# MODELING AND RENDERING REALISTIC TREES

by

Deepali Bhagvat

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

ARIZONA STATE UNIVERSITY

May 2008

## ABSTRACT

This thesis presents a shape grammar for the procedural modeling of trees that faithfully reproduces the botanical characteristics visible in nature. The shape grammar allows for modeling of the trunk and higher-level branches, as well as the details of compound leaves without resorting to image-based approximation strategies. The prioritized grammar rules illustrate the development of hierarchical and detailed tree models. This thesis introduces a novel idea of using implicit equations to ray trace tree surfaces as an alternative to triangle meshes. Each atomic shape in the tree model is an individually ray-traced conical frustum. This method yields a tree surface with correct behavior at the joins between adjacent shape geometries. The thesis also presents a new algorithm for relief mapping the ray traced tree surface. Relief mapping provides additional realism by closely imitating the irregular bark structure found on natural trees. The algorithm uses concentric conical frustums, represented by individual implicit equations, to perform relief mapping. This thesis combines these methods with the existing rendering methods such as shadow mapping to produce detailed and high quality realistic trees suitable for close-up views in games and movies at real-time frame rates.

## ACKNOWLEDGMENTS

I take this opportunity to acknowledge all the people who made this thesis possible.

This work would not have been possible without the invaluable guidance and encouragement from my advisor, Dr. Peter Wonka. The very first course I took with him stimulated my interest in Computer Graphics. I learned innumerable things from him since then, be it staying up-to-date with research, reading and analyzing papers, writing my thesis, presenting the research, thinking clearly, staying organized, the list can go on and on... His valuable inputs have been instrumental to help me understand the intricacies of Computer Graphics in general, and my thesis topic in particular. I would also like to express my deepest gratitude to my co-advisor, Dr. Marco Janssen, who showed great faith in my abilities by offering me continued research assistantship. His work and vision helped me find the topic of this thesis. I am also grateful to Dr. Gerald Farin for reviewing my work and agreeing to be on my committee. I appreciate his excellent teaching and am thankful for his understanding, encouragement and support.

I express my profound thanks to Stephan Jeschke for his valuable contribution in the final stages of my research. He proposed and assisted me with the relief mapping algorithm, reviewed my work and provided constructive comments and suggestions. I salute his practical knowledge and enthusiasm. I would also like to extend special thanks to Kerstin Müller for the discussions about subdivision surfaces, especially explaining the special features and limit points of Catmull-Clark subdivision scheme.

Big thanks to all my colleagues in PRISM and CSID for providing a stimulating and fun environment. I am especially thankful to Allen Lee for his support and assistance in many different ways. I also wish to thank all my friends for providing an invigorating setting to learn and grow.

I am extremely grateful to my parents, Ashok and Aparna Bhagvat, for providing their support by creating and maintaining a motivating environment at home, and teaching me to give my best. I am also obliged to my brother, Ajit, and my sister-in-law, Pallavi, for their unconditional help and love.

Lastly, and most importantly, I am indebted to Pushpak who is the root cause of my degree. He motivated me to pursue the Masters program, and since then has shared my anxieties and achievements, giving infinite ideas and tips, helping me get through difficult times, providing a loving and caring environment. To him I dedicate this thesis.

TABLE OF CONTENTS

Page

LIST OF TABLES

# LIST OF FIGURES

Figure Page

# 1. INTRODUCTION

## 1.1. Motivation

Visualizing botanical structures is an important task in many computer graphics applications. Games with outdoor scenes and urban environments [51], [32], [33] require vegetation rendering. Social scientists explore the use of virtual environments to study governance of natural renewable resources [27], [26]. This includes dynamic vegetation visualization which may result as a different learning experience with the added context [25]. To provide an authentic immersion experience to the user, it is imperative that virtual world is modeled as detailed as its real-world counterpart. Two factors that affect the integration of realistic tree models into graphics applications are the complexity and the volume of data involved. Plants have a naturally complex structure with sufficient variance between species that makes it difficult to model them. A detailed reproduction of this complexity leads to generation of large amounts of data that need to be processed in real-time.

The idea of modeling trees procedurally using grammars was presented by Prusinkiewicz and Lindenmayer [39] in their seminal work on L-systems. L-System and its parameters can very well capture the tree architecture and define a tree model. However, converting it to a more concrete model for graphically representing the tree is still a challenge. Parametric modeling is another approach presented for tree modeling. Aono and Kunii [4] were precursors in this field. They identified the main parameters defining a tree and proposed four models with different levels of complexity. More recently, Weber and Penn [49] proposed a more complete model capable of generating a large variety of trees. But it requires the user to specify a large amount of parameters which can be a cumbersome process. Tree visualization is also addressed by another large class of papers. The main approaches used in these papers are polygonal meshes [6], billboards [13] or particle systems [42]. However, smooth surfaces with realistic shading and illumination effects still remains a challenge in tree visualization.

We address these challenges in this thesis by proposing (a) an approach for modeling trees using shape grammars at interactive speeds, (b) a pixel exact, yet fast method for tree surface generation suitable for close-up views, and (c) a method to render realistic trees that exhibit subtle details such as bark roughness at real-time frame rates for added realism.

## 1.2. Contribution of the Thesis

In this thesis, we present a unified framework for modeling and rendering realistic trees. Our framework uses a grammar engine to algorithmically model trees of a wide variety. The terminal shapes in the model are instantiated to skin the trees. Our skinning method employs implicit equations to ray trace the tree surface. For realistic visualization of tree bark, we add bumps to this ray traced surface and make the silhouettes irregular. The contribution of this thesis is as follows:

1. We are the first to propose the use of shape grammars to procedurally produce tree models.
2. We are the first to use implicit cone equation in a GPU-based ray tracer to generate smooth tree surfaces.
3. We introduce a relief mapping technique to add details to ray traced, non-triangulated conical surfaces.

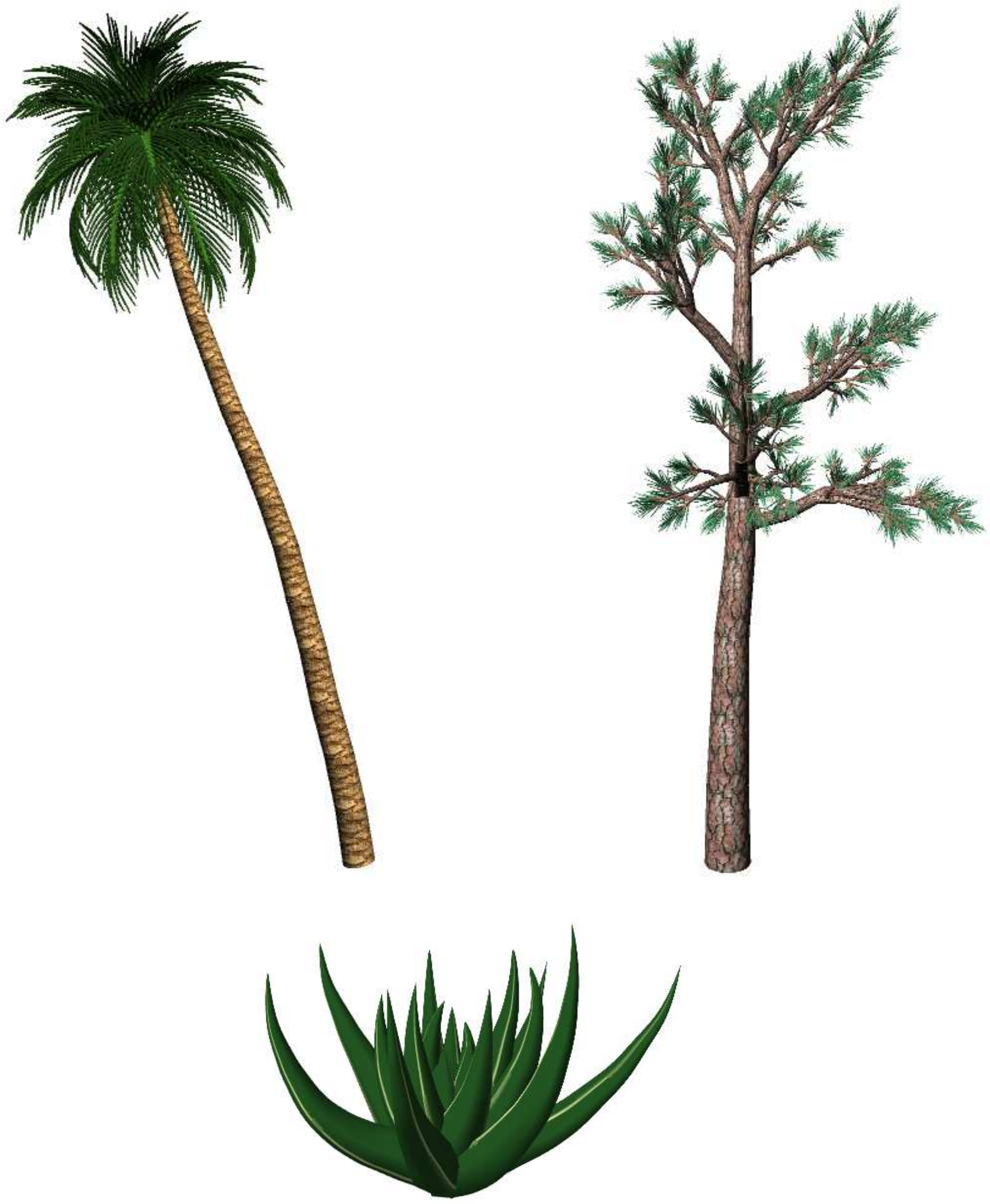


FIGURE 1. Trees rendered using our method

## 1.3. Organization of the Thesis

The thesis is structured as follows: We will explain previous work in Chapter 2. This will involve reviewing the work related to procedural methods, tree modeling and most relevant approaches for rendering. In chapter 3 we will describe a unified framework to model trees, generate surface geometry and add rendering effects. We will also present several tree modeling examples in this chapter. The results in chapter 4 will present modeling and rendering statistics for our methods. In chapter 5 we discuss advantages and limitations of our approach and future work involved.

# 2. RELATED WORK

We will review related work in three parts: (a) tree structure modeling, (b) tree surface modeling and rendering and (c) detail rendering.

## 2.1. Tree Structure Modeling

In this section we will review various approaches used to generate a logical tree model.

### 2.1.1. Procedural Modeling with Grammars

Prusinkiewicz and Lindenmeyer [39] introduced L-systems to model and simulate plant development. Originally, they did not include enough detail to allow for comprehensive modeling of higher plants. The emphasis was on plant physiology that deals with biomechanics, genes and tissues. Their geometric aspects were beyond the scope of the theory. Subsequently, several geometric interpretations of L-systems were proposed with a view to turning them into a versatile tool for plant modeling [38], [40]. The central concept of L-systems is that of rewriting. In general, rewriting is a technique for defining complex objects by successively replacing parts of a simple initial object using a set of rules. The most extensively studied and the best understood rewriting systems operate on character strings. The first formal definition of such a system was given at the beginning of this century by Thue [11], but a wide interest in string rewriting was spawned in the late 1950s by Chomskys work on formal grammars [44]. He applied the concept of rewriting to describe the syntactic features of natural languages. Other production systems that serve as a basis for procedural modeling are graph grammars [18], shape grammars [45] and attributed grammars [30]. Shape grammars were introduced by G. Stiny [45], [46] and can be simplified to set grammars [47], [51] to make them more amenable to computer implementation. The original formulation of the shape grammar operates directly on an arrangement of labeled points and lines. However, the derivation is intrinsically complex and usually done manually, or if by computer then with a human deciding on the rules to apply. Shape grammars are successfully used for the construction and analysis of architectural design [16], [17], [48]. Procedural modeling using shape grammars was recently explored by Wonka *et al.* [51] and Muller *et al.* [33] to model buildings and cities. They introduced *CGA shape*, a shape grammar to model CG architecture with production rules that iteratively evolve a design by creating more and more details.

### 2.1.2. Parametric Modeling

Aono and Kunii [4] identified the main parameters defining a tree and proposed four models with different levels of complexity. Reeves and Blau [42] used a set of characteristics and dimensions to create a tree skeleton recursively. They used stochastic parameters to model tree foliage. Weber and Penn [49] coined the term "parametric modeling" to create tree skeletons. They presented parametric equations to calculate branch part attributes and leaf densities. Yu *et al.* [52] extended this approach by creating parameter templates that drive the procedural model to create tree skeleton.

### 2.1.3. Other Methods

Methods other than procedural and parametric have also been proposed to model the tree structure. Bloomenthal [6] presented maple tree axes generation using interpolating cubic splines from sparse point data. Deussen and Lintermann proposed a graph networking system in their work [31], [15] to model the tree skeleton interactively. Various methods such as [41] and [34] used volumetrically reconstruction to create tree models with photographs.

## 2.2. Tree Surface Modeling and Rendering

In this section we will review various approaches used to represent trees graphically. This includes (a) geometry based approaches (b) image based approaches (c) hybrid approaches and (d) other approaches.

### 2.2.1. Geometry Based Approaches

Bloomenthal [6] proposed using approximate Frenet frame to generate branch surfaces whereas Weber and Penn [49] used cylinders and predefined polygons to create tree surface geometry. Galbraith *et al.* also used cone primitives to model branches and proposed using implicit equations to model smooth branch junctions [23] as well as non-smooth features such as branch bark ridges and bud scale scars [24]. Another approach to create tree surface is to use interactive modeling tools such as XFrog [3] and TreeMaker [2]. Deussen *et al.* extended this approach by interpolating between family of models created using XFrog to render plant ecosystems [14].

### 2.2.2. Image Based Approaches

Sprites and billboards are the basic techniques to represent trees in virtual environment. Décoret *et al.* proposed an advanced version of this technique as billboard clouds [13]. Here a complex 3D model is represented as a small number of planar primitives. Its extension [21] calculated billboards for foliage and branches separately from the trunk and merged the result to produce a superior quality representation. [41] and [12] created view dependent textures to represent trees. These advanced methods produce very realistic results but they need to perform certain preprocessing steps which is unsuitable for modeling and rendering at run time.

### 2.2.3. Hybrid Approaches

Hybrid methods that use meshes as well as images have been presented to improve modeling and rendering speed. Szijártó *et al.* [22] and Yu *et al.* [52] represent stems using geometry and foliage using textures for real-time rendering of large forests. Remolar *et al.* [43] create dynamic foliage textures from a pre-created tree mesh. Another method [7] which is also an extension of the billboard clouds, recursively generate billboards and simplified mesh to reduce gaps in the model. Colditz *et al.* [9] use tree mesh for close-up views and dynamically changing billboards for farther views. One of the modeling tools that use hybrid approach by merging low-polygon branches with view-aligned billboard foliage is SpeedTree [1]. The hybrid methods may introduce popping effects while switching between meshes and textures, compromising the visual quality.

### 2.2.4. Other Approaches

Reeves and Blau [42] proposed use of a particle system to render complex scenes involving vegetation. Mantler *et al.* [32] presented a two pass point based approach. The first pass renders the tree in a buffer from many different directions and stores as a sorted list of points. The second stage selects the set that closely matches the current view direction. However as these methods use approximate shading calculations, they are more appropriate to view from a medium to far distances.

## 2.3. Detail Rendering

In this section, we will review work related to detail rendering in two parts (a) rendering surface details using texture based approaches and (b) rendering shadows.

### 2.3.1. Rendering Surface Details

Simulation of rough surface by perturbing normals used for illumination, without changing the underlying surface geometry was introduced by Jim Blinn [5]. Peercy *et al.* [37] extended it to perform normal mapping in tangent space. Nvidia white paper [28] described how this can be done on the modern GPUs in real-time. Although these approaches create an illusion of rough surfaces, silhouettes appear smooth and self-shadows are not possible. Oliveira and Policarpo [36] presented a relief mapping technique that creates correct silhouettes along with shadows and self-occlusion. Their approach uses a quadric surface to locally approximate the object geometry at each vertex. The quadric coefficients are computed using a least-squares fitting algorithm as a pre-processing stage and are interpolated during rasterization. Thus, each fragment contributes a quadric surface for a piecewise-quadric object-representation that is used to produce correct renderings of geometrically-detailed surfaces and silhouettes.

### 2.3.2. Rendering Shadows

Shadows provide important information about relative positions of the objects and lights in a scene. Shadow volumes is an object space shadowing technique introduced by Crow [10] whereas shadow mapping is an image based shadowing technique introduced by Williams [50]. Again, Nvidia white papers [19] and [20] described how this can be done on the modern GPUs in real-time. Shadow mapping techniques are less sensitive to geometric complexity and need not draw additional geometry, but are more prone to aliasing artifacts as against the shadow volume approach. Techniques to improve shadows rendered by the shadow mapping technique are presented by Mark Kilgard [29]. Our framework implements shadow mapping with percentage closer filtering.

# 3. OUR APPROACH

## 3.1. Overview

A tree is composed of a hierarchy of elements such as a trunk, which is level 0 branch, higher level branches, twigs, leaves, leaflets and so on. The leaves if compound, have a main stalk to which leaflets are attached with secondary stalks. Leaflets of some trees also have web of veins as shown in Figure 2.

FIGURE 2. Botanical patterns in nature (Courtesy: Google Images)

Thus tree elements follow a well defined stem pattern at various levels. We exploit this pattern to design our tree visualization framework (see Figure 3). Various parts of the framework are as follows:

1. Procedural grammar engine to create tree models in terms of terminal shapes using a shape grammar.
2. Tree surface modeler and renderer to generate and render tree surface using OpenGL cone primitives, Catmull-Clark subdivision scheme or a ray tracer.
3. Detail renderer to add bark details to the ray traced surface using relief mapping and to render tree shadows using shadow mapping.

In the following sections, we describe each stage in our framework in more detail.

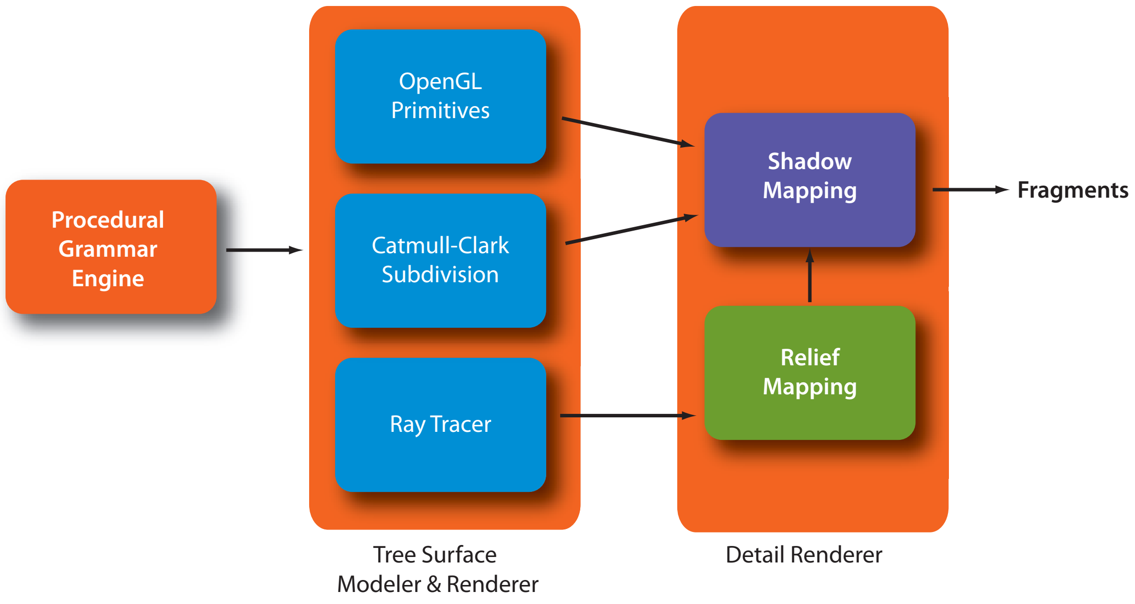

Procedural Grammar Engine
OpenGL Primitives
Catmull-Clark Subdivision
Ray Tracer
Tree Surface Modeler & Renderer
Shadow Mapping
Relief Mapping
Detail Renderer
Fragments

FIGURE 3. Overview of tree visualization framework

## 3.2. Procedural Modeling

We present a procedural approach to create detailed tree models. Our approach represents tree parts as shapes and uses shape grammars to modify them and their attributes to create complex tree models. In this section, we review the important concepts of shape grammars and explain the rules for modeling trees, with examples. We also illustrate how it can be applied to model several tree species having detailed branching structure and compound leaves.

### 3.2.1. Shape

A shape is represented by a symbol and attributes. Symbol is a string representing the shape. Symbols can be terminal $\in \Sigma$ or non-terminal $\in V$. The corresponding shapes are called terminal shapes and non-terminal shapes. The most important non-terminal shapes in our procedural model are *tree* and *stem* whereas *segment* is a terminal shape. Attributes associated with these shapes are categorized as geometric attributes and numeric attributes. A size vector **S**, position **P** and three orthogonal vectors **X**, **Y** and **Z**, describing a coordinate system are the most important geometric attributes used to define geometry of the shape (see Figure 4). Examples of numeric attributes are *level* and *type* associated with a *stem*. The attributes are defined in the form *shape_attribute*. For example *stem_type* represents the type attribute of a *stem* which can be either $BRANCH$, $LEAF$ or $LEAFLET$ indicating that the stem is part of a branch, leaf or leaflet.

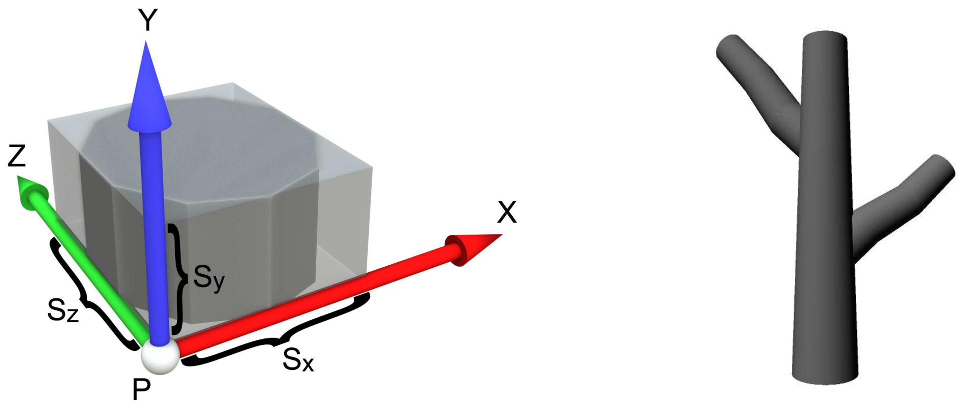


FIGURE 4. Definition of a *shape*

## 3.2.2. Shape Grammar

A shape grammar consists of rules that are applied to shapes to create complex shapes or models. They are defined in the following form

$id : predecessor : cond \leadsto successor$

where *id* is a unique identifier for the rule, $predecessor \in V$ is a symbol identifying a shape that is to be replaced or assembled with *successor* and *cond* is a guard (logical expression) that has to evaluate to true in order for the rule to be applied.

For example, the rule

$1 : stem(L_n) : stem\_type = BRANCH \rightsquigarrow stem(BRANCH, L_{n+1})$

appends a child branch to an existing stem if it is of type $BRANCH$. Different types of rules are as follows:

Replace rule: Replace rule replaces a predecessor shape with a successor shape. For example, the rule

$1 : tree \rightsquigarrow stem(BRANCH, L_0)$

specifies that a tree is made up of a trunk (see Figure 5).

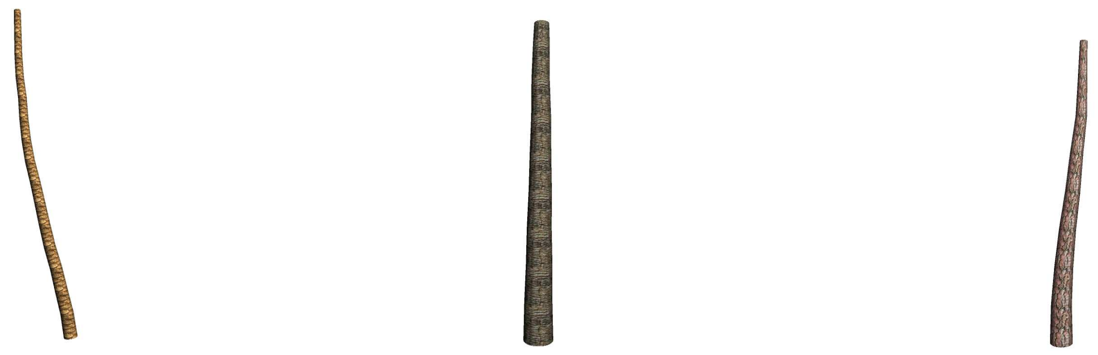


FIGURE 5. Replace rule example

Append rule: Append rule appends a successor shape to a predecessor shape. For example, the rule

$1 : stem : stem\_type = LEAF \& stem\_level = L_0 \rightsquigarrow stem(LEAF, L_1)$

adds a secondary leaf stalk to the main leaf stalk (see Figure 6).

General rules: General rules modify shape attributes similar to the L-systems. $T(T_x, T_y, T_z)$ translates the shape position, $P$. $R_x(roll)$, $R_y(yaw)$ and $R_z(pitch)$ rotate the respective axis of the coordinate system. $S(S_x, S_y, S_z)$ sets the size of the

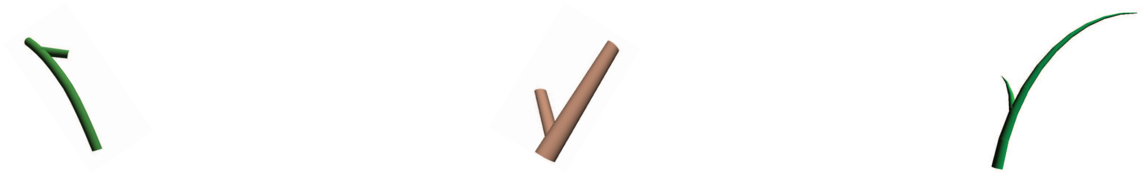


FIGURE 6. Append rule example

shape (see Figure 4). We use the [ and ] to push and pop the current scope on the stack.

Commands are used in the rules to perform certain tasks. Different commands used in our shape grammar are as follows:

Instance command: Instance command adds an instance of a geometric primitive to a shape and is defined in the form

$I(\text{“}objId.obj\text{”})$

where objId can be a line, a cone, a pyramid or any other primitive (see Figure 7).

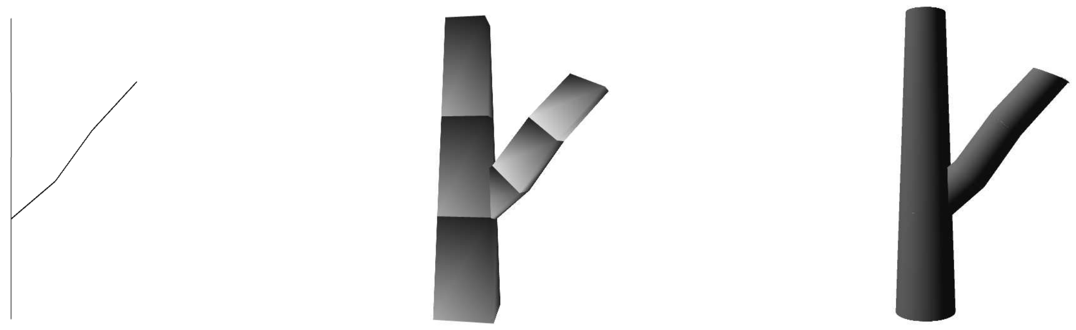


FIGURE 7. Instance command example

Repeat command: Repeat command is used to replace or append multiple successors to a predecessor. For example, the rule

$1 : stem \rightsquigarrow Repeat(15)segment$

uses repeat command to replace the *stem* with 15 *segments* (see Figure 8)

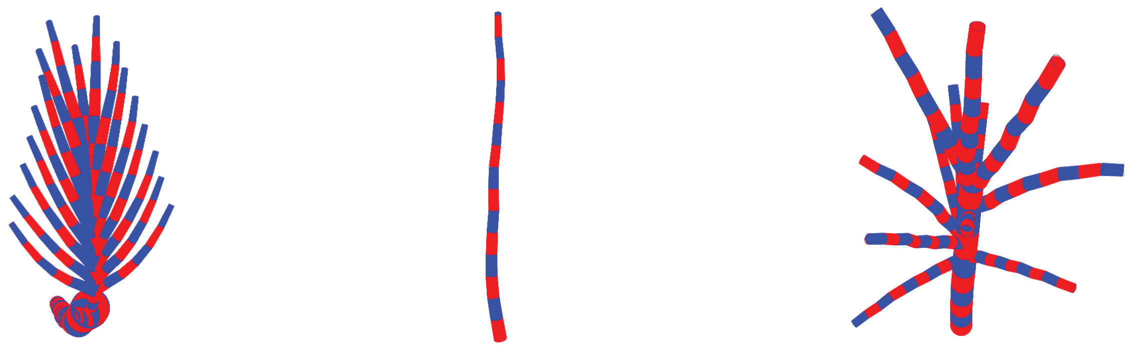


FIGURE 8. Repeat command example

Set command: Set command is used to set value of a variable and is written as

$Set(variable, 54)$

Random command: Random command is used to create a random number between the given limits. For example,

$Random(-10, 10)$

returns a random number between -10 to 10 with a Gaussian distribution.

Variables: Variables are values specified by the user at the start of the modeling process or calculated internally during the modeling process using shape attributes and other variables. For example, $no\_of\_segments\_per\_stem$ is a variable calculated using $ceil(\frac{stem\_length}{segment\_length})$ where $stem\_length$ and $segment\_length$ are shape attributes.

### 3.2.3. Modeling Process

A configuration is a finite set of basic shapes. The production process can start with an arbitrary configuration of shapes $A$, called the axiom, and proceeds as follows: (1) Select an active shape with symbol $B$ in the set, (2) choose a production rule with $B$ on the left hand side to compute a successor for $B$, a new set of shapes $BNEW$, (3) mark the shape $B$ as inactive and add the shapes $BNEW$ to the configuration and continue with step (1). When the configuration contains no more non-terminals, the production process terminates. For the selection algorithm in step one we assign a priority to all rules according to the detail represented by the shape to obtain a (modified) breadth-first derivation: we simply select the shape with the rule of highest priority in step one. This strategy guarantees that the derivation proceeds from low detail to high detail in a controlled manner. The use of different seeds and random values leads to a variety of trees in the same specie.

The remainder of this section describes modeling of four different tree species with our shape grammar.

### 3.2.4. Modeling Example: Elm Tree

This section illustrates modeling an elm tree (see Figure 9) with grammar rules. For simplicity, we consider the tree as leafless and focus on its branches. The grammar starts with the *tree* as its axiom and creates a trunk which is a *stem* of type, $BRANCH$ and level, $L_0$.

The grammar then selects priority 2 rules to add higher level branches. Rule 2 is a generic rule used to add one or more child branches to a branch of any level.

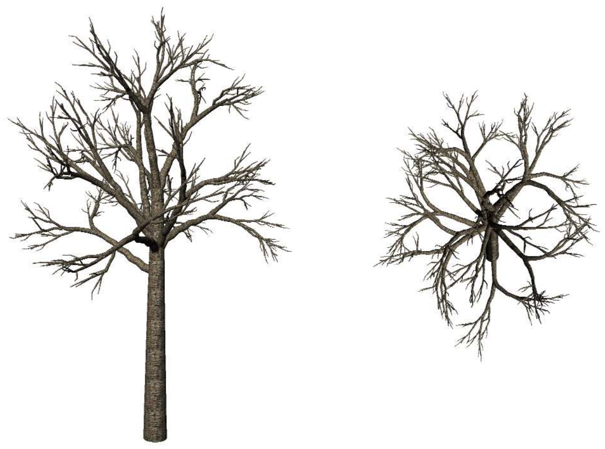


FIGURE 9. Elm tree modeled using our shape grammar

This rule uses variables calculated using the *stem* attributes to set attributes of the child levels to create branches specific to the tree specie being modeled. For example, *pitch_step* is calculated using *stem_min_pitch* and *stem_max_pitch* to create child branches making different angles with the parent branch. This takes care of tropism occurring in plants due to gravity and light. We start with the topmost child branch and go down along the branch. Our grammar uses lookup tables to read level-wise *stem* attributes set by the user as a preprocessing step. For example, *stem_child_offset* table stores child offsets per level for each type of *stem*. The tree displayed in Figure 9 uses six levels of branches. Our tree model at this stage consists of branches where each branch is represented as a single shape *stem*.

To add next level of detail to the model, our grammar selects the priority 3 rules. Rule 3 is again a generic rule to replace the *stem* of any type and any level with multiple *segments* using the *Repeat* command. Again, this rule uses variables calculated using the *stem* attributes, and sets the *segment* attributes using these variables and other *stem* attributes. These are used while creating the *segment* to set its translation, rotation and size as specified in the third rule. For example, the variable $pitch\_step$ is calculated using the stem attribute $stem\_base\_to\_top\_pitch\_difference$ and used to set the $segment\_pitch$ attribute. The *segments* are then instantiated using a cone object as elm tree branches are conical. Some attributes such as $segment\_pitch$ are then modified before the next *segment* creation to generate curved branches.

The set of rules used to model this tree is as follows:

PRIORITY 1:

$1 : tree \rightsquigarrow [stem(BRANCH, L_0)]$

PRIORITY 2:

$2 : stem(L_n) : stem\_type = BRANCH \rightsquigarrow$

$Set(pitch\_step, (stem\_min\_pitch - stem\_max\_pitch)/no\_of\_child\_stems)$

$Set(stem\_yaw, Random(-stem\_min\_yaw, stem\_max\_yaw))$

$Set(stem\_pitch, stem\_min\_pitch)$

$Set(stem\_y, stem\_child\_offset[index])$

$Repeat(no\_of\_child\_stems)[stem(BRANCH, L_{n+1})]$

$Set(stem\_pitch, stem\_pitch + pitch\_step)$

PRIORITY 3:

$3 : stem \rightsquigarrow$

$Set(radius\_step, (stem\_top\_radius - stem\_base\_radius)/no\_of\_segments\_per\_stem)$

$Set(pitch\_step, (stem\_base\_to\_top\_pitch\_difference)/no\_of\_segments\_per\_stem)$

$Set(segment\_y, 0)$

$Set(segment\_pitch, stem\_pitch)$

$Set(segment\_base\_radius, stem\_base\_radius)$

$Set(segment\_top\_radius, segment\_base\_radius + radius\_step)$

$Repeat(no\_of\_segments\_per\_stem)[segment]$

$4 : segment \rightsquigarrow T(0, segment\_y, 0)$

$R_y(stem\_yaw + Random(-stem\_yaw\_min\_limit, stem\_yaw\_max\_limit))$

$R_z(segment\_pitch + Random(-stem\_pitch\_min\_limit, stem\_pitch\_max\_limit))$

$S(segment\_length, segment\_base\_radius, segment\_top\_radius)I(\text{“}cone.obj\text{”})$

$Set(segment\_y, segment\_y + segment\_length)$

$Set(segment\_pitch, segment\_pitch + pitch\_step)$

$Set(segment\_base\_radius, segment\_top\_radius)$

$Set(segment\_top\_radius, segment\_base\_radius + radius\_step)$

### 3.2.5. Modeling Example: Aloe Vera Bush

In this section we describe the shape grammar for aloe vera bush that illustrates modeling of simple leaves. The bush does not have a trunk or branches and starts directly with leaves. Also, as opposed to trees, bushes have multiple level 0 stems. The first rule replaces the axiom *tree*, with many leaves as specified by the first rule. The model in the Figure 10 has twenty leaves. As the leaves in aloe vera are arranged in a spiral fashion, our grammar rotates the leaves around y axis by *stem_yaw_increment* and increases the y coordinate by *stem_y_increment*, both specified by the user at the start. The *index* variable goes from 1 to *no_of_leaves*

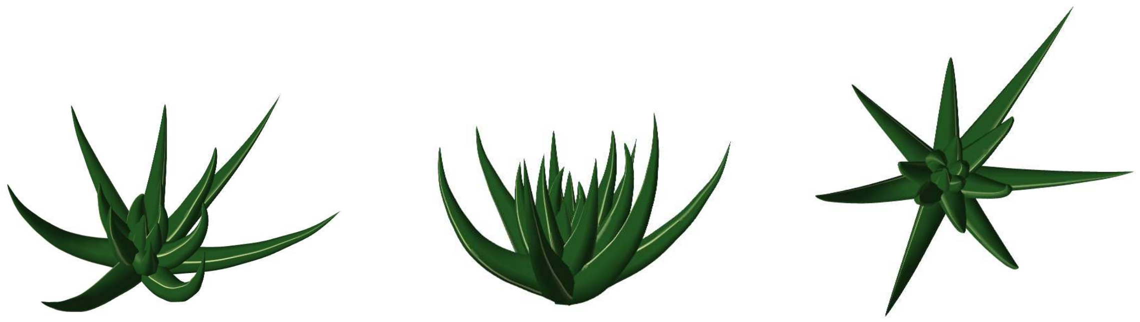


FIGURE 10. Aloe vera bush modeled using our shape grammar

pointing to the index of the current leaf being modeled.

The second and third rule add next level of detail by replacing the leaves with multiple *segments* and instantiating *segments* with cone objects. The rule set is as follows:

PRIORITY 1:

$1 : tree \rightsquigarrow Set(stem\_yaw, index \times stem\_yaw\_increment)$

$Set(stem_y, index \times stem\_y\_increment)$

$Repeat(no\_of\_leaves)[stem(LEAF, L_0)]$

PRIORITY 2:

$2 : stem \rightsquigarrow$

$Set(radius\_step, (stem\_top\_radius - stem\_base\_radius)/no\_of\_segments\_per\_stem)$

$Set(pitch\_step, (stem\_base\_to\_top\_pitch\_difference)/no\_of\_segments\_per\_stem)$

$Set(segment\_y, 0)$

$Set(segment\_pitch, stem\_pitch)$

$Set(segment\_base\_radius, stem\_base\_radius)$

```
    $Set(segment\_top\_radius, segment\_base\_radius + radius\_step)$
    $Repeat(no\_of\_segments\_per\_stem)[segment]$
$3 : segment \rightsquigarrow T(0, segment\_y, 0)$
    $R_y(stem\_yaw + Random(-stem\_yaw\_min\_limit, stem\_yaw\_max\_limit))$
    $R_z(segment\_pitch + Random(-stem\_pitch\_min\_limit, stem\_pitch\_max\_limit))$
    $S(segment\_length, segment\_base\_radius, segment\_top\_radius) I(\text{"cone.obj"})$
    $Set(segment\_y, segment\_y + segment\_length)$
    $Set(segment\_pitch, segment\_pitch + pitch\_step)$
    $Set(segment\_base\_radius, segment\_top\_radius)$
    $Set(segment\_top\_radius, segment\_base\_radius + radius\_step)$
```

### 3.2.6. Modeling Example: Coconut Tree

A coconut tree has a trunk with no higher level branches and spiral arrangement of leaves similar to the aloe vera bush. But the leaves in this case are compound with higher level leaf stalks and leaflets. Thus, the set of rules to model this tree is a combination of rules described in the previous two sections with some additional rules. The trunk is created in the same way as the elm tree described in the section 3.2.4.

Level 0 leaf stalks are created in the same way as the aloe vera bush, however, here we start from the top of the trunk and go down decrementing the y coordinate of the leaf position. Level 1 leaf stalks are modeled similar to the higher level branches except one difference. The stalks are on the opposite sides of the main

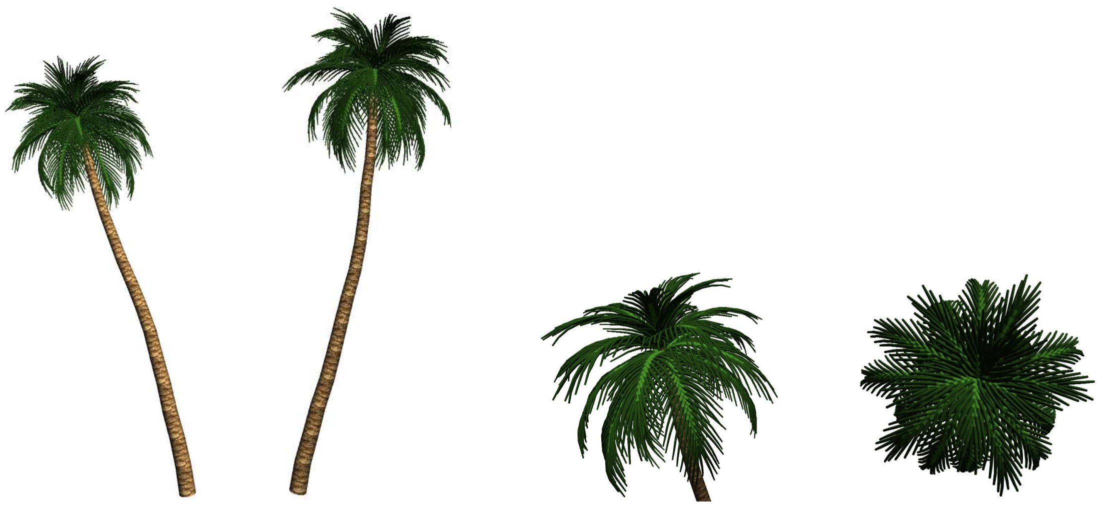


FIGURE 11. Coconut tree modeled using our shape grammar

stalk creating opposite leaf arrangement. Hence instead of using a random yaw between the specified branch yaw limits, our grammar adds $\pi$ each time it creates the leaf stalk.

A leaflet is added at the tip of each level 1 leaf stalk. *Segments* for branches, leaves and leaflets are created in a similar manner as in 3.2.4. The rule set for modeling coconut tree is as follows:

PRIORITY 1:

$1 : tree \rightsquigarrow [stem(BRANCH, L_0)]$

PRIORITY 2:

$2 : stem : stem\_type = BRANCH \rightsquigarrow Set(stem\_yaw, index \times stem\_yaw\_increment)$

$Set(stem_y, index \times stem\_y\_increment)$

$Repeat(no\_of\_leaves)[stem(LEAF, L_0)]$

$3 : stem(L_n) : stem\_type = LEAF \rightsquigarrow$

```
        $Set(pitch\_step, (stem\_min\_pitch - stem\_max\_pitch)/no\_of\_child\_stems)$
        $Set(stem\_yaw, stem\_yaw + \Pi))$
        $Set(stem\_pitch, stem\_min\_pitch)$
        $Set(stem\_y, stem\_child\_offset[index])$
        $Repeat(no\_of\_child\_stems)[stem(LEAF, L_{n+1})]$
        $Set(stem\_pitch, stem\_pitch + pitch\_step)$
$4 : stem : stem\_type = LEAF \rightsquigarrow [stem(LEAFLET, L_0)]$
PRIORITY 3:
$5 : stem \rightsquigarrow$
        $Set(radius\_step, (stem\_top\_radius - stem\_base\_radius)/no\_of\_segments\_per\_stem)$
        $Set(pitch\_step, (stem\_base\_to\_top\_pitch\_difference)/no\_of\_segments\_per\_stem)$
        $Set(segment\_y, 0)$
        $Set(segment\_pitch, stem\_pitch)$
        $Set(segment\_base\_radius, stem\_base\_radius)$
        $Set(segment\_top\_radius, segment\_base\_radius + radius\_step)$
        $Repeat(no\_of\_segments\_per\_stem)[segment]$
$6 : segment \rightsquigarrow T(0, segment\_y, 0)$
        $R_y(stem\_yaw + Random(-stem\_yaw\_min\_limit, stem\_yaw\_max\_limit))$
        $R_z(segment\_pitch + Random(-stem\_pitch\_min\_limit, stem\_pitch\_max\_limit))$
        $S(segment\_length, segment\_base\_radius, segment\_top\_radius)I(\text{“}cone.obj\text{”})$
        $Set(segment\_y, segment\_y + segment\_length)$
        $Set(segment\_pitch, segment\_pitch + pitch\_step)$
```

$Set(segment\_base\_radius, segment\_top\_radius)$

$Set(segment\_top\_radius, segment\_base\_radius + radius\_step)$

### 3.2.7. Modeling Example: Pine Tree

Our last example describes modeling a pine tree which has multiple branching levels and compound leaves with multiple leaflets. Hence the grammar presented in this section is a superset of grammar presented so far. The trunk and higher level branches are created similar to the Elm tree.

As the leaves of pine tree are not arranged spirally around a branch but are present at the end of twigs, leaf addition is simplified as specified by rule 3. Rule 4 adds higher level leaf stalks similar to the Coconut tree. But as the leaf arrangement is whorled, we add $\frac{\pi}{2}$ to the $stem\_yaw$.

As two leaflets are attached to each leaf stalk, we use the *Repeat* command to add leaflets at the end of leaf stalks with a random yaw. Again, *segments* are added in the same fashion as earlier. The rules used to model the pine tree are as follows:

PRIORITY 1:

$1 : tree \rightsquigarrow [stem(BRANCH, L_0)]$

PRIORITY 2:

$2 : stem(L_n) : stem\_type = BRANCH \rightsquigarrow$

$Set(pitch\_step, (stem\_min\_pitch - stem\_max\_pitch)/no\_of\_child\_stems)$

$Set(stem\_yaw, Random(-stem\_min\_yaw, stem\_max\_yaw))$

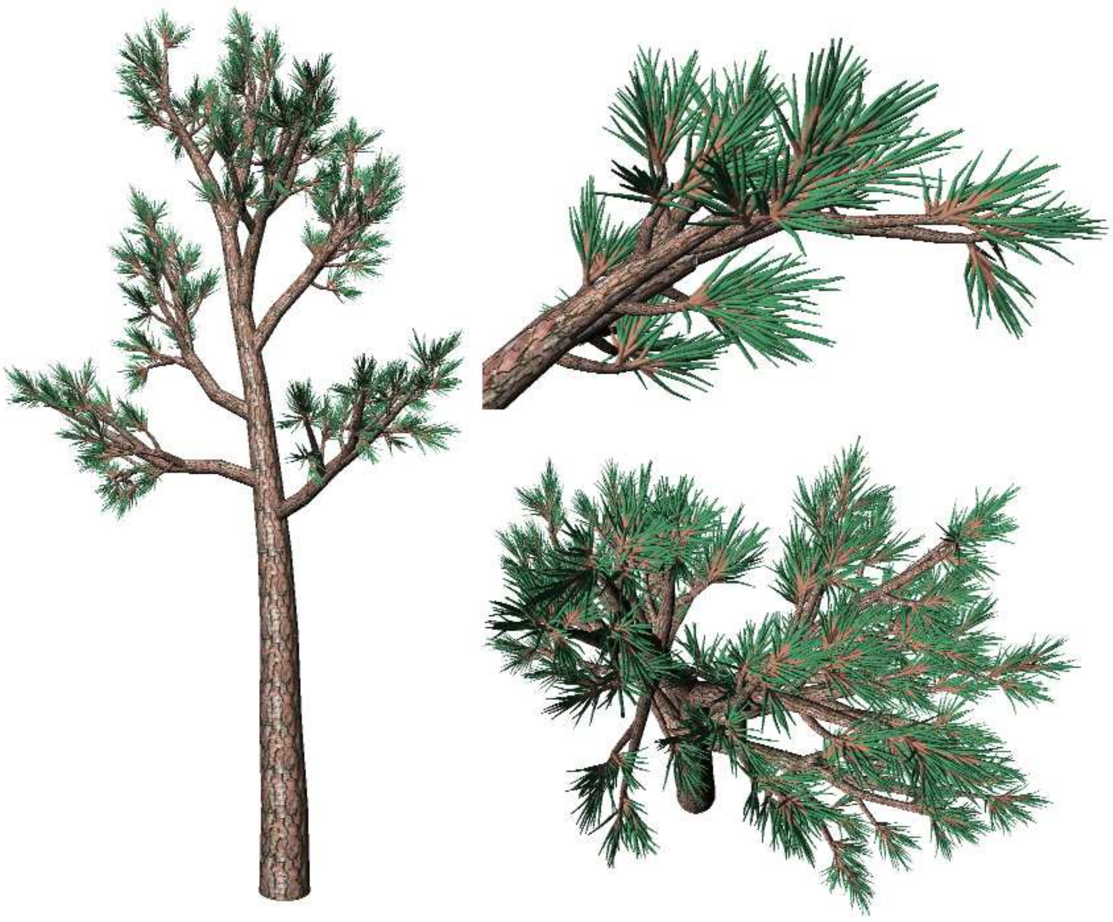


FIGURE 12. Pine tree modeled using our shape grammar

```
    $Set(stem\_pitch, stem\_min\_pitch)$
    $Set(stem\_y, stem\_child\_offset[index])$
    $Repeat(no\_of\_child\_stems)[stem(BRANCH, L_{n+1})]$
    $Set(stem\_pitch, stem\_pitch + pitch\_step)$
$3 : stem(L_n) : stem\_type = BRANCH \rightsquigarrow [stem(LEAF, L_0)]$
$4 : stem(L_n) : stem\_type = LEAF \rightsquigarrow$
    $Set(pitch\_step, (stem\_min\_pitch - stem\_max\_pitch)/no\_of\_child\_stems)$
    $Set(stem\_yaw, stem\_yaw + \pi/2)$
    $Set(stem\_pitch, stem\_min\_pitch)$
    $Set(stem\_y, stem\_child\_offset[index])$
    $Repeat(no\_of\_child\_stems)[stem(LEAF, L_{n+1})]$
    $Set(stem\_pitch, stem\_pitch + pitch\_step)$
$5 : stem : stem\_type = LEAF \rightsquigarrow$
    $Set(stem\_yaw, Random(-stem\_min\_yaw\_limit, ste\_max\_yaw\_limit))$
    $Repeat(no\_of\_lealets)[stem(LEAFLET, L_0)]$
PRIORITY 3:
$6 : stem \rightsquigarrow$
    $Set(radius\_step, (stem\_top\_radius - stem\_base\_radius)/no\_of\_segments\_per\_stem)$
    $Set(pitch\_step, (stem\_base\_to\_top\_pitch\_difference)/no\_of\_segments\_per\_stem)$
    $Set(segment\_y, 0)$
    $Set(segment\_pitch, stem\_pitch)$
    $Set(segment\_base\_radius, stem\_base\_radius)$
```

```
        $Set(segment\_top\_radius, segment\_base\_radius + radius\_step)$

        $Repeat(no\_of\_segments\_per\_stem)[segment]$

$7 : segment \rightsquigarrow T(0, segment\_y, 0)$

        $R_y(stem\_yaw + Random(-stem\_yaw\_min\_limit, stem\_yaw\_max\_limit))$

        $R_z(segment\_pitch + Random(-stem\_pitch\_min\_limit, stem\_pitch\_max\_limit))$

        $S(segment\_length, segment\_base\_radius, segment\_top\_radius) I(\text{“}cone.obj\text{”})$

        $Set(segment\_y, segment\_y + segment\_length)$

        $Set(segment\_pitch, segment\_pitch + pitch\_step)$

        $Set(segment\_base\_radius, segment\_top\_radius)$

        $Set(segment\_top\_radius, segment\_base\_radius + radius\_step)$
```

## 3.3. Surface Generation and Rendering

As branches of most of the trees are conical, *segments* generated using the procedural modeling approach described in the previous section are instantiated using the cone object as a geometric primitive. We describe three different approaches to create the conical surface. We start with OpenGL cone primitive. We then explain creation of a conical surface using the Catmull-Clark subdivision scheme. The third approach uses implicit equations to ray trace tree surface on a GPU.

### 3.3.1. OpenGL Cone Primitive

To instantiate *segments* with cone objects, we use the OpenGL utility library function, gluCylinder where the base and top radii, length and transformations of the cylinder are set according to the geometric attributes of the *segment* shape. We use eight slices i. e. eight subdivisions around the *segment* axis and one stack i. e. one subdivision along the *segment*. As shown in Figure 13 (middle), this creates a quad mesh based tree surface, which is then textured and rendered as shown in Figure 13 (right). However, there may exist gaps between the adjacent conical frustums creating cracks as shown in Figure 14.

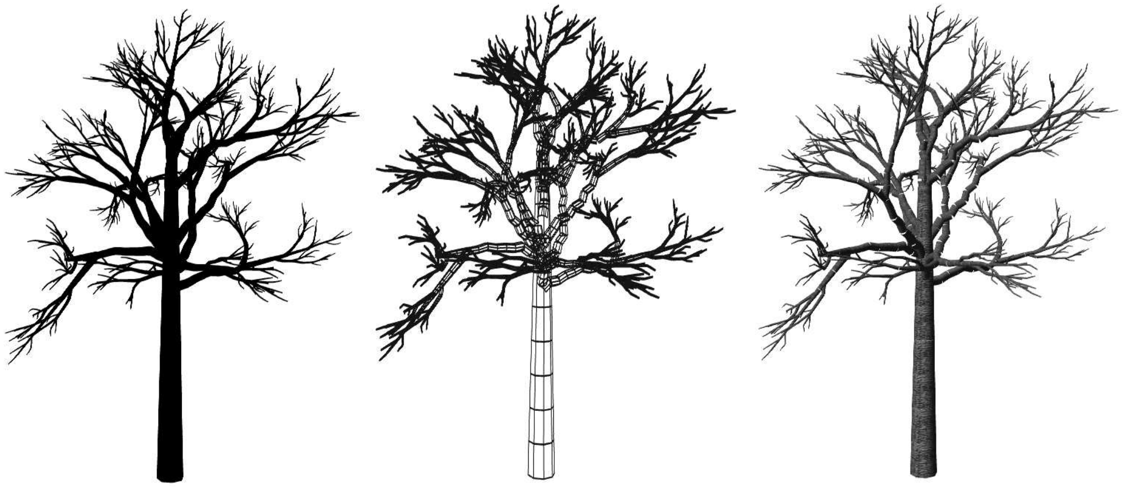


FIGURE 13. Tree surface using OpenGL primitives

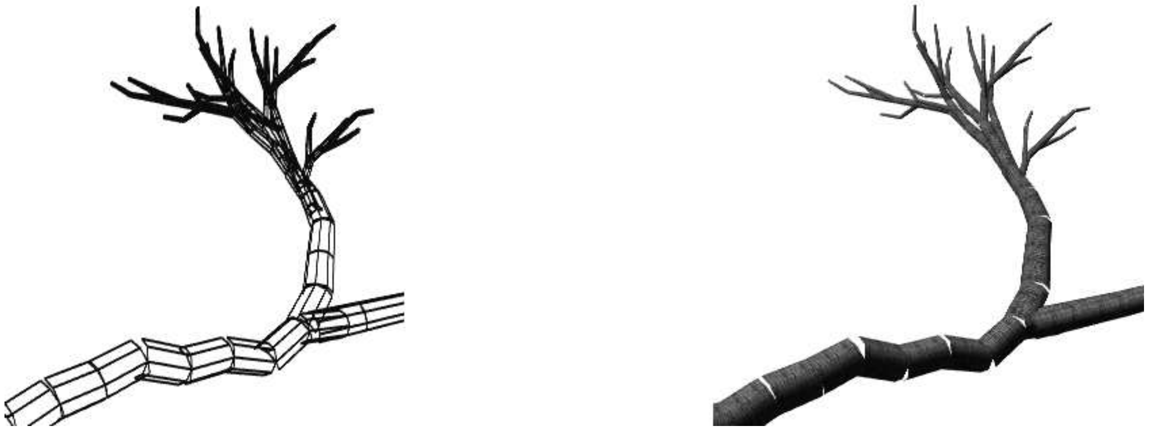


FIGURE 14. Problems with OpenGL primitives

### 3.3.2. Catmull-Clark Subdivision Surface

To create a connected surface, we consider the neighboring *segments* along with the *segment* in question. We begin by instantiating each *segment* of the tree model with a pyramid. A close-up view of a partial branch instantiated using pyramids is shown in Figure 15 (left). The lengths of pyramid sides and transformations applied to it are calculated using the geometric attributes of the *segment*. Then we update the vertices of the pyramidal frustum using intersection points with the adjacent frustums to bridge the gaps between them as shown in Figure 15 (right). This pyramidal geometric model then becomes a basis of our more advanced surface generation approaches.

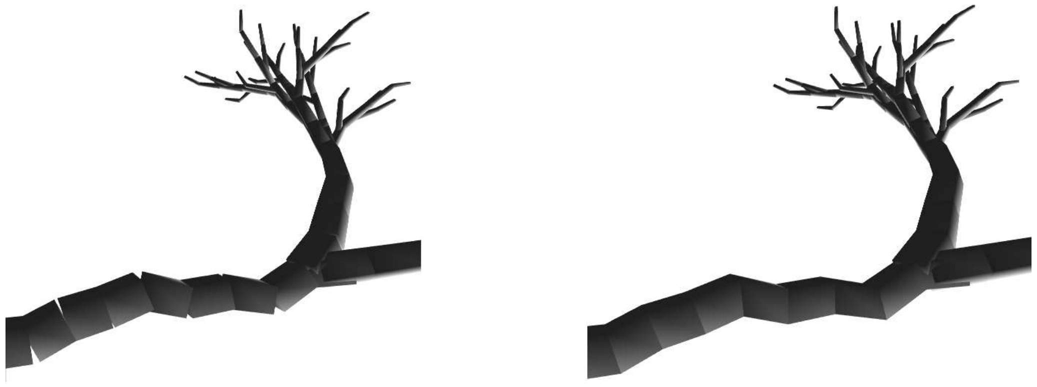


FIGURE 15. Pyramidal geometry for branch surface generation

To create a smooth tree surface, we employ the Catmull-Clark subdivision scheme [8] that creates a subdivision surface per *stem* in the tree model. The chain of intersected pyramidal frustums representing *segments* of a *stem* serve as our initial mesh in the subdivision scheme (see Figure 16 (left)). This initial coarse mesh is then refined using the Catmull-Clark subdivision scheme. Refer Figure 17 displaying

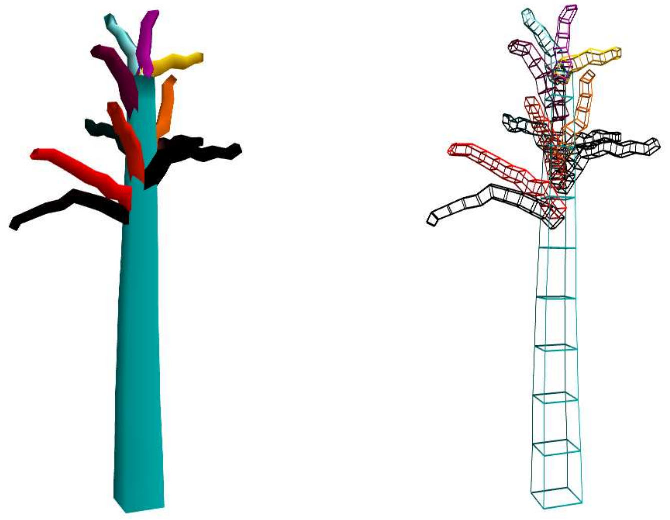


FIGURE 16. Initial mesh for Catmull-Clark subdivision surface

a quad mesh used in the subdivision process that carries out following steps:

1. For each face in the mesh, calculate a face point as

$$\mathbf{F} = \frac{1}{4}(\mathbf{P_0} + \mathbf{P_1} + \mathbf{P_6} + \mathbf{P_2}) \tag{1}$$

2. For each edge in the mesh, calculate an edge point as

$$\mathbf{E} = \frac{3}{8}(\mathbf{P_0} + \mathbf{P_1}) + \frac{1}{16}(\mathbf{P_4} + \mathbf{P_5} + \mathbf{P_6} + \mathbf{P_2}) \tag{2}$$

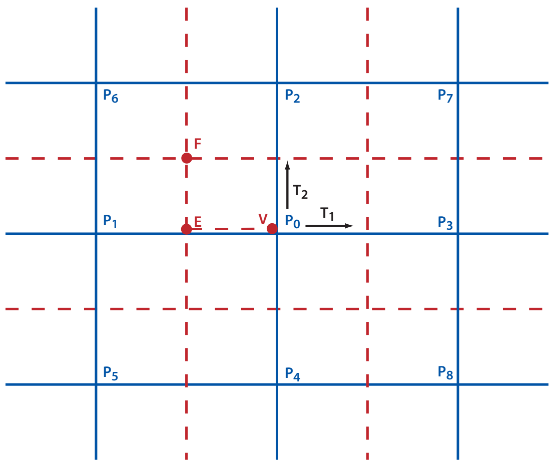
3. For each vertex in the mesh, calculate a vertex point as

$$\mathbf{V} = \frac{9}{16}\mathbf{P_0} + \frac{3}{32}(\mathbf{P_1} + \mathbf{P_2} + \mathbf{P_3} + \mathbf{P_4}) + \frac{1}{64}(\mathbf{P_5} + \mathbf{P_6} + \mathbf{P_7} + \mathbf{P_8}) \tag{3}$$

4. Form new faces using the newly calculated face points, edge points and vertex points

Normals at each new vertex are calculated as

$$\mathbf{N} = \mathbf{T_1} \times \mathbf{T_2} \tag{4}$$



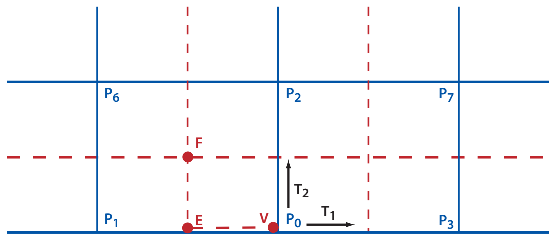


FIGURE 17. Catmull-Clark subdivision scheme

where

$$\mathbf{T_1} = -4\mathbf{P_1} + 4\mathbf{P_3} - (\mathbf{P_5} + \mathbf{P_6}) + (\mathbf{P_7} + \mathbf{P_8}) \tag{5}$$

and

$$\mathbf{T_2} = 4\mathbf{P_2} - 4\mathbf{P_4} - (\mathbf{P_5} + \mathbf{P_8}) + (\mathbf{P_6} + \mathbf{P_7}) \tag{6}$$

When the above steps are followed for all the *stems*, a refined tree mesh is generated as shown in Figure 18 (left). But the side effect of this is that bottom and top edges become rounded in the tree geometry creating an undesirable result as shown in Figure 18 (middle). To correct this, we treat bottom and top edges, and vertices on those edges as special cases (see Figure 17 (right)) and apply the following formulae:

1. For each edge on the boundary of the mesh, calculate an edge point as

$$\mathbf{E} = \frac{1}{2}(\mathbf{P_0} + \mathbf{P_2}) \tag{7}$$

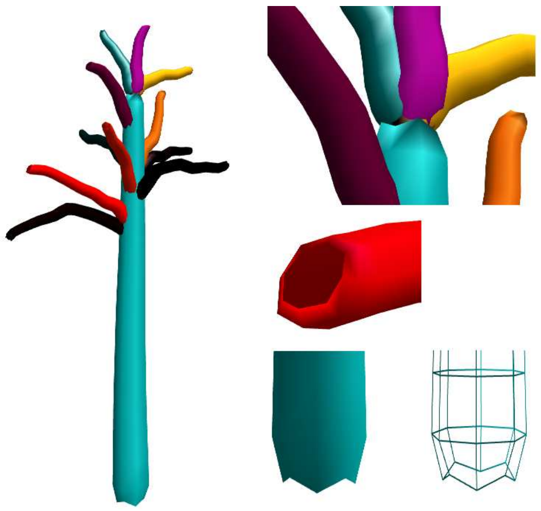


FIGURE 18. Problems with Catmull-Clark general case

2. For each vertex on the boundary of the mesh, calculate a vertex point as

$$\mathbf{V} = \frac{1}{8}(\mathbf{P_1} + \mathbf{P_2}) + \frac{3}{4}\mathbf{P_0} \tag{8}$$

Tangents $\mathbf{T_1}$ and $\mathbf{T_2}$ are calculated as

$$\mathbf{T_1} = \frac{-2}{3}\mathbf{P_0} + \frac{-1}{6}(\mathbf{P_1} + \mathbf{P_2}) + \frac{1}{6}(\mathbf{P_3} + \mathbf{P_5}) + \frac{2}{3}\mathbf{P_4} \tag{9}$$

and

$$\mathbf{T_2} = \mathbf{P_1} - \mathbf{P_2} \tag{10}$$

Rest of the steps are same as the general case. This fixes the problem as shown in Figure 19 creating the desired tree geometry. Note that as more iterations are performed, finer meshes are generated making the tree surface smoother as shown in Figure 19 (right).

We can extend this approach by considering the pyramidal geometric model of the entire tree as our initial mesh and applying the subdivision scheme to it, creating

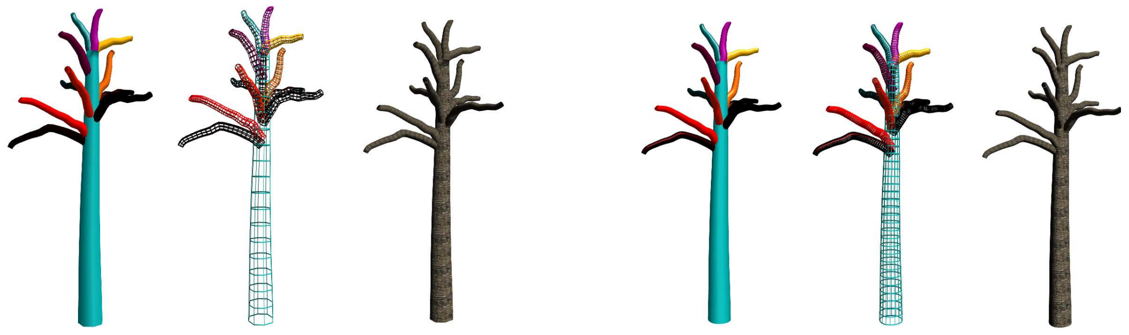


FIGURE 19. Tree surface after Catmull-Clark iterations

smooth branch junctions. But this makes the mesh more generic with one vertex being shared by more than four faces making the geometry creation more complex.

The Catmull-Clark subdivision surface created with the above steps is smooth at *segment* junctions with no gaps between them. But when the mesh is coarse, facets of the branch geometry are visible in the close-up views. If we refine the geometry iteratively so that the surface is smoother, then the amount of geometry information increases affecting the rendering speed. Another problem of making more iterations is the shrinkage of the original mesh increasing the difference between attributes specified by the user and the actual geometry. To overcome these problems, we take another approach to create the branch surfaces.

### 3.3.3. GPU Based Ray Tracer

To create a smooth surface even in the close-up views, we ray trace the *segments* instead of using a quad mesh to represent them. The ray tracer reads the chain of intersected pyramidal frustums created as described in section 3.3.2 and traces conical frustums that fit in the pyramids (see Figure 20).

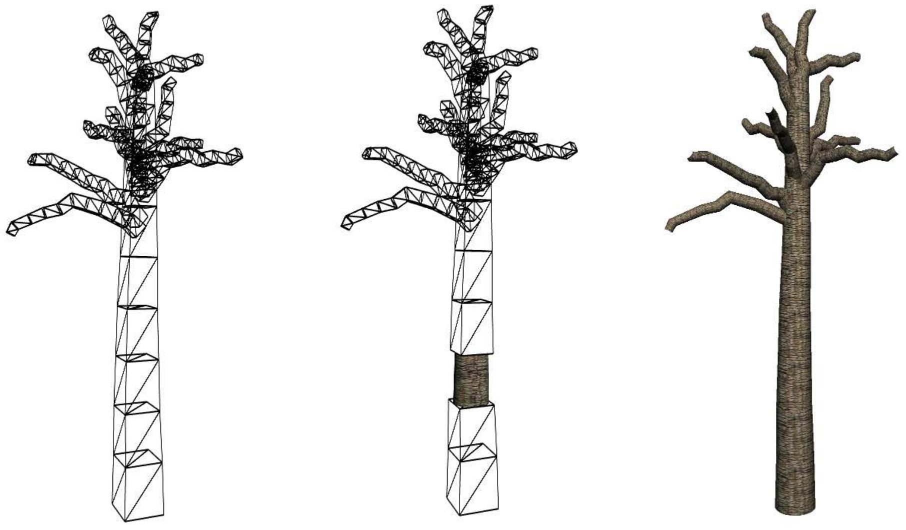


FIGURE 20. Ray tracing one frustum of a tree

Let us see how this is done for a single *segment* in the tree model. The *segment* when instantiated by a pyramid looks like Figure 21 (top left). To render surface of a cone that fits the pyramid (see Figure 21 (bottom left)), we use implicit equation of a cone. Assuming y axis as the up vector, a cone equation as a function $f(x, y, z)$ is given as

$$x^2 + z^2 = (y - h)^2 \frac{r^2}{h^2} \tag{11}$$

where $r$ is base radius and $h$ is height of the cone.

To ray trace a cone, following ray equation is used

$$\mathbf{P}(t) = \mathbf{O} + t\mathbf{D}, t \geq 0 \tag{12}$$

where $\mathbf{O} = (O_x, O_y, O_z)$ is the eye point and $\mathbf{D} = (D_x, D_y, D_z)$ is the view direction. To find an intersection between the cone and the ray, substitute the ray equation in the cone equation

$$(O_x + tD_x)^2 + (O_z + tD_z)^2 = (O_y + tD_y - h)^2 \frac{r^2}{h^2} \tag{13}$$

and solve for t

$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{14}$$

where $a = D_x^2 + D_z^2 - D_y^2 \frac{r^2}{h^2}$,

$b = 2O_xD_x + 2O_zD_z - 2D_y(O_y - h)\frac{r^2}{h^2}$ and

$c = O_x^2 + O_z^2 - (O_y - h)^2 \frac{r^2}{h^2}$

The point on the cone is then calculated using equation 12. If $b^2 - 4ac$ is less than an epsilon value, we discard the pixel. The resulting cone looks like the Figure 21 (bottom left).

Once we calculate a point, $\mathbf{P}$ on the cone surface, we calculate normal, $\mathbf{N}$ at that point as

$$\mathbf{N} = (f_x(\mathbf{P}), f_y(\mathbf{P}), f_z(\mathbf{P})) \tag{15}$$

where $f_x = \frac{\partial f}{\partial x}, f_y = \frac{\partial f}{\partial y}, f_z = \frac{\partial f}{\partial z}$

For a cone, using equation 11,

$$\mathbf{N} = (2x, -2\frac{r^2}{h^2}(y - h), 2z) \tag{16}$$

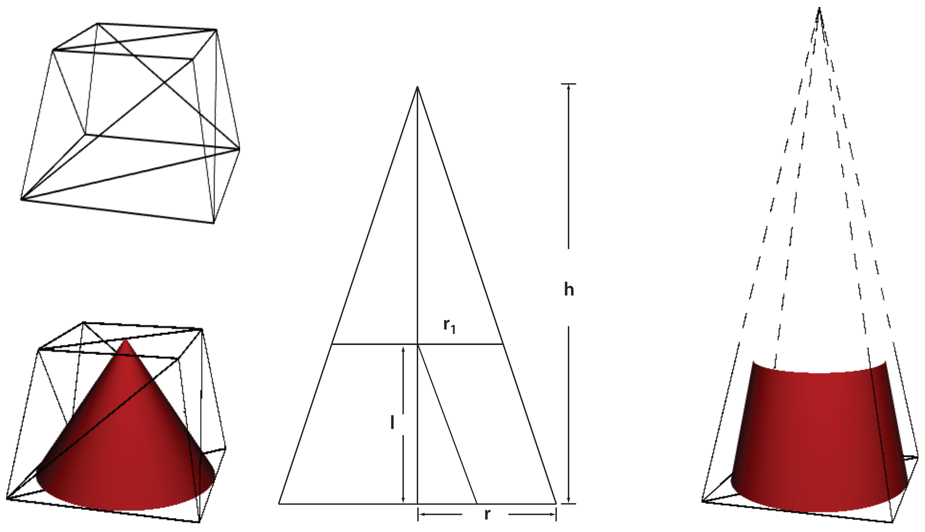


FIGURE 21. Ray tracing using implicit equation

We also calculate texture coordinates at that point as

$$u = \frac{\cos^{-1}\theta}{2\Pi} \tag{17}$$

where $\theta = \mathbf{X}.\mathbf{N}$, $\mathbf{X}$ is the positive X axis and

$$v = \frac{P_y}{l} \tag{18}$$

where $l$ is the *segment* length.

However, since we want to fit a frustum of the cone and not the whole cone in the pyramid, we extrapolate the *segment* length to calculate cone height as follows:

$$h = \frac{r * l}{r - r_1} \tag{19}$$

where $r_1$ is the top radius of the conical frustum and $l$ is the length of the *segment*. We then discard the pixels below and above the *segment* length to render a finite frustum as shown in Figure 21 (right).

When this technique is applied to all the *segments* of a *stem*, the surface looks smooth with no facets even in the close-up views. But as shown in Figure 22 (left), gaps between the adjacent *segments* are still visible. This is because even though we update vertices of the pyramidal frustum as described in section 3.3.2, we still use the *segment* length to discard pixels while tay tracing the cone. Instead, we need to use some other technique for discarding pixels so that we do not see any gaps between the adjacent *segments*. We use planes formed by the intersection points of adjacent pyramids for cutting the cone to render correct conical frustums as shown in Figure 22 (right).

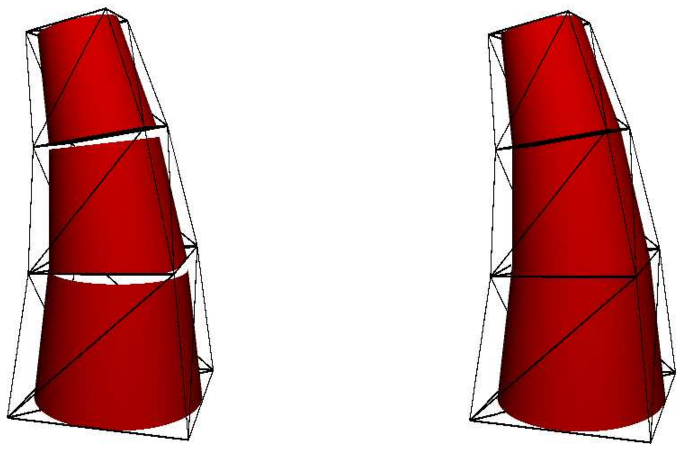


FIGURE 22. Conical frustum with cutting planes

With this new scheme, texture coordinate $v$ specified by equation 18 can go beyond 0 or 1. For correct texture mapping, we update this value as

$$v = \begin{cases} 1 + v & \text{if } x < 0, \\ v - 1 & \text{if } x > 1. \end{cases} \tag{20}$$

Texture coordinate $u$ and normal are calculated as specified earlier by equations 17 and 15 respectively.

This approach creates a smooth surface with very little geometry transfer from the application to the graphics card. The triangles represented using the black lines indicate the actual geometry transferred. The tree models described in section 3.2 are ray traced using this approach.

## 3.4. Detail Rendering

The preceding section describes our method to generate a smooth tree surface. However, trees found in natural environments often possess structural irregularities on their bark (see Figure 23). Additionally, trees also cast shadows on the ground. Realistic tree rendering should therefore support (a) surface detail and (b) shadow generation. In this section we present a method to generate surface details. We also describe the shadow mapping algorithm used to generate tree shadows.

### 3.4.1. Bark Details

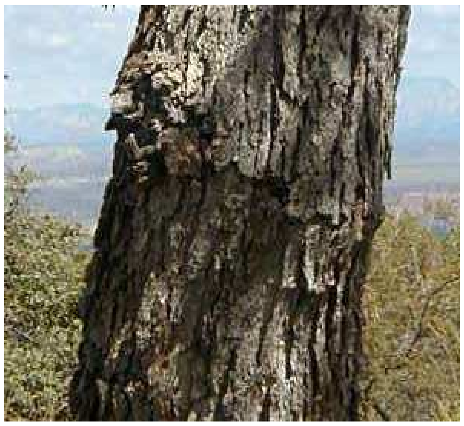


FIGURE 23. Photograph of a real tree branch (Courtesy: Google Images)

Figure 23 shows a photograph of a real tree branch in a close-up view. Although the branch is conical, the surface is detailed with rough and irregular silhouette. The ray tracing approach we described in section 3.3.3 creates smooth surfaces with straight edges. To render details on top of these smooth surfaces without perturbing the geometry, we use a texture map approach. The following sections describe how we use the normal mapping technique to add details to the ray traced surface and how it evolves into the relief mapping to create a realistic silhouette.

3.4.1.1. *Normal Mapping.* Normal mapping [28] simulates bumps in a surface by perturbing the surface normal used for shading. This is accomplished by storing the perturbed normals for the surface as a normal map. Then as shown in Figure 24 for each fragment of the surface, the perturbed normal is fetched from this map. We store the normals in a conventional RGB texture map whose components are in

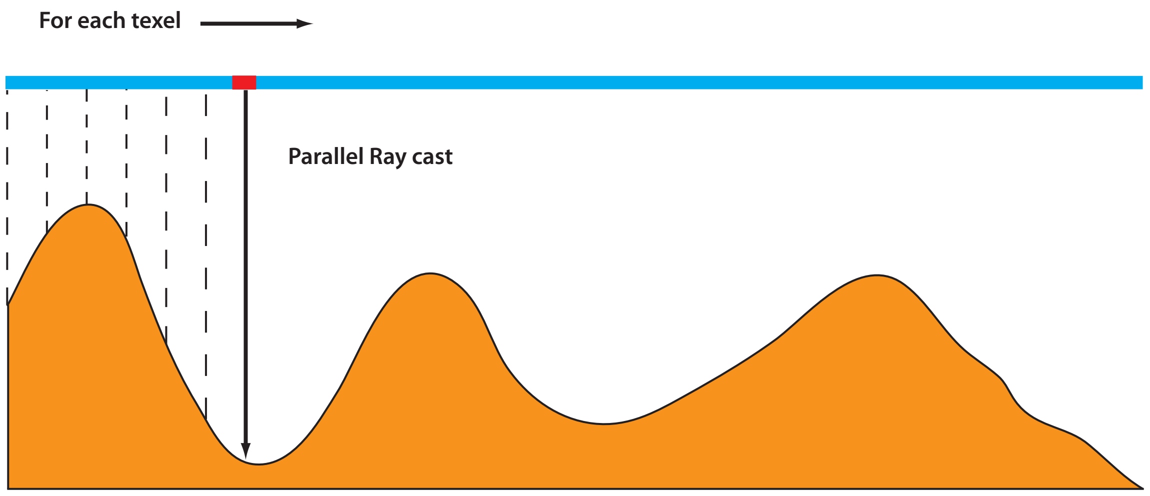


FIGURE 24. Normal mapping technique

the range [0, 1]. However, the normals which are normalized, are in the range [-1, 1]. Hence we use a scale and a bias to compress these into the range [0, 1] using equation 21

$$\mathbf{N_c} = 0.5\mathbf{N} + 0.5 \tag{21}$$

where $N$ is the perturbed normal and $N_c$ is the compressed normal stored in the normal map. After fetching from the normal map, the normals are decompressed using equation 22

$$\mathbf{N} = 2.0\mathbf{N_c} - 1.0 \tag{22}$$

The normals stored in the normal map are in the tangent space of the surface. Hence to use them in the shading equation, the light direction vector **L** and view vector **D** are converted to tangent space using a conversion matrix

$$\begin{pmatrix} T_x & B_x & N_x \\ T_y & B_y & N_y \\ T_z & B_z & N_z \end{pmatrix}$$

where **N** is the fragment normal,

**T** is the fragment tangent calculated as $(\frac{\partial x}{\partial u}, \frac{\partial y}{\partial u}, \frac{\partial z}{\partial u})$

**B** is the binormal calculated as $(\frac{\partial x}{\partial v}, \frac{\partial y}{\partial v}, \frac{\partial z}{\partial v})$

where

$$x = \frac{h-v}{h} r \cos(2\Pi u) \tag{23}$$

$$y = v \tag{24}$$

$$z = \frac{h-v}{h} r \sin(2\Pi u) \tag{25}$$

are cone equations in terms of $u$ and $v$.

Therefore,

$$\mathbf{T} = (2\Pi rk \sin(2\Pi u), 0, 2\Pi rk \cos(2\Pi u)) \tag{26}$$

$$\mathbf{B} = (\frac{-r}{h} cos(2\Pi u), 1, \frac{-r}{h} sin(2\Pi u)) \tag{27}$$

where $k = \frac{h-v}{h}$

Figure 25 shows a conical frustum before and after the normal mapping. Figure 26

FIGURE 25. Normal mapped frustum

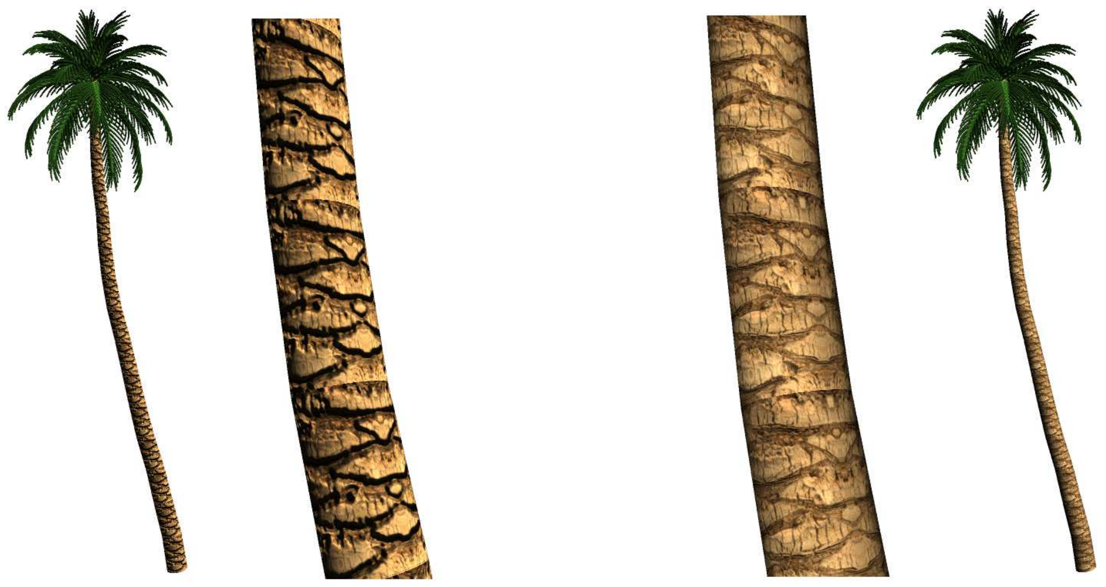


FIGURE 26. Comparison of normal mapped and texture mapped coconut tree

shows comparison of the normal mapped tree trunk with the flat texture mapped bark. It can be seen that the normal mapped surface looks as if it possesses a physical texture. However, since we are not perturbing the geometry, the effect is absent along silhouette edges.

3.4.1.2. *Relief Texture Mapping.* To overcome this limitation, we use relief texture mapping method introduced by M. Oliveira *et al.* [35]. Policarpo [36] extended the method to a triangle mesh of any arbitrary topology including curved surfaces. In the proposed approach, each vertex of the mesh is enhanced with a quadric surface locally approximating the objects geometry at the vertex. The quadric coefficients are computed during a pre-processing stage using a least-squares fitting algorithm and are interpolated during rasterization. We introduce a new relief mapping algorithm that can be applied to ray traced conical frustums and extended to any ray tracer using implicit equations. Our algorithm does not require any calculations in the pre-processing stage and is appropriate for real-time applications.

We consider that inside the conical frustum representing a *segment*, there lies another infinite conical frustum, concentric with the outer one, distance between the two equal to the silhouette width, specified by the user.

Figure 27 (left) shows side view and (right) shows top view of the concentric conical frustums centered at point **C** with the view rays. We highlight one of the view rays in orange for explaining our algorithm. The two points closest to the camera, at which the ray hits the conical frustums are called $ht_1$ and $ht_2$ as shown in Figure 27 (right). We divide the angular space between these two hit points linearly. Then at each step (see Figure 28), we compare the depth looked up in the relief map texture,

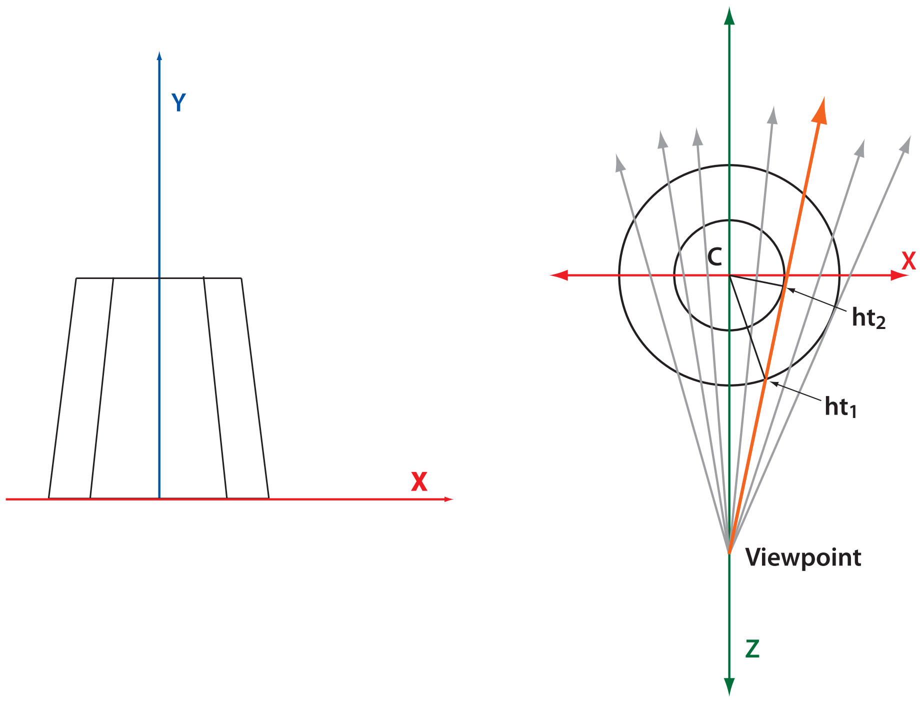


FIGURE 27. Relief mapping using concentric frustums

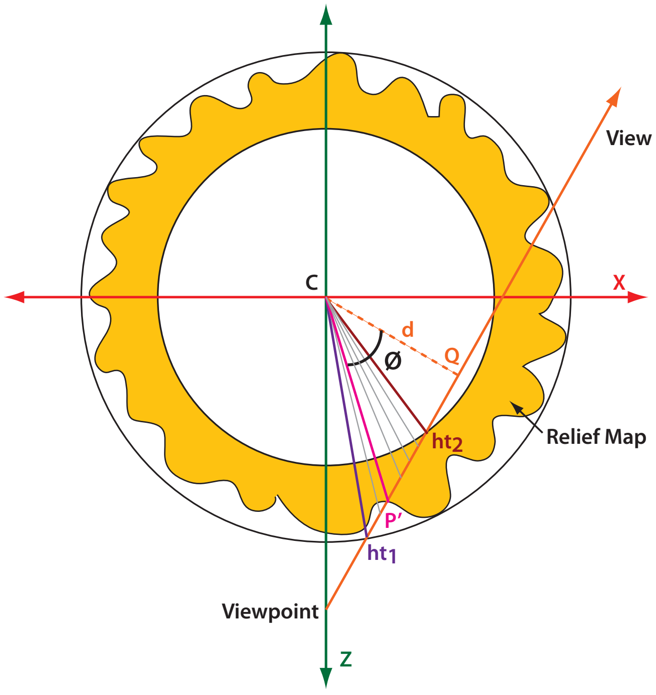


FIGURE 28. Relief mapping: Detailed view

$d_t$ with the depth of the view ray, $d_r$ and use a normal from the displacement map where $d_r = d_t$. $d_r$ is calculated as

$$d_r = \frac{l_r - r_i}{r_o - r_i} \tag{28}$$

where $l_r = |\overrightarrow{CP'}|$ and

$P'$ is a point on the view ray for the current step.

$l_r$ is calculated as

$$l_r = d\cos\phi \tag{29}$$

$d = |\overrightarrow{CQ}|$, distance to the view ray from the frustum axis,

$\phi$ is angle between $\overrightarrow{CQ}$ and $\overrightarrow{CP'}$,

$r_i$ is the radius of the inner conical frustum at $ht_2$ and

$r_o$ is the radius of the outer conical frustum at $ht_1$

If the view ray does not intersect the inner conical frustum, $ht_2$ is considered as the second intersection point of the view ray with the outer conical frustum. We discard the fragments for which view ray does not intersect the relief map, rendering the correct silhouette. Figure 29 shows a conical frustum before and after the relief mapping.

Figure 30 shows comparison of the normal mapped tree trunk with the relief mapped bark correctly displaying the silhouette. Figure 31 shows the relief mapped bark of the pine tree.

FIGURE 29. Relief Mapped frustum

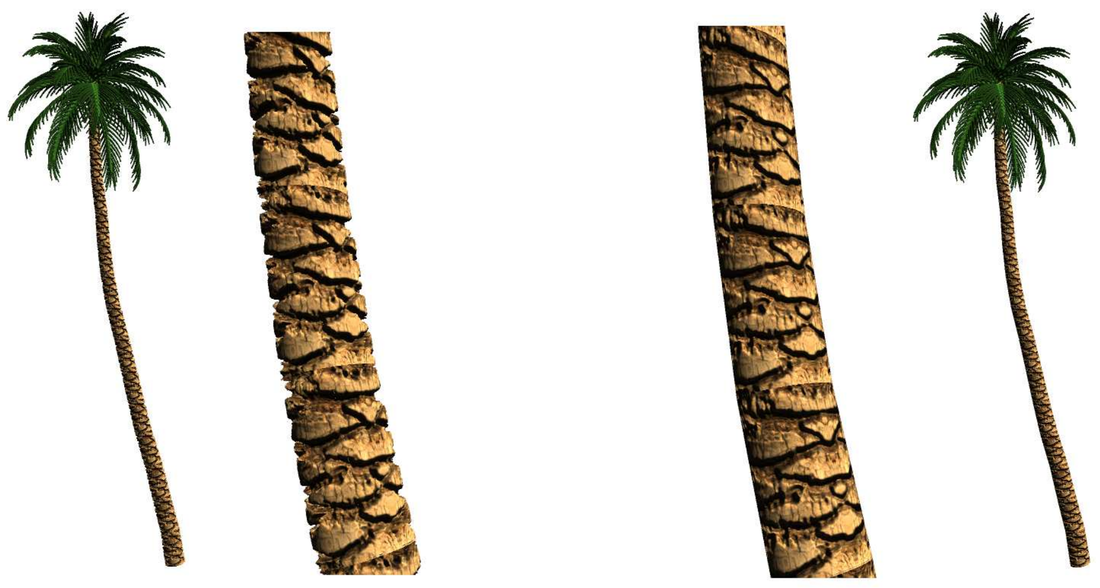


FIGURE 30. Comparison of relief mapped and normal mapped coconut tree

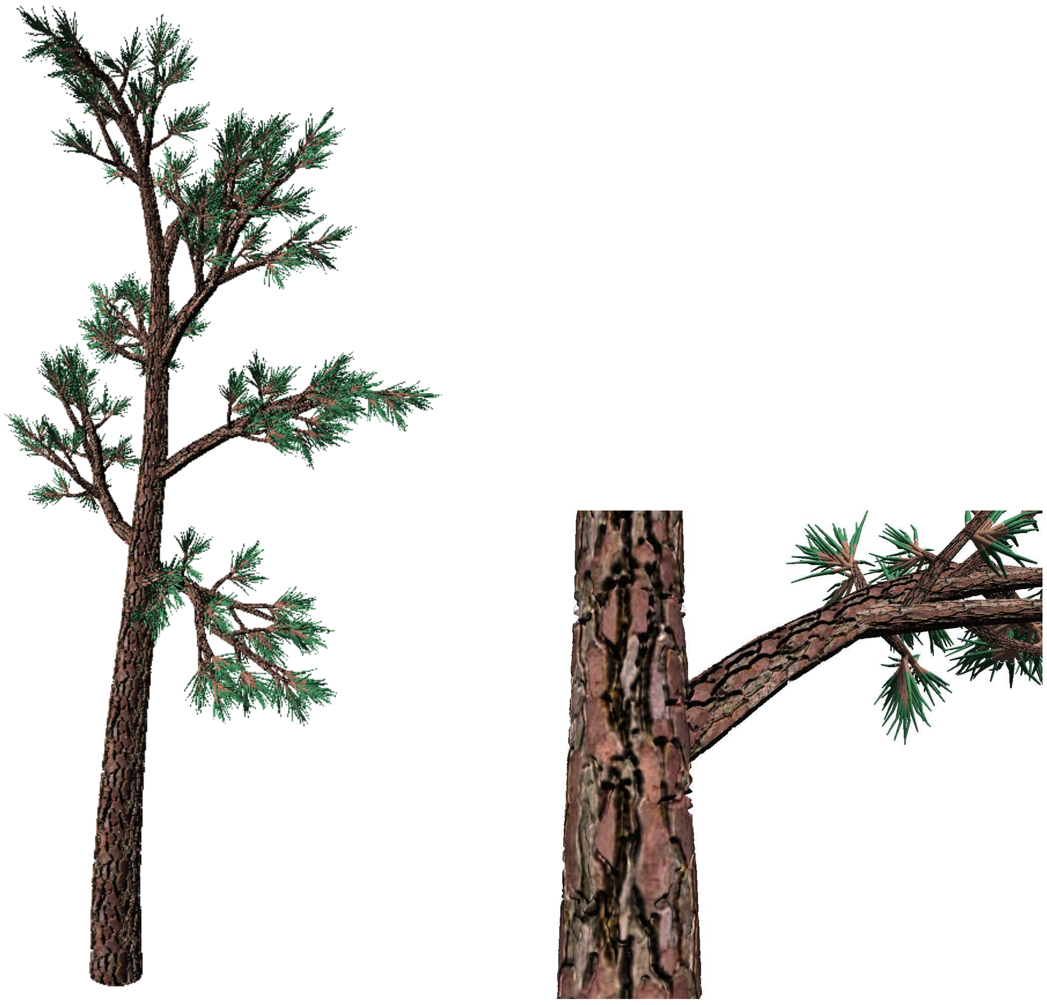


FIGURE 31. Relief mapped pine tree

### 3.4.2. Shadows

Shadows are an important element in achieving realistic rendering. They impart important clues that inform us of the relative positions of objects and light sources.

We use the shadow mapping technique [20] to render shadows casted by trees on the ground. Shadow map is a depth map rendered from the point of view of the light source, created in the first pass of rendering. In the second pass, we consider that this depth map is projected onto the ground. For each fragment, $P$ as shown in Figure 32, we compare its depth with the depth stored in the shadow map. If

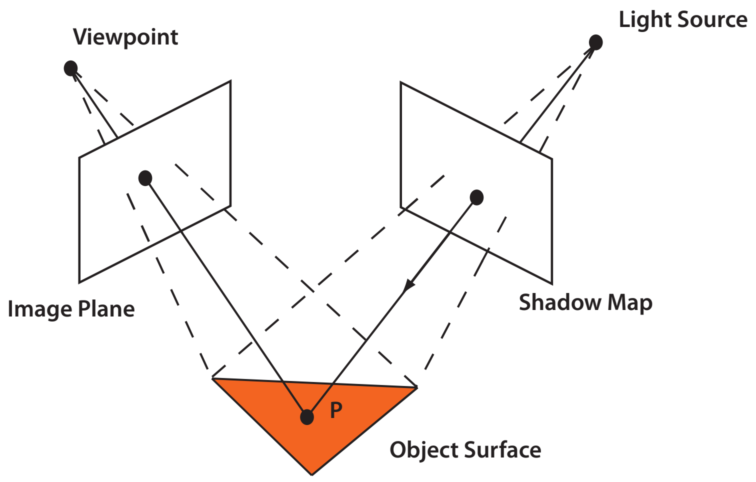


FIGURE 32. Projective texture for shadow mapping

the depth of $P$ is greater than the shadow map depth then there is an object closer to the light source, and $P$ is in shadow. To project the depth map, we calculate $(u, v)$ coordinates using the fragment $P$. In our case, we do this by transforming the ground geometry from world space to light space. This is done as follows:

$$\begin{pmatrix} u \\ v \\ z \\ q \end{pmatrix} = \begin{pmatrix} 0.5 & & & 0.5 \\ & 0.5 & & 0.5 \\ & & 0.5 & 0.5 \\ & & & 1 \end{pmatrix} \begin{pmatrix} \\ Light \\ Space \\ Transformation \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ P_z \\ P_w \end{pmatrix}$$

where the first matrix multiplication scales values in the range $[-1, 1]$ to $[0, 1]$ required for texture coordinates. Figure 33 shows a scene with a tree whose shadow is rendered on the ground.

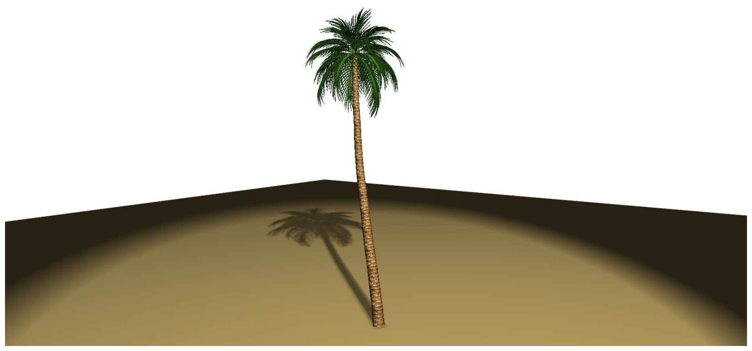


FIGURE 33. Coconut tree shadow

One of the problems of the shadow mapping technique described above is aliasing. It arises from the inaccuracies in depth representation and point sampling, manifesting as jagged shadow borders (see Figure 35). In the point sampling problem, projection of a pixel onto the object surface and into the shadow map produces a footprint in the shadow map which may include many texels (see Figure 34). The area of this footprint is a function of the transformation and the inclination of the surface. To make a correct in-shadow/not in-shadow decision we need to integrate the

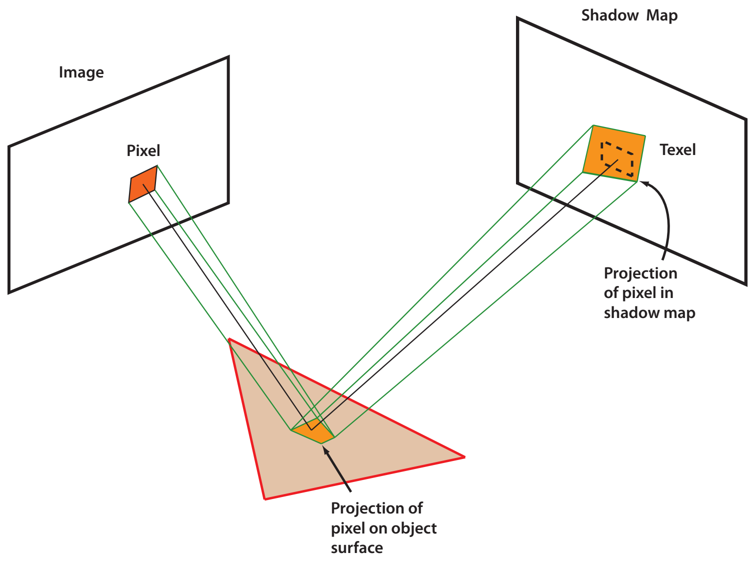

Shadow Map
Image
Pixel
Texel
Projection of pixel in shadow map
Projection of pixel on object surface

FIGURE 34. Shadow mapping aliasing problem

information over the footprint. But simply averaging the shadow map texels would result in wrong depth values. Instead we use percentage closer filtering [29] to average the results of the boolean comparisons by fetching the surrounding texels. Figures 35 and 36 compare coconut tree shadow before and after percentage closer filtering respectively. We use a neighborhood of size 3x3 for percentage closer filtering.

Compared to the shadow volume approach, shadow mapping allows rendering shadows cast by triangulated as well as ray traced objects. Also, this technique is efficient as it requires only two passes and there is no extra geometry to draw.

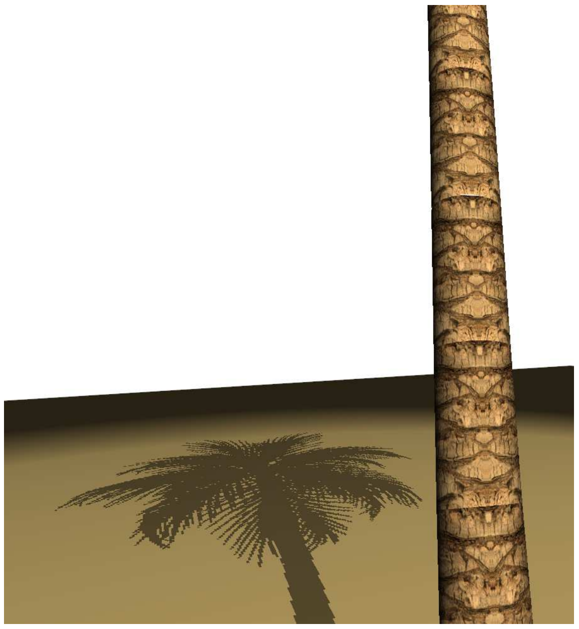


FIGURE 35. Aliased shadow of coconut tree

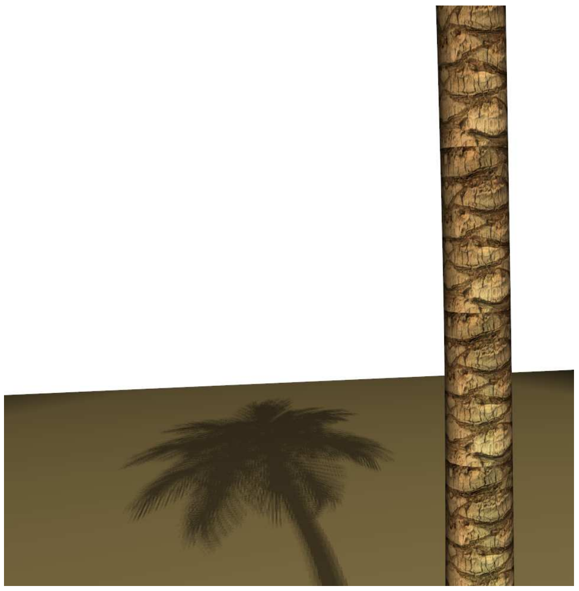


FIGURE 36. Anti-aliased shadow of coconut tree

# 4. RESULTS

Figures 37, 38, 39 and 40 show output of our tree visualization framework, and are rendered on a NVIDIA GeForce 8800 GT video card with 512 MB RAM. The window size is 1024×1024 on a 2.4 GHz Intel Core 2 Quad CPU with 3 GB RAM. Our framework is implemented in C++ and OpenGL with the shader code in GLSL.

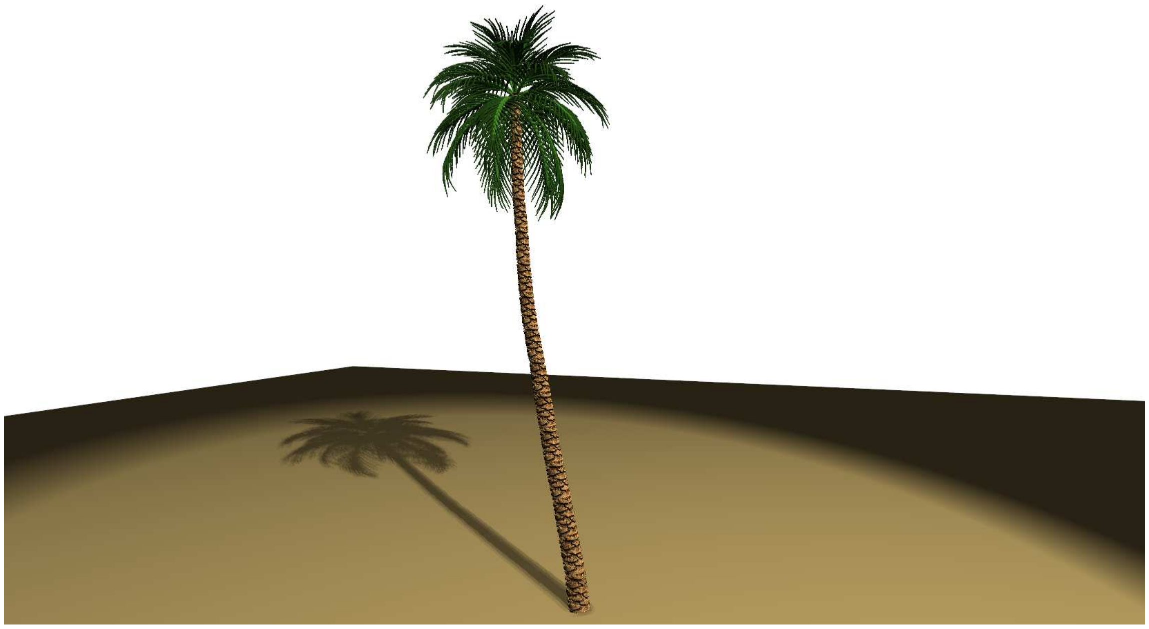


FIGURE 37. Coconut tree: final rendering

TABLE 1. Tree modeling statistics

| Model | No. of Terminal Shapes | $T_m$(msec) |
|---|---:|---:|
| Aloe Vera Bush | 242 | 65 |
| Elm Tree | 1,869 | 317 |
| Coconut Tree | 4,243 | 764 |
| Pine Tree | 16,780 | 2196 |
| Test Scene | 26,004 | 3681 |

Our procedural grammar engine reads a rule set and variable values specified by the user. It then applies these rules according to the rule priorities to create a tree

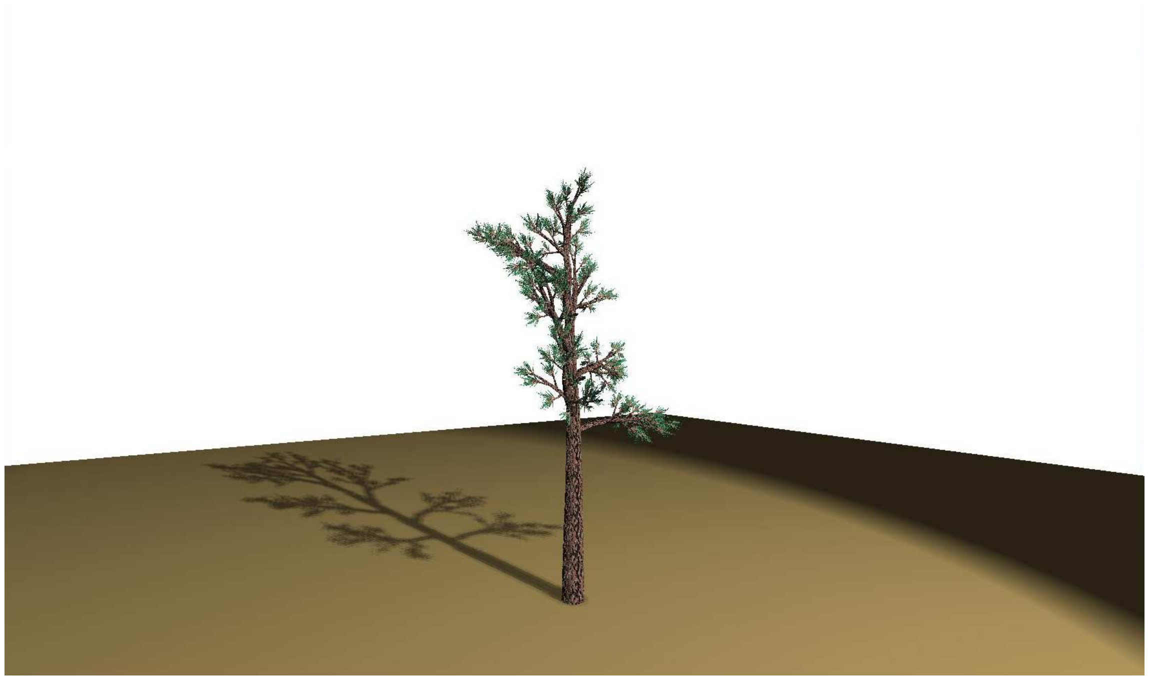


FIGURE 38. Pine tree: final rendering

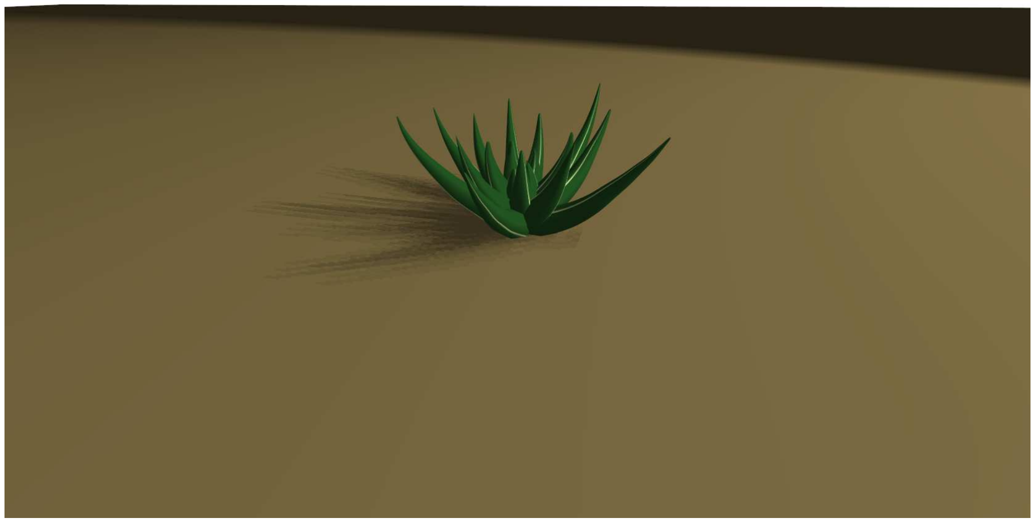


FIGURE 39. Aloe vera bush: final rendering

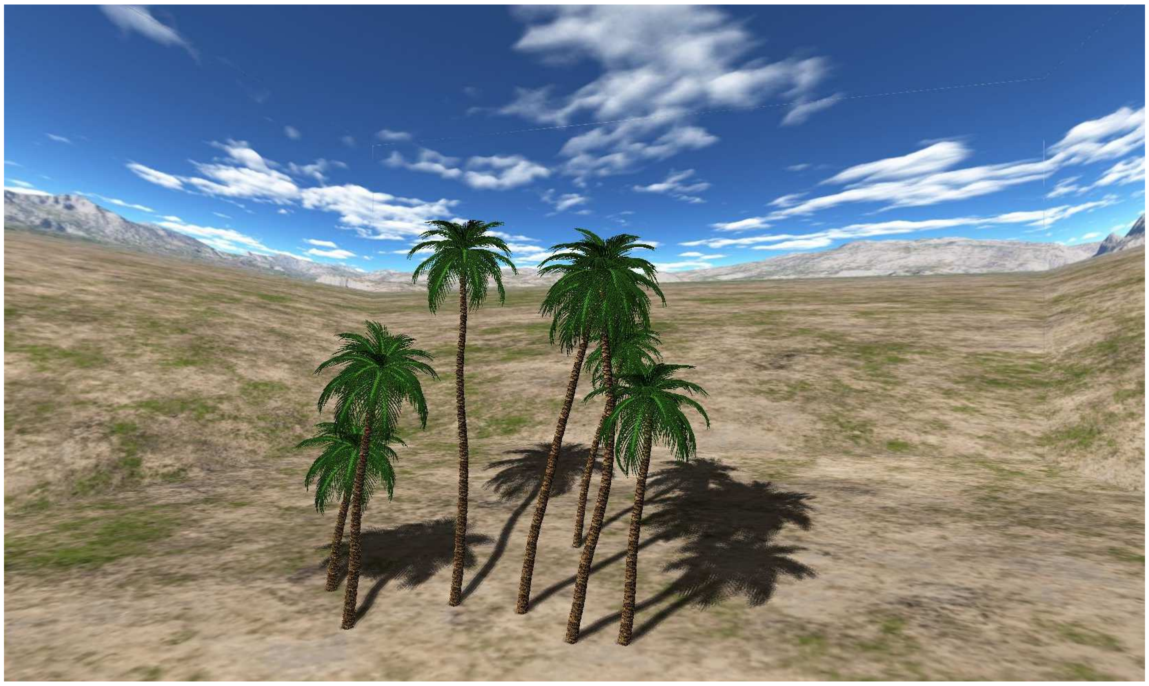


FIGURE 40. A virtual oasis with seven coconut trees

model. The derivation proceeds in a top-down fashion, terminating at the terminal symbols that instantiate the geometric attributes of a particular shape. The output of the derivation is a tree model composed of *segments*. Table 1 lists time, $T_m$, required to model different trees and number of *segments* in each model.

Once the tree modeling is complete, we generate the tree surface to represent the above tree model graphically. The tree surface is generated by either (a) using mesh created by OpenGL cone primitive, (b) performing Catmull-Clark subdivision scheme on a coarse mesh, or (c) by ray tracing the chain of triangulated pyramidal frustums. Table 2 lists the number of triangles and time ($T_s$) required to generate the surface using the OpenGL and ray tracing approaches. Table 3 similarly lists statistics for the Catmull-Clark subdivision scheme with different number of iterations. The tables also list rendering speed in frames per second.

TABLE 2. Surface generation statistics for OpenGL primitive and ray tracing

| Model | OpenGL Primitive | | | Ray Traced | | |
|---|---|---|---|---|---|---|
| | Triangles | $T_s$(msec) | fps | Triangles | $T_s$(msec) | fps |
| Aloe Vera Bush | 3,872 | 0 | 779 | 2,904 | 16 | 889 |
| Elm Tree | 29,904 | 0 | 125 | 22,428 | 63 | 522 |
| Coconut Tree | 67,888 | 0 | 64 | 50,916 | 94 | 392 |
| Pine Tree | 268,480 | 0 | 15 | 201,360 | 514 | 128 |
| Test Scene | 416,064 | 0 | 9 | 312,048 | 765 | 87 |

TABLE 3. Surface generation statistics for Catmull-Clark subdivision scheme

| Model | Catmull-Clark (1 itr) | | | Catmull-Clark (2 itr) | | |
|---|---|---|---|---|---|---|
| | Triangles | $T_s$(msec) | fps | Triangles | $T_s$(msec) | fps |
| Aloe Vera Bush | 7,744 | 62 | 945 | 30,848 | 172 | 740 |
| Elm Tree | 59,808 | 452 | 645 | 239,232 | 4087 | 365 |
| Coconut Tree | 135,776 | 858 | 509 | 543,104 | 6193 | 213 |
| Pine Tree | 536,960 | 8596 | 189 | 2,147,840 | 22,120 | 64 |
| Test Scene | 951,104 | 21,170 | 116 | 3,801,600 | 47,970 | 17 |

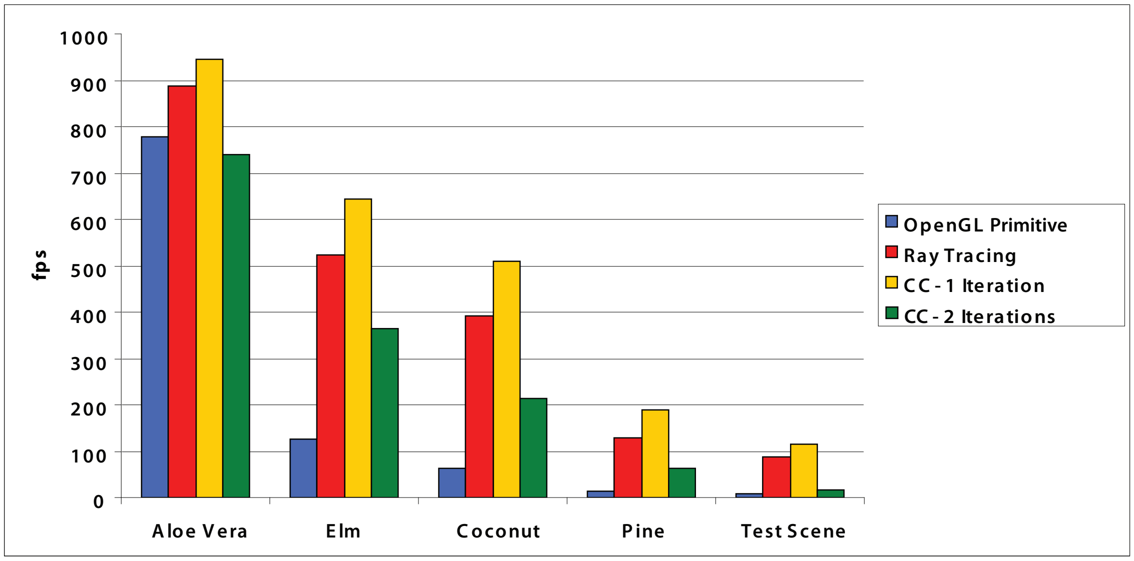


FIGURE 41. Frames per second to render tree models using various surface types

TABLE 4. Rendering statistics

| Model | Shadow Mapped (fps) | Relief Mapped (fps) | Relief and Shadow (fps) |
|---|---:|---:|---:|
| Aloe Vera Bush | 482 | n/a | n/a |
| Elm Tree | 282 | n/a | n/a |
| Coconut Tree | 200 | 355 | 192 |
| Pine Tree | 67 | 116 | 64 |
| Test Scene | 46 | 64 | 41 |

Tables 2 and 3 are used to plot the graph shown in Figure 41 comparing frames per second (fps) for rendering various tree models using the surfaces types described earlier. As seen in the bar graph, OpenGL primitive method does not provide real-time frame rates for all the tree models. Catmull-Clark subdivision method with one iteration is the fastest. However, the generated surface is faceted. Executing another iteration of the same surface gives a finer surface, but the frame rate declines considerably. GPU based ray tracer provides the best trade-off between fps and visually aesthetic results.

In the third stage of our framework, bark details are added to the ray traced surface using our relief mapping technique. Shadows are also rendered for the mesh surface as well as the ray traced surface. Table 4 lists the frame rates for the ray traced surface generation case. Since shadow mapping is an image space approach, we obtain similar statistics for the other surfaces. The table shows that relief mapping reduces the frame rate marginally, however, shadow mapping makes a greater impact on the frame rate.

# 5. CONCLUSION AND FUTURE WORK

This thesis presents a shape grammar for the procedural modeling of trees at interactive speeds. The grammar rules allow creation of a wide variety of actual tree species showcasing their specific characteristics. The method requires no modeling expertise or botanical knowledge and is capable of generating detailed tree models suitable for close-up views. We demonstrate this with several examples. The thesis is the first to use implicit equations to ray trace tree surface on the GPU with very little geometric data. It also presents a comparison with two other geometry creation approaches and shows that the ray tracing approach is better to create smooth, non-faceted surfaces at faster speed. The thesis also describes a relief mapping algorithm to render subtle details such as bark roughness and silhouette edges. The technique works with non-triangulated surfaces at real-time frame rates and does not require estimating surface curvature using quadrics as in the previous method. It implements shadow mapping technique for added realism.

The proposed shape grammar approach requires playing around with the rules and initial values to model close match of a real tree. Also, as we ray trace each individual *segment* in the tree model, trees with large number of *segments* suffer from frame rate reduction.

The methods presented in this thesis can be extended to include:

1. Rule and model editor to modify grammar rules and variables at run-time
2. Wind animation
3. Illumination effects for leaves
4. Creation of massive forests that can be rendered in real-time

## REFERENCES

[1] Speedtree. Modeling Editor by IDV, Inc, http://www.speedtree.com.

[2] Treemaker. a tree and plant solution by Onyx Computing. http://www.speedtree.com.

[3] Xfrog. an interactive modeling and animation system for organic structures with focus on plants by Greenworks Organic Software. http://www.xfrog.com.

[4] M. Aono and T. Kunii. Botanical tree image generation. *IEEE Comput. Graph. Appl.*, 4(5):10–34, 1984.

[5] J. F. Blinn. Simulation of wrinkled surfaces. *SIGGRAPH Comput. Graph.*, 12(3):286–292, 1978.

[6] J. Bloomenthal. Modeling the mighty maple. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 305–311, New York, NY, USA, 1985. ACM Press.

[7] E. Bromberg-Martin, A. M. Jónsson, G. E. Marai, and M. McGuire. Hybrid billboard clouds for model simplification. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Posters*, page 30, New York, NY, USA, 2004. ACM Press.

[8] E. Catmull and J. Clark. Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 10(10):350–355, September 1978.

[9] C. Colditz, L. Coconu, O. Deussen, and H.-C. Hege. Real-time rendering of complex photorealistic landscapes using hybrid level-of-detail approaches. In E. Buhmann, P. Paar, I. Bishop, and E. Lange, editors, *Trends in Real-Time Landscape Visualization and Participation*, pages 97–106. Wichmann Verlag, 2005.

[10] F. C. Crow. Shadow algorithms for computer graphics. In *In Computer Graphics (Proceedings of SIGGRAPH 77)*, volume 11, pages 242–248. ACM Press, 1977.

[11] M. D. Davis, R. Sigal, and E. J. Weyuker. *Computability, complexity, and languages (2nd ed.): fundamentals of theoretical computer science.* Academic Press Professional, Inc., San Diego, CA, USA, 1994.

[12] P. Decaudin and F. Neyret. Rendering forest scenes in real-time. In *Rendering Techniques (Eurographics Symposium on Rendering - EGSR)*, pages 93–102, june 2004.

[13] X. Décoret, F. Durand, F. Sillion, and J. Dorsey. Billboard clouds for extreme model simplification. In *Proceedings of the ACM Siggraph*. ACM Press, 2003.

[14] O. Deussen, P. Hanrahan, B. Lintermann, R. Měch, M. Pharr, and P. Prusinkiewicz. Realistic modeling and rendering of plant ecosystems. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 275–286, New York, NY, USA, 1998. ACM Press.

[15] O. Deussen and B. Lintermann. A modelling method and user interface for creating plants. In W. A. Davis, M. Mantei, and R. V. Klassen, editors, *Graphics Interface '97*, pages 189–198. Canadian Human-Computer Communications Society, 1997.

[16] F. Downing and U. Flemming. The bungalows of buffalo. *Environment and Planning B*, 8:269–293, 1981.

[17] J. Duarte. *Malagueira Grammar – towards a tool for customizing Alvaro Siza's mass houses at Malagueira.* PhD thesis, MIT School of Architecture and Planning, 2002.

[18] H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors. *Handbook of graph grammars and computing by graph transformation: vol. 2: applications, languages, and tools.* World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1999.

[19] C. Everitt and M. Kilgard. Practical and robust stenciled shadow volumes for hardware-accelerated rendering, 2002.

[20] C. Everitt, A. Rege, and C. Cebenoyan. Hardware shadow mapping, 2005.

[21] A. Fuhrmann, E. Umlauf, and S. Mantler. Extreme model simplification for forest rendering. Technical report TR VRVis 2004 039, Donau-City-Strasse 1, A-1220 Wien, 2004.

[22] J. K. Gábor Szijártó. Real-time hardware accelerated rendering of forests at human scale. In *Proceedings of the 12-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision'2004, WSCG*, 2004.

[23] C. Galbraith, P. MacMurchy, and B. Wyvill. Blobtree trees. In *CGI '04: Proceedings of the Computer Graphics International (CGI'04)*, pages 78–85, Washington, DC, USA, 2004. IEEE Computer Society.

[24] C. Galbraith, L. Mundermann, and B. Wyvill. Implicit visualization and inverse modeling of growing trees. *Computer Graphics Forum*, 23(3):351–360, 2004.

[25] R. L. Goldstone and J. Y. Son. The transfer of scientific principles using concrete and idealized simulations. *The Journal of The Learning Sciences*, 14(1):69–110, 2005.

[26] M. A. Janssen, R. L. Goldstone, F. Menczer, and E. Ostrom. Effect of rule choice in dynamic interactive spatial commons. *International Journal of the Commons*, In press.

[27] M. A. Janssen and E. Ostrom. Turfs in the lab: Institutional innovation in dynamic interactive spatial commons. *Rationality and Society*, In press.

[28] M. Kilgard. A practical and robust bump-mapping technique for today's gpu's, 2000.

[29] M. Kilgard. Shadow mapping with today's opengl hardware, 2001.

[30] D. E. Knuth. Semantics of context-free languages. *Theory of Computing Systems*, 2(2):127–145, June 1968.

[31] B. Lintermann and O. Deussen. Interactive modeling of plants. *IEEE Comput. Graph. Appl.*, 19(1):56–65, 1999.

[32] S. Mantler, G. Hesina, S. Maierhofer, and R. F. Tobler. Real time rendering of vegetation and trees in urban environments. In *International Symposium on Information and Communication Technologies in Urban and Space Planning (CORP 2005)*, 2005.

[33] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. V. Gool. Procedural modeling of buildings. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 614–623, New York, NY, USA, 2006. ACM Press.

[34] B. Neubert, T. Franken, and O. Deussen. Approximate image-based tree-modeling using particle flows. *ACM Trans. Graph.*, 26(3):88, 2007.

[35] M. M. Oliveira, G. Bishop, and D. McAllister. Relief texture mapping. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 359–368, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[36] M. M. Oliveira and F. Policarpo. An efficient representation for surface details. Technical report RP-351, January 2005.

[37] M. Peercy, J. Airey, and B. Cabral. Efficient bump mapping hardware. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 303–306, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[38] P. Prusinkiewicz, M. James, and R. Měch. Synthetic topiary. In A. Glassner, editor, *Proceedings of ACM SIGGRAPH 94*, pages 351–358. ACM Press, July 1994.

[39] P. Prusinkiewicz and A. Lindenmayer. *The Algorithmic Beauty of Plants*. Springer Verlag, 1991.

[40] P. Prusinkiewicz, P. Mündermann, R. Karwowski, and B. Lane. The use of positional information in the modeling of plants. In E. Fiume, editor, *Proceedings of ACM SIGGRAPH 2001*, pages 289–300. ACM Press, 2001.

[41] A. Reche-Martinez, I. Martin, and G. Drettakis. Volumetric reconstruction and interactive rendering of trees from photographs. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 720–727, New York, NY, USA, 2004. ACM Press.

[42] W. T. Reeves and R. Blau. Approximate and probabilistic algorithms for shading and rendering structured particle systems. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 313–322, New York, NY, USA, 1985. ACM Press.

[43] I. Remolar, M. Chover, O. Belmonte, J. Ribelles, and C. Rebollo. Real-time tree rendering. Technical report DLSI 01/03/2002, Campus de Riu Sec, E-12080 Castelln, Spain, March 2002.

[44] M. Sipser. Introduction to the theory of computation. *SIGACT News*, 27(1):27–29, 1996.

[45] G. Stiny. *Pictorial and Formal Aspects of Shape and Shape Grammars.* Birkhauser Verlag, Basel, 1975.

[46] G. Stiny. Introduction to shape and shape grammars. *Environment and Planning B*, 7:343–361, 1980.

[47] G. Stiny. Spatial relations and grammars. *Environment and Planning B*, 9:313–314, 1982.

[48] G. Stiny and W. J. Mitchell. The palladian grammar. *Environment and Planning B*, 5:5–18, 1978.

[49] J. Weber and J. Penn. Creation and rendering of realistic trees. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 119–128, New York, NY, USA, 1995. ACM Press.

[50] L. Williams. Casting curved shadows on curved surfaces. In *In Proceedings of SIGGRAPH 1978*, pages 270–274. ACM Press, 1978.

[51] P. Wonka, M. Wimmer, F. Sillion, and W. Ribarsky. Instant architecture. *ACM Transactions on Graphics*, 22(4):669–677, july 2003. Proceeding.

[52] Q. Yu, C. Chen, Z. Pan, and T. Chi. Interactive 3-d visualization of real forests. In *Proceedings of International Conference on Artificial Reality and Telexistence*, December 2003.