THE UNIVERSITY OF CALGARY

The Use of Subdivision Surfaces in the Modeling of Plants

by

Peter MacMurchy

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA

April, 2004

THE UNIVERSITY OF CALGARY

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled "The Use of Subdivision Surfaces in the Modeling of Plants" submitted by Peter MacMurchy in partial fulfillment of the requirements for the degree of Master of Science.

Abstract

Plants and plant organs are commonly modeled using skeletal techniques. Plant structure is then described in terms of axes, around which limb surfaces are built using standard geometric modeling techniques, such as generalized cylinders or implicit surfaces. In this research, an alternative, based on the use of subdivision techniques, is described. The approach is to build a simple polygon mesh around the skeleton and use a surface subdivision algorithm to get a smooth surface. Special attention is given to the modeling of surfaces surrounding branching points. It is shown that subdivision surfaces provide a useful alternative to approaches based on implicit surfaces. The method is illustrated using models of tree branches and compound leaves.

Acknowledgments

Firstly, I would like to thank my supervisor, Dr. Brian Wyvill, for allowing me to pursue this topic, and for support and encouragement throughout my studies. It has been an honour to work with my co-supervisor, Dr. Przemek Prusinkiewicz. His expertise and insight is impressive and unparalleled. He also provided financial support and several of the photographs in this thesis. Dr. Faramarz Samavati inspired significant improvement in both this project and my understanding of subdivision surfaces. I would like to thank my committee for making the defense experience an entirely positive one.

My fellow students have made my graduate studies the best time of my life. Many have also provided invaluable assistance with my research. Dr. Lars Mündermann provided frequent assistance with plant modeling and turtle geometry. Julia Taylor-Hell's models inspired improvement in, and validated, my software. Callum Galbraith supplied the implicit surface images and timings. I've benefitted tremendously from the knowledge and insight of Erwin de Groot, Martin Fuhrer, Pauline Jepp, Dr. Radek Karwowski, Brendan Lane, Jeff Mahovsky, Mark J. Matthews, Colin Smith, and Jing Yu.

Several people have proofread this thesis. In particular, Joanna Gaskell suffered through the extremely rough first draft, providing many helpful suggestions. Norm MacMurchy proofread several drafts, always leaving it sounding more professional.

Additionally, I've drawn inspiration from the well-tended gardens of Marg MacMurchy and Theresa Molinaro. Finally, my eternal gratitude goes to my parents, for their unwavering support, encouragement, patience, and inspiration.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Problem Statement

Branching structures have long been of interest to scientists in various fields. Visualization of branching structures has ranged from simple line drawings to sophisticated, realistic renderings. Scientific studies often only require schematic renderings to describe the branching topology in question. On the other hand, image synthesis applications, such as in the entertainment industry, often require more realistic renderings. In these applications, details such as surface topology and texture are important. Modeling of branching topologies for botanical structures is a well-studied problem. However, generating a smoothly connected surface around a branching skeleton remains difficult.

1.2 Contributions

In this research, a solution is proposed for generating smooth surfaces for a variety of branching structures. The method bridges two well-studied computer graphics techniques: *skeletons* and *subdivision surfaces*. An L-system is used to generate a string that encodes a skeletal branching structure. The string is used to generate a coarse polygonal mesh surrounding the skeleton. A subdivision scheme is applied to the coarse mesh. The final mesh retains the overall structure of the model described by the L-system, but is designed with limbs that blend smoothly at branching points. A coarse mesh can be generated as either a closed volume or an open surface. For trunks and stems, a closed cross-section is used (Figure 1.1, top row). For compound leaves, an open cross-section is used (Figure 1.1, bottom row). Several techniques are proposed for applying texture to the meshes. Finally, the method is shown to perform favourably compared to an implicit surface method.

1.3 Thesis Overview

In Chapter 2, the application of various surface modeling techniques to the creation of smoothly blended branching structures is examined. Chapter 3 contains a brief review of subdivision surfaces. Chapter 4 contains a review of L-systems and turtle geometry. In Chapter 5, the proposed method is described in detail, with attention paid to generating geometric structures of leaves and stems. In Chapter 6, the generation of texture coordinates for the meshes generated in Chapter 5 is described.

Chapter 7 contains images that demonstrate the proposed method applied to several plant models. In Chapter 8, the performance of the proposed subdivision-based method for modeling trees is compared to an implicit surface-based method, revealing that subdivision is up to thirty times faster. Finally, Chapter 9 contains an evaluation of the results, with respect to both the initial goals and to related research, and includes suggestions for further work.

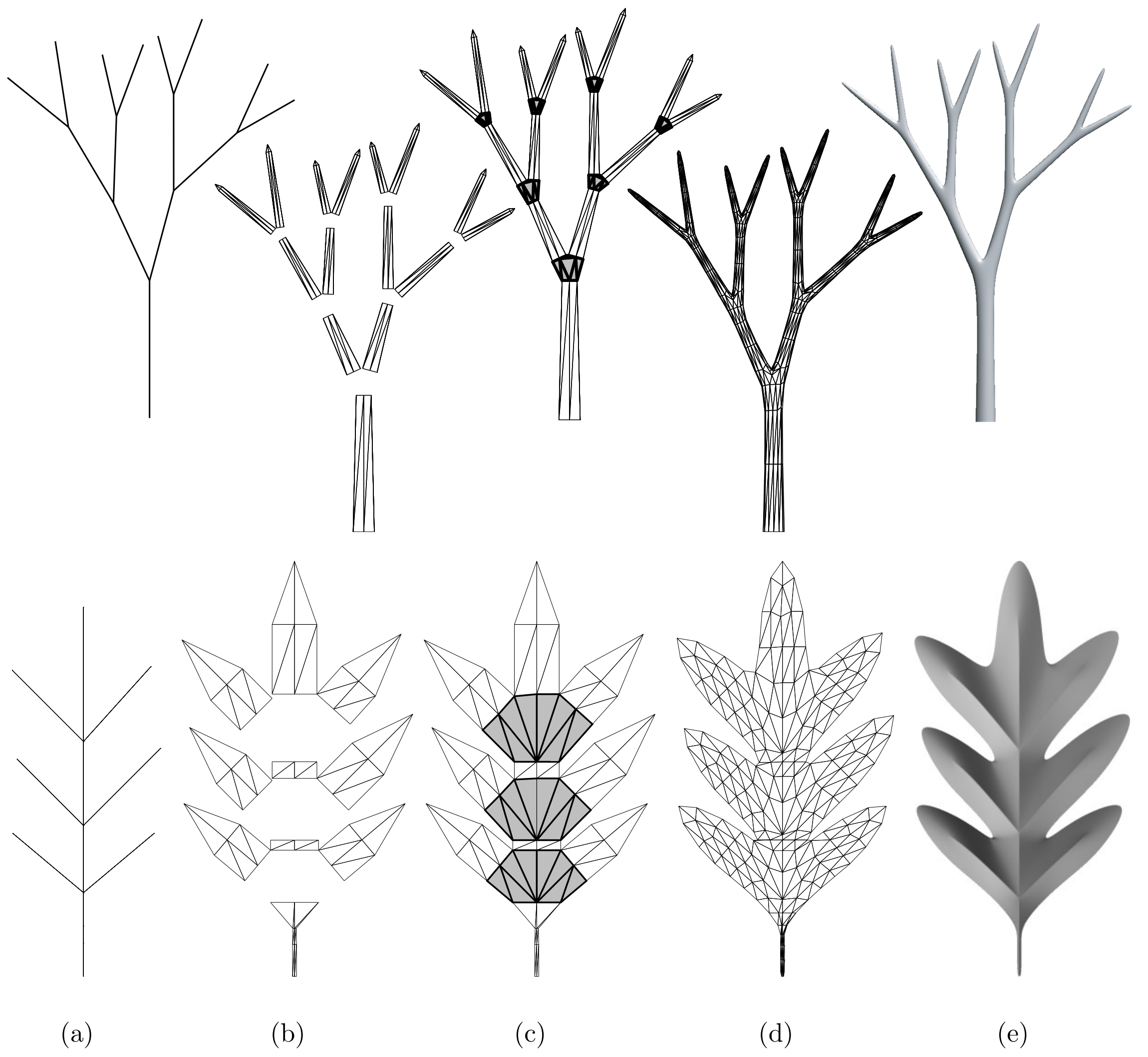


Figure 1.1: Generating a branching tree structure (top) and a compound leaf structure (bottom): (a) initial skeleton; (b) coarse mesh for the unbranched portions of the skeleton; (c) complete coarse mesh, highlighting structures inserted at branching points; (d) mesh resulting from a twofold Loop subdivision of the coarse mesh (e) shaded rendering of the subdivided mesh.

Chapter 2

Background

2.1 Overview

The problem of generating smooth surfaces for tree branching junctions is not a new one. After defining surface smoothness (Section 2.2), the three key computer graphics techniques that have been applied to tree models are reviewed: parametric surfaces (Section 2.3), implicit surfaces (Section 2.4), and subdivision surfaces (Section 2.5).

2.2 Quantifying smoothness

Derivative continuity is a traditional measure of smoothness, characterizing change in the directions and magnitudes of derivative vectors across vertices. The set of curves and surfaces denoted by C^n are those where n is the highest derivative that is changing smoothly across any given point. C^0 is used to denote connected curves and surfaces. C^1 refers to the set of curves and surfaces possessing tangent continuity, in both direction and magnitude. For surfaces, C^1 implies both tangent plane continuity (G^1) and a one-to-one projection of the surface onto the tangent plane [ZS00]. C^2 is the set of curves and surfaces whose second derivative is continuous across all interior points. As second derivative continuity implies first derivative continuity, all C^2 curves and surfaces are also C^1. Since the converse is not true, C^1 is a subset of C^2.

Geometric continuity [Bar88, FvD82] characterizes the continuity of the unit tangent and curvature vectors at vertices. G^0 is equivalent to C^0. G^1 denotes curves and surfaces where the tangent directions are equal on opposite sides of a vertex. G^1 continuity does not require the magnitudes of the tangents to be equal across a vertex. G^2 refers to continuous curvature vector directions. G^n thus denotes continuity in the direction of the nth derivative, but orders greater than two are seldom (if ever) used in practice.

Some surfaces, such as those generated using Loop subdivision (Section 3.3), are C^2 in the regular case and G^1 elsewhere. Subdivision surface researchers generally consider derivative continuity to be stronger than tangent-plane continuity [Rei95, Zor98, Zor00b, Zor00a]. By convention, a subdivision algorithm that generates a surface that is C^2-continuous everywhere is considered ideal [WW02].

2.3 Parametric surfaces

2.3.1 Generalized Cylinders

Modeling of botanical structures was initially focused on the skeletal branching structure of trees and herbaceous plants [AK84, FL74, Hon71, Smi84, VEJA89]. The simplest step in the quest for realism was the representation of internodes as 3D cylinders [dREF$^+$88, PL90, PLH88]. A geometrically more advanced technique was the use of generalized cylinders [Blo85, DL97, Hol94, Leo91, Měc97, Opp86, PMKL01, AMZ99]. A *generalized cylinder* [Blo90] (Figure 2.1) is a parametric surface generated by sweeping a planar cross section along a space curve. Generalized cylinders can produce polygons or spline patches (described below) as output. Unfortunately,

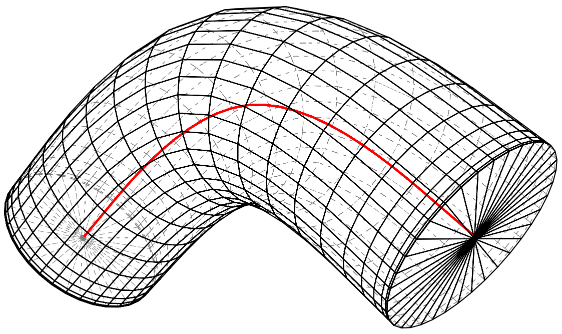


Figure 2.1: A generalized cylinder. The red line represents a spline path along which a circular cross section has been swept.

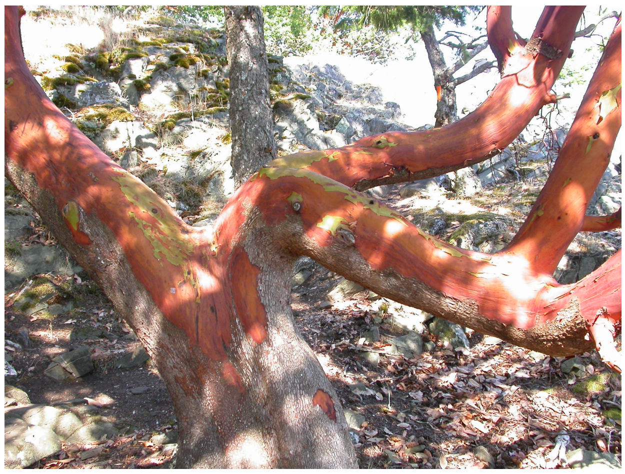


Figure 2.2: Photograph of an arbutus tree, showing smooth junctions between branches.

---

generalized cylinders do not properly capture the smooth geometry of branching points, such as that shown in Figure 2.2. Additionally, the resulting self-intersecting meshes create problems for some non-photorealistic rendering methods [SP03].

Jules Bloomenthal initially addressed the problem of generating continuous surfaces for trees by modeling tree limbs as generalized cylinders and forming smooth junctions using lofted surfaces to generate a bifurcating ramiform [Blo85].

2.3.2 Spline patches

A *spline* ([Far02], Chapters 3-8) is a continuous piecewise polynomial curve in euclidean 3-space ($\mathbb{E}^3$). A spline is defined by control points in $\mathbb{E}^3$ and has a linear (1D) parametric domain $t \in [0, 1]$. A spline is typically drawn by evaluating each polynomial segment for several sample values of the parameter t. The spline is approximated by connecting the resulting points with a series of lines. Cubic B-splines are a popular spline formulation that produces C^2-continuous curves which approximate, rather than interpolate, their control points.

Spline patches ([Far02], Chapter 14) extend the concept of splines to a bilinear (2D) parameter space, with parameters $u, v \in [0, 1]$. A surface is generated by evaluating a set of curves in the u domain over a set of curves in the v domain. This results in a rectangular configuration of control points. Any given point in the u domain corresponds to a curve in the v domain, and vice versa. The two-dimensional parameterisation is useful as the parameter coordinates can also be used for other purposes, in particular as texture coordinates.

Spline patches are homomorphic to a bounded disc. Decomposing an arbitrary surface to a continuous set of spline patches can be tricky, even when the surface is homomorphic to a bounded disc. Decomposition of models homomorphic to other topologies, such as spheres, into spline patches often requires degenerate patches ([Far02], Chapter 14). A tree, in particular, is homomorphic to a sphere [Fir91].

2.4 Implicit surfaces

While parametric surfaces provide reasonable results, their application is, in some cases, difficult. In particular, one must generate a new set of patches or loft curves for each degree of branching, taking care to prevent discontinuities. For example, the configuration described in [Blo85] for bifurcation cannot be automatically extended to support trifurcation or general multifurcation. Implicit surfaces [Blo97] provide an automatic means of smoothly blending such general branching configurations.

An *implicit surface* is defined by a function $f : \Re^3 \rightarrow \Re$. That is, given a point in $\Re^3$, f returns a scalar value. The surface is defined to lie wherever f returns some particular value, known as an *isovalue*. While there are several varieties of implicit surfaces, the forms best suited to tree modeling are skeletal implicit surfaces [Blo95b]. Here, f is a combination of the implicit functions for a set of primitive shapes, such as cylinders or spheres, represented by line and point skeletons. The combination utilizes a blending function which ensures smooth blends (for example, C^1 continuity) between parts of the implicit surface defined by nearby skeletal primitives.

The use of implicit surfaces addresses the problem of fitting a surface to a branching structure in a general way [Blo95b], but bulging, an unnatural-looking increase in girth, especially at branching points (Figure 2.3), is a problem [Blo95a]. Bloomenthal addressed [Blo95b] these problems using convolution surfaces [BS91], which reduce, but do not eliminate, this bulging problem.

In the context of plant modeling, the use of implicit surfaces leads to rendering problems. Direct rendering methods, such as ray-tracing [JW88], are time-consuming. Alternatively, the surfaces can first be polygonized, but this requires

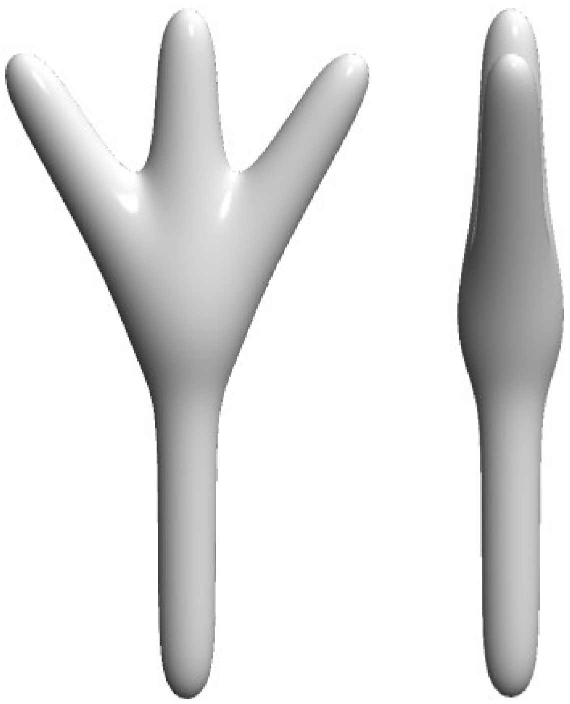


Figure 2.3: A bulging convolution surface.

a very fine voxel grid for long, thin limbs, resulting in a large number of triangles and long compute times. The performance of implicit and subdivision techniques is compared in Chapter 8.

Implicit surfaces have been applied to models generated by string rewriting systems. Compound leaves were modeled by generating their structure with L-systems and their geometry with 2D implicit surfaces [HPW92]. 3D implicit surfaces, with texture and displacement maps, were used to generate surfaces for L-system-derived tree models [Mar03]. Tree models were used as an example for a convolution surfaces system that could easily be extended to support string rewriting systems [JTFP01].

2.5 Subdivision surfaces

A different solution to the modeling of surfaces surrounding branching points is to use subdivision surfaces. A polygon mesh subdivision algorithm (described in more detail in Chapter 3) iteratively refines a polygonal mesh, using rules which at the limit produce a smooth surface. Subdivision surfaces can be applied to a wide variety of complex shapes, making them attractive for visualizing branching structures. Their application to branching structures was pioneered by Tobler et. al. [TMW02a, Mai02, TMW02b], who procedurally grew a mesh by repetitively adding predefined template meshes to an initial mesh and then subdivided the resulting coarse mesh. Subdivision surfaces have also been used for scientific visualization of branching structures, namely the vessels of the liver [FFW01, FFKW02]. Related techniques, using subdivision to model branches, were introduced by Weta Digital in the film "Lord of the Rings: The Two Towers" [AP03] and, in a research setting, by Akleman et. al. [ACS03].

Of course, a hybrid solution is also possible, with a coarsely polygonized implicit surface used as input to a subdivision scheme [WJvOW00, MLP01], but such a solution requires all the computational machinery of both techniques, as well as the tree model. Furthermore, a coarsely polygonized implicit surface may miss entire limbs of a plant model.

Proposed solution

Here, we describe a method in which the skeleton of the branching structure is produced first, then a low-resolution mesh is built around it. This makes it possible to use standard skeleton-building methods, in particular L-systems, to define the topology and skeletal geometry of the branching structure. We then apply a standard surface subdivision algorithm, in particular Loop's subdivision scheme [Loo87], to construct a surface with smooth branching points around this skeleton.

Chapter 3

Subdivision surfaces

Polygon subdivision has become a popular computer graphics technique for surface modeling. Its appeal lies in its simplicity and its applicability to a wide variety of surfaces. Initially developed in the seventies as an arbitrary-topology alternative to spline patches [CC78, DS78], subdivision has become popular recently [DKT98] thanks to increased computer memory [Sab01] and the development of rigorous mathematical analysis of its properties [WW02].

3.1 Definition

A subdivision scheme can be expressed as a set of *masks*, or *stencils*, a schematic representation of an affine transformation of the geometry and a transformation of the topology. The masks are defined for vertices of arbitrary *valence*; that is, vertices connected to any number of neighbouring vertices. Rather than evaluating polynomials defined by control points over a parametric domain, subdivision operates directly on the vertices of a polygon mesh. Unfortunately, this means 2D parameterizations of the surface is not straightforward, and may introduce significant distortions or discontinuities.

3.2 Relevant Properties

To be suitable for plant modeling, the surfaces used to dress the skeletons must satisfy two properties: continuity and local control. Subdivision surfaces satisfy both criteria inherently, with the degree of continuity and local control dependent on the scheme chosen. They were introduced to create smooth surfaces on the basis of coarse polygonal representations. Since they are based on masks, subdivision surfaces are local in character.

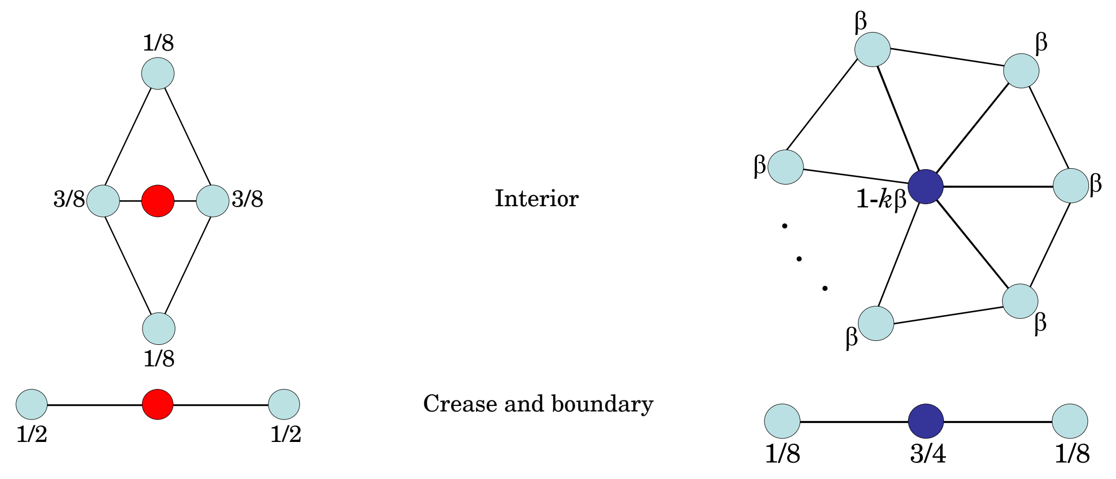
3.3 Loop scheme

There are a variety of subdivision schemes, varying by the type of polygons they use and in their masks. An accessible and thorough overview of subdivision and the most common subdivision schemes is presented in [ZS00]. The scheme implemented and used in the context of this research was proposed by Loop [Loo87]. His scheme works exclusively with triangles, and is one of the simplest subdivision schemes.

Loop subdivision takes as input a set of vertices S^0 and refines them, producing a new set of vertices S^1. *Even vertices* are those that are in S^0, the input polygon mesh. These vertices are carried forth into the new mesh, S^1. *Odd vertices* are those which are created during the subdivision process, and thus are unique to S^1.

Interior vertices

Loop subdivision can be considered in two steps. In the first step, odd vertices are generated, using the masks shown in Figure 3.1(a), where the red circle represents the new vertex. The new vertices are created by splitting each edge in S^0. The



(a) Odd vertices. (b) Even vertices.

Figure 3.1: Loop scheme masks. Redrawn from [ZS00]. β is $\frac{1}{k}(\frac{5}{8} - (\frac{3}{8} + \frac{1}{4}\cos\frac{2\pi}{k})^2)$, and k is the valence of the vertex being considered.

second step of Loop subdivision adjusts the position of the even vertices so that each is closer to its limit position. This step uses the two-dimensional masks shown in Figure 3.1(b), where vertices labeled β are neighbouring even vertices. For interior vertices, this amounts to a weighted average of the position of the vertex in question and the positions of the vertices in its 1-neighbourhood.

Creases and Boundaries

In Loop subdivision, creases and boundaries are handled as special cases, using one-dimensional masks [HDD+94]. *Boundaries* are edges belonging to only one triangle. If a modeler wants an internal edge, belonging to two triangles, to remain apparent in the limit surface, it must be tagged as a *crease* and subdivided with the one-dimensional masks shown in Figure 3.1. As with the interior masks, the crease and boundary masks operate by splitting the edge in S^0, then moving the even vertices closer to their limit positions.

Semi-sharp creases, as described in [DKT98], can be used instead of sharp ones. A semi-sharp crease is achieved by applying the sharp crease masks for some number of iterations, then using the usual smooth masks for the remaining iterations. This allows an edge in the coarse mesh to have a noticeable influence on the limit surface, without introducing a discontinuity.

Discussion

The limit surface of Loop's scheme is C^2 continuous at regular vertices; that is, vertices with valence six and thus exactly six neighbours. At extraordinary vertices (the remaining vertices), the limit surface has G^1 continuity. Extensions to Loop's scheme, featuring larger masks for extraordinary vertices, guarantee C^2 continuity at all interior points [Loo01, BLZ00]. The boundary and crease masks are equivalent to cubic B-spline subdivision; thus, boundaries and sharp creases are C^2 continuous curves along the crease. The surface will have C^0 continuity across sharp creases. Furthermore, Loop's scheme has entirely positive coefficients in its masks (Figure 3.1), giving it the property that the limit surface is entirely contained within the convex hull of the coarse mesh. Loop's scheme is based on quartic box splines, a triangular variety of spline patch. He chose quartic splines so that his scheme would produce an adequately smooth surface. The only disadvantage of Loop's scheme is that it requires the input mesh to consist entirely of triangles. From the perspective of plant modeling, where meshes are generated procedurally, it is no problem to generate triangles. Loop's scheme also works well with meshes triangulated from quadrilateral meshes [ZS00].

Chapter 4

L-systems

The input to the algorithm proposed in this work is a plant model described using an *L-system*. A model specification in the L-system formalism is used to generate a string that encodes the topology and geometry of a skeletal branching structure. This method of plant modeling is described in detail in [PHHM96], upon which the following discussion is based. The skeleton is subsequently used as the basis for a polygonal mesh to be subdivided.

4.1 L-systems

L-systems were originally introduced by Lindenmayer for describing simple multicellular organisms [Lin68] and later adapted for modeling higher plants [FL74, FL76, PLH88, PL90]. An L-system operates on a string of symbols taken from an *alphabet* V, which is a finite set of symbols used to represent diverse components of the modeled structure. The structure is represented by a string over V. An L-system specification also includes a string over V, called the *axiom*, which describes the initial structure at the beginning of development, and a set of *productions*, or rewriting rules, that state how to replace symbols in the string by new substrings. Productions are intended to simulate the development of individual components of the structure over given time intervals. They are applied in parallel, to all symbols in a string, in a sequence of derivation steps. This parallelism simulates the simultaneous progression of time throughout the entire developing plant structure.

4.2 Bracketed String Notation

Lindenmayer introduced a *bracketed string notation* [Lin68, Lin71] for representing branching structures. Skeletal segments are assigned labels from an alphabet V. A branching structure is represented as a string α of symbols over the alphabet $V_E = V \cup \{[,]\}$:

$$\alpha = x_1[\alpha_1]x_2[\alpha_2]...x_n[\alpha_n]x_{n+1}$$

The substrings x_i do not contain brackets. Substrings α_i may include bracketed substrings, with pairs of brackets nested to arbitrary degree. The string $x_1x_2...x_nx_{n+1}$ represents the primary (zeroth-order) axis of α, which could represent the trunk of a tree. The substrings α_i represent the first-order branches, those which extend directly from the trunk. These may in turn support second-order branches and so on.

In parametric L-systems [Han92], one or more numeric parameters may optionally be associated with each symbol in the string. Parameters afford continuous variation of component characteristics, and thus greater realism, of a plant model.

4.3 Turtle Geometry

The bracketed string notation, introduced in the previous section, only defines the topology of the branching structure. Geometric aspects can be added using turtle geometry [Pru86, PL90]. Reserved string symbols, along with associated parameters, are interpreted as commands for a LOGO-style *turtle* [Ad80]. The turtle is a drawing

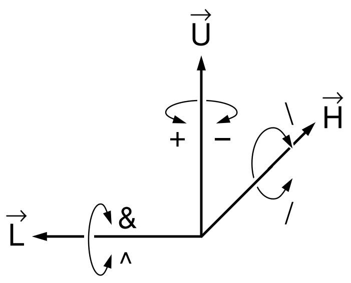


Figure 4.1: Turtle reference frame, from [PLH88].

entity which encompasses state information. It starts at a default drawing position, which is at the origin, with a default orientation, described next.

The basic turtle information, called a *reference frame*, consists of a position and three orthogonal unit-length vectors: *Heading*, *Left*, and *Up* (Figure 4.1). For this work, a right-handed coordinate system is assumed. In this case, *Heading* is initially defined to be aligned with the positive direction of the Y Cartesian axis, and *Left* is aligned with the negative direction of the X Cartesian axis. *Up* is defined as the vector perpendicular to both the *Heading* and *Left* vectors, so that $\vec{H} \times \vec{L} = \vec{U}$. Turtle position and orientation are updated incrementally as the turtle moves.

The alphabet of an L-system using turtle interpretation is assumed to contain several symbols that have a predefined interpretation. A complete list of such reserved symbols, used in the L-system-based plant modeling program CPFG employed in this research, is included in [Měc98]. For the current work, the symbols of interest are those which are interpreted as commands to move the turtle, rotate the turtle, and affect the radius of the branch at the current turtle position. These symbols may be complemented by numerical parameters included within parentheses immediately following the symbol.

Specifically, "`F`", for *Forward*, moves the turtle one unit, or a distance specified in a parameter, in its current *Heading* direction. For example, "`F(2)`" means to move the turtle two units along its current heading.

The rotation commands, "`/`", "`\`", "`&`", "`^`", "`+`", and "`-`", rotate the turtle around the *Heading*, *Left*, and *Up* axes, as shown in Figure 4.1. Arguments to these commands are rotation angles in degrees.

The "`[`" and "`]`" commands respectively "push" and "pop" a stack of turtle frames. This stack is used to create branching structures in a depth-first fashion during string traversal by the turtle. When the turtle encounters a "`[`", it pushes the current turtle information onto the stack, thus initiating a branch. Upon encountering a "`]`" during interpretation, the turtle pops the top frame from the stack, and uses it to restore the turtle's state to the saved values, thus terminating the branch.

Commands "`#`" and "`!`" are used to respectively increase and decrease the width of a branch. With an argument, both commands set the current width to the value of the argument.

4.3.1 Example

As an example of an L-system derivation and turtle string, let us consider a stochastic bifurcating system. The L-system is shown in Figure 4.2. Three derivations, along with a visualization of the third derivation, are shown in Figure 4.3. The example L-system consists of one axiom, ω, and three productions, p_1, p_2, and p_3.

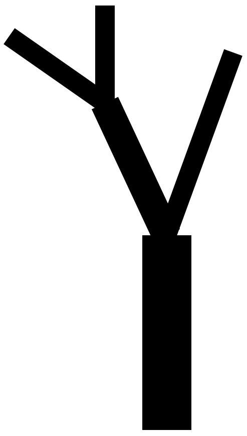
In this system, the symbol A refers to an apex, where new modules (branches or apices) are added. B refers to a branch, or internode, and is where growth occurs. Productions p_1 and p_2 are stochastic, meaning that at each derivation step the in-

$$\begin{aligned}
\omega &: A \\
p_1 &: A \rightarrow B(1)[+(20)A][-(20)A] : 0.6 \\
p_2 &: A \rightarrow A : 0.4 \\
p_3 &: B(x) \rightarrow B(1.4 \times x)
\end{aligned}$$

Figure 4.2: L-system 1.

$$\begin{aligned}
0 &: A \\
1 &: B(1)[+(20)A][-(20)A] \\
2 &: B(1.4)[+(20)B(1)[+(20)A][-(20)A]][-(20)A] \\
3 &: B(1.96) \\
&\quad [+(20)B(1.4) \\
&\quad [+(20)B(1)[+(20)A][-(20)A]] \\
&\quad [-(20)B(1)[+(20)A][-(20)A]]] \\
&\quad [-(20)B(1.4)[+(20)A][-(20)A]]
\end{aligned}$$

(a) Three derivations of the L-system from Figure 4.2



(b) Line drawing of derivation 3.

Figure 4.3: Derivation of L-system 1.

terpreter makes a pseudo-random choice as to which of them to apply. Production p_1 replaces each apex by a branch and a symmetric pair of apices. p_1 is assigned a probability of 0.6, meaning that this rule should be applied in sixty percent of the derivations. Production p_2 is an identity production. It is assigned a probability of 0.4, meaning that forty percent of the time no growth occurs. This serves to introduce asymmetry to the model. Production p_3 elongates all existing branches by forty percent at every derivation step; in this way, a branch's length becomes proportional to its age.

Chapter 5

Mesh Generation

5.1 Overview

The method proposed in this work consists of five stages, as illustrated in Figure 1.1:

1. A skeleton is generated using an L-system.
2. An initial coarse mesh is generated for the unbranched portions of the skeleton.
3. Junction structures are generated based on templates and inserted at branching points.
4. The completed coarse mesh is subdivided, using Loop's scheme.
5. The refined mesh is rendered, using graphics hardware or ray tracing.

Steps two through four of the algorithm are presented in this chapter. The resulting mesh is composed of triangles and is suitable for the subsequent application of the Loop subdivision scheme. How the segments' lengths are adjusted to make space for blending is described in Section 5.2. The enumeration of the branching configurations supported by the method is set out in Section 5.3. Preprocessing of branching junctions, to aid in distinguishing the configurations, is described in Section 5.3. Finally, the *junction structures*, branching meshes derived from *junction templates*, are described. These structures are specialized for open contours, suitable for generation of lobed leaves, as described in Section 5.4, and for closed contours, suitable for stems, as described in Section 5.5.

5.2 Segment length adjustment

When directly interpreted, the turtle strings produce cylinders that intersect at branching points. The cylinders need to be shortened along their axes so that a mesh can be generated without intersections or overlap (Figure 5.1).

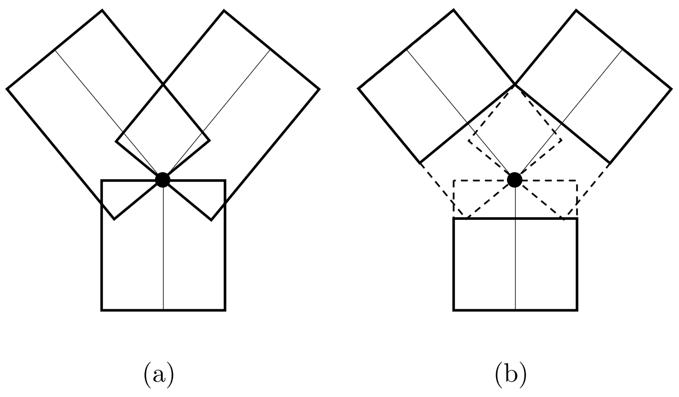
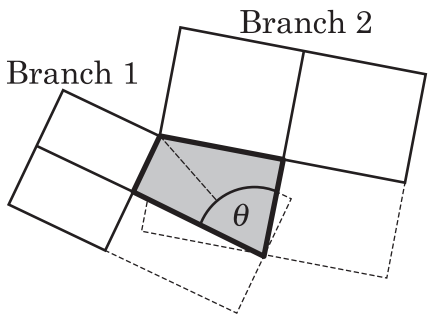
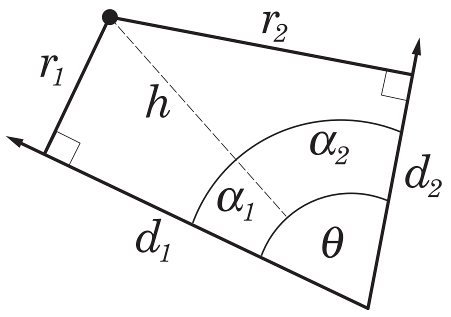


Figure 5.1: Segment length adjustment: (a) Before, with cylinders intersecting; (b) After, with the junction region clear.

Cylinder length adjustments are performed locally, at each branching point. To compute the correct segment lengths, branches are considered in pairs. The trunk is also treated as a branch. A pair of branches is first placed in a plane, as shown in Figure 5.2. Given the radii r_1 and r_2 of the branches under consideration, and the angle θ between the axes of the branches, the offset d_1 can be calculated for the first branch as follows:



(a)



(b)

Figure 5.2: Offset computation. Figure (b) corresponds to the shaded region of Figure (a). For each pair of branches, the offset distances d_1 and d_2 are computed as a function of the branch radii (r_1 and r_2) and the divergence angle θ.

$$\begin{aligned}
\frac{r_1}{\sin\alpha_1} &= h = \frac{r_2}{\sin(\theta-\alpha_1)} \,, \\
r_2\sin\alpha_1 &= r_1\sin(\theta-\alpha_1) = r_1\sin\theta\cos\alpha_1 - r_1\cos\theta\sin\alpha_1 \,, \\
r_1\sin\theta\cos\alpha_1 &= r_2\sin\alpha_1 + r_1\cos\theta\sin\alpha_1 = \sin\alpha_1(r_2+r_1\cos\theta) \,, \\
\frac{r_1\sin\theta}{r_2+r_1\cos\theta} &= \frac{\sin\alpha_1}{\cos\alpha_1} = \tan\alpha_1 \,, \\
d_1 &= \frac{r_1}{\tan\alpha_1} = \frac{r_2+r_1\cos\theta}{\sin\theta} \,. \qquad (5.1)
\end{aligned}$$

Similarly,

$$d_2 = \frac{r_1+r_2\cos\theta}{\sin\theta} \,. \qquad (5.2)$$

For a given branch, we compute offsets d_i with respect to all branches that meet the branch at its base. The largest such offset is applied to the branch.

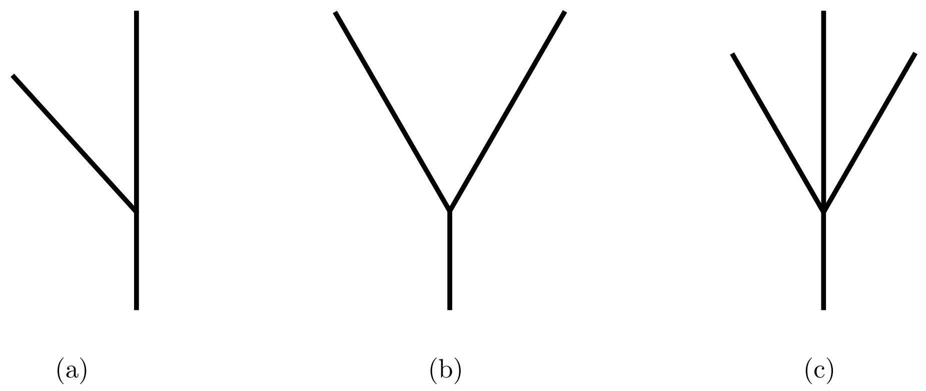


Figure 5.3: Supported branching configurations. (a) Trunk with lateral branch; (b) symmetric branching; (c) distichous branching.

5.3 Supported branching configurations

The essence of the proposed method lies in enumerating the cases for different configurations that occur in branching structures, and creating appropriate junction structures for each case. The cases considered are: a single branch positioned laterally with respect to the main axis (Figure 5.3(a)), symmetric bifurcation (Figure 5.3(b)), and branches on both sides of a main axis (*trifurcation*) (Figure 5.3(c)). The latter case occurs in the botanically important symmetric distichous and decussate branching patterns [Bel91].

The algorithm assumes planar branching; that is, at a given branching point, the trunk and all the child branches lie in a common plane. This assumption limits the number of distinct junction templates required. In fact, the method handles an arbitrary number of coplanar branches arranged around a common branching point. It has been observed that the configurations described above are indeed planar in many real plants [Col65].

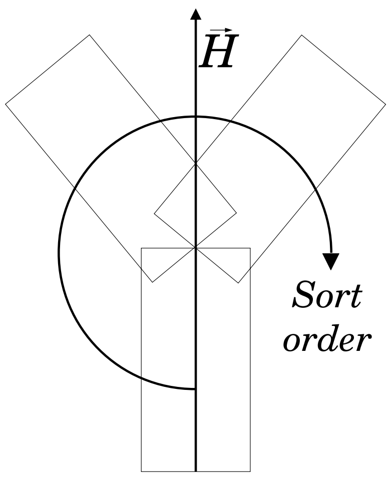


Figure 5.4: Sorting branches

Sorting Branches

The order in which branches are specified in an L-system string need not be related to the angles between these branches. In order to create the junction structures at the branching point using fixed templates, branches are ordered by increasing angle with respect to the basal segment (Figure 5.4).

5.4 Junction structures for leaves

Leaves are assumed to have open cross-sections. The template for the symmetric branching point is shown in Figure 5.5. Between each pair of branches, there will be a pair of identical vertices at the branches' bases (marked with a dot in Figure 5.5(a)). These vertices are merged, to avoid singularities in the refined mesh.

To achieve shading suggesting a midrib (i.e. primary venation), a V-shaped cross-section can be used. Edges along the midrib are tagged as creases. These creases are preserved during the subsequent subdivision using the proper masks [HDD$^+$94].

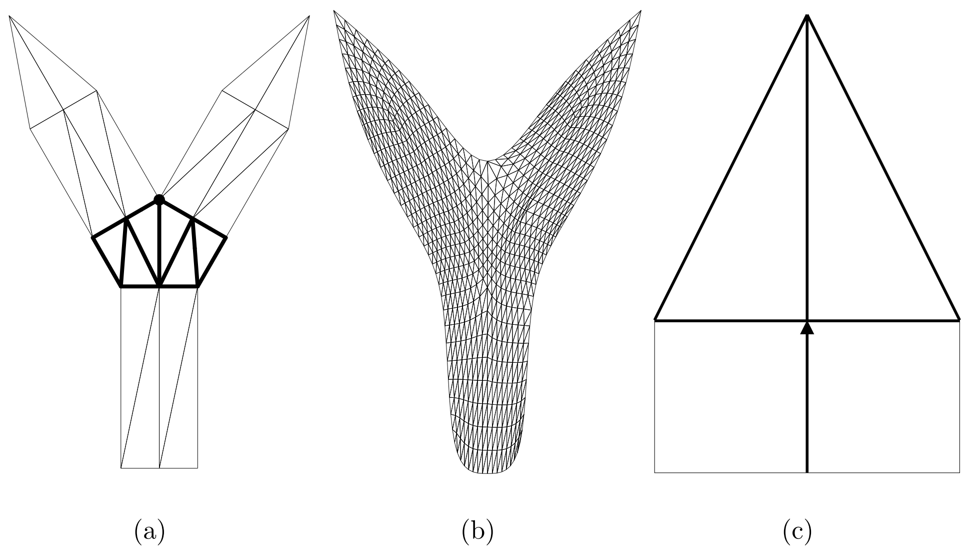


Figure 5.5: Branching leaf structure. (a) Coarse mesh with the branching region outlined in dark lines, with vertices to be merged indicated by the dot; (b) refined mesh; (c) placement of triangles at the end of a leaf branch.

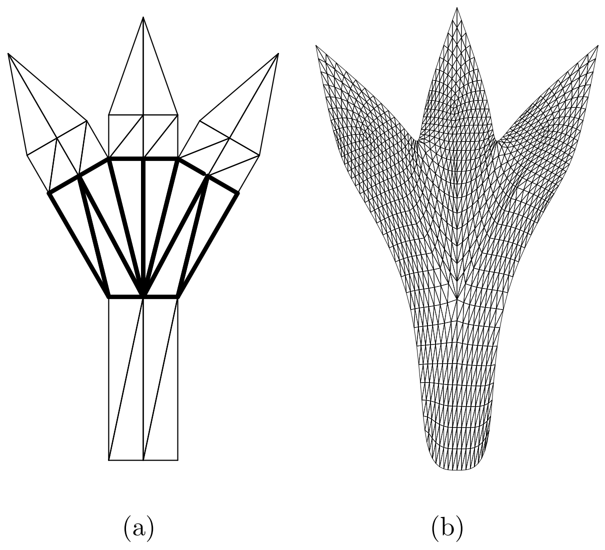


Figure 5.6: Leaf trifurcation. (a) coarse mesh; (b) after three iterations of Loop subdivision.

Semi-sharp creases can also be used, as described in [DKT98], rather than sharp ones. A semi-sharp crease is achieved by applying the sharp crease masks for some number of iterations, then using the usual smooth masks for the remaining iterations.

Some leaves observed in nature have pointed tips. To model this, a pair of triangles is appended to each tip (Figure 5.5(c)). The triangles are defined using the vector from the second-last cross-section to the final cross-section. The height of the triangles is set by the user in proportion to the width of the supporting leaf segment. If needed, the vertex at the tip of the triangle can be considered as sharp or semi-sharp during the subsequent subdivision.

The branching structure for leaves readily generalizes to planar branching of arbitrary degree. An example of a trifurcated structure is shown in Figure 5.6.

5.5 Junction structures for stems

5.5.1 Stem cross-sections

While the planar cross-section of stems is often circular, it is not clear what cross-section should be used for the coarse mesh. In looking for a simple polygon mesh, a square cross-section was initially chosen. However, joint structures based on a square cross-section produce poor results in the case of asymmetric branching. In Figure 5.7, a-d, the blend region, while smooth, is much larger than the control mesh junction. Moreover, the front view exhibits a distinct asymmetric puckered shape centered at the junction. Several methods of constructing the junction cross-section to reduce these effects were explored. For example, Figure 5.8 illustrates the effect of creating an extra ring of triangles around the base of the child branch. In the

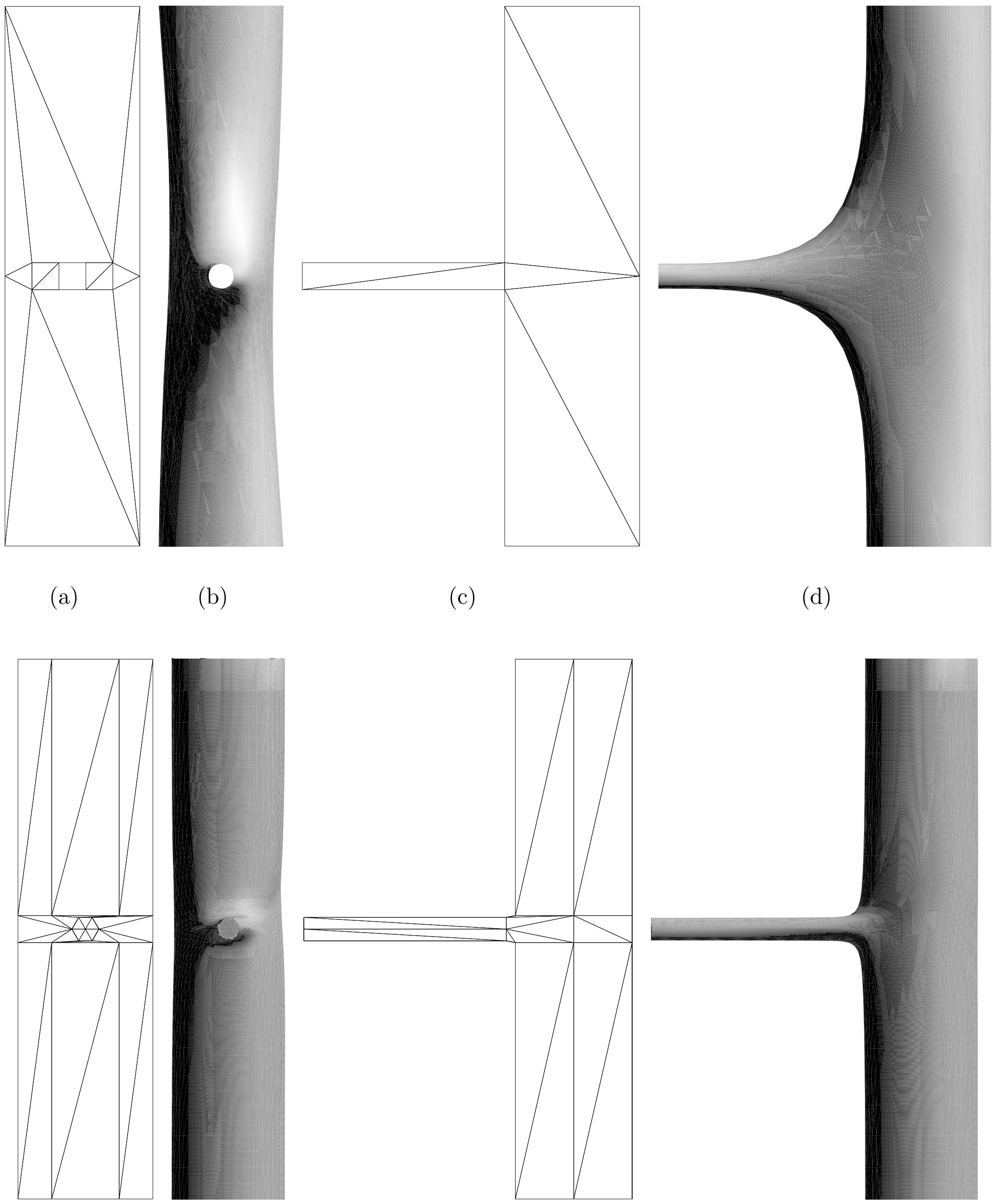


Figure 5.7: Lateral branching. Top row: square cross-section; bottom row: hexagonal cross-section. (a),(e) Front view of coarse mesh; (b),(f) Front view of refined mesh, comparing side concavities; (c),(g) Side view of coarse mesh; (d),(h) Side view of refined mesh, comparing blend sizes.

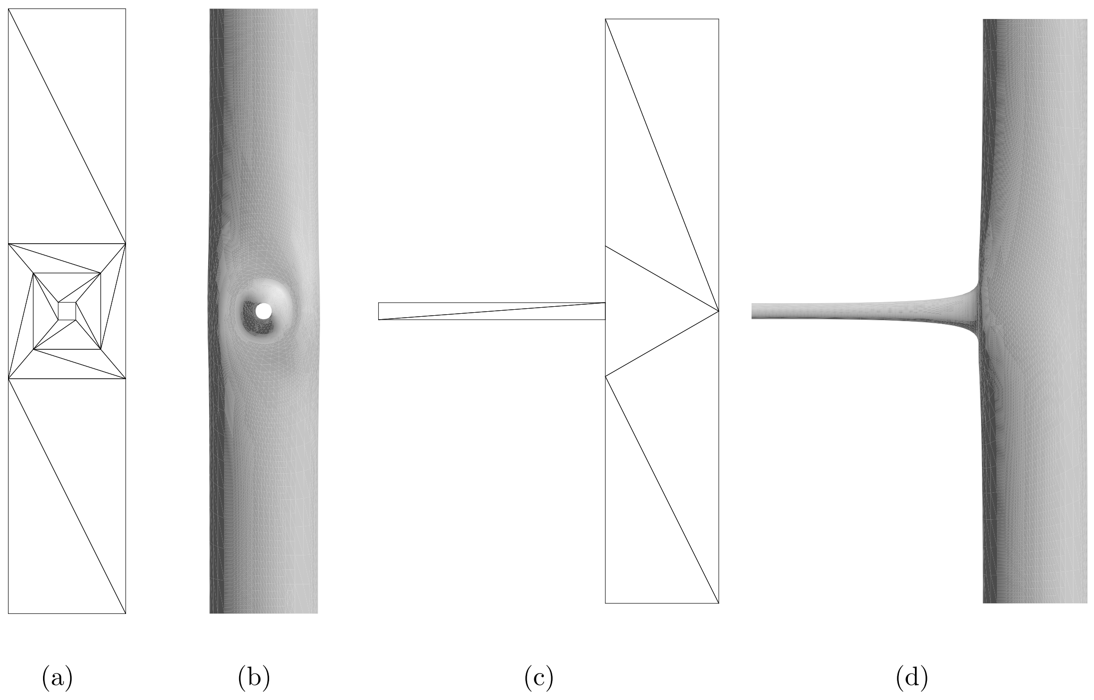


Figure 5.8: An alternate lateral branching configuration for a square cross-section. This configuration includes a ring of vertices around the base of the child branch. An unnatural-looking flat region is created by the additional triangles connecting the ring and the child branch base.

limit surface, this creates an acceptable blend but results in a flat region where the extra triangles are added, distorting the cross section of the trunk and the large child branch at the junction. The finer geometry generated using hexagonal cross-sections (Figure 5.7, e-h) shrinks the zone of influence of the blend. This eliminates the vertical concavities seen on either side of Figure 5.7(b), without introducing the bulges seen in Figure 5.8. A hexagonal coarse mesh cross-section also produces more circular limit surface cross-sections.

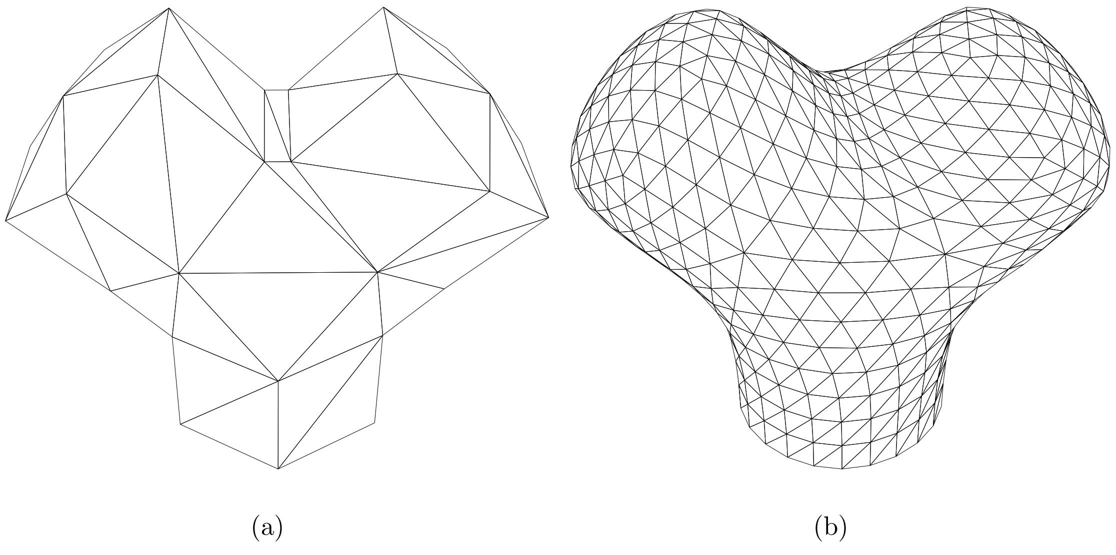
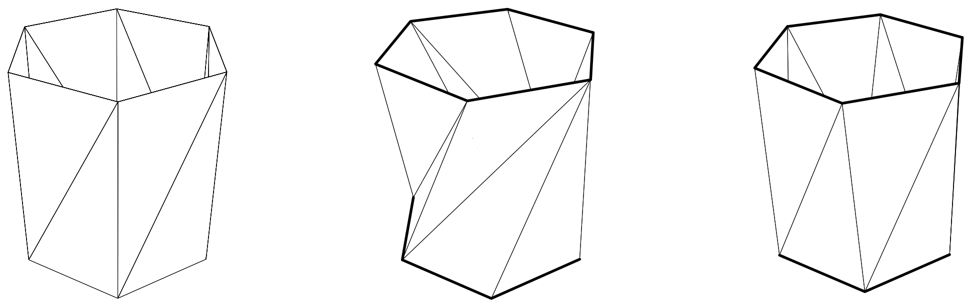


Figure 5.9: Branching junction, showing symmetric bifurcation (a) Coarse mesh (b) Refined mesh after 3 iterations of Loop subdivision

5.5.2 Twist compensation

The method for creating a mesh for straight portions of a branching structure is related to generalized cylinders [Blo90]. The proposed method starts with the reference frames obtained during turtle interpretation. Given a pair of consecutive reference frames, the method involves placing hexagonal cross-sections at each, connecting corresponding vertices from one cross-section to the next.

When choosing corresponding cross section vertices, any twist between consecutive frames must be taken into account. In turtle geometry, twist is rotation of the turtle about the *Heading* axis [PL90]. If unaccounted for, twist may produce geometric artifacts, such as self-intersecting geometry. This problem has been solved in the domain of generalized cylinders by using the parallel transport frame [Bis75, PMKL01], also known as the rotation-minimizing frame [Blo90].

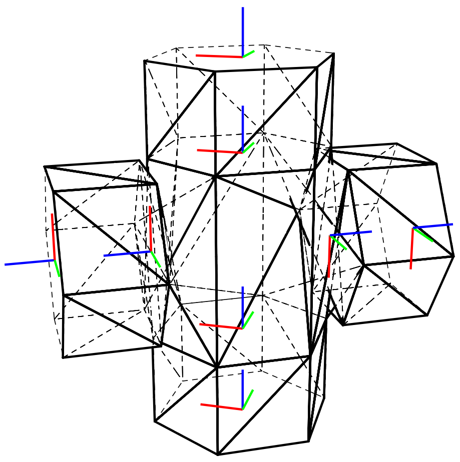


(a) No twist (b) 50° twist. (c) 50° twist, compensated.

Figure 5.10: Twist. Two cross-sections and the edges connecting them are shown.

Parallel transport works well in this implementation. As parallel transport requires storing both the parallel transport frame (for mesh construction) and the corresponding turtle frame (for construction of the axes), an alternative method may also be employed. In this case, cross-sections are oriented given the turtle frame, connecting a cross-section to the next cross-section so as to minimize twist, as shown in Figure 5.10. Even with this technique, up to thirty degrees of twist may remain between consecutive cross-sections. This does not affect the quality of the refined mesh, and sometimes leads to natural-looking twisting effects when limbs are texture mapped (Chapter 6).

Across junctions, arbitrary turning and bending of child branches can also produce twist, even with parallel transport (Figure 5.11). As with the non-branching case described above, the goal is to connect the closest vertices from one cross-section to the next. In particular, for a pair of child branches, each half of each child branch cross-section must be connected to the closest half of the trunk cross-section. These half-sections are referenced as being on the *outside* of the junction (Figure 5.12).


(a) Twisted junction structure.

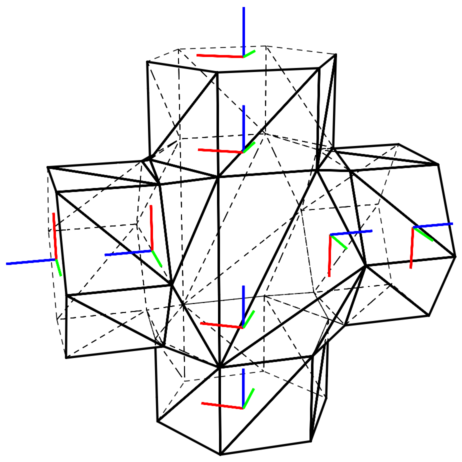

(b) Corrected junction structure.

Figure 5.11: Junction twist with parallel transport frames.

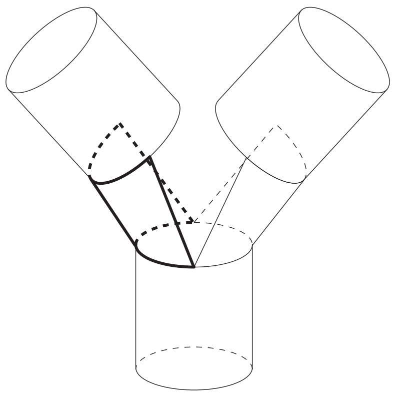
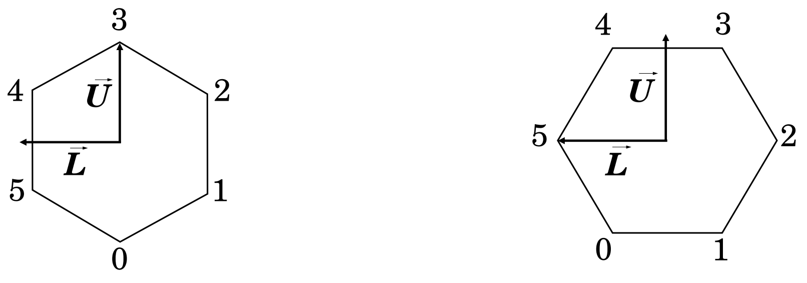


Figure 5.12: Outside half-section. The left half-sections are drawn bold and connected. The half-sections are oriented with respect to the branching plane and used to define corresponding vertices.

The remaining, *inside*, child branch half-sections will be connected to each other. Corresponding vertices are chosen to best align the outside half-sections, with respect to the *branching plane*, defined as the plane whose normal is the cross product of any two child branch *Heading* vectors.



(a) Vertex forward. (b) Edge forward.

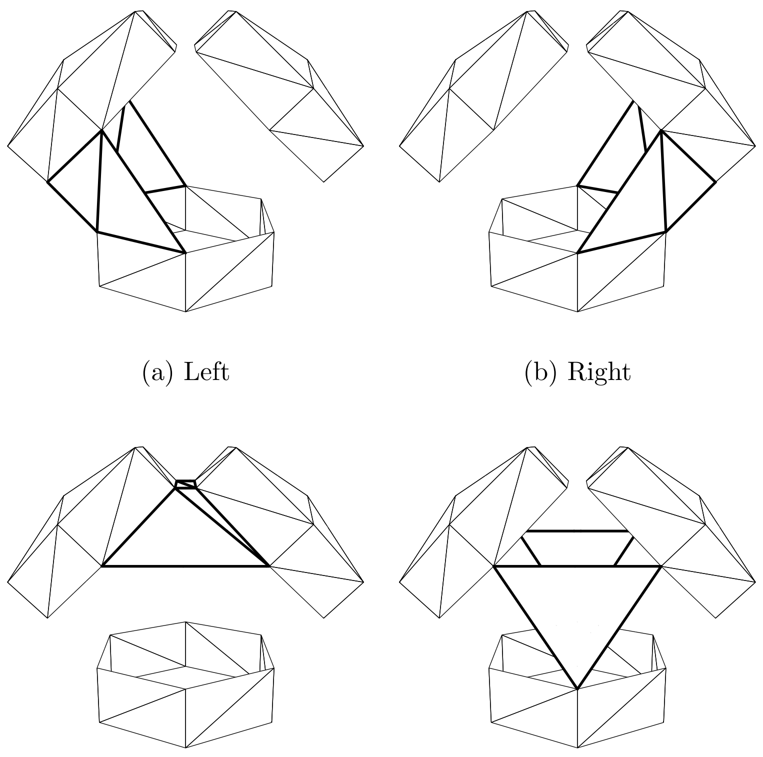
Figure 5.13: Cross-section vertex numbering, assuming that the branching plane is aligned with the *Left* axis.

5.5.3 Junction templates

Junction templates are the core of the proposed method. Templates are decomposed into subtemplates; this reduces the total number of templates required by allowing the use of combinations of subtemplates. In addition, this decomposition facilitates extension to higher-order branching nodes. In order to define the subtemplates, *vertex-forward* (Figure 5.13(a)) and *edge-forward* (Figure 5.13(b)) configurations of the cross-sections are distinguished when constructing junction structures. Subtemplates are defined based on combinations of these orientations.

For the symmetric bifurcation case, four classes of subtemplates are defined: left, right, top, and bottom. The position of each subtemplate is shown in Figure 5.14. These templates only add triangles to the mesh, using existing vertices. There are four left subtemplates, four right subtemplates, four top subtemplates, and eight bottom subtemplates (Figure 5.15). The left subtemplates are identical to the right subtemplates except for the numbering of the vertices they connect.

The distinction between vertex-forward and edge-forward orientations is impor-



(a) Left (b) Right

(c) Top (d) Bottom

Figure 5.14: Subtemplate placement. This figure illustrates vertex-forward orientation; the other cases are analogous.

tant. Otherwise, artifacts such as that shown in Figure 5.16 can arise.

Selecting among the templates can be accomplished using a bitmask, where each bit corresponds to one cross-section participating in the template. The figures are enumerated based on the bitmasks shown in Figure 5.17, with a bit value of 1 implying the corresponding cross-section is edge-forward.

Trifurcation

The method easily generalizes to higher-order planar branching, thanks to the decomposition of the junction templates.

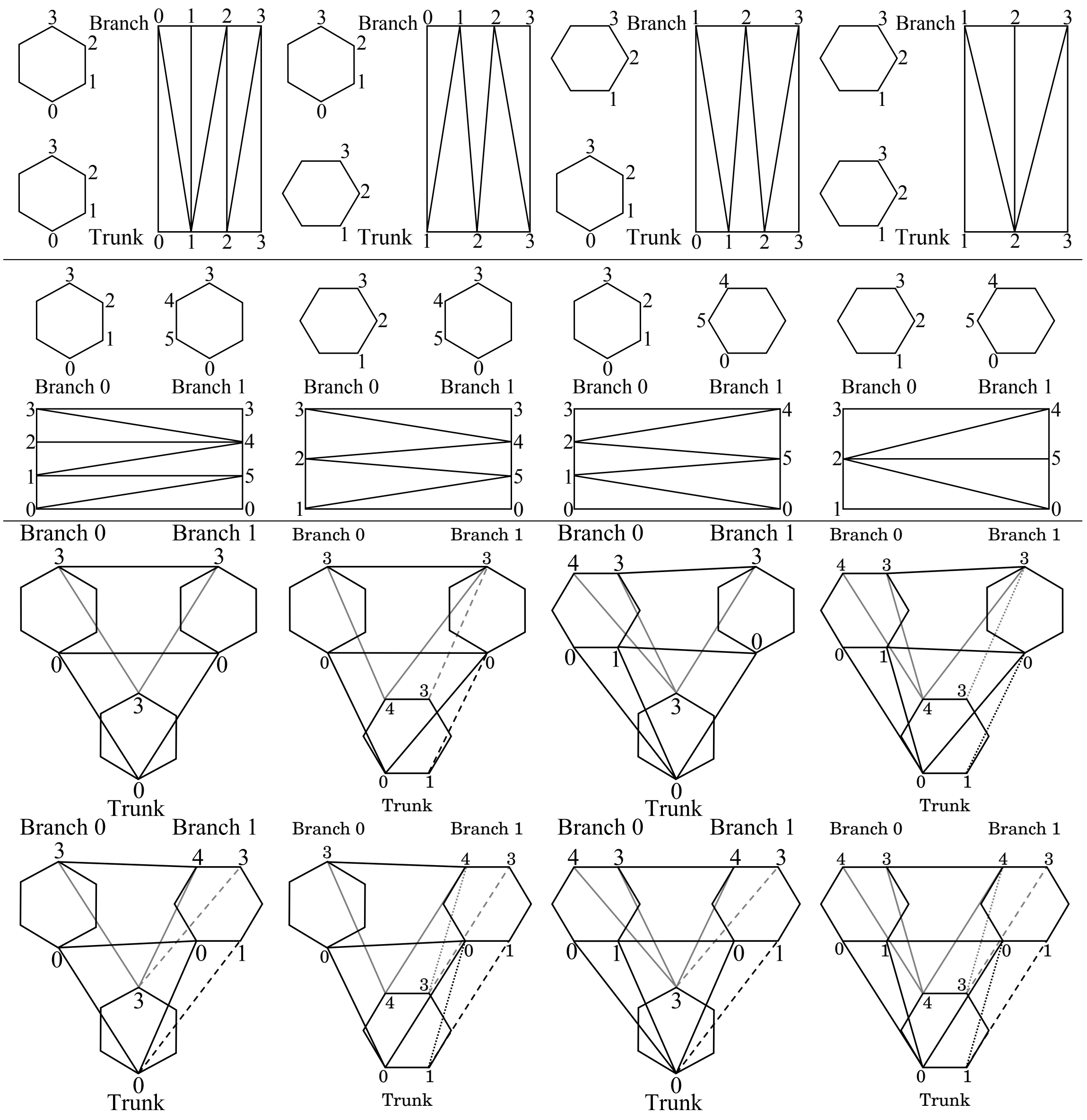
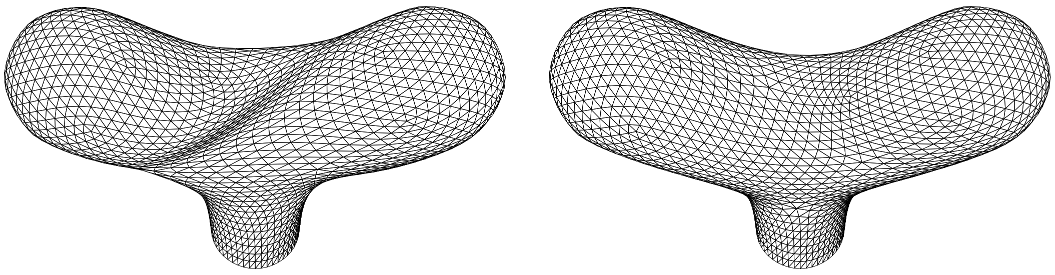


Figure 5.15: Junction templates. First row: right templates; second row: top templates; last two rows: symmetric bottom templates. Dashed lines represent edges of triangles potentially excluded if there are additional branches to the right, i.e., for trifurcation.

In the case where all the participating cross-sections are oriented vertex-forward, trifurcation can be performed trivially, by applying a mirror image of the structure



(a) (b)

Figure 5.16: (a) Pinching artifact caused by the application of a vertex-forward template to an edge-forward configuration; (b) Same configuration, with an edge-forward template applied.

Branch 1 Edge Forward	Branch 0 Edge Forward
2	1

(a) Top bitmask

Branch Edge Forward	Trunk Edge Forward
2	1

(b) Right and left bitmask

Branch 1 Edge Forward	Branch 0 Edge Forward	Trunk Edge Forward
4	2	1

(c) Bottom bitmask

Figure 5.17: Template-labeling bitmasks.

used for bifurcation (Figure 5.18). However, when edge-forward templates are incorporated, the front and back cross section edges are used by both templates. Therefore, to prevent overlapping triangles, the bottom subtemplates must be assembled with some awareness of their neighbouring branch. To this end, four more bottom subtemplates are added, shown in Figure 5.19. These templates are applied to the right of centre at trifurcated and higher-order junctions. To the left of centre, the appropriate templates from Figure 5.15 are applied, without the dashed lines. For the centremost child branch, the appropriate template from Figure 5.15 is applied, including, for edge-forward cases, the triangle whose rightmost edge is indicated by

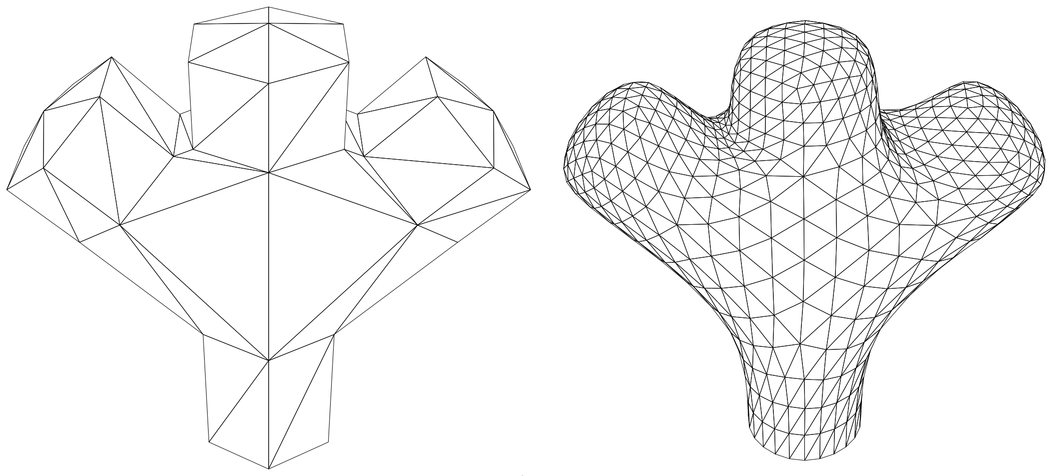


Figure 5.18: Trifurcated tree structure

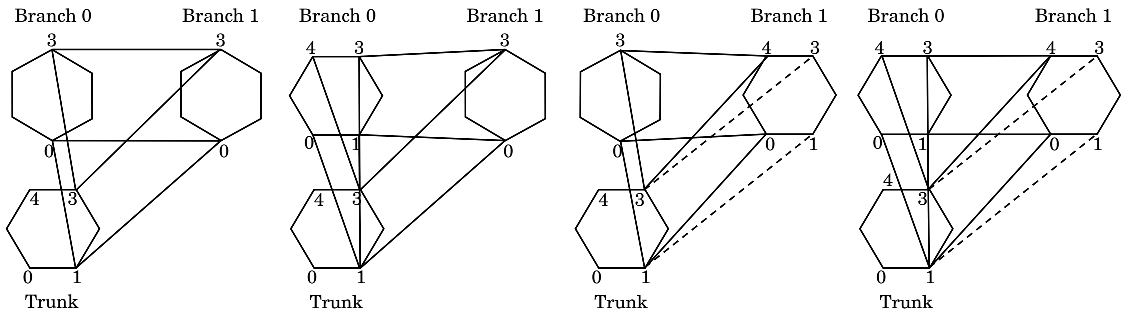


Figure 5.19: Bottom trifurcation templates. These templates are applied to branches right of centre at branching nodes with three or more child branches.

the dashed line.

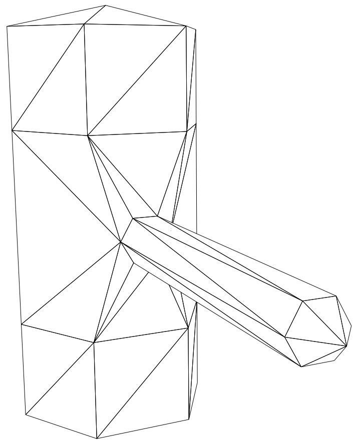
Asymmetric branching

For trees, symmetric and asymmetric branching are differentiated. *Symmetric* branching refers to the biological case of sympodial branching, where all outgoing branches are of similar radius and extend from their parent limb at opposite angles. Asymmetric branching commonly occurs in the botanical case of monopodial branching. Here, one finds a primary branch or trunk from which smaller *lateral branches* extend. In this case, a simple junction template one might use for uniform branching

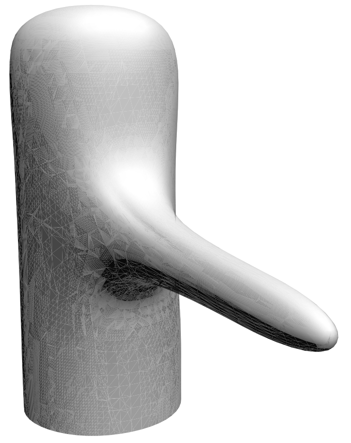
(Figure 5.20(a)) can produce unwanted creases at the crotch when subdivided (Figure 5.20(b)). Creating a more complex template, which isolates the smaller branch, somewhat solves this problem (Figures 5.20(c) and 5.20(d)). However, such a mesh tends to be too fine for the symmetric case and thus produces tighter blends than was intended (Figure 5.21). Therefore, an empirically-derived ratio of branch radii is used as a threshold, at or below which the asymmetric junction template is used. A given branch's radius is compared to its neighbour's radius. The threshold used for the images presented herein is one-third. Thus, if one branch's radius is less than or equal to one-third of the other branch's radius, the asymmetric template is used; otherwise, the symmetric template is used.

The asymmetric case uses more complex left and right subtemplates (Figure 5.22(a)) that add vertices as well as triangles. An asymmetric template produces the triangles in the topological positions that, in the symmetric case, would be produced by a top subtemplate, a middle subtemplate, and either a left or right subtemplate. As each side of the asymmetric case thus depends on vertex-forward versus edge-forward for three cross sections, there are 2^3 templates for the left side and another eight for the right side, resulting in a total of sixteen possible asymmetric templates. The case with both branches vertex-forward is shown in Figure 5.22(b). These templates create a ring of vertices near the base of the smaller-radius branch.

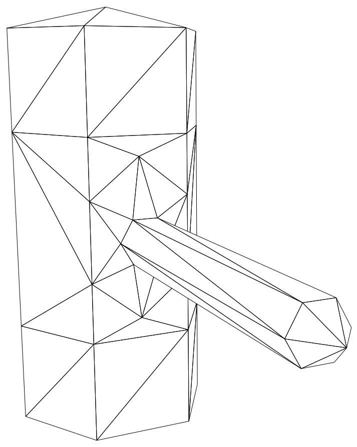
In many cases, the portion of the branching structure exhibiting artifacts when a symmetric template is applied to a lateral branch is hidden from view. Therefore, the extra effort and machinery to explicitly handle asymmetry is of minimal value, so the remaining fourteen (of the aforementioned sixteen) asymmetric templates are left as future work.



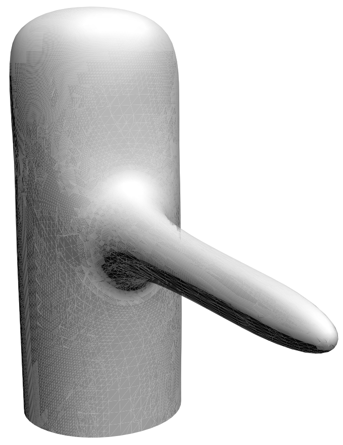
(a)



(b)

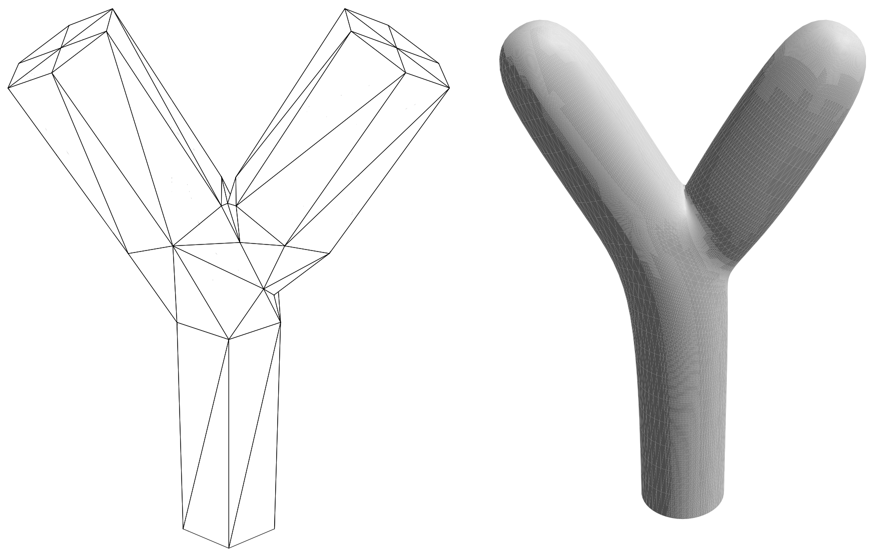


(c)



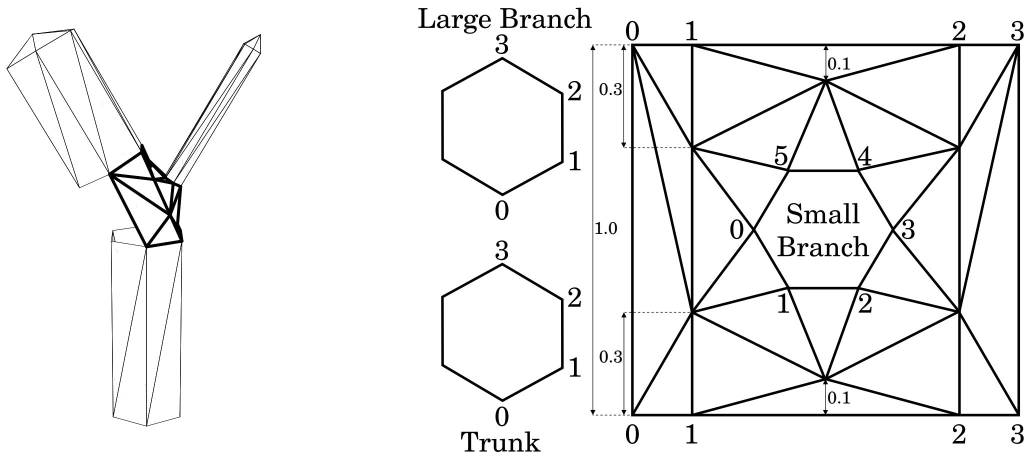
(d)

Figure 5.20: Asymmetric branching. (a, b): Symmetric junction template applied to asymmetric branching configuration; (c, d): Asymmetric junction structure. (a, c): Control meshes; (b, d): Refined meshes, after three iterations of Loop subdivision.



(a) Control mesh. (b) After three iterations of Loop subdivision.

Figure 5.21: Asymmetric junction template applied to a symmetric branching configuration.



(a) Template placement. (b) Right asymmetric template.

Figure 5.22: Right asymmetric template. Only the all-vertex-forward case is shown. The left template is identical except for the indices connected.

Chapter 6

Texturing

Smooth-shaded surfaces are often an adequate approximation for a computer graphics model. However, to achieve visual verisimilitude, a model must include some surface texture detail. In particular, leaf models need veins and tree branches need bark.

Various techniques have been applied to texturing leaves (Section 6.1) and branches (Section 6.2). In the context of this research, three texturing methods were investigated: solid texture (Section 6.3), texture mapping (Section 6.4), and fractal noise (Section 6.5). Numerous other methods could be adapted to texturing of branching structures, several of which are proposed as future work (Section 6.6).

6.1 Previous leaf texturing methods

Leaves are typically represented by a small number of polygons textured using *texture mapping* [Cat74]. The texture image may consist of acquired data, for example, from a photograph [Blo85] or a flatbed scanner [Mar03], or be synthetically generated [THB02]. With texture mapping, the texture image may be used as both a colour map and as a *bump map* [Bli77], simulating the effect of light on a rough surface.

6.2 Previous branch texturing methods

The texturing method used for branches is typically dependent on the modeling methodology chosen. A 2D parameterization can easily be derived for generalized cylinders, making texture mapping them an attractive option. As with leaves, a bark texture image may be derived from photographic data [Blo85] or it can be synthesized [Fed02, LN02]. The resulting images may be used as colour, bump, or displacement maps [Coo84]. Recently, view-dependent displacement mapping has been used to texture tree models in real time [WWT+03].

Texturing implicit surfaces is more complicated, as they do not have an inherent 2D parameterization. One approach is to use a support surface to parameterize each skeletal primitive and interpolate the resulting texture coordinates across blend regions [TW99]. Alternatively, solid texture has been applied to convolution surface tree models [JTFP01]. Rough tree bark has also been modeled on implicit models of branches using a particle flow approximation [HB96].

6.3 Solid texture

A simple, yet effective, approach to texturing is to use a solid texture [Pea85, Per85], as demonstrated in Figure 7.8. Solid texture uses three-dimensional textures, eliminating the need for a 2D surface parameterization. Many 3D texture patterns allow distortion-free texturing for any arbitrary shape. A disadvantage of solid texture is that it is difficult to handle textures with strong features that are oriented along a surface, such as venation patterns and bark fractures. Therefore, the use of texture mapping, for both leaves and branches, was also explored.

6.4 Texture mapping

Texture mapping [Cat74, WD97] involves globally assigning locations on a 2D image to vertices on a 3D surface. Typically, the effect is to wrap the 2D image around the 3D mesh as if the 2D image was a decal or wallpaper. The 2D coordinates are referred to as *uv* coordinates. During rendering, the renderer linearly interpolates between the *uv* coordinates of neighbouring vertices to compute pixel colours. The texture image may be filtered to reduce aliasing artifacts. Filtering is necessary for the common case where the 2D image is rendered in a 3D scene at a resolution different from its native resolution. Texture mapping allows a highly detailed texture to be applied effectively to a low resolution mesh. The entire process is accelerated by modern graphics hardware.

The key issue with texture mapping a surface is defining a 2D parameterization. For a subdivision surface, this requires generating *mapping coordinates* for the coarse mesh. The mapping coordinates are then subdivided along with the geometric coordinates [DKT98].

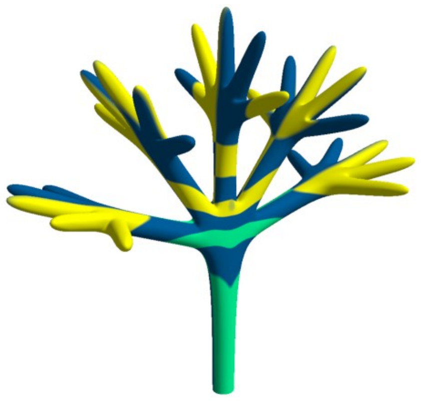
Generating mapping coordinates for a mesh can be difficult, as there is no inherent continuous 2D parameterization of an arbitrary surface. One approach is to project a mapping onto the mesh from a parametric support surface [BKRS86, Tig99]. Two common projections are sphere mapping and cylinder mapping [BKRS86]. Their effects on a branching structure are shown in Figure 6.1. Sphere mapping (Figures 6.1(a) and 6.1(b)) entails generating texture coordinates by projecting each vertex onto the unit sphere. Cylinder mapping (Figures 6.1(c) and 6.1(d)) similarly projects to the uncapped unit cylinder. u is wrapped around the cross-section of the cylinder so that the texture's edges touch. v is mapped linearly from the bottom to the top of the cylinder. Both mappings can be accomplished either by projecting the normalized position of the vertex relative to the mesh center, or projecting the vertex normal.

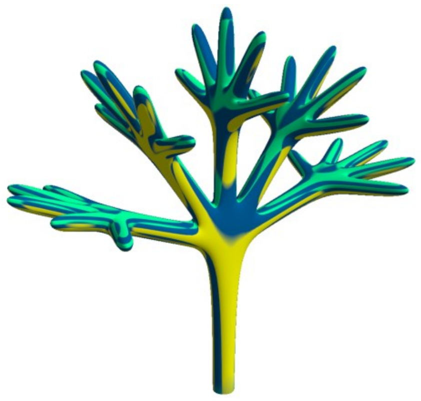
As demonstrated in Figure 6.1, globally-applied projective mappings don't tend to follow the shape of a branching structure well. Therefore, specialized texture mapping methods were developed for leaves and stems.

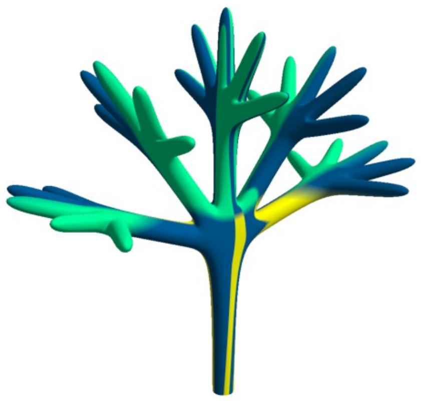
6.4.1 Leaves

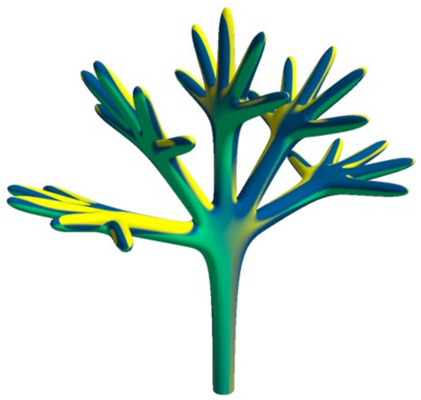
For leaves, one common approach is to map a single texture image to the whole leaf, as if the leaf geometry was cut from the texture image [Blo85, MMPP03]. Since the leaves are 3D shapes, a correspondence between the leaf surface and the 2D image plane needs to be defined. To this end, two interpretations of a given turtle string are performed. The first interpretation generates a *proxy mesh* (Figure 6.2(b)). Bend and twist are ignored when generating the proxy, allowing only rotations around the *Up* axis. This ensures the resulting mesh is in the XY plane. The geometry of the proxy mesh does not necessarily correspond to the texture image, so they are manually aligned (Figure 6.2(c)), using the interactive software described in Appendix A.3. During the second interpretation of the mesh, all rotations are processed. Finally, the x and y values of the proxy mesh's vertices are assigned to the u and v texture coordinates of the final mesh (Figure 6.2(d)).

Another approach investigated was mapping u across the width of a branch while mapping the v domain to an *internode*, a branch and its originating junction. With this method, the texture image corresponds to one *lobe* of a leaf (Figure 6.3). This


(a) Sphere map, using positions.


(b) Sphere map, using normals.


(c) Cylinder map, using positions.


(d) Cylinder map, using normals.

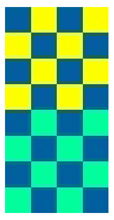

(e) Checker texture.

Figure 6.1: Projective texture mapping.

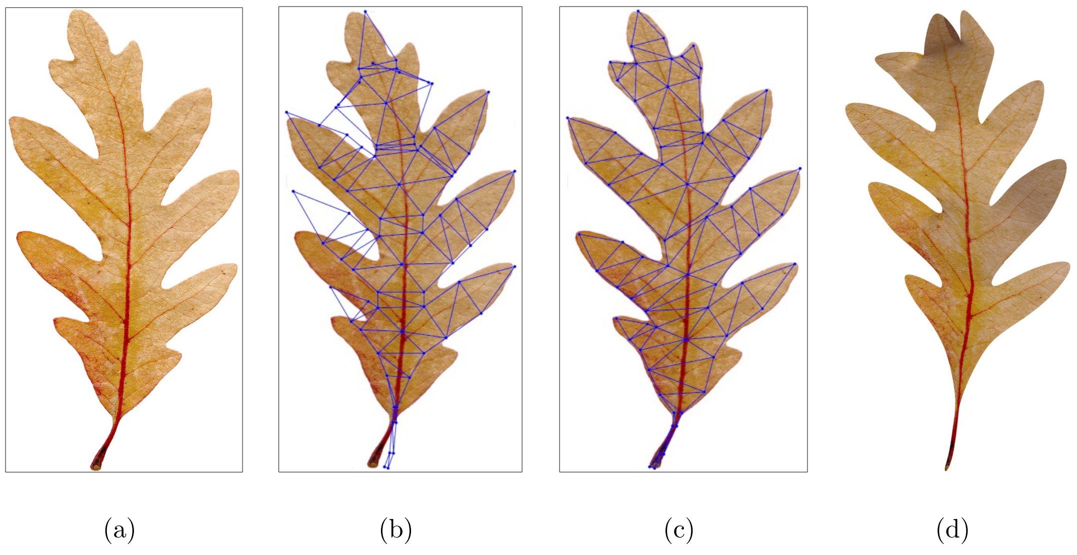
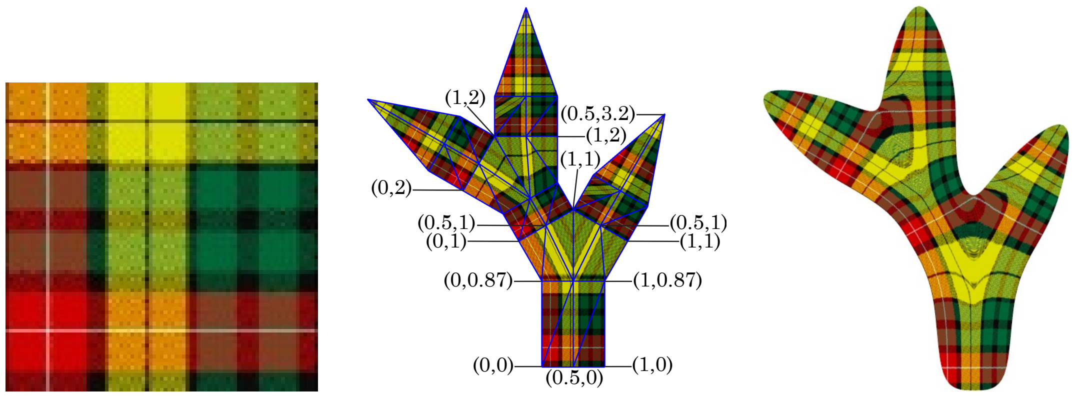


Figure 6.2: Leaf texture mapping: (a) oak texture; (b) proxy mesh; (c) proxy mesh edited to fit the texture; (d) final model, subdivided and textured.

entirely-procedural method requires vertically-tileable textures, and was used for Figures 7.4 and 7.11.

6.4.2 Stems

For stems, an ideal texture mapping would be similar to what has been achieved using generalized cylinders [Blo85, LN02]. Unfortunately, subdivision surfaces cannot easily be parameterized in the manner of generalized cylinders. Generalized cylinders are parameterized by wrapping the texture's u coordinate domain around the cylinder, creating a seam where the u coordinate is 0.0 on one side and 1.0 on the other (Figure 6.4(a)). An equivalent parameterization on a subdivision surface would require multiple texture mapping coordinates at vertices on the seam. Instead, texture coordinates can be assigned around the cross section of a cylinder which reflect



(a) (b) (c)

Figure 6.3: Tiled leaf coordinate mapping: (a) tartan texture map; (b) coarse mesh with (u, v) coordinates; (c): refined mesh, showing distortion at junctions but with the flow of the texture preserved.

the parameter range [TMW02a]. This ensures a unique texture mapping coordinate for each vertex (Figure 6.4(b)). However, applying the parameterization from Figure 6.4(b) to a hexagon results in edges with the same u coordinate on either end (Figure 6.4(c)). For branching as described in Chapter 5, a symmetric mapping is required. Any symmetric rotation of the mapping in Figure 6.4(c) would have the same problem. Therefore, to prevent discontinuities, the texture's entire u domain is mapped to each cross-section edge, reflecting the u domain across each cross-section vertex (Figure 6.4(d)).

Generating texture mapping coordinates for branching structures is difficult, so an heuristic solution was developed which works well for many textures, despite producing noticeable distortions at junctions. The u coordinate is held constant across the junction structure itself; this distorts the texture at the branching points, but also prevents discontinuities. The v domain is mapped from the base of the stem

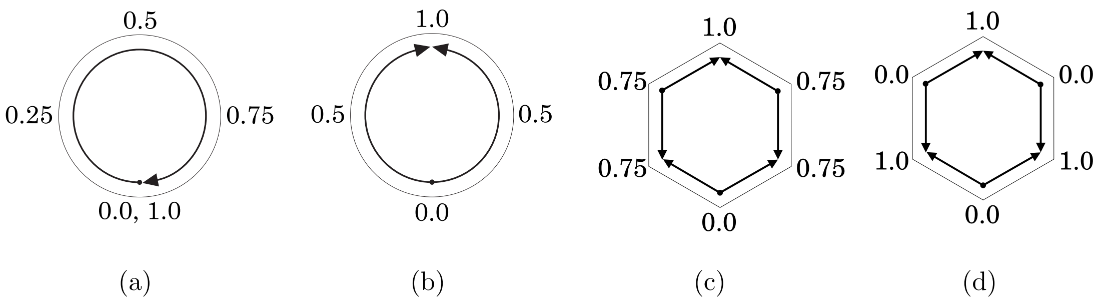


Figure 6.4: u coordinate mapping around the cross section of a cylinder: (a) standard mapping used for cylinders; (b) reflected mapping, suitable for subdivision surfaces; (c) reflected mapping, mapped to a hexagonal cross section; (d) the proposed method of reflecting the u domain across each vertex of a hexagonal cross-section.

to the tip of the most distal branch (Figure 6.5). Since space is introduced between the end of a trunk and the base of its child branches, the v coordinates for the child branches must be incremented appropriately across the junction. The increment is proportional to the ratio of the height of the junction structure to the length of the trunk.

A disadvantage of this approach, as implemented, is that the texture coordinates tend to shrink away from the edges of the texture map, much as the vertices shrink away from the control points, causing artifacts when using tiled textures (Figure 6.6). This could potentially be alleviated to some extent by using an interpolating scheme, such as the Butterfly scheme [DLG90], on the texture coordinates.

6.5 Fractal Noise

Due to the distortion artifacts inherent in texture mapping of branches, a geometry-based local approach to texture generation was briefly investigated. The hope was that a local approach would produce a texture resembling bark without the distor-

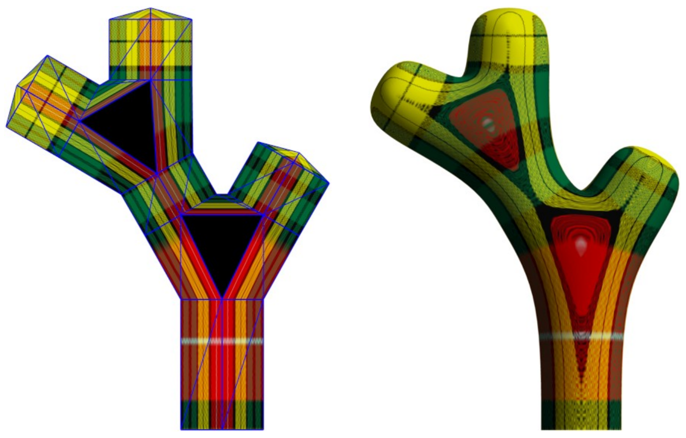
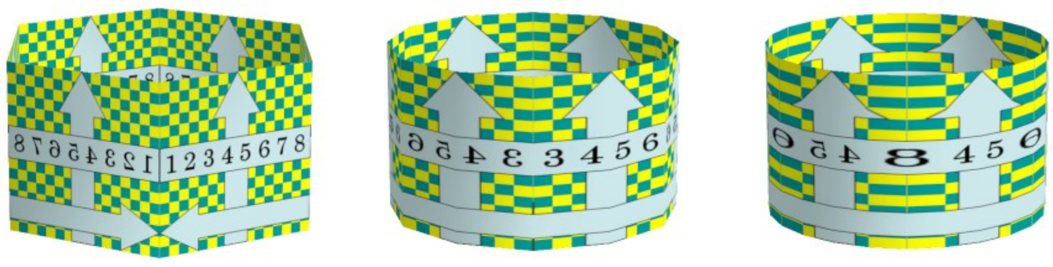

(a) (b)

Figure 6.5: v coordinate mapping: (a) coarse mesh with the v domain mapped to the height of the entire plant; (b) refined mesh, showing distortion at junctions but with the flow of the texture preserved.



(a) Coarse mesh. (b) Subdivided once. (c) Subdivided twice.

Figure 6.6: Applying Loop subdivision to texture mapping coordinates: when an approximating subdivision scheme is applied to texture mapping coordinates, the mapping coordinates tend to shrink away from the image's borders.

tions at branching points seen when using texture mapping. Fractal noise is a simple means of generating rough textures that is well suited to subdivision. Simple triangle subdivision with fractal noise can been used to generate heightfields for terrain modeling [FFC82]. Alternatively, terrain can be generated by applying noise to a smooth subdivision scheme. Gavin Miller [Mil86] proposed a method for terrain generation that incorporates fractal noise into a version of Doo-Sabin subdivision [DS78], simplified for use with heightfields. Specifically, Miller perturbed the Y coordinates of a randomly-generated heightfield's vertices between each subdivision step.

In this research, an adaptation of Miller's technique is used to apply noise to branches between each Loop subdivision step. Each vertex is perturbed along its normal. As in [Mil86], the standard deviation of the perturbation's magnitude is generated using the formula $S = kL^{-iH}$. In this case, k is a scaling factor initially determined by the size of the mesh, L is the lacunarity [MKM89, Man94], and i is the iteration level. Noise is not applied to the coarse mesh's vertices, so i starts at 1. H is the desired fractal dimension, a value in the range $[0, 1]$ that determines how rough the resulting surface looks.

To simulate variation in bark thickness and roughness due to age, k is additionally scaled by a feature-size factor computed for each vertex as the surface is generated. A nonlinear rate of decrease in feature size can be achieved by using the Bias function (Figure 6.7) proposed by Perlin in chapter 9 of [EMP+94]: $\beta(t) = t^{-\frac{ln(b)}{ln2}}$. This function is intended to alter the result of a one-dimensional function to favour either its low or high values. It takes two parameters: a bias factor b which controls the slope of the function, and the point t on the function to evaluate. Both parameters to Bias() must be in the range $[0, 1]$, with a bias of 0.5 resulting in a linear decrease in noise feature size. t starts at 1.0 at the base of a tree and is reduced at each branching point. The effect is to decrease the noise feature size for younger branches. Bias() produces a value in the range $[0, 1]$, with a value of 0 eliminating noise altogether

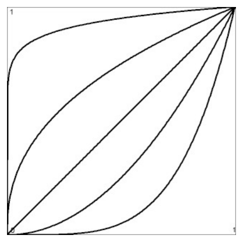


Figure 6.7: Bias function, for b =0.05, 0.25, 0.5, 0.75, and 0.95.

and 1.0 allowing full noise.

An example branching structure with fractal noise applied is shown in Figure 6.8. The method exhibits consistent behavior at branching points, with no noticeable distortions or discontinuities. A downside of this approach is that good results require five or six iterations of subdivision. This can easily produce a gigabyte of geometry data for a modestly bifurcated model. Furthermore, this method is not appropriate for simulating the venation patterns characteristic of leaf textures. Thus, despite their limitations, solid texture and texture mapping are the preferred methods for texturing plants.

6.6 Future work

Since bark texture in nature is formed by local processes, it seems a natural fit to use a local process to generate synthetic texture. This notion is reinforced by the distortion-free fractal texture just presented. Unfortunately, the large meshes produced by that method were prohibitive. Reaction-diffusion [Tur52, WK91, Tur91] is another texture synthesis technique that uses local rules and could thus be applied to subdivision surfaces. However, it is based on per-face colouring, so a large number of triangles would have to be rendered to display a detailed pattern. We

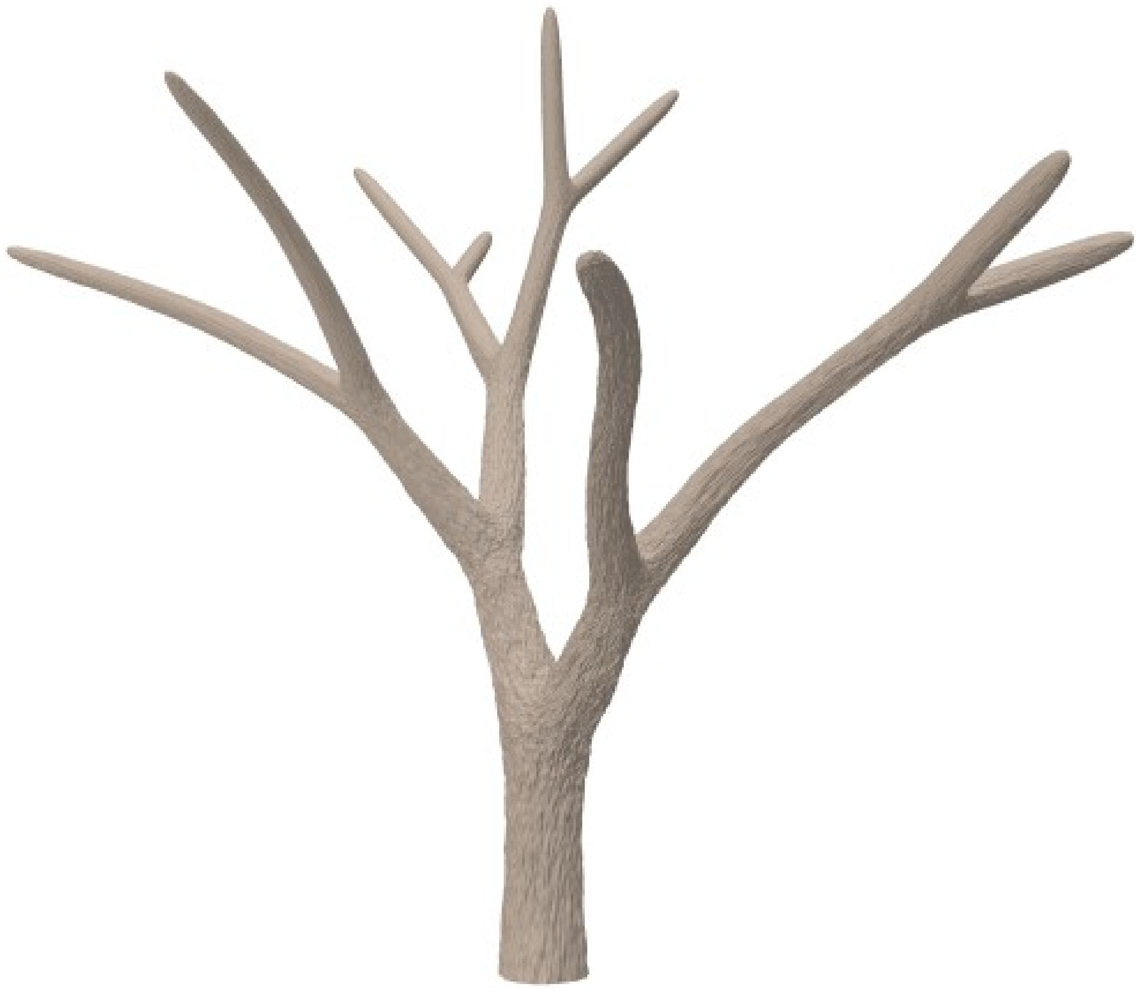


Figure 6.8: Branching surface with six iterations of fractal noise. The L-system used for this model was adapted from [PJM94].

need a method for local specification of texture that allows texture features with a substantially higher frequency than the mesh.

Fortunately, recent developments in the area of procedural texturing have presented several different techniques for local specification of texture. These algorithms are typically quite involved but produce impressive results for any mesh topology. Texture synthesis [Tur01, WL01, LLX$^+$01, ZZV$^+$03] starts with a small user-supplied texture sample and procedurally synthesizes a complete texture directly on a model's surface. Pattern-based texturing [Sta97, NC99] involves judiciously selecting a few triangular texture samples with certain boundary conditions and aperiodically placing these on the mesh.

Both of these methods require a separate mesh for placing the texture. This mesh is usually derived from the target mesh by retiling [NC99, Tur01]. This step could be avoided for subdivision surfaces, as subdivided meshes are already semi-regular with locally-consistent mesh density: virtually ideal for placing or synthesizing texture samples.

6.7 Summary

Several approaches have been presented for texturing meshes derived from the methods in Chapter 5. For leaves, a texture mapping method with an interactive phase was proposed, with results similar to previous leaf texture mapping approaches. An entirely procedural approach for texture mapping leaves was also briefly introduced. For stems, solid texture, texture mapping, and fractal noise were used. As none of these methods are entirely satisfactory, it was suggested that future work explore the application of advanced local texture specification techniques to plant models.

Chapter 7

Implementation and Results

7.1 Implementation

The implementation of the proposed method combines the L-system simulation program CPFG [Měc98] with the surface-creation program implementing the algorithms discussed in this work. Given a string produced by CPFG, this latter program makes it possible to interactively preview, subdivide, and texture the resulting meshes. It also supports rudimentary mesh editing operations. In the case of the models presented in this paper, these operations were only applied to the proxy meshes for leaf texture mapping purposes (Section 6.4.1). The final models are output in either OBJ or POV-Ray format. All of the models presented in this section were rendered using POV-Ray [pov]. An overview of the system's architecture is presented in Figure 7.1, and a user manual, describing both the graphical and command-line interfaces, is presented in Appendix A.1.

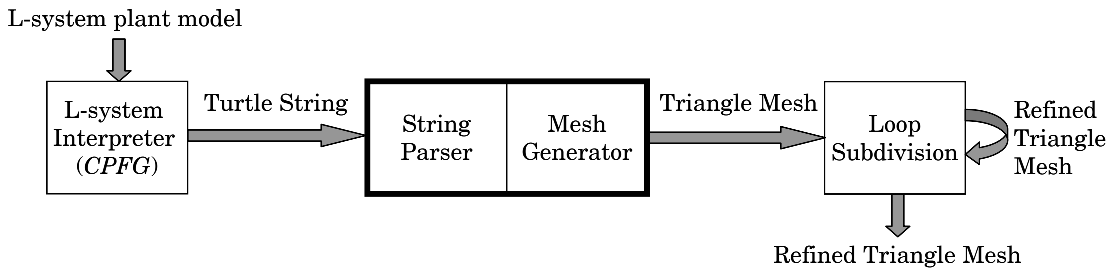


Figure 7.1: System overview

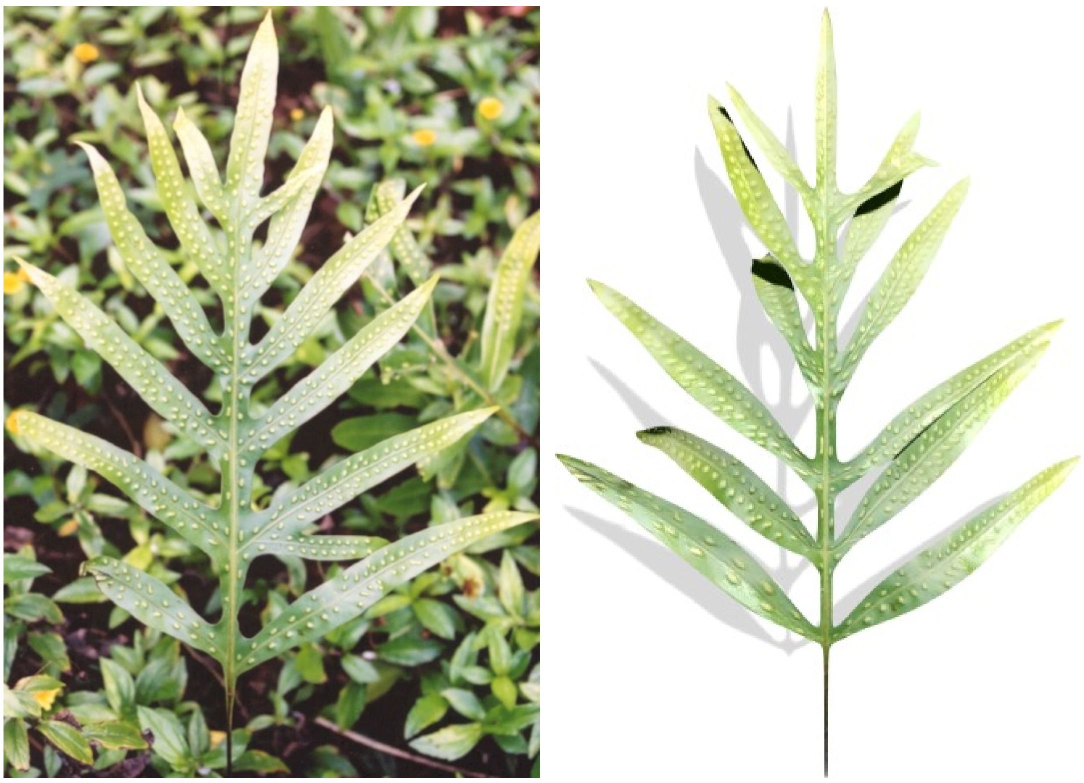


Figure 7.2: Photograph and model of a Hawaiian fern leaf. Photo courtesy P. Prusinkiewicz.

7.2 Results

Figures 7.2, 7.3 and 7.4 show results from the methods, as applied to the modeling of leaves. All three models display the expected smooth blending between lobes. For Figures 7.2 and 7.3, the models were texture mapped using the method described in Section 6.4.1. The Elkhorn model is the most geometrically complex of the three, yet the coarse mesh was generated in well under one second. The mesh was then subdivided thrice, producing 48,256 triangles in 780 ms on a 733MHz Pentium III.

Figures 7.5 to 7.8 illustrate the method using models of tree branches and a young tree. Figure 7.5 shows a twice-subdivided mesh of a model similar in structure to the arbutus tree from Figure 2.2. Figure 7.6 shows another structure with wide branches,

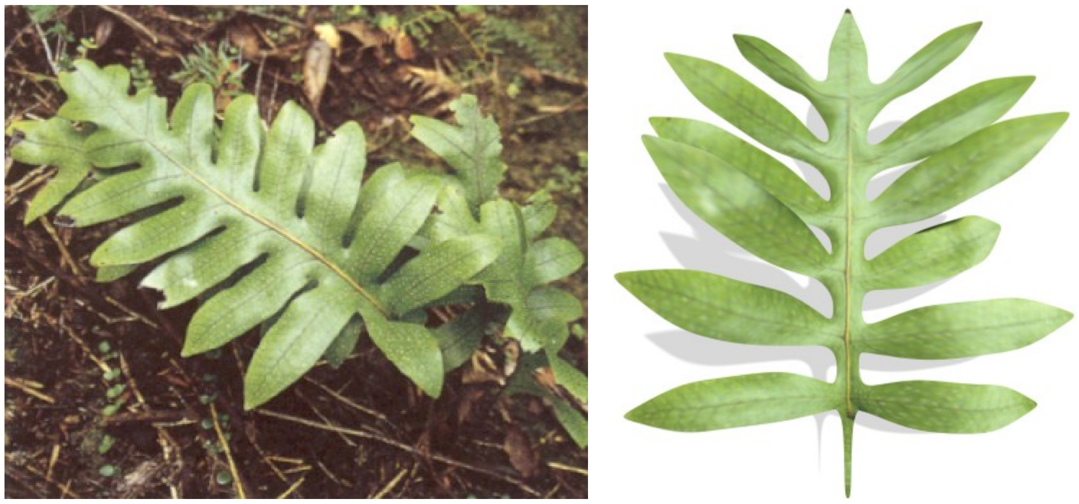


Figure 7.3: Photograph and model of Phymatosorus pustulatus.

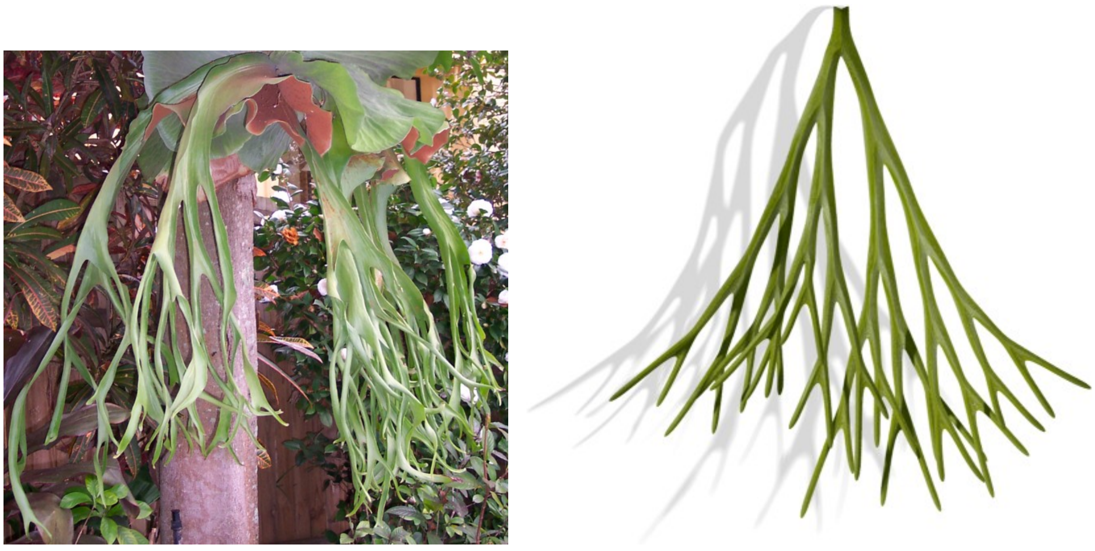


Figure 7.4: Photograph and model of an Elkhorn fern leaf. Photo courtesy Birgit Loch, University of Queensland.

demonstrating the basic symmetric bifuraction templates from Figure 5.9. For the driftwood model, parsing and three iterations of subdivision each took under a second, producing 58,496 triangles. It was rendered using texture mapping, as described in Section 6.4.2. The model shown in Figure 7.7 combines a branching structure and leaf models into a whole-tree model. The tree demonstrates both bifurcation and trifurcation templates. Finally, the poplar model shown in Figure 7.8 is the most complex one, as the stems includes the scars from discarded branches. The scars use the basic bifurcation templates with no special considerations necessary. Generating the coarse mesh for the poplar took under a second, while three iterations of Loop subdivision generated 202,240 triangles in about five seconds. The refined mesh was rendered using POV-Ray's implementations of solid textures and bump maps. As with the leaves, all these figures display the expected smooth blending of branches.

Figures 7.9, 7.10, and 7.11 demonstrate the twig and leaf methods incorporated into scenes.

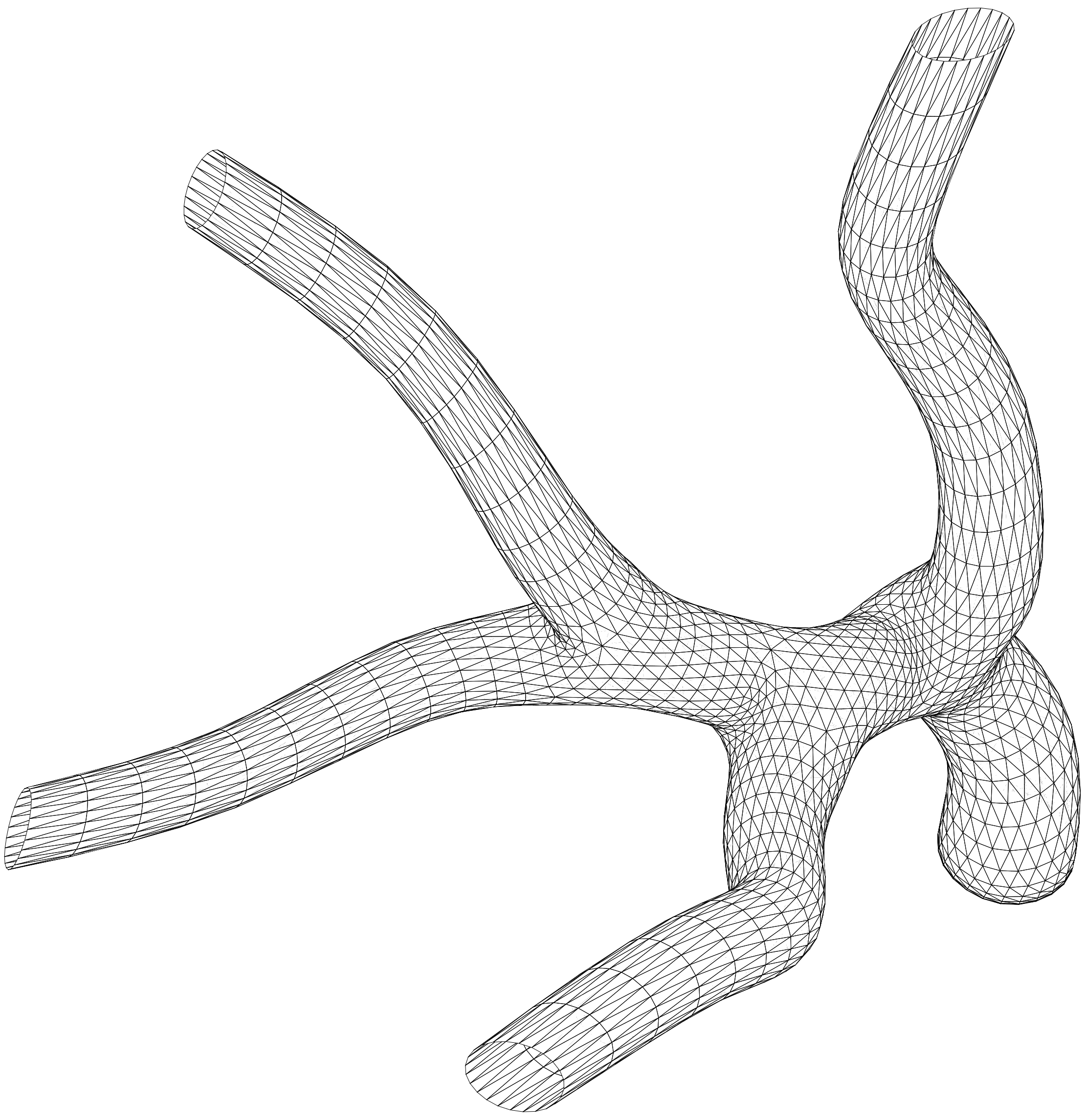


Figure 7.5: Subdivided branching mesh; compare with Figure 2.2.

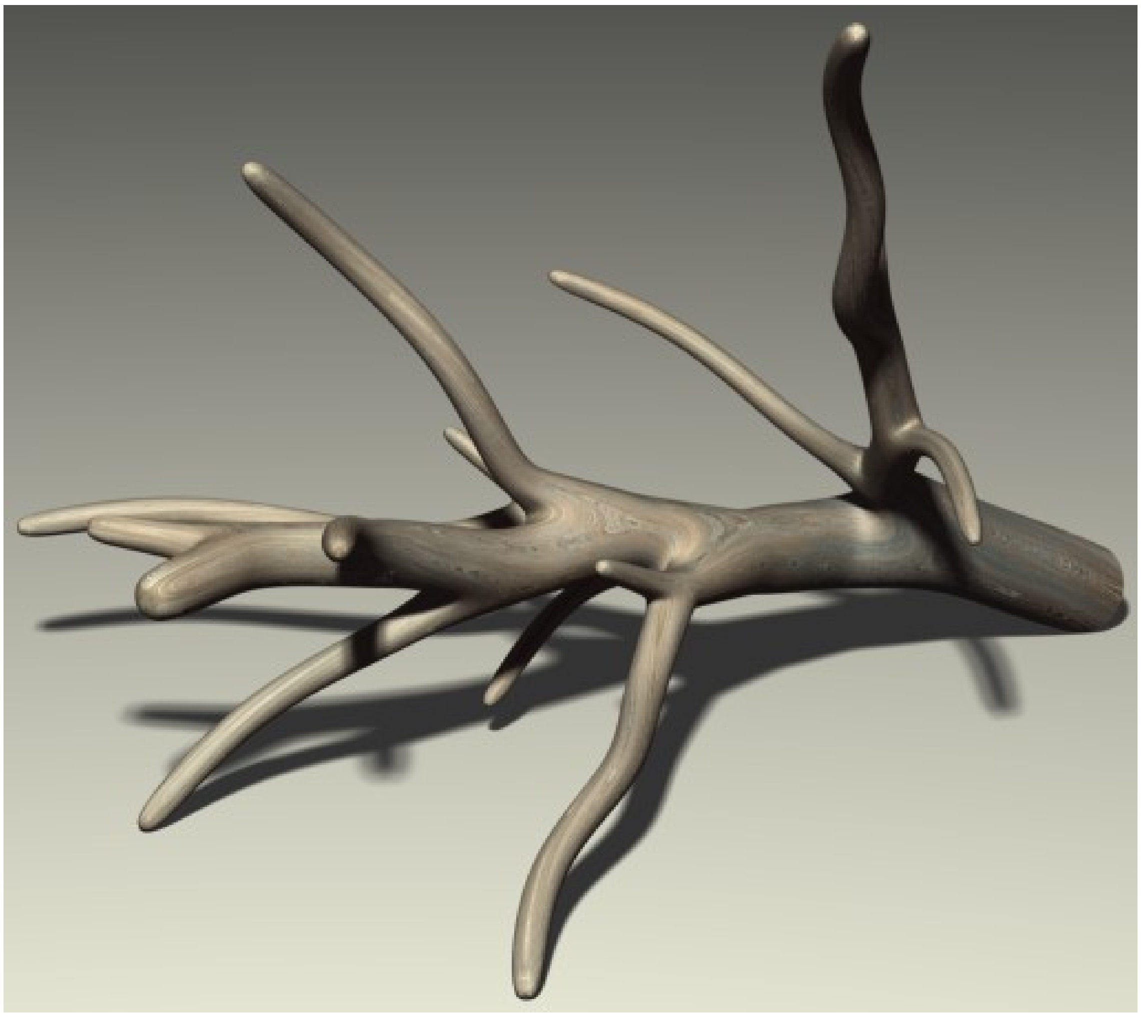


Figure 7.6: Driftwood

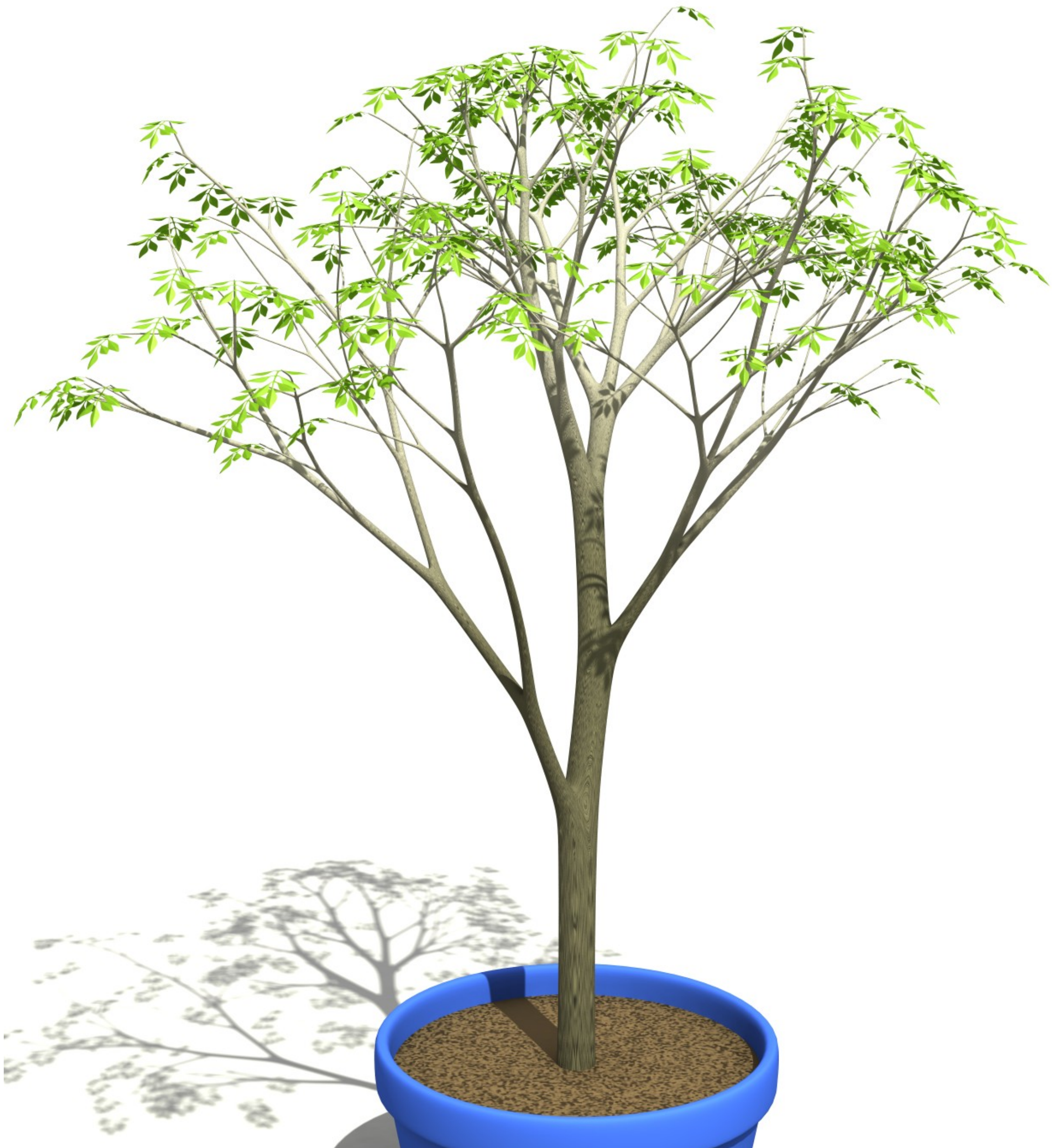


Figure 7.7: Spring sapling

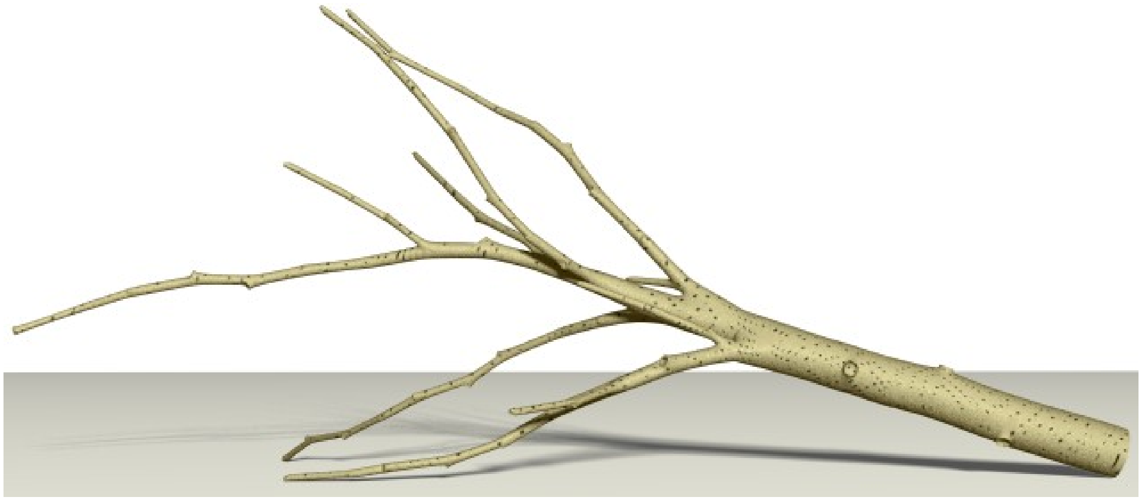


Figure 7.8: Poplar twig

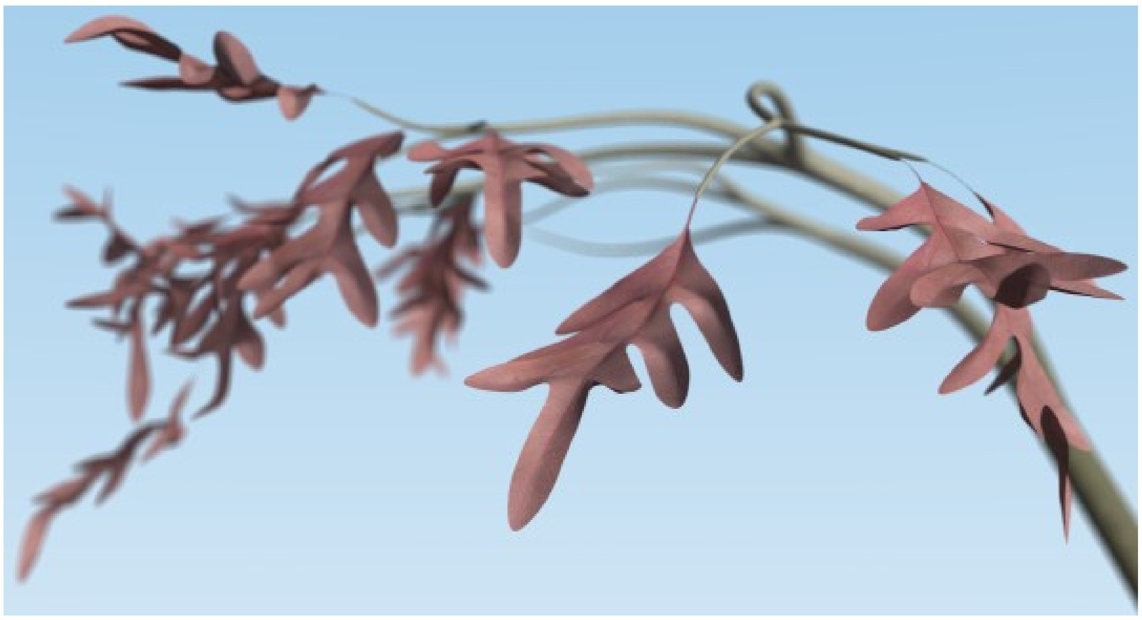


Figure 7.9: Oak bough

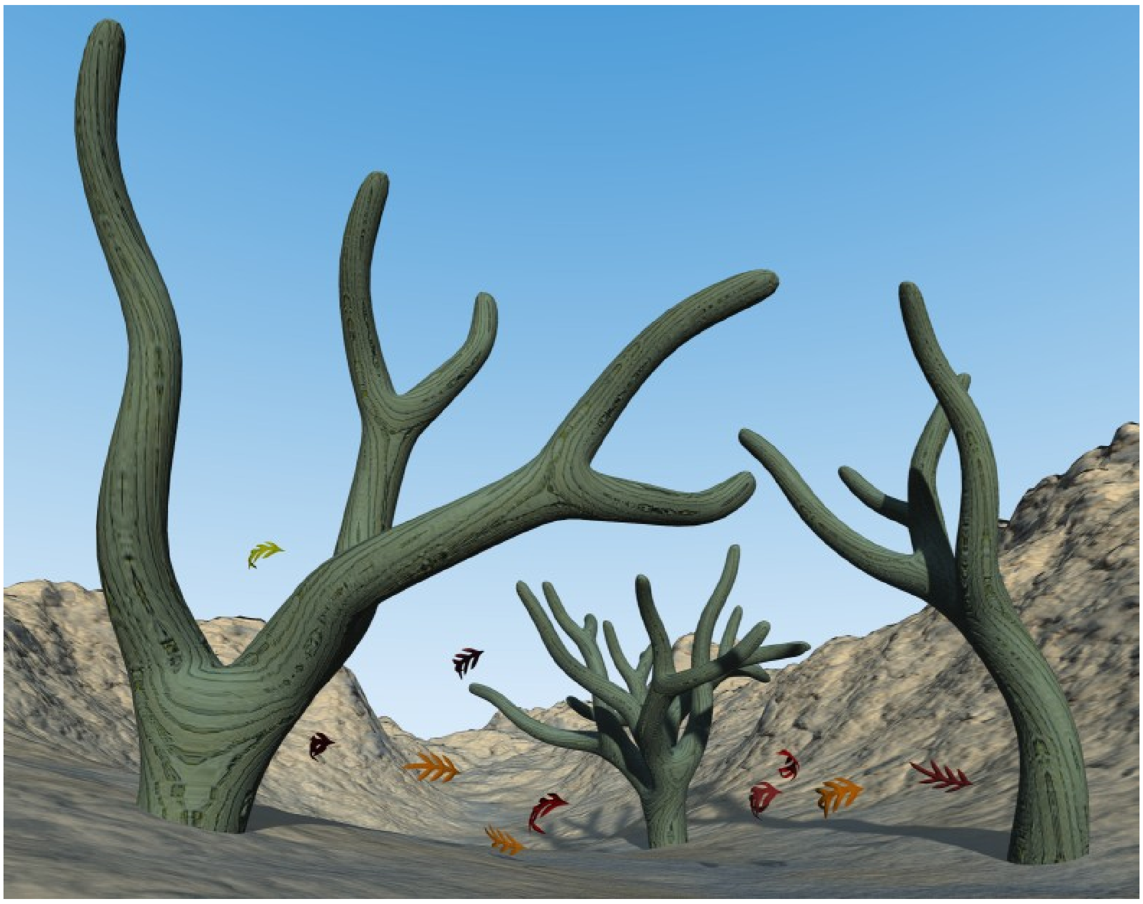


Figure 7.10: “The Bigger They Are, The Harder They Autumn”

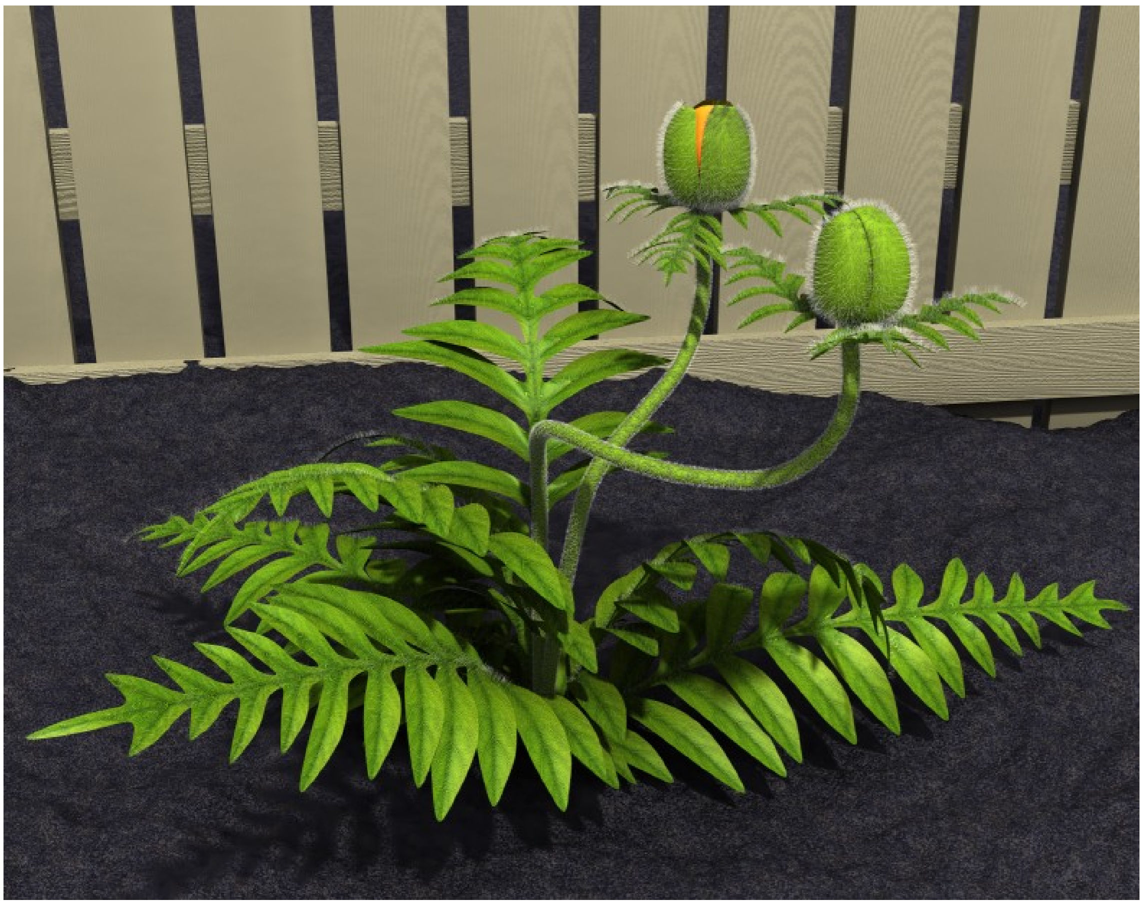


Figure 7.11: “I Know What You Grew Last Summer”

Chapter 8

Comparison Between Subdivision and Implicit Surfaces

8.1 Comparison

The key alternative to subdivision methods for creating smooth branching points uses implicit surfaces [Blo95b, Mar03]. Therefore, the use of subdivision surfaces was compared with the creation of implicit surfaces, given the same L-system-generated skeletons. The models are shown in Figure 8.1. The comparisons were obtained with a recent implementation [FGW00] of the skeletal implicit surface modeler *Blob-Tree* [WGG99]. By comparing the columns in Figure 8.1, it can be observed that the models shown have comparable surface quality, with the subdivision method having produced somewhat smoother surfaces.

The comparison results are collected in Table 8.1. The implementation of Loop subdivision used produces about eight times as many triangles per second as the

Model	Method	Time [sec]	Triangles	Triangles/sec
Scars	Implicit	115.28	726964	6306.07
	Subdiv (2 iterations)	3.05	153280	50255.74
	Subdiv (3 iterations)	12.84	613120	47750.78
No scars	Implicit	46.38	294528	6350.32
	Subdiv (2 iterations)	1.43	73632	49088.00
	Subdiv (3 iterations)	6.28	294528	46899.36

Table 8.1: Comparison of the performance of implicit and subdivision methods. Tests were run on a dual Pentium III 733MHz.

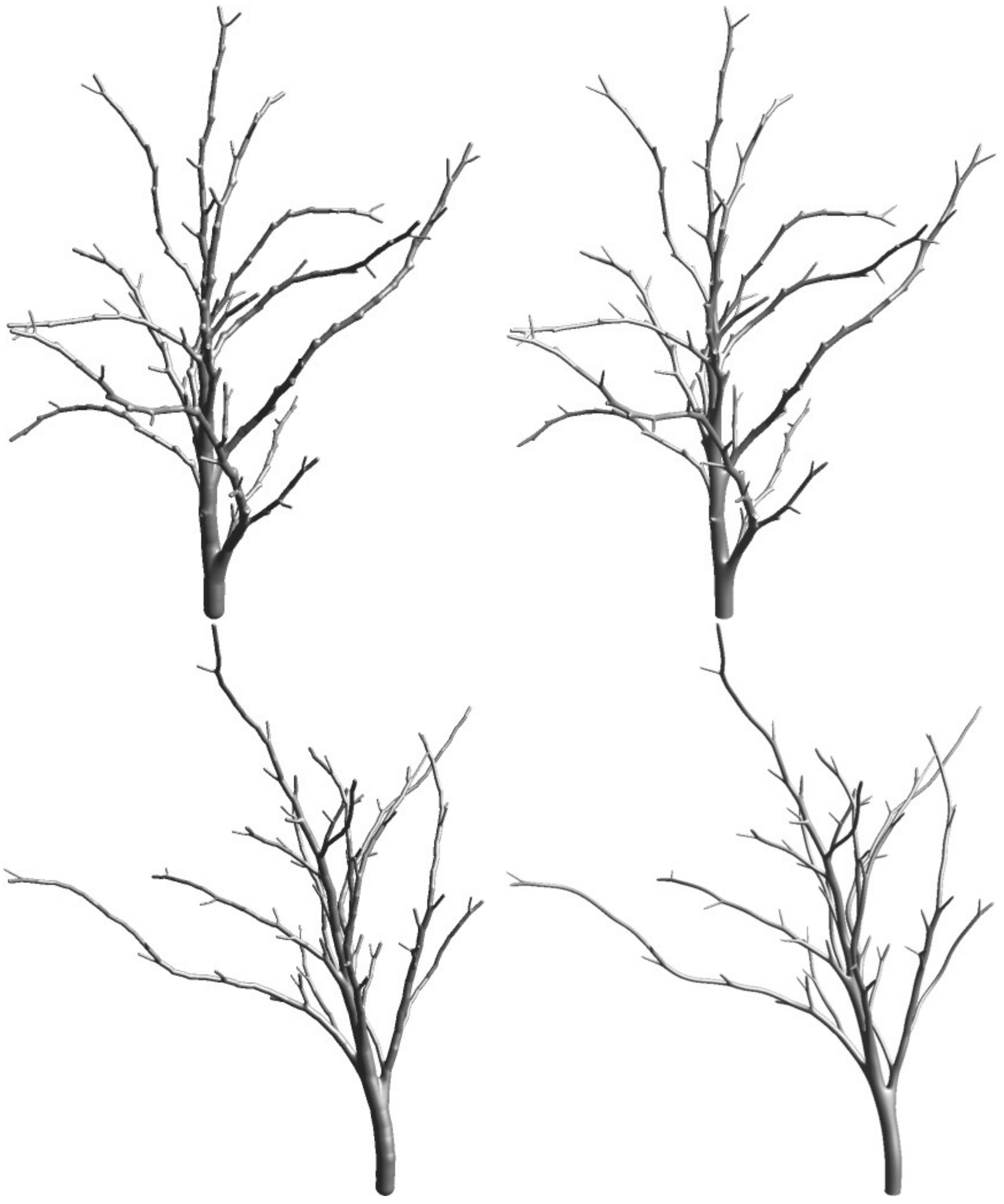


Figure 8.1: Surface comparison. Top: “Scars” model; bottom: “No scars” model. Left: Polygonized implicit surface; right: surface produced by the proposed method after two iterations of Loop subdivision. All images were rendered using OpenGL at 1024^2 resolution.

BlobTree's surface tiler. Furthermore, only two levels of subdivision were needed to produce meshes of the same visual quality as those produced by the implicit surface method for the given resolution (1024^2 pixels). Thus, the number of produced triangles was several times smaller for the subdivision method. Taking this into account, subdivision was over thirty times faster than the implicit method.

8.2 Discussion

Part of subdivision's advantage likely comes from the fact that the coarse mesh is already a topologically accurate representation of the branching structure. In fact, this suggests another advantage: the coarse mesh is already suitable for interactive previews and rendering models at a distance, whereas a *BlobTree* surface polygonized to a similar number of triangles would likely be topologically inaccurate and fail to capture thin limbs.

The *BlobTree*'s polygonizer uses a continuation method based on uniform voxel subdivision [WMW86]. Other polygonization algorithms exist, but were not locally available for comparison. Some of these, such as the particle-based method of Witkin and Heckbert [WH94], are known to produce more regular triangles. Additionally, several of the alternative methods, such as Marching Triangles [HSIW, AG01], can be faster than uniform-voxel continuation. These methods typically exhibit two to four times the performance of the polygonization algorithm tested by requiring approximately one-quarter of the triangles to produce an equivalently accurate approximation. This would produce triangles at half of the rate of the Loop subdivision, resulting in polygonization times about eight to ten times longer than with subdivision, for comparable visual quality. Thus, subdivision is likely to keep much of its performance advantage, even with the application of the highest-performance implicit surface techniques.

Future work would be to investigate the use of adaptive subdivision [ZSS97, AFR02] and adaptive implicit surface polygonization [VdFG99], which may reduce the performance and triangle count discrepancies.

Chapter 9

Conclusions

In this work, a method for generating leaves and stems with smoothly blended branches, based on the use of subdivision surfaces, has been presented. The method produces results visually comparable to those obtained using implicit surfaces, but yields a smaller number of triangles and is over an order of magnitude faster. A modeling system for creating subdivision surfaces for skeletal branching structure has also been implemented and illustrated with examples. The realism of the final renderings was enhanced using solid textures and texture mapping.

Previous work involving generating subdivision surfaces for branching structures includes *Mesh-Based PL-systems* [TMW02a, Mai02, TMW02b] and *Surface Models from by-Axis-and-Radius-defined Tubes* [FFW01, FFKW02]. The former authors used growing meshes, rather than skeletons, while the latter focused only on branching tubes with square cross sections and symmetric branching. The proposed method adds to the previous research a skeletal method for modeling compound leaves as well as considerations for modeling asymmetric branching structures.

The current method has several limitations, which present problems open for further research:

- In nature, many trifurcations are asymmetric, which means that one branch is slightly offset with respect to the other branch along their supporting axis (in other words, the trifurcation consists of two nearby bifurcations). The proposed template method does not support such branching configurations, whereas implicit surfaces would.

- In many plants (e.g., in coniferous trees), branches occur in *whorls*, where several branches are arranged in a tight spiral pattern around the trunk. This is a generalization of the case described above, so the template method does not support these configurations either.
- Ideally, leaves and stems should be integrated into one continuous mesh. The methods presented in this work support leaves and stems separately, but do not support a continuous connection between the two classes of objects. With stems modeled as closed surfaces and leaves as open surfaces, as with this research, attaching leaves to stems would result in a non-manifold surface. The importance of non-manifold surfaces to the modeling of biological objects was observed by Bloomenthal [Blo95b]. Non-manifold subdivision exists for Loop's scheme [YZ01], but has only C^0 continuity at non-manifold joints, making it equivalent to using multiple abutting meshes, as the proposed methods already do.
- The texture mapping introduces significant distortions at the branching points, which in some cases produce visually noticeably artifacts.

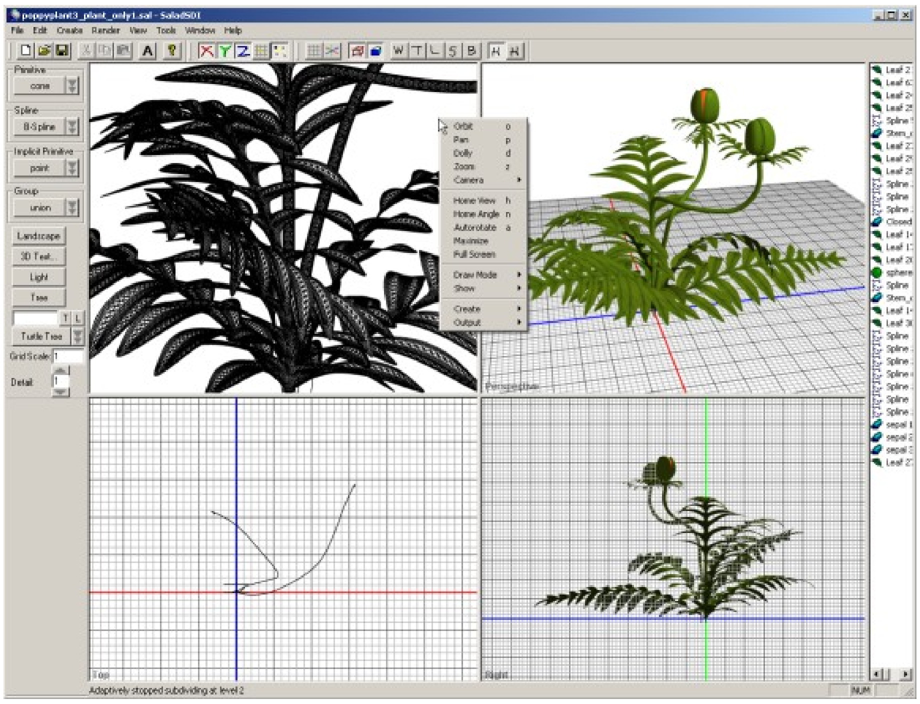
Appendix A

Software User Manual

A.1 Introduction

The methods presented in this thesis are implemented in a software environment called *Salad*. The name is derived from “Sweep Application”, as the environment was initially created as an editor for generalized cylinders and other swept surfaces. Salad’s graphical user interface is inspired by various commercial 3D modeling packages. It provides an interactive graphical environment for visualizing and editing a variety of interactively defined and procedurally generated objects. The objects can be composed into scenes for rendering with POV-Ray.

Salad includes a variety of features that are unrelated to the research described in this thesis; only features pertinent to this thesis are presented here. In Section A.2, the graphical user interface is described, including features for visualizing, editing, and composing scenes. Section A.3 contains a description of the interactive features specific to the methods presented in this thesis. Most of these features can also be accessed in batch mode via the command line, as described in Section A.4.



Adaptively stopped subdividing at level 2

Figure A.1: Salad Graphical User Interface

A.2 Salad GUI

Interface overview

At the top of the screen is the *main menu*, in which the vast majority of Salad's functionality may be found. Below this are some *toolbars*, which have popup tooltips to explain their usage.

Situated on the left side of the screen, below the toolbars, is a collection of *controls* for creating new objects and adjusting the displayed level of detail (such as the number of iterations of subdivision to perform). On the right side of the screen is an *object list*, which identifies all objects in the scene. By default, object names are comprised of their type name followed by a pseudorandom number. Between the controls on the left and the object list on the right are the four *viewports*.

All features are available via either the main menu or the controls. Additionally, right-clicking activates a *context menu*, through which all modeling features and visualization options are also available.

Viewports

Viewports are activated on mouse-over, so that the current viewport is the one which most recently had the mouse cursor in it. The current viewport's camera name (Front, Perspective, etc.) is drawn black; the others are drawn grey.

"Spacebar" expands the current viewport to fill the viewport area (between the control bars and the status bar). A full-screen mode is also available from the context menu and the `View` menu.

Each viewport can display a *grid*, which can be used to snap objects to world coordinates that are multiples of the grid scale. Display of the grid can be toggled via the toolbar or the `G` key. The size of the spaces between grid lines is set using the `Grid Scale` control.

Cameras

Each viewport is assigned a camera, which has a *position* in world space and a *target*, or focal point, that it is aimed at. The target is initially the origin, but can be set to the center of any object by choosing `Focus` from the object's context menu, or pressing the `F` key when the object is selected. A viewport's camera can be changed via the `Camera` submenu on the viewport's context menu. However, camera-to-viewport assignments are not saved.

Camera transformations can be initiated in two ways:

1. Maya's camera transformation commands

   Camera movement is executed by holding the `Alt` key while clicking and dragging with the appropriate mouse button:

   `Alt-Left:` **orbit** Rotate the camera about it's target, keeping it oriented towards the target.

   `Alt-Right:` **dolly** For the perspective camera, translate the camera towards and away from its target; for orthographic cameras, zoom in and out of the scene.

   `Alt-Middle:` **pan** Translate the camera's target and position in the plane of the screen.

   Additionally, if `Alt` is not held down, the middle mouse button initiates `orbit`.

2. Context menu:

   - Choose a transformation from the context menu in the appropriate viewport.
   - Move the mouse.
   - Click when done.

   The context menu offers one camera transformation unavailable using the Maya commands: *zoom*, which adjusts the field of view.

Objects

Individual objects can be selected in two ways: by clicking on them in the viewports, or by clicking on their names in the object list. A selected object's name may be edited by clicking on the name in the object list. Multiple objects may be selected at one time using *sweep selection*, a click-drag-release action which defines a rectangle in screen space. All objects intersecting the rectangle are selected.

Most objects have a *property sheet*, a floating window used to edit the object's parameters. The property sheet can be activated by double-clicking on the object, provided that it is the only selected object.

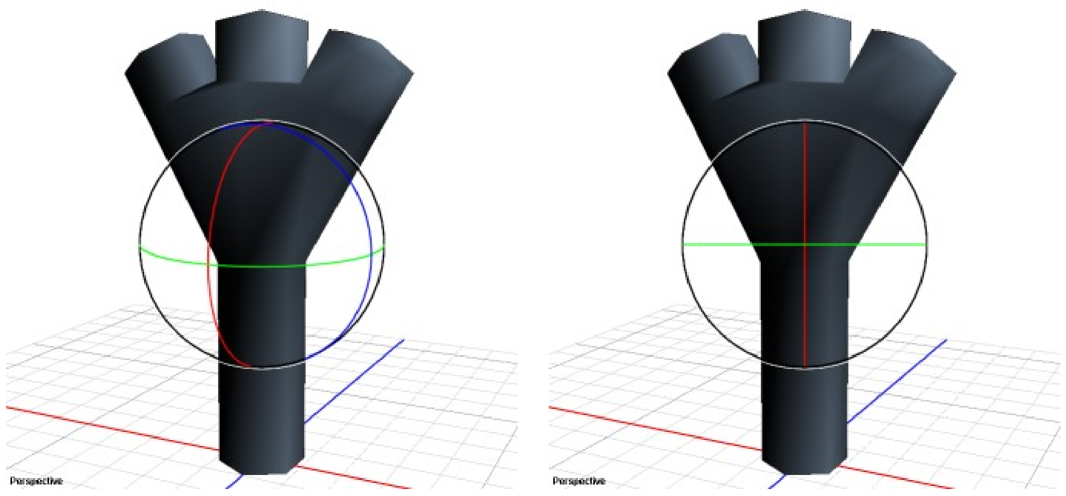
To translate or scale an object, first select the object, then hold the appropriate key while dragging the mouse:

`t`	translate selected objects
`s`	scale selected objects

Additionally, the `Transformations` editor, available from an object's context menu, allows direct numeric specification of the transformations. The `Reset` button sets all transformations to their defaults: zero translation and rotation, with unit scale.

Rotations are specified using quaternions. As these are unintuitive to specify numerically, object rotation is best accomplished using the Arcball rotation controller [Sho94]. The arcball is activated and deactivated with the `r` key. Clicking inside the arcball will initiate a 3D rotation; hold the mouse and move it to rotate the object. Dragging outside the arcball will rotate the object around an axis perpendicular to the viewport.

The arcball has the ability to constrain rotation to a specified axis. The axes



(a) Object-local constraints (b) Viewport constraints

Figure A.2: Arcball Rotation Controller

are drawn as coloured lines (Figure A.2). When the mouse cursor is over an axis, the axis will turn yellow, indicating that movement is constrained to that axis. By default, the object's local axes are displayed (Figure A.2(a)). Holding down `Ctrl` before clicking will display axes in the plane of the viewport (Figure A.2(b)).

Visualization modes

The lettered icons control the visualization options:

`W` Wireframe

`T` Texture

`L` Vertex lighting

`S` Smooth shading

`B` Bounding boxes. If bounding boxes are enabled, objects are hidden (except for their bounding boxes) during camera movements. This enables quick camera movement when working with complex scenes.

`H` Hidden-line mode

`H` **(shadowed)** Hidden-line mode, with hidden lines drawn stippled.

The first five options are toggles that can be combined arbitrarily. The last two modes are exclusive.

Files

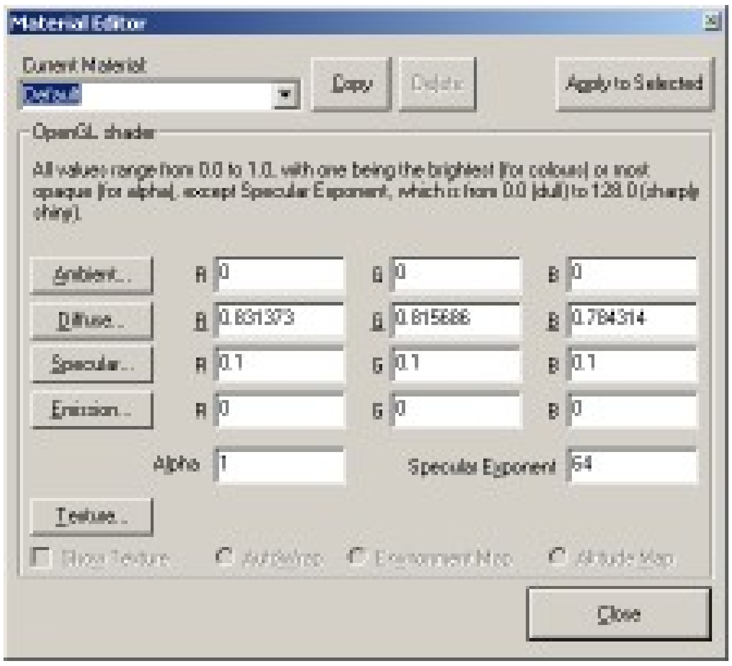
Salad provides its own native `.sal` scene file format, which can save most of Salad's features. However, viewport settings, which are not related to the scene itself, are saved in the Windows Registry, rather than with the scene file.

Material Editor

The Material Editor (Figure A.3) sets OpenGL rendering parameters. There are three ways to access the Material Editor:

1. From the main menu, choose `Edit|Material`.
2. Right-click on the object, and choose `Material`.
3. Double-click on the object, displaying its property sheet. Every property sheet has a `Material` button.

If the editor is activated via an object's context menu, that object's material will be made current in the editor.



Current Material: Default

All values range from 0.0 to 1.0, with one being the brightest (for colours) or most opaque (for alpha), except Specular Exponent, which is from 0.0 (dull) to 128.0 (sharply shiny).

	R	G	B
Ambient...	0	0	0
Diffuse...	0.831373	0.815686	0.784314
Specular...	0.1	0.1	0.1
Emission...	0	0	0

Alpha: 1

Specular Exponent: 64

Figure A.3: Material Editor

In the top-left corner of the dialog is the `Current Material` combo box, which lists the materials in the scene. Materials are created and edited here, and can be applied to any object. To create a new material, use the `Copy` button to create a new material with the same parameters as the current one. Use the `Delete` button to delete the current material. While this will remove the material from the list of available materials, a deleted material will only actually disappear from the scene when all objects that reference that material are assigned a different one.

The “Default” material has a diffuse colour taken from the system’s button face colour, a black ambient colour, a dark specular highlight, and no emission, transparency, or texture.

The `Ambient`, `Diffuse`, `Specular`, and `Emission` parameter buttons launch the standard Windows system colour selector dialog. The selected colour’s RGB values are normalized and placed in the edit boxes in the row of the corresponding parameter button. The RGB values for these parameters can also be edited directly.

The `Alpha` parameter corresponds to the `A` parameter to glColour* and is thus 1 for opaque, 0 for transparent, and something in between for transmissive. Note that Salad doesn't sort objects or triangles front-to-back, so transparent objects may be rendered incorrectly.

The `Specular Exponent` is an integer parameter in the range 0-128 (a range imposed by OpenGL).

Texture mapping

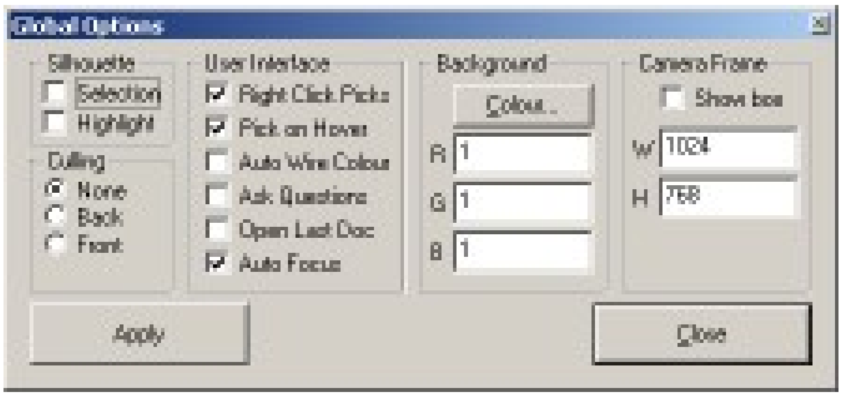
The `Texture...` button launches a standard file open dialog box. From there, one can choose a `BMP` or `PNG` image to use as a texture. Three texture mapping options are available:

AutoWrap Uses standard per-vertex texture mapping.

Environment Map Uses OpenGL 1.0 environment mapping.

Altitude Map 1D texture mapping for use with fractal terrain (not described in this work).

Texture images must be 24-bit RGB or 32-bit RGBA "True Colour" images in either `BMP` or `PNG` format. Palettes are unimplemented. The Alpha channel can be used in the 32-bit version of either file format to create transparent textures.


Global Options

Silhouette
- [ ] Selection
- [ ] Highlight

Culling
- (•) None
- ( ) Back
- ( ) Front

User Interface
- [✓] Right Click Picks
- [✓] Pick on Hover
- [ ] Auto Wire Colour
- [ ] Ask Questions
- [ ] Open Last Doc
- [✓] Auto Focus

Background
- Colour...
- R: 1
- G: 1
- B: 1

Camera Frame
- [ ] Show box
- W: 1024
- H: 768

Figure A.4: Global Options and their default settings

Lights

The `Light` button will place a point light source at the origin. Lights can be translated like any other object, and their colours can be set using the Material Editor. Lights are particularly useful when preparing scenes for export to POV-Ray.

Global Options

Figure A.4 shows the global options dialog, accessed via the `Tools` menu's `Options` command. The options are as follows:

Selection toggles display of a silhouette around selected objects. An object's silhouette is generated by drawing a scaled-up instance of the object, with the actual object masked out using the stencil buffer. As a result, displaying silhouettes can reduce performance.

Highlight toggles display of a silhouette around objects which are under the mouse cursor. The Highlight silhouette is a different colour from the Selection silhouette. This feature is only relevant if `Pick on Hover` is checked.

Culling controls OpenGL's polygon culling feature. Back-face culling is particularly useful when exporting closed meshes to PostScript.

Right Click Picks toggles whether right-clicking in a viewport, which activates the context menu, also performs a pick operation.

Pick on Hover toggles whether picking is performed after each mouse movement. This is a useful feature when choosing or editing splines, as it allows the user interface to graphically highlight objects under the cursor. However, for complex scenes, it can result in a lag between the initiation of a camera or object transformation command and the first result. For instance, the interface may appear to stall at the start of an `Orbit` operation.

Auto Wire Colour If this is checked, the colour for the wireframe visualization (excluding the `Hidden Line` modes) is computed from the complement of the object's diffuse colour. Otherwise, the wire is drawn black.

Ask Questions toggles whether Salad should warn the user if they are about to close a modified scene.

Open Last Doc toggles whether Salad should automatically open the last-used scene the next time Salad starts.

Auto Focus determines whether various operations (including importing objects) should include executing a `Focus` command in every viewport, centering the view on the newly created or modified object.

Background allows the user to choose the background colour for the viewports.

Camera Frame toggles the display of a rectangle of the specified screen-space dimensions, centered in each viewport. This is useful for composing a scene to be rendered in POV-Ray at a specific resolution. The default is NTSC DVD resolution.

All of these options are saved in the Windows registry key
`HKEY_CURRENT_USER\Software\Pete the Dud\Salad2K`.

A.3 GUI for turtle features

This section explains how to use the implementation of the methods proposed in this work; that is, generating subdivision surfaces for plant models. There are four steps:

1. Export a string from a CPFG model.
2. Import the string into the system.
3. Process the mesh.
4. Export to POV-Ray for rendering.

For step 1, users should consult the CPFG manual [Měc98].

Importing strings

The user must decide whether they are modeling a tree or a leaf. Trees are modeled using a closed cross section, whereas leaves are modeled with an open cross section, with an optional crease representing the midrib.

To generate a tree model, choose `Turtle Tree` from the drop-down menu next to the `Turtle` button, situated in the controls on the left of the interface. This will bring up an `Import File` dialog, letting you browse to the `.str` file produced by CPFG. The selected strings will be interpreted as trees, and the resulting meshes will be displayed. For Leaf models, follow the same process, but choose `Turtle Leaf` from the menu.

Processing meshes

This subsection describes how to set mesh options such as edge sharpness, material, etc., and subdivide.

Double-click a mesh to launch its properties dialog. The dialog contains three pages (Figure A.5), accessed via the tabs just below the title bar.

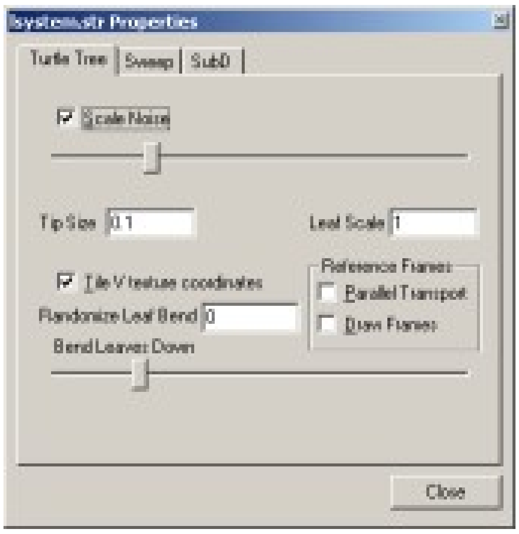
Turtle Tree tab

For tree models, the first tab is called `Turtle Tree` and controls parameters of the tree interpretation, as well as leaf orientation.

Scale Noise on this tab merely determines whether the noise is uniformly distributed throughout the model (slider full right), or scaled in proportion to branch "age" (slider full left). It is presented here rather than with the noise controls on the SubD tab as it is specific to tree models.

Tip Size determines the height of the cone attached to the end of each terminal branch, relative to the width of the terminal branch.

Tile V texture coordinates toggles between the two available texture mapping



lsystem.str Properties

Turtle Tree | Sweep | SubD

[✓] Scale Noise

Tip Size: 0.1

Leaf Scale: 1

[✓] Tile V texture coordinates

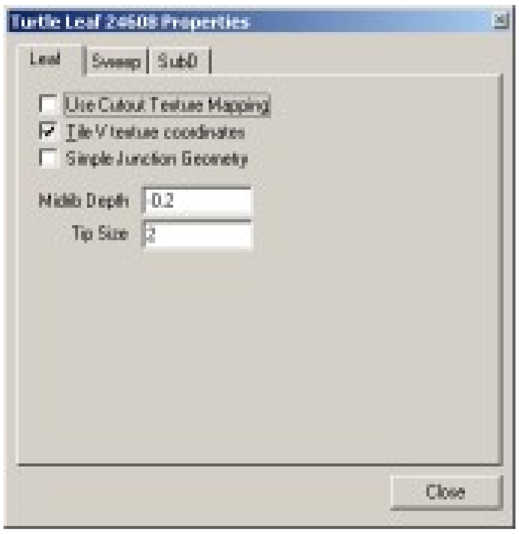
Randomize Leaf Bend: 0

Bend Leaves Down

Reference Frames: [ ] Parallel Transport, [ ] Draw Frames

Close

(a) Turtle Tree tab



Turtle Leaf 24608 Properties

Leaf | Sweep | SubD

[ ] Use Cutout Texture Mapping

[✓] Tile V texture coordinates

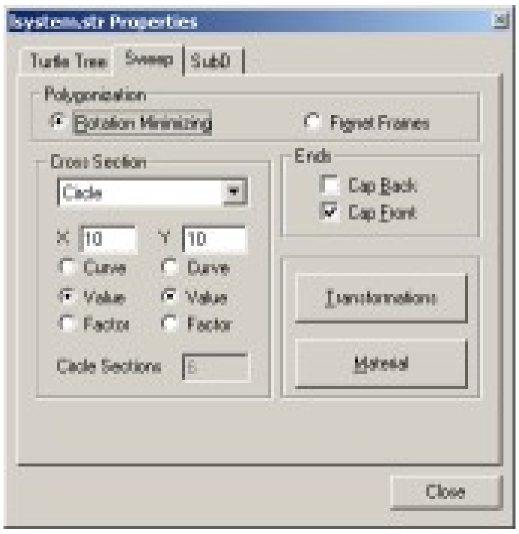
[ ] Simple Junction Geometry

Midrib Depth: -0.2

Tip Size: 2

Close

(b) Leaf tab



lsystem.str Properties

Turtle Tree | Sweep | SubD

Polygonization: (•) Rotation Minimizing, ( ) Frenet Frames

Cross Section: Circle

X: 10, ( ) Curve, (•) Value, ( ) Factor

Y: 10, ( ) Curve, (•) Value, ( ) Factor

Circle Sections: 6

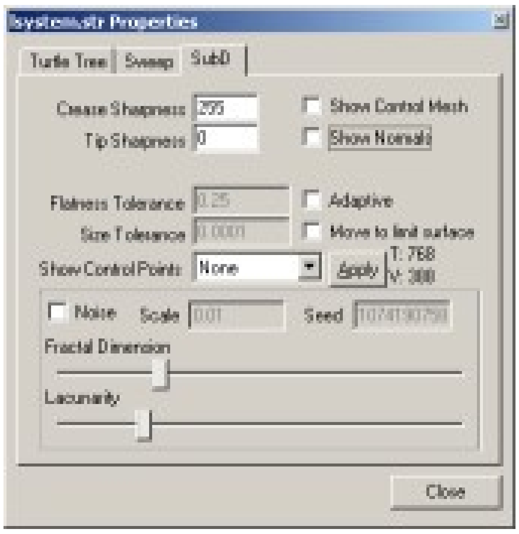
Ends: [ ] Cap Back, [✓] Cap Front

Transformations

Material

Close

(c) Sweep tab



lsystem.str Properties

Turtle Tree | Sweep | SubD

Crease Sharpness: 255

Tip Sharpness: 0

[ ] Show Control Mesh

[ ] Show Normals

Flatness Tolerance: 0.25

Size Tolerance: 0.0001

[ ] Adaptive

[ ] Move to limit surface

Show Control Points: None

Apply

T: 768

V: 388

[ ] Noise, Scale: 0.01, Seed: 1074190758

Fractal Dimension

Lacunarity

Close

(d) SubD tab

Figure A.5: Properties Dialog: A given turtle-generated object will be either a tree or a leaf; only the appropriate tab will be shown.

modes. Tiling maps the entire V domain to each internode; not tiling means the V domain is mapped to the longest path in the tree.

Randomize Leaf Bend determines the magnitude by which each leaf's actual angle is randomly perturbed.

Bend Leaves Down affects the base angle from which leaves subtend the tree. The default is horizontally. Leaves are added to trees as described in Section A.3.

Parallel Transport allows the user to choose whether to use parallel transport frames instead of the default of directly using the frames specified by the turtle string.

Draw Frames toggles display of local-coordinate axes at each reference frame. This is useful for visualizing the effect of parallel transport frames.

Leaf tab

This tab (Figure A.5(b)) controls parameters of the leaf interpretation.

Use Cutout Texture Mapping toggles between *cutout* and *tiled* texture mappings for leaves. Cutout mapping assumes the texture represents the entire leaf surface, whereas tiling maps the texture to an internode, as with the tiling option for trees. The cutout mode is described in Section A.3.

Tile V texture coordinates toggles between the two available tiled texture mapping modes. Checking this option maps the entire V domain to each internode; otherwise, the V domain is mapped to the longest path in the leaf.

Simple Junction Geometry toggles between two templates for leaf branching junctions. The `Simple` template creates fewer triangles and a larger blend area than the template described in Section 5.4.

Midrib Depth controls the size of the midrib, relative to the width of each cross section.

Tip Size determines the height of the cone attached to the end of each terminal branch, relative to the width of the terminal branch.

Sweep tab

Most of this tab (Figure A.5(c)) is used for a generalized cylinder implementation which is not described in this documentation. Several features are relevant to turtle string models, however:

Material launches a dialog for manipulating objects' material properties (Section A.2).

Cap Front: If checked (the default), this places round tips on the terminal branches, which can be sharpened using the SubD tab, Section A.3. If unchecked, flat caps with hard edges are applied to the terminal branches, for a pruned effect.

Cap Back: If checked, places a flat cap with hard edges at the base of the model. If unchecked (the default), the base is open.

SubD tab

This tab (Figure A.5(d)) controls the coarse mesh and subdivision algorithm, and is common to leaves, trees, and generalized cylinders.

Crease Sharpness indicates the number of subdivision iterations during which "sharp" crease rules will be applied to edges that are marked as creases. Setting this to 0 will ignore the creases, whereas setting this to 255 (the maximum number of subdivision iterations allowed) will result in sharp creases.

Tip Sharpness is equivalent to "Crease Sharpness", but for vertices marked as points. This only affects the tips of leaf meshes.

Flatness Tolerance controls the adaptive subdivision algorithm's tolerance for curvature. It is based on the dihedral angle, in radians, between adjacent triangles. A lower tolerance will result in more subdivision of curved regions of a mesh. This also affects subdivision of triangles on boundaries or creases, where the parameter is based on the collinearity of two edges.

Size Tolerance controls the minimum allowable world-space area for a triangle. This prevents subdivision of sufficiently small triangles.

Show Control Points controls the display of the vertices of the coarse mesh. Only relevant if "Show Control Mesh" is checked. Set this to `Point` for interactive editing of a mesh (only available if the mesh is "`Extract`ed" via the context menu). The default is set to `None` for fastest visualization, and the `Index` and `Tex Coord` options are for debugging.

Show Control Mesh: Toggles display of the coarse mesh when the object is selected.

Show Normals: If checked, vertex normals are displayed in orange and face normals in blue. Note that for tree meshes, the true normals point inwards, but they are shown facing outwards to make them visible.

Adaptive toggles adaptive subdivision.

Move to limit surface toggles a post-subdivision step which uses Hoppe's limit masks to move vertices to their limit positions. As the changes are indeed applied to the vertices stored in the mesh, only use this after choosing the final level of subdivision to render.

Apply is for the adaptive subdivision tolerances and Noise; it initiates complete resubdivision.

Subdividing

The `Detail` edit control on the `Settings Bar` determines the level of subdivision. Level 0 refers to the coarse mesh. For adaptive subdivision, use level 1. (Adaptive subdivision must first be enabled, as described above.) The mesh is always stored at the highest level of subdivision that's been applied to it. Therefore, if the user subdivides to level four, then reduces the detail to level three, only the third-level vertices will be displayed, but they will be in their fourth-level positions. However, whenever Level 0 is selected, the mesh is reset to the control vertices, so that Level 0 always corresponds to the original coarse mesh.

Leaf Texturing

As described in Section 6.4.1, one method of texture mapping leaves involves using a *proxy mesh* to define a mapping from the 2D texture image to the 3D model. To use this feature effectively, models should be created such that the topology is entirely represented by rotations about about the *Up* axis, with rotations about *Left* and

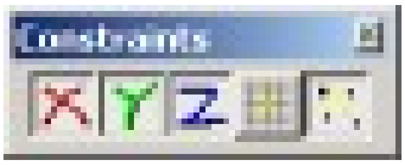


Figure A.6: Constraints Toolbar, with *Y* constrained

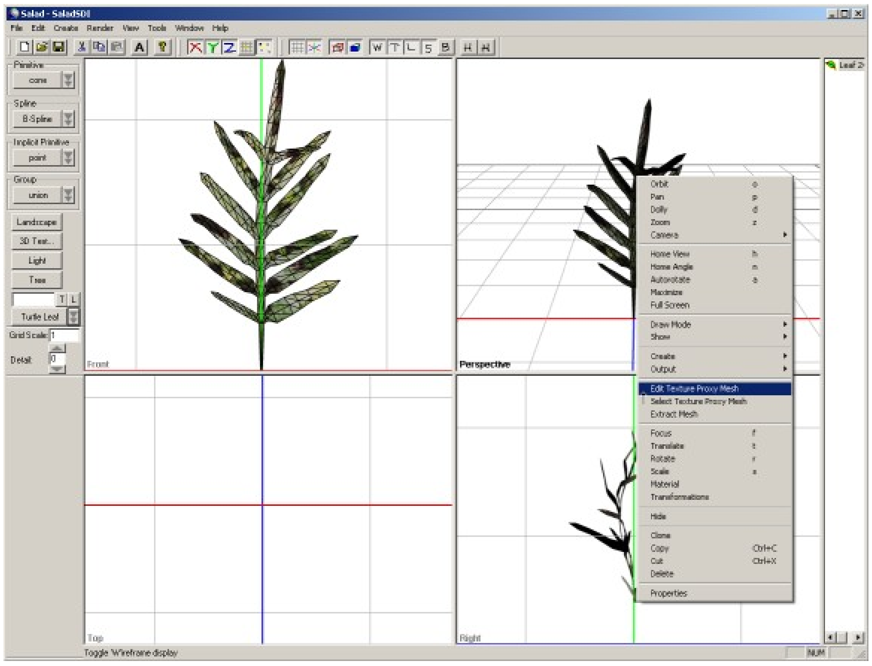


Figure A.7: Context Menu, with “Edit Texture Proxy Mesh” highlighted

Heading used only to bend and twist the leaf into a 3D shape. The proxy mesh is edited as follows:

1. Import the turtle string as a leaf.

2. Double click the mesh. Check `Cutout Texture Mapping` on the Leaf tab.

3. Select the Sweep Tab, and click Material. Assign a texture using the Texture button (Section A.2).

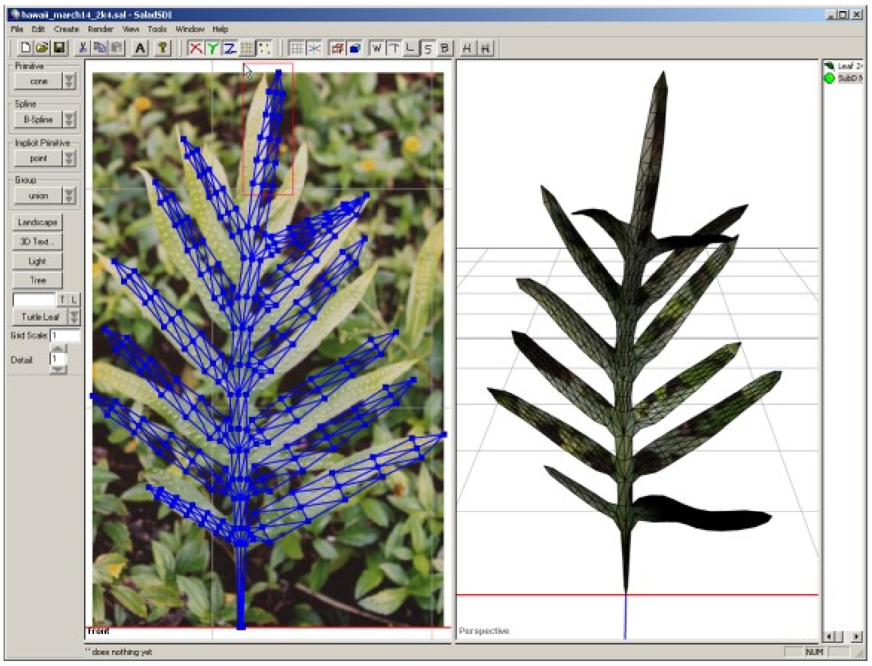


Figure A.8: Sweep Selection while editing the proxy mesh

4. Right click on the resulting mesh and choose Edit Texture Proxy Mesh (Figure A.7). The proxy mesh will be selected and displayed in the Front view, with the texture slightly behind it.

5. Consider using sweep selection (click and drag with the left mouse button) (Figure A.8) and the Proportional Movement tool (the rightmost tool in Figure A.6) to allow editing of an entire lobe at once. The selection can be appended to by `Shift`-dragging, and toggled using `Ctrl`-click.

   Note that any subdivision surface can be edited in this way, provided that it is made available using its generator's context menu's `Extract Mesh` command.

Leaves on Trees

Salad includes the ability to attach leaves to the ends of tree branches.

1. (optional) Create a material called *"Bark"* and a material called *"Leaf"* (case insensitive). These will be automatically applied in the next two steps.
2. Import a `Turtle Tree`.
3. While the tree is selected (which it is immediately after importing), import a `Turtle Leaf`. An instance of the leaf will then be attached to all the branch tips.

Leaf scale and orientation can be controlled using the Tree Tab (Section A.3). To edit the leaves' properties, such as creases, right-click on the tree and choose `Leaf Properties`.

Export to POV-Ray

`File|Export|POV-Ray...` will generate a POV-Ray scene (`*.pov`) corresponding roughly to the `Perspective` viewport's current contents. Each mesh will be placed in its own include file (`*.inc`), whose name is generated from the corresponding object's name in Salad (as shown in the object list). If you have not set up a light (Section A.2), Salad will create a sky sphere in the scene's background colour, and use it for radiosity-based illumination. You can launch POV-Ray directly from Salad using `Alt-G`; this will generate a POV-Ray scene in the scene's folder, or the current working folder if the scene is unsaved, then launch POV-Ray.

A.4 Command Line usage

Salad, when built with the `ReleaseConsole`, `ReleaseDebug`, or `Debug` configuration, includes a command line interface to the tree surface generation code.

Salad takes as input string (".str") files, which are text files containing turtle commands. Multiple strings may be specified in one call, and wildcards may be used to specify groups of files (i.e., `*.str`)

Several switches are available to control the surface generated. Switches may be prefixed with '/', '-', or '+'. Any numeric parameters to the switches must immediately follow the switch, without whitespace in between.

`/A<size>` Apex size. The floating-point parameter indicates the proportion of the tip length to its base width.

`/C` Consistent name. The exported objects are all given the same name. File names will still be unique. Useful for animation.

`/E<iter>` Edge sharpness. The integer parameter must be in the range [0,255], and represents the number of "sharp" subdivision iterations. For leaves (ignored without /L).

`/?,/H` Help. Print this text.

`/L` Leaf. Generate an open surface, for leaf modeling.

`/N[scale]` Noise. Fractal noise is generated by perturbing vertex positions after each subdivision step. The optional floating-point parameter determines the magnitude of the perturbations.

`/O` OBJ. Output OBJ format (*.obj). If neither /O nor /P are specified, this is the default format.

`/P` POV-Ray. Output POV-Ray format (*.inc). Both /O and /P may be specified to get both formats simultaneously.

`/S[iter]` Subdivide. If no parameter is given, adaptive subdivision is is used. The optional integer parameter specifies the number of steps of static subdivision to perform instead.

`/T<size>` Tip sharpness. The integer parameter must be in the range [0,255], and represents the number of “sharp” subdivision iterations.

`/V` Vertex limit positions: vertices are moved to their limit positions after subdivision is completed.

Bibliography

[ACS03] E. Akleman, J. Chenb, and V. Srinivasan. A minimal and complete set of operators for the development of robust manifold mesh modelers. *Graphical Models*, 65(5):286–304, September 2003.

[Ad80] H. Abelson and A. A. diSessa. *Turtle Geometry: The Computer as a Medium for Exploring Mathematics*. MIT Press, Cambridge, MA, 1980.

[AFR02] A. Amresh, G. Farin, and A. Razdan. Adaptive subdivision schemes for triangular meshes. In G. Farin, H. Hagen, and B. Hamann, editors, *Hierarchical and Geometric Methods in Scientific Visualization*, pages 319–327. Springer-Verlag, 2002.

[AG01] S. Akkouche and E. Galin. Adaptive implicit surface polygonization using marching triangles. *Computer Graphics Forum*, 20(2):67–80, 2001.

[AK84] M. Aono and T. L. Kunii. Botanical tree image generation. *IEEE Computer Graphics & Applications*, 4(5):10–34, May 1984.

[AMZ99] A. S. Aguado, E. M., and E. Zaluska. Modeling generalized cylinders via fourier morphing. *ACM Transactions on Graphics (TOG)*, 18(4):293–315, October 1999.

[AP03] M. Aitken and M. Preston. Foliage generation and animation for "The Lord of the Rings: The Two Towers", 2003. SIGGRAPH 2003 DVD-ROM, ACM SIGGRAPH, New York.

[Bar88] B. A. Barsky. *Computer Graphics and Geometric Modeling Using Beta-splines.* Springer-Verlag, 1988.

[Bel91] A. D. Bell. *Plant Forum: An Illustrated Guide to Flowering Plant Morphology.* Oxford University Press, Oxford, 1991.

[Bis75] R. L. Bishop. There is more than one way to frame a curve. *American Mathematical Monthly*, 82(3):246–251, March 1975.

[BKRS86] E. A. Bier and Jr. K. R. Sloan. Two-part texture mappings. *IEEE Computer Graphics and Applications*, 6(9):40–53, September 1986.

[Bli77] J. F. Blinn. *Computer Display of Curved Surfaces.* PhD thesis, Dep. Comptr. Sci., U. of Utah, Salt Lake City, Fall 1977.

[Blo85] J. Bloomenthal. Modeling the Mighty Maple. In *Proceedings of SIGGRAPH 1985*, volume 19, pages 305–311. ACM Press, July 1985.

[Blo90] J. Bloomenthal. Calculation of reference frames along a space curve. In *Graphics Gems*, pages 567–571. Academic Press, Boston, 1990.

[Blo95a] J. Bloomenthal. Bulge elimination in implicit surface blends. In *Implicit Surfaces '95*, April 1995.

[Blo95b] J. Bloomenthal. *Skeletal Design of Natural Forms.* Ph.D. dissertation, University of Calgary, 1995.

[Blo97] J. Bloomenthal, editor. *Introduction to Implicit Surfaces.* Morgan Kaufmann, 1997.

[BLZ00] H. Biermann, A. Levin, and D. Zorin. Piecewise smooth subdivision surfaces with normal control. In *Proceedings of SIGGRAPH 2000*, pages 113–120. ACM Press, 2000.

[BS91] J. Bloomenthal and K. Shoemake. Convolution surfaces. In *Proceedings of SIGGRAPH 1991*, pages 251–256. ACM Press, 1991.

[Cat74] E. E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, Dept. of CS, U. of Utah, December 1974.

[CC78] E. Catmull and J. Clark. Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer Aided Design*, 10(6):350–355, 1978.

[Col65] R. V. Cole. *The Artistic Anatomy of Trees: Their Structure & Treatment in Painting*. Dover, New York, second edition, 1965.

[Coo84] R. L. Cook. Shade trees. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 223–231. ACM Press, 1984.

[DKT98] T. DeRose, M. Kass, and T. Truong. Subdivision surfaces in character animation. In *Proceedings of SIGGRAPH 1998*, pages 85–94. ACM Press, 1998.

[DL97] O. Deussen and B. Lintermann. A modelling method and interface for creating plants. In *Graphics Interface '97*, May 1997.

[DLG90] N. Dyn, D. Levine, and J. A. Gregory. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Transactions on Graphics (TOG)*, 9(2):160–169, 1990.

[dREF$^+$88] P. de Reffye, C. Edelin, J. Françon, M. Jaeger, and C. Puech. Plant models faithful to botanical structure and development. In *Proceedings of SIGGRAPH 1988*, pages 151–158. ACM Press, 1988.

[DS78] D. Doo and M. Sabin. Analysis of the behaviour of recursive division surfaces near extraor-dinary points. *Computer Aided Design*, 10(6):356–360, 1978.

[EMP$^+$94] D. Ebert, K. Musgrave, D. Peachey, K. Perlin, and S. Worley. *Texturing and Modeling: A Procedural Approach.* Academic Press, San Diego, CA, 1994.

[Far02] G. Farin. *Curves and Surfaces for CAGD, A Practical Guide.* Morgan Kaufmann, San Francisco, fifth edition, 2002.

[Fed02] P. Federl. *Modeling Fracture Formation on Growing Surfaces.* PhD thesis, University of Calgary, September 2002.

[FFC82] A. Fournier, D. Fussell, and L. Carpenter. Computer Rendering of Stochastic Models. *Communications of the ACM*, 25(6):356–362, June 1982.

[FFKW02] P. Felkel, A. L. Fuhrmann, A. Kanitsar, and R. Wegenkittl. Surface reconstruction of the branching vessels for augmented reality aided surgery. In *Biosignal 2002*, 2002.

[FFW01] P. Felkel, A. L. Fuhrmann, and R. Wegenkittl. Smart - surface models from by-axis-and-radius-defined tubes. Technical Report TR VRVis 2001 026, VRVis Zentrum für Virtual Reality und Visualisierung Forschungs-GmbH, http://www.vrvis.at/TR/2001/TR_VRVis_2001_026_Full.pdf, 2001.

[FGW00] M. Fox, C. Galbraith, and B. Wyvill. Efficient implementation of the blobtree for rendering purposes. In *Proceedings of the Eleventh Western Computer Graphics Symposium*. University of Alberta, April 2000.

[Fir91] P. A. Firby. *Surface topology*. Ellis Horwood, New York, second edition, 1991.

[FL74] D. Frijters and A. Lindenmayer. A model for the growth and flowering of Aster novae-angliae on the basis of table (1,0)L-systems. In G. Rozenberg and A. Salomaa, editors, *L-systems, Lecture Notes in Computer Science 15*, pages 24–52. Springer-Verlag, Berlin, 1974.

[FL76] D. Frijters and A. Lindenmayer. Developmental descriptions of branching patterns with paracladial relationships. In A. Lindenmayer and G. Rozenberg, editors, *Automata, languages, development*, pages 57–73. North-Holland, Amsterdam, 1976.

[FvD82] J. D. Foley and A. van Dam. *Fundamentals of interactive computer graphics*. Addison-Wesley, Reading, Massachusetts, 1982.

[Han92] J. S. Hanan. *Parametric L-systems and Their Application to the Modelling and Visualization of Plants.* PhD thesis, University of Regina, June 1992.

[HB96] J. C. Hart and B. Baker. Implicit modeling of tree surfaces. In *Proceedings of Implicit Surfaces '96*, pages 143–152, October 1996.

[HDD$^+$94] H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, and W. Stuetzle. Piecewise smooth surface reconstruction. In *Proceedings of SIGGRAPH 1994*, pages 295–302. ACM Press, July 1994.

[Hol94] M. Holton. Strands, gravity, and botanical tree imagery. *Computer Graphics Forum*, 13(1):57–67, 1994.

[Hon71] 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 on the shape of the tree-like body. *Journal of Theoretical Biology*, 31:331–338, 1971.

[HPW92] H. Hammel., P. Prusinkiewicz, and B. Wyvill. Modelling compound leaves using implicit contours. In *Proceedings of Computer Graphics International 1992*, pages 199–212. Springer-Verlag, 1992.

[HSIW] A. Hilton, A. J. Stoddart, J. Illingworth, and T. Windeatt. Marching triangles: Range image fusion for complex object modelling. In *IEEE 1996 International Conference on Image Processing.*

[JTFP01] X. Jin, C.-L. Tai, J. Feng, and Q. Peng. Convolution surfaces for line skeletons with polynomial weight distributions. *Journal of Graphics Tools*, 6(3):17–28, 2001.

[JW88] D. Jevans and B. Wyvill. Ray Tracing Implicit Surfaces. Research Report 88/292/04, University of Calgary, Dept. of Computer Science, 1988.

[Leo91] M. K. De Leon. Branching object generation and animation system with cubic Hermite interpolation. *Journal of Visualization and Computer Animation*, 2(2):60–67, 1991.

[Lin68] A. Lindenmayer. Mathematical models for cellular interaction in development, Parts I and II. *Journal of Theoretical Biology*, 18:280–315, 1968.

[Lin71] A. Lindenmayer. Developmental systems without cellular interaction, their languages and grammars. *Journal of Theoretical Biology*, 30:455–484, 1971.

[LLX$^+$01] L. Liang, C. Liu, Y.-Q. Xu, B. Guo, and H.-Y. Shum. Real-time texture synthesis by patch-based sampling. *ACM Transactions on Graphics (TOG)*, 20(3):127–150, 2001.

[LN02] S. Lefebvre and F. Neyret. Synthesizing bark. In *Proceedings of the 13th Eurographics workshop on Rendering*, pages 105–116. Eurographics Association, 2002.

[Loo87] C. Loop. Smooth subdivision surfaces based on triangles. Master's thesis, University of Utah, 1987.

[Loo01] C. Loop. Triangle mesh subdivision with bounded curvature and the convex hull property. MSR Tech Report MSR-TR-2001-24, Microsoft Research Graphics Group. Microsoft Corporation, http://research.microsoft.com/˜cloop/msrtr2001-24.pdf, 2001.

[Mai02] S. Maierhofer. *Rule-Based Mesh Growing and Generalized Subdivision Meshes*. PhD thesis, Technischen Universität Wien, January 2002.

[Man94] B. B. Mandelbrot. *Fractals in biology and medicine*, chapter A Fractal's Lacunarity, and how it can be Tuned and Measured. Birkhäuser Boston, 1994.

[Mar03] K. Maritaud. *Rendu réaliste d'arbres vus de près en images de synthèse*. PhD thesis, University de Limoges, France, December 2003.

[Měc97] R. Měch. *Modeling and simulation of the interaction of plants with the environment using L-systems and their extensions*. PhD thesis, University of Calgary, 1997.

[Měc98] R. Měch. CPFG version 3.4 user's manual, 1998. Manuscript, Department of Computer Science, University of Calgary.

[Mil86] G. S. P. Miller. The Definition and Rendering of Terrain Maps. *Computer Graphics (Proceedings of SIGGRAPH 86)*, 20(4):39–48, August 1986.

[MKM89] F. K. Musgrave, C. E. Kolb, and R. S. Mace. The synthesis and rendering of eroded fractal terrains. *Computer Graphics (Proceedings of SIGGRAPH 89)*, 23(3):41–50, July 1989. Held in Boston, Massachusetts.

[MLP01] K.-H. Min, I.-K. Lee, and C.-M. Park. Component-based Polygonal Approximation of Soft Objects. *Computers & Graphics*, 25:245–257, 2001.

[MMPP03] L. Mündermann, P. MacMurchy, J. Pivovarov, and P. Prusinkiewicz. Modeling lobed leaves. In *2003 Computer Graphics International (CGI 2003)*, pages 60–67. IEEE Computer Society, July 2003.

[NC99] F. Neyret and M.-P. Cani. Pattern-based texturing revisited. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 235–242. ACM Press/Addison-Wesley Publishing Co., 1999.

[Opp86] P. E. Oppenheimer. Real time design and animation of fractal plants and trees. In *Proceedings of SIGGRAPH 1986*, volume 20, pages 55–64, August 1986.

[Pea85] D. R. Peachey. Solid texturing of complex surfaces. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 279–286, July 1985.

[Per85] K. Perlin. An Image Synthesizer. *Computer Graphics (Proceedings of SIGGRAPH 85)*, 19(3):287–296, July 1985. Held in San Francisco, California.

[PHHM96] P. Prusinkiewicz, M. Hammel, J. Hanan, and R. Měch. *Handbook of Formal Languages*, chapter Visual models of plant development. Springer–Verlag, Berlin, 1996.

[PJM94] P. Prusinkiewicz, M. James, and R. Měch. Synthetic topiary. In *Proceedings of SIGGRAPH 1994*, pages 351–358, Orlando, Florida, July 1994. ACM Press.

[PL90] P. Prusinkiewicz and A. Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag, New York, 1990.

[PLH88] P. Prusinkiewicz, A. Lindenmayer, and J. Hanan. Developmental models of herbaceous plants for computer imagery purposes. In *Proceedings of SIGGRAPH 1988*, pages 141–150. ACM, 1988.

[PMKL01] P. Prusinkiewicz, L. Mündermann, R. Karwowski, and B. Lane. The use of positional information in the modeling of plants. In *Proceedings of SIGGRAPH 2001*, pages 289–300. ACM Press, August 2001.

[pov] POV-Ray - The Persistence of Vision Raytracer. Web page. http://www.povray.org/.

[Pru86] P. Prusinkiewicz. Graphical applications of L–systems. In *Graphics Interface '86*, pages 247–253, May 1986.

[Rei95] U. Reif. A unified approach to subdivision algorithms near extraordinary points. *Computer Aided Geometric Design*, 12(2):153–174, 1995.

[Sab01] M. Sabin. Subdivision: Tutorial notes. SMI2001 Course Notes, Genoa, 2001.

[Sho94] K. Shoemake. Arcball rotation control. *Graphics Gems IV*, pages 175–192, 1994. ISBN 0-12-336155-9. Held in Boston.

[Smi84] A. R. Smith. Plants, Fractals and Formal Languages. In *Proceedings of SIGGRAPH 1984*, pages 1–10, New York, 1984. ACM Press.

[SP03] M.C. Sousa and P. Prusinkiewicz. A few good lines: Suggestive drawing of 3d models. *Computer Graphics Forum (Proc. of EuroGraphics '03)*, 22(3), 2003.

[Sta97] J. Stam. Aperiodic texture mapping. Research Report R046, ERCIM, January 1997.

[THB02] J. Taylor-Hell and G. Baranoski. State of the art in the realistic simulation of plant leaf venation systems. Technical Report CS-2002-17, University of Waterloo, April 2002.

[Tig99] M. Tigges. Two dimensional texture mapping of implicit surfaces. Master's thesis, University of Calgary, June 1999.

[TMW02a] R. F. Tobler, S. Maierhofer, and A. Wilkie. A Multiresolution Mesh Generation Approach for Procedural Definition of Complex Geometry. In *Proceedings of the International Conference on Shape Modeling and Applications (SMI 2002)*, pages 35–42. IEEE Computer Society, May 2002.

[TMW02b] R. F. Tobler, S. Maierhofer, and A. Wilkie. Mesh-Based Parametrized L-Systems And Generalized Subdivision for Generating Complex Geometry. *International Journal of Shape Modeling*, 8(2):173–191, December 2002.

[Tur52] A. Turing. The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society of London*, (237), 1952.

[Tur91] G. Turk. Generating textures on arbitrary surfaces using reaction-diffusion. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 289–298. ACM Press, 1991.

[Tur01] G. Turk. Texture synthesis on surfaces. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 347–354. ACM Press, 2001.

[TW99] M. Tigges and B. Wyvill. A field interpolated texture mapping algorithm for skeletal implicit surfaces. In *Proceedings of Computer Graphics International 1999*, pages 25–33. IEEE Computer Society, June 1999.

[VdFG99] L. Velho, L. H. de Figueiredo, and J. Gomes. A unified approach for hierarchical adaptive tesselation of surfaces. *ACM Transactions on Graphics*, 18(4):329–366, October 1999.

[VEJA89] X. G. Viennot, G. Eyrolles, N. Janey, and D. Arques. Combinatorial Analysis of Ramified Patterns and Computer Imagery of Trees. In *Proceedings of SIGGRAPH 1989*, pages 31–40. ACM Press, 1989.

[WD97] F. M. Weinhaus and V. Devarajan. Texture mapping 3d models of real-world scenes. *ACM Computing Surveys (CSUR)*, 29(4):325–365, 1997.

[WGG99] B. Wyvill, E. Galin, and A. Guy. Extending The CSG Tree. Warping, Blending and Boolean Operations in an Implicit Surface Modeling System. *Computer Graphics Forum*, 18(2):149–158, June 1999.

[WH94] A. P. Witkin and P. S. Heckbert. Using particles to sample and control implicit surfaces. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 269–277. ACM Press, 1994.

[WJvOW00] B. Wyvill, P. Jepp, K. van Overveld, and G. Wyvill. Subdivision surfaces for fast approximate implicit polygonization. Research Report 2000-671-23, University of Calgary, 2000.

[WK91] A. Witkin and M. Kass. Reaction-diffusion textures. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 299–308. ACM Press, 1991.

[WL01] L.-Y. Wei and M. Levoy. Texture synthesis over arbitrary manifold surfaces. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 355–360. ACM Press, 2001.

[WMW86] G. Wyvill, C. McPheeters, and B. Wyvill. Data structure for soft objects. *The Visual Computer*, 2(4):227–234, February 1986.

[WW02] J. Warren and H. Weimer. *Subdivision Methods for Geometric Design: A Constructive Approach.* Morgan Kaufmann, 2002.

[WWT$^+$03] L. Wang, X. Wang, X. Tong, S. Lin, S. Hu, B. Guo, and H.-Y. Shum. View-dependent displacement mapping. *ACM Transactions on Graphics*, 22(3):334–339, 2003.

[YZ01] L. Ying and D. Zorin. Nonmanifold subdivision. In *IEEE Visualization 2001*, pages 325–331, October 2001.

[Zor98] D. Zorin. *Stationary Subdivision and Multiresolution Surface Representations.* PhD thesis, California Institute of Technologoy, 1998.

[Zor00a] D. Zorin. A method for analysis of c1-continuity of subdivision surfaces. *SIAM Journal of Numerical Analysis*, 37(5):1677–1708, 2000.

[Zor00b] D. Zorin. Smoothness of subdivision on irregular meshes. *Constructive Approximation*, 16(3):359–397, 2000.

[ZS00] D. Zorin and P. Schröder. Subdivision for Modeling and Animation. In *SIGGRAPH 2000 Course Notes.* ACM SIGGRAPH, 2000.

[ZSS97] D. Zorin, P. Schröder, and W. Sweldens. Interactive multiresolution mesh editing. In *Proceedings of SIGGRAPH 1997*, pages 259–268, August 1997.

[ZZV$^+$03] J. Zhang, K. Zhou, L. Velho, B. Guo, and H.-Y. Shum. Synthesis of progressively-variant textures on arbitrary surfaces. *ACM Transactions on Graphics*, 22(3):295–302, 2003.