University of Alberta

3D Stochastic Tree Generation From Multiple Views

by

Qiongyan Fang 

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta
Spring 2004

Your file Votre référence
ISBN: 0-612-96472-8
Our file Notre référence
ISBN: 0-612-96472-8

Abstract

In this thesis, we examine how L-Systems can be extended as Hierarchical Hidden Markov Model L-Systems to enable the generation of 3D trees with more naturalistic variations yet with less tedious script generation demands.

The model also allows for L-System optimization and, in this case, it is tuned to fit within visual hulls, which are extracted from multiple images of trees, in order to enable the efficient and realistic generation of 3D tree models to be consistent with what is sensed by cameras. An automatic process is designed to create 3D trees from multiple images of a single tree.

Acknowledgements

I would like to thank Dr. Terry Caelli for his guidance and his enthusiasm for this research work at University of Alberta. I would also like to thank Li Cheng for his many helpful discussions on the optimization algorithms and his kind support for my graduate studies at the University of Alberta.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Tree simulation and modeling has important practical and theoretical applications. Biology, ecology, and agriculture are important application areas. It also plays a significant role in the entertainment industry, education, architecture where realistic three dimensional (3D) tree models are required. For example, it is used for studying plant growth processes and it is also applied in many visualization tasks, such as movies, games where landscapes are common scenes. Tree simulation and modeling is also an important research problem in computer graphics.

In most cases 3D trees have been created simply according to the tree species and their sizes with little constrains put on the overall shape of the trees. Such a 3D tree creating procedure is usually termed a *bottom-up* procedure or *forward simulation* [38]. In reality there are no two identical trees in the world and there are cases where one is required to simulate a particular tree. One typical application is in *Virtual Reality* where there is a need to reconstruct digital real world scenes. In such a type of tree simulation the input is an objective form of a tree and the output is the reconstructed 3D geometrical structure of the tree. This is called a *top-down* tree simulation approach or a *backward* simulation.

This thesis lies within the area of forestry inventory systems where there is a need to validate and verify the interpretation of aerial or terrestrial images for different tree types, crown geometry by computer vision algorithms, where

3D tree models are needed for both visualization and verifying purposes.

1.2 Background

L-Systems, short for *Lindenmayer Systems*, are an effective plant modeling tool and were first introduced in 1968 by Lindenmayer [23]. They were primarily conceived as a mathematical theory for plant development. Now they are widely used to create plants and other fractal objects [37]. The key concept behind L-Systems is that complex biological objects can be generated by successively replacing parts of a simple initial object using a set of rewriting or production rules. In this scenario, the initial object is considered as a seed or an axiom and the rewriting or production rules determine the development of the seed.

Markov processes form an important class of stochastic processes. A Markov process is a system in which the process can be in one of several (numbered) states, and the process can pass from one state to another at each time step with some probability. A first-order Markov process assumes that the next state (at time $t + 1$) is only dependent on the current (time t) system's state. Markov models "are very rich in mathematical structure and hence can form the theoretical basis for use in a wide range of applications" and "when applied properly, work very well in practice for several important applications" [39].

1.3 Previous Work

Past researchers have developed different methods for backward tree simulation and modeling [42, 38, 45]. For example, Prusinkiewicz *et al.* [38] made use of positional information such as posture, gradual variation of features, and the progression of the drawing process from overall silhouette to local details, to interactively control the generation of plant forms. Sakaguchi *et al.* [42] introduced a method to reconstruct volume data, the visual hull, from images, and simulated branch structure by applying simple branching rules with some restrictions. Shlyakhter *et al.* [45] introduced another method for reconstructing 3D tree models from images. They first compute the *visual hull* [21] from

silhouettes and then extract trunks and branches from the visual hull and, finally, add leaves and branch lets onto the branches interactively.

1.4 Proposed Approach

The proposed approach to tree simulation is based on and extend the work of Shlyakhter *et al.* [45]. Briefly speaking, a stochastic version of L-System is introduced to create stochastic tree components such as branches and leaves, and an automatic process is used to append these components onto the extracted large branches to fit the shape of the visual hull. The extensions of our work mainly lie in the following two parts: first, we introduced a new version of stochastic L-System the automatic process for fitting small branches and leaves within the visual hull. Our work provides a basis for generating 3D tree geometrical models from images of trees via a stochastic version of L-System. In what follows, we focus on *terrestrial* images since the application to *aerial* images is straightforward.

Fig. 1.1 illustrates the system model. There are three parts: the visual hulls creation and skeleton finding from images, Hierarchical Hidden Markov Model (HHMM) L-System setting, and branches fitting into visual hulls. The initial input to the system is a series of tree images associated with their respective camera parameters. These images are generated from either synthetic or real data. Either way, silhouettes of target trees are manually segmented from the input images. A visual hull generation method is then used on the silhouettes where visual hull corresponds to a polyhedral approximation of the 3D object [21].

A skeleton approximation method proposed by Blum [5], extended by Teichmann *et al.* [28] and Shlyakhter *et al.* [45], is employed in our work to extract 3D medial axes of visual hulls. These axes are considered as skeletons, *i.e.*, trunks and branches, of the reconstructed 3D trees (see Chapter 4). HHMM L-Systems are initially designed with basic botanical knowledge. An Expectation Maximization (EM) algorithm is then proposed to train the HHMM L-System parameters (see Chapter 3). The final step is to generate

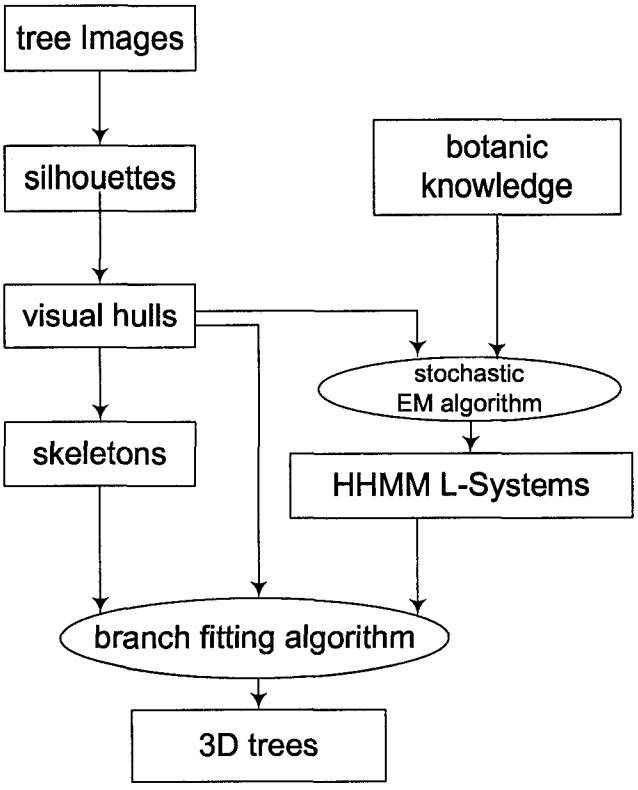


Figure 1.1: System model

a 3D tree based on the extracted skeletons and visual hull information using the HHMM L-System to add adequate branches and leaves within the visual hulls.

1.5 Organization

In Chapter 2 a basic introduction to plant biology is provided alone with some plant modeling methods, and an important and widely used plant modeling tool: L-Systems and its variants. We also review Markov models: Markov Chain, Hidden Markov Model (HMM) and Hierarchical Hidden Markov Model (HHMM). In Chapter 3 We extend L-Systems to Hierarchical Hidden Markov Model (HHMM) L-Systems. In Chapter 4 we introduce how to create visual hulls from multiple views of objects. In Chapter 5 we examine how a skeletonization method can be applied to extract skeletons from polyhedral visual hulls, In Chapter 6 we introduce an optimization procedure for fitting HHMM

L-System models within visual hulls. In Chapter 7 we report some experimental results. In Chapter 8 we conclude this work and discuss its limitation as well as future developments.

Chapter 2

Survey

In this chapter we review some background knowledge and previous studies. We first introduce some basic botanic terms related to plant modeling. These terms are necessary for readers to better understand the plant modeling description. We then review previous work done on plant modeling and, in particular, L-Systems are explored in detail. Finally, we review Markov processes, including Markov chains, Hidden Markov models, and Hierarchical Hidden Markov models.

2.1 Botanic Terms

Most plant models are described in botanic terms. The parts of a plant can be divided into two types: *vegetative (asexual)* and *reproductive* parts. The vegetative parts include *roots*, *stems*, *shoot buds*, and *leaves*. The reproductive parts include *flowers*, *fruits*, and *flower buds*. In this work, our applications deal with forests, and therefore our main interests are on the vegetative parts of plants.

Roots are the structures which are the lowest parts of a plant, usually underground, and are essential for supplying water and nutrition. Since most roots cannot be seen directly, they are not important for visualization, or for identifying plants. Therefore we ignore the modeling of root in this work.

Stems are structures that support buds and leaves. There are several types of stems. A *shoot* is a young stem with leaves. A *twig* is a stem that is one year old or less without leaves. A *branch* is a stem that is more than one year and

is typically a lateral stem. A *trunk* is a main stem of a woody plant. *Buds* are undeveloped shoots from which leaves or flower parts will arise. A *terminal bud* is located at the apex of a stem (see Fig. 2.1).

A *node* is the part of the stem where one or more leaves are attached. A *internode* is the region of the stem between two successive nodes. In Fig. 2.1, buds, leaves, internodes, and nodes are shown.

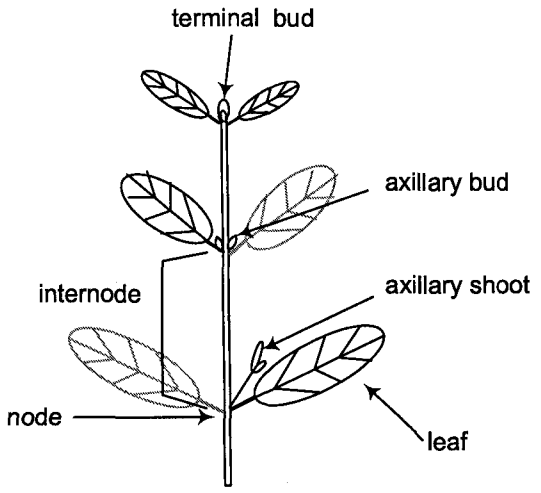


Figure 2.1: A shoot with leaves, an axillary shoot, axillary buds, nodes, internodes, and buds.

The growth of a plant is the result of the successive division of some specific cellular tissues, called *meristems*. Apical buds are one type of meristem. A *growth unit* is a set of nodes and internodes that are produced by the apical buds of the previous node. Growth units can be short, sometimes with only one internodes or can be very long. When a growth unit is very long, it usually consists of numerous short internodes [13, 16].

The *order of an axis* reflects the order of the apical buds from where the current axis develops. The order 1 axis is a sequence of nodes and internodes, and each node or internode is developed from the apical bud of the previous. The first internode of a sequence is developed from the seed of the plant. The order 2 axis includes another sequence of nodes and internodes with the same property except that the first internode is developed from an auxiliary bud on an order 1 axis. Likewise, an order i axis develops from an auxiliary bud on

an order $i-1$ axis. Usually, trunks are order 1 axes. In our work, we denote an order $i-1$ axis as the *parent axis* of an order i axis if the order i axis is developed from the apical bud on the order $i-1$ axis. Subsequently, the order i axis is a *child axis* of the order $i-1$ axis. An axis will have at most one parent axis, but might have a number of child axes (see Fig. 2.2).

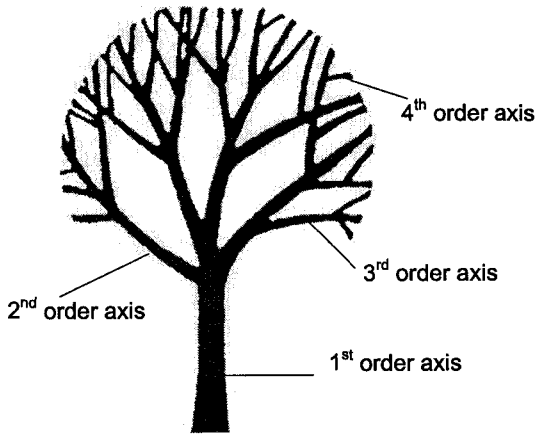


Figure 2.2: A tree marked with different orders of axes where the trunk is an order 1 axis, branches developed directly from the trunk are order 2 axes, and so on.

According to different ramification properties, where "ramification" means the act or process of branching out or dividing into branches, we can divide different ramification processes into the following three classes: *continuous ramification*, *rhythmic ramification*, and *diffuse ramification* [13].

- Continuous ramification: Each node of an axis will develop a higher order axis, and this node is the root of the higher order axis.

- Rhythmic ramification: Not all nodes of an axis can develop a higher order axis. Some nodes will develop into leaves and some into the roots of higher order axes.

- Diffuse ramification: The nodes that can develop higher order axes have random locations.

Another important feature of ramification is the trend of the direction of axes (or branches). If a child axis develops following the horizontal direction,

it is *plagiotropic*; If the orientation of an axis is vertical, it is *orthotropic* (see Fig. 2.3). Most trunks grow up-wards to compete for lights. Therefore, order 1 axes are usually orthotropic.

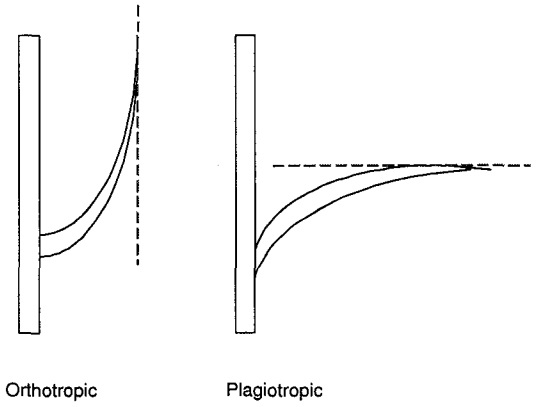


Figure 2.3: Illustration of two different branch trends: orthotropic and plagiotropic.

The term *phyllotaxy* means the arrangement of leaves on the stem or the relative positions of leaves of a node with respect to the lateral leaves of the previous node. Such arrangements follow regular rules known for each species and each order. There are three types of phyllotatic patterns recognized by the number of leaves at a node:

- *Alternate* phyllotaxy — one leaf at a node (see Fig. 2.4). In this type, leaves may be spirally arranged, that is, the angles between successive leaves are approximately 137.5 degrees (related to the *Golden ratio*i and the Fibonacci series). In one sub-type which is called *distichous*, the angles between successive leaves are 180 degrees. For example, aspen, elm, and beech leaves are of this type.

- *Opposite* phyllotaxy — two leaves at a node (see Fig. 2.4), which are opposite to each other; this pair of leaves is offset by 90 degrees relative to leaves at adjacent nodes. For example, maple, and ash leaves are of this type.

- *Whorled* phyllotaxy — three or more leaves at a node (See Fig. 2.4)). For example, smooth bedstraw and carpetweed have whorled leaves.

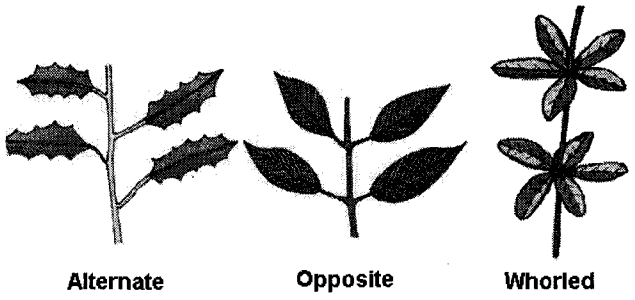


Figure 2.4: Three types of phyllotaxy: alternate, opposite, and whorled.

Qualitative and Quantitative Characters of Plants

The characters of a plant can be divided into two categories: *qualitative* and *quantitative*. Qualitative data are data which can be acquired without using any measurement tools. For example, the trend direction of branches, orthotropic or plagiotropic, phyllotaxy of leaves or branches, ramification types, etc. Qualitative data of the same species plants are the same.

Quantitative data are data collected by measurement. These data include internode length, angles within lateral shoots, thickness of trunks, etc. Quantitative data vary from specimens other than species. Both qualitative and quantitative data make a plant unique in the real world.

2.1.1 Plant Architecture

Plant structures have particular patterns that can be characterized by the combinations of very few characters such as qualitative characters which include branches trend, phyllotaxy, etc. Botanists analyze plants topological structures and have defined 23 plant architecture models. Each of these 23 plant architecture models has its own particular combination such that it covers a large number of species and is typically dedicated to a botanist, for example, Rauh's model, Massart's model, etc. Plants in the same model may appear quite different since plant architecture defines the growth strategies to occupy space which is shown as presence or absence of sympodial growth, ramification types, continuous or rhythmic, the direction trend of axes, plagiotropy or

orthotropy, etc [13].

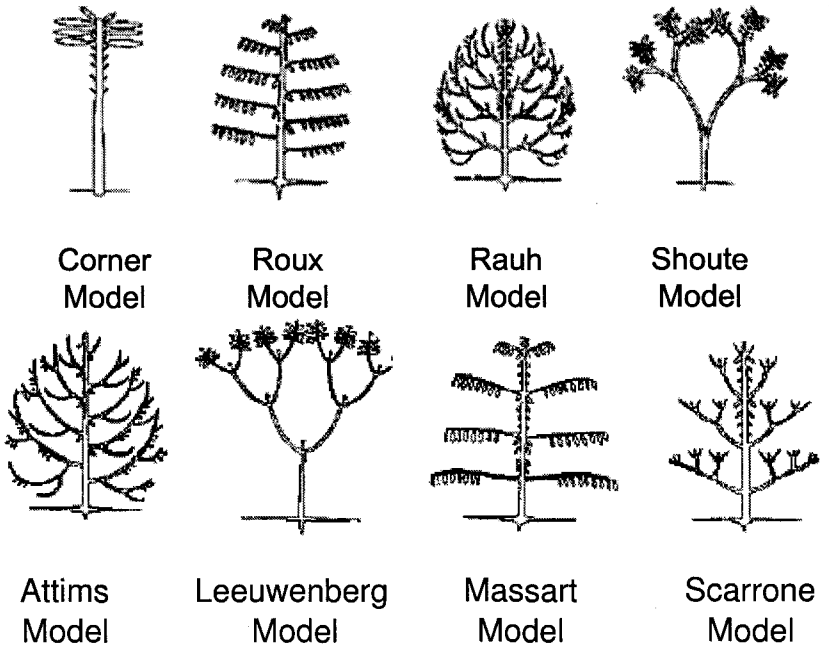


Figure 2.5: Illustrate some plant architecture models.

Plants belonging to Rauh's model have orthotropic axes for all order axes and rhythmic ramification, for example, an aspen tree. Plants belonging to Massart's Model have orthotropic order one axes and plagiotropic order i axes when $i > 1$. They also have rhythmic ramification [13], for example, a spruce tree. Plants belonging to Corner's model have one order 1 axis and no ramification, that is they have monopodial trunks but no sub-branches. Some tropical trees belong to these type, such as coconut trees. In Fig. 2.5, 8 types of plant architectures are shown to illustrate the difference among plant architectures, please refer to Halle *et al.* [17] for more details on plant architectures.

Plant architecture analysis was first developed as a qualitative method. Soon after, many researchers worked on architectural concepts and their applications [16]. Now it is regarded as a classic work on plant topological structure research and it plays an important role in verifying the effectiveness of plant modeling methods.

2.2 Plant Modeling Methods

The purpose of computational modeling is to construct simple mathematical models, which can simulate objects in the physical world [32]. The task of plant modeling is to describe plant spatial structures and their development process, where an individual plant is considered as a unit which contains its parts such as internodes, branches, and leaves. According to the difference among them, the modeling methods can be grouped into two classes: *empirical* models and *causal* models. Thornley *et al.* [47] have described the distinction between these two classes of models. Essentially, empirical models capture statistical characters of plants based on collected empirical data. Therefore, they are also called *descriptive* or *statistical* models. These models have advantages on making practical predictions based on the analysis of the acquired data. On the other hand, causal models can be used to explain the underlying mechanisms of plant development. They are also called *explanatory*, *physiologically based*, or *functional* models [19].

The relationship between empirical and causal models is very similar to the distinction between *analytic* (top-down) and *synthetic* (bottom-up). In the analytic case, we analyze the acquired empirical data based on the selected features and then construct models. While in the synthetic case we use models to synthesize known mechanisms of plant development.

In the literature, researchers have developed many algorithms to simulate plants geometrical structures in the past decades. To name a few, in 1962, Ulma [48] simulated branching pattern development with cellular automata. Cohen [12, 27] introduced a more realistic model in continuous space. Lindenmayer introduced the concept of L-System for cellular interaction modeling in 1968. Thereafter, Honda [18] introduced a model using parameters, including the branch angles and the branch length, to represent the shape of a tree and his model is considered as the first computer model of tree structures [27]. Currently, there are three methods that are mainly used. They are L-Systems, fractal methods and stochastic methods.

An L-System is a formal language and it uses symbol rewriting meth-

ods to create complex fractal objects. L-Systems are clearly introduced in the book "The Algorithmic Beauty of Plants" by Prusinkiewicz and Lindenmayer [37]. The early stage of L-System is called *D0L-System*, where the prefix "D0" indicates that it is deterministic and context free. We will give its detail description in the next section. *Parametric D0L-Systems* can produce the same structures with varying attributes. *Differential L-Systems* are developed on top of parametric L-Systems by allowing continuous time flow instead of the discrete derivation steps and are suitable for animating simulated developmental processes [35]. *Stochastic D0L-Systems* overcome the limitation of deterministic L-Systems and can produce stochastic plant geometry. Communication between plants modules affects the plants developmental processes significantly. Lindenmayer distinguished two types of communications: *cellular descent* and *interaction*. Cellular descent transfers information from parents to their children and interaction transfers information between co-existing modules [23, 24]. Cellular descent can be represented by context-free L-Systems and context-sensitive L-Systems are able to model some extrogenous factors during the plants development process. However, context-sensitive L-Systems are considered to have some limitation on modeling interactions between plants and environment and this type of interaction is a crucial factor affecting the development of plants and plant ecosystem. *Environmentally-sensitive L-Systems* [36] and *Open L-Systems* [27] are introduced to model more complicated interactions between plants within a typical environment. These two systems extend the formalism of L-Systems and are able to model bi-directional information transformation between plants and their environment.

Fractal methods use mathematical tools which can simulate the self-similarity structures of plants. The *Iterated Function System* (IFS) is a typical fractal method. Based on IFS Bransley *et al.* developed *recurrent IFS* [3] and Prusinkiewicz *et al.* developed *language-restricted IFS* [33]. Aono *et al.* developed a model to produce complex 3D branch patterns and their model uses attractor algorithm to simulate plant developments affected by some environment factors such as lighting, gravity, and wind [2]. Oppenheimer [30]

developed another fractal method which uses parameters such as branching angle, branch-to-parent size ratio, and stem taper rates. This method is limited to a small number of basic trees. Reeves *et al.* developed a partial system to create trees and grass [41]. The partial system is good at modeling landscape such forests and grasses lands.

Stochastic methods mainly include the following works. de Rayffe *et al.* [13] developed a procedural model which encodes probabilities of birth and death of plant components. Godin and Caraglio [16] further developed a multiscale model of plant topological structures. Zhao *et al.* [49] introduced dual-scale automaton which uses microstates and macrostates and semi-Markov chains to model plants.

2.3 L-Systems

L-Systems are an effective modeling tool and were first introduced in 1968 by Lindenmayer [23]. They were primarily conceived as a mathematical theory of plant development. The key concept behind L-Systems is that complex biological objects can be generated by successively replacing parts of a simple initial object using a set of rewriting or production rules each consisting of a specific geometric operation on an object. They are widely used to create plants and other fractal objects [37]. Each L-System object is ultimately defined as a string generated from an L-System rule, which includes an initial string, the *axiom*, and a set of rewriting rules called *productions*. An axiom consists of symbols with associated numerical parameters. Each production consists of a predecessor and a successor, connected by an equal sign, as shown in the following example:

- production: *predecessor* = *successor*
  - production 1: A = F[+A][-A]

    where "A" before "=" is a predecessor and the string after "=" is a successor of this production.
  - production 2: F = FF

where "F" before "=" is a predecessor and the string "FF" after "=" is a successor of this production.

The meaning of a production is that its predecessor is replaced by its successor during the rewriting processes. At each step of the rewriting process, axioms are updated, by replacing all the symbols with their corresponding productions, once.

For the previous example, if the axiom is "*A*", then the results of rewriting processes are shown as follows:

- Step 0: *A*
- Step 1: F[+*A*][-*A*]
- Step 2: FF[+F[+*A*][-*A*]][-F[+*A*][-*A*]]
- Step 3: FFFF[+FF[+F[+*A*][-*A*]][-F[+*A*][-*A*]]][-FF[+F[+*A*][-*A*]][-F[+*A*][-*A*]]]

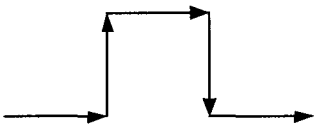
Notice that from Step 1 to Step 2, the italic letters "*A*" were replaced by "F[+*A*][-*A*]" according to production 1, and letter "F" was replaced by "FF" according to production 2. In the result of every step, all replaceable symbols ("*A*" and "F" in this example) are replaced by their successors according to the corresponding productions. Other symbols which have no productions ("+", "-", "[", and "]" in this example) are simply copied to the next step string.

Upon completion of these rewriting processes to a specified recursion step, a final string is generated. The final string corresponds to a set of geometric operations which can be interpreted by "*turtle interpretation*" [37].

The turtle interpretation converts the resultant L-String into a 3D geometrical model. In the 2D case, we can imagine that a turtle is on a blank sheet, and when the turtle moves, it will leave moving trails on the sheet. Four symbols, "F", "f", "+", and "-" are used to record the trails of the turtle. "F" means the turtle moves one unit distance in the current heading direction; The meaning of "f" is the same as "F", except that the turtle lifts its tail and doesn't leave a trail on the sheet. Symbols "+" and "-" denote that the turtle rotates to the left or to the right by a specific angle. By this simple interpretation, a two dimensional geometrical structure is drawn on the sheet.

As an example, we can interpret the string "F+F-F-F+F" as a simple 2D geometrical structure using the turtle interpretation rules just provided.

The resultant structure is shown in Fig. 2.6. A turtle is oriented to the right direction before it starts, and it moves forwards for one unit distance ("F"). It then turns left for 90 degrees ("+"), and its heading direction is upwards . "+" is short for "+(90)" when the default degree is 90. After turning, it keeps moving along in its heading direction for another unit distance ("F"). These two movements create the first two line segments in Fig. 2.6. Upon the ending of the full string, the whole structure will be generated as shown in Fig. 2.6.



F+F-F-F+F

Figure 2.6: An L-System string and its resultant structure after the turtle interpretation.

In the 3D case, the turtle states are defined by its position and three mutually perpendicular orientation vectors $\vec{H}$, $\vec{L}$, and $\vec{U}$ which indicate the turtle's heading direction, the direction to the right, the direction to the left, and up direction, respectively [38] (see Fig.2.7).

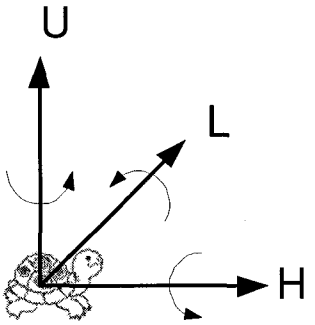


Figure 2.7: Illustration of the three directions: heading, left, and up directions of a Turtle in 3D space.

The interpretations of some basic symbols in 3D are given below (see the Appendix for a full list of symbols):

- "F": Move forward a step of unit length and connect the new position to the last position by a line segment. (The new position is computed by a formula.)
- "+": Rotate left by a unit angle (counterclockwise) around the up axis.
- "–": Rotate right by a unit angle (clockwise) around the up axis.
- "&": pitch down by a unit angle around the left vector.
- "∧": pitch up by a unit angle around the left vector.
- "<": roll left by a unit angle (counterclockwise) around the forward vector.
- ">": roll right by a unit angle (clockwise) around the forward vector

Using these 3D interpretation symbols, we extend turtle interpretation from 2D (sheet) space to 3D space and 3D geometrical structures can be recorded.

In order to specify the data structure for presenting axial trees, the concept of "strings with brackets" was introduced by Lindenmeyer [23]. With brackets, the current state of the turtle can be pushed onto a stack. One or more sub-structures can be drawn before the saved state is popped up and then we can keep drawing the main structure. In this way, with brackets, multi-layer structures or tree structures can be represented. In L-Systems, symbols "[" and "]" are used as brackets.

- "[": Push the current state of the turtle onto a stack.
- "]": Pop a state from the stack and make it the current state of the turtle.

In Fig. 2.8, a simple axial structure is shown where the main axis consists of three segments, from bottom up, "A", "C", and "E". It includes two sub-axes, which are drawn inside of two dashed circles, "B", and "D". The whole structure is represented as "A[B]C[D]E", which can be viewed as three parallel

growth processes existing at the same time: the process of the main axis and the two sub-axes.

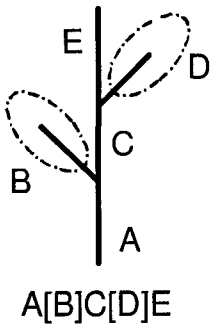


Figure 2.8: An axial structure represented using L-System with bracket structure.

As a plant modeling language, the L-System has the following benefits [32].

- Comparing to a general programming language, the programming effort needed to develop L-System models of plants is significantly reduced.
- Plant models can be easily modified in practice.
- The L-Systems make plant models documenting compact and precise.

Prusinkiewicz [34] has surveyed the applications of L-Systems to the modeling of plants and those L-Systems fall into three types: *Deterministic*, *Parametric*, and *Stochastic* L-Systems. The latter is of interest in our work.

2.3.1 Deterministic L-Systems

A *deterministic* L-System is the simplest type of L-System. An example of using deterministic L-Systems to generate a simple geometrical structure is given in Table 2.1 and the resultant object is shown in Fig. 2.9. It is obvious that the patterns of the object parts are the same as the patterns of the whole object, and this property is also called *self-similarity*, which is an important property of fractal objects.

Axiom:
F
Rules:
F $\longrightarrow$ F[+F]F[-F]F
Resulting strings:
Step 1: F[+F]F[-F]F
Step 2: F[+F]F[-F]F[+F[+F]F[-F]F]F[+F]F[-F]F[-F[+F]F[-F]F]F[+F]F[-F]F

Table 2.1: A typical deterministic L-System including an axiom, production rules for generating fractal objects.

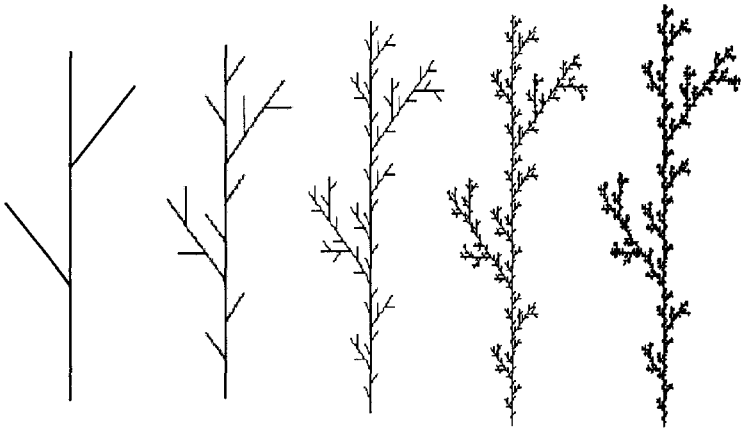


Figure 2.9: The object generated by (after scaled) the L-System defined in Table 2.1 with recursive depths 1, 2, 3, 4, and 5, respectively.

2.3.2 Parametric L-Systems

To include continuous variations in detail, Lindenmeyer proposed that numerical parameters, functions and rules can be associated with L-System symbols. *Parametric* L-Systems operate on parametric words, which are strings of modules consisting of their symbolic names with associated parameters.

Correspondingly, there are three parts in a production rule, which are predecessor, conditions, and successor. Symbols ":" and "$\rightarrow$" are used to separate these three parts:

$$\text{predecessor: conditions} \rightarrow \text{successor}$$

A predecessor is rewritten only when the production condition is met. For example, the parametric production given below works only when $x < 64$,

where $A(x)$ is replaced by $A(x+1)$. For A(100), since $x = 100 > 64$, this production does not work.

- $A(x)$: $(x < 64) \rightarrow A(x+1)$

An example used in Prusinkiewicz *et al.* [34] illustrates parametric L-Systems with different parameters to create different structures.

$w : A(100, w_0)$

$p1 : A(s, w) : s >= min \longrightarrow !(w)F(s)[+(\alpha_1)/(\psi_1)A(s * r_1, w * q \wedge e)]$
$[+(\alpha_2)/(\psi_2)A(s * r_2, w * (1-q) \wedge e)]$

Fig. 2.10 shows the parameter setting of each variable of the above Parametric L-System. The corresponding resultant structures are shown in Fig. 2.11.

Figure	r_1	r_2	α_1	α_2	φ_1	φ_2	w_0	q	e	min	n
a	.75	.77	35	-35	0	0	30	.50	.40	0.0	10
b	.65	.71	27	-68	0	0	20	.53	.50	1.7	12
c	.50	.85	25	-15	180	0	20	.45	.50	0.5	9
d	.60	.85	25	-15	180	180	20	.45	.50	0.0	10
e	.58	.83	30	15	0	180	20	.40	.50	1.0	11
f	.92	.37	0	60	180	0	2	.50	.00	0.5	15
g	.80	.80	30	-30	137	137	30	.50	.50	0.0	10
h	.95	.75	5	-30	-90	90	40	.60	.45	25.0	12
i	.55	.95	-5	30	137	137	5	.40	.00	5.0	12

Figure 2.10: 9 groups of parameter settings of the Parametric L-System defined in the text. This example is from Prusinkiewicz *et al.* [34], pages 17–18.

2.3.3 Stochastic L-Systems

All plants generated by a deterministic L-System are identical; With different input parameters sets, a parametric L-System can create different objects. However, the appearances of these objects might look different but they have the same development patterns, e.g., in Fig. 2.10, every apex develops into one internode with a pair of apices. In reality, we all know that there are no two plants in the world growing in the exact same way. Therefore, it is impossible to reuse deterministic or parametric L-Systems to create multiple trees. Consequently, *stochastic* L-Systems have been proposed to overcome this problem by introducing probabilistic transitions between symbols. The

Axiom:
F
Rules:
$R1$: F $\longrightarrow$ (0.33) F[+F]F[-F]F
$R2$: F $\longrightarrow$ (0.33) F[+F]F
$R3$: F $\longrightarrow$ (0.34) F[-F]

Table 2.2: A common stochastic L-System rule description for generating variant objects.

simulating process by using probability is a simple approach of simulating tree structure regardless the underlying natural rules.

For example, a stochastic L-System in Table 2.2, its production rules are three rewriting rules for the letter "F". In one derivation step, either $R1$, $R2$, or $R3$ is applied to each occurrence of an "F" according to the given probabilities: 0.33, 0.33, and 0.34. This concept of probability-based selection of turtle operators and numerical ranges for the operators to be formulated in terms of a stochastic rewriting form of L-Systems.

The first rule, "$R1$: F $\longrightarrow$ F[+F]F[-F]F", replaces "F" by "F[+F]F[-F]F" with probability 0.33. To interpret it graphically, consider "F" as a straight line with unit length. Rule 1 will replace a unit line by a pattern shown in Fig. 2.12.

The resultant structures in Fig. 2.13 are dissimilar, where each rewriting process follows probabilistically selected production.

2.4 The Process View of Plant Development

Plant development can be viewed as a process where some activities occur at discretized times. Usually, buds activities are emphasized since they are the most active plant parts. de Reffye *et al.* [13] state that the activity of buds can be one of the following:

- becoming a flower and die;
- going into sleep (doing nothing during that time period);
- becoming an internode which includes axial buds and apical buds;

- dying and disappear.

Once one of these activities is activated, more buds may be developed and they, in turn, create more activities. The activity is initiated with some probability that can be acquired from experimental statistic data.

We consider the bud development and all the possible activities as states, and consider the states activating processes as state transition processes. Then we can describe this plant activity progression by a state transition graph or a Markov chain. The states, state transitions, and probabilities among state transitions make it natural to use Markov chains as models [32]. In [32], it has been proved that a stochastic L-System model can be converted to a Markov chain model. A stochastic L-System can also be represented by Markov chains and stochastic L-System rules jointly.

2.5 Markov Processes

Compared with deterministic processes stochastic processes are much more common in reality. Markov processes are one of the most used stochastic processes. In the following section, we introduce *Discrete Markov Processes*, *Hidden Markov Models*, and *Hierarchical Hidden Markov Models.*

2.5.1 Discrete Markov Processes

Given a system with a finite set of states $\{S_1, S_2, \ldots, S_N\}$ at discrete times $t = 1, 2, \ldots$, the states of the system will change according to a set of probabilities. We denote the state at time t as q_t and the probability of the current state being chosen (at time t) is conditional on all predecessor states. For a first order Markov chain, the probability only depends on the preceding state:

$$P(q_t = S_j | q_{t-1} = S_i, q_{t-2} = S_k, \ldots q_1 = S_m) = P(q_t = S_j | q_{t-1} = S_i).$$

We consider that the process is stationary in time and is defined by a set of state transition probabilities a_{ij},

$$a_{ij} = P(q_t = S_j | q_{t-1} = S_i), \qquad 1 \leq i, j \leq N,$$

where a_{ij} obeys the following standard stochastic constraints:

$$a_{ij} \geq 0, \quad \forall i, j; \quad \sum_{j=1}^{N} a_{ij} = 1, \quad \forall i.$$

An example of a Markov chain with three states is shown in Fig. 2.14, where the state transition probabilities can be written in matrix form as

$$A = \{a_{ij}\} = \begin{bmatrix} 0.3 & 0.2 & 0.5 \\ 0.3 & 0.4 & 0.3 \\ 0.2 & 0.7 & 0.1 \end{bmatrix} \tag{2.1}$$

The observations are a set of states which are chosen at each time step and the states correspond to physical (observable) events. For this reason, the above stochastic process is called an *observable* Markov model [39].

2.5.2 Hidden Markov Models

Observable Markov models have many restrictions for real life applications as they do not dissociate states from observations or parametric values. For this reason, a more general model, an *Hidden Markov Model* (HMM), is typically used. The observations of states in HMMs are unobservable, but they can be predicted through another set of stochastic processes which generate a sequence of observations[39].

The elements of an HMM are:

- the number of states: N;
- the number of distinct observation symbols per state: M;
- the prior probability state vector: π;
- the state transition matrix: A;
- the state-dependent observation matrix for each state: B.

We use the compact standard notation $\lambda = (A, B, \pi)$ for an HMM. There are three questions to answer:

1. Compute the probability of this sequence generated by the coin model.

2. Determine which coins most likely produce "head" or "tail" observations over time.

3. How do we update the coin tossing model given this observation sequence?

The interested readers may see Rabina *et al.* [39] for detailed discussion of these three problems.

In this application, we do not come up with solutions to these three problems. Rather, HMMs are used here to predict observation sequences. As we will see, a variant of the *Expectation Maximization* (EM) algorithm for solving the third problem can be used to update the model parameters.

2.5.3 Hierarchical Hidden Markov Models

Hierarchical Hidden Markov Models (HHMMs) are generalized HMMs and are structured multi-level discrete stochastic processes introduced by Fine *et al.* [15]. They are originally designed for the study of some pattern recognition areas, such as language, handwriting, speech, since their natural sequences have complex multi-scale structures.

Each HHMM state is also an HHMM, *i.e.* each state is a similar stochastic model and a sequences. Note that in HMMs, most observations are single symbols. A sequence is produced by recursively activating the sub-HHMMs of a state. These recursive processes will end when they reach a special type of states called *production states*, which are considered leaf nodes of the HHMM. Only these production states can emit output symbols or observations. These symbol emitting processes are done in the same way that HMMs emit observations. Thus, HHMMs observations are chosen according to some probability matrices as well. Other states that can't emit observable symbols are called *internal states.*

An HHMM can be viewed as a tree structure and it has two types of state transitions: *vertical* and *horizontal.* Vertical transitions are the state

transitions that occur when a state activates its sub-HHMM and the sub-HHMMs are denoted at a lower level. So vertical transitions are the transitions between two different levels. Horizontal transitions only occur in the same level. In HMMs only horizontal transitions exist because of their single level structures. A production state has horizontal transitions only and an internal state has both types of transitions. In addition, there is a final state at each level, denoted by q^d_{end}, except the root level. When a final state is activated, the control will return to the parent level and jump out of the current level. So it's also a special vertical transition but it is different from the vertical transitions which activate sub-states. As we can see, for these final states, the control is returned to the parent level determinately.

More formally, let Σ be a finite alphabet and Σ^* be the set of all possible sequences over Σ. An observation sequence $\bar{O} = o_1 o_2 o_3 \ldots o_T$ is a finite sequence in Σ^*. The level index of the root state is 1 and that of the production states is D. A state of an HHMM is denoted by q^d_i, where d is the level index and i is the state index at this level. The numbers of internal states at different levels are not necessary to be the same and the number of sub-states of an internal state q^d_i is denoted as $|q^d_i|$. Similar to HMMs, HHMMs also have state transition matrices and observation matrices. An HHMM has two types of state transition matrices for its vertical and horizontal transitions, respectively. The horizontal transition matrix is denoted by $A^{q^d} = (a^{q^d}_{ij})$, where $a^{q^d}_{ij} = P(q^{d+1}_j | q^{d+1}_i)$ is the probability of making a transition from q^{d+1}_i to q^{d+1}_j, *i.e.* a horizontal transition from the ith state to the jth state at level $d+1$, which are children of state q^d. Similarly, $\Pi^{q^d} = (\pi^d(q^{d+1}_i)) = P(q^{d+1}_i | q^d)$ is the initial distribution vector or a vertical transition vector and includes the probability that the ith state at level $d+1$ is initially activated by state q^d and The production states are associated with observation vectors: $B^{q^D} = (b^{q^D}(k))$, where $b^{q^D}(k) = P(\sigma_k | q^D)$ is the probability that the symbol $\sigma_k \in \Sigma$ is emitted by the production state q^D. Thus an HHMM has the parameters set [15]:

$$\lambda = \{\lambda^{q^d}\}_{d\in\{1,..,D\}} = \{\{A^{q^d}\}_{d\in\{1,..,D-1\}}, \{\Pi^{q^d}\}_{d\in\{1,..,D-1\}}, \{B^{q^D}\}\}.$$

A string (observation) is generated in the following way. The root state

has the control initially and activates one of its sub-states at level 2 with probability Π^{q^1}. Every internal state q^d activated by its parent state, will in turn choose one of its own sub-states with probability Π^{q^d}. In Fig. 2.15, the dash lines with arrows illustrate such processes. The recursive process will keep on until a production state q^D is activated, where a single symbol is emitted with probability B^{q^D}. In Fig. 2.15, the small black circles are attached on production states and these circles denote the observations. Subsequently, the control returns to the internal state which activated q^D. This internal state will pass the control to another state in the same level according to this level's state transition matrix (A^{q^d}). In Fig. 2.15, the solid horizontal lines with arrows illustrate horizontal transitions. The newly chosen state will begin another string generation process. When a final state q^d_{end}, shown in grey circles, is reached, the control returns to the parent state of the whole hierarchy. In Fig. 2.15, the double dash lines with arrows illustrate such processes. The process will repeat till the root state is reached again and an observation sequence is thus created [15].

Fine *et al.* used HHMMs in handwriting recognition applications and introduced how to estimate all the parameters in an HHMM. For more information, please refer to [15].

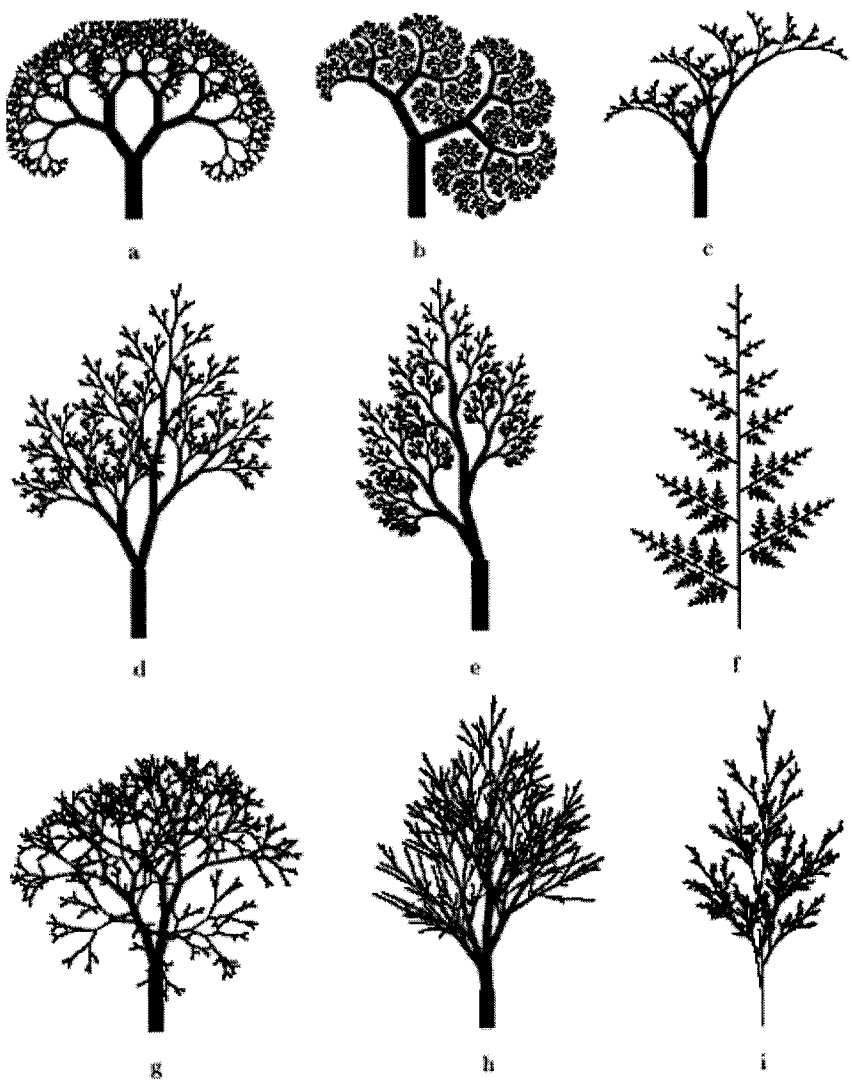


Figure 2.11: 9 structures generated by a parametric L-system with different initial parameters given in Fig. 2.10. This example is from Prusinkiewicz *et al.* [34], pages 17–18.

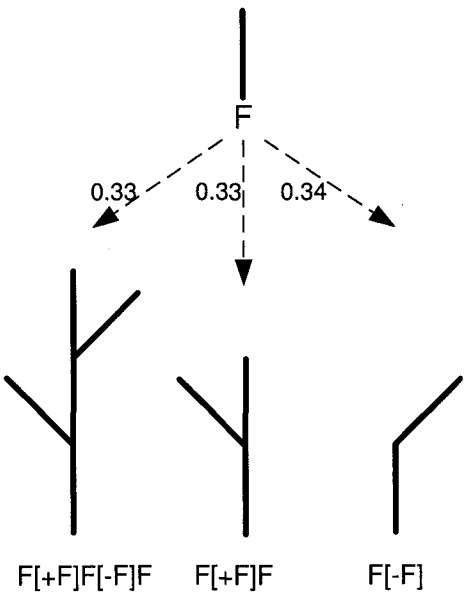


Figure 2.12: The rewriting option rules defined in Table 2.2 with their associated probabilities.

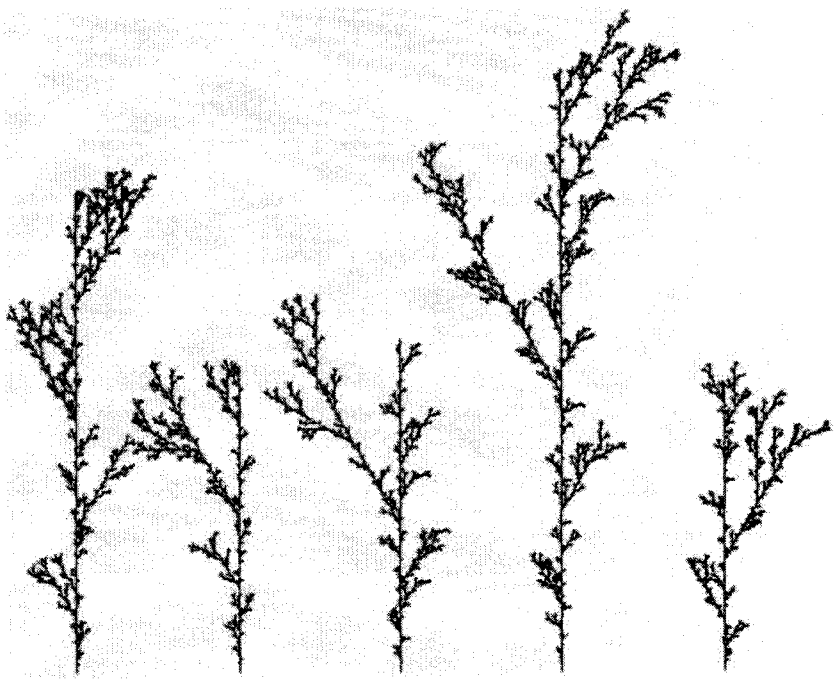


Figure 2.13: The objects generated from the stochastic L-System rules defined in Table 2.2.

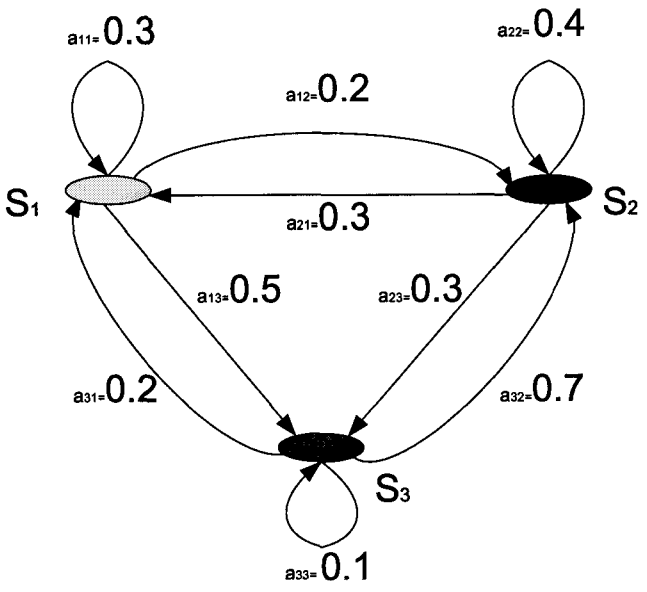


Figure 2.14: A 3-state Markov chain $\{S_1, S_2, S_3\}$, with state transition probabilities corresponding to directed edges on the finite state graph.

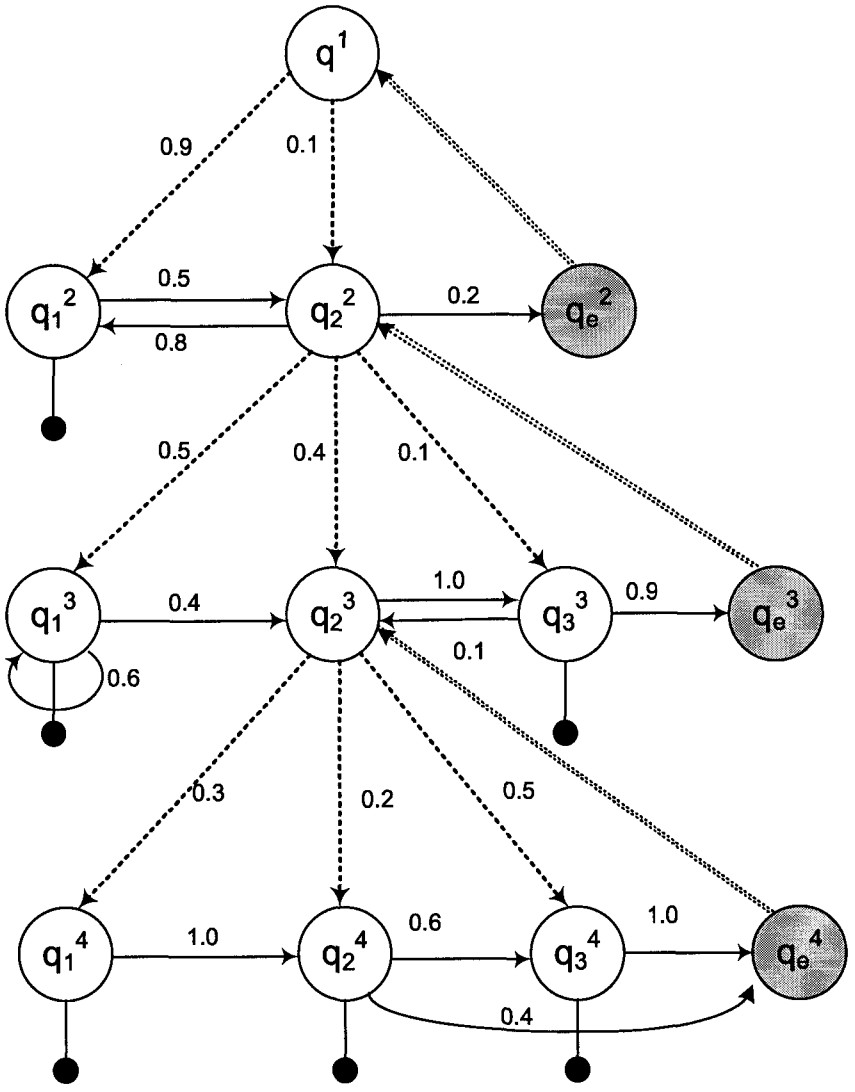


Figure 2.15: The topological structure of an hierarchical hidden Markov model (HHMM), where single dash lines illustrate vertical transitions, solid lines illustrate horizontal transitions, and numbers attached on transition lines denote the transition probabilities. The black circles are observations of production states.

Chapter 3

Hierarchical Hidden Markov Model L-Systems

In this chapter we introduce a new type of stochastic L-Systems: *Hierarchical Hidden Markov Model L-Systems* or HHMM L-Systems for short.

3.1 HHMM L-Systems

What differentiates trees within a given species is the pattern of their branching process. To present all these different patterns as well as all possible stochastic rules, a standard stochastic L-System has to define numerous rules and these rules appear independent to each other. Moreover, a stochastic L-System lacks a clear structure, which causes a designer more managing all its parameters and rules.

It has been proved that stochastic L-Systems can be re-represented jointly by L-System rules and Markov chains [32]. There exists a multi-layer Markov model, the HHMM, and we integrate the stochastic L-System and an HHMM into an HHMM L-System. The HHMM L-System is a new type of stochastic L-System and it is highly structured.

Accordingly, different types of object features, at different scales, can be created by changes at a given level of the rewriting procedures. See Fig.3.1 for the development process of a bud, where the numbers indicate the possibilities of the transitions and they don't reflect the natural bud development probabilities. The layers in the structure can be simply viewed as the increas-

ing process time step and the top layer is the starting. The bud at the top layer has some probabilities to be developed into branches or branches with buds, and the bud at the second layer can be developed into fruits by some probabilities, etc. These transitions can be modeled by HHMMs' vertical and horizontal transitions. Also tree parts can have different attributes, e.g, leaves can vary from sizes, colors, orientations etc. These attributes can be modeled by HHMMs' hidden observation functions.

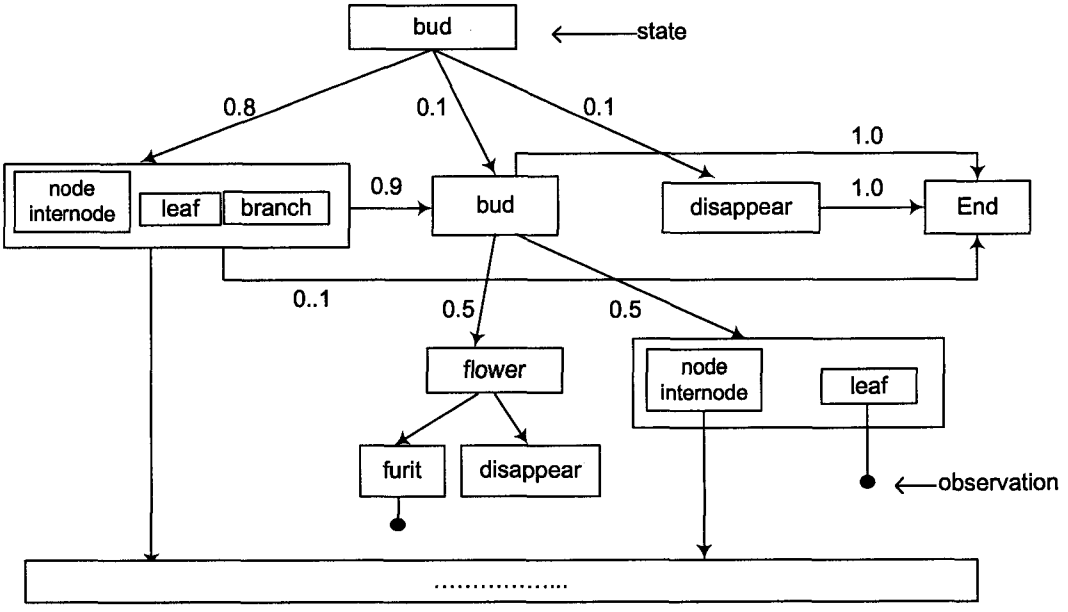


Figure 3.1: An illustration of the development trend of a bud

As with HHMMs, an HHMM L-System also includes two types of states: *internal* and *production*. The observations of internal states are sub-HHMMs. Only production states can emit observation symbols with probabilities. Each observation is an L-System symbol associated with a parameter and this parameter is randomly generated according to some distribution, which is usually Gaussian distribution and distribution vectors. For example, the observed symbol is "F" and its associated parameter is generated from a Gaussian distribution (1, 0.5) i.e. the mean $\mu = 1$ and the standard deviation $\sigma = 0.5$. Its observation, denoted as "F(x)", has the fixed symbol "F" and numerical variable x, which is randomly sampled from the Gaussian distribution at each process. For example, "F(0.8)", "F(1.2)", and "F(1.1)" are three possible

observations. In turtle graphs, these different observations are three line segments with different lengths. Recall that, as we introduced in Chapter 2, plant characters can be classified into two categories: quantitative and qualitative, where quantitative data of plants include internode lengths, angles within lateral shoots, thickness of trunks, etc. and they vary from difference specimens. To simulate different quantitative data, we use the distribution parameters for some L-System symbols to enable the function of creating stochastic lengths, angles, and thickness etc.

Accordingly, There are two types of state transitions in an HHMM L-System: vertical and horizontal. A vertical transition occurs when a state q_i^d activates its child state q_i^{d+1} [1]. This type of transitions can also be regarded as a predecessor which activates the rewriting process, and the predecessor is going to be replaced by its successors. Here the state q_i^d is the predecessor and the successors will be the observation sequence generated when the control is given back to the state q_i^d. A horizontal transition is the transition within the states at the same level. So when the same child state is activated, its observations are not determinate and unpredictable. With the functions of both vertical and horizontal transitions, a predecessor can be replaced by different successors. *Terminal states*, or end states, are used to indicate the end of a horizontal transition at a certain layer. When a terminal state is reached, the generation procedure will return from the current level back to its upper level. We observed that the structure is defined by a hierarchy and so it is appropriate to call it a type of Hierarchical Hidden Markov Model (HHMM) [2].

In an HHMM L-System, a state can appear at different levels and these states that appear later are usually omitted. The L-System symbols that have their default turtle interpretations can also be redefined as HHMM states. Thus they can be considered as either production states or internal states depending on the recursive depth, i.e, when the recursive depth is reached, these

[1] The denotations are used for HHMM states [15] and have been introduced in Chapter 2.

[2] There are many different formulations of hierarchical hidden Markov models and this is the most similar to the one discussed in Singer *et al.* [15].

symbols are considered as L-System symbols with meaningful turtle interpretation, otherwise they are replaced by user defined sub-HHMMs.

An L-String is generated from an HHMM L-System in the following way. The root or the first level of the hierarchy, which is considered as the axiom of the L-System, has the control initially. Each symbol in the axiom will activate a sub-state at the second level by Monte Carlo sampling of the underlying probability densities. Every internal state activated by its parent state, will in turn activate one of its own sub-states with probability. If a production state is activate, it will emit an observation with probabilities. The "downwards" recursive process will keep on until a production state is activate or the specified iteration depth is reached and the control will be returned back to the parent state which was latest activated. This parent state, which is an internal state, will pass the control to another state in the same level. The whole process will repeat until the last root state is reached again and an observation sequence is thus created. This process is running in the same way with HHMMs except that a recursive depth is used to control the recursive depth.

Alternatively, we can describe this recursive process as L-Systems rules rewriting process. At step one, each symbol in the axiom will be replaced by the strings created from its sub-HHMM. The strings consist of both the symbol of internal states name and observations emitted by production states and are regarded as the observed L-Strings after step one. At step two, each internal states in the L-String will be replaced by the sub-HHMM from the next level. That is, elements in the L-String will be further replaced by observations inferred from the HHMMs. This rewriting procedure iterates until the specified iteration depth is reached.

We consider some examples to help understand HHMM L-Systems and their relationships to stochastic L-Systems. In a standard stochastic L-System, with rules given in Table 2.2(Chapter 2), its first production rule $R1$ replaces "F", which is a unit length straight line interpreted as a tree structure that includes a line segment of three unit lengths and two sub-branches(see Fig. 2.12). The second rule $R2$ replaces "F" with a simpler tree structure which includes

a main axis of two unit lengths and a left branch. The third rule $R3$ replaces "F" with a branch structure with one unit length axis and a right branch from the axis (see Fig. 2.12). These three rules $R1$, $R2$, and $R3$ are chosen during the rewriting processes using the probability vector $[0.33, 0.33, 0.34]$. The resulting structures are different combinations of the above three rules (see Fig. 2.13). This rewriting process can be regarded as a Markov chain.

We developed an HHMM L-System that can produce similar stochastic structures to the above standard stochastic L-System. See Fig. 3.2 for its topological structure and the observation matrix of the state "D" is

$$\begin{bmatrix} value & 1 & 2 \\ probability & 0.34 & 0.68 \end{bmatrix}.$$

Also, to reduce the number of states and the matrix size, brackets "[" and "]" are omitted from the symbol set. States inside a pair of brackets are predefined, in this example "B" will be replaced as "[B]".

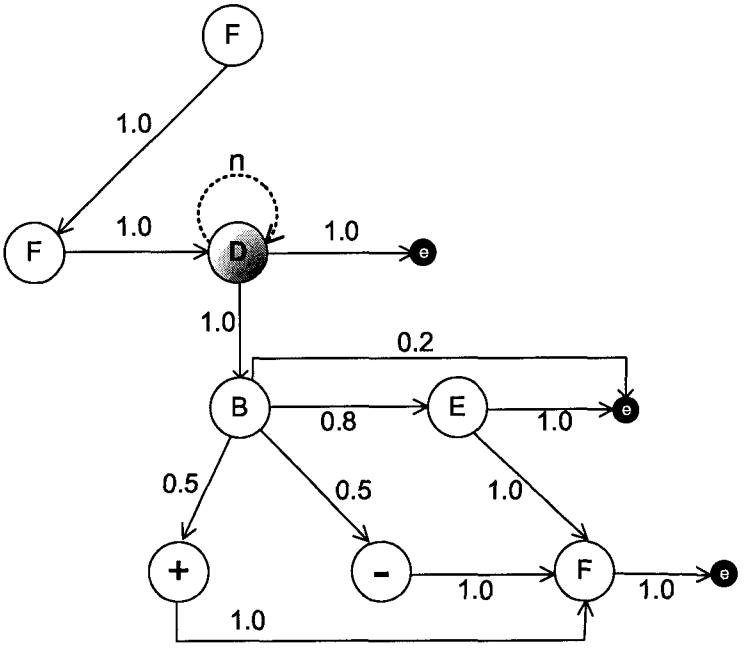


Figure 3.2: The topological structure of the HHMM L-System model generating objects in Fig. 3.3.

As shown in Fig. 3.2, the states of an HMM are L-System symbols (such as "F", "+", and "-") and HMMs symbols (such as "F", "D", "E", and "B").

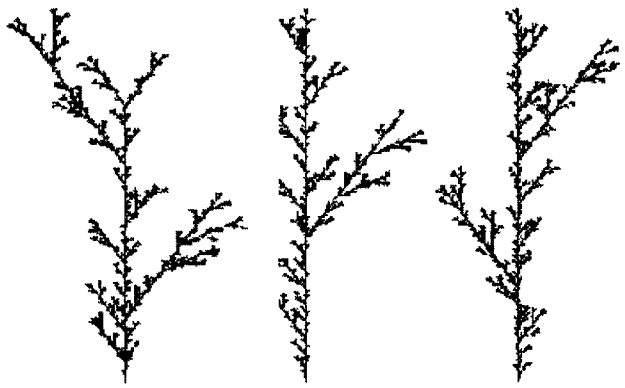


Figure 3.3: The objects generated by the HHMM L-System defined in Fig. 3.2 to be compared with the Stochastic L-System objects shown in Fig. 2.13.

Observations of an HMM are variable combinations of these symbols with numerical parameter values, for example, "FD(2)", and these parameter values are drawn from given distributions. The root of the HHMM L-System is the state "F", which is also a predefined L-System symbol. "F" has the successor "FD(x)". The observation matrix for "D" shows that two discrete values 1 and 2 have 66% and 34% probabilities to be chosen, respectively. Thus, the results are written as "D(1)" or "D(2)", respectively. The state "D" is defined as a special type of state and it involves a self repeat function. The number of the self-repeat times is decided by its associated parameters. That is, "D(1)" will repeat "D" once; "D(2)" will repeat "D" twice, *i.e.*, "D(2) = DD"; and "D(n)" will repeat "D" n times. This type of self-repeat state is termed as "superstate". For example, during the growth procedure of a tree, from a node of a branch, the number of child branches is not fixed. Also, in a growth unit, the number of internodes varies. All child branches from the same node can be modeled using the same production rules and the internode in the same growth unit can also follow same production rules. That is, we can reuse some production rules but we need to control the number of these rules for each case and the superstate is used for this task. We can use botanical knowledge to define superstate properties to control the number of repeat processes and we control the times by the possible values in an observation matrix B. In this example we set these values as 1 and 2 with the associated probabilities.

For another example, the HMM used to generate

$$axiom = \text{NNSNNSNNSNNSNNSNNSNNSNNS}$$

has the state set $\{\text{N}, \text{S}, \epsilon\}$. Using this basic HMM, it is difficult to control the number of "NNS" in the observations by adjusting the transition matrix parameters. However, using superstates, the number of "NNS" can be easily controlled by creating a sub-HHM "M" which can model "NNS" and treating "M" as a super-state, and adjusting its self-repeat time, i.e. its observation parameters . The sub-states of "M" can either be a simple L-System rule such as "M = NNS" or a sub-HHM "M " with states "{N, S".

To balance the growth speeds of all states, we define a growth speed vector to control the number of rewriting times in each rewriting step. For example, we design three-layer HMMs to simulate a group of tree parts including stems and lateral branches with buds, and two-layer HMMs to simulate a single branch with buds, and a single HMM to simulate a single bud. And these three groups of tree components are possible candidate states of a same parent state and one of the three components will be developed in the next growth time unit. Whichever HMM states is activated, we expect that all their states can be traversed in the next rewriting step. Therefore, it is necessary to specify the growth speed of each production rule of an HHMM L-System such that tree parts which are designed by multi-layer HMMs can be completely developed in a single time unit. Growth speeds allow variant number of layers of HMMs for constructing different tree parts. As we can see from Fig. 3.2, "F" is replaced by "FD(x)", $x \in \{1, 2\}$, so a line segment is replaced by a unit line segment ("F") and branch structures ("D"). "D" is replaced by either "BF" or "B". That is, "D" is replaced by a sub-branch with an unit line segment("BF") or without an unit line segment("B"). "B" is not a L-system symbol and doesn't have any structure meanings, so we let "B" to be replaced by either "+F" or "-F", which are two sub-branches in turtle graphics. That is, "D" needs to be replaced twice to generate branch structures. Thus, the growth speeds for states "F","D","B","E" in Fig. 3.2 are set as 1,2,1,1 respectively. Therefore, stochastic structures can be generated using this HHMM L-System

Axiom:
NNSNNSNNSNNSNNSNNSNNSNNSNNB
Rules:
1. S=$'(.9)!(.9)$
2. N=$tF[\&'(0.8)!LBL|zL] > (137)[z\&'(.7)!LBL|zL] > (137)$
3. B=$tF[-'(.8)!(.9)\$LCL|zL]'(.9)!(.9)C$
4. C=$tF[+'(.8)!(.9)\$LBL|zL]'(.9)!(.9)B$
5. L=[$\{+(30)f(10) - (30)f(10) - (30)f(10) - (120)f(10) - (30)f(10)$
$-(30)f(10)\}$]

Rule 1 is used to reduce the lengths and diameters of the stems generated later.
Rule 2 is used to generate trunk stems and two lateral branches.
Rule 3 and 4 are used to generate branches.
Rule 5 is used to generate a single leaf.
The axiom consists of a number of "NNS", where "N" is a basic unit of the trunk and "S" can reduce diameters and lengths of upper stems.

Table 3.1: A deterministic L-System for creating simple conifer type trees.

to approximate the stochastic L-System in Table 2.2.

For more examples, to generate a more complex object, such as a natural tree, an HHMM L-System model is created from a series of existing deterministic L-System rules. A deterministic L-System rule is given in Table3.1 and a 3D tree generated using this L-System is shown in Fig. 3.4.

These two models share the same transition matrices at each hierarchy, and the same structure(see Fig. 3.5). Fig. 3.6 shows their resulting trees created by Monte Carlo sampling of the model parameters.

These two models are different in few observation distributions. All observation distributions are given in Table 3.2, where the Gaussian distribution function for state "s" is briefly denoted as "s" $\sim (\mu, \sigma)$:

In addition to Table 3.2, the state "C" at layer 3 has its distribution vector as follows:

$$\begin{bmatrix} value & 0 & 1 & 2 & 3 \\ probability & 0.05 & 0.1 & 0.75 & 0.1 \end{bmatrix}$$

To generate the second row of trees in Fig. 3.6, we set the axiom as "N(12)", the recursive depth as 5, and the growth speed vector for states [N, a, M, C, B, D] as [1, 1, 4, 3, 2,1].

The HHMM L-System model used to generate the first row of trees has

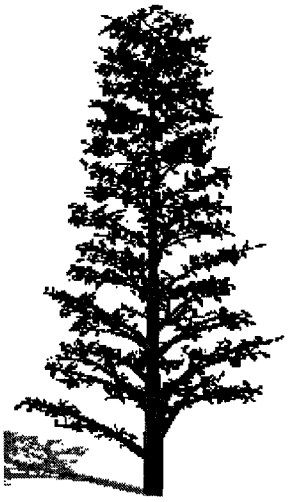


Figure 3.4: The conifer generated by the deterministic L-System defined in Table 3.1.

Layer 2:	'	$\sim$	$(0.95, 0.05)$	!	$\sim$	$(0.95, 0.05)$
Layer 3:	F	$\sim$	$(1.0, 0.1)$	$\wedge$	$\sim$	$(137, 10)$
Layer 4	&	$\sim$	$(40, 10)$	'	$\sim$	$(0.7, 0.01)$
	!	$\sim$	$(0.7, 0.01)$			
Layer 5	'	$\sim$	$(0.9, 0.02)$	!	$\sim$	$(0.9, 0.02)$
Layer 6	+	$\sim$	$(35, 10)$	$-$	$\sim$	$(35, 10)$
	'	$\sim$	$(0.9, 0.02)$	!	$\sim$	$(0.9, 0.02)$

Table 3.2: The state distribution functions of the HHMM L-System with structure in Fig.3.5.

the same setting except that this "&" Gaussian distribution function for "&" at layer 4 is $(40, 0)$ instead of $(40, 10)$. Since the variance of the Gaussian is 0, from turtle interpretation, the turtle pitches down around left vector for the fixed 40 degrees.

In summary, there are four basic elements in an HHMM L-System:

1. An Axiom.

   An axiom of an HHMM L-System consists of a sequence of symbols with their associated parameters. It is same with an axiom of a standard L-System.

2. Production rules, which can be defined jointly by classical deterministic L-System rules and HHMM rules.

   Like an HHMM, an HHMM rule has a hierarchical structure, . It includes states, transition matrices, observation matrices, and prior probabilities.

   There are two other types of states in an HHMM L-System besides production and internal states.

   - terminal states. These states indicate the end of the HMM sampling processes. When a terminal state is reached, the generation procedure will return from the current level back to its upper level.
   - superstates. Every superstate is followed by an integer n which indicates that this superstate will be repeatedly replaced by its successor n times.

   Usually, horizontal transition matrices are of *left-right model.* Moreover, self-loops are reduced by using super-states. In the other words, horizontal transition matrices are restricted left-right model. Observation matrices can vary according to the different levels of an HHMM.

3. A recursive depth vector.
   If an HHMM L-System axiom consists of a series of parallel HHMM rules, we can use a vector to specify recursive depths, *i.e.*, each HHMM rule can have a different recursive depth. Intuitively, branches closer to roots are better developed than these on top.

4. A growth speed vector.
   It can be used to control the repeating process. When no growth speed vector is specified, all rules are rewritten once at each rewriting step.

3.1.1 The Relation between HHMM L-Systems and Stochastic L-Systems

An HHMM L-System has more constraints on production rule selection and observation symbols and thus it has the following advantages:

- A clear structure which is more intuitive for tree generation.
- Observation matrices generate variable observations which allows to generate stochastic quantitative data of tree components.
- Superstates reduce the complexity of horizontal transition matrices and eliminate loops on horizontal transition and thus allow users to apply numerical botanic parameters, such as the number lateral branches, into the L-System rules directly.
- Growth speed vectors can control the repeating speed of each HMM and enable the flexibility of designing the HHMM rules.

3.2 Parameter Estimation for HHMM L-Systems

Hierarchical stochastic methods do play increasingly important roles in many application domains, as recently surveyed in [29], including image, video and audio processing, robot mapping, etc.. The specific advantage of these hierarchical methods lies in their capacity to model long-term interactions among different portions of the targeting signal sequence. L-Systems, and their variants, are special cases of the general modeling method that caters for the task of 3D plant modeling.

Fortunately, parameter estimation of HHMM L-Systems can be further constrained by only re-estimating the observation matrices (B) while preserving the transition matrices (A) and priori (π). To solve this simplified problem, we develop a simple extension of the stochastic EM method as follows.

Prior to the detailed descriptions of the algorithms, it is important to note that our task is to construct 3D CAD models based on the following:

1. A series of images, Y, that have the target object (tree) in the scene.
2. A 3D object D, that is, a 3D object encoding feature of real trees, *e.g.* a visual hull [21] constructed from Y.

In theory, the standard Baum-Welsh (EM) algorithm for HMMs can be utilized here for parameter estimation [40]. However, in our situation, the ob-

servation sequence (the L-String) is not directly observable, and, even worse, it is very hard to derive the L-String representation from a given 3D botanical object since several different L-Substrings may have the same 3D result. Instead, a sampling approach is a natural choice to overcome such difficulties. Also, because expert knowledge is encoded in the transition matrices and needs to be preserved, we propose to estimate only the observation matrices via our proposed stochastic EM algorithm as follows.

Define the hidden variables as $x_1, x_2, \ldots, x_L$ and they are matched with observation data $D = \{d_1, d_2, \ldots, d_L\}$ and the associated normal parameters for the observation matrix are $\theta_l \in (\theta_1, \theta_2, \ldots, \theta_L)$, where $\theta_l = (\mu_l, \sigma_l)$, μ_l is the mean and σ_l is the variance of variable. Notice that all the related variables are assumed independent. For variable v_l, it is difficult to compute the likelihood $p(d_l|\theta_l)$ from d_l alone, so we augment d_l with the hidden variable x_l to:

$$\overrightarrow{d_l} = [d_l, x_l] \tag{3.1}$$

and

$$\overrightarrow{D} = \{\overrightarrow{d_l}\}, \tag{3.2}$$

and similarly,

$$x = \{[x_1, x_2, \ldots, x_L], x_l \sim N(\mu_l, \sigma_l)\}. \tag{3.3}$$

We then formulate the *Maximum Likelihood* (ML) problem as:

$$\begin{aligned} \theta^* &= \arg\max_{\theta} p(\theta|\overrightarrow{D}) \\ &\propto \arg\max_{\theta} p(\overrightarrow{D}, \theta) \\ &= \arg\max_{\theta} p(\overrightarrow{D}|\theta) \cdot p(\theta) \end{aligned}$$

Assume $\overrightarrow{d_l}$ as *iid*, and θ_l as having uniform priors. Then

$$\begin{aligned} \theta^* &= \arg\max_{\theta^*} \prod_{l=1}^{L} [p(\overrightarrow{d_l}|\theta_l) \cdot p(\theta_l)] \\ &\propto \arg\max_{\theta^*} \sum_{l=1}^{L} \log p(\overrightarrow{d_l}|\theta_l). \end{aligned}$$

The stochastic EM method is employed to solve this ML problem. Define

$$\begin{aligned} L &= \log p(D|\theta) \\ &= \log \int_x p(D, x|\theta) dx \\ &= \log \int_x q(x|D) \frac{p(x, D|\theta)}{q(x|D)} dx \end{aligned}$$

where $q(x)$ is a distribution function.

Using Jensen's inequality, we have:

$$\begin{aligned} L &\geq \int_x q(x|D) \frac{\log p(x, D|\theta)}{q(x|D)} dx \\ &= \int_x q(x|D) \log p(x, D|\theta) dx - \int_x q(x|D) \log q(x|D) dx \end{aligned}$$

Let $Q(\theta) = \int_x q(x|D) \log p(x, D|\theta) dx$ and $H = -\int_x q(x|D) \log q(x|D) dx$. $Q(\theta)$ is the expected complete log-likelihood function and H is the entropy and it does not depend on θ.

EM algorithm:

E-Step : $q^{t+1} = \arg\max_q L(q, \theta^t)$

M-Step: $\theta^{t+1} = \arg\max_\theta Q(\theta)$

We can draw S samples for x according to $p(x|\theta)$, and let:

$$\widehat{h_{l,s}} = p(d_l|x_l, \theta_l).$$

Given

$$\begin{aligned} \mu &= < x > \\ \sigma^2(x) &= < (x - \mu)^2 > \\ &= < x^2 > -u^2, \end{aligned}$$

it follows that

$$\mu_l = < x_l >$$

$$
\begin{aligned}
&= \frac{\sum_{s=1}^{S} \widehat{h_{l,s}} \cdot x_{l,s}}{\sum_{s=1}^{S} \widehat{h_{l,s}}} \\
(\sigma_l)^2 &= < x_l^2 > - u_l^2 \\
&= \frac{\sum_{s=1}^{S} \widehat{h_{l,s}} \cdot x_{l,s}^2}{\sum_{s=1}^{S} \widehat{h_{l,s}}} - u_l^2.
\end{aligned}
$$

Furthermore, considering the hierarchical characteristics of the HHMM L-System, we can derive the following equations:

$$
\begin{aligned}
\mu_l &= < x_l > \\
&= \frac{\sum_{s=1}^{S} \widehat{h_{l,s}} \cdot \gamma_s \cdot x_{l,s}}{\sum_{s=1}^{S} \widehat{h_{l,s}} \cdot \gamma_s} \\
(\sigma_l)^2 &= < x_l^2 > - u_l^2 \\
&= \frac{\sum_{s=1}^{S} \widehat{h_{l,s}} \cdot \gamma_s \cdot x_{l,s}^2}{\sum_{s=1}^{S} \widehat{h_{l,s}} \cdot \gamma_s} - u_l^2,
\end{aligned}
$$

where $\gamma_s = p_{n(0)}^{n(1)} \cdot p_{n(1)}^{n(2)} \cdots p_{n(pa(l))}^{n(l)}$ are the product of the sampled vertical/horizontal transition probabilities along the path to the current node l.

Finally, the complete stochastic EM algorithm is defined as the follows:

1. Initialization: set the initial θ values $\theta^0 = \{\theta_l^0 : \mu_l^0, \Sigma_l^0\}$;

2. Sampling step:

   At time step t (t= 1, 2, 3,...), draw S samples of x by drawing in batch mode from each distribution $\theta^{t-1} = \{\theta_l^{t-1} : \mu_l^{t-1}, \Sigma_l^{t-1}\}$.

3. E step:

   Calculate

   $$\widehat{\Gamma^t(\theta)} = \sum_{l=1}^{L} \widehat{\Gamma^t(\theta_l)} = \sum_{l=1}^{L} \sum_{s=1}^{S} \widehat{h_{l,s}^t} \cdot \gamma_s, \tag{3.4}$$

   where $\widehat{h_{l,s}^t} \cdot \gamma_s$ is the weights of drawn samples;

4. M step:

   Compute the parameters

$$
\begin{aligned}
\mu_l^t &= \frac{\sum_{s=1}^{S} \widehat{h_{l,s}^t} \cdot \gamma_s \cdot x_{l,s}}{\widehat{\Gamma^t(\theta_l)}} \\
(\Sigma_l^t)^2 &= \frac{\sum_{s=1}^{S} \widehat{h_{l,s}^t} \cdot \gamma_s \cdot x_{l,s}^2}{\widehat{\Gamma^t(\theta_l)}} - (u_l^t)^2
\end{aligned}
$$

5. Iterate through step 2 to step 4 until θ converges.

3.2.1 Experiments

In our experiments, we initially designed an HHMM L-System according to the basic botanic knowledge of the tree species to be estimated. A 3D shape was provided to help estimate the HHMM L-System parameters. We set the recursive depth of the HHMM L-System according to the height, width, area of the 3D shape, etc. We chose k image planes and projected the 3D shape onto these k image planes to obtain k 2D images, each containing a 2D silhouette of the 3D object. These 2D silhouettes were used to compare with each sample, and denoted as φ_j, $(j = 1, 2, \ldots, k)$.

We then initiated a recursive process and at each loop to generate n samples from the HHMM L-System model. During the generation process, all parameters were recorded in stacks and were used for updating the HHMM L-System model at the end of each loop.

From each HHMM L-System generated tree sample we obtained its 3D geometrical structure and projected the 3D object onto these same k image planes and obtained k 2D silhouettes. This resulted in k new silhouettes for each sample, denoted by

$$\{\lambda_{i,j}\} = \{\lambda_{i,1}, \lambda_{i,2}, \ldots, \lambda_{i,k}\}, i = 1, 2, \ldots, n,$$

where i is the index of the generated sample in a loop.

We compared each silhouette $\lambda_{i,j}$ of this sample with the corresponding 2D silhouettes φ_j of the 3D shape. That is, we compared φ_j with $\lambda_{i,j}$ and we calculated a similarity score of the ith sample by

$$w_{i,j} = \frac{\cap(\varphi_j, \lambda_{i,j})}{\cup(\varphi_j, \lambda_{i,j})},$$

for "$\cap$" corresponding to the intersection area of the jth silhouette of rendered samples and the input jth 2D silhouette. To simplify computation, we used the number of overlapped pixels of two silhouettes as the "$\cap$" area. Similarly, "$\cup$" corresponding to the union area of the jth silhouette of rendered samples

and the input jth 2D silhouette. We use the number of union pixels of two silhouettes as the "$\cup$" area.

We reduced the similarity score when parts of the 3D sample were outside of the input 3D shape by adding a penalty. For example,

$$w_{i,j} = \frac{\cap(\varphi_j, \lambda_{i,j})}{\cup(\varphi_j, \lambda_{i,j})} \cdot \frac{\varphi_j}{\cup(\varphi_j, \lambda_{i,j})C}$$

where C is a constant used to adjust the penalty. The larger C is, the smaller the similarity score will be.

After getting all similarity scores for k silhouettes, we can compute the overall similarity score of the current sample level by taking w_i to be the average of the $w_{i,j}$ for $j = 1, 2, \ldots, k$, i.e.

$$w_i = \frac{1}{k} \sum_{j=1}^{k} w_{i,j}$$

After computing n samples, we have a weight vector:

$$\{w_i\} : \{w_1, w_2, \ldots, w_n\}$$

The similarity score at each loop was computed by:

Similarity score $= \sum_{i=1}^{n} w_i$

We found the samples which have low weights, say lower 25and set their weights as 0 and then normalized the samples weights.

$$\{w_i'\} : \{w_1', w_2', \ldots, w_n'\} \text{ and } \sum_{i=1}^{n} w_i' = 1$$

The normalized weight vector is used to update the parameters of the HHMM L-System. The samples with low weights didn't count on the updating processes.

Different probability distributions need different updating method. Gaussian distribution and distribution vectors were used in our work. For Gaussian distribution (μ, σ),p_i was the value drawn from the Gaussian distribution at ith sample, and at the end of one loop we obtained new parameter values(μ^t, Σ^t) by setting

$$\mu^t = \sum_{i=1}^{n} w_i' \cdot p_i,$$
$$(\Sigma^t)^2 = \sum_{i=1}^{n} w_i' \cdot p_i^2 - (\mu^t)^2.$$

where $p_{l,i}$ was the drawn value from $\theta_l : \mu_l, \Sigma_l$ for the ith sample and w_i' was the normalized weight of the ith sample.

For a discrete distribution vector, we simply computed the percentage of each value and used the new percentage to replace the old parameter. After updating all applicable parameters, we obtained a new HHMM L-System and repeated the above process for t times, or until parameters converged.

In one experiment, 100 independent samples were generated from the HHMM L-System at each loop, and 12 loops were repeated to update the parameters of the HHMM L-System. At each loop, 75 samples were counted for updating parameters. The scaling factor C was set as 5. One sample from each loop was randomly chosen and shown in Fig. 3.8 and a silhouette is shown in Fig. 3.7, and in this experiment, 4 identical silhouettes were used for weight estimation. The similarity score curve of estimating an HHMM L-System is shown in Fig 3.9. This curve contains 12 increasing similarity scores for 12 loops.

We noticed that the similarity scores are relatively small since the input silhouettes are usually much larger than the areas of samples and even the weight of an ideal sample could not reach 1 since the silhouettes used are the overall shape of trees and they don't contain holes. However, the projections of the sampled trees have much spaces among branches.

Moreover, when a sample goes outside of the silhouette, a penalty will rapidly decrease the similarity score.

In the experiments we obtained reasonable branch length and angle parameters, which makes the generated trees have close live crown ratios [3] with the silhouettes.

The limitation of the estimation is due to insufficient information used for

[3]Live crown ratio is the percentage of the length of the stem which has live branches.

weight estimation. In our case, only silhouettes are used to do the comparison and the similarity scores only reflect partial weight of the samples. with a global comparison strategy some parameters are re-estimated more sufficiently than others. In this cases the lower order of axes such as trunks and large branches are better estimated.

To this stage we have developed an HHMM L-System for the generation of 3D objects where all rewriting rules are replaced by sets of matrices that define state transitions and expected values of transformations. It would be ideal if we could fit such models with sensed image data. To that goal, then, we need to explore how to integrate such a model into the visual hull of an object constructed from multiple images, the topic of the following chapters.

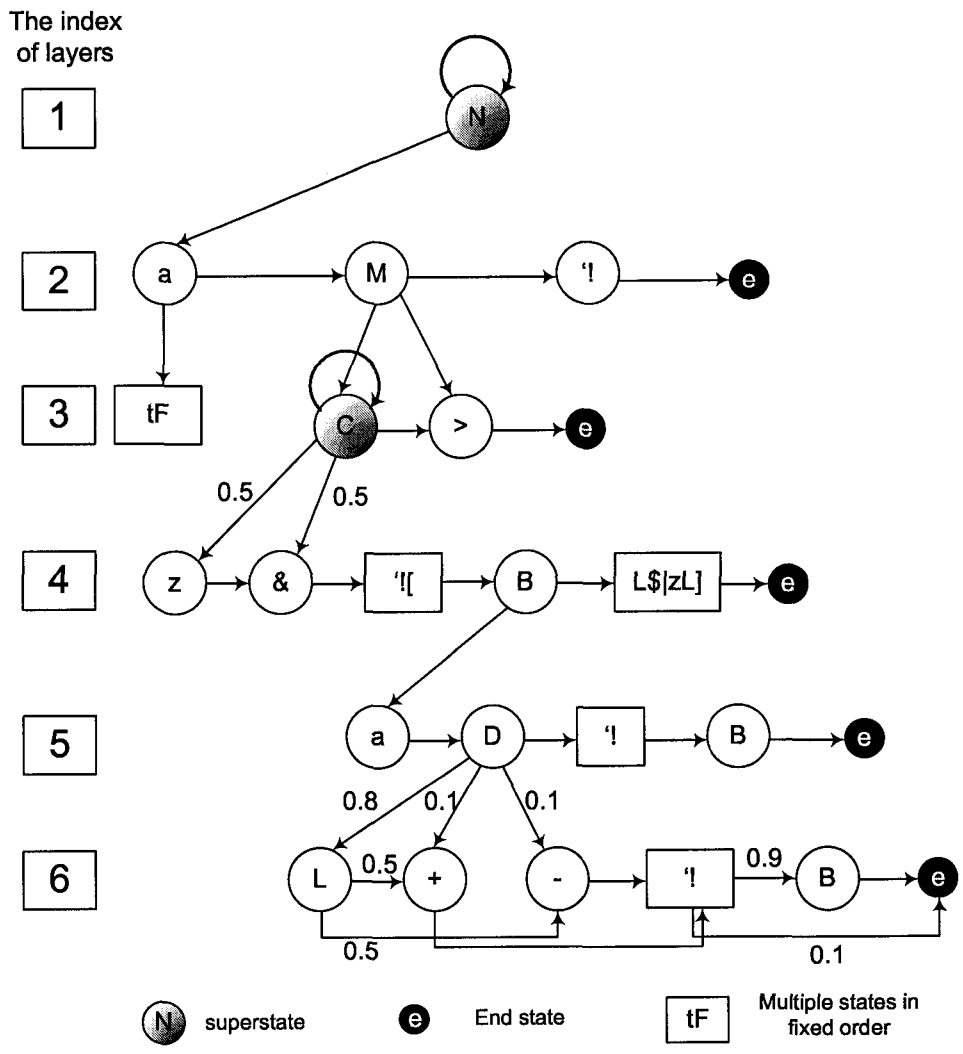


Figure 3.5: The structure of the HHMM L-System to generate trees in Fig.3.6. Probabilities of 1.0 are omitted in the graph. States "C", "B", "D" are defined as included in brackets "[" and "]", thus "C" will be converted to "[C]". The HHMM L-System observation functions are given in the text.

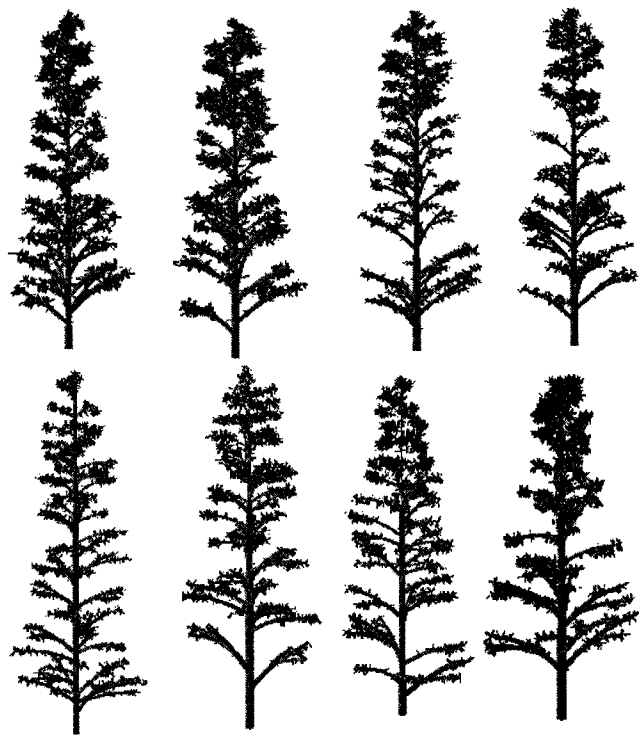


Figure 3.6: Two groups of trees generated from the two HHMM L-System Models. To show the branch structures more clearly, the trunk diameters are reduced.

Figure 3.7: A silhouette is used for the stochastic EM algorithm in our experiment.

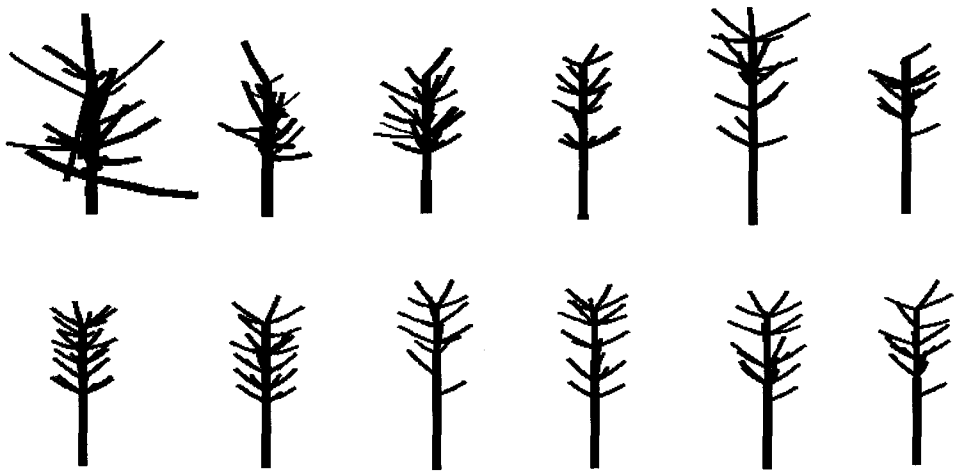


Figure 3.8: Samples chosen from the 12 loops, one sample from each loop, shown from left to right. Samples from the first to the sixth are on the first row and seventh to twelfth on the second row.

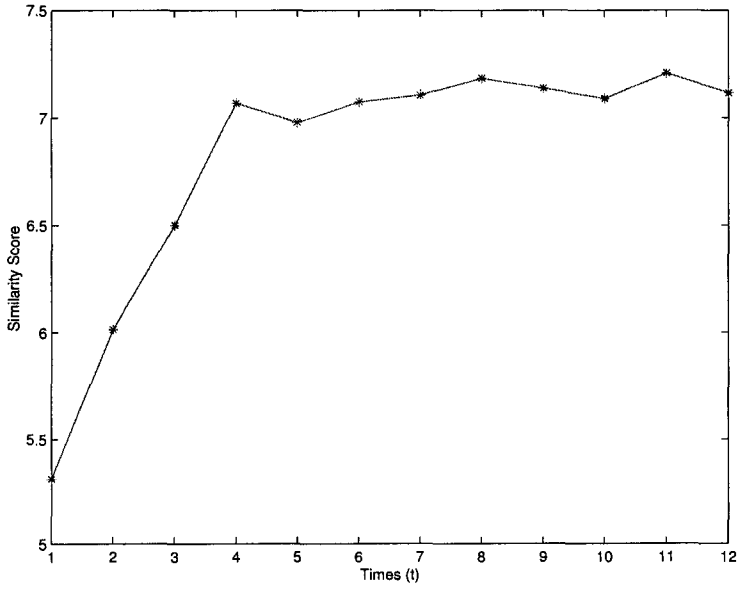


Figure 3.9: The similarity score for updating a HHMM L-System model in first 12 loops.

Chapter 4

Visual Hull Construction

In this chapter, we first introduce the concept of the visual hull and its properties and then compare four approaches to constructing visual hulls.

4.1 Introduction

A central problem in computer vision is to identify and reconstruct the 3D content of a scene. Many approaches are based on using 2D images as sources and they are classified as "shape from X", where "X" is the information used, such as textures, shadows. [21]. A 2D silhouette, the projection of a 3D object onto an image plane, is effective for shape understanding [1] and is relatively simple to compute. When the silhouettes of a 3D object are used, the approach is named *shape from silhouettes* (SFS) and the resultant 3D object is called the *visual hull* of the 3D object and the concept of visual hulls was first introduced by Laurentini [21].

Visual hulls created by using SFS methods are the maximal silhouette-equivalent to the original object. In other words, a visual hull has the same silhouettes as the original object. Compared to the corresponding convex hull, a visual hull is a more accurate approximation. A visual hull depends on both the shape of the object and the visible region from the viewpoints [21]. Only the parts of the surface of the original object that also lie on the surface of the visual hull can be reconstructed. Unfortunately, a visual hull is not guaranteed to be identical to the original object due to the lack of complete information from the silhouettes including specific concave sub-surfaces which

cannot be viewed and in practice only a small number of views are used to construct the visual hull. Like all contour-based approaches, it has limited use for understanding non-convex shapes [21]. Laurentini [21, 22] discussed how far a 3D object can be understood by its 2D silhouettes.

4.2 Visual Hull Construction

The implementation of SFS methods is relatively straightforward and can be described as follows [10].

Suppose an object O is surrounded by k pinhole cameras. Each reference view r has a silhouette S_r where all pixels inside the silhouette belong to the object. From each view r, we define a cone-like volume V_r, whose peak is at the current projection point, *i.e.*, the camera center, and it passes through all interior points of the silhouette on the image plane (see Fig. 4.1). The object then lies inside the volume outlined by all silhouettes $VH = \cap_{r \in k} V_r$. When the number of views increases, VH converges to the visual hull of the original object.

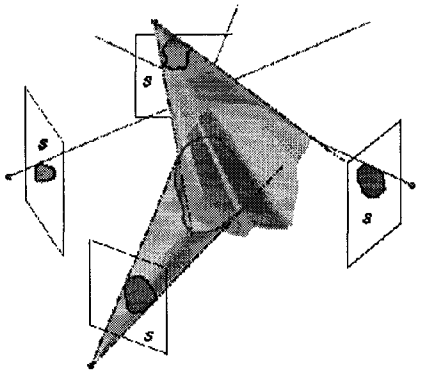


Figure 4.1: An illustration of the bounding cone intersection modeling (from Matusik [26]) for visual hulls.

Many algorithms have been proposed in the literature to compute visual hulls, and can be roughly classified into four approaches:

- Voxel-based approaches
- Surface-based approaches

- Hybrid approaches
- Image-based approaches

4.2.1 Voxel-based approaches

Voxel-based approaches, also named *volumetric approaches*, use a technique called "voxel carving" to compute visual hulls. The underlying idea is to divide the space of interest into discrete voxels and to classify them into two classes: *inside* and *outside*. Each voxel to be classified is projected onto each image plane. If all projections on k image planes are completely inside the silhouettes, then this voxel is classified as inside; otherwise it is outside. Outside voxels are removed from the voxel space [10] and remained inside voxels are considered as belonging to the visual hull.

Voxel-based visual hull construction is a robust method and is able to handle objects with complex geometries. It can be easily implemented giving reasonable results regardless of the format of the silhouettes [10]. Obviously, the more voxels used, the smaller each voxel is, and the more precise the visual hull will be. However, a large amount of memory is required for computing a precise visual hull. For a volume of $N \times N \times N$ voxels, N^3k memory units are required for generation and testing, which makes the algorithm slow and expensive.

Octree-hierarchies [9] are often used to accelerate the speed of the computing procedures for voxel-based visual hull construction. Some effective approaches [10] make it possible for real time visual hull construction. Sullivan *et al.* [46] introduced a method to improve visual hull shapes by using splines. Seitz *et al.* [43] made use of color consistency information for carving voxels.

4.2.2 Surface-based Approaches

Surface-based approaches construct the surfaces of a visual hull and provide an explicit 3D model. The most direct way to construct the visual hull of an object from a set of silhouette images is by intersecting bounding cones: *bounding cone intersection.* A bounding cone is formed from the edges of

the silhouette and the camera center, as shown in Fig. 4.1. Objects inside a bounding cone can be seen from the corresponding camera, and the faces of such cones define the bounds of object areas and none-object areas. The original object must lie inside all the bounding cones and the faces of bounding cones are also on the object surface. Thus, a visual hull can be computed by intersecting all bounding cones and the resulting visual hull consists of a set of surface facets, which are on the surfaces of the bounding cones. In this way the visual hull gives a polyhedral approximation to the object.

Surface-based approaches can be very precise and they give explicit 3D models which make them easy to be used in applications such as texture mapping and rendering. However, it is not easy to express the visual hull facets effectively by simple geometrical attributes. The computational complexity and the numerical instability of intersecting surfaces with lines and planes in 3D make them impractical for computing visual hulls of complex objects [10].

An early contribution on surface-based approaches was made by Baumgart [4], who used polygonal approximations of the occluding contours. Other researchers [20, 14, 11] also used local second order surface approximations to reconstruct individual points [8]. Some effective algorithms [8] have been introduced by compute surface patches [46] or surface strips [25].

4.2.3 Hybrid Approaches

Recently, Boyer *et al.* [8] introduced a new approach which takes advantage of both the robustness of voxel-based approaches and the precision of surface-based approaches. It uses voxels but only the voxels on the surface of the visual hull are computed. It then extracts the visual hull surface from a Delaunay triangulation by taking the surface delimiting the polyhedra that project inside the silhouettes. This hybrid approach has equivalent efficiency to voxel-based approaches and gives more precise visual hulls together with lower time and space complexities [8]. For detailed description, please refer to [8].

4.2.4 Image-Based Approaches

Traditional visual hull construction algorithms are geometry based since they all compute 3D geometrical representations. Nonetheless, if the purpose of an SFS application is to render new images from different views of an object, then an explicit 3D geometry representation is not necessary. Matusik *et al.* [26] proposed an *image-based visual hull* (IBVH) algorithm which is a practical alternative to the traditional modeling/rendering framework.

In this method, views of a visual hull can be directly rendered from its silhouette images without constructing a volumetric or polyhedral model by using view interpolation. This is accomplished by merging the cone intersection calculation with the rendering process. It is efficient for applications that do not require an explicit geometric or volumetric representation.

4.3 Experiments: Visual Hull Construction Using Bounding Cone Intersections

In our study, we are required to infer 3D tree models from images so we use one of the geometric modeling methods: the surface-based approach and we have applied the following steps to construct polyhedral visual hulls.

Image Acquisition and Silhouette Extraction. In this study, images came from two sources: rendered synthetic 3D digital objects images and real photos. For the first source, we first render a virtual 3D tree by assigning different camera setting parameters to collect a set of images. In this way, all parameters including camera position, orientation, etc., were accurately set. We could also eliminate all complex background and get a object without any background noise. Moreover after we obtained the visual hulls, we could compare them with the original 3D objects at all view points, so it is an effective source for testing and estimating a visual hull construction algorithm.

The second source was a set of photos of real trees taken by digital cameras. and required more efforts. Experimental errors from facilities, human operations, and background noise, cannot be easily eliminated. Once images

were obtained, the objects of interest were segmented from its background manually and the silhouettes were extracted by encoding the object boundary in a polygon format, with a silhouette being denoted by S_i.

Compute visual hulls Suppose there are k reference views and associated images taken from a calibrated pinhole camera. For each reference view a local coordinate system was built which included the camera's position as point of origin, the optical axis as the z axis, the horizontal direction of the camera as the x axis, and the vertical direction as the y axis. Notice that the camera coordinate system was left-handed.

For each such local camera coordinate system an image plane P_i was perpendicular to the z axis passing through the camera point of focus. The image plane could also be represented by $z = f$, where f was the focal length of the camera. Therefore, each scene view was projected to their own image planes. For each such silhouette, a bounding cone was then computed relative to its specific camera model and it consisted of a bundle of lines which all originated from the camera center and passed through all the silhouette edges on the image plane. Since a silhouette was in polygon format, its bounding cone V_r was then represented by a set of triangular faces which converged at the camera center. The visual hulls could be computed by finding the intersection of all the bounding cones, $VH = \cap_{r \in R} V_r$. Moreover, all the polygons in the resulting visual hulls were on the bounding cone triangular planes.

To compute the visual hull, every intersection point between the faces of the bounding cones must be computed. To overcome the complexity of this operation in 3D, Matusik *et al.* [25] proposed, instead of computing the intersection of faces in 3D, to project bounding cones V_i to the images plane P_j and to find the intersection part of the projected bounding cone v_i with the S_j silhouette, for all views i, j.

Upon the completion of the intersection process, a series of polygons were obtained. These polygons composed the surfaces of the visual hull. The pseudo-code for the above algorithm is given in the following Table 4.1, after we define some notations.

- O_r: the rth camera center;
- k: the number of reference views;
- r: the index of reference views;
- S_r: the rth silhouette;
- E_i: a vertex on S_r;
- $E_{i,i+1}$: an edge on S_r;
- $S_r = E_1E_2E_3\ldots$;
- $T_i = E_iO_rE_{i+1}$ is an open triangular plane consisting of two lines which meet at the rth camera center.
- $v_r = \{T_i\} = T_1T_2T_3\ldots$: a bounding cone consists of a number of triangular planes, which share a common peak point O_r;
- P_i: the projection of T_i on a image plane.

$VH = v_1$
for $r = 1, 2, \ldots, k$
    for $i = 1, 2, \ldots, k$ and $i \neq r$
        for each edge $E_{j,j+1}$ of silhouette S_r
            $T = E_jO_rE_{j+1}$
            compute P by projecting T onto the ith image plane
            compute p by intersecting S_i and P
            compute p_T by projecting p back onto plane T
            update VH by merging p_T into VH
        end
    end
end

Table 4.1: The pseudo-code for the visual hull construction algorithm.

Fig. 4.2 shows an example visual hull on the second row created from the four input images on the first row. Objects on the same column are viewed at the same view points.

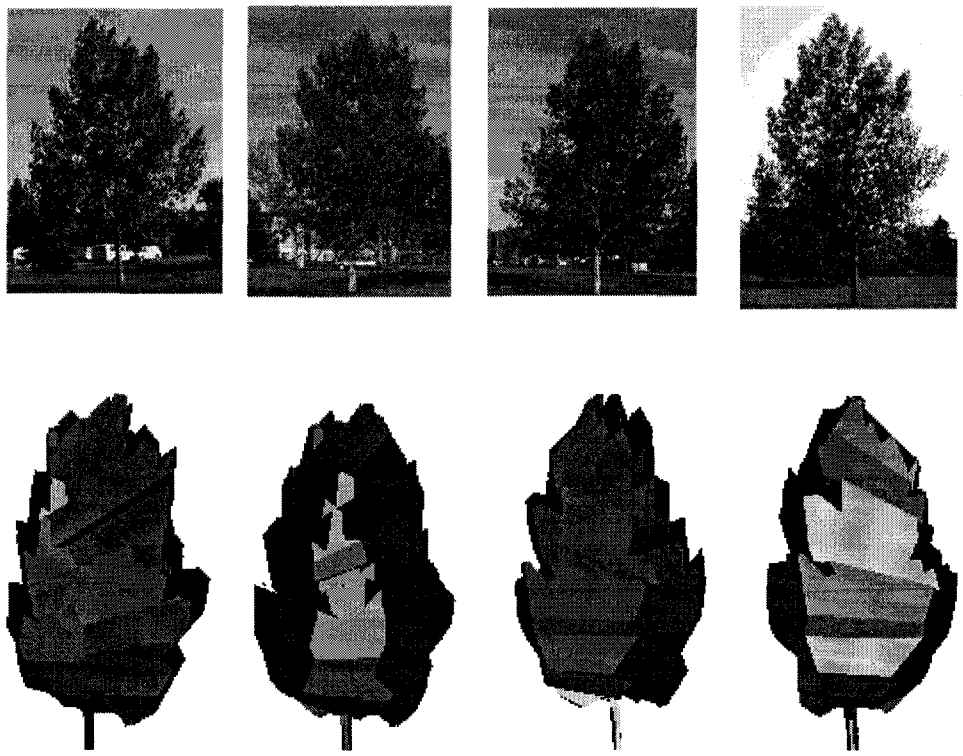


Figure 4.2: Four different views of a visual hull created from the four different views of an aspen correspondingly.

Chapter 5

Extracting Skeletons from Visual Hulls

Visual hulls do not provide an "object-centered" model and so for more accurate and robust 3D models we need to integrate full 3D model structures with a given visual hull. To generate a 3D tree model, we need to find its skeleton from its visual hull. In such a case, tree skeletons refer to the trunks and the main large branches of a tree that we assume determine the predominant view of the tree shape. In our study, we assume that the *Medial Axis* of a visual hull approximates the tree skeleton.

5.1 Medial Axis Transform

The medial axis of an object is the locus of centers of a sequence of circles, or spheres in 3D space, and these circles and spheres have maximal radii within the object. The *Medial Axis Transform* (MAT) is the medial axis together with the associated radius [44]. The MAT encodes important visual cues such as local diameters and symmetries. The concept of MAT was first introduced by Blum *et al.* [5] and further developed by Blum *et al.* to apply to biological shapes [6, 7]. A 2D example illustrates a rectangle and its MAT in Fig. 5.1

In general, there are two approaches to compute the MAT of an object. The first approach is to use a type of morphological thinning that successively erodes away pixels from the boundary and preserves the end points of line segments at the same time until no more thinning is possible. After the thinning

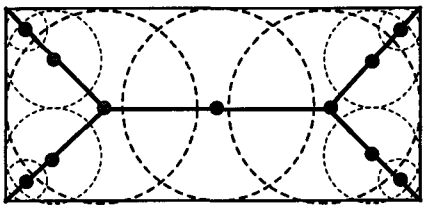


Figure 5.1: The medial axis of a rectangle is shown as solid lines inside the rectangle. The maximal circles in dash lines, and the circle centers are also shown.

process, the remaining pixels approximate the medial axis. The alternative approach is to first calculate the distance transform of the image. The skeleton then lies along the singularities, *i.e.*, creases or curvature discontinuities in the distance transform [44].

5.1.1 Voronoi Diagram and Delaunay Diagram

Before we go through the detailed MAT algorithm, we introduce two terms: *Voronoi diagram* and *Delaunay diagram.*

The definition of a 2D Voronoi diagram is given as follows:

- Let P be a set of n distinct points on a plane. These points are called sites.
- The Voronoi diagram of P is the subdivision of the plane into n cells or subdivisions and each cell includes one and only one site.
- Any point q lies in the cell corresponding to a site $p_i \in P$ if and only if for each $p_j \in P$ $(j \neq i)$ the Euclidean distance between q and p_i is less than the Euclidean distance between q and p_j.

Some example illustrations of 2D Voronoi diagrams are shown in Fig.5.2. The Voronoi diagram for one site is the plane itself (see Fig.5.2(a)). The Voronoi diagram for two sites is a line that extends infinitely in both directions and the two half planes on the sides. This line is the perpendicular bisector of the two sites (see Fig.5.2(b)). The Voronoi diagram for n sites that are collinear is a series of parallel lines that extends infinitely in both directions. Each line is

the perpendicular bisector of each two neighboring sites (see Fig.5.2(c)). The Voronoi diagram for n sites that are non-collinear is half lines that meet at vertices. Each line is the perpendicular bisector of two sites and the *Voronoi vertices* are the centers of empty circles with three or more sites on their boundaries. Empty circles mean that no more sites are inside of them (see Fig.5.2(d)).

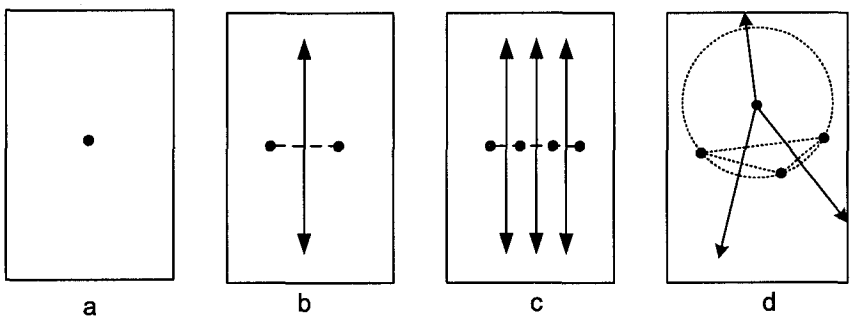


Figure 5.2: Voronoi diagrams for different set of sites on a plane, a. the Voronoi diagram of a single site is the plane itself b. The Voronoi diagram for two sites is the bisector of these two sites. c. The Voronoi diagram for collinear n sites is a series of bisectors of each two neighboring sites. d. The Voronoi diagram for non-collinear n sites is a series of bisectors of sites on the edges of empty circles. The circle centers are called Voronoi vertices.

A *Delaunay Diagram* is the dual structure of the Voronoi diagram, see Fig 5.3. Each site is a Delaunay vertex and two vertices are connected if their associated cells in the Voronoi diagram share a common boundary edge.

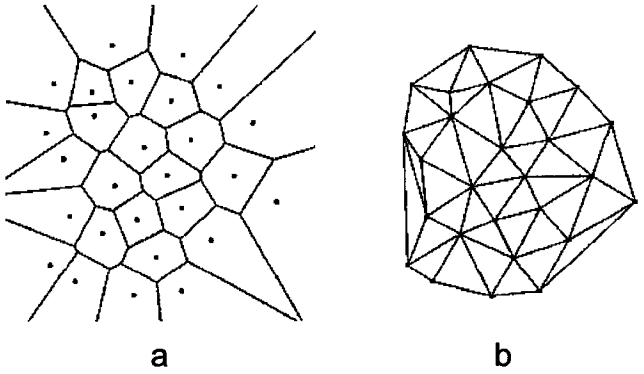


Figure 5.3: The Voronoi diagram (a) and Delaunay diagram (b) of a same set of points.

5.2 Computing the MAT

In this study, we use the algorithm proposed by Hubbard [31], which approximates the medial axis of a polyhedron. To compute a better approximation of the medial axis, we require dense mesh vertices. The density of mesh vertices is closely related to the precision of the resultant medial axis and the computation time. In our application, we set a distance threshold and then measure the distance among vertices in each polygon. If the measured distances of a polygon are larger than the threshold, more points will be added to the interior of the polygon. The distance between any pair of points including vertices of the polygon must be less than the threshold. Based on these 3D points, we first compute the Delaunay diagram consisting of a number of tetrahedra. Then for each tetrahedron, we compute its circumsphere. Both the centers and the radii of the circumspheres are kept for further use. As mentioned before, Delaunay triangulation is the dual structure of Voronoi diagram and the centers of circumspheres are the Voronoi vertices. In addition, for every tetrahedron we can find the neighboring tetrahedra that share one face with the current one. Once we determine that two tetrahedra are neighbors, an edge is added to link their Voronoi vertices. Upon completion we have a Voronoi diagram consisting of Voronoi vertices and Voronoi edges.

We examine all the edges in the 3D Voronoi diagram. If an edge completely lies inside of the polyhedron, which is the visual hull in our application, it is remained. Otherwise, this edge will be removed from the Voronoi diagram.

Teichmann *et al.* [28] introduced a skeleton simplification method which allows the use to specified Voronoi vertices that are the closest to the "ends" of each branch. These vertices will be kept when simplifying the Voronoi diagram. The Voronoi diagram also includes bi-connected components [1]. Our task is to create a tree structure so no such bi-connected components are allowed. To remove bi-connected components, a skeleton simplification method is used as follows:

[1] A bi-connected component is one that when one edge of the component is removed, the component is still connected.

1. Compute the bi-connected components and remove the vertex which is the closet to the mesh surface. The distances from the vertex to the surfaces of the polyhedron can be set to the radii of the circumsphere approximately.

2. All the articulation points [2] and the user specified vertices are not allowed to be removed.

3. Repeat the above 2 steps, until the graph is a tree or no more points are removed.

Shlyakhter *et al.* [45] extended the above algorithm by developing a method to automatically find a set of Voronoi vertices which match the tips of major branches. These vertices are obtained by finding "interesting" vertices in 2D silhouettes by the following steps.

1. "Interesting" vertices are selected from the convex hull and the even-order convex hulls of silhouettes. Such vertices are chosen by some heuristic, for example, vertices where the convex hull makes a shape angle or vertices adjacent to a long edge.

2. Once these interesting 2D points are located, their corresponding 3D points can be obtained by finding the Voronoi vertices which are the closest to these 3D points and are marked as "un-removable" for the skeleton simplification algorithm.

Using this simple algorithm, no user interaction is required and the interesting points for capturing the significant features of the objects can be found.

5.3 Post-processing

The obtained medial axes can be represented in graph form: $G = \{V, E\}$, where V are a set of 3D points, and E are the edges connecting points. The

[2] An articulation point is one which will cause the diagram fall into two parts when it and its incident edges are removed.

medial axes were computed simply using a mathematical method and didn't have any botanic meanings. To convert the graphs into tree structures, we used a simple heuristic method to classify all points into different axis orders, where the order of axis has been introduced in Chapter 2 (on page 8). After this classification all order one points belong to the trunk of the target tree, and order i axes consist of all the order i points. A point belonging to order $i-1$ connects to the order i axis.

The classification was done as follows. We first found the root point by its coordinate since the root point is on the ground or the lowest point. Then starting from the root point, we searched for a longest path. For monopodial trunks without ramifications, these paths are very close to the expected trunks. A dichotomous trunk ramifies into several order two axes and so the found longest path consists of the trunk and a order two axis. In this case, we keep the partial points of the path which locate at the lower part of the path and their projections on the ground are close enough to the root point. By this method, order one axes, or the trunks, were extracted. We then started from each growth point on the trunk and found the longest path starting from it. During the new search procedure all classified points were removed from consideration. We regarded the points on the new longest paths as order two axes. By recursively repeating these steps all higher order axes were found.

As a limitation of the MAT method, a shape's boundary noise can induce several small branches or spurs on its MAT, even though the noise is minor and does not significantly contribute to the overall structure (see Fig. 5.4). Moreover, the MAT is maximally axial to the shape, *i.e.*, it provides the local axis of symmetry of the shape everywhere. Nonetheless, in general the growth of a natural tree is fairly symmetric along its trunk at all direction yet its symmetry is not strict. In reality, an individual tree is never perfectly symmetric in all directions at all heights. So the tree skeletons extracted using this MAT algorithm are different from the botanical trunks as we expected. Indeed, sometimes the extracted skeletons have too many undesirable artifacts which make them unnatural tree axes. Therefore, it is necessary to use some post-process to eliminate these artifacts.

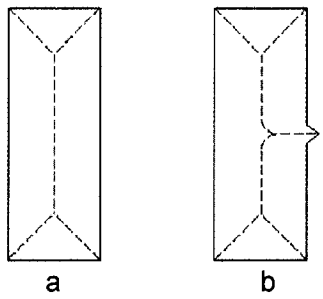


Figure 5.4: Skeletons constructed from 2 objects: a) the medial axis of a symmetric object; b) the medial axis of a non-symmetric object and the asymmetry is caused by minor noise.

In our experiments, we used the B-Spline curve fitting method to smooth all the axes since B-Spline curves have an advantage over Bzier curves in that they are smoother and easier to control. Furthermore, we assumed that the trunk of some species trees are straight, We then derived the best fitting straight trunks from their curve shape trunks.

As we can see in Fig.5.5, the first medial axis includes zigzag axes. After spline fitting all the axes appear more smooth and look more natural.

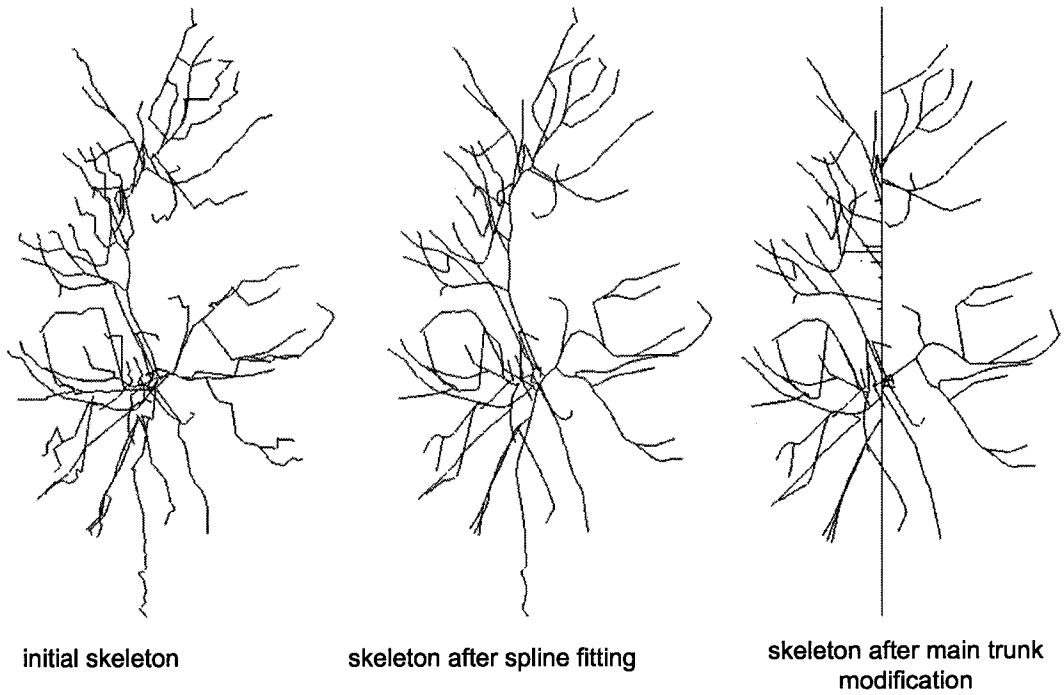


Figure 5.5: Skeletons constructed from the visual hull shown in Fig. 4.2. The left skeleton, initial skeleton, is created from Medial Axis extraction method. The middle skeleton is created by applying B-Spline fitting on the initial skeleton The right skeleton is created by replacing the middle skeleton's trunk with a straight vertical trunk.

Chapter 6

Fitting Branches inside Visual Hulls

The skeletons created from visual hulls include trunks and main branches. To create more realistic trees, we also need to create small branches and leaves. The HHMM L-System models created for the same tree type can be used to fulfill the task.

Basically, the process starts from the root of the skeleton and traverses all branches. During the process, new branches are added if the distance between two growth points, *i.e.* the distance between two neighboring points, is greater than a specified threshold branches will be added. All branches are fitted within the visual hull.

6.1 Previous work

Previous researchers proposed some methods for generating branches inside of visual hulls. Sakaguchi *et al.* [42] simulated branch structure by applying simple branching rules with some restrictions. They first generated a number of temporary branches inside of volume data of input tree images, where the volume data are very similar to visual hulls. These branches were generated at different orientations, and the temporary branches, that occupied the most space, and met the predefined restrictions, were kept. Using this method, a large number of temporary branches were created during the fitting process and the generated branches used were in simple form and not specialized to

make a botanical tree, thus the process is slow and the resulting trees are not natural enough.

Shlyakhter *et al.* [45] extracted the tree skeleton from visual hulls and wrote the tree skeleton as an axiom to an L-System. Buds were appended on the last two levels of branches. They then applied open L-System models of the tree type for tree growth process. They manually decided the recursive depth of each open L-System to make the generated branches fit the tree shape the best. Using this method, interaction between human and computer is required.

In our work, we proposed a fully automatic process to do the task. The fitting process is described on the following sections.

6.2 Converting MAT Skeletons into L-Strings

Branches created by HHMM L-System models are represented in the L-String format. The skeletons are represented in a common graph format $G = \{V, E\}$ with order information. We first unify them by converting the MAT skeletons into the L-Strings format. L-System symbols "&", "∧", "+", "−", "F", "[", "]", ..., will be used in this conversion.

A skeleton consists of a set of axes and each axis consists of numerous connected line segments. For example, an order i axis consists of a set of connected line segments: $\{\overrightarrow{p_0}\ \overrightarrow{p_1}\ \ldots,\overrightarrow{p_n}\}$, where $\overrightarrow{p_i} = (x_{p_i}, y_{p_i}, z_{p_i})$, and each point is connected to its neighbors, *i.e.* point $\overrightarrow{p_0}$ is connected to point $\overrightarrow{p_1}$, which in turn is connected to point $\overrightarrow{p_2}$, and in general, point $\overrightarrow{p_i}$ is connected to point $\overrightarrow{p}_{i+1}$.

We initialized a standard coordinate system with three unit orthogonal axes $\overrightarrow{X}$, $\overrightarrow{Y}$, and $\overrightarrow{Z}$, which correspond to the "left", "up", and "forward" directions, respectively. We regarded this coordinate system as the standard coordinate with the original point placed on the root position of the skeleton. We assumed that there is a "turtle" coordinate system which is initially the same as the world system.

To represent a sequence of joint line segments in the L-Strings form, we first rotate the turtle coordinate to align its "forward" and the first line segment

directions Then move the turtle coordinate forward the length of the first line segment. After that, rotate the turtle coordinate to align its "forward" and the second line segment direction, and then move forward the length of the second line segment. This process is repeated for all line segments.

To rotate the coordinate and align its "forward" axis, or Z axis,and the line direction $\overrightarrow{p_{i+1}p_i}$. we first transformed the original turtle coordinate $(\overrightarrow{X}, \overrightarrow{Y}, \overrightarrow{Z})$ to the target coordinate system $(\overrightarrow{X'}, \overrightarrow{Y'}, \overrightarrow{Z'})$. In the target system, only $\overrightarrow{Z'}$ is known to be $\frac{\overrightarrow{p_{i+1}p_i}}{|\overrightarrow{p_{i+1}p_i}|}$, where $|\overrightarrow{p_{i+1}p_i}|$ is the norm (or the length) of $\overrightarrow{p_{i+1}p_i}$.

We assumed that no rotation around the $\overrightarrow{Z}$ axis was required. Based on this assumption, we first rotated the coordinate system around its $\overrightarrow{Y}$ axis for α degrees to align $\overrightarrow{X}$ and $\overrightarrow{X'}$. Using L-System symbols, we can denote this rotation process as $\&(\alpha)$ when $\alpha > 0$ and $\wedge(|\alpha|)$ when $\alpha < 0$, where function $|X|$ returns the absolute value of X. After this is done, we rotated the new coordinate system around $\overrightarrow{X'}$ axis for β degree, and aligned $\overrightarrow{Y}$ and $\overrightarrow{Y'}$, $\overrightarrow{Z}$ and $\overrightarrow{Z'}$, respectively. Using L-System symbols, it could be denoted as "$+(\beta)$" if $\beta > 0$ or "$-(|\beta|)$" if $\beta < 0$. After these two rotation steps, we translated the turtle coordinate system to the end point of the current line segment and measure the distance of the translation d. This movement using L-System symbols is denoted as "F(d)".

We repeated these processes to present the whole sequence of line segments in the L-String format. When a point which connects more than one points was encountered, we first processed children axes one by one. To do so, the current states were first saved in a stack and then used the same method to process a child axis. When the axis was done, the saved states were popped up and were used to process the next axis. When all children axes were processed, we popped the states from the stack and then continued to process the rest part.

We used brackets "[" and "]" to push and pop the states into and out of stacks:

$$L_i[L_{i+1}[L_{i+2}]L_{i+1}][L_{i+1}[L_{i+2}\ldots]L_{i+1}]L_i$$

where L_i is a substring of an order i axis L-String. Taper functions were

used to modify the thickness of trunks at different height.

An example of a resultant trunk L-System string is given as below (only a small part of the string is shown):
F(12) [&(145)−(52)F(17) ∧(23)−(7)F(1) [&(82)+(15)F(10) ∧(18)−(12)F(3) ∧(4)−(3)F(9)∧(4)−(5)F(2) ∧(11)−(7)F(3) ∧(4)F(4) &(10)+(10)F(15)]]F(2)

6.3 Adding Branches and Leaves

During the above converting procedure, we examined the length of each line segment of the medial axis. If the length was greater than a threshold, it was divided into several short line segments and each is shorter than the threshold. The distance threshold was varying at different positions of the skeleton. The threshold decreased when either the height or the order of axes increased to make the internode at the bottom of a tree longer than the top of the tree.

At each line segment joint point we inserted an axiom to create a stochastic sub-branch with a selected probability. We used the extracted medial axis as the main skeleton of the tree, so we don't used axioms to generate trunks. At a growth point on the first order axis of the skeleton, we added an axiom with recursive depth n which can generate second order branches. At a growth point on the ith order axis of the skeleton, we added an axiom with recursive depth $n - i + 1$ which can generate $i + 1$ order branches. The ith order axes and $(i + 1)$th order axes might share the same HHMM L-System axiom, and recursive depths are the only difference. At each growth point, an axiom can generated more than one branches, and the number of branches is decided by the tree species and is designed in the axiom.

For example, if we use the HHMM L-System model defined in Fig. 3.5 (Chapter 3) as the tree model, then "M" can be appended on the first order axis to generate its sub-branches, which are lateral branches on the trunk. And "B" can be appended on the second or higher order axes to generate their sub-branches.

Once an axiom was inserted, the HHMM L-System commenced rewriting

to expand the axiom to an L-String for a branch. We used the turtle interpretation to convert the L-String into a 3D geometrical structure. We compared the 3D branch structure with the visual hull to test if the branch is inside the visual hull.

We noted that the 3D structure was represented in the turtle coordinate system. Therefore, before comparison, we needed to transform the 3D structure into the standard coordinate system. For example, an axiom was inserted at $\overrightarrow{p} = [X_p, Y_p, Z_p]$ on the line segment and the turtle interpretation started at p. In the resulting 3D geometry, $\overrightarrow{p}$ would have its new coordinate in the turtle coordinate system as $\overrightarrow{p} = [0, 0, 0]$, which was inconsistent.

We computed a transform matrix to transform the turtle coordinate system back into the standard coordinate system. Using the transformation matrix, we transformed the 3D branch structure back into the standard coordinate system and compared the whole branch with the visual hull.

On the standard coordinate system, to test if a point $\overrightarrow{p}$ was inside a visual hull or not, we projected it onto k 2D image planes each of which contains a silhouette, and compare the projected 2D points with all 2D silhouettes. If all projected points were inside its respective silhouettes, then we concluded that the point was inside the visual hull. This method avoids complex 3D computations since an important property of a visual hull is that it is equivalent to the silhouettes of the object [21].

To meet the optimum criterion that the branch was completely inside the visual hull and at the same time, occupied as much space as possible, we started a recursive process, at each step of which we measured all points on the branch. If all the points were inside the visual hull, a scalar was increased to enlarge the branch; otherwise the scalar was decreased. This recursive process continued until the scalar difference between two adjacent steps reached a minimum threshold. Once a branch associated with its scalar was accepted, its L-String included in two brackets replaced the axiom.

After traversed all the axes on the skeleton, we stopped the process.

Four sets of experimental results are reported on the next chapter.

Chapter 7

Experimental Results

We have performed experiments on both synthesized and natural trees. Some results are shown below.

7.1 A Synthesized Tree 1

Four input images a synthesized tree are shown in the first row in Fig 7.1. The generated visual hull and 3D tree viewed at the four corresponding view angles are shown on the second and third row respectively in Fig 7.1.

7.2 A Synthesized Tree 2

Four input images which contain a synthesized tree are shown in the first row in Fig 7.2. The generated visual hull and 3D tree viewed at the four corresponding view angles are shown on the second and third row respectively in Fig 7.2.

7.3 A Natural Aspen

Four input images which contain an aspen are shown in the first row in Fig 7.3. The generated visual hull and 3D tree viewed at the four corresponding view angles are shown on the second and third row respectively in Fig 7.3.

Figure 7.1: Four different views of a tree which has a dichotomous trunk are on the first row,. The visual hull and reconstructed 3D tree are shown on the second and third row.

7.4 A Natural Spruce

Four input images which contain an spruce are shown in the first row in Fig 7.4. The generated visual hull and 3D tree viewed at the four corresponding view angles are shown on the second and third row respectively in Fig 7.4.

Figure 7.2: Four different views of a tree which has a monopodial trunk are on the first row. The visual hull and reconstructed 3D tree are on the second and third row.

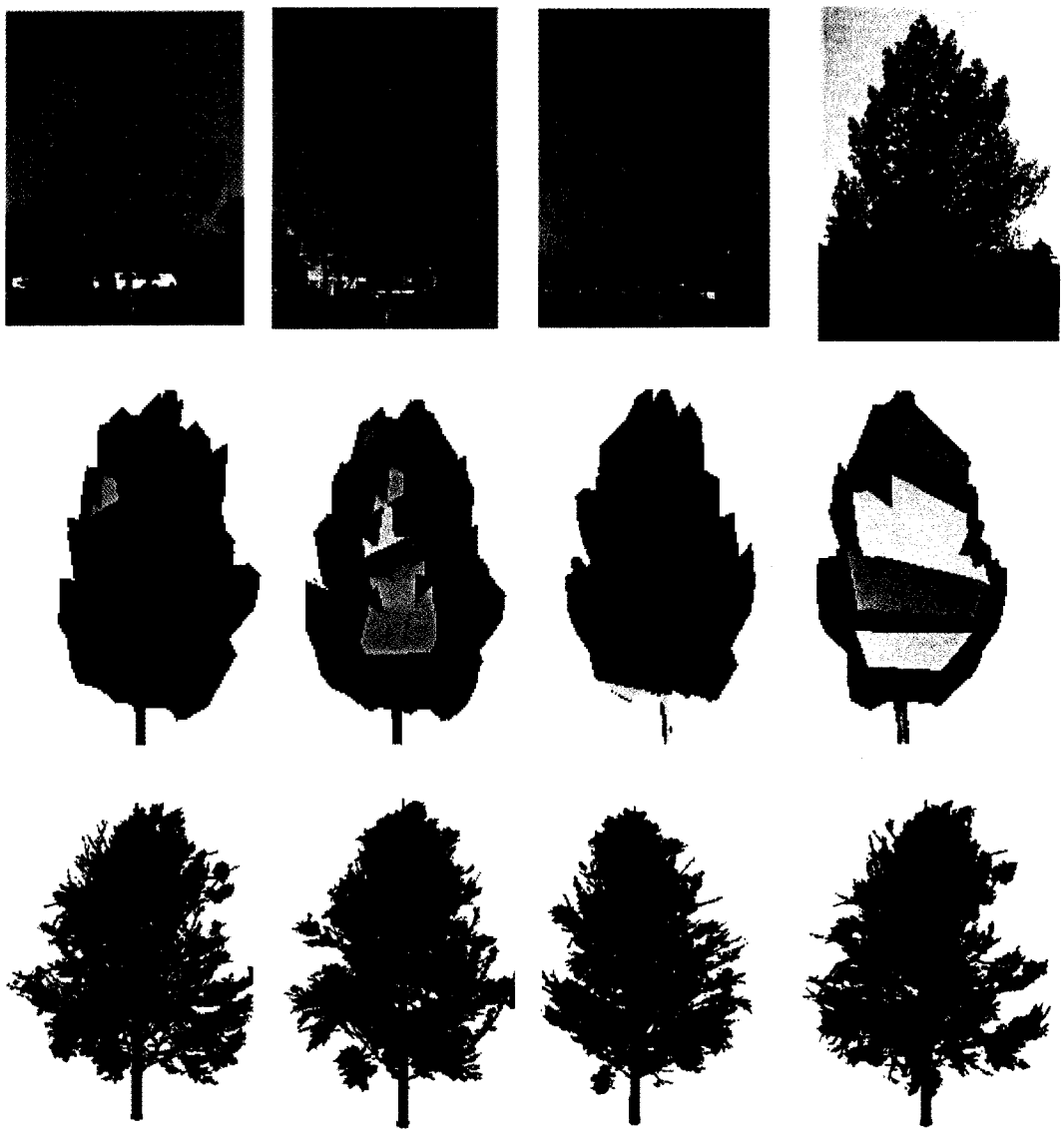


Figure 7.3: Four different views of a real aspen are shown on the first row. The visual hull and reconstructed 3D tree are shown on the second and third row.

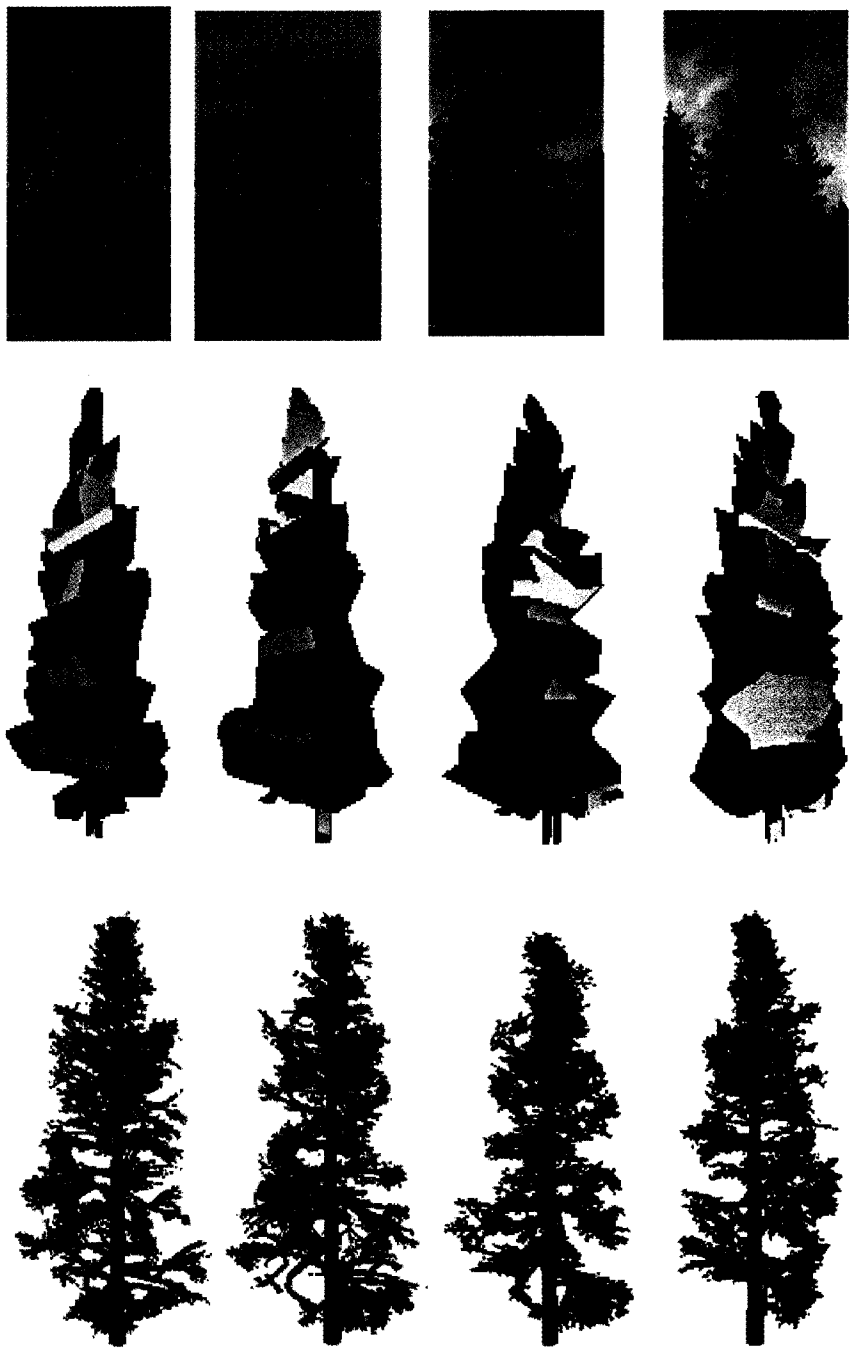


Figure 7.4: Four different views of a real spruce are shown on the first row. The visual hull and reconstructed 3D tree are shown on the second and third row.

Chapter 8

Conclusions

This is an example of how computer vision, and machine learning, can be integrated with stochastic process to create new types of computer graphics models.

A new type of stochastic L-System, HHMM L-Systems, were developed for creating trees with stochastic appearances. This HHMM L-System inherits the properties of standard stochastic L-Systems, but has a better hierarchical structure, and provides more parameters to admit stochastic behavior of tree growth. The stochastic EM algorithm is used to train the partial parameters of the HHMM L-Systems.

We created a framework to generate 3D trees from 2D images. In our system model, immediately after the tree images are manually segmented, the intermediate stages such as visual hull generation, medial axes creation, and branches fitting process are all automatically done. The new fitting algorithm was designed to automatically add adequate branches and leaves within the visual hull. Our work distinguishes from existing work through a number of automated processes. The automation is potentially valuable to those applications that required automated reconstruction of trees that fit within visual hulls.

Bibliography

[1] J. Aloimonos. Visual shape computation. *IEEE proc.*, 76(8):899–916, 1988.

[2] M. Aono and T. L. Kunji. Botanical tree image generateion. *IEEE Computter Graphics and Applications*, 4(5):10–34, 1984.

[3] M. Barnsley and S. Demko. Iterated function systems and global construction of fractals. *R Soc London Ser*, A399:243–275, 1985.

[4] B.G.Baumgart. A polyhedron reprsentation for computer vison. *AFIPS National Computer Conference*, 1975.

[5] H. Blum. A transformation for extracting new descriptors of shape. In Wathen-Dunn W, editor, *Models for the Perception of Speech and Visual Form*, pages 362–381. MIT Press, 1967.

[6] H. Blum. Biological shape and visual science (part I). *Journal of Theoretical Biology*, 38:205–287, 1973.

[7] H. Blum and R.N. Nagel. Shape description using weighted symmetric axis features. *Pattern Recognition*, 10(3):167–180, 1978.

[8] E. Boyer and J-S Franco. A hybrid approach for computing visual hulls of complex objects. In *Computer Vision and Pattern Recognition*, pages 695–701, June 2003. Madison, Wisconsin, USA.

[9] C.H.Chien and J.K.Aggarwal. Volume/surface octress for the reprsentation of three-dimensional objects. *CVGIP*, 36(1):100–113, 1992.

[10] G. Cheung, S. Baker, and T. Kanade. Shape-from-silhouette of articulated objects and its use for human body kinematics estimation and motion capture. *CVPR03*, 1:77–84, 2003.

[11] R. Cipolla and A. Blake. Surface shape from the deformation of apparent contours. *IJCV*, 9:83–112, 1992.

[12] D. Cohen. Computer simulation of biological pattern generation processes. *Nature*, 216:246–248, 1967.

[13] P. de Reffye, C. Edelin, J. Francon, M. Jaeger, and C. Puech. Plant models faithful to botanical structure and development. *Computer Graphics, SigGraph*, 22(4):151–158, August 1988.

[14] E.Boyer and M.-O.Berger. 3d surface reconstruction using occluding contours. *IJCV*, 22(3):219–233, 1997.

[15] S. Fine, Y. Singer, and N. Tishby. The hierarchical hidden markov model: Analysis and applications. *Machine Learning*, 32(1):41–62, 1998.

[16] C. Godin and Y. Caraglio. A multiscale model of plant topological structures. *Journal of Theoretical Biology*, 191:1–46, 1998.

[17] Halle, Oldeman, and Tomlinson. *Tropical Trees and Forest: an Architectural Analysis.* Springer-Verlag, 1978.

[18] H. Honda. Description of the form of trees by the parameters of the tree-like body: Effects of the branching angle and the branch length of the shape of the tree-like body. *Journal of Theoretical Biology*, pages 331–338, 1971.

[19] B. Hu, X. Zhao, H. Yan, Ph. de Reffye, F. Blaise, F. Xiong, and Y. Wang. Plant growth modeling and visualization – review and perspective. *ACTA Automatica Sinica*, 27(6), November 2001.

[20] J.J. Koenderink. What does the occluding contour tell us about solid shape? *Perception*, 13:321–330, 1984.

[21] A. Laurentini. The visual hull concept for silhouette-based image understanding. *IEEE Trans.Pattern Anal. Machine Intell.*, 16, February 1994.

[22] A. Laurentini. How far 3d shapes can be understood from 2d silhouettes. *IEEE Trans.Pattern Anal. Machine Intell.*, 17:188–195, February 1995.

[23] A. Lindenmayer. Mathematical models for cellular interactions in development: parts I and II. *Journal of Theoretical Biology*, 18, 1968.

[24] A. Lindenmayer. *Developmental order: Its origin and regulation*, chapter Developmental algorithms: Lineage versus interactive control mechanisms, pages 219–245. In S. Subtelny and P. B. Green, 1982. editor.

[25] W. Matusik, C. Buehler, and L. McMillan. Polyhedral visual hulls for real-time rendering. *Procedings of Eurographics Workshop on rendering*, 2001.

[26] W. Matusik, C. Buehler, R. Raskar, L. McMillan, and Steven J. Gortler. Image-based visual hull. *SIGGRAPH*, 2000.

[27] R. Mech and P. Prusinkiewicz. Visual models of plants interacting with their environment. *SIGGRAPH*, pages 397–410, 1996.

[28] M.Teichmann and S.Teller. Assisted articulation of closed polygonal models,. *Proc.9th Eurographics Work-shop on Animation and Simulation*, pages 254–268., 1998.

[29] K. Murphy. Representing and learning hierarchical structure in sequential data. http://www-anw.cs.umass.edu/ cs691t/SS02/readings/murphy-hhmm-tr.ps, Novermber 2001.

[30] P E Oppendheimer. Real time design and animation of fractal plants and trees. *Computer Graphics*, 5:3–31, 1986.

[31] P.M.Hubbard. Approximating polyhedra with spheres for time-critical collision detection. *ACM Transactions on Graphics*, 15(3):179– 210, 2001.

[32] P. Prusinkiewicz. Modeling of spatial structure and development of plants: a review. *Scientia Horticulturae*, 74:113–149, 1998.

[33] P. Prusinkiewicz and M. Hammel. Language-restricted iterated function systems, 1994. Koch constructions and L-systems. In New Directions for Fractal Modeling in Computer Graphics. ACM Press, 1994. SIGGRAPH'94 Course Notes.

[34] P. Prusinkiewicz, M. Hammel, J. Hanan, and R. Mech. *Handbook of Formal Languages*, chapter Visual models of plant development. Springer–Verlag, 1996.

[35] P. Prusinkiewicz, M. S. Hammel, and E. Mjolsness. Animation of plant development. *Computer Graphics*, 27(Annual Conference Series):351–360, 1993.

[36] P. Prusinkiewicz, M. James, and R. Měch. Synthetic topiary. *Computer Graphics*, 28(Annual Conference Series):351–358, 1994.

[37] P. Prusinkiewicz and A. Lindemayer. *The Algorithmic Beauty of Plants.* Springer Verlag, New York, 2 edition, 1996.

[38] P. Prusinkiewicz, L. Muendermann, R. Karwowski, and B. Lane. The use of positional information in the modeling of plants. *Proceedings of SIGGRAPH*, pages 289 – 300, August 2001.

[39] L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, February 1989.

[40] L. Rabiner and B. Juang. *Theory and Implementation of Hidden Markov Models. In Fundamentals of Speech Recognition.* Prentice Hall, 1993.

[41] W. Reeves and R. Blau. Approximation and probabilistic algorithms for shading and rendering structured particle systems. *Computer Graphics*, 19(3):313–322, 1985.

[42] T. Sakaguchi and J. Ohya. Modeling and animation of botanical trees for interactive virtual environments. *Proceedings of the ACM VRST symposium*, pages 139–146, 1999.

[43] S. Seitz and C. Dyer. Photorealistic scene reconstruction by voxel coloring. *CVPR97*, pages 1067–1073, 1997.

[44] E. C. Sherbrooke, N. M. Patrikalakis, and E. Brisson. An algorithm for the medial axis transfrom of 3d polyhedral solids. *IEEE transaction on Visualization and Computer Graphics*, 2(1):44–61, 1996.

[45] I. Shlyakhter, M. Rozenoer, J. Dorsey, and S. Teller. Reconstructing 3d tree models from instrumented photographs. *IEEE CGA*, 21(3):53–61, June 2001.

[46] S. Sullivan and J. Ponce. Automatic model construction, pose estimation, and object recognition from photographs using triangular splines. *ICCV98*, pages 510–516, 1998.

[47] J.H.M Thornley and I.R Johnson. *Plant and crop modeling: a mathematical approach to plant and crop physiology.* Oxford Univ. Press, 1990.

[48] S. Ulam. On some mathematical properties connected with patterns of growth of figures. *In Proceedings of Symposia on Applied Mathematics*, 14:215–224, August 1962. American Mathematical Society.

[49] X. Zhao, P. de Reffye, F. Xiong, B. Hu, and Z. Zhan. Dual-scale automaton model for virtual plant development. *Chinese J. Computers*, 24(6), June 2001.

Appendix A

Appendix

A.1 Turtle Orientation commands

+	turn left around up vector
+(x)	turn x left around up vector
−	turn right around up vector
−(x)	turn x right around up vector
&	pitch down around left vector
&(x)	pitch x down around left vector
∧	pitch up around left vector
∧(x)	pitch x up around left vector
<	roll left (counter clockwise) around forward vector
<(x)	roll x left around forward vector
>	roll right (clockwise) around forward vector
>(x)	roll x right around forward vector

A.2 Special Orientation commands

|	turn 180 degrees around up vector
%	roll 180 degrees around forward vector
$	roll until horizontal
∼	turn/pitch/roll in a random direction
∼(x)	in a random direction with a maximum of x degrees
t	correction for gravity with 0.2
t(x)	correction for gravity with x

A.3 Movement commands

		when {} active
F	move forward and draw full length	record vertex
F(x)	move x forward and draw	record vertex
Z	move forward and draw half length	record vertex
Z(x)	move x forward and draw	record vertex
f	move forward with full length	record vertex
f(x)	move x forward	record vertex
z	move forward with half length	record vertex
z(x)	move x forward	record vertex
g	move forward with full length	don't record vertex
g(x)	move x forward	don't record vertex
.	don't move	record vertex

A.4 Structure commands

[	push current state
]	pop current state
{	start polygon shape
}	end polygon shape

A.5 Increase/Decrease commands

"	increment length with 1.1
'	decrement length with 0.9
"(x)	multiply length with x also '(x)
;	increment angle with 1.1
:	decrement angle with 0.9
:(x)	multiply angle with x also ;(x)
?	increment thickness with 1.4
!	decrement thickness with 0.7
?(x)	multiply thickness with x also !(x)

A.6 Additional commands

c	increment color index
c(x)	set color index to x
@	end of object