Report Number: WUCSE-2005-16

2005-05-05

# Non-Photorealistic Rendering of Algorithmically Generated Trees

Nathan C. Dudley and Cindy Grimm

This work presents a novel rendering technique inspired by artistic approaches. Instead of trying to recreate the appearance of a traditional medium, such as charcoal or watercolor, this approach is a mixture of both photo-realism and abstraction. Artists use a process of abstraction to provide structural information about subjects that do not have clearly defined shapes, such as groups of leaves in a tree. For example, an artist will first use a color wash to approximate a group of leaves, then add detail on top of parts of this wash to indicate individual leaves. Similarly, we use an abstract... Read complete abstract on page 2.

# Non-Photorealistic Rendering of Algorithmically Generated Trees

Nathan C. Dudley and Cindy Grimm

Complete Abstract:

This work presents a novel rendering technique inspired by artistic approaches. Instead of trying to recreate the appearance of a traditional medium, such as charcoal or watercolor, this approach is a mixture of both photo-realism and abstraction. Artists use a process of abstraction to provide structural information about subjects that do not have clearly defined shapes, such as groups of leaves in a tree. For example, an artist will first use a color wash to approximate a group of leaves, then add detail on top of parts of this wash to indicate individual leaves. Similarly, we use an abstract shape that approximates the image of leaves clustered at the end of a branch. To prevent oversimplification, we add photo-realistic detail using a blending process. Inter-frame coherence is achieved by smoothly interpolating the abstract shapes as well as by the continuity inherent in the photo-realistically rendered detail.

2005-16

# Non-Photorealistic Rendering of Algorithmically Generated Trees, Master's Thesis, May 2005

Authors: Dudley, Nathan C.

April 15, 2005

Abstract: This work presents a novel rendering technique inspired by artistic approaches. Instead of trying to recreate the appearance of a traditional medium, such as charcoal or watercolor, this approach is a mixture of both photo-realism and abstraction. Artists use a process of abstraction to provide structural information about subjects that do not have clearly defined shapes, such as groups of leaves in a tree. For example, an artist will first use a color wash to approximate a group of leaves, then add detail on top of parts of this wash to indicate individual leaves. Similarly, we use an abstract shape that approximates the image of leaves clustered at the end of a branch. To prevent oversimplification, we add photo-realistic detail using a blending process. Inter-frame coherence is achieved by smoothly interpolating the abstract shapes as well as by the continuity inherent in the photo-realistically rendered detail.

SEVER INSTITUTE OF TECHNOLOGY

MASTER OF SCIENCE DEGREE

THESIS ACCEPTANCE

(To be the first page of each copy of the thesis)

DATE: April 15, 2005

STUDENT'S NAME: Nathan C. Dudley

This student's thesis, entitled Non-Photorealistic Rendering of Algorithmically Generated Trees has been examined by the undersigned committee of three faculty members and has received full approval for acceptance in partial fulfillment of the requirements for the degree Master of Science.

APPROVAL: _____ Chairman

_____

_____

WASHINGTON UNIVERSITY

SEVER INSTITUTE OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

---

# NON-PHOTOREALISTIC RENDERING OF ALGORITHMICALLY GENERATED TREES

by

Nathan C. Dudley, B.S. Computer Science

Prepared under the direction of Professor Cindy Grimm

---

A thesis presented to the Sever Institute of
Washington University in partial fulfillment
of the requirements for the degree of

Master of Science

May, 2005

Saint Louis, Missouri

WASHINGTON UNIVERSITY
SEVER INSTITUTE OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

---

ABSTRACT

---

# NON-PHOTOREALISTIC RENDERING OF ALGORITHMICALLY GENERATED TREES

by Nathan C. Dudley

---

ADVISOR: Professor Cindy Grimm

---

May, 2005
Saint Louis, Missouri

---

This work presents a novel rendering technique inspired by artistic approaches. Instead of trying to recreate the appearance of a traditional medium, such as charcoal or watercolor, this approach is a mixture of both photo-realism and abstraction. Artists use a process of abstraction to provide structural information about subjects that do not have clearly defined shapes, such as groups of leaves in a tree. For example, an artist will first use a color wash to approximate a group of leaves, then add detail on top of parts of this wash to indicate individual leaves. Similarly, we use an abstract shape that approximates the image of leaves clustered at the end of a branch. To prevent oversimplification, we add photo-realistic detail using a blending process. Inter-frame coherence is achieved by smoothly interpolating the abstract shapes as well as by the continuity inherent in the photo-realistically rendered detail.

To my wife Jennifer

# Contents

# List of Figures

# Chapter 1

# Introduction

This work proposes a novel non-photorealistic method to render algorithmically generated trees. Instead of trying to recreate the look of the traditional medium that an artist might use, such as watercolor or charcoal, our approach is inspired by how artists use a combination of abstraction and detail to convey both shape and texture in the scene. In particular, we look at how artists render trees.

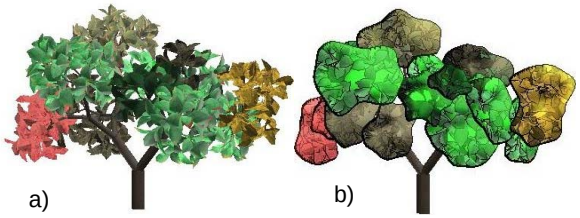


Figure 1.1: Goal of project. A traditional photorealistic rendering(a) transformed into an alternative rendering(b) that blends abstraction with photorealism.

Long before computers created their first computer generated images, humans developed many methods, some realistic, some not, for depicting complicated shapes such as trees. The first step in painting a tree with real paint is to consider the outline shape of the tree [11]. Painting every leaf would result in an overwhelming amount of detail. Instead, leaves should be viewed as a mass and painted in clumps [1]. To give a sense of texture, a couple of the leaves should be drawn accurately [2]. The goal of this work is to create images inspired by this approach.

Artists use abstraction to provide structural information about subjects that do not have obvious shapes. For example, a wash of color represents a group of leaves. To convey detail, artists then paint in representative leaves, but only in

places. Similarly, large branches are depicted in detail, while smaller branches might be left out altogether.

We mimic this approach to render algorithmically generated trees. Leaves at the end of branches are clustered into one coherent unit which is then rendered with substitute geometry that suggests the shape of the original leaves. The resulting image has a cartoon-like appearance which is then offset by introducing photo-realistic detail using a blending process. We render only the larger branches, omitting smaller ones in the leaf groups.

We provide several artistic options to alter the appearance of the final rendering. Outlining the *blobs* [1] provides visual structure to the image and enhances perspective. The location and strength of detail are important when painting. The rendering application allows the user to control the amount of detail to use and where to use it.

Ensuring inter-frame coherence is a challenge in non-photorealistic approaches. Because we are using detail drawn from the photo-realistic rendering process, this detail is visually stable from frame to frame. To ensure that the abstract information is stable, we apply the full rending process every $n$ frames, then interpolate the abstract data for the intermediate frames.

The goal of this work is to produce visually appealing images using new methods that mimic traditional artistic approaches. Since the term "visually appealing" is subjective, we provide the user with many inputs to control the results. Creating these user controls was an iterative process. Initial iterations of this work generated images that appeared washed out and simple. To address these shortcomings, silhouettes and outlines were added. Silhouettes made the leaves more distinct and outlines conveyed a sense of depth. A height field replaced the sphere blob shape approximation to create more variation over the blob shape. We achieve our goal by delivering a portfolio of tools that produce glass-like, animated images with varying levels of abstraction.

The contribution of this work is the presentation of a new, art-based rendering technique that combines abstraction with photorealism. This approach mirrors the artistic *process* rather than re-creating the appearance of a traditional medium. Although in this work we focus on algorithmically generated trees, the method could easily be extended to arbitrary scenes.

[1] A blob is a single grouping of leaves represented by an approximate shape geometry.

Previous work is reviewed in chapter 2. A description of L-systems is provided in chapter 3. Chapter 4 describes how the groups of leaves are rendered. Chapter 5 discusses maintaining inter-frame coherence. User inputs are described in chapter 6, while results are shown and discussed in chapter 7. Chapter 8 provides a conclusion.

# Chapter 2

# Previous Work

There is a large (and growing) body of work in non-photorealistic rendering. We focus on methods that have been adapted specifically for rendering trees. In the following chapter we discuss the plant generation software (L-Systems) used to generate the models for this work.


Figure 2.1: Sample previous work renderings. (a) Pen and ink illustration of trees [4]. (b) L-System generated trees [18]. (c) Stylized tree [10].

In the past, computer graphics rendering focused primarily on producing photographic images [12]. Non-photorealistic rendering, on the other hand, may focus on abstraction and simplification, such as in the work by Kowalski et al. [10]. In this approach, the authors use strokes to render 3D computer graphics scenes in a stylized manner. They suggest the complexity of the scene by drawing a few leaves in silhouettes without representing them explicitly. Other approaches mimic particular media, such as stippling [3], charcoal rendering [13], engraving [15], and half toning [7]. Deussen [4], in particular, focused on using pen-and-ink algorithms for trees.

Alvy Smith was the first to use Lindenmayer systems (L-systems) to render representations of trees [18]. The method in this work also uses L-systems to create the

base 3D models from which to draw. In particular, we use Prusinkiewicz's approach to modeling and visualization of plants using L-systems [16].

# Chapter 3

# Background

## 3.1 Using L-systems to create trees

We use L-systems to create the 3D tree mesh data. We post-process the mesh data to reduce the mesh complexity and decrease rendering time, to ensure that the meshes are closed, and to extract the branches and leaf group information.

### 3.1.1 Definition of a mesh structure

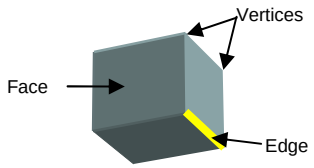


Figure 3.1: Example of cube mesh.

The 3D data used to generate the trees in this work is stored in a data structure known as a mesh. A mesh is made up of vertices, edges, and faces. A vertex is a point in space with an x, y, and z color value. A face is a subset of a plane in space. The boundary of the face is comprised of multiple edges. Each edge is created by linking two vertices together, so two neighboring edges will share one vertex. A face must have a minimum of three vertices and edges. The closed cube example has six faces, one for each side of the cube. The storage order of the vertices in a face is important, since the order of the vertices determines the direction of the surface normal. Two

adjoining faces that share two vertices cannot have the vertices in the same order. If the faces did have the same order, the two adjoining faces would face in opposite directions.

### 3.1.2 Creating the mesh

The advantage of using 3-D data [4] is that we can use automated tools to generate the trees. A 3D model of the tree provides information about the structure of the tree without having to guess it from visual clues in the image. We used an L-systems software tool, L-studio 4.0 [9], to generate the trees used in this work. L-studio consists of a set of sample models, a graphical browser, a series of editors and other modeling tools, and a library of environmental programs which simulate environmental processes that affect plant development.

In its basic form, a Lindenmayer system, or L-system, consists of a starting string of symbols from an alphabet and a series of transitions specified by a list of search-and-replace rules. The starting string is called the axiom. During each recursive step, zero or more rule-based transitions are applied to the string. Each rule consists of an "input" and an "output". The input is the substring to be found, and the output is the substring with which the input is replaced. The ordering of the rules is significant [6].

In the L-system file in L-studio there are several functions, based on growth time, that can be adjusted interactively. These functions control the size of the leaves, the diameters of branches, and the length of the branches. L-studio also provides an interactive tool to create the surface that defines the shape of the leaves. Each tree used in this work has a single leaf shape.

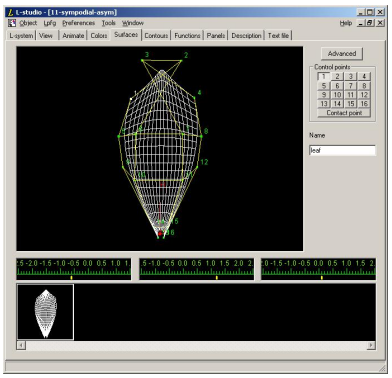

Figure 3.2: Creating a leaf mesh using L-studio.

Once the parameters are adjusted to the user's satisfaction, L-studio writes the mesh structure to a data file.

### 3.1.3 Modifying the raw output mesh

Once the tree is extracted from L-Studio, several modifications need to applied to the mesh before rendering. Branches and leaves are labeled. The resolution at which L-Studio creates the leaf and branch meshes is higher than we need, so we simplify them. Also, the leaf meshes are one sided, so we add thickness.

A conversion process reads the L-studio data file and imports the data into the format used by the rendering application. Each group is already labeled by L-studio. All leaves have 100 vertices, so anything else must be a branch, a dichotomy which is used to divide the dataset into lists of leaves and branches. Leaves and branches are treated differently and must be sorted.

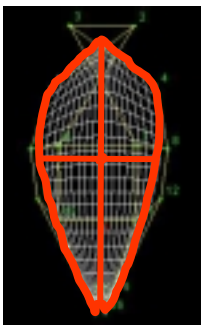

Figure 3.3: Simplifying the leaf mesh.

L-studio was designed to render in detail, but in this application the tree is zoomed out and will be visually simplified later. A high initial level of detail is not required and would add significant overhead. After the branch and leaf identification is complete, the compression process starts. To simplify leaves, only the perimeter vertices and a center vertex of the leaf are kept. The number of faces is reduced from 81 to 4 by dividing the leaf into four quadrants, as shown in fig 3.3. A branch consists of a cylinder shape with 10 vertices at the top and bottom, but to reduce the number of vertices used to store a branch, every other vertex is used.

At this point, the leaves are one sided surfaces without volume. Creating a surface for the other side of the leaf is important for lighting and rendering the leaves. One-sided surfaces rendered using OpenGL graphics tools will either be invisible from

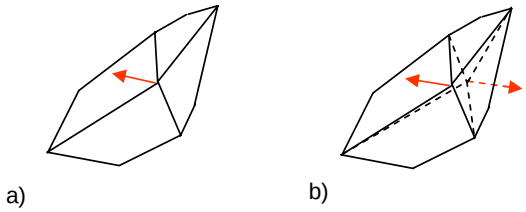


Figure 3.4: Creating leaf thickness.

one direction or will be lit from the wrong direction. To solve this problem, the center of the leaf surface is expanded in the direction of the surface normal, flipped, and added to the other side of the leaf, as seen in fig 3.4.

To improve computation time, the rendering application makes use of a graphical rendering technique known as an identification buffer, or an ID buffer. The color of every face belonging to a single group is set to a unique id. The resulting rendered image indicates which face was rendered to which pixel. We use this information to determine which pixels a leaf group (or branch) occupies. This process is much faster than using ray-tracing and checking the volume that every leaf or branch occupies. The ID buffer is also used for user selections to the pick textures and intensities applied to the mesh.

We apply textures to the leaves and branches as seen in figure 4.3, by applying bitmap images to the faces of the mesh. The OpenGL graphics tools align the vertices of a face with pixels in the bitmap and then stretch the image accordingly. The textures are a mix of photographs and hand-painted images. The rendering application automatically determines the orientation of the textures so that the top of the texture corresponds to the tip of the leaf. The texture for any particular leaf is controlled by the user.

## 3.2 Clustering leaves

We need to group the leaves based on their level in the hierarchy (essentially, all leaves produced after level $l$ are grouped). When L-studio creates the output file, it lists the leaves and branches in the order that they were generated. To determine the group to which a leaf belongs, we use a threshold value based on the diameter of a branch. The program reads leaves until the next time that the diameter of a branch crosses the pre-defined threshold value, which triggers a new group of leaves to be

started. This process is repeated until all of the leaves are read and placed into a unique grouping. The meshes belonging to the branches below level $l$ (i.e., the trunk and big branches) are separated and labeled as a single group.

In summary, this project uses L-systems to generate a 3D data structure representing a tree. For practicality, the 3D data is reduced to a low-resolution tree whose leaves and branches are then identified and marked. Once the leaves and branches are identified, textures can be applied to the faces to provide a photorealistic appearance. In the following sections, the photorealistic tree will be transformed into a non-photorealistic tree.

# Chapter 4

# Rendering Process

This section describes the rendering process for creating a single frame, as shown in figure 4.1. The rendering process begins by clustering leaves into groups (blobs). We create approximate, 3D, screen-aligned geometry for each group (the blob geometry). This geometry serves two purposes. First, we render the blob geometry using approximate colors to produce an abstract rendering of the group. Second, we use the blob geometry to produce an alpha mask which tells us where to add in detail. This alpha mask is used to combine the abstract rendering with a photo-realistic rendering of the leaves to create a *billboard image* [1] for the blob. (Note: By photo-realistic we mean traditional OpenGL, or ray tracing, Phong, etc., rendering.) Once the billboard images are created, we add depth and combine them into a single image with the large branches and trunk.

## 4.1 Lighting

The user specifies two sets of lighting arrangements. The first set of lights, the scene lights, lights the tree in the traditional manner. The second set of lights (the *blend lighting*) determines where to mix the photorealistic rendering with the non-photorealistic rendering. This process is described in more detail in section 4.2.2.

[1] A billboard image is a multi-layered image, where several layers are stacked to create a final image.

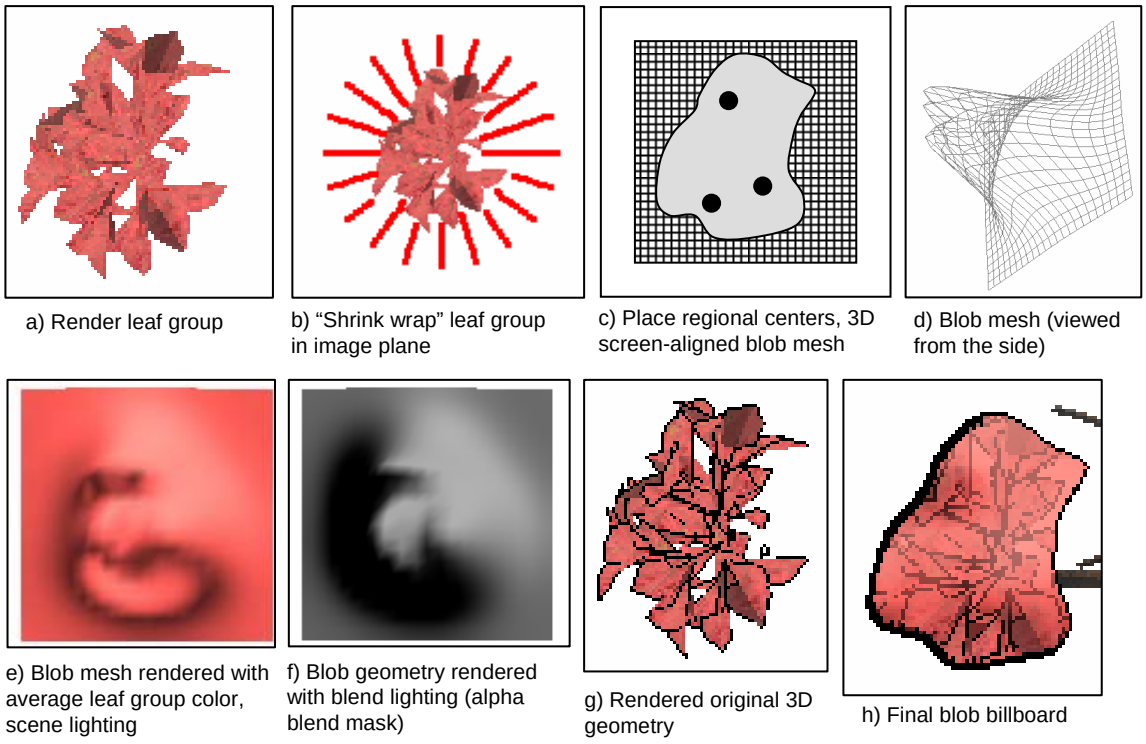


Figure 4.1: Creating a single blob. First, we render the geometry and shrink wrap it in image space. Second, we construct the approximate 3D blob geometry. This geometry is a screen-aligned height field pointing in the direction of the viewer. This geometry is used to create two images. The first image is the approximate, cartoon-like rendering. The second is the alpha-blend mask used to blend between approximate image and the detailed image. This image becomes a billboard, trimmed to the shrink-wrapped blob outline.

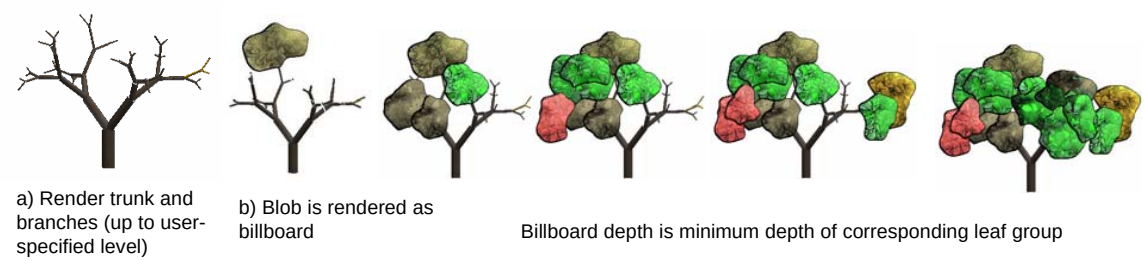


Figure 4.2: Combining branches and blobs into single image

## 4.2 Rendering blobs

Each group of leaves is processed independently. We construct several images: an ID image (which is used to create the approximate geometry), a rendering of the approximate geometry, an alpha blend mask, and the detail rendering. The formula for combining these images is below.

> Final = approximate geometry $*$ alpha blend mask + detail $*$ (1 - alpha blend)

### 4.2.1 Blob approximate geometry

The goal of grouping the leaves together is to make a blobby image shape that captures some of the 3D detail and shading of the leaf group. This process is done in image space, essentially "shrink-wrapping" the leaves.

The group of leaves is first rendered with all other branches and leaves turned off. A circle typically comprised of 20 control points is placed around the leaves. The control points are "shrink wrapped" to the group of leaves by moving them 20% of the distance between the current position and the center of the blob. This process is repeated until the point is on a leaf (see fig 4.1b). A B-spline connects the points into a smooth curve, and is also used later to draw the blob's outline.

Next, we create 3D geometry to approximate the shape of the blob, which takes the form of a screen-aligned height field, with the height field pointing toward the user. A 10 by 10 unit mesh square, oriented to face the viewer, is substituted in the position of the leaves in 3D space.

To create the height field, we place three Gaussian peaks inside the blob boundary. To find the centers of the peaks, we take the control points of the outline and divide them into thirds. The average point of one third of the boundary is found and

combined with the center point of the entire blob to create three regional centers of the blob. The regional centers are in pixel coordinates, and the shape approximation is in world coordinates. The ID buffer converts from pixel coordinates to world coordinates. The actual height is determined by scaling the maximum distance from any vertex in the blob to the center (fitting the mesh around the leaves).

To create the color wash, the blob approximation color must match the leaves in the blob. The color is determined by taking the average color of the leaf texture. The blob geometry is rendered using the same lighting used to render the tree (see fig 4.1e).

### 4.2.2 Adding detail

The detail texture is created by rendering the leaves. To emphasize the detail in the leaves, we optionally let the user highlight the silhouette [17] edges of each leaf, which emphasizes the outline of the leaf and deemphasizes the texture of the leaf. The silhouette color can vary from black to the blob color to white, depending on input from the user. The blob color is determined from the average color of the leaf texture.

$$C_{user} * C_{blob} \Rightarrow C_{silhouette}$$

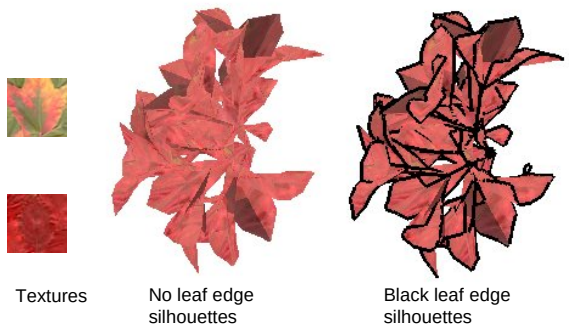


Figure 4.3: Options for drawing a group of leaves. (Left) Textures used (Center) Leaves drawn without edges highlighted. (Right) Leaves drawn with edges hilighted.

To created a combination of photorealism and non-photorealism, the cartoon image and the detailed leaves are blended. To determine "how" and "where" to blend, an alpha blend mask is created by rendering the blob geometry with the blend lights. Any place on the shape approximation rendering that is darker will get detail copied

over it. The user can select the location of the blend light and the strength of the light.

At this point there are several saved images, including an image of the leaves with edges (see fig 4.3), an image of an outline (see fig 4.1b), an image of the shape approximation (see fig 4.1e), and an image used to calculate the alpha blend mask (see fig 4.1f). All of these images are combined into a single billboard image (see fig 4.1h) for the group, which is clipped to the outline.

We look at all of the pixels that constitute the blob, and then copy detail according to the alpha/gray scale image created by rendering the blob. We use a flood fill algorithm to find all of the pixels in the blob.

To help differentiate neighboring blobs, an outline is drawn around the blob, following its boundary. The color of the outline is controlled as a percentage of the color of the underlying shape approximation. The thickness of the outline is controlled by direction. When the edge is to the left of a current pixel, the outline is drawn wider than if the edge is in a different direction. This varying thickness provides a greater feeling of depth.

## 4.3 Final image

The Painter's algorithm [14] is used to combine the individual blob images (since all of the blob images are screen aligned). The large branches are rendered without leaves as the base layer, and each blob is then layered into the image. The order of the layers is determined from the minimum depth value of any leaf in the blob, as seen in figure 4.2.

# Chapter 5

# Animation

An advantage of computer graphics over traditional hand-painted images is that moving the camera around the scene is "free". The biggest challenge is to maintain inter-frame image continuity. Rendering each frame individually causes the boundaries of blobs to jump, regional centers to bounce around, and blobs to pop in and out as the depth changes between frames. To solve these problems, we generate new blobs every *nth* frame, then interpolate depth and shape information between those frames.

## 5.1 Pre-rendering

To pre-render, we assume a pre-defined camera path. For a user-defined $n$, a completely new set of abstract data is calculated every *nth* frame by "shrink wrapping". The control points for the blob outline and the depth are interpolated between the *nth* frames. The regional centers for each blob remain fixed relative to the changing size of the blob. The detail rendering is generated for each frame using the interpolated control points of the "shrink wrap".

The number of control points in the boundary is the same between all blobs in all frames. The boundary between the the control points is interpolated using the formula for a B-spline. A B-spline is also used to interpolate the location of the control points. Since a B-spline is used, four frames are required to interpolate the location of the control points.

For each blob in each key frame, the minimum depth of the leaves in the blob is determined. If one blob will pass in front of another, we alpha blend between the two bobs over a period of $n$ frames as shown in figure 5.1. This prevents one blob from "popping" in front of the other.

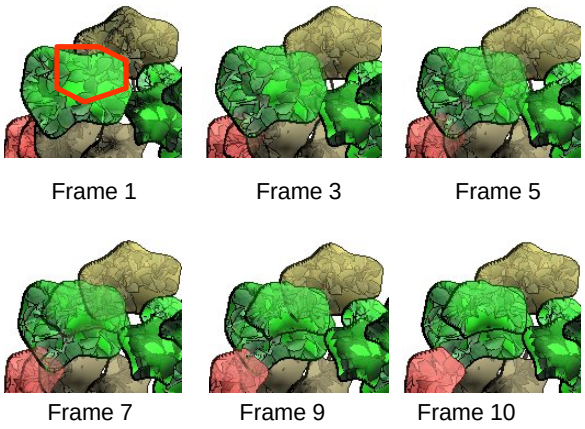


Figure 5.1: Depth blending over 9 frames.

# Chapter 6

# User Controls

The many inputs of the rendering application constitute a new medium for an artist. The rendering application is a tool to create art; the application alone is not the artist. The user has control over basic camera views, the mesh appearance, the scene lighting, how detail is placed, and the camera path during animation.

## 6.1 Mesh Controls

The key item controlling the appearance of the mesh is texture. Before the texture is applied, each mesh element has its own greyscale value. The greyscale value of the mesh element determines the brightness or intensity of the texture. To adjust the greyscale values, the user selects mesh elements and then adjusts a slider to change the greyscale value, ranging from black to white. Before textures can be applied, the user must tell the rendering application how the textures will be applied to the mesh. The texture can either be optimized to fit over a single face or stretched to fit across an entire leaf. To choose the textures, the user loads bitmap images. The bitmap images must be squares, with the value of the image dimensions in powers of two. Some valid bitmap sizes are 32 x 32, 64 x 64 pixels, etc. An index value is assigned in the order that the images are loaded. The user selects a leaf or branch to change and enters a new index value. The tree is then redrawn with the new corresponding texture.

The user can select the textures used in the mesh and the intensity of the texture. Each leaf and branch in the tree is individually selectable. Options for selecting include leaf only, branch only, recursively, and randomly. The leaf only option allows the user to select only leaves to modify. Likewise the branch only option

allows the user to select only branches. In recursive selection, when the user clicks on a branch, every group above the selected branch is also selected. The branch only and leaf only filters still apply when the recursive option is selected. If the user wants to create more variation and contrast in the tree, there is also a random select option. Invoking this option means that when the user clicks on a branch, the rendering application searches up the tree. At each leaf or branch, a random number is computed. Depending on a user-set threshold value, the leaf or branch might be selected.

## 6.2 Lighting Controls

Another input controlled by the user is the scene lighting. A graphical user interface is provided to modify the lighting properties. The rendering application uses OpenGL standard lighting, which provides for eight different lights and an ambient light. The eight lights can be either positional, directional, or spot lights, and the user can turn each light on and off. Each light has a diffuse, specular, direction, and position property. The lights can be fixed either relative to the scene or relative to the camera.

## 6.3 Rendering Controls

The control that provides the most visual impact in the rendering process is the *division diameter*. The division diameter controls the number and size of the blob groups. When selecting a division diameter, the easiest way to determine the diameter of a specific branch is to click on the branch to print the diameter in the console window.

The user has the option to render only a single blob by entering the blob index, which is useful when experimenting with different parameters. A toggle switch flips between the photorealistic rendering in real-time and a saved image of the the non-photorealistic rendering.

To change the appearance of the blobs, the user can adjust the number of control points, the silhouette size and color, the color of the edge of the blob, where to place the detail, the translucency of the blob, and the height of the blob peaks. The number of control points adjusts the smoothness of the blob. The color of the silhouette is determined as a red, green, and blue fraction of the unlit blob color, determined by the average color of the texture. For example, setting all of the fraction

values to zero makes the silhouette black. Setting the fraction values to one makes the silhouette the same color as the blob color. A ceiling is implemented to prevent overflow of the colors, allowing the user to enter a high value such as 10 to make a white silhouette. The color of the blob edge is determined in a similar manner, except that instead of using the unlight blob color, it uses the lit blob on a pixel by pixel basis.

While the rendering application automatically chooses the location of the blob peaks, the user controls the height of the peaks. The user input is a fraction. An input of zero makes the shape flat, while an input of one makes the blob peak height equal to the width of the blob. In this work, the number of blob peaks was arbitrarily set to 3, but it could be any number.

The user has some control over the placement of the original detail by manipulating the blend lighting, which is separate from the scene lighting. The user adjusts the amount of ambient, diffuse, and specular light, using a positional, directional, or spot light. The location of the light is determined relative to the viewpoint. As a fraction of the distance from the look from and look at points, the light position can be moved along the right and up vectors. To preview the lighting, the user renders a single blob and then clicks the show blob button. This displays the blob approximation using the lighting for the detail selection.

## 6.4 Animation Controls

For this implementation the user defines a camera path and how often to take key frames. The camera path for this application is basic: The user determines the zoom amount per step, the rotation amount per step, and the axis of rotation around the center of the mesh. The number of steps to take determines the length of the movie, and the sample every $Nth$ frame value determines how often to render frames in the pre-rendering process. Future applications could use any arbitrary camera path.

Taken together, the user-defined controls enable a tool set of non-photorealistic rendering effects through mesh appearance, lighting, rendering, and animation.

# Chapter 7

# Results

In this section we show examples of renderings for a tree and a bush. We compare adjusting parameters for the number of control points, light settings for alpha blob, and color settings for outlines. We were unable to include multimedia attachments with this version of the work.

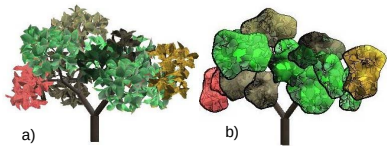


Figure 7.1: Tree rendering

The first test subject used is a tree with a division diameter of 0.042443 shown in fig 7.1. The division diameter automatically generates the groupings of leaves by measuring the diameter of the branches. Twenty control points were used. The edges are black. The alpha blending controls were set to 70% diffuse and 0% ambient, and the light position was set to right:1 up:1. The right and up vectors are proportional to the distance from the view point to the look point.

The second test subject used is a bush with a division diameter of 0.068513. Twenty control points were used. The edges are black. The alpha blending controls were set to 70% diffuse and 0% ambient, and the light position was set to right:1 up:1.

The number of control points affects the smoothness of the blob shapes. In figure 7.3a ten control points were used to outline the blob, in figure 7.3b twenty control points were used to outline the blob, and in figure 7.3c thirty control points were used to outline the blob.

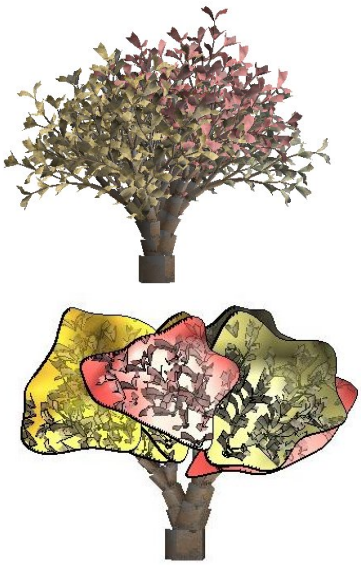

Figure 7.2: Bush rendering

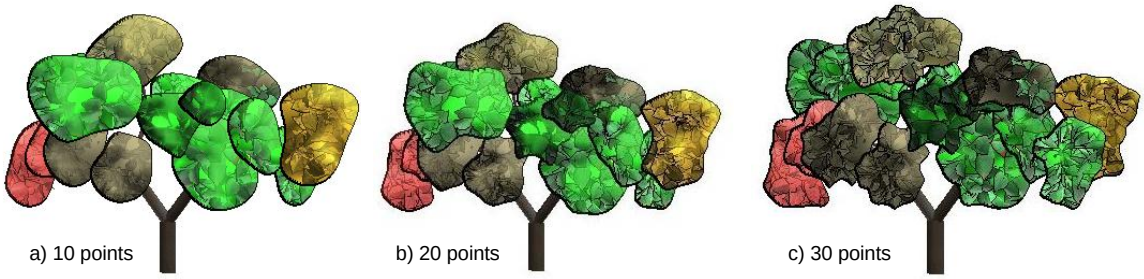


Figure 7.3: Number of control points used to create outlines. (a) 10 control points. (b) 20 control points. (c) 30 control points

The user has the option to affect where detail is placed on the blob by placing a light relative to the viewpoint. In both cases below, the diffuse light is set to 100% and the ambient and specular lights are turned off. The examples below show the effect of moving the light from the top right corner to the bottom left corner. Where the alpha blending control image is darker, there is more detail in the resulting image.

The translucency of the blobs is an optional setting, set in most cases to zero. In figure 7.6, the translucent setting is varied between 20% and 60%. Detail from underlying blobs is more appearent with the 60% setting than the 20% setting.

In figure 7.7, five different sets of textures and intensities are used. The first set uses one green texture, with one intensity level. Any variation in the rendering is due to scene lighting. The following tree uses one texture, while the intensity of the

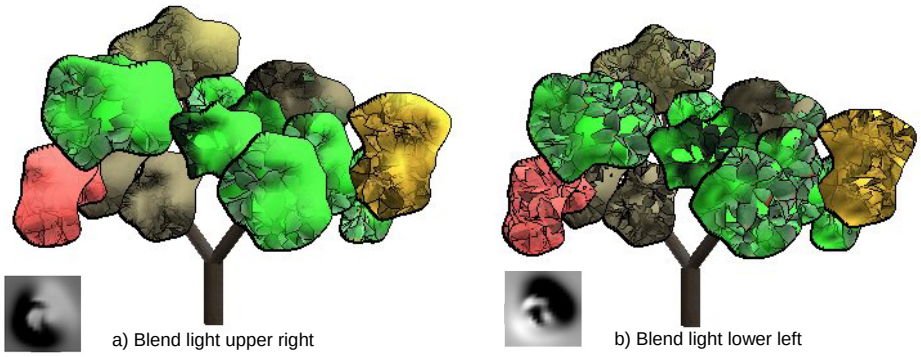


Figure 7.4: Blending light. (a) Blending light is placed above and to right of viewer. (b) Blending light is placed below and to left of viewer.

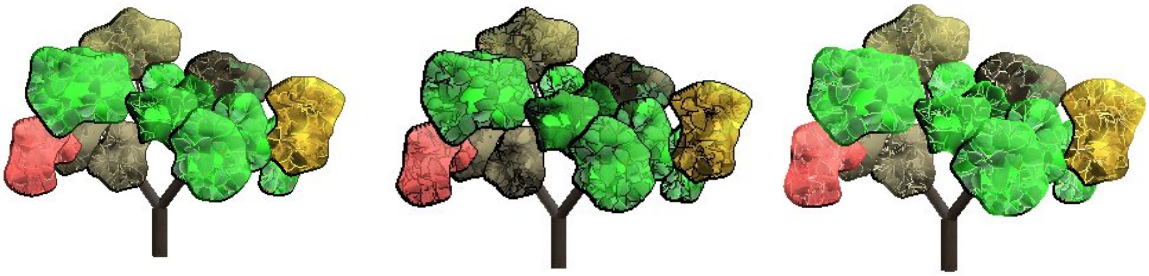


Figure 7.5: Adjusting control for edge color. (a) 0% edge color setting. (b) 100% edge color setting. (c) 200% edge color setting.

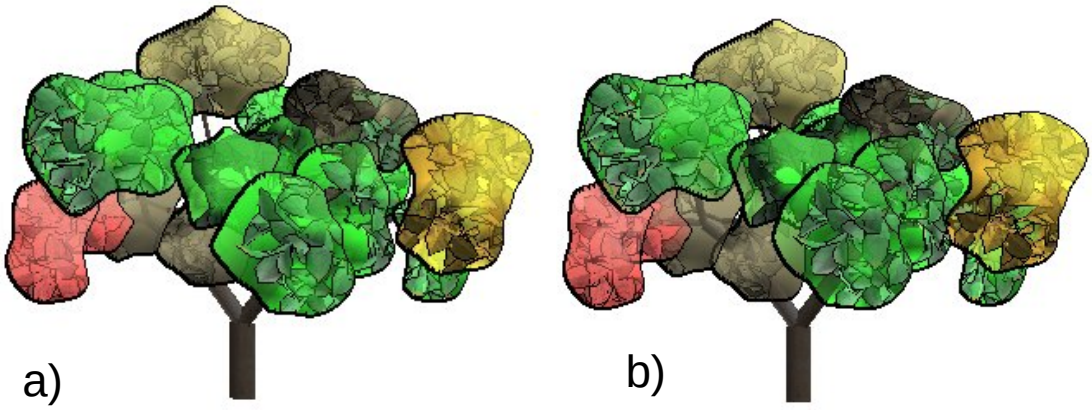


Figure 7.6: Adjusting control for translucent value. (a) 20% translucent setting. (b) 60% translucent setting.

texture varies from blob to blob. In the two next sets, new textures are introduced. The last set uses a red texture only.

The user has the option to adjust the color of the blob outline and the silhouette of the leaves. The images in figure 7.5 illustrate three different settings.

Figures 7.8 and 7.9 depict the results of the animation for the tree and the bush. At this time, we are unable to include the actual movies with this work. The sequence of images shows only every 10th frame as the tree and bush are rotate around the Y axis.

## 7.1 Rendering Time

On a typical tree with 16 blobs, the rendering time per frame is approximately 1 minute. The largest variable that affects time to render is the number of blobs rendered. The average blob takes 4 seconds to render, and each step to render a blob takes about 1 second. Most steps require a unique rendering and image save. Surprisingly, the fastest step is the flood fill algorithm used to combine the detail image with the shape approximation, which takes less then 1 second. A tree with approximately 8 blobs takes 30 to 45 seconds to render, while a tree with 20 blobs takes 60 to 90 seconds to render.

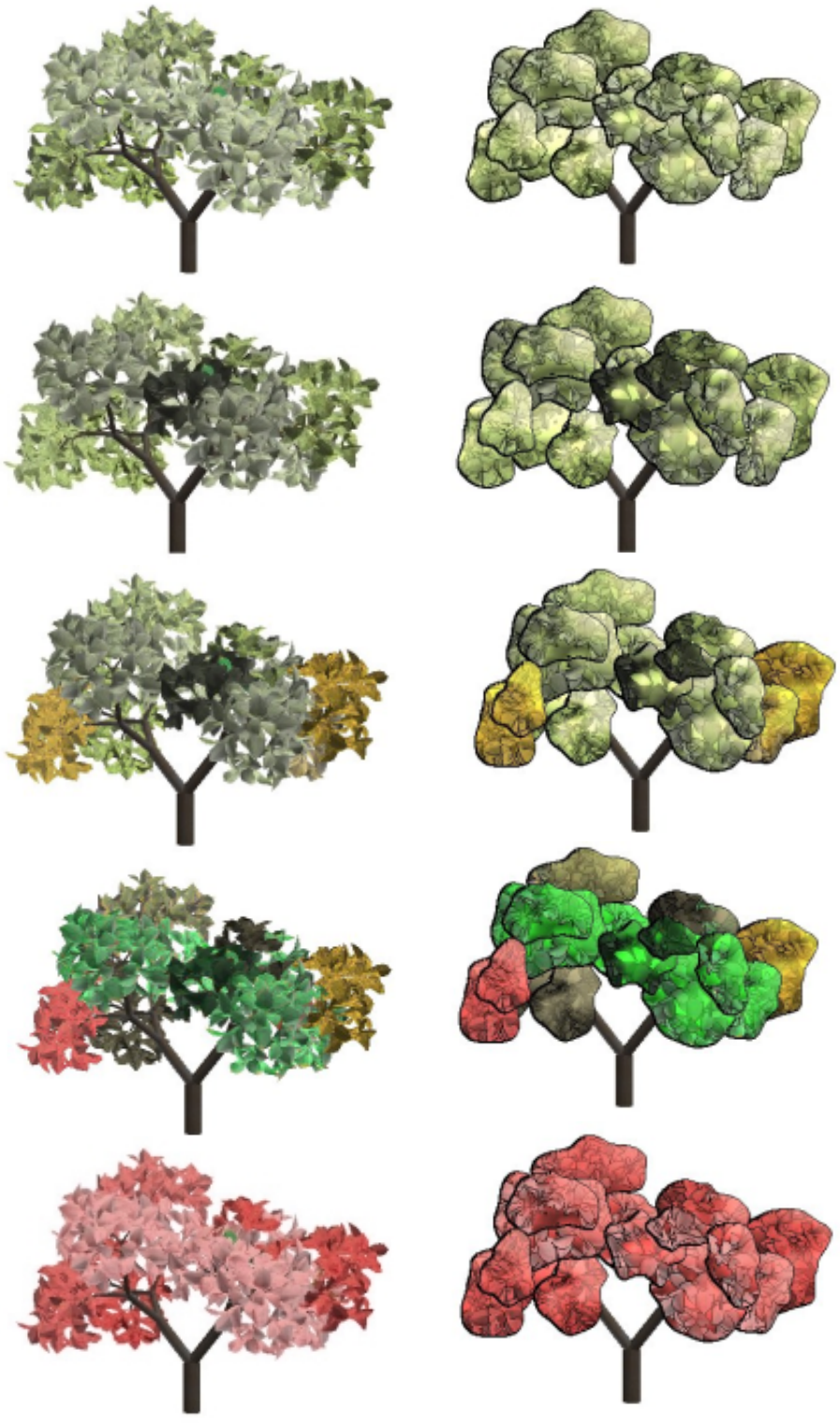

Figure 7.7: Adjusting variation in texture and intensity between groups of leaves.

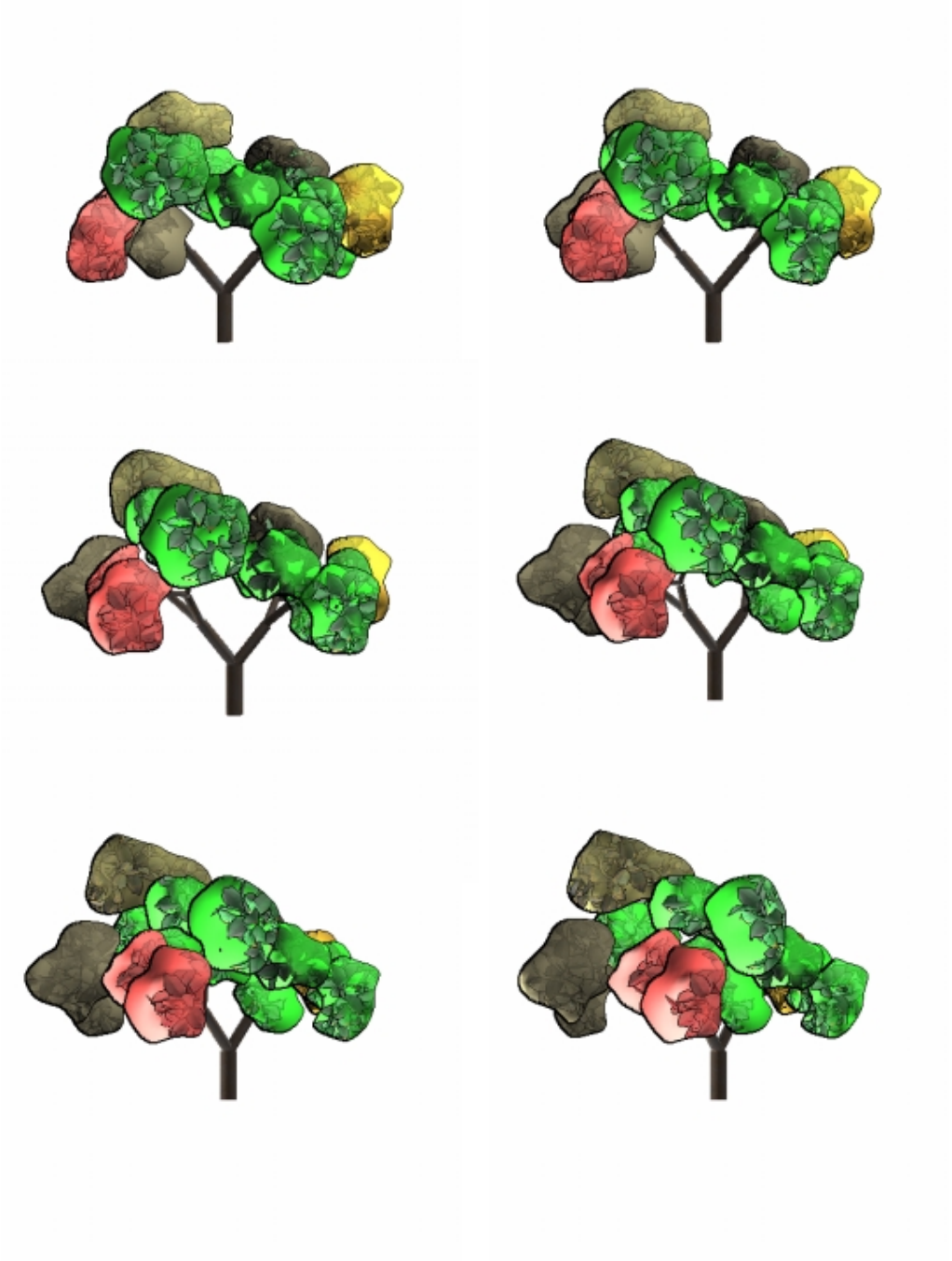

Figure 7.8: Showing every 10th frame of a rotating tree.

Figure 7.9: Showing every 10th frame of a rotating bush.

# Chapter 8

# Conclusion and Future Work

## 8.1 Conclusion

Non photorealistic rendering of photorealistic images draws inspiration from the age-old techniques employed by artists in depicting detailed structures, while extending the foundational work of Kowalski's stylized 3D renderings [10] and Smith's L-system implementations [18] in the field of computer graphics. This project transforms a realistic tree mesh into a work of art, by methodically grouping, lighting, sorting, outlining, and texturing the leaves and branches of an L-system structure to produce an animated tree.

This research combines a collection of non-photorealistic rendering techniques. A mesh produced using L-systems plays the role of an artist's armature - giving a structural frame work to the rendering. Recursive application of user-selected patterns to leaves and branches provides the crucial approximation of an artist's use of texture.

User configured and ambient lighting approximate an artist's use of shading. Approximating the shape of the photo-realistic structure creates the illusion of depth, much like that produced by an artist when he employs a light base wash to create a background with leaves or other shapes in darker colors. Together these non-photorealistic techniques transform a series of realistic frames into an animated work.

## 8.2 Future work

Future work may consider speeding up the rendering process or making the rendering near real time. Many of the algorithms implemented in this work are performed in

software image space. Moving the workload to hardware using texture maps and alpha masks would significantly improve rendering times. Rendering pine or spruce trees would be more challenging to render than a leaf texture. Additional research may also include simulating a tree swaying in the wind. There are many other types of scenes, such as flowers, to which the methods presented in this work could be applied.

# Appendix A

## A.1 Texture maps used

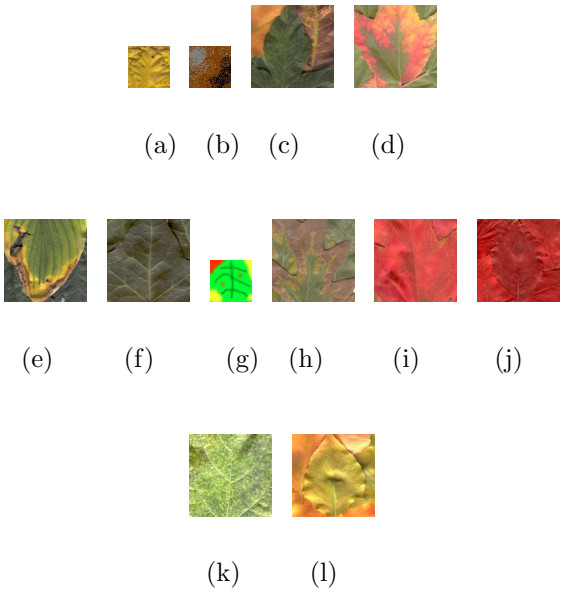


Figure A.1: Texture maps applied to tree mesh. (a) Fall yellow. (b) Hand created bark. (c) Fall leaves. (d) Maple. (e) Hosta. (f) Ivy. (g) Hand leaf. (h) Oak. (i) Fall red. (j) Fall red modified by hand. (k) Spotty green leaf. (l) Greenish-yellow leaf.

## A.2 L-systems tree file

```
define STEPS 200
    define dt 0.05
    define PL 1.00000 /* plastochron */
```

```
define D 10 /* elongation time */
define ANG 40 /* branching angle */
derivation length: STEPS;
module A(float);
module I(float);
module K(float);
Axiom: SetWidth(0.03) RollL(60) A(0);
production:
A(t):   produce A(t+dt);
I(t):   produce I(t+dt);
K(t):   produce K(t+dt);
decomposition:
A(t):
t = t - PL;
if(t greater than 0)
produce
I(t)
RollR(90)
SB() Left(ANG) A(t+0.3) EB()
SB() Right(ANG) A(t-0.3) EB()
K(t);
interpretation:
maximum depth: 3;
I(t):
produce
SetWidth(0.1 * func(line, t/D))
F(func(internode, t/D))
SB() Down(45+30*t/D) Surface(0, 0.4*func(leafSize, t/D)) EB()
SB() RollR(180)Down(45+30*t/D)
Surface(0, 0.4*func(leafSize, t/D)) EB();
K(t):
produce
F(0.4*func(flower, t/D))
IncColor() SetWidth(0.1*func(flower, t/D)) Sphere0();
```

# References

[1] Marion Boddy-Evans. Painting trees: Tips for rendering realistic and believable trees. *About*, 2005.

[2] Susan D Bourdet. Painting the allure of nature. pages 56–77. Northern Light Books, 2001.

[3] Oliver Deussen, Stefan Hiller, Cornelius van Overveld, and Thomas Strothotte. Floating points: A method for computing stipple drawings. *Computer Graphics Forum*, 19(3), 2000.

[4] Oliver Deussen and Thomas Strothotte. Computer-generated pen-and-ink illustration of trees. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 13–18. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[5] Fabian Di Fiore, William Van Haevre, and Frank Van Reeth. Rendering artistic and believable trees for cartoon animation. In *Proceedings of Computer Graphics International (CGI 2003)*, pages 144–151, July 2003.

[6] Erik M. Francis. L-system. 2002.

[7] Bert Freudenberg, Maic Masuch, and Thomas Strothotte. Real-time halftoning: a primitive for non-photorealistic shading. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*, pages 227–232. Eurographics Association, 2002.

[8] Amy Gooch, Bruce Gooch, Peter Shirley, and Elaine Cohen. A non-photorealistic lighting model for automatic technical illustration. *Computer Graphics*, 32(Annual Conference Series):447–452, 1998.

[9] Radoslaw Karwowski and Brendan Lane. L-studio/lpfg: A software system for modeling plants. 2004.

[10] Michael A. Kowalski, Lee Markosian, J. D. Northrup, Lubomir Bourdev, Ronen Barzel, Loring S. Holden, and John Hughes. Art-based rendering of fur, grass, and trees. *Proceedings of SIGGRAPH 99*, pages 433–438, August 1999. ISBN 0-20148-560-5. Held in Los Angeles, California.

[11] John Lovett. How to paint trees. 2004.

[12] Lee Markosian, Michael A. Kowalski, Samuel J. Trychin, Lubomir D. Bourdev, Daniel Goldstein, and John F. Hughes. Real-time nonphotorealistic rendering. *Computer Graphics*, 31(Annual Conference Series):415–420, 1997.

[13] Barbara J. Meier. Painterly rendering for animation. *Computer Graphics*, 30(Annual Conference Series):477–484, 1996.

[14] M. E. Newell, R. G. Newell, and T. L. Sancha. A solution to the hidden surface problem. In *ACM'72: Proceedings of the ACM annual conference*, pages 443–450. ACM Press, 1972.

[15] Victor Ostromoukhov. Digital facial engraving. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 417–424. ACM Press/Addison-Wesley Publishing Co., 1999.

[16] P. Prusinkiewicz. Art and science of life: Designing and growing virtual plants with l-systems. In *ISHS Acta Horticulturae 630*, pages 15–28. Acta Horticulturae, 2004.

[17] Ramesh Raskar and Michael Cohen. Image precision silhouette edges. In *SI3D '99: Proceedings of the 1999 symposium on Interactive 3D graphics*, pages 135–140. ACM Press, 1999.

[18] Alvy Ray Smith. Plants, fractals, and formal languages. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 1–10. ACM Press, 1984.

# Vita

Nathan C. Dudley

**Date of Birth** June 10, 1979

**Place of Birth** Grand Rapids, Michigan

**Degrees** B.S. Computer Science

**Publications**

May 2005