphics and Application, pp.52-64, Nov. 1988.
9. Stephen D. Casey, Nicholas F. Reingold, “Self-similar fractal sets: theory and procedure”, IEEE Computer Graphics and Application, Vol. 14, No. 3, pp.73-82, May 1994.
10. Alvy Ray Smith, “Plants, fractals and formal languages”, In Proceedings of SIGGRAPH’84, pp1-10, 1984.
11. Przemyslaw Prusinkiewicz and Aristid Lindenmayer, The Algorithmic Beauty of Plants, Springer-Verlag, New York, 1990.
12. De Reffy, P. C. Edelin, J. Francon, M. Jaeger, and C. Puech, “Plant models faithful to botanical structure and development”, In Proceedings of SIGGRAPH’88, pp.151-158, 1988.
13. Prusinkiewicz Prusinkiewicz, Aristid Lindenmayer, and James Hanan, “Developmental models of herbaceous plants for computer imagery purposed”, In Proceedings of SIGGRAPH‘88, pp.141-150, 1988.
14. Michael T. Wong, Douglas E. Zongker, and David H. Salesin,“Computer-Generated Floral Ornament”, In Proceedings of SIGGRAPH’98, pp.423-434, 1998.
15. John Kacher,“Interaction of multiple L-systems”, HTTP DOC, Presented at NCUR 98 in Salisbury, Maryland, 1998. 《 http://www.owlnet.rice.edu/~jkacher/lsys98.html 》
16. William McWorter, “Fractint L-Systems”, HTTP DOC, Version 1.4, January 1997. 《http://fractal.mta.ca/fractint/lsys/》