ABSTRACT AND REALISTIC TREE MODELING

by

Ling Xu

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

Carleton University
Ottawa, Ontario

Abstract

Tree models are widely needed in 3D movies, games, and artistic presentations. Their styles abound. Due to the complexity and the variety of trees, realistic tree modeling remains a challenge. For automated methods, the objectives of speed and controllability are elusive. Data-driven and manual approaches have their own drawbacks: it is inconvenient to obtain laser scans or photographs of a desired tree, and human intervention can be costly. Another different style of trees, abstract trees, possesses stylized branches and aesthetic patterns. However, abstract tree modeling with an emphasis on branching structures receives less attention. Abstract trees in the art style of Gustav Klimt has never been explored in computer graphics.

In this thesis we provide procedural methods in two regimes – realistic tree modeling and abstract tree modeling. For realistic tree modeling, our work is based on the idea of using least-cost paths – in a weighted graph the least-cost paths between selected endpoints and a root point form a tree structure. Through the sophisticated design of graph creation, endpoint placement, and the setting of edge weights, our tree modeling system is capable of generating a wide range of realistic tree models. For abstract tree modeling, we propose magnetic curves, a particle tracing method to create stylized trees in the art style of Gustav Klimt. The method is also versatile to create many aesthetic forms with curves. Overall, this thesis contributes effective procedural tree modeling methods, and the resulting high-quality results can be used in many computer graphics applications.

Acknowledgements

First and foremost I would like to thank my supervisor Dr. David Mould. It has been an honor of being your student and working with you in this long journey. Thanks for your encouragement, your advice, and your patience. Your attitudes towards research and teaching will guide me in my future career.

I also want to thank the thesis committee – Michiel Smid, WonSook Lee, Stephen Brooks, and Andrew Simons. Your invaluable suggestions and comments helped me to better finish the work and achieve this thesis.

Thanks to my family – my parents Youying Feng and Yinghao Xu, my husband Chunhua, and my son little Henry. It is your support that makes me brave enough to pass through the difficult times. You are my warm harbor forever.

I would like to thank my old friends in GIGL lab – Hua Li, Gail Carmichael, Raj Wasson, Jamie Madill, Maryam Ariyan, and Jacquelin Caron. Thanks for sharing those beautiful days with me in Ottawa. I also thank my new friends in GIGL, especially Ramin Modarresiyazdi, for your extensive comments and interesting talks.

Last but not least, I am grateful for the financial support from Carleton University, GRAND, and NSERC.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Tree modeling

Trees are commonplace in the natural world. Geometric tree models are widely used in computer-generated virtual environments, where styles of trees abound. Realistic trees are important components of outdoor scenes in 3D games and movies; they are also useful in scientific areas such as forestry and plant taxonomy. Abstract trees may also appear in cartoon films and artistic decorations.

Due to the complicated tree structures and their rich variations, creating trees is not an easy task. It is tedious and time-consuming to create tree models manually. We want to create realistic-seeming, complex, intricate tree models automatically for wide applications in computer-animated movies, games, and scientific research areas.

Our work considers two different styles of tree models: realistic trees and abstract trees. Realistic tree modeling pursues an effect of realism. The resulting models should possess branching patterns similar to trees in the real world. Abstract tree modeling pursues an aesthetic effect. The resulting models should have stylized tree structures that evoke a sense of art.

In this thesis, we emphasize realistic tree modeling. Methods to create realistic tree models roughly fall into two categories: data-driven methods and procedural methods. Next we briefly describe these methods.

Data-driven methods for tree reconstruction use data from scanning or photos to create models that are faithful to real trees. A 3D scanner can be used to sample points on the surface of a tree; it outputs position, color and texture of each point. Users can obtain an accurate description of the scanned tree. However, most 3D scanning devices are too specialized and complex for novice users to operate [6] and there are common problems associated with scanned data such as the occlusion of leaves and overlapping of adjacent trees. Reconstruction methods emphasize how to deal with the drawbacks of data to recover

occluded tree branches. Due to the use of data from real trees, data-driven methods can generate high-quality tree models, but they are not capable of generating new trees.

Procedural methods create models by abstracting complex details of objects into a function or an algorithm [70]. They are capable of generating models with complicated structures such as trees, owing to their data amplification ability – "a few parameters (or a small amount of geometry) magically expand into a large, detailed model" [70]. Procedural methods can involve botanic knowledge to create tree models by simulating the growth process of real trees, or pursue a visual simulation of tree appearance based on a recursive or hierarchical branching pattern. For previous procedural methods, to effectively control the global tree shape and branch shapes at the same time is a challenge, and the objectives of speed and controllability are elusive. In this thesis, we propose a procedural tree modeling system to model visual appearances of tree structures. It is fast, and capable of controlling global shapes and branch shapes.

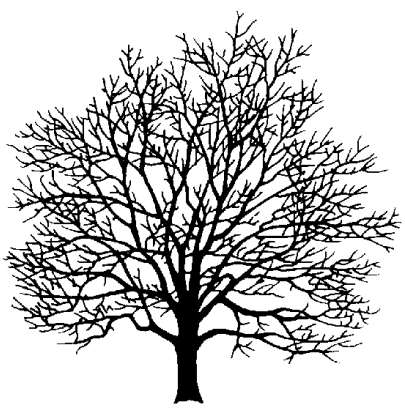
To create a realistic tree model, firstly we need to look at tree structures. Two examples of tree structures are shown in Figure 1.1. In the top image, we see some typical characteristics of trees: irregular branch shapes, long and thick primary branches, and slim twigs. The tree seen in the lower image includes a single trunk, some main branches, and smaller branches that form a hierarchical structure.

Although a tree has many components, such as roots, leaves, bark, flowers, and fruits, we are interested in branches. Branching tree structures, such as examples in Figure 1.2, are evocative of tree shapes even without other tree components.

Trees have rich variations of branching structures [79]. We are not just trying to create one tree, but want to create a wide range of trees that have variations in the following two aspects: global shape and intermediate-scale architecture. In the following, we use the trees in Figure 1.2 as examples to illustrate these two aspects.

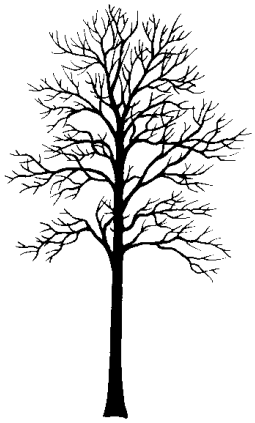
In this thesis, *global shape* refers to a tree's overall appearance. For example, a tree could be roughly mushroom-shaped or cone-shaped. In our opinion, the following aspects affect the global shape of a tree: 1) the tree has a typical shape with a central trunk supporting a crown such as tree (e), or a shrub-like shape lacking a discernible central trunk such as tree (d); 2) properties of crown and trunk: shape, width, height, and position. For example, the tree could be slim and tall like tree (b) or wide and short like tree (d).

Figure 1.1: Examples of tree structures. Images from Flickr.com.

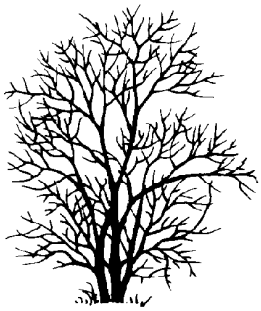


(a) red mulberry

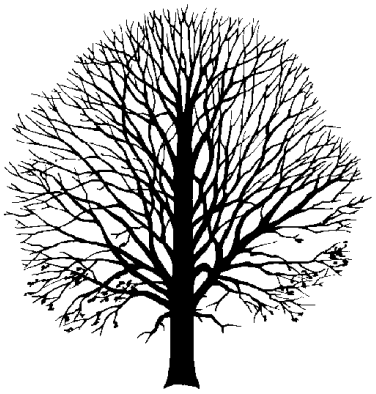
(b) water birch



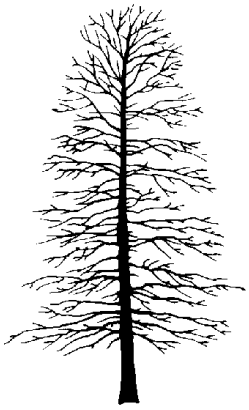
(c) tulip-tree



(d) American mountain-ash



(e) American beech



(f) black-gum

Figure 1.2: Tree silhouettes with only branches. Images from Natural Resources Canada [39–44].

In this thesis, *intermediate-scale architecture* refers to a tree's branching pattern and branch shapes. The branching pattern of a tree is how a trunk splits into primary branches. A trunk may split into a few primary branches at about the same place such as in tree (a), or have side branches along most of its length such as tree (f). Branch shape shows how a branch changes orientation along its length. Branch shapes differ among trees: in tree (b), all branches point upward; in tree (c), lower branches curve downward and upper branches grow upward; in tree (f), most side branches are horizontal and only a few top branches point upward.

We are interested in the global shape and the intermediate-scale architecture of trees for the following reasons. The global shape greatly affects a tree's appearance. If two trees have distinct global shapes we can tell they are different at first sight without looking at branch details. However, modeling trees by only describing their global shapes is not sufficient. Even if two trees have similar global shapes, they may still look quite different, such as tree (a) and (e) in Figure 1.2. We need to look at their intermediate-scale architectures, which give a more detailed description of tree structures. We do not care much about small-scale features such as twigs because they are not a key factor that affects the appearance of a tree.

We propose a tree modeling system based on the idea of path planning. Path planning is the problem of finding least-cost paths in a weighted graph [37]. We use path planning for tree modeling because least-cost paths connecting selected endpoints and a root point form a tree-like branching structure. Using path planning also provides the following benefits. First, path planning is a well-studied problem in computer science which makes it easy to understand and implement. Second, computing least-cost paths is fast. Third, a wide range of tree structures can arise from variations of controllable elements in the path planning process.

We present two designs for path planning tree modeling systems: the method with iterated graphs (MIG) and the method with guiding vectors (MGV). In MIG, we explore how hierarchical graph creation and endpoint placement affect the resulting tree structures. MIG provides effective control over the global shape of trees, but the control of intermediate-scale tree architectures is weak. The resulting branches do not curve in a way resembling real tree branches. Our experience in MIG helped us to develop MGV. In MGV, we use

guiding vectors, a vector associated with each graph node, to set edge weights based on the relationship between the edge vector and the guiding vector. MGV can effectively control the global shape and the intermediate-scale architecture of trees. The ideas of MIG and MGV are described as follows.

In MIG, we create a hierarchical tree structure through iterations. We first create a user specified graph shape, in order to control the global tree shape by restricting least-cost paths in the graph volume. In each later iteration, we create subgraphs based on the previous tree structure. Least-cost paths in subgraphs are added to the previous structure and make the tree more and more complicated. Variations of results can be accomplished by modifying parameters of the initial graph and subgraph creation processes and endpoint placement.

MIG possesses the following properties:

- It is automated. It does not require any user guidance or external data such as point cloud from scanning.
- It is moderately fast, requiring only a few seconds to generate a full tree.
- It provides effective control over the global shape of trees
- It is versatile, capable of generating a wide variety of trees.

However, the control of intermediate-scale architecture in MIG is weak. We attempt to control branch shapes by specifying the orientation and dimensions of subgraphs. Each long branch is composed by piecewise linking up short paths from individual subgraphs, making the resulting trees lack a natural unity of shape.

In MGV, we create the entire tree structure in a unified graph. Each node has an associated guiding vector. Outgoing edges have weights set according to whether the edge aligns with the guiding vector direction, thus inducing least-cost paths in the graph to conform to the guiding vector field. MGV not only possesses the above properties that MIG has, but also has more effective control over the intermediate-scale architecture of trees through specifying guiding vectors. The resulting models resemble a variety of real-world tree species.

Our work has enlarged the space of available algorithms for realistic tree modeling. The resulting tree models can be combined with other models such as terrains and clouds

to create landscapes. The idea of tree modeling is also useful to the modeling of other branching shapes such as rivers, lightning, and cracks. Moreover, since our system is built based on the idea of path planning, though we did not explore other possibilities beyond modeling, we consider the ideas in our work might also be useful to applications where least-cost paths and graphs are used, such as image processing with graph propagation, texture synthesis, and graph-based motion planning.

Unlike realistic trees, abstract trees convey visual information in a different style. For example, in Gustav Klimt's painting [22] shown in Figure 1.3, the tree has a central trunk which splits into a few secondary branches. The secondary branches develop some spirals, and each splits into more spirals. Branches do not intersect. Different sizes of spirals are arranged compactly, filling most of the image. Overall, the stylized tree structure and the arrangement of spirals show an appealing aesthetic effect.

Figure 1.3: "Tree of Life" by Gustav Klimt [22].

In Klimt's tree, stylized branches/curves are the key element that conveys a sense of art. Artistic curves were discussed by Ruskin in *"The Elements of Drawing"* [68], where he advocates using curved forms in artistic compositions, and advises a constantly changing curvature: "Graceful curvature is distinguished... by its variation, that is to say, its never remaining equal in degree at different parts of its course... a steady change through the whole line, from less to more curvature, or from more to less, so that *no* part of the line is a segment of the circle."

To create aesthetic curves that Ruskin described, we propose *magnetic curves*, a particle tracing method for creating curvature-controlled curves. Previous particle methods can generate curves but lack the ability to systematically control their curvatures. Our method can create curves with constantly changing curvature, which can be used to create abstract tree branches.

The idea of our method is as follows. A charged particle moving in a constant magnetic field leaves a circular (helical) trajectory. If we permit charge and/or field strength to vary, the particle will trace out a path whose curvature varies.

We use magnetic curves to create aesthetic abstract trees in Klimt's style. We do not intend to exactly reproduce Klimt's tree, but are interested in creating similar trees that have the following characteristics based on our observation. We also ignore other decorative elements on the tree such as circles, triangles, and mushroom shapes.

- stylized spiral branches
- a hierarchical branching tree structure
- noncrossing spirals covering most of the image

To create abstract trees with the above aspects, we continuously change the charge of particles to make them trace out the spirals for tree branches. Further, appealing branching structures can be produced in a simple way, by splitting a particle into two particles that are identical except for having charges of opposite sign. The two particles will move to different directions from that point, and their trajectories will form a branching shape. To create a hierarchical tree structure, we release new particles at intervals along the initial particle's path. Finally we create space-filling patterns using different sizes of spirals. In addition to artistic abstract trees in Klimt's art style, our method is also versatile and can generate different aesthetic forms composed of curves.

1.2 Contributions

This thesis provides procedural methods in two regimes – realistic tree modeling and abstract tree modeling. It makes the following contributions.

– It proposes using the Yao graph [93] for graph-based tree modeling. The Yao graph is a particular way of organizing points into a graph. It automatically limits the number of edges while ensuring edges in every direction. Previous graph-based methods [78, 86–88] create a graph using a lattice or by connecting points within a user-specified distance, but have their limitations. Using a lattice leads to artifacts in the resulting least-cost paths. To connect points within a distance, deciding the distance parameter is a challenge: too large, and the memory cost is high; too small, and the graph may become divided. We are the first to use the Yao graph for graph-based tree modeling.

– It proposes a graph construction method and a graph-based tree construction method with iterated graphs. Prior graph-based methods obtain the graph volume from real-world data or using a simple shape such as cube, lacking a sophisticated method to construct a graph. We create an initial graph by placing nodes in a designated volume, and create a hierarchical tree structure by placing subgraphs around each endpoint and beginning again through some number of iterations. Our graph construction method and the associated tree modeling method are new. They are more elaborate than previous graph-based tree modeling methods.

– It proposes guiding vectors for setting edge weights. Prior methods (particle tracing, self-organizing trees, graph-based trees) do not have an effective way to specify local directions of branches; architectural control is available only coarsely, through attraction point placement, through endpoint placement, or through a density map. In our method, we set the weight of an edge according to whether the edge aligns with the guiding vector direction, thus inducing least-cost paths in the graph to conform to the guiding vectors. Guiding vectors provide local control over branch orientation.

– It proposes a mechanism for assigning guiding vectors. We propose to determine the guiding vectors incrementally as the shortest-paths algorithm progresses, computing a guiding vector for a newly visited node as an incremental rotation of its parents guiding vector. Our suggested parametric assignment of guiding vector direction is flexible and controllable, while the path planning element enforces a unity over the resulting tree. We can specify intermediate-scale tree architectures with a few simple rules. This thesis demonstrates a wide range of high-quality results from variations

of guiding vectors.

– It contributes magnetic curves, a particle-tracing method to create curvature-controlled curves. Prior particle tracing methods lack systematic control over curvature of the trajectories of particles. In our method, we continuously change the charge on a simulated particle in a magnetic field so that it can trace out a complex curve with continuously varying curvature. Our method is capable of generating curvature-controlled curves that are difficult for previous particle tracing methods. It is also a potential tool for creating stylized depictions of classes of objects and phenomena that are well described by curves such as trees, hair, and conventionally-drawn cartoon flames.

– It contributes a method for abstract tree modeling. We are the first to model abstract trees in the art style of Gustav Klimt. Previous methods to create abstract trees emphasize rendering [11, 25, 35]. Prior modeling methods cannot create trees that have spiral branches in Klimt's style. Our work fills in the blank of exploring Klimt's style in computer graphics, and also enlarges the space of available algorithms for abstract tree modeling.

1.3 Organization

The thesis is organized as follows. In Chapter 2 we review the previous work in realistic tree modeling and abstract tree modeling, with an emphasis on the former. In Chapter 3, we introduce the background knowledge related to our realistic tree modeling work. In Chapter 4 and 5, we introduce MIG and MGV. In Chapter 6, we show results with evaluations, and discuss the pros and cons of MIG and MGV. In Chapter 7, we introduce the method of magnetic curves for abstract tree modeling. In Chapter 8, we conclude the thesis and describe future work.

Chapter 2

Previous Work

Generating virtual complexity is a major task in computer graphics [70]. Techniques of modeling, texturing, rendering, and animation are important tools to achieve the ingredients that evoke visual complexity in computer-generated virtual worlds such as shape, texture, and motion. Static or animated natural scenes containing complex shapes such as mountains, clouds, and trees play an important role in many computer graphics applications such as 3D movies and computer games.

In the pipeline of creating the natural scenes, our work falls in the early stage – modeling. Modeling natural phenomena is a topic of great interest in computer graphics. In this thesis, we focus on trees, a commonly seen natural phenomenon. They have been a particular source of fascination, with numerous automatic and semi-automatic methods for tree modeling proposed over more than thirty years. In this chapter, we review related work on tree modeling, with a focus on notable procedural methods. We discuss some commonly used realistic tree modeling methods by classifying them into the following categories according to the principles they follow: L-systems, geometric methods, graph-based methods, data-driven methods, biologically-based methods, sketch-based methods, and tree editing methods.

L-systems, geometric methods, and graph-based methods are focused on the visual simulation of tree structures, based on recursive and hierarchical branching patterns. L-systems are a grammar that generates strings which are subsequently interpreted as branching structures such as trees. Geometric methods create tree structures through recursion, generating tree models by explicitly varying geometric quantities at each level such as splitting of stems and branching angles. Graph-based methods create trees with least-cost paths in a weighted graph. The collection of paths forms a hierarchical tree structure. For L-systems, geometric methods, and graph-based methods, to effectively control the global tree shape and branch shapes is a challenge.

Biologically-based methods create natural tree shapes with a biologically plausible process. A tree structure is formed in a bottom-up way motivated by competition for light or space. It is a challenge to generate a desired tree shape using the unfamiliar botanical meaningful parameters.

Data-driven methods are based on real-world measurements such as images or laser scans of trees, with an intention to reconstruct existing trees. They are also often combined with sketch-based methods, where sketches are used to select tree elements from a database.

Sketch-based methods provide direct control to users: users sketch strokes to guide where tree branches are to grow. The branches are generated with other procedural methods such as L-systems, or biologically-based methods, or come from a database. Sketch-based methods depend heavily on user intervention.

Tree editing methods emphasize creating variations of input tree models. Trees can deform with the influences from environmental factors such as wind, obstacles, and lighting changes. These methods are not intended to create tree models from scratch.

Some of the above categories may overlap due to multiple techniques used in the methods. Details of the methods are discussed in the following sections.

2.1 L-systems

Lindenmayer systems (L-systems) are a theoretical framework widely used for modeling branching structures. The basic idea of L-systems is that of rewriting: a complex object can be created "by successively replacing parts of a simple initial object using a set of rewriting rules or productions" [62].

The rewriting of L-systems builds on strings. Each symbol in an initial string is associated with a rewriting rule. Starting from the initial string, the rules are applied simultaneously through iterations. The resulting string can be translated to a geometric structure using a turtle interpretation [57,60,62], in which a symbol represents a graphical operation. An example of turtle interpretation is shown as follows.

F Move forward a step of length d

$+$ Turn left by angle δ

$-$ Turn right by angle δ

[Push the current state onto a pushdown stack

] Pop the current state onto a pushdown stack

A structure generated by L-systems using the above turtle interpretation is shown in Figure 2.1. The rewriting process starts from an initial string F and applies a few iterations of replacements using the rule: $F \rightarrow F[+F]F[-F]F$. In the right side of the rule, $[+F]$ creates a left side branch that splits from where the state is pushed and popped. Similarly, $[-F]$ creates a right side branch. The first iteration creates a structure that has a central trunk and two side branches, shown in the middle of Figure 2.1. In later iterations, a more and more complicated structure is created by replacing F in the previous string. The right shape in Figure 2.1 is a tree-like structure obtained after $n = 5$ iterations of replacements.

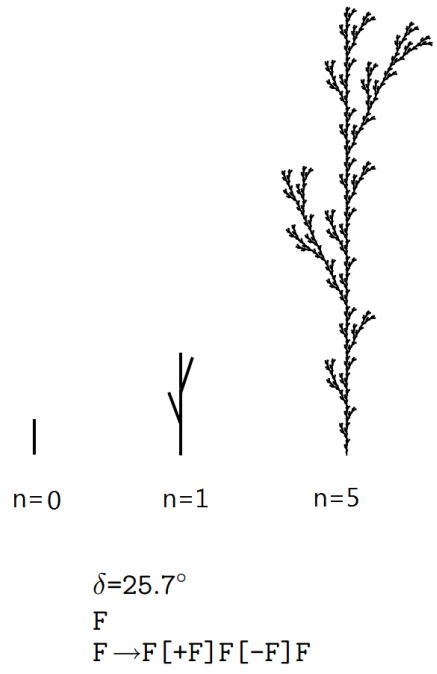


Figure 2.1: Structures generated by iterations of rewriting. The example is taken from "The Geometric Beauty of Plants" [62].

Early L-systems were applied to simulate cell development [28, 29]. Later extensions were applied to model growth of plants [36, 61] and even entire ecosystems [10]. Stochastic L-systems [14, 59] introduce the idea of using production probabilities. A string can be replaced using different rewriting rules depending on probabilities, leading to variations

of results. Environmentally-sensitive L-systems [61] involve query symbols, which can return environmental information, such as local position and orientation in space, to affect the future development. To model the development of plants and plant ecosystems, modeling bi-directional information exchange between trees and their environment is important. Open L-system [36] was devised, in which a communication symbol is imported in the grammar. An occurrence of the communication symbol leads to an information exchange. With this method, a large ecosystem can be created by involving competition between plants for space, light, nutrients, and water.

One issue with L-systems is the difficulty of global control. It is a challenge to devise a system of rules to obtain a particular tree shape. Prusinkiewicz et al. [61] proposed the synthetic topiary, in which a user-specified bounding shape restricts the plant growth. Branches growing outside of the bounding volume are pruned off. Power et al. [56] and Boudon et al. [4] interactively control the plant model by pruning and bending branches between steps of the generation process, to achieve more precise control on the resulting structures. Both their methods need modeler intervention. Ijiri et al. [20] proposed a sketch based L-system. In their system, the growth direction of a tree is guided by a user-drawn stroke, and the branches are created with predefined replacement rules. This method provides control over growth directions of branches rather than global tree shapes.

Talton et al. [76] presented a method to iteratively optimize over the space of possible productions from a given grammar according to an input specification of a sketch or a volumetric shape, in order to generate a structure that matches the specified shape. The method has good control over the global shape of trees, but the optimization takes thousands of iterations and can take hours. Another modification of L-systems is the guided procedural modeling method proposed by Benes et al. [2]. In their method, space is divided into guides, which are geometric volumes for branch growth. Each guide has a procedural model and can interact with other guides. Users can control the overall tree shape by editing guides or adjust small-scale features with procedural rules. Since the method is based on open L-system, it still does not eliminate the need of manually writing the L-system rewriting rules.

2.2 Geometric methods

Geometric methods generate models with visual approximations of tree structures. Users can create different tree models by changing parameters that represent geometric properties.

Honda [19] presented a recursive method for tree modeling, in which a tree structure is made up of repeated locally-defined bifurcations of branches. By varying parameters – angles between parent branches and their child branches, and the relative ratio of branch lengths – the method can create various tree structures. Based on Honda's work, Aono and Kunii [1] proposed a series of geometric models that describes tree geometries more precisely. Their system provides parameters to control the arrangement of branches or leaves on the stem, how clearly the main axis can be detected, and apical dominance that decides the contour shapes. The parameters can be set according to statistical data of real tree structures. Oppenheimer et al. [50] build tree structures based on the recursive pattern as well. The process begins with a tree node, which is defined as a branch with one or more tree nodes attached, and augments it by replacing tree nodes with transformed branches through iterations. Their model contains parameters similar to those in Aono and Kunii's method [1], but involves randomness by adding some perturbation to parameters in order to create irregular gnarled trees. With an idea similar to Oppenheimer et al.'s method, the method of Weber and Penn [81] is more detailed. In their method, the tree structure is treated as a primary trunk in segments. A branch can split off cloned and transformed segments along its length. A hierarchical tree can be created by extending and splitting branches through levels of recursion. To make the resulting trees fit inside a specific volume, they tentatively set the lengths of segments and keep adjusting them until they fit.

Based on observations or statistical investigations of the external appearances of trees, geometric methods focus on visual simulation of tree structures rather than following internal physical or botanical mechanisms. However, to create a model, users still have to understand and control many parameters. For example, creating a new tree using the method of Weber and Penn [81] requires adjustment of up to 80 parameters. Lack of effective and direct global control is also an issue.

2.3 Data-driven methods

Data-driven methods reconstruct existing trees instead of modeling trees from scratch. The data is obtained from photos or laser scans of existing trees.

Image-based methods use multiple images [48, 63, 64, 78] or a single image [17, 77] for realistic tree modeling. Methods using multiple images find point constraints for 3D reconstruction with a sequence of images taken from controlled camera positions. Reche et al. [64] proposed a method to reconstruct and render trees. The tree is treated as a volume with opacity and color values, computed from a small set of calibrated images around a tree. Their method achieves realistic results from rendering, but is not able to discover the explicit geometry of branching structures. In the method of Neubert et al. [48], input photos are used to create an approximate voxel-based tree volume. Particles are placed randomly in the voxels. The tree skeleton is extracted from images or sketched by users. Under the attraction from tree skeleton, particles move downward to the tree root and their trajectories form a branching structure. Tan et al. [78] recover a 3D point cloud of the photographed tree from a set of images. Their method creates a graph based on the 3D points. The shortest paths from the root point to other points are divided into segments. Centroids of points in each segment are linked up, forming a tree skeleton. Occluded branches are created using subtrees of the recovered visible branches. In a follow-up, Tan et al. [77] proposed a method in which users sketch the main trunk and the crown region on a single tree image. The system traces along the trunk stroke from the root to extract visible branches. The branch strokes are then converted to 3D through greedy adjustments of their orientations by maximizing distances among branches. The entire tree is created by augmenting the tree skeleton with sub-trees from visible branches or from predefined patterns.

With advances of scanning techniques, laser-scanned data of tree geometry are used to facilitate reconstruction of real trees. Xu et al. [86] proposed a method similar to Tan et al.'s method [78], but using scanned points to create a graph. Their work is focused on synthesizing new branches based on sparse point clouds. They used least-cost paths to produce the tree's main skeleton. New branches are created by scaling and transforming existing branches. Their method is capable of generating tree models with plausible details not present in the scan. Bucksch et al. [5] approached the problem similarly, but created a

graph based on an octree to organize the point cloud data. Their method can precisely reconstruct a tree without leaves. Instead of generating a single tree, Livny et al. [31] sought to reconstruct multiple overlapping trees simultaneously. They applied a series of global optimizations based on biologically-derived heuristics to fit tree skeletal structures to the point data. Their method is automatic and does not require user intervention. In order to simplify the reconstruction process, some methods approximate a tree's geometry by dividing the scanned points into a few parts, and create branches in each part. In Livny et al.'s method [30], the tree's foliage is abstracted into canonical geometry parts called lobes. In each lobe, branches are generated using L-systems and rearranged to fit inside the lobe. Inspired by Livny et al.'s method, Zhang et al. [96] divide scanned points into multiple layers for main branches, secondary branches, tertiary branches, twigs, and leaves. A hierarchical particle tracing algorithm is applied to create branches in each layer, obtaining the tree model consistent with the scan data. This method is an extension of single-layer particle tracing method of Neubert et al. [48] but has better control over tree structures. Zhang et al.'s method [96] and Livny et al.'s method [30] can create realistic models that are generally faithful to the original scanned trees without caring about enormous amounts of local details. Rather than reconstructing static tree models, Li et al. [27] are more interested in the dynamic development of trees. They use scanning to acquire 4D data of tree geometry over time. Their method can model plant growth process by detecting and analyzing the growth events such as budding.

Since data-driven methods are based on real-world measurements, they can generate high-quality tree models. However, the requirement of existing trees also limits their application: it can be inconvenient to obtain laser scans or photographs of a desired tree, and also not easy to design new trees without manual editing. Different from data-driven methods, our methods in this thesis are aimed at generating tree models from scratch, and do not require any image or point cloud as input.

2.4 Biologically-based methods

Biologically-based methods are proposed for the purpose of generating realistic tree models in a process that follows botanical rules. The resulting structures are controlled by inner motivators or environmental factors.

Reffye et al. [9] simulate the tree growth by modeling the activity of buds at discretized times. The growth of a tree is characterized with probabilities of growth, ramification, and death. The associated parameters are calculated according to botanical laws for specific species of trees. Their method can create tree models that are faithful to botanical structures and development. Runions et al. [67] proposed a space colonization method, in which attraction points are used to signal the availability of growth space. The tree extends towards nearby attraction points, which are removed when the space is no longer empty. The tree structure is formed gradually from the root towards the attraction points. As a later extension, the method of self-organizing tree modeling [52] focuses on competition between branches. The resulting tree shapes can be controlled by constraining the space for tree growth and properly distributing the initial attraction points. Different from these bottom-up growth methods, the method of Wang et al. [80] uses a top-down method to create a tree that matches a given guidance shape. The method starts from a structure defined by botanically meaningful parameters. Sample nodes distributed uniformly on the surface of guidance shape are directly connected to the initial structure. The obtained connected structure is iteratively adjusted by minimizing the difference from the guidance shape until no visible change happens or a fixed iteration number is reached. This method is intended to control global shapes rather than branch shapes.

This group of methods can create realistic tree models and rich variations, due to the faithfulness to the botanical nature of trees. They also have effective control over the global shape of results. However, to create detailed branching structures, users need to have background knowledge to understand the botanical meaningful parameters in order to control the results. In this thesis, we aim at creating visually realistic trees instead of pursuing their botanical correctness. The concrete parameters in our system simplify parameter selection for generating novel tree types.

2.5 Sketch-based methods

Full manual creation of trees is exhausting, but sketch-based methods reduce the needed user effort by providing a direct and flexible way for tree modeling. In sketch-based methods, user-sketched strokes are used to guide the tree growth directions or to infer branch shapes.

Sketch L-system [20] is the first attempt to interactively control the tree growth in L-systems using strokes as input, based on which the system gradually generates a tree that grows along the strokes. Unfortunately, the system only includes few simple rules, leading to limited tree structures. TreeSketch [34] is an extension of self-organizing tree modeling [52] with a tablet interface. Users directly sketch strokes with a procedural brush, which is used to distribute attraction points within a sphere invoked and manipulated by users. Branches are generated using the space colonization algorithm in a bottom-up way towards attraction points. Users can control the global shape of trees with sketches and edit parameters to change the intermediate-scale architecture.

Sketching and data-driven approaches can be combined, with sketches used to facilitate the inference of tree structures or the extraction of tree information from a database. Okabe et al. [49] convert 2D freehand sketches to 3D branches under the assumption that branches maximize the space around them. New branches are added using existing branches as examples. This method allows novice users to create natural-looking tree models interactively and quickly. Chen et al. [7] infer a tree template from a database based on user-supplied strokes, and then augment it by adding more branches based on similarity to the existing branches. Their method requires a database in order to reduce user intervention. Users can create a realistic tree with only a few strokes. Wither et al. [84] combine user sketches of foliage with botanical heuristics to determine branch connections and distributions. Their method can generate realistic tree models with flexible user controls.

The strengths of sketch-based methods lie in effective and flexible user control. Various results can be generated with the guide from user sketched strokes. However, these methods depend heavily on user input; creating a complicated tree structure or many trees may require a lot of user work. Compared with sketch-based methods, procedural methods have their benefits. Due to the ability of data amplification, procedural methods can create a complicated tree structure once parameters are known. A wide variety of new trees can be automatically created by adjusting parameter settings. Many instances of similar models can be generated with the same parameters, requiring zero human effort. A limitation of procedural methods is that setting parameters to create a desired tree shape is not easy. In this thesis, we propose a procedural method, MGV, to create a wide variety of tree models. It requires no user intervention beyond specifying parameters. The parameters

are geometrically meaningful and the parameter space is smooth, making it less difficult to create a desired tree shape compared with other procedural methods such as L-systems and biologically-based methods.

2.6 Tree editing

With an intention to create variations of trees, tree editing methods analyze existing tree models which are imported as input or generated using a customized modeling system. The tree models are converted into graph-based representations and altered according to environmental influences, such as external obstacles and wind.

Pirk et al. [55] proposed a method that converts an input tree model into a skeletal graph and a set of leaf clusters. With obstacles and lighting changes, the branching skeleton is bent and pruned, and the leaf clusters are deformed, leading to variations of results. Later the idea was extended to model trees under wind effects [54]. In this method, sensor particles are distributed along the tree structure, and each particle is surrounded by a sensor volume to quantify the passing wind particles in order to measure wind forces. The tree graph transforms according to the integration of wind forces and other forces from the internal structure and leaves, producing various tree shapes. To simulate natural growth of trees, Pirk et al. [53] analyze a given static tree model and determine a graph-based description of tree structure. Branches are added or pruned off based on similarity to existing branches, used to generate structures at different growth stages.

The above tree editing systems work well with models from different sources, but are not capable of generating new tree models. To address the problem, Stava et al. [73] aim to determine parameters for a given stochastic procedural model, in order to use the parameters to generate new trees that are similar to the model. Their method automatically determines optimal parameters of a given input tree using Monte Carlo Markov Chains, by gradually increasing the similarity of the generated trees to the input tree. The method is also capable of generating tree models without input by directly setting the botanically meaningful parameters, in which case it is a biologically-based method.

The emphasis of tree editing methods is different from our method while our interest is tree synthesis from scratch. We are more concerned about the capability of a method to generate different species of tree shapes. The output models from our method can be used

as input of tree editing methods.

2.7 Graph-based methods

Graph-based methods refer to those using path planning to create tree structures. The idea of using least-cost paths to create/extract tree structures has been used in some previous work.

Xu et al. [86] build a graph based on 3D points of a tree skeleton. In the graph, edges connect two nodes within a user-specified distance, and the weight of each edge is set to the length of the edge. They compute shortest paths connecting all points to the root. The shortest paths are segmented into bins according to path lengths. The centroids of points in the bins are linked to form a tree skeleton. Tan et al. [78] model trees based on the similar idea of path planning, though they extract the graph points from a set of images while Xu et al. [86] get the points directly from scanning. Tan et al.'s method and Xu et al.'s method are both data-driven methods with the objective of tree reconstruction, which is different from our pursuit of generating new trees.

Prior to this thesis work, we have explored using path planning for modeling natural phenomena such as lightning and trees, by proposing the basic constructive method (BCM) [87, 88]. In BCM, the graph is procedural and not obtained from real-world data. We compute least-cost paths between multiple endpoint nodes and a single root node in a randomly weighted graph. The collection of paths forms a branching structure. The global shape can be controlled by endpoints placed manually or randomly selected from a simple geometric shape such as a circle. We generate a hierarchical structure through iterations. In each iteration we cheapen the costs of previous paths, and select endpoints in the vicinity of the existing structure. By successively creating least-cost paths to the newly added endpoints, we can generate a complicated branching structure, which can be used to model some natural phenomena such as coral, lichen, and trees. BCM provides good control over the global shape of the results, but the details of the branch shapes were unconvincing.

The realistic tree modeling work introduced in this thesis shares the basic path planning idea with BCM, but has more effective control over the global shape and the intermediate-scale architecture of trees. More details of the differences between BCM and our thesis work will be discussed in section 3.3.

2.8 Abstract tree modeling

Computer-generated abstract forms have had a long history in computer graphics [3, 12, 24, 85, 94]. Abstract trees convey visual information different from realistic trees, with stylized branch shapes, artistic depiction of silhouette, and aesthetic patterns of branches and leaves. Existing methods for abstract trees fall into two regimes – rendering and modeling.

Most methods of creating abstract trees are in the regime of non-photorealistic rendering (NPR). Users can sketch a stylized tree using computer simulated brushes [69, 75, 95], or render existing tree models from input or generated in an embedded existing modeling mechanism. Kowalski et al. [25] render 3D trees by procedurally adding stroke-based texture near tree silhouettes in a view-dependent way. Their method can produce stylized trees evocative of a sense of complexity. Deussen et al. [11] proposed a method to illustrate an input 3D tree model in pen-and-ink styles. A tree is rendered with silhouette lines and cross hatching. Each leaf is represented by a size-controllable disk or polygon. This method is capable of generating a variety of trees in illustration styles. Similarly, Kalnins et al. [23] also render the silhouettes of plant forms using strokes. However the rendering effect is simple due to the use of same shape of strokes. Their actual focus is creating frame-to-frame coherence in animations. Luft et al. [35] are interested in the rendering of foliage. They place a set of points based on foliage geometry to create implicit surfaces. The visibility of leaves in the tree crown is controlled by the general shape of implicit surface. Their method can produce trees in different styles such as watercolor, but the tree structures and branch shapes are regular due to the use of realistic tree models.

Some methods generate stylized trees from the philosophy of modeling. Strothotte et al. [74] create stylized trees based on L-systems. Users can change parameters to add their specified stylized leaves or branches. However, controlling branch shapes is not easy. Long and Mould [33] proposed a dendritic stylization method using path planning. In the computing of least-cost paths, comparisons between two paths are made first by comparing their maximum edge cost values, or the cumulative distances in case of a tie. The resulting long and winding paths are suitable for artistic effects. To create a stylized tree, they used an input image to set edge weights and for endpoint selection. The least-cost paths from a root to selected endpoints form a tree. Their method is capable of generating stylized trees that conveys the structure of the input image.

In this thesis, we create abstract trees by modeling the branching and hierarchical structures. Our objective is to create trees that have similar features in Klimt's painting – spiral branches, a stylized hierarchical structure, and aesthetic branching patterns. Our work is the first exploration of Gustav Klimt's art style in computer graphics.

Chapter 3

Background

Our realistic tree modeling work depends on least-cost paths. The thesis work shares the philosophy of using least-cost paths with our previous basic constructive method (BCM) [87, 88]. This chapter will introduce background knowledge related to least-cost paths: path planning and the Yao graph. Path planning is about how to find least-cost paths in a graph. The Yao graph [93] is about how to organize points into a graph where least-cost paths exist. We will also introduce BCM and the distinction of the thesis work.

3.1 Path Planning

Path planning, as defined by Shih [71], is the problem of "finding paths connecting different locations in an environment (e.g., a network, a graph, or a geometric space)". The desired paths need to satisfy some criteria such as avoiding obstacles. A common task of path planning is computing the least-cost path between two nodes in a weighted graph [37]; the cost of a path is the sum of the weights of its constituent edges.

Dijkstra's algorithm [13] is an algorithm to compute the least-cost paths from a source node to other nodes in a weighted graph. Each node in the graph has a cost, which is the cost of the path from the source to the node. Initially the source node is assigned cost 0, and other nodes have an infinite cost. Dijkstra's algorithm proceeds from the source node in a greedy way until every graph node has its least cost and path computed. The description of Dijkstra's algorithm given by Lee and Hubbard [26] is reproduced in Figure 3.1. In this process, there are two sets of graph nodes that the algorithm maintains: an unvisited set and a visited set. The unvisited set is vertices to be considered when computing least-cost paths. The visited set is vertices that have their least cost computed. At the beginning, the source vertex is first visited. Then the algorithm keeps visiting the unvisited vertices adjacent to the visited vertices and updating the two sets, until all vertices are visited. At every visit, the cost to reach a vertex is calculated and compared with the vertex's current

cost. The vertex always takes the cheaper cost and remembers the corresponding previous vertex. When the algorithm terminates, every vertex has its minimum cost computed, and the least-cost path can be determined by keeping track of the previous vertex backwards towards the source.

Initially the source vertex, with its cost of 0, is added to the unvisited set. Then the algorithm proceeds as follows as long as there is at least one vertex in the unvisited set.

1. Remove the vertex we'll call *current* from the *unvisited* set with the least cost. All other paths to this vertex must have greater cost because otherwise they would have been in the *unvisited* set with smaller cost.
2. Add current to the *visited* set.
3. For every vertex, *adjacent*, that is adjacent to current, check to see if *adjacent* is in the *visited* set or not. If *adjacent* is in the *visited set*, then we know the minimum cost of reaching this *vertex* from the source so don't do anything.
4. If *adjacent* is not in the *visited* set, compute a new cost for arriving at *adjacent* by traversing the edge, *e*, from *current* to *adjacent*. A new cost can be found by adding the cost of getting to *current* and *e*'s weight. If this new cost is better than the current cost of getting to *adjacent*, then update *adjacent*'s cost and remember that *current* is the previous vertex of *adjacent*. Also, add *adjacent* to the *unvisited* set.

Figure 3.1: The process of Dijkstra's algorithm as described by Lee and Hubbard [26].

In this thesis, we use Dijkstra's algorithm to create least-cost paths in a graph. The least-cost paths are tree branches. Next we introduce the Yao graph, which we use to create our graph.

3.2 The Yao Graph

In our work, we desire the following properties of a graph for realistic tree modeling. First, in order to avoid lattice artifacts in the least-cost paths such as in the upper example in Figure 3.3, we want edges from all directions and without specific preferred directions.

Second, the graph should be connected, to guarantee that the paths can connect the root point and selected endpoints. Third, the graph has a bounded outdegree, for the purpose of making memory usage manageable. It also simplifies the data structure of each graph node to store outgoing edges.

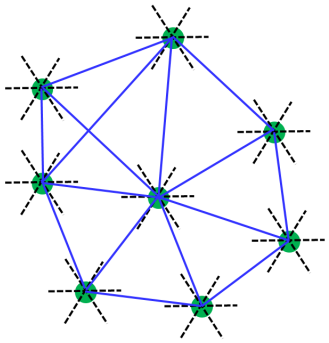


Figure 3.2: A Yao graph with 6 sectors.

In order to avoid lattice artifacts, we place graph nodes according to the Poisson disc distribution [16], where nodes are randomly distributed and no closer to each other than a specified minimum distance. The Poisson disc distribution of nodes helps to get edges from all directions.

Based on the nodes, we build a Yao graph [93]. The Yao graph is a graph in which the space around each node v is equally separated into a few sectors, and edges connect v and its nearest neighbor in each sector. Figure 3.2 illustrates a Yao graph in 2D. There may be more than one edge per sector for a given node: in the example, each node originates at most 6 edges, but may receive additional edges originating from other nodes, such as the central node receiving an edge from the top left node.

The Yao graph automatically limits the number of edges while ensuring edges in every direction. The upper of Figure 3.4 shows the distribution of edge directions in a Yao graph. In the graph, edges are almost evenly distributed in every direction. This property will avoid the lattice artifacts in the least-cost paths due to the limited edge directions. A least-cost path in a Yao graph is shown in the lower of Figure 3.3. It does not have the staircasing problem in the upper path from a lattice graph which has only horizontal and vertical edges.

In our tree modeling work, our graph is a Yao-8 graph in 3D, where 8 is the outdegree of each node. We use the Yao-8 graph because dividing the space around each node into 8 sectors is straightforward to implement in 3D, and it also possesses our desired properties:

it gives a connected graph, and also has a bounded outdegree that is not too high. We do not have a strict restriction about the upper bound of outdegree, but a higher outdegree, say 20, will increase the burden on path planning and memory usage.

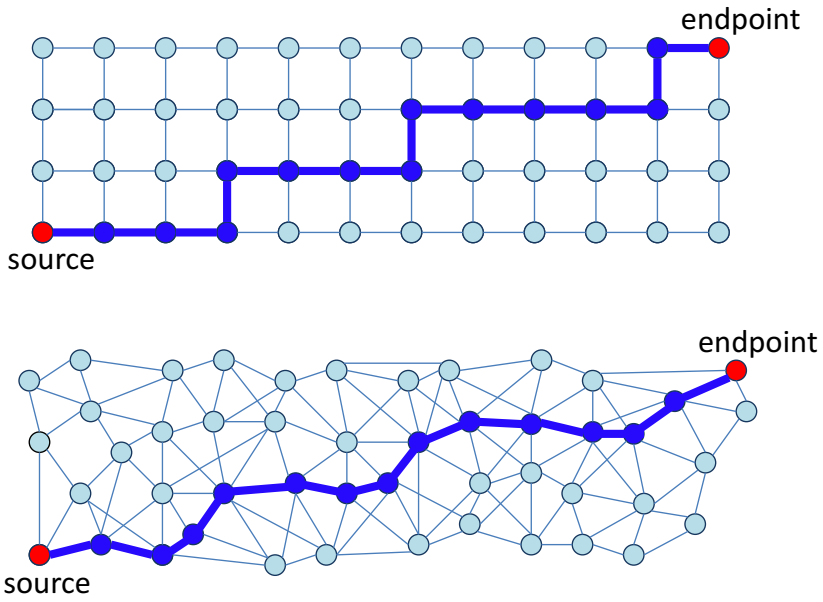


Figure 3.3: Upper: a path in a regular graph of a lattice; lower: a path in a Yao graph.

In addition to the Yao graph, we have considered other options to create a graph such as using a lattice, connecting nodes within a constant distance, and the Delaunay triangulation. We are concerned about whether these options can achieve our desired graph properties, as discussed in the following.

The first option that we have explored is using a lattice graph. In BCM [87,88], we used a regular lattice (4-connected in 2D) or grid (6-connected in 3D) to create a graph. The resulting paths have lattice artifacts. In addition to the problem of lattice artifacts, if we use a lattice graph in MGV, we will have another problem: due to the limited edge directions in the graph, the least-cost paths will not closely follow the guiding vector directions.

For the above two problems, we considered two possible solutions. One possible solution is to apply antialiasing methods such as smoothing to the resulting least-cost paths from a lattice graph. Smoothing can effectively remove the lattice artifacts, but at the same time it may also weaken or even remove irregularity of the paths, changing our desired branch shapes. The other possible solution is to jitter the lattice. This method can make

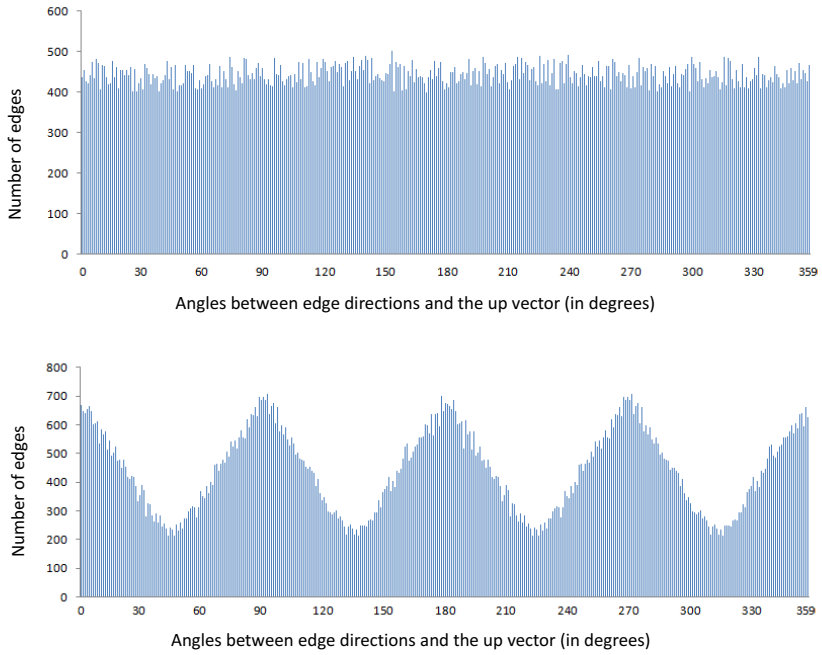


Figure 3.4: The distribution of edge directions in a Yao graph built from a Poisson disc distribution of nodes (upper) and in a jittered lattice (lower).

the graph more irregular and remove the lattice artifacts to some extent. However, a simple jittering method cannot guarantee an unbiased distribution of edge directions. For example, in a lattice jittered by 1/2 of the lattice spacing, edges are concentrated in certain directions, as shown in the lower of Figure 3.4. Despite the above deficiencies, the most important reason for not using these two possible solutions is that, neither jittering the lattice graph nor antialiasing the resulting paths can make the least-cost paths closely follow the guiding vectors as desired.

The second option to create a graph is to connect nodes within a constant distance. This method has been used by Xu et al. [86] and Tan et al. [78]. They created a graph based on point clouds from scanning. An edge connects two nodes if their distance is smaller than a threshold. However, choosing the threshold is problematic: too large, and the number of edges per node is excessive; too small, and the graph can become disconnected. In a disconnected graph, it is possible that there is no path connecting a selected endpoint and the root node, leading to a failure of branch creation. Examples of both cases are shown in Figure 3.5.

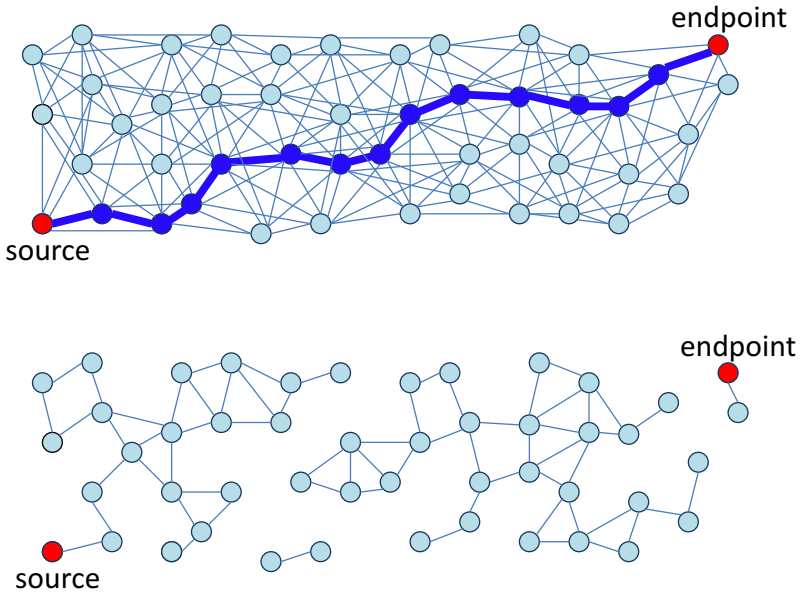


Figure 3.5: Top: a graph with a big threshold for edge connection; lower: a disconnected graph from a small distance threshold.

Another option to create a graph is the Delaunay triangulation [8]. The Delaunay triangulation is the dual of the Voronoi diagram [51]. The Voronoi diagram of a set of nodes is a partitioning of space into cells, where each cell is corresponding to a node, and all the points in the cell is closer to the node than to any other. The Delaunay triangulation can be obtained by connecting two adjacent nodes in the Voronoi diagram. We do not use the Delaunay triangulation because it does not have a bounded outdegree.

There may also be other options to create a graph. We use the Yao graph because it is our initial decision and it works. We do not strongly feel it necessary to seek other options. Of course, other alternatives that can create a graph possessing our desired properties would also work.

3.3 BCM and the thesis work

The basic construction method (BCM) [87, 88] is our initial attempt of using least-cost paths for modeling purposes. The basic idea of BCM is that, in a weighted graph, the least-cost paths between selected endpoints and a root point form a branching structure. The framework of the idea, illustrated in Figure 3.6, contains the following modules: organizing

points into a graph, setting edge weights, placing endpoints, and placing a root point. We typically place the root point at the bottom center of the graph, and care more about the other three modules, stated below.

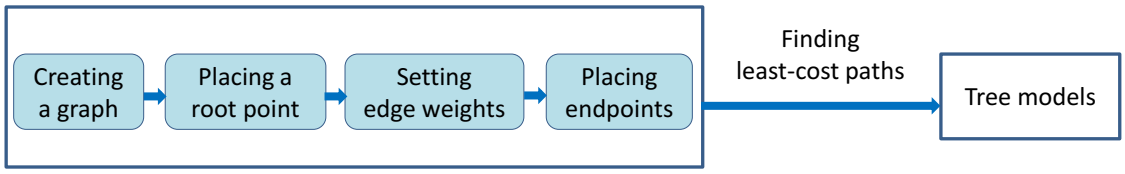


Figure 3.6: The framework of using least-cost paths for tree modeling.

To create a graph, we first place nodes in a graph volume, and build edges. The graph provides a discrete representation of the space within which the tree will reside.

We set weights to edges. The edge weights affect the shape of least-cost paths in the graph.

We select a few nodes in the graph as endpoints if they meet some criteria. Since the least-cost paths connect the root and the endpoints, the position of an endpoint in the graph decides where we want the path to extend from the root. The placement of endpoints affects the spatial distribution of paths and the resulting tree structure.

BCM gives a superficial design of the above modules. The overall resulting structures are potentially useful for modeling some branching shapes such as coral, lightning, and trees. However, the design in BCM is very shallow, lacking a specific modeling target.

In our thesis work, in order to make least-cost paths fit in the slot of tree modeling, we provide a sophisticated design of the three modules – creating a graph, setting edge weights, and placing endpoints. Our design has clear motivations. We pursue the following specific properties in the path planning framework: 1) control over the global shape of trees, and 2) control over the intermediate-scale architecture of trees. Our design is capable of generating a wide variety of trees. The tree models created in the thesis work resemble real-world tree species and have better quality than the previous results from BCM.

Next we discuss BCM along with the thesis work, with respect to the design of the three modules. Although BCM is not part of the thesis, we introduce it for the purpose of distinguishing the thesis work.

3.3.1 Design decisions of BCM

In BCM, the graph is a lattice. A lattice is a simple way to organize points into a graph. The predictable number of edges in the graph is a benefit. The drawback is the lattice artifacts in the resulting least-cost paths.

BCM varies edge weights in order to obtain some interesting resulting path shapes. We set edge weights through spatial variation or global variation. In the spatial variation we set the weight of an edge according to the edge's spatial information, in order to make paths bend toward the region where edges are cheaper. In the global variation we vary edge weight distribution globally, using a parameter to control the amount of path variation. Details of the two variation methods are described in the following.

In the spatial variation, we set the weight of each edge according to the spatial information of the edge, such as the height of the edge's midpoint or the distance from the edge's midpoint to the graph center. Examples are shown in Figure 3.7. From left to right, in each graph we set the weight of each edge to the edge's horizontal distance to the left, or the edge's horizontal distance to the right, or the edge's horizontal distance to the central axis, or the edge's vertical distance to the bottom. The resulting structures demonstrate corresponding tropisms towards the left, the right, the center, or the bottom.

Figure 3.7: Different structures obtained by different settings of edge weights.

In the global variation, we set edge weights to $1+v^{\beta}$, where v is a random value between $(0,R)$. With small R the resulting paths are close to Manhattan paths since the constant term dominates, while with larger R the paths are more erratic since the random component is relatively more important. A $R=100$ gives reasonable irregular paths in our experiments. As the value of β increases, the disparity between the cheapest and most expensive edges becomes greater, and therefore, the path planner has more incentive to seek cheap edges. It is no longer profitable to seek short cuts through an expensive edge if many cheaper edges

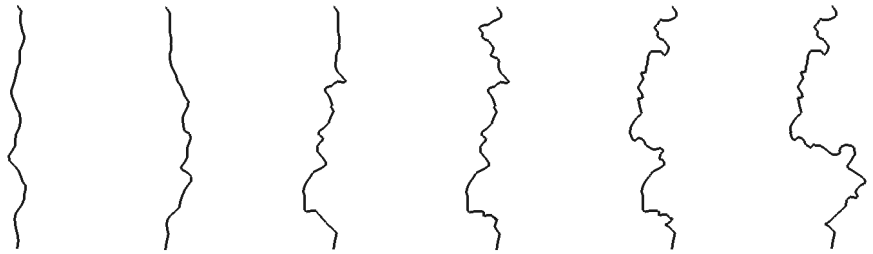


Figure 3.8: Paths with edge weights $W = 1 + v^{\beta}$, v is a random value between 1 and 100. From left to right: β = 0.0, 0.5, 1.0, 1.5, 2.0, 3.0.

could be used instead. Higher β will result in structures with longer, more roundabout paths through the graph than the structures made with lower β. The difference is illustrated by the structures in Figure 3.8. This parameter can globally control the irregularity of branches.

In BCM, we select the endpoints in the first iteration in a geometric bounding volume such as a sphere, in order to control the global shape of the resulting structure; we select endpoints in later iterations around the existing paths, in order to augment the existing structure with more branches and make the tree grow outward.

Figure 3.9 shows a visualization of the process to create a hierarchical branching structure. The root/source node is at the graph center. In the first iteration, we create a basic branching structure with manually selected endpoints. In later iterations, new endpoints are repeatedly and automatically selected in the vicinity of existing paths. We cheapen the cost of existing paths, in order to make the new paths attach to the existing paths. The resulting structure has a neat hierarchy with clusters of branches and twigs.

Another example is shown in Figure 3.10. The process includes two iterations. We place the root at the bottom center of the graph. The endpoints of the first iteration are chosen manually and tentatively, with trials to verify that they can create long and disjoint paths for primary branches. If a path merges with other existing paths too early, we discard its endpoint and select a new endpoint by hand. In the second iteration, endpoints are selected automatically if they meet both the following criteria: 1) their spatial distance to the primary branches is less than a user-specified distance value, and 2) their distance to the root is greater than a user-specified threshold. The purpose of the first criteria is to create side branches distributed around the primary branches and also to roughly control the lengths

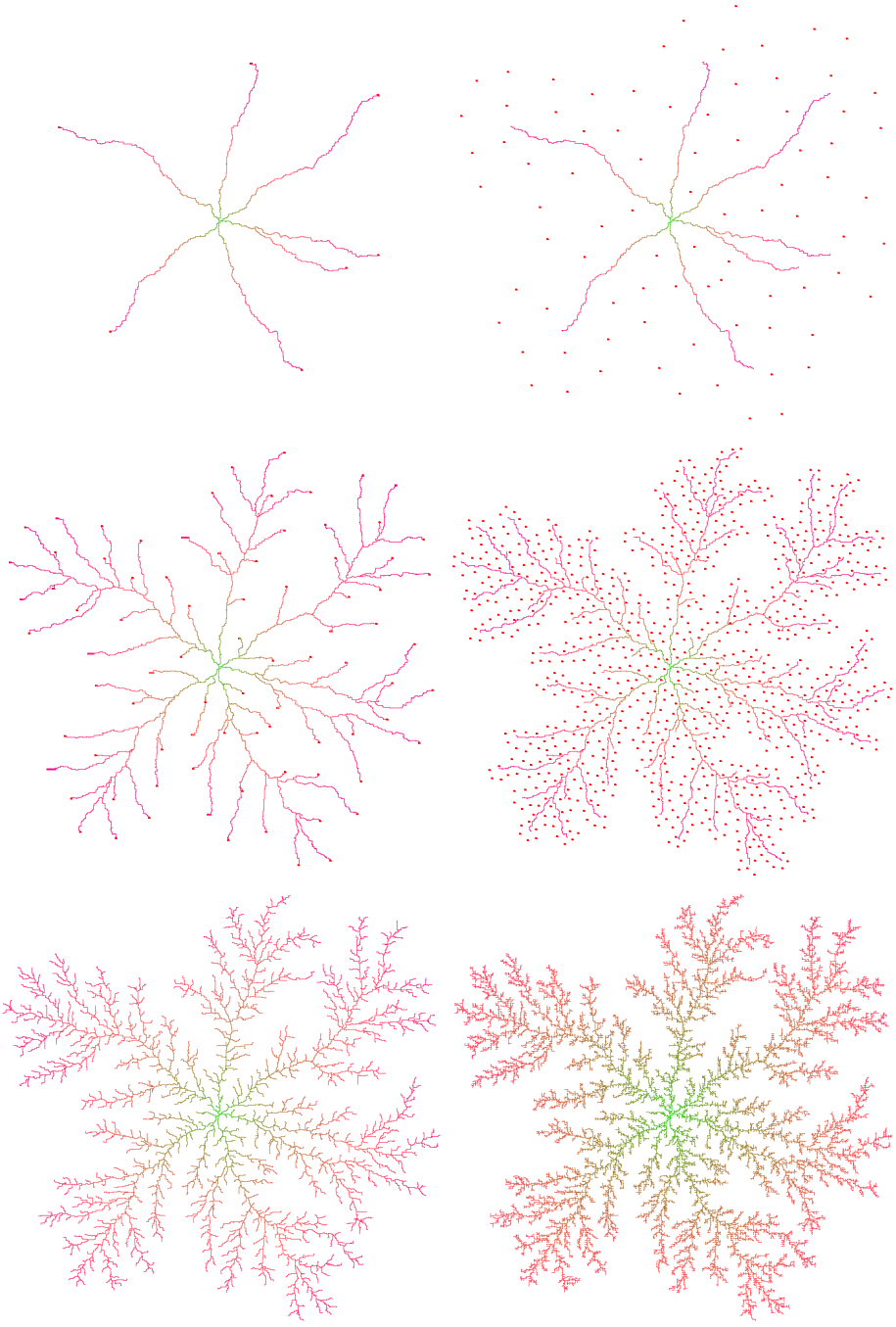


Figure 3.9: The process to create a hierarchical structure with four iterations [87].

Figure 3.10: An elm tree model from two iterations [88].

of side branches. The purpose of the second criteria is to make side branches concentrate in the top part of the tree. The model in Figure 3.10 only has a basic branching structure, lacking details such as twigs. All branches point upward, lacking the heterogeneity of real trees.

Although BCM is capable of generating some interesting structures, it has limitations as well. BCM uses a lattice graph, which leads to artifacts in paths. It also cannot effectively control intermediate-scale tree architectures.

3.3.2 Design decisions of our thesis work

In our thesis work, we developed two designs – the method with iterated graphs (MIG) initially and the method with guiding vectors (MGV) later. MIG is focused on the module of graph creation. It can create hierarchical tree structures and control the global shape of trees, but its control over intermediate-scale tree architectures is weak. Our experience from MIG inspired the design of MGV. MGV is focused on the module of setting edge weights. It is capable of creating high-quality tree models by controlling their global shapes and intermediate-scale architectures. Next we briefly describe the design decisions of MIG and MGV. More details appear in Chapter 4 – 6.

In MIG, we use the Yao graph to organize points into a graph, in order to eliminate the lattice artifacts and to have a bounded outdegree. We create a hierarchical graph through iterations. In the first iteration, we compose the graph shape using geometric primitives

and build a Yao graph with the nodes in the volume. At each later iteration, we create subgraphs based on the least-cost paths from the previous iteration. Least-cost paths to the endpoints placed in the graph volume form the desired global tree shape. More detailed branch features such as diversity of branch lengths and branch orientations can be controlled by specifying subgraph properties such as dimensions and orientation.

In MIG, we set the weight of each edge to the length of the edge, to get paths that reflect the Euclidian distance between the root and endpoints. Paths in each individual subgraph do not curve as we want. We create a long curving branch by incrementally rotating subgraphs and linking up paths from the subgraphs. This method can control branch shapes, but the branches do not curve in a natural way.

The strength of MIG is the control over the global shape and the intermediate-scale architecture of trees through hierarchical graph. The resulting tree models have better quality and more variations than BCM trees. The drawback is that the control over intermediate-scale tree architectures is weak, lacking a scheme to systematically control the branch shapes.

In MGV, our graph is a unified Yao graph, where all graph nodes are placed in an integral graph volume. We compute least-cost paths in this unified graph volume, for the unity of resulting branch shapes. We control a tree's global shape partly by placing endpoints in a user-specified volume and partly by specifying guiding vectors.

We control intermediate-scale tree architectures with inspiration from MIG, in which branch shapes can be specified by piecewise rotating the branch's local orientations. In MGV, we set edge weights according to guiding vectors. Edge weights will be less for edges oriented along the guiding vector, and more expensive as the directions diverge. Thus, least-cost paths will follow the vectors, allowing us to control local branch orientations. We specify a node's guiding vector through an incremental rotation of its parent's guiding vector. The resulting paths will curve according to the guiding vectors. Compared with MIG, the control of MGV is loose and flexible. While MIG constrains branch connections in subgraphs, MGV does not have this hard constraint, leading to more naturally curving branches.

The strength of MGV is the effective control over the global shape and the intermediate-scale architecture of trees. With the design of MGV, we can create a wide variety of high-quality tree models. Some examples are shown in Figure 3.11. The experience in creating graphs, setting edge weights, and placing endpoints contribute to the entire path planning tree modeling system, and may be enlightening to other applications where path planning is used.

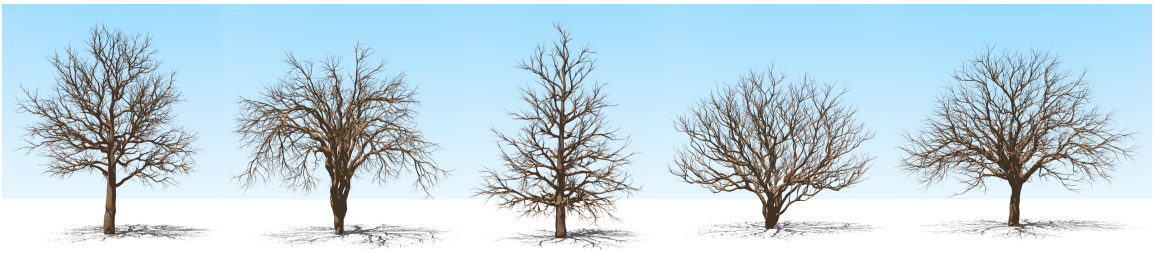


Figure 3.11: Some tree models created using MGV.

Chapter 4

Realistic Tree Modeling with Iterated Graphs

4.1 Introduction

In this chapter we present the method with iterated graphs (MIG) for realistic tree modeling. MIG emphasizes the design of the graph creation module. It tends to control the resulting structures through hierarchical graph and endpoint placement.

We decompose a tree into different levels of branches: the first level is a basic structure composed of a trunk and a few primary branches, and later levels are smaller branches that gradually augment the structure in the previous level. MIG creates a tree through iterations; each iteration corresponds to a level in the tree structure. In the first iteration, we create the basic branching structure. The shape of the initial graph is composed of simple geometric primitives. It restricts the paths inside the volume, controlling the global tree shape. In later iterations, we create subgraphs around each endpoint in the previous iteration and begin again to create least-cost paths to newly selected endpoints. The purpose of using subgraphs and creating subgraphs in such a way is to add paths to the existing structure, so as to create a more and more complicated tree shape expanding outward. By modifying parameters of the initial graph and subgraph creation processes and by changing the endpoint distribution mechanisms, we can create a wide range of tree models.

4.2 The first iteration

We create a basic structure in a Yao graph. The graph is a designated volume filled with nodes. To create the graph, we place each node at a random location, and determine whether it is within the permitted graph volume. The nodes themselves take a Poisson disc distribution, where the size of the disc dictates the branch feature size and the minimum branch tip spacing. Once a specified number of nodes have been placed, we connect nodes with edges to build a Yao-8 graph [93].

We set the weight of each edge in the graph to the length of the edge. This is not the only option to set edge weights, but sufficient in MIG to generate a variety of trees.

We choose endpoints in a specified region in the graph. To choose an endpoint, we randomly choose a graph node and inspect whether it is in a designated region of the graph, such as a thin shell near the surface of the graph; the thickness of the shell can be varied to decide the diversity of branch lengths.

4.3 Later iterations

To create a hierarchical tree structure, we iteratively extend the original graph by adding subgraphs at earlier endpoints. New branches in subgraphs are added to the previous structure in order to build a more complicated tree structure.

The overall process operates as follows. In level $i = 1$ we build a base graph and create a preliminary tree model by connecting the endpoints with the root using least-cost paths. Subsequently, for level $i > 1$, we create a subgraph around each endpoint from level $i-1$, again defining a volume and distributing nodes and endpoints within it. The endpoints are connected to the structure obtained in level $i-1$, producing a new structure. We define each subgraph volume as a cone-shaped subset of a sphere with the tip at the endpoint, with an intention to make new paths originate from the tip and spread out. The process repeats for some number of levels, say 4; depending on the desired complexity of the final structure, the number could be higher.

The above process is illustrated in Figure 4.1. The initial graph is a composition of a hemisphere and a cylinder. Endpoints are randomly positioned in the hemisphere, and paths are planned from the root to the endpoints. Then, a subgraph is created for each endpoint, as illustrated in the third image. The growth vector, $\vec{v_g}$, is the orientation of subgraph, represented using the red arrow in the bottom left image. The last image shows the structure once the second level has been completed. Pseudocode describing the building process is given in Figure 4.2.

The preceding gives the process to construct the schematic of the model; we then interpret the paths as geometry, placing a cylinder (truncated cone) around each edge in the structure. The radii of the cylinders are computed as follows. Working backwards from each endpoint of the final level, we compute the sum of segment lengths to a given node;

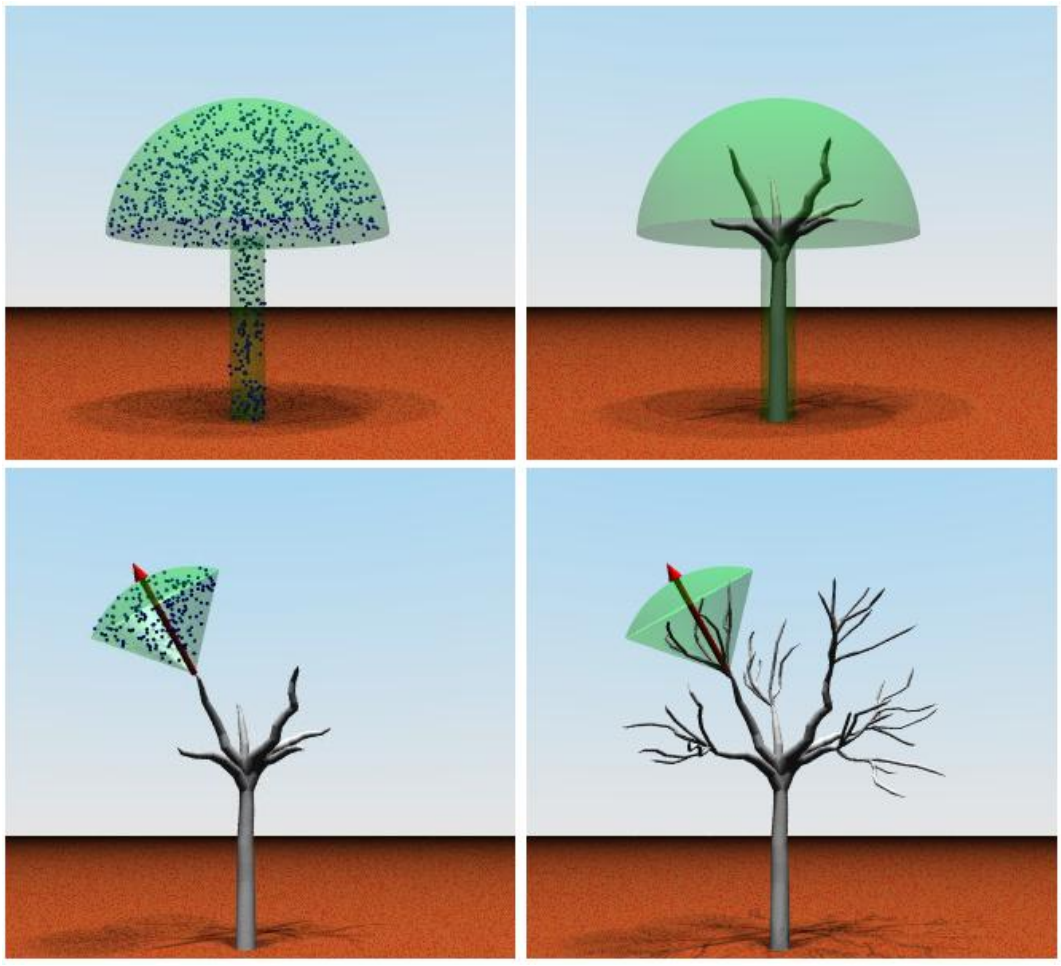


Figure 4.1: Illustration of the iterative tree building process.

Maketree

Global parameters:
subgraph resolution k (number of nodes)
node linking distance d
subgraph shrinking parameter a ($a < 1$)
subgraph sizing angle α
Note, a node N has property $\vec{x}$, spatial position.
Arguments:
V (graph volume),
R (root node),
b (branching factor),
K (number of nodes in graph),
r (subgraph size),
L (remaining lifespan)
Output:
T, a list of all path segments making up the tree.

1. Create the graph G for the current volume:
   1A. $G \leftarrow \emptyset$.
   1B. Find a random location $\vec{p}$.
   1C. If $\vec{p}$ is outside V, reject $\vec{p}$.
   1D. For all nodes $N \in G$, if $|\vec{p} - N.\vec{x}| < d$ reject $\vec{p}$.
   1E. If $\vec{p}$ not rejected, create node m with $m.\vec{x} = \vec{p}$ and set $G \leftarrow \{G, m\}$.
   1F. Repeat 1B to 1E while $|G| < K$.
   1G. Create edges between nodes in G such that G is a Yao graph.
   1H. For all edges, set weights to edge lengths.
2. Create a set of endpoints S:
   2A. $S \leftarrow \emptyset$.
   2B. Choose a random node $e \in G$.
   2C. Set $S \leftarrow \{S, e\}$.
   2D. Repeat 2B to 2C while $|S| < b$.
3. Create paths for all endpoints $e \in S$; before starting, initialize $T \leftarrow \emptyset$.
   3A. Find the least-cost sequence of edges P from e to R.
   3B. For all edges $E \in P \setminus T, T \leftarrow \{T, E\}$.
4. Recurse on all endpoints:
   4A. For each endpoint e:
   4B. $\vec{v_g} = (e.\vec{x} - R.\vec{x})/|e.\vec{x} - R.\vec{x}|$.
   4C. define V as the portion of the sphere centered at $e.\vec{x}$ with radius r that lies within angle α of $\vec{v_g}$
   4D. If $L > 0$, $T \leftarrow \{T$, maketree$(V, e, b, k, a * r, L - 1)\}$
5. Return T.

Figure 4.2: Pseudocode for tree construction.

call this distance s. We then compute a thickness w for the node using a tapering parameter ζ and the distance: $w = s^{\zeta}$. Larger values of ζ make the branches taper more quickly. A typical choice for ζ is 0.3. The purpose to set branch widths according to s is to make branch widths taper gradually from the root to the tip. Because our target is procedural modeling, we did not place emphasis on the rendering of branches. Other existing methods such as the pipe model [67] or any functions that can achieve similar visual effects will also work. In the pipe model, the cross-section of a branch is decided by the combined cross-sections of the branches above. A tree from the pipe model is shown in Figure 6.2.

A tree structure created through four iterations is shown Figure 4.3. The initial shape is a mushroom shaped cylinder plus hemisphere, reflected in the overall shape of the final tree. The top view reveals the desired horizontal anisotropy of the tree, while the close view allows better appreciation of the detailed small-scale structure.

4.4 Elements of the algorithm

In this section we discuss the two main elements of the algorithm: initial graph creation and subgraph creation. The initial graph shape and associated endpoint placement have a profound effect on the overall shape of a tree. Subgraphs control how a tree develops at levels beyond the first, and affect the general appearance of the tree in a more subtle way.

4.4.1 Initial Graph Shape

The graph shape in the first level controls the overall shape of the resulting model. The general shape of a tree can be described using combinations of geometric primitives including cylinder, sphere, hemisphere, partial sphere, and cone. We build the first level of the graph by placing graph nodes in the combination of a cylinder where the trunk is to exist and a hemisphere (or other shapes) where the tree crown is to exist. Users can decide dimensions and orientation of the primitives to create different tree shapes. The height of the cylinder decides the height of the trunk; the angle of the cylinder to the ground controls how tilted the trunk is; the radius of the hemisphere decides the maximum size of the crown. The width of the cylinder affects the tree architectures: a wide cylinder provides a large volume for paths to exist, used to create trees with multiple primary branches; a narrow cylinder

Figure 4.3: A tree model with a four-level structure.

restricts the space for paths, so that paths are forced to be close to each other and usually merge to a single path to the root.

Figure 4.4 shows three different tree shapes along with their corresponding graph shapes. For all three trees, we use cylinders to approximate tree trunks. From left to right, tree crowns are a cone-shaped subset of a sphere, an ellipsoid, and a sphere. Because of paths developed in later levels, the final model does not strictly match the original graph shape but shows the lasting influence of the initial volume. Note the difference between the endpoint selection mechanism we use and the clipping mechanism of the synthetic topiary [61]; our final trees do not conform very closely to the specified volume, so as to produce a less structured and more organic shape. Figure 4.5 shows an example of simulating an irregular tree shape by specifying the initial graph shape. The graph in the first level is composed of a cone-shaped subset of a sphere and a tilted cylinder; endpoints are chosen manually to match the desired outcome.

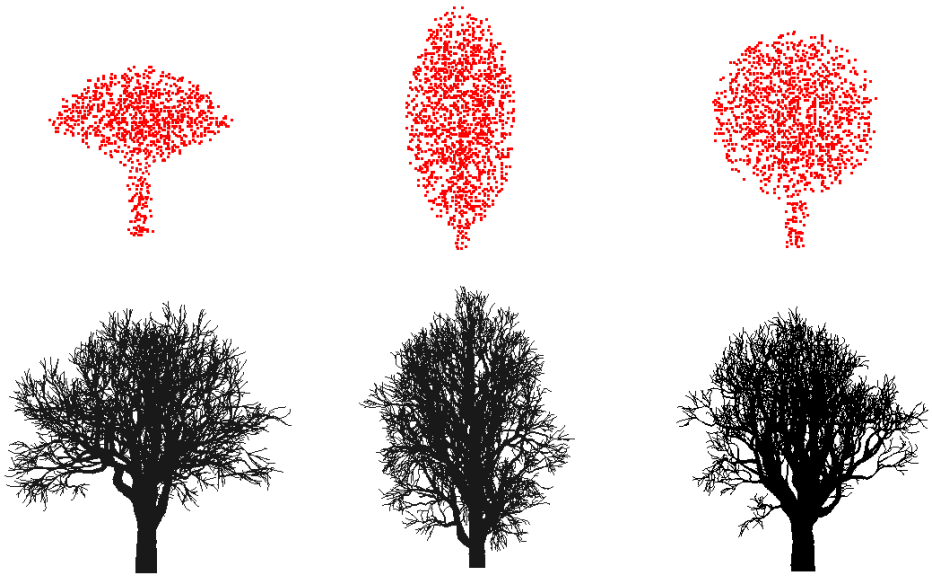


Figure 4.4: Structures obtained by different shapes of graph.

Using simple geometric primitives to build the graph in the first level can generate a uniform-looking tree structure. The overall tree silhouette follows the primitive shape. In general, natural trees have more irregular silhouettes with clusters of branches with diversity in length and are separated by gaps. We can control the small-scale irregularity of tree silhouettes by restricting endpoints to a shell near the surface of the volume, as

Figure 4.5: A tree with a tilted trunk. The middle photo comes from Flickr.

described next.

We select endpoints from a shell of thickness d_e near the surface of the graph. A greater value of d_e, yielding a large interval, produces branches with greater diversity of length; smaller d_e, and a small interval, forces endpoints into a thin shell so that the resulting paths have little diversity of length. Figure 4.6 illustrates how d_e affects branch length variability. The graph is a composition of a hemisphere and a cylinder, shown in green. Endpoints are placed in the shell between the green and orange hemispheres. The thickness of the shell is indicated by the yellow arrow. The resulting tree skeleton has not only long branches but also short branches. In the right tree, endpoints are placed close to the surface of the hemisphere; the branch lengths do not vary much. The use of a greater or smaller d_e depends on which kind of branches users intend to create. Figure 4.7 shows the tree models after four growth levels. The left tree obtained with greater d_e has more diversity of branches and irregularity of silhouette, while the right one has a more homogeneous appearance.

For yet higher degrees of irregularity in the tree shape, we can refine the first-level graph, populating the large-scale structure with smaller elements. We chose to use spheres in the examples, although other shapes could be substituted instead. We fill the coarse geometry with a Poisson disc distribution of points, where the disc spacing is quite large, allowing (say) only 4-10 points to be placed. For each such point, we place a sphere; we also place a conical volume connecting the sphere to the trunk or tree root. The union of all such volumes is filled with graph nodes, forming the first level in the iterated graph building. Subsequent levels are added as before. The process is visualized in Figure 4.8.

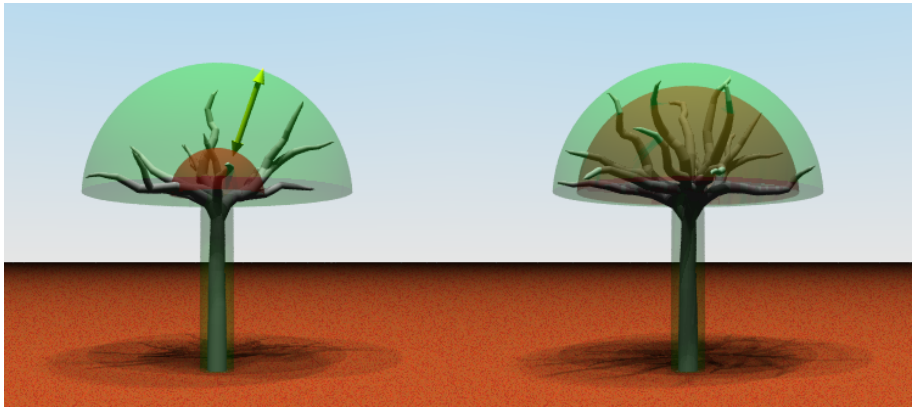


Figure 4.6: Two structures with different shells to place endpoints. Left: a thicker shell can generate more branch variations in length; right: a thin shell provides more uniform looking branches.

Figure 4.7: Two structures with different values of d_e.

The purpose of adding a conical volume attaching the sphere to the trunk is to ensure that endpoints within each sphere will have a path to the root. Further, it allows us to ensure that no edges link nodes within different subvolumes at the first level. As a consequence, individual subvolumes will remain distinct: their structure will not be blurred by shared edges. Also, when we place endpoints in the first level, we can allocate a fixed number to each subvolume, ensuring balanced coverage of the global graph shape.

Figure 4.9 compares trees from the single-volume process to the subvolume process. Enforcing distinct subvolumes has provided noticeably more irregularity in the global shape, with greater complexity of branch distribution and higher diversity of silhouette; further, the branches are visibly organized into clusters, which enhance the natural appearance of the trees.

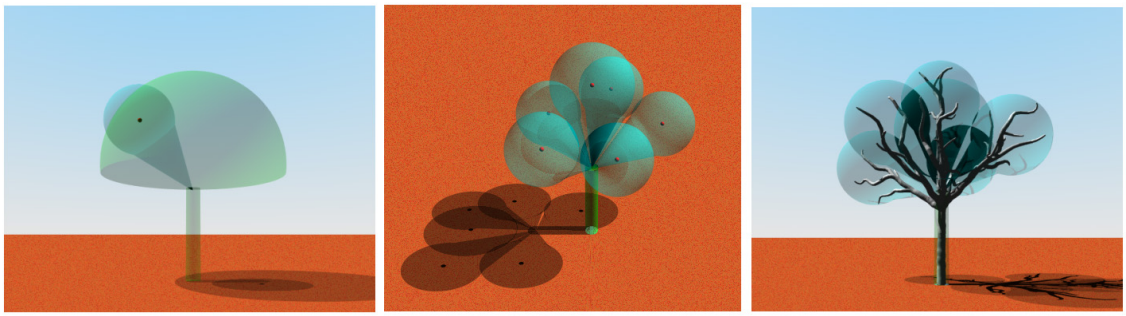


Figure 4.8: The process to refine the basic graph volume with spheres and cones.

4.4.2 Subgraph Creation

We require users to specify the graph shape for the first level, providing control over the tree's large-scale appearance. While subgraph shapes for subsequent levels can in principle also be user-defined, in practice it is tedious to do so, so we compute the subgraph volumes procedurally, as follows.

We use a cone-shaped subset of the sphere as our subgraph, where the cone's tip is placed at the endpoint in the previous level. First, we compute an orientation $\vec{v}_g$ for the subgraph by taking the normalized vector from the root of this subgraph to the root of the previous subgraph. The volume is defined as all points whose vector from the root lie within an angle α of the vector $\vec{v}_g$. The volume is populated with nodes in the same way as the initial graph, and the subgraph is also a Yao-8 graph. The purpose of computing $\vec{v}_g$ and

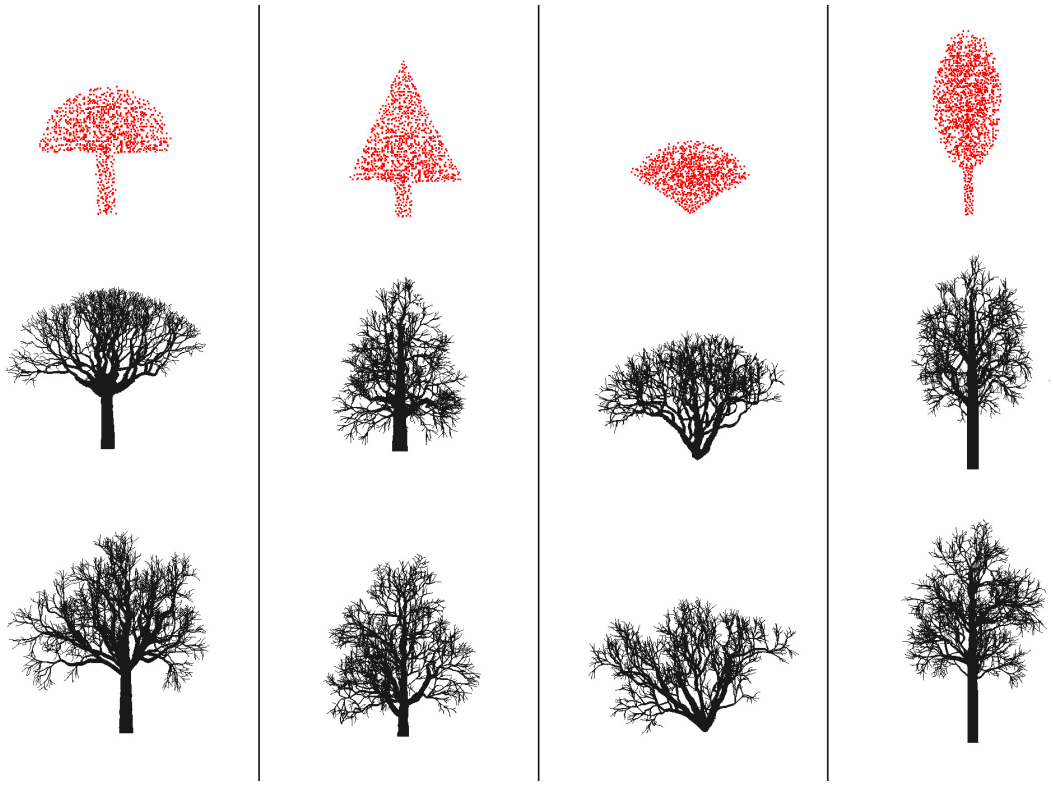


Figure 4.9: Top row: basic volume to build graph; middle row: tree models obtained without a refinement of graph volumes.; lower row: tree models obtained with a refinement of graph volumes.

defining the subgraph volume in this way is to make the added least-cost paths naturally attach to the previous branch based on its local orientations, allowing us to roughly control how branches develop. In addition, edges that connect two nodes belonging to different subgraphs are forbidden, in order to avoid subgraphs merging. For a controllable tree construction process, we want to make branches spread out from their subgraph's root by restricting the paths inside their own subgraphs. Merging subgraphs may result in paths that cross different subgraphs and attach to other subgraph's root.

The crown shape is directly influenced by the initial graph shape and subgraph dimensions. If we use the same shape for the crown part in the first level and subgraphs, the parameter α will directly affect the resulting crown shape. Figure 4.10 shows the variations of crown size with different α values for each row in four growth levels (from left to right): smaller α values result in tightly crowded crown branches, and greater α values generate a more widely spread-out shape.

Another parameter, branching factor b, decides how branches split globally. A branch can split into more child branches with a larger b. To create our tree models, we use a constant value of b, but if desired we can vary the value of b with levels. Figure 4.11 shows trees with different branching factor b. When the number of endpoints increases, the resulting tree has more branches. In our opinion, $b = 6$ produces an appealing tree whose branches are dense but not overly crowded.

The size of the subgraph sphere decreases as we progress to higher levels: the parameter a, where usually $a < 1$, is the ratio between the sizes of spheres at two successive iterations. The purpose to use $a < 1$ is to make branches shorter when the tree expands outward with the progressing of iterations. Figure 4.12 shows two trees obtained with same parameter settings except a. The left tree, with $a = 0.7$, demonstrates an obvious hierarchical relationship in branch length. The branch segments closer to the trunk are long and those near the tips are short. The tree in the right has a constant subgraph size (with $a = 1$) at each level, leading to a more homogeneous appearance.

We create paths with long-term curvature (similar to willow branches, for example) using paths from successively rotated subgraphs. We have previously described the subgraph orientation $\vec{v}_g$, obtained by finding the vector from the subgraph root to the preceding root; now, we apply a consistent transformation to $\vec{v}_g$ at each level. When the transformation is a

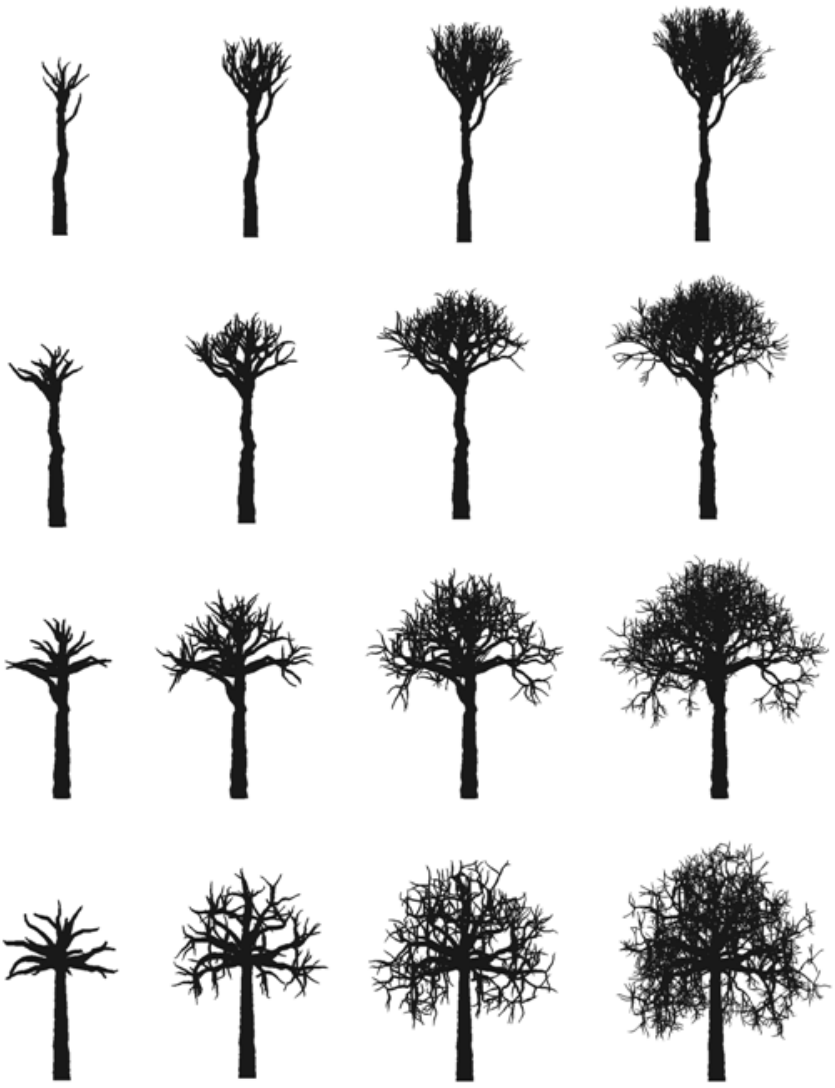


Figure 4.10: Different tree shape that vary with α. From left to right: trees in four growth levels. Each column: trees with different α.

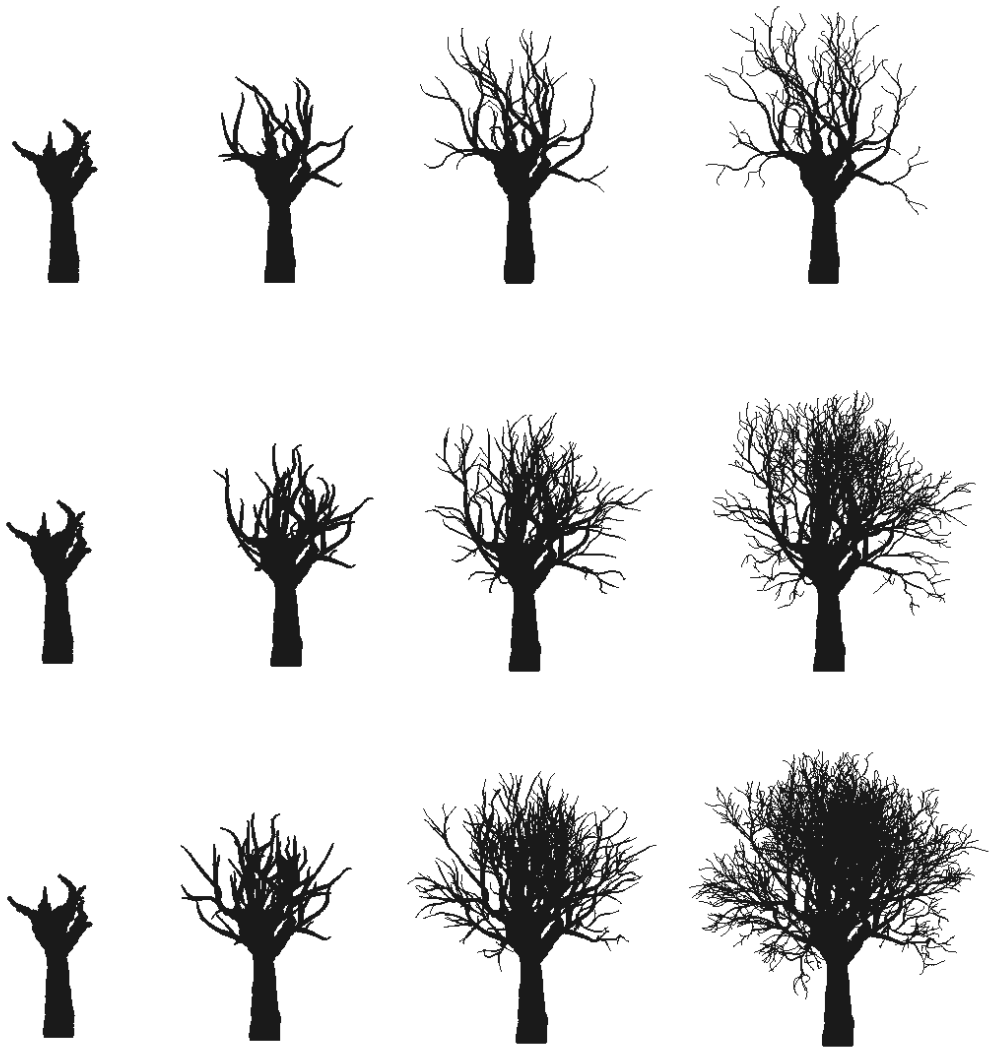


Figure 4.11: Trees with dense or sparse branches in four growth levels (from left to right). From top to bottom, $b = 2, 4, 6$.

Figure 4.12: Left: tree with $a = 0.7$; right: tree with $a = 1$.

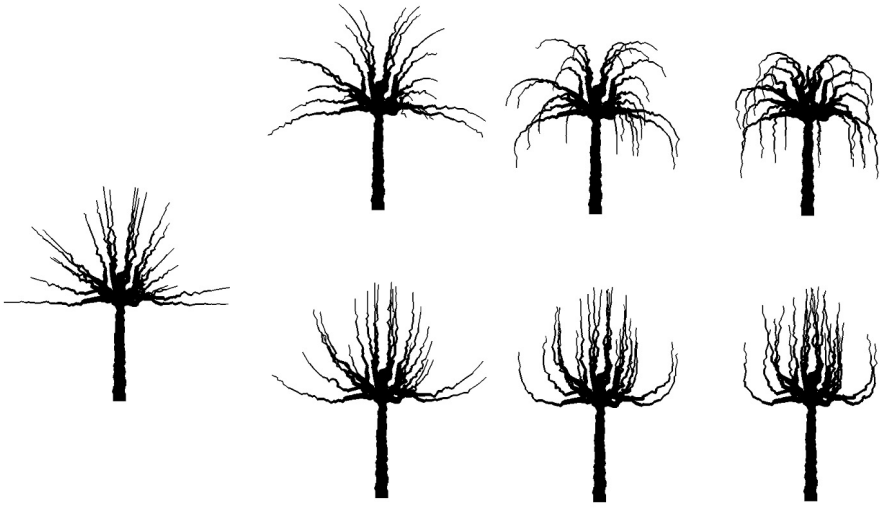


Figure 4.13: Curving branches from progressively rotating subgraphs.

rotation about the horizontal, the overall branch curves upward or downward. Figure 4.13 shows structures obtained by the method, with four iterations applied. To create individual long branches without bifurcation, every iteration has only one endpoint per subgraph.

Note that this particular procedural approach to subgraph shape is not the only possibility, although it is a convenient option that we rely heavily on in the work. Other possibilities include the following: different subgraph shapes, e.g., spheres; or different mechanisms for computing the orientation, e.g., using a fixed orientation to make branches constantly grow toward a fixed direction.

The method in Figure 4.13 provides control over branch shapes by piecewise adjusting local orientations of paths, but it still has limitations. First, we only make a branch constantly curve upward or downward, but do not have a systematic control to create more variations, such as a branch that has an S-shape. Second, because each branch is formed by linking up individual paths in subgraphs, and the paths in each subgraph do not curve as we want, the resulting branches do not curve in a natural way.

Despite the limitations, the method contributes an idea of controlling branch shapes: we can vary a path's local orientations along its length towards its tip through incremental rotations. This idea is promising to create more variations of branch shapes if we could design a sophisticated method to specify the rotations.

4.5 Results

Using MIG we can create tree models by controlling the global shape through graph construction and adding fine-scale branches through iterations. Elements of the algorithm – initial graph shape, subgraph dimensions, and endpoint placement – provide rich variations of results. Some of the results are shown in this section, and some are shown in chapter 6.

Due to the involvement of random elements – particularly random placement of graph nodes and endpoints – we can generate similar but distinct trees by keeping parameter settings fixed. Figure 4.14 shows three variations on a base tree type. In each case, the initial graph is composed of a cylinder and a hemisphere, but each tree has a slightly different structure while keeping large-scale characteristics in common, such as the crown shape and the branch distribution. Figure 4.15 shows more such trees in a forest.

By varying graph shapes and available parameters, we can create a wide variety of trees;

Figure 4.14: Three trees of the same type.

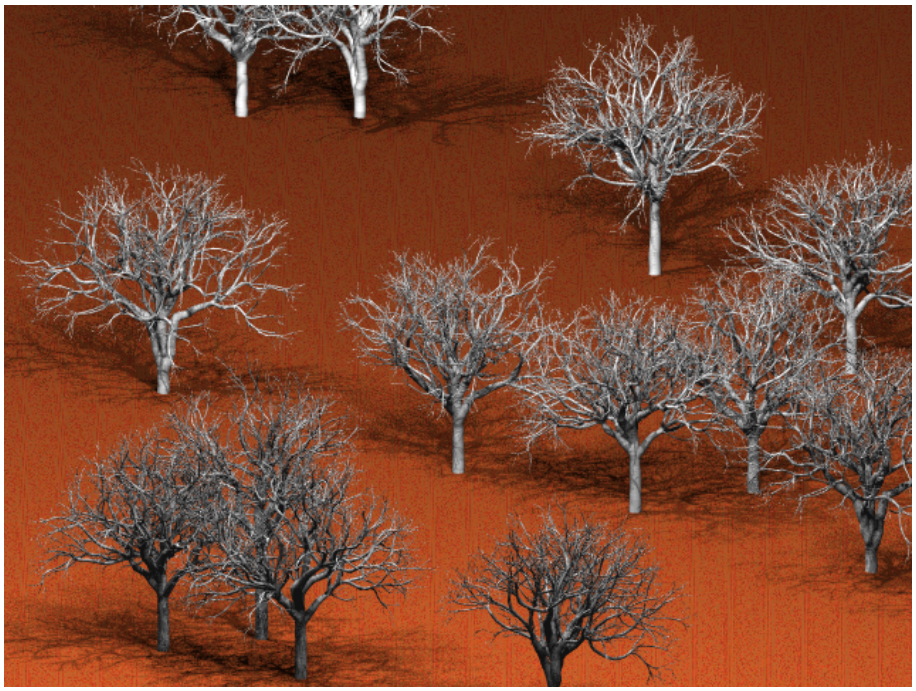


Figure 4.15: A scene with a forest of trees.

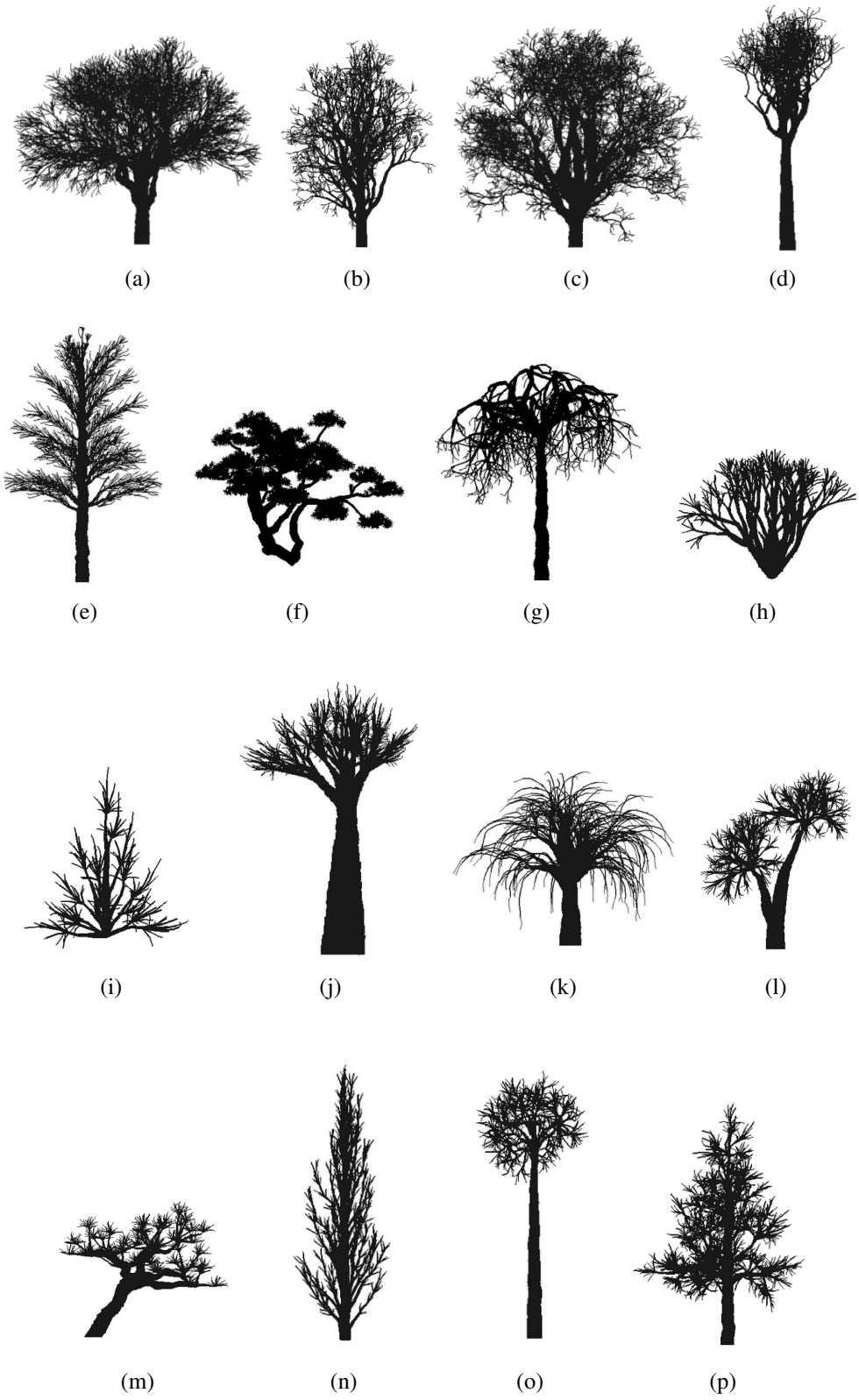


Figure 4.16: Different types of trees.

examples are shown in Figure 4.16. Each tree demonstrates a distinct global shape, from graph volumes composed with different geometric shapes: tree (a) has a hemispherical crown; tree (c) and (o) possess a spherical crown; tree (p) has a cone shaped crown; tree (m) has a short and tilted trunk, and tree (o) has a straight and tall trunk. Distinct tree architectures can be created using different number of endpoints in the first level: some trees have a trunk e.g. tree (b), (d), (e), and (j); some trees have several primary branches, e.g. tree (f) and (l); and some possess bushy structures with dense branches e.g. tree (h) and (i). The orientation of subgraphs and endpoint placement affect the branch shapes and are used to create appealing structures. Tree (g) has hanging branches from subgraphs oriented downward. Tree (k) is intended to mimic a willow tree, created by rotating the subgraphs, using the same method to create the structures in Figure 4.13. Tree (e), (n) and (p) have a noticeable structure different from others: all have a central trunk attached with side branches, and twigs are symmetrically distributed along the side branches. To create the structure, we create the trunk first, and progressively create individual side branches as we did for the willow branches. At the final level, we build subgraphs around endpoints from all previous levels. Paths from each subgraph form the twigs. The parameter information for the trees in Figure 4.16 is given in Table 4.1.

We can combine and mix parameter settings to create more trees as shown in Figure 4.17. Parameters of trees in each crossing are partly from the corresponding trees in the left column and in the top row. For example, to create the tree in the second row and third column, we first build the basic skeleton using the pine tree framework to get the first level of structure, and in the following levels we use the parameter settings of the willow tree to create long and curving branches. The resulting structure possesses characteristics from both trees: the general shape from the pine tree, and the branch pattern from the willow tree.

In Figure 4.18, our synthetic tree images are compared with photographs. Our trees have similar structures to the photographed trees: similar tree crown shapes, a few thick main branches, and a large volume filled with twigs yet with natural-seeming irregularities and gaps. To create leaves in the second and fourth tree, we draw different sizes of polygons; each is centered at an endpoint. To make the leaves look natural, we randomly rotate polygons. To simulate pine needles in the left tree in the third row, we draw a few lines

Tree	Level i	Graph Shape	b	Note
a	1	cylinder and portion of sphere with $\alpha = 0.25\pi$	12	
	2, 3	$\alpha = 0.3\pi$	6	
	4	$\alpha = 0.3\pi$	12	
b	1	cylinder and ellipsoid	35	
	2-4	$\alpha = 0.4\pi$	4	
c	1	cylinder and portion of sphere with $\alpha = 0.4\pi$	25	
	2-4	$\alpha = 0.4\pi$	6	
d	1	cylinder and portion of sphere with $\alpha = 0.3\pi$	12	
	2, 3	$\alpha = 0.3\pi$	3	
	4	$\alpha = 0.3\pi$	6	
e	1	cylinder and cone	32	Rotate $\vec{v}_g$ in levels 2-7 downwards with 0.3π. At final level, build subgraphs around all previous endpoints.
	2-7	$\alpha = 0.25\pi$	1	
	8	$\alpha = 0.25\pi$	8	
f	1,2	portion of sphere with $\alpha = 0.3\pi$	6	
	3	$\alpha = 0.4\pi$	30	
g	1	portion of sphere with $\alpha = 0.5\pi$	10	$\vec{v}_g$ of level 2 are oriented in the negative vertical direction.
	2,3	$\alpha = 0.5\pi$	4	
	4	$\alpha = 0.3\pi$	6	
h	1	portion of sphere with $\alpha = 0.3\pi$	42	
	2,3	$\alpha = 0.3\pi$	3	
	4	$\alpha = 0.3\pi$	4	
i	1	cone	16	
	2	$\alpha = 0.25\pi$	1	Rotate $\vec{v}_g$ of level 2 upwards with 0.03π
	3	cone	10	
j	1	cylinder and portion of sphere with $\alpha = 0.25\pi$	10	
	2	$\alpha = 0.25\pi$	5	
	3	$\alpha = 0.25\pi$	10	
k	1	cylinder and portion of sphere with $\alpha = 0.5\pi$	30	
	2,5	$\alpha = 0.25\pi$	2	Rotate $\vec{v}_g$ of level 2-8 downwards with 0.12π
	3-8	$\alpha = 0.25\pi$	1	
l	1	cylinder and portion of sphere with $\alpha = \pi$	2	
	2	$\alpha = 0.4\pi$	10	
	3,4	$\alpha = 0.4\pi$	6	
m	1	tilted cylinder and hemisphere	8	
	2	$\alpha = 0.5\pi$	6	
	3	$\alpha = 0.5\pi$	12	
n	1	cone	40	Rotate $\vec{v}_g$ in levels 2-4 upwards with 0.05π. At final level, build subgraphs around all previous endpoints.
	2-4	$\alpha = 0.25\pi$	1	
	5	$\alpha = 0.25\pi$	3	
o	1	cylinder and portion of sphere with $\alpha = 0.7\pi$	11	
	2,3	$\alpha = 0.45\pi$	6	
p	1	cylinder and cone	16	
	2,3	cone	8	

Table 4.1: Parameters for the models in Figure 4.16

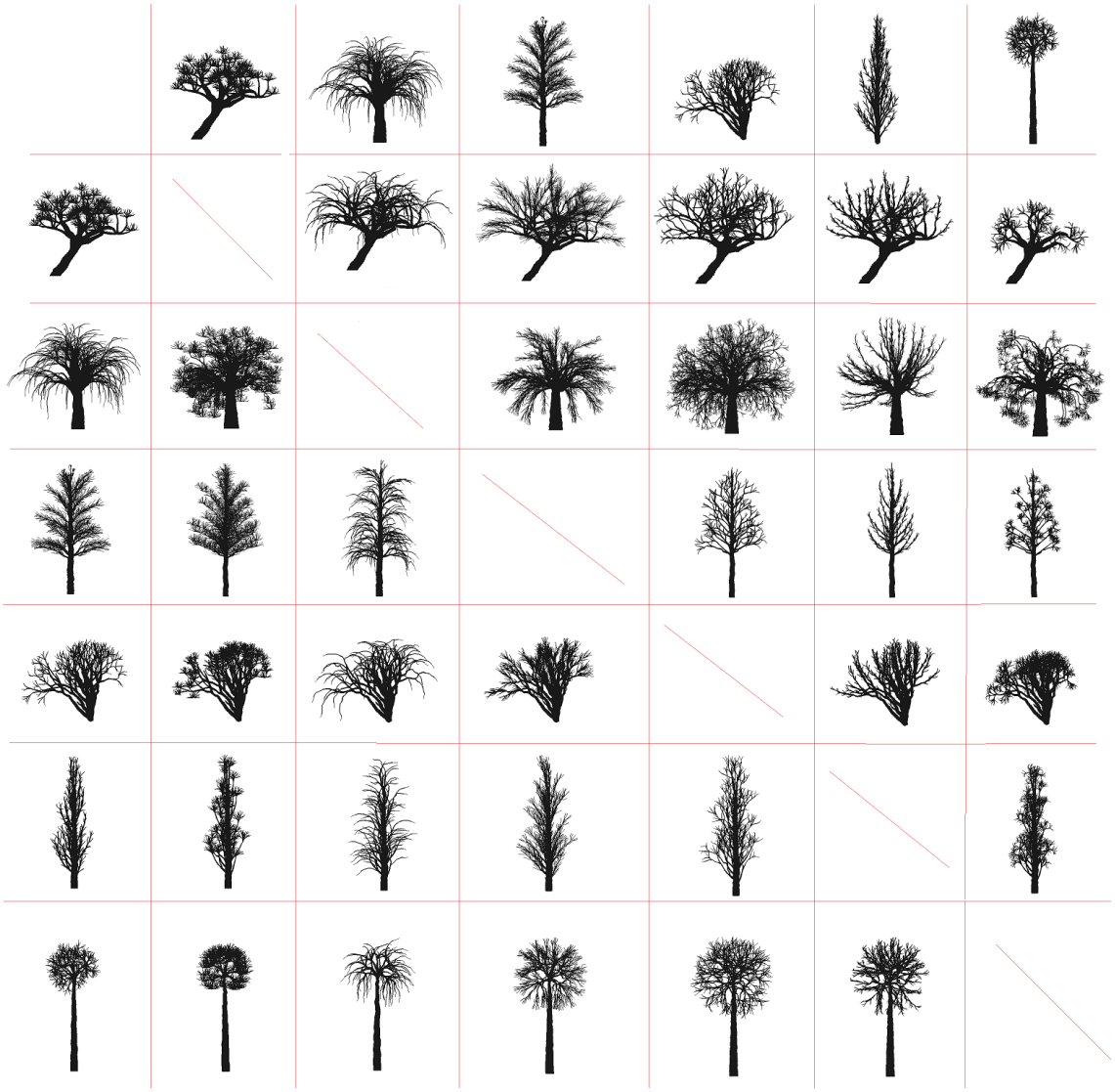


Figure 4.17: Hybrid trees obtained by existing tree models

Figure 4.18: Left: our tree model; right: real photograph from Flickr.

starting from each endpoint; each line is about 5 pixels long and deviates from the upper vector at random angles. Overall, it is difficult to distinguish between the real and synthetic trees from these images, and we judge that our method is quite effective.

In addition to the wide variety of trees shown, our algorithm can be used to model the root system. Figure 4.19 shows two trees with different shapes of root systems; the roots were created in a graph bounded by a hemisphere. The taproot is a path from one single endpoint to the root point. The lateral roots of both trees are obtained by placing endpoints randomly around the taproots.

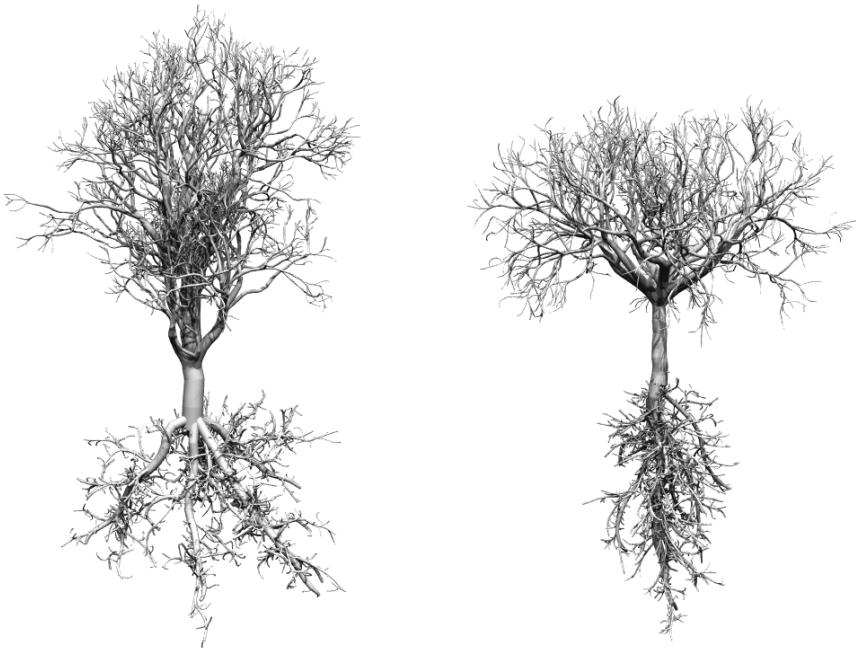


Figure 4.19: Two trees with different root systems

4.6 Summary

In this chapter, we introduced the method with iterated graphs (MIG) for realistic tree modeling. We demonstrated that MIG is capable of producing some elaborate and realistic trees. Wide variations are possible, producing many different shapes of trees. Since the algorithm involves random placement of graph nodes and endpoints, many different but similar models can be constructed from the same parameter settings. We provide large-scale control through specifying graph shape. Optionally, the finer structure can be guided

by specifying graph shapes to use in later iterations; we provide defaults which provide generically appealing results.

Although MIG offers certain benefits, the lack of effective and systematic control over branch shapes is still a problem. In the next chapter we will introduce the design of MGV and show how it deals with this problem.

The work in this chapter has been published as follows:

- L. Xu and D. Mould. Synthetic tree models from iterated discrete graphs. In *Proceedings of Graphics Interface*, pages 149–156, 2012
- L. Xu and D. Mould. A procedural method for irregular tree models. *Computers and Graphics*, 36(8):1036–1047, Dec. 2012

Chapter 5

Realistic Tree Modeling with Guiding Vectors

5.1 Introduction

In MGV, we propose *guiding vectors*. Each graph node has an associated guiding vector; the weight of each edge will be set such that travelling in the direction of the vector is inexpensive, while travelling perpendicular to the vector or in the opposite direction is extremely costly. Thus, least-cost paths will follow the vectors, allowing us to specify the medium-scale flow of the branches while retaining the advantages of single-source shortest paths, such as guarantees of no loops or self-intersections. MGV can effectively control intermediate-scale tree architectures and create high-quality tree models resembling a wide variety of real-world tree species.

5.2 Algorithm Design

Compared with MIG, MGV emphasizes the setting of edge weights. A tree's global shape is controlled partly by placing endpoints in a user-specified volume and partly by specifying guiding vectors. Its intermediate-scale architecture is controlled by guiding vectors. The process to create a hierarchical tree structure includes the following steps.

First, we create a unified graph, for systematic control of branch shapes and the unity of resulting tree shape. The graph is a Yao-8 graph, where each node has directed edges, pointing from the node to its nearest neighbor in each sector.

Second, we merge the guiding vector creation and the path creation processes when applying Dijkstra's algorithm. We compute edge weights under lazy evaluation: a node's guiding vector is set when it is visited, as a rotation from its parent node's vector, and undefined edge weights are assigned based on the angle between the edge vector and the guiding vector. Edges are cheapest along the guiding vector direction.

Third, as before, we select endpoints in the graph and determine the least-cost paths

from the source to these endpoints. Because of the effect from guiding vectors, the least-cost paths form our desired curving shapes.

Finally, we set the nodes in the obtained paths as the new source, and repeat the second and third steps for a number of iterations. Each iteration augments the previous tree by adding new branches.

In the following, we introduce more details of the above steps. We emphasize the second step – setting guiding vectors and edge weights, a design that is distinct from BCM and MIG.

5.2.1 Graph initialization

We create a unified Yao-8 graph, where graph nodes are placed in a 2D square or 3D cube according to the Poisson disc distribution. Each node has an associated data structure to store needed per-node information: its ID; distance from the source; ID of parent node; guiding vector; and outgoing edges. The root node has distance zero, and has a vertical guiding vector. As before, we place the root at the bottom centre of the graph. All other nodes have distance infinity and undefined guiding vectors; their vectors will be set in the next step.

5.2.2 Guiding vector and edge weight definition

In our method, lazy evaluation of guiding vectors and outgoing edge costs is done in concert with the path planning using Dijkstra's algorithm; the event of finalizing a node's cost also fixes its guiding vector: the guiding vector is determined by rotating its parent's guiding vector. The individual branch shapes thus follow a well-defined trajectory while maintaining an overall organization of the global tree structure. Typically, the rotation axis is the cross product of the up vector and the parent's guiding vector; the rotation angle is determined algorithmically, using specific rules or user-assigned parameters. A constant rotation angle causes branches to curve upwards or downwards in a predictable trajectory. Other strategies are possible, as detailed in section 5.3.1.

Once the newly opened node has a guiding vector, the weights of outgoing edges are set. The weight of an edge E is $d_E(1 - \vec{e}_E \cdot \vec{v}_E)$, where d_E is the length of the edge, $\vec{e}_E$ is a unit vector along the edge, and $\vec{v}_E$ is the guiding vector. Note that both $\vec{e}$ and $\vec{v}$ are unit

vectors. The weight has a minimum when the directions coincide and a maximum when the directions are opposite. Other functions are of course possible, but we have found this equation to be effective. Variations arise from altering the endpoint placement and the rules governing the guiding vectors.

In the absence of obstacles, shortest-path calculations tend to produce paths heading approximately along a straight line linking the origin and destination. Our use of guiding vectors, however, can produce natural-seeming curving paths. Figure 5.1 shows a comparison with and without guiding vectors. The crooked path on the left follows the shortest path through the graph. On the right, the guiding vectors push the shortest path upwards at first; only gradually does the path change course. Notice that the path is not a streamline through the vector field: it is still computed according to the shortest-path algorithm, just with edge weights that have been set according to guiding vector directions.

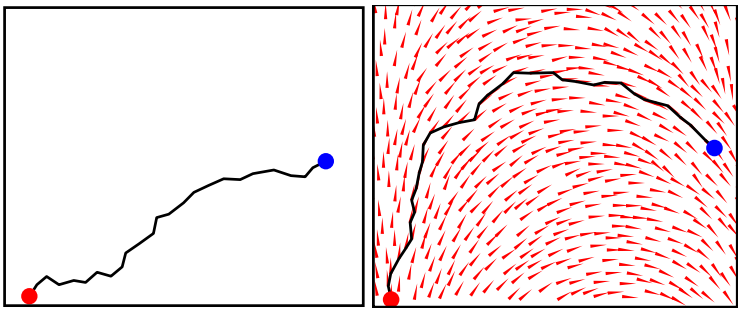


Figure 5.1: Left: a path from graph without guiding vectors; right: a path from graph with guiding vectors.

5.2.3 Branch construction

We extend the tree with a new set of branches. A subset of graph nodes is selected as endpoints; the least-cost paths from the source to these points will be new branches augmenting the emerging tree. We randomly choose the endpoints for the first level in our hierarchy from a user-specified volume, as we did in MIG. For later levels, a strategy that is aware of the current tree shape is called for; we suggest choosing endpoints from among the nodes that are k hops from the source in the previous iteration, thus causing the tree to extend outward. More details appear in section 5.3.3.

5.2.4 Hierarchical structure

To complete the tree, we repeatedly create new paths for typically 4 or 5 iterations in total. In each iteration, nodes that are not part of the structure are reinitialized to have unknown cost and unknown guiding vectors; edge weights also become undefined. Then, the current tree is set to be the source; optionally, the guiding vectors for the source nodes are changed; and the shortest-paths algorithm is used to propagate guiding vectors and costs into the remainder of the graph. A new set of endpoints is selected, the paths from the source to these endpoints recorded, and these paths are added to the source, whereupon the process repeats again.

We illustrate the iterative tree construction in 2D in Figure 5.2 and in 3D in Figure 5.3. The background of Figure 5.2 shows the guiding vector directions; the coherence arising from the vector propagation method is apparent.

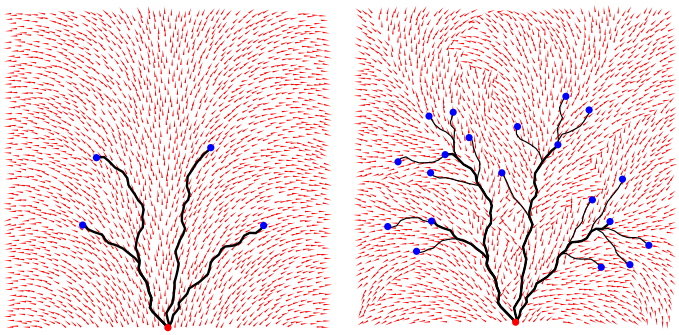


Figure 5.2: Iterative shortest paths and guiding vector assignment. Left: first iteration; right: second iteration.

As before, we use cylinders to represent edges in the resulting paths, creating a 3D tree model. Least-cost paths do not intersect. However, in principle the branches may intersect, since they have some thickness; sufficiently thick cylinders will indeed intersect. Of course, intersections at junctions (Y-shape) are unavoidable, but junctions do not represent a visual defect. In practice, we do not have branch intersections that lead to a wrong-looking tree, due to the following factors that oppose branch intersections in our algorithm.

First of all, branches are generally long and thin. The thickest part of the tree is the base of the trunk. Since the tree is spreading out, there are no branches in this region to produce intersections. In the outer part of the crown, where there are many branches

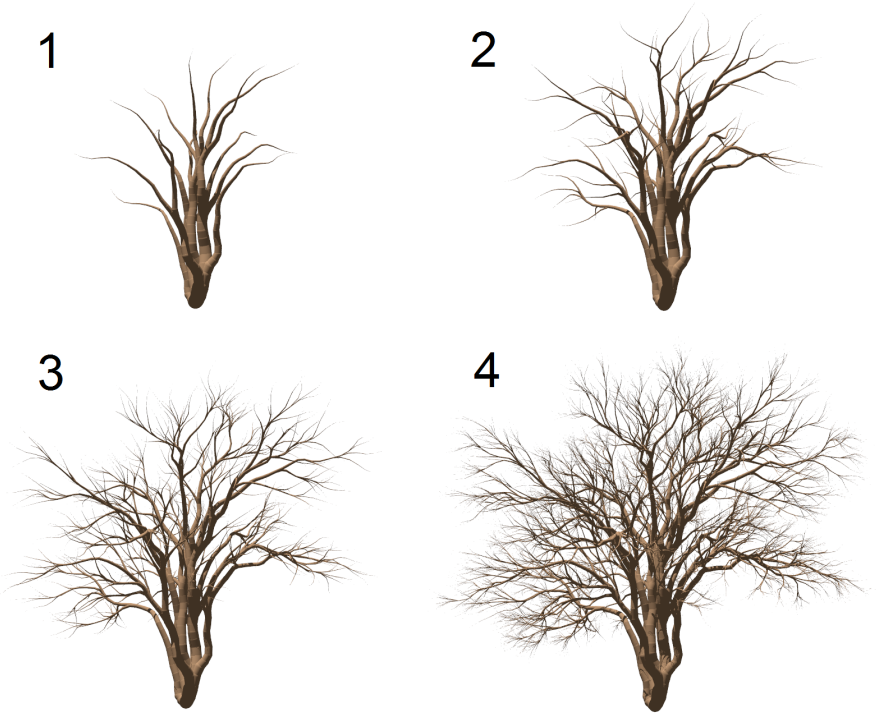


Figure 5.3: Four iterations of tree construction.

in different directions, each branch is quite thin, approximating a line. Very thin branches will not intersect. The structure of the graph also helps to prevent branch intersections. The Poisson disc node spacing guarantees a minimum distance between endpoints, in practice greater than the branch diameter. When the edges are well separated, the cylinders will not intersect. Intuitively, the Yao-8 graph with dense Poisson disc nodes should not allow non-adjacent edges to approach nearer than the endpoints; we have no proof of this, though.

In addition to the above factors, we have some empirical support for the claim that intersections will not occur: we have created more than 200 models, but did not observe any intersections. With parameter settings similar to ours, we do not anticipate branch intersections in the future. We cannot guarantee lack of intersections, though; trivially, a huge branch diameter relative to the node spacing will produce intersections.

5.3 Elements of the Algorithm

5.3.1 Guiding vector transformation

The guiding vector for a newly opened node is an incremental rotation of its parent's: the vector is rotated by an angle α about an axis perpendicular to both the up direction and the traversed edge. An example with $\alpha = 4^o$ is shown in Figure 5.4, upper left. With constant rotation angle α, we are limited in the types of tree we can create.

Making α a function of distance along the branch gives us more flexibility. Even a piecewise constant function can produce worthwhile results. Discretizing distance into the count of edges traversed (number of hops, h), we can write

$$\alpha(h) = \begin{cases} \alpha_1, & \text{if } h \leq h_0; \\ \alpha_2, & \text{otherwise.} \end{cases} \tag{5.1}$$

By varying the angles α_1 and α_2 and the switchover point h_0, we can obtain different types of trees. Figure 5.4 (upper right) shows an example with $\alpha_1 = 6^o$, $\alpha_2 = -6^o$, and $h_0 = 20$; the branches bend outwards and downwards for some distance, then begin to bend upwards. Unless otherwise stated, we use Equation 5.1 for all remaining trees.

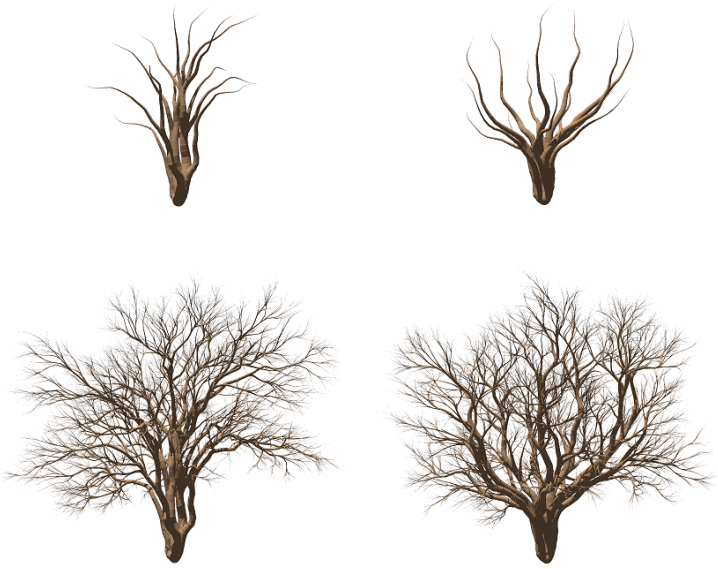


Figure 5.4: Left: constant rotation angle; right: rotation angle reverses after some distance. Above: first iteration; below: after four iterations.

The rotation parameters can be changed for later iterations; an example is shown in the lower row of Figure 5.4. The upper row shows the initial tree structure, and for both trees, later iterations use $\alpha_1 = 30^o$, $\alpha_2 = 0$ and $h_0 = 1$. A large α_1 produces a noticeable fork between a branch and its children, while $\alpha_2 = 0$ ensures that the child branches are generally straight.

Some additional variations on the rotation settings are shown in Figure 5.5. These examples differ solely on the rotation policy in the second iteration; the first iteration is used only to create a trunk (so $\alpha = 0$) and the later iterations produce straight branches ($\alpha_1 = 30^o$, $\alpha_2 = 0$ and $h_0 = 1$).

The partial trees after the second iteration are shown in the top row of Figure 5.5. Tree (a) has $\alpha = 10^o$, producing branches that strongly curve downward. With a smaller value of α, say 4^o, we can create branches that slightly curve downward, as in the oak tree model shown in Figure 6.3. Tree (b) uses $\alpha_1 = 60^o$, $\alpha_2 = 0$ and $h_0 = 1$: its branches emerge at a fixed upwards angle from the trunk and are fairly straight. Tree (c) uses $\alpha_1 = R$, $\alpha_2 = 0$ and $h_0 = 1$, where R denotes a random angle between 30 and 120 degrees; i.e., each node on the trunk has a different random direction. The branches point in different directions, but are fairly straight. Finally, tree (d) uses $\alpha_1 = 15^o$, $\alpha_2 = -5^o$ and $h_0 = 5$.

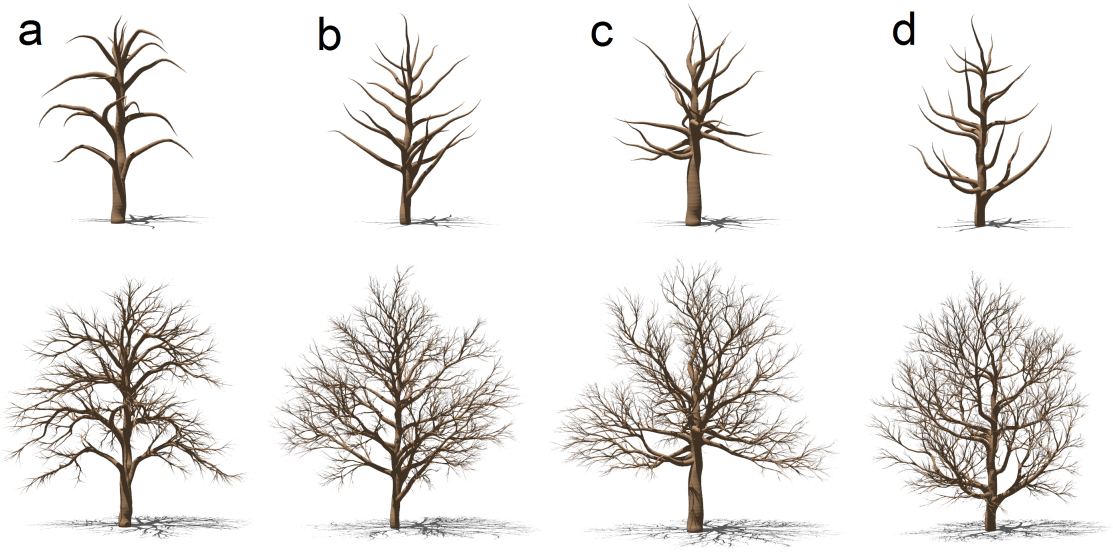


Figure 5.5: Trees generated using different rotation settings. Above: results after two iterations; below: results after five iterations.

Figure 5.6: Left: the structure after one iteration; right: after four iterations.

Guiding vectors can also be set using a rule that incorporates global information. For example, we might set the rotation angle based on the angle to the central axis:

$$\alpha = \theta \times \sqrt{\beta/\pi} \tag{5.2}$$

where β is the angle, in radians, between the node's position vector and the up vector, with the root at the origin; θ is the parameter controlling the amount of rotation. In Equation 5.2, we used the square root: dividing by π ensures that the argument ranges from 0 to 1, and the square root then changes most quickly when close to the central axis, with more slowly changing angles when further away. A result obtained with $\theta = 25^o$ is shown in Figure 5.6, resembling an elm tree. Note that Equation 5.2 is used only for the first iteration; later iterations use $\alpha_1 = 30^o$, $\alpha_2 = 0$ and $h_0 = 1$.

Instead of using incremental rotations to get guiding vectors, we can set vector fields directly. Existing tools for vector field construction can be deployed, such as field primitives [83]. The advantage is that the vector field can be directly manipulated. The disadvantage is that it may be difficult to produce the highly divergent fields that yield natural-looking trees. We have found it more straightforward to produce believable trees using incremental transformations. Nonetheless, direct generation of fields of guiding vectors is a possibility.

We directly generated the vector field to produce the trees in Figure 5.7: for examples (a), (b), and (c), we use a helical field, a rotational field about the z-axis, and a field tracing

upwards along spherical surfaces, respectively. The branches approximately follow the field directions. Note, however, that the branches are not streamlines; the guiding vectors alter edge weights, but the edges do not conform to vector field directions, and branches can deviate from field directions whenever doing so produces a shorter path. Accordingly, singularities in the vector field are not as grave a concern as they would be for streamline-based tree structures.

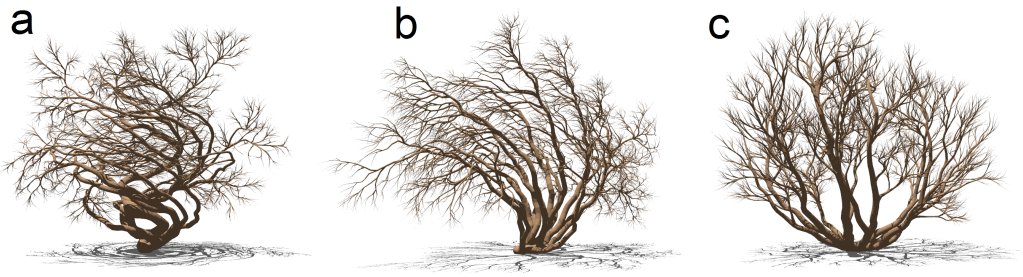


Figure 5.7: Tree models from global guiding vector fields.

5.3.2 Early endpoint placement

Typical trees have a central trunk and a collection of primary branches from which secondary, tertiary, and smaller branches spring. In MIG, the skeleton of a typical tree can be created in the first iteration: branches/paths from the root are restricted in a radius-controllable cylinder when growing upward to reach the endpoints in the crown volume. A narrow cylinder contains few nodes and edges. Paths have a big chance to take same edges in order to pass through the narrow tunnel and reach the endpoints, merging into a single trunk. On the contrary, in a wide cylinder, the paths have small chances to merge: they tend to spread out and go directly towards the endpoints when different edge directions are available, leading to a shrub-like structure. In MGV, we use a bounding volume for endpoint placement. The tree architecture is decided by guiding vectors and not by the bounding volume. This aspect is different from MIG. More details appear below.

Because each endpoint creates a separate path, to create shrub-like trees, we use more endpoints in the first iteration. The overall tree shapes are influenced by the bounding volume for endpoint placement. Figure 5.8 shows examples: all three trees use the same parameters, but their first-generation endpoints are placed within different volumes. We

have found half-ellipsoids to be useful shapes, both because they approximate some commonly seen trees and because they offer a convenient parameterization. To create a typical tree, we dedicate the first iteration to the trunk by using a single endpoint to create a central trunk. Endpoint placement in the second iteration then becomes critical in defining the overall shape of the tree; we choose endpoints randomly in a Poisson disc distribution within a bounding volume approximating the tree crown. Later iterations will fill out the tree from this basic shape. Figure 5.9 shows two examples; the first iteration places the trunk, the second defines a crown shape with an ellipsoid, and later iterations add further definition to the tree shape. Note that in both above two figures, paths are not restricted to stay inside the bounding volume.

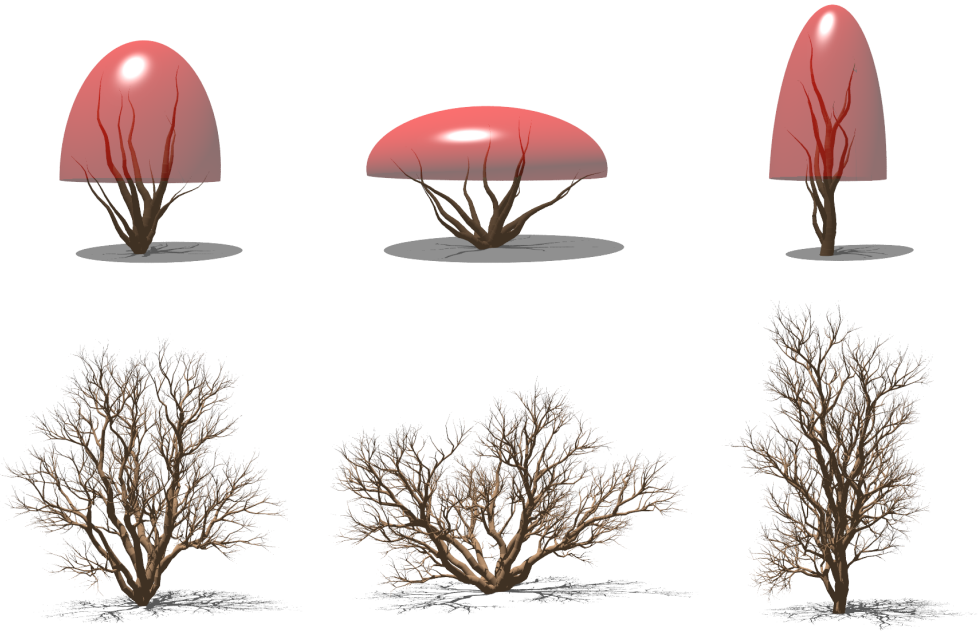


Figure 5.8: Tree models from different bounding volumes.

5.3.3 Late endpoint placement

While the earliest endpoints govern the general tree shape, endpoint placement in later iterations determines the further development of the tree. We must strike a balance between preserving the initial shape and fleshing out the preliminary structure of the early iterations. We place later generations of endpoints on a surface extrapolated from the tree so far.

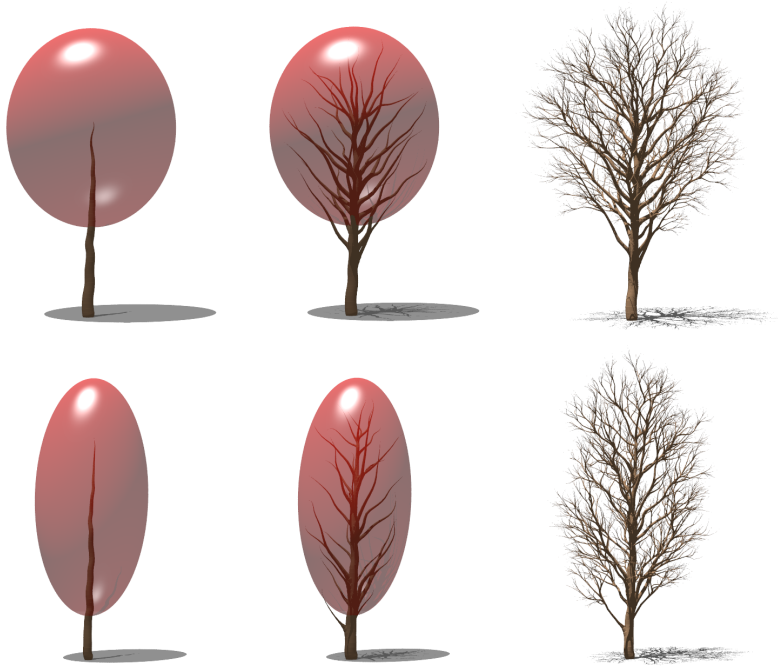


Figure 5.9: Trees with trunks: bounding volumes control endpoint placement in the second iteration.

The surface is defined by the set of nodes exactly k hops from the existing tree, for some parameter k. We use the heuristic that trees grow outwards from the root, and hence order the surface nodes by the number of edges from the root; we then take the subset of the surface consisting of its outermost fraction, for some proportion t. Endpoints are selected from among the nodes on this surface subset.

By using hop count rather than Euclidean distance, we exploit the irregular graph structure to obtain an irregular surface shape. The exclusion zone for the initial inflated tree approximates a hemisphere but has some variation, as illustrated in Figure 5.10. In this figure, the inflated tree appears on the left; the exclusion zone and the remaining surface are shown on the right. Endpoints for the next iteration are chosen from the portion of the surface lying outside the exclusion zone. Further variation is also possible, for example by using graph distance with random weights rather than strictly counting edges. We opted to parameterize the size of the exclusion zone as a proportion of nodes, making it scale-free.

Figure 5.11 shows trees created with different exclusion zone sizes. With $t = 0.3$, endpoints are concentrated in the outer regions. At $t = 0.6$, more of the tree shape is

available for endpoint placement; we consider this to strike a good balance and have used it for most of our examples. At $t = 1$ there is no exclusion zone – all nodes on the initial surface are potential endpoints – and the entire tree is covered with small twigs.

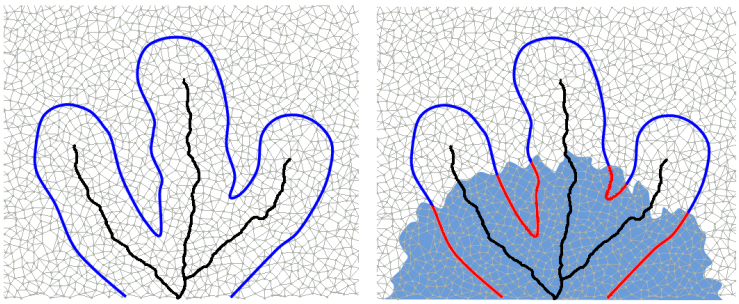


Figure 5.10: Left: the surface of nodes k hops to the source. Right: the exclusion zone.

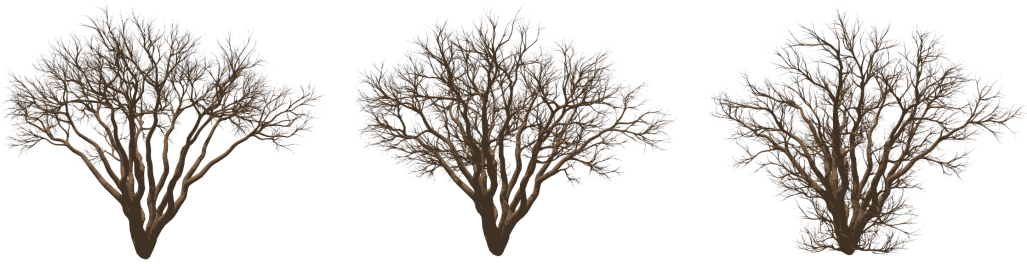


Figure 5.11: From left to right: $t = 0.3$, $t = 0.6$, $t = 1$.

Additional policies can augment the above method to synthesize particular tree shapes. One possibility is to allow only endpoints that are lower than their source nodes; the resulting branches appear to hang down, as in an example shown in Figure 5.12.

5.3.4 Endpoint density

Our synthetic trees can be made more sparse or full by adjusting the density of endpoints. Figure 5.13 shows three trees; from left to right, the trees have approximately 1000, 2000 and 4000 endpoints. Since there is a one-to-one correspondence between endpoints and branches, the models with more endpoints are more filled out. Figure 5.14 shows a tumbleweed model with highly dense strands of branches created using the same method for Figure 5.7(c) with 35,000 endpoints, which we had not seen created by other methods.

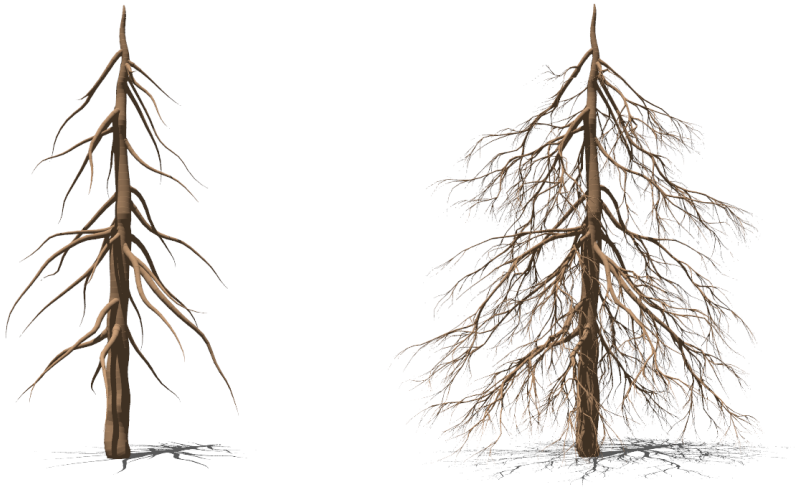


Figure 5.12: A tree with branches growing downward. Left: result from two iterations; right: five iterations.

Figure 5.13: Tree models with approximately 1000, 2000, and 4000 endpoints from left to right.

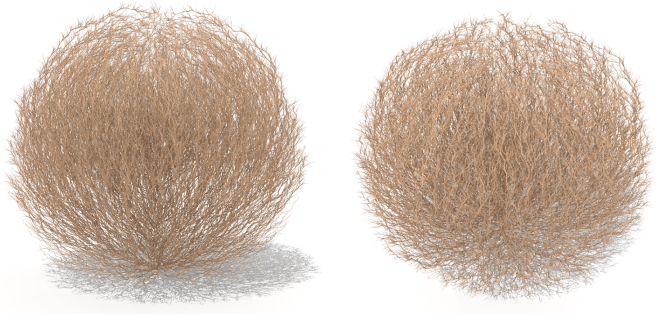


Figure 5.14: A tumbleweed model viewed from two different angles.

5.4 Summary

In this chapter, we presented MGV, a design of the path planning modeling system. MGV emphasizes the module of setting edge weights. In MGV, guiding vectors are used to set edge weights, thus inducing least-cost paths in the graph to conform to the guiding vector field. By specifying guiding vectors we can systematically control the intermediate-scale architecture of trees. Compared with BCM and MIG, MGV can create more realistic tree models.

The work in this chapter has been published as follows:

– L. Xu and D. Mould. Procedural tree modeling with guiding vectors. *Computer Graphics Forum*, 34(7), 2015

Chapter 6

Results and Discussion

In this chapter we show more results from MIG and MGV with emphasis on MGV. First we show some MGV tree models and compare them with real tree structures. Second we explore the parameter space of MGV and show the resulting models. Third we compare MGV trees, MIG trees, and trees from selected previous methods. In addition to these results, we will discuss timing and construction data for selected models, the factors involved in the creation of new trees, and the strengths and limitations of MIG and MGV.

6.1 Methodology of evaluation

We evaluate our results using the idea of authorial subjective evaluation (ASE) proposed by Mould [38]. ASE is an evaluation scheme for non-photorealistic images, in which researchers list specific goals and visually inspect resulting images by identifying the objective characteristics. It is applicable when the following three considerations are met.

1. "When the overall goal is to reproduce a distinct style for which examples are available..."

2. "When there is no relevant objective function..."

3. "When the goal is to make images that interest an audience..."

ASE is proposed for non-photorealistic rendering (NPR) research. NPR creates images "made to resemble artistic images within a known historical style or made using traditional media", "with artistic or aesthetic intent but lacking a pragmatic purpose" [38]. ASE is used to evaluate artistic images, not realistic tree models. However, we can borrow the idea of ASE for our evaluation, due to the following similar considerations used in our work.

As for the first consideration, the overall goal of our work is to create realistic models. Examples of real trees are available. We can visually compare our results with real tree examples for evaluation.

As for the second consideration, we do not have an objective function and it is also not feasible to make an objective function for tree modeling. This problem has been discussed by Prusinkiewicz [72] who has worked on the modeling of plant forms for more than 30 years. He pointed out the potential problems of using an objective function – "An easily measurable characteristic may turn out to be irrelevant; on the other hand, a feature that eludes precise definition or measurement may be of central importance". In his work, he suggested using visual inspection for evaluating tree models: "Lacking a formal measure of what makes two patterns or forms (such as trees) look alike, we rely on visual inspection while comparing the models with the reality" [58]. The same concern is also stated by Runions [66]. Visual inspection, an effective but simple method, has been used for evaluating tree modeling since the earliest synthetic trees in the 1980s. Unfortunately, there is no objective function for evaluating tree models. Although some researchers proposed precise measurements [15, 73] where a distance measure incorporates shape, geometry, and structure, these measurements are used to inspect the difference/similarity between two trees, not for evaluating the quality of a tree model. They are not applicable to our work.

Mould's third criterion is that the images must contain some inherent interest for an audience: they should be appealing on their own, not for any pragmatic purpose. Computer graphics usually has entertainment-related applications, and our tree modeling is no exception; intended applications include films and computer games, in which synthetic trees will serve as an element of the virtual environment. In such cases, the audience will admire and appreciate the trees rather than analyzing them, just as an audience for artistic images admires the images. We do not suggest that our trees are art. Nonetheless, for some applications we are interested in their overall visual impact on the audience, and hence evaluating tree quality using ASE is appropriate.

In our evaluation method, we compile a list of tree characteristics based on real tree structures and visually inspect whether the desired tree properties are acquired or not, by showing a range of results with discussions on the objective characteristics. Our evaluation method depends heavily on visual inspection; the use of the philosophy from ASE makes

visual inspection more substantial: we can look for specific characteristics in the results to decide whether we captured the desired important tree features.

6.2 Results

6.2.1 Examples of trees

In the following we show models of elm, oak, and spruce trees. They are familiar and iconic trees. In our images, the elm and the oak have some similar characteristics: a wide crown, a short trunk, and a few primary branches. Their intermediate-scale architectures are different. The elm tree has recognizable curving branches, while the oak tree has wide-spreading gnarled branches. The spruce has a distinct global shape and intermediate-scale architecture: it is generally cone-like, and has a tall central trunk and side branches along the trunk. We use the elm model, the oak model, and the spruce model to show the strength of MGV in controlling the global shape and the intermediate-scale architecture of trees.

We want to find lists of key properties that can describe appearances of these trees. We will try to evaluate our synthetic trees based on whether they possess the characteristics in the lists. We examined professional references about trees, and Natural Resources Canada (NRC) [45–47] and Weeks et al. [82] give textual descriptions of elm, oak, and spruce trees. Their descriptions are for tree identifications, and we believe they capture important features of trees. Although they also give details about other factors such as leaves and flowers, we are only concerned about those related to tree structures.

NRC describes a tree structure with respect to the crown shape, the branch shapes, and how the trunk splits. An elm tree has a trunk that "divides into a few large, upright limbs and many outwardly fanning branches" [45], and a crown that is "graceful, spreading, vase- or umbrella-shaped" [45]. Oak trees have different trunk and crown features: the trunk is "often branch-free for two-thirds of its height" and "distinct well into the crown" [46], and the crown is "composed of large branches" and "many wide-spreading gnarled and twisted side branches" [46]. A spruce tree is distinct for its "broadly conical" global shape. The branch shapes are "bushy, generally horizontal, sometimes sloping downward in the lower part of the crown" [47] and the branch tips are "gradually upturned".

Another source of information is the book *Native Trees of the Midwest: Identification,*

Wildlife Values, & Landscaping Use by Weeks et al. [82]. Weeks et al. also consider global shape and branch shape important factors that make a tree distinct. In their description, elm trees are "scattered and easily recognized from a distance by their tall, clear trunks, open, spreading crowns, and graceful, drooping limbs that form a vase shape"; oak trees have "short trunks with broad, rounded crowns"; spruce trees have "a form that resembles a cone with gaps between the large limbs".

We summarized NRC and Weeks et al.'s descriptions into the key points shown in the bulleted lists in the following comparisons between our models and real trees. We use these points to evaluate the specific tree types that we modeled. Next, we discuss our elm, oak, and spruce models, in that order.

Figure 6.1: Left: a real elm; right: our elm model. The photo comes from Flickr.

We compare our elm model with a real elm in Figure 6.1. We are interested in the following aspects in the real elm.

- an overall vase-like shape
- a straight trunk that splits into a few strong and long primary branches
- primary branches that are initially close to each other and later widely spread out

Our tree possesses a hemispherical crown supported by multiple thick primary branches developed from a short trunk. In our model, the primary branches further from the central axis are less straight than those in the center. They grow upward initially then gradually fan out, and slightly bend upward at a distance of about 2/3 of their lengths to the root. The curving branches and the overall vase-like shape of our model look similar to the real tree. Of course, our model is not perfect. Its branches are little wavier than real branches,

because the paths follow the irregular graph. Despite of this kind of small-scale differences, we judge our model generally resembles the real elm.

Figure 6.2 shows the comparison of the same tree geometry rendered in two different ways. We render the left tree by setting the radii of branch segments according to their distances to the root (as described in section 4.3). We render the right model using the pipe model [67]. From our observation, in the left tree the tapering of primary branches is more noticeable, resembling that of the real elm in Figure 6.1. The right tree has a short thick trunk more distinct than that of the left tree; this aspect is more similar to the real elm.

Figure 6.2: Left: our elm model rendered using the method in section 4.3; right: our elm model rendered using the pipe model [67].

Figure 6.3: Left: a real oak tree; right: our oak tree model. The photo comes from Flickr.

A comparison of our model with a real oak tree is shown in Figure 6.3. We are interested in the following aspects in the left real oak tree.

- a wide crown and a short thick trunk
- multiple primary branches originating from the same place on the trunk
- wide-spreading gnarled branch shapes

Our model has a wide crown in a half-ellipsoid shape that takes the top 2/3. A short straight trunk that supports the crown tapers gradually and finally splits into some primary branches at about 1/4 up the height. Branches are gnarled and twisted, filling the crown. From our visual inspection, our model looks very similar to the left oak tree.

Figure 6.4: Left: a real spruce tree; middle: a cluster of spruce branches; right: our spruce tree model. The photos come from Flickr.

In Figure 6.4, we show our model compared with a real spruce. We are interested in the following characteristics.

- a conical crown that gradually tapers to the top
- clusters of branches and big gaps in between
- branch tips gradually upturned

Although spruce trees have distinct twig patterns where most twigs attach to parent branches in a symmetric way, as shown in the middle example, we do not create this level of detail.

Our model has a conical crown supported by a tall and straight trunk. Large primary branches attach to the trunk and grow upward. Small branches attach to each primary branch, forming clusters. There are gaps between branch clusters, such as those close to the trunk, but the drawback is that the shapes of clusters and gaps do not have regular patterns as in the real spruce. In our model, child branches grow around the parent branch, not on a fan-like surface as real branches do. Although in our judgement the global shape

and the architecture of our model resemble the real tree, our method is not successful in modeling other features in a real spruce tree.

In addition to the above tree models, we would like to show our tumbleweed model. Tumbleweed has dense branches, making it difficult for particle tracing methods because of the possible crossing of resulting branches. It is also a challenge for data-driven methods due to the occlusion of dense branches. Our tumbleweed model shows the benefit of using least-cost paths in a unified graph: the dense branches do not cross.

Figure 6.5: Left: a real tumbleweed; right: our tumbleweed model. The photo comes from Flickr.

Figure 6.5 compares our tumbleweed model with a photo of real tumbleweed. We are interested in the following aspects of the real tumbleweed.

- a spherical global shape
- dense strands of branches
- each strand of branches curving toward the central axis

Our model has a spherical shape filled with dense strands of branches. Each strand of branches is hierarchical, composed of a long branch attached with short twigs. Branches curve toward the central axis. We judge the overall structure also demonstrates heterogeneity similar to the left structure: some strands are short such as those in the lower part; some are long, originating from the root and extending to the top.

In the above examples, we show the ability of MGV to create some example trees. We expect to create a wide range of trees by varying parameters. Next we will show the robustness of MGV to create trees that have similar structures and trees that have distinct global shapes and intermediate-architectures.

6.2.2 Robustness of MGV

The elements of our tree modeling algorithm – graph shape, placement of graph nodes and endpoints, and settings of guiding vectors – provide a wide range of results. We can create a series of similar trees with fixed parameters, or different trees by varying parameters.

Figure 6.6: A series of similar trees generated with fixed parameter settings but different random choices.

Fixed parameters

We begin with a set of trees from fixed parameter settings. The random placement of graph nodes and endpoints means that similar but distinct trees can be generated by rerunning the algorithm with the same parameters. We used the parameters for Figure 5.5 (c) to make additional ash trees, shown in Figure 6.6. The trees are recognizably similar: all tree crowns have approximately same dimensions; each tree has a rounded top, a straight central trunk, and many side branches; all have a similar density of branches, clusters of branches, and natural gaps in between. An observer could identify them as belonging to the same virtual

species. The sequence demonstrates the robustness of our method; we display the first six results we generated, none of which could be deemed a failure case in our judgement.

Variation of parameters

The parameter space for our method is smooth. Intermediate settings produce intermediate trees, i.e., a small change of the parameter settings will not lead to a large difference. We can smoothly vary the parameter settings to create a series of trees. Figure 6.7 shows trees obtained by varying bounding volumes and guiding vector rotations. These trees were created with half-ellipsoids as bounding volumes. From left column to right column, the half-ellipsoid bounding volumes have aspect ratios of 1.5, 0.9, 0.5, and 0.3. From top row to bottom row, θ takes on values of 14^o, 20^o, 26^o, and 32^o to set rotation angle α of guiding vectors. The distributions of twigs and the branch densities are similar among all trees, but the overall shapes differ; there is a clear and smooth progression from one to the next.

Varying the parameters more widely can create more trees, some of which are shown in Figure 6.8. The trees have distinct global shapes and intermediate-scale architectures. Some trees have a wide crown such as tree (m); some trees are slim and tall such as tree (n). Some have a central trunk such as tree (e), and some are shrub-like such as tree (f). Branch shapes also vary: such as branches of tree (j) that curve upward, and branches of tree (c) that are horizontal. Each tree in the figure has an interesting structure, showing the versatility of MGV. Details of the parameter settings are shown in Table 6.1.

Figure 6.9 shows a detailed view of a shrub-like tree. The model has a few endpoints in the first iteration for the primary branches. We select endpoints higher than their source node to create the branches growing upward. The tree has a desirable combination of detail and plausibility.

6.2.3 Comparisons to trees from other methods

Next we compare our tree models with previous tree synthesis algorithms: Neubert et al.'s data-driven method [48], self-organizing method [52], and BCM [87,88]. We pick Neubert et al.'s method and the self-organizing method because of their appealing results. We also compare trees from MIG, MGV and BCM, which share the idea of using least-cost paths, to show the strengths of MGV.

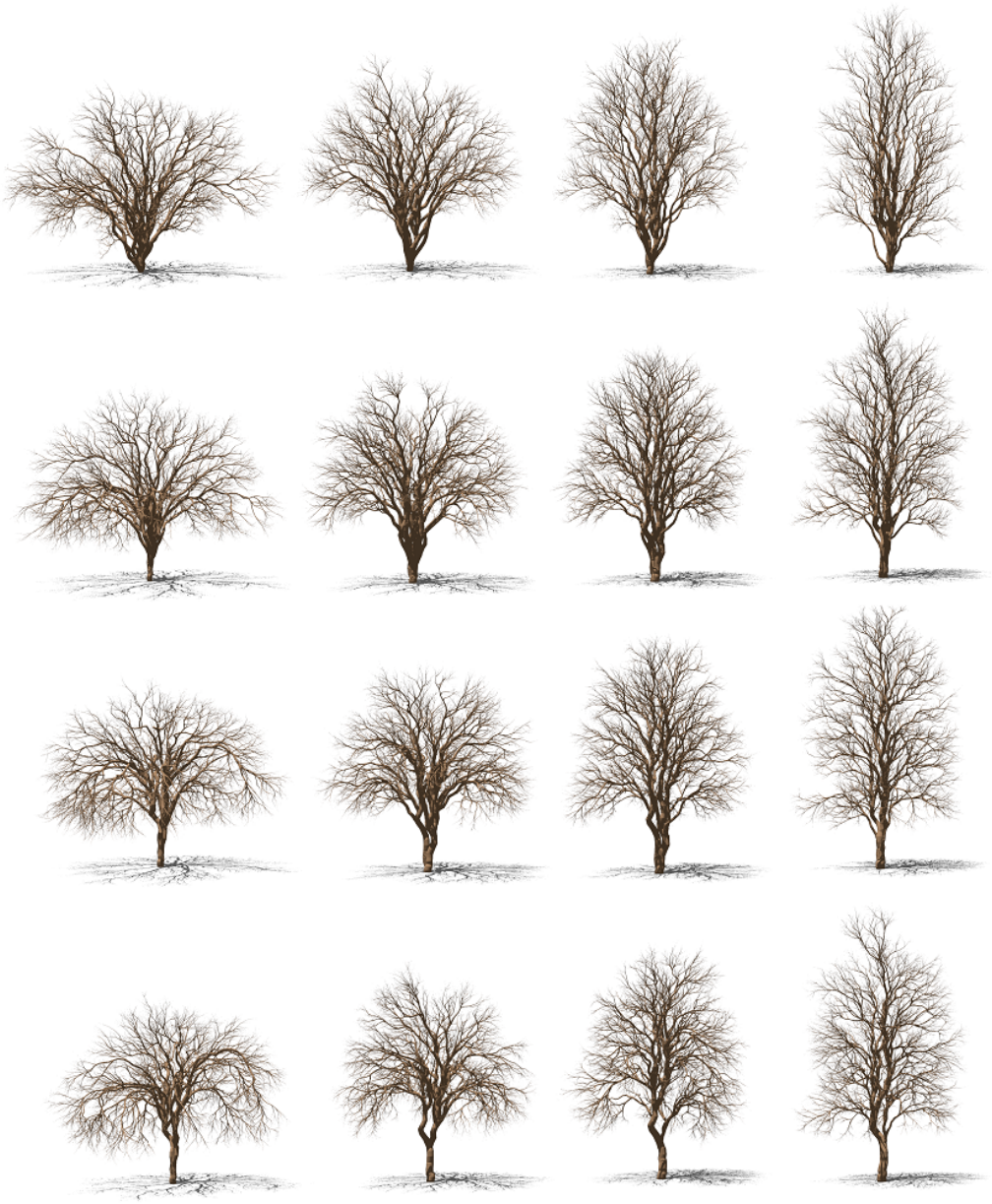


Figure 6.7: A series of trees with different crowns and branch shapes.

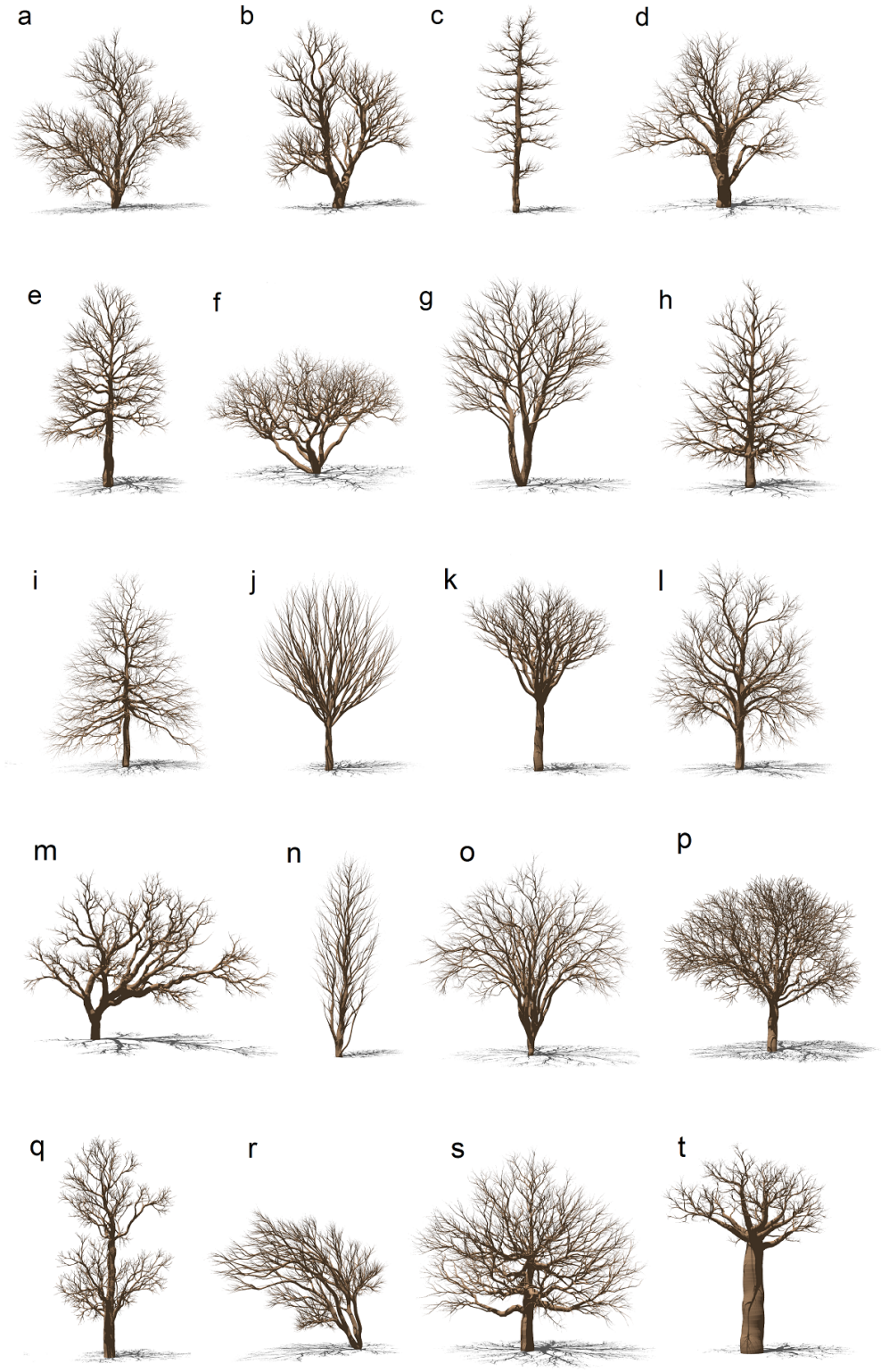


Figure 6.8: A variety of different types of trees.

Figure 6.9: A shrub-like tree.

Tree	Iteration *i*	Parameter Settings
(a) and (b)	1	$\alpha = 2$
	2 - 4	$\alpha_1 = 30$, $\alpha_2 = 0$, $h_0 = 1$
(c) and (h)	2	$\alpha_1 = 90$ in (c) or 60 in (h), $\alpha_2 = 0$, $h_0 = 1$
	3	$\alpha_1 = 30$, $\alpha_2 = 0$, $h_0 = 1$
(d), (f) and (g)	1	$\alpha = 6$ in (d) or 0 in(f) and (g); in tree (f) endpoints are higher than their source point.
	2 - 4	$\alpha_1 = 30$, $\alpha_2 = 0$, $h_0 = 1$
(e) and (i)	2	$\alpha_1 = 180 \times (1 - N.y/H)$, $\alpha_2 = 0$, $h_0 = 1$, where H is the tree's height, N.y is node N's height
	3, 4	$\alpha_1 = 30$, $\alpha_2 = 0$, $h_0 = 1$
(j)	2	$\alpha_1 = 40$, $\alpha_2 = 0$, $h_0 = 1$
(k), (p) and (t)	2	$\alpha_1 = 40$ in (k), or 60 in (p) and (t), $\alpha_2 = 0$, $h_0 = 1$
	3, 4	$\alpha_1 = 30$, $\alpha_2 = 0$, $h_0 = 1$
(l)	2	$\alpha_1 = 30$, $\alpha_2 = -10$, $h_0 = 3$ for the upper portion; $\alpha = 8$ for the lower portion
	3, 4	$\alpha_1 = 30$, $\alpha_2 = 0$, $h_0 = 1$
(m)	1	$\alpha = 4$, the source is at the bottom left of graph
	2	$\alpha_1 = 50$, $\alpha_2 = 0$, $h_0 = 1$
	3, 4	$\alpha_1 = 30$, $\alpha_2 = 0$, $h_0 = 1$
(n)	2	$\alpha_1 = 20$, $\alpha_2 = -5$, $h_0 = 2$
	3, 4	$\alpha_1 = 30$, $\alpha_2 = 0$, $h_0 = 1$
(o)	2	$\theta = 20$
	3, 4	$\alpha_1 = 30$, $\alpha_2 = 0$, $h_0 = 1$
(q)	2	$\alpha_1 = 40$, $\alpha_2 = 0$, $h_0 = 1$; endpoints are selected in two half-ellipsoids.
	3, 4	$\alpha_1 = 30$, $\alpha_2 = 0$, $h_0 = 1$
(r)	1	Using the vector field in Figure 5.7 (b)
	2 - 4	$\alpha_1 = 30$, $\alpha_2 = 0$, $h_0 = 1$
(s)	1 - 4	same to Figure 6.5, but with a wider bounding volume

Table 6.1: Parameters for the models in Figure 6.8

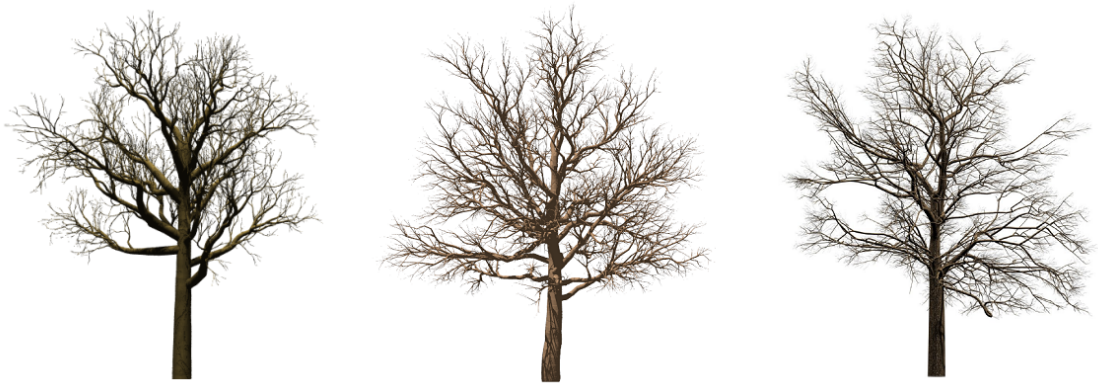


Figure 6.10: Left: a tree from MIG; middle: a tree from MGV; right: a tree created by Neubert et al. [48].

Figure 6.10 compares a tree from MIG, a tree from MGV, and a tree from Neubert et al.'s image-based particle tracing method [48]. We used the right tree as a visual reference to create the MIG tree and the MGV tree.

We first compare the MIG tree and the MGV tree. Although both trees possess realistic hierarchical structures, the MGV tree has a more complicated structure with more subtle features of twigs and gaps than the MIG tree. The MIG tree has more visible clusters of branches due to the paths concentrated in subgraphs, leading to a less natural appearance. In addition, side branches of the MIG tree all point upward, having less diversity in branch orientations than the MGV tree. The MIG tree does not have branches pointing downwards, which exist in the MGV tree.

We compare Neubert et al.'s tree and the MGV tree. Neubert et al.'s tree is highly realistic, corresponding closely to the photograph on which it is based. The MGV tree has a straight central trunk supporting a crown. The height of the trunk and the dimensions of the crown in both trees are similar. They also have similar intermediate-scale architectures: in both trees, side branches attach to the trunk and point to different orientations; upper branches point upward and lower branches are more horizontal. In our judgement, the visual quality of the MGV tree is comparable to Neubert et al.'s tree, but our algorithm did not actually require any image data.

Figure 6.11 shows a model from MIG, a model from MGV, and Palubicki et al.'s self-organizing tree [52]. All three trees look similar, due to the use of the right tree as a reference to create the other two trees.

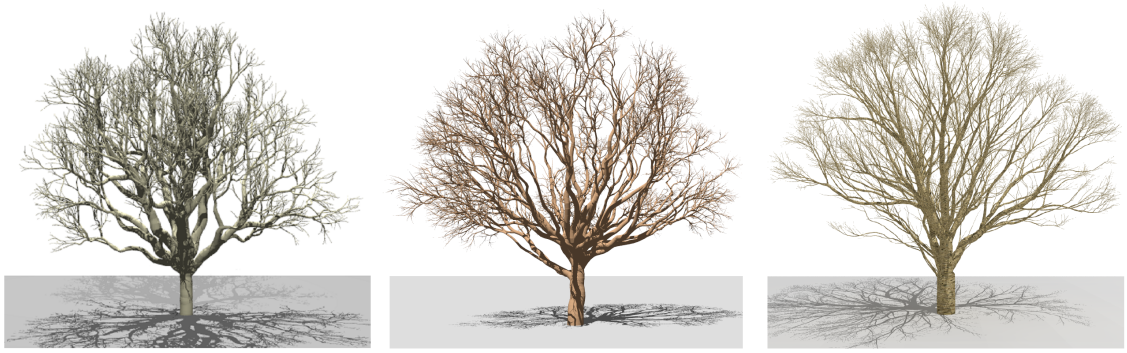


Figure 6.11: Left: a tree from MIG; middle: a tree from MGV; right: a self-organizing tree model [52].

The overall shape of the MIG tree is similar to that of the MGV tree, but the MIG tree has more visible clusters of branches, showing a stronger hierarchy than the MGV tree. Although both the MIG tree and the MGV tree have irregular outer contours, the MGV tree has a higher resolution and denser branches, possessing more small-scale features of twigs and irregular gaps in between.

Both the MGV tree and the self-organizing tree have a short trunk, a hemispherical crown, and outwardly fanning branches. The central hierarchy of the two trees is not strong. A drawback of the MGV tree is the visible crookedness of the branches as they follow the irregular graph. The progression of branches is not as natural as in the self-organizing tree. In addition, the twigs in the outer contour of the MGV tree are not detailed as in the self-organizing tree. In our judgement, despite these small-scale differences, the overall effect of the MGV tree is comparable to the self-organizing tree.

In Figure 6.12 we compare the results from BCM [87, 88], MIG and MGV. All three trees are elm models.

The left model from BCM has a simple tree skeleton. The primary branches grow straightly upward and do not curve as much as real elm branches do. The MIG tree contains different scales of branches – several thick primary branches, a few drooping slimmer branches, and many twigs in the outer contour of the crown far from the root. We judge the MIG tree is more realistic than the BCM tree.

The crown of the MIG tree is filled with branches extending out, due to the use of cone-like subgraphs, but the overall flow of branches is not as smooth as that in the MGV

tree. In our opinion, the MGV tree has a better depiction of branch shapes than the MIG tree: it has long and disjoint primary branches that spread out and slightly curve upward, resembling real elm branches. In addition, the MGV tree has denser branches and more detailed features than the MIG tree, making it more realistic and appealing.

Figure 6.12: Left: an elm model from BCM [87, 88]; middle: a model from MIG; right: a model from MGV.

Figure 6.13: Left: an oak tree from MIG; right: an oak tree from MGV.

We compare oak tree models from MIG and MGV in Figure 6.13. From our visual inspection, both trees have similar global shape: a half ellipsoid crown supported by a short and straight trunk. Each has a hierarchical structure with different scales of branches, similar to the real oak in Figure 6.3. The MGV tree has a more pleasing unity of shape, believable sweep of branches, and overall a cleaner and more attractive appearance.

In the above comparisons, we judge the results from MGV are comparable to those from the notable methods. Compared with the results from BCM and MIG, the models from MGV are more realistic, due to the power of MGV in controlling the branch shapes.

6.2.4 Timing

In above sections we showed the results created by specifying parameters. Specifying parameters for a tree is affected by the following factors: knowledge about geometry, expected quality of results, and how complex the objective tree structure is. Users are required to have geometric knowledge to create a tree model by specifying the properties including crown shapes, branching patterns, and tree hierarchies. The progress is also affected by how users evaluate the quality of tree models subjectively. If they are not satisfied with a result, they may change the parameter settings, and run the program and reinspect the new result. This process may repeat for some iterations until they judge the result is good enough. It is possible that some users may need longer time for more iterations of parameter adjustments but others may need shorter time and less iterations. The complexity of the objective tree structure also affects the speed to create a tree. A complicated tree may take longer time for specifying the rotations of guiding vectors and bounding volume properties than a tree with simpler global shape and branch shapes.

Next we describe the process to set parameters for MGV. The setting of a parameter is simplified by its concrete nature. Our experience is that we can select parameters for desired tree structures within approximately one hour, most devoted to the design of tree architectures by specifying the following.

- Number of early endpoints, to control growth pattern (i.e. a tree has a typical structure with a central trunk or a shrub-like structure with a few primary branches)
- Position and shape of the bounding volume to place early endpoints, to control crown shape
- The rotation angle of guiding vectors, to govern branch shapes

To decide proper parameter settings for a desired tree shape, we first tentatively estimate the specifications for the above aspects. Since we have already provided the parameter settings for many trees and also demonstrated the creation of new trees through varying the parameter space smoothly, we can make an estimate based on the existing settings. We

then run the algorithm with the estimated parameter settings and inspect the resulting tree structure. We can make further adjustments of parameters if the current settings do not produce a desired shape.

For most trees, we can leave the distribution of twigs unchanged by using the default settings of the size of exclusion zone and density of later endpoints. We do not vary the distribution of twigs, because from our observation it is not a key factor that affects the appearance of a tree. We also do not vary tree hierarchy, because with the default number of iterations, i.e. 4, our trees can have different scales of branches such as the trunk, primary branches, and twigs. The quality of our trees is also comparable to trees from existing methods. Of course we can increase the number of iterations to create more complicated tree structures if desired.

Once parameters had been selected, most results from MGV were created in a graph of 150,000 nodes and have between 4000 and 8000 endpoints. Each tree can be generated in under ten seconds by running the algorithm. The trees of Figure 6.6, for example, required around 9.5 s each: about 6 s to construct the Yao graph, and another 2 s for path planning, with endpoint selection and other minor tasks making up the balance. Note that the graph does not necessarily need to be regenerated for each tree: in an interactive setting, the same graph could be reused for each experiment.

Timing information and statistics for more trees from MIG and MGV are shown in Table 7.1. Our timing figures are with respect to a desktop computer with 2.8 GHz CPU and 3 GB RAM. Since our development of MGV included some code optimizations that were not included in the MIG implementation, MGV could probably be twice as fast as MIG. Overall, MGV is capable of generating a high-quality tree model at a high speed.

6.3 Advantages and limitations

In this thesis we presented two designs of path planning tree modeling system – MIG and MGV. MIG is focused on graph creation, and MGV is focused on setting edge weights. Next we discuss these two designs and point out their advantages and limitations.

MIG offers certain benefits. First, it is completely automated, with no need for a database of measurements or exemplars, and no modeler intervention beyond specifying parameters. Second, tree shape is governed by a few parameters and by direct spatial and

Tree		# of graph nodes	# of endpoints	timing
Figure 6.9: comparison with Neubert et al.'s tree	MIG	440k	1.5k	56s
	MGV	150k	7.5k	9s
Figure 6.10: comparison with self-organizing tree	MIG	200k	9.0k	20s
	MGV	600k	19.0k	21s
Figure 6.11: three elm models	MIG	85k	4.0k	5s
	MGV	150k	5.5k	9s
Figure 6.12: two oak models	MIG	220k	5.0k	17s
	MGV	200k	4.0k	10s

Table 6.2: Timing and construction data for selected models.

geometric considerations, such as the shape of the bounding geometry for early endpoint placement. Unlike the unfamiliar parameters sometimes seen in biologically-inspired algorithms, the concrete parameters we use simplify parameter selection for generating novel tree types. Third, MIG is versatile and can generate a wide range of tree models by specifying the parameters. Due to the random placement of graph nodes and endpoints, once parameter settings have been chosen, the same settings can be used to create numerous trees of the same general type by rerunning the algorithm.

MIG also has some drawbacks. Due to the lack of systematic control over branch shapes, the synthetic branches do not curve in a way resembling real tree branches. Another problem is that, because subgraphs do not communicate with each other, it is possible that paths in different subgraphs may intersect.

In addition to the benefits that MIG possesses, MGV has some extra advantages. First, path planning in a unified graph guarantees the absence of self-intersections in the final tree. Second, our suggested parametric assignment of guiding vector direction is flexible and controllable, while the path planning element enforces a unity over the resulting tree; although particle tracing techniques can be equally flexible, they do not naturally create a tree, and when their flexibility is exploited, they demand extra attention to ensure that the result is a tree without self-intersections and other defects. A key strength of our method is its ability to create irregular, gnarled trees, an area where particle tracing struggles. In addition, the iterative construction process does not constrain the branch connections, creating

a recognizable but loose hierarchy.

MGV has some limitations as well. We have found our approach to incremental control over guiding vectors to be useful and powerful, but recognize that not all users will like creating vector fields in this way. Endpoint placement is controlled indirectly, a consequence of our decision to automate the tree synthesis process. The process needs tentative experiments to find proper parameter settings. The settings that deviate far from our suggested settings may lead to a wrong structure. Figure 6.14 shows an example: a wrong rotation angle α leads to a structure that does not resemble a real tree.

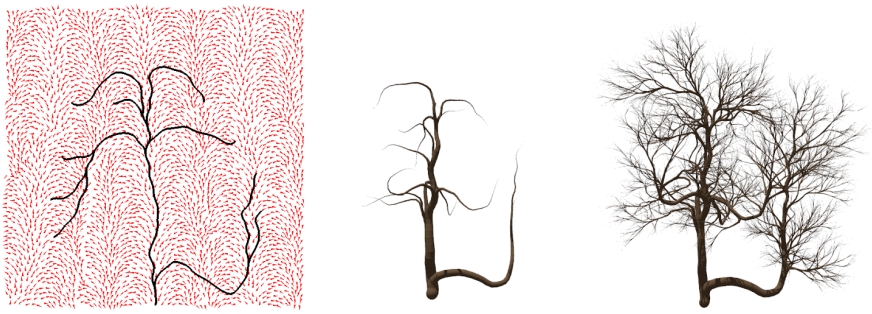


Figure 6.14: A failure example that does not resemble a real tree.

Overall, MGV provides comparable results to existing methods, and its algorithmic underpinnings are simpler. The guiding vectors offer a subtle yet powerful variation on the general path planning technique used previously for tree synthesis and reconstruction, allowing branch tropism to be included in this framework. By improving on the results of earlier graph-based procedural tree modeling to the point where it is competitive with other procedural techniques, we have enlarged the space of available algorithms for tree modeling.

Chapter 7

Abstract Tree Modeling with Magnetic Curves

7.1 Introduction

Realism and abstraction exist on two ends of a continuum in the art world [21]. In this chapter, we will go beyond realism, and look at abstract trees. We are interested in the abstract trees in Gustav Klimt's art style. In Klimt's painting, shown in Figure 1.3, the tree has a hierarchical structure. The repetitive spiral patterns of branches fill most of the painting. The strong appeal comes from the branch shapes and their spatial arrangement.

To create abstract trees that have similar characteristics to Klimt's tree, the primary task is to generate the curvilinear branches. Using curves for realistic plant modeling has been explored in computer graphics with particle tracing methods [48, 65]. Prior particle tracing methods use the trajectories of particles as branches, however lack systematic control over curvature and cannot create our desired spiral shapes.

Here we propose "*magnetic curves*", a particle tracing method with the ability to create curvature-controllable branch shapes. The basic idea of the algorithm is to trace charged particles in magnetic fields. By varying the charge and the magnetic field, we can generate trajectories of particles that have continuously varying curvature. The trajectory curves are tree branches. More details appear below.

7.2 Algorithm

7.2.1 Overview

A particle with mass m and charge q moving with velocity $\vec{v}$ in a magnetic field $\vec{B}$ experiences the Lorentz Force $\vec{F} = q\vec{v} \times \vec{B}$ causing circular motion [32]. The trajectory of particle is perpendicular to $\vec{B}$. In our method, we restrict the particle to move in the xy plane, and set the magnetic field in the z direction. Future states of the particle are obtained using numerical integration; in our implementation, we used forward Euler integration. For a

constant magnetic field $\vec{B}$ and a constant q, the particle traces out a circle in the xy plane. When we change the particle charge and the magnetic field, the motion track of the particle is no longer a circle; instead, the particle can trace out some interesting curves.

The basic idea to create a tree that has our desired characteristics is as follows. To create spiral branches, we vary charge q with time t. The continuously changed q will cause continuously changed curvature. We can create spiral shapes by continuously increasing q. To create a branching structure, at a same position we release two particles having charges of opposite sign, in order to make their trajectories wind to opposite directions. To create space-filling patterns, we fill the space with different sizes of curves and prevent them from crossing each other. Next we introduce more details.

7.2.2 Varying curvature

When a charged particle moves in the xy plane, it experiences a force from the magnetic field. A constant acceleration a perpendicular to the direction of motion will cause circular motion with radius r, where $r = v^2/a$. Combined with $a = F/m = qvB/m$, we can see that for fixed velocity and magnetic field, higher values of q give smaller values of radius r. If we increase q, the radius of the curve will decrease accordingly. Similarly, with a decreasing magnitude of q, the resulting curve will have an increased radius. If q changes continuously, the curvature of the path will also vary continuously. Note also that if we instantaneously change the sign of q while preserving its magnitude, the direction of acceleration will be instantaneously reversed, leading to a curve that winds in the opposite direction.

Specific relationships between $q(t)$ and resulting curves are shown in Figure 7.1. The red point is the initial position. In the first row, the left image shows the charge q decreasing with time t according to the function $q(t) = t^{-\alpha}$. Here we use $\alpha = 0.7$. Other functions are also possible. The right image shows the resulting curve growing with an increasing radius. In the second row, $q(t)$ is a triangular wave with amplitude 1 and mean 0.5. Again, the right image is the resulting curve; the point where the charge drops to 0 is visible at the location on the curve where it abruptly straightens (no charge means the particle will travel in a straight line). In the third row, $q(t)$ is a triangular wave with amplitude 2 and mean 0. Here, reversing the charge makes the curve switch the growth direction up/down, leading

to a wavy shape. In the fourth row, $q(t)$ is a square wave with 0 mean. Because the charge is constant in each half-period, the curve follows a piecewise circular route; whenever the sign of q flips, the circle is reversed. It holds less visual interest than the other graphs because of the large sections of constant curvature.

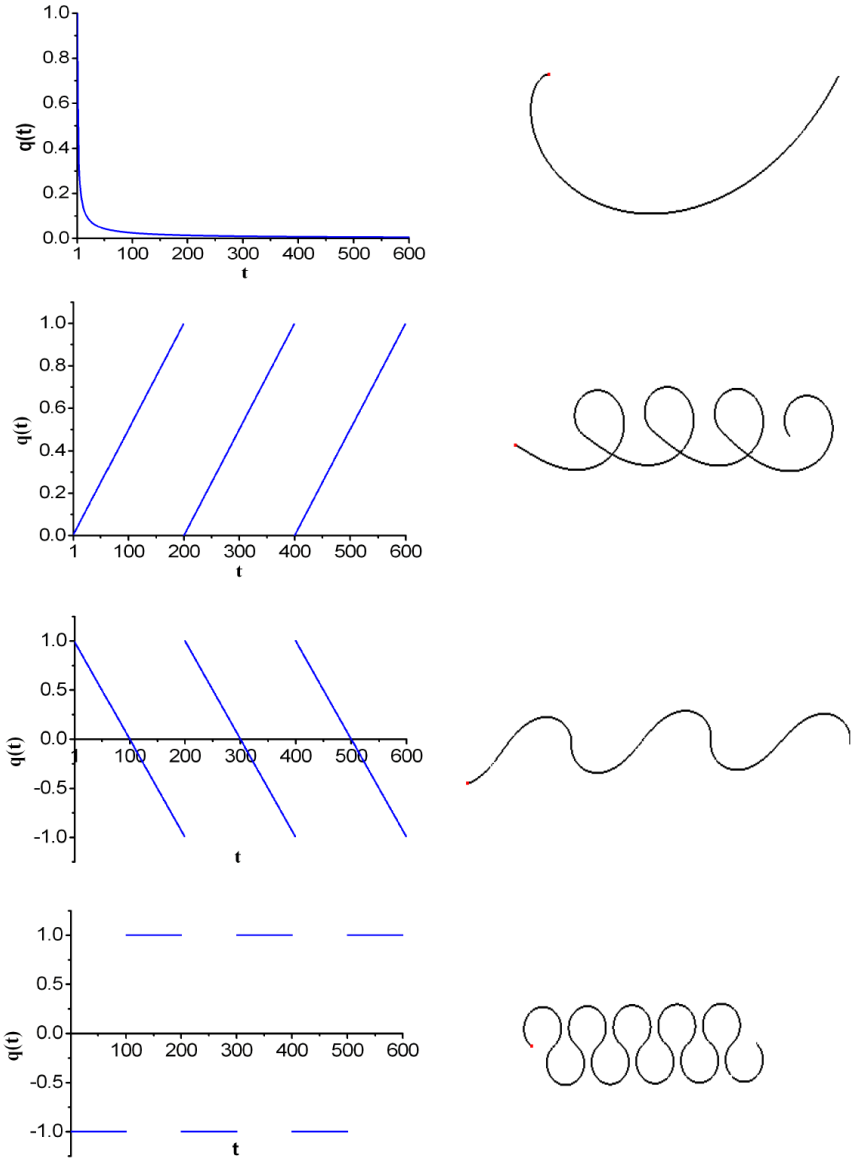


Figure 7.1: Different q(t) (left) and the resulting curves (right)

7.2.3 Branching curves

Instead of tracking one particle to get a single curve, we can use multiple particles to get complicated branching patterns. We suggest releasing new particles at intervals along the initial particle's path. We refer to the initial particle as the "parent particle" of the new particles. Each new particle will be tracked and produce a new curve. The process can be made recursive: we can spawn new particles from the child particle's path if we want. To create an effect reminiscent of vegetation, with curves growing to either side of the current branch, we set the sign of the child particles' charge randomly. Figure 7.2 shows the process. In the left image, we grow a single curve. In the right image, we grow six new curves from six positions at equal intervals along the initial curve.

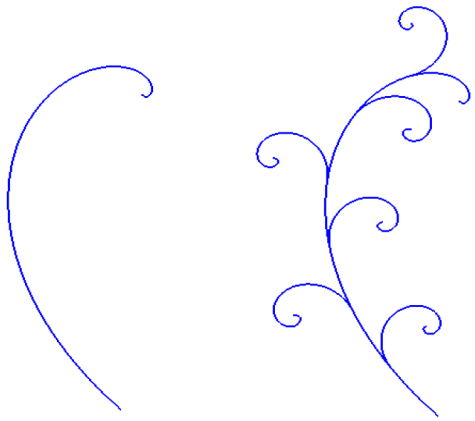


Figure 7.2: Left: a single curve; right: six new curves growing from the left initial curve

7.2.4 Space-filling curves

To create patterns with different sizes of spirals filling the space, we decompose the task into the following two stages: creating different sizes of spirals, and filling space using the obtained spirals. The policy to fill the space is that we first use the largest possible spiral, and then use the smaller spirals to fill the remaining small spaces.

We first use the equation $q(t) = (T-t)^{-\alpha}$ to create spirals, where t is the elapsed time for a given curve and α is a parameter that governs the rate of change of curvature for a curve; T dictates the length of a curve, where a given curve ranges from $t = 0$ to $t = T$. Figure 7.3 shows different curves with different values for T and α. From top to bottom,

T=650, 450, 250, 50; from left to right, α=0.8, 0.75, 0.7, 0.65. Observe that in each column from top to bottom the curve becomes shorter as T decreases; in each row from left to right the spiral is looser as α decreases.

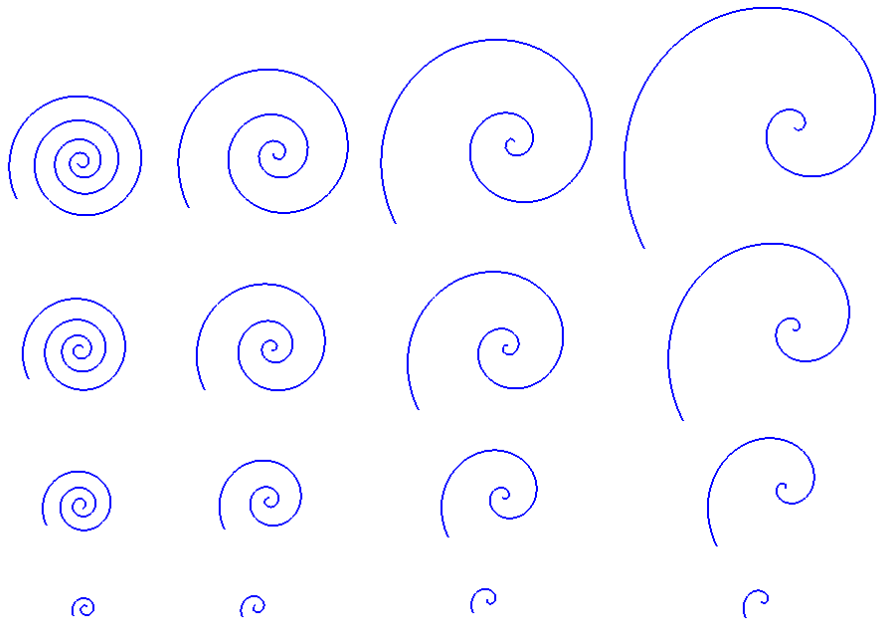


Figure 7.3: Different curves with different T and α.

Next, we use curves grown with four pairs of (T,α): (T_1, α_1), (T_2, α_2), (T_3, α_3), (T_4, α_4), where $T_1 > T_2 > T_3 > T_4$ and $\alpha_1 > \alpha_2 > \alpha_3 > \alpha_4$, to fill a space of size $x_{\max}, y_{\max}$. Here we use the parameters for the curves from top-left to bottom-right in the diagonal of Figure 7.3.

To fill the space with curves, we first release a particle at a position (x_0, y_0). We choose a position located at bottom center. Initially, the curve has parameters T_1 and α_1. We terminate the progress of a particle if either of the following is met: 1) the particle travels beyond the boundary, that is $x > x_{\max}$ or $x < 0$ or $y > y_{\max}$ or $y < 0$; 2) the curve crosses previously drawn curves.

To detect whether a curve crosses other curves we use the following method. We divide the whole space into cells. Each cell stores an initial ID number to indicate that it is unoccupied. If a particle moves into one cell and either the cell is unoccupied or the cell is marked by its parent particle, the particle can survive; otherwise, the curve crosses another curve, leading to a crossing event. A formerly unoccupied cell is marked with the

ID number of the entering particle. Crossing is prohibited; if a crossing event is detected, we remove the markers for the current particle and restart it at its initial position with the next smaller (T, α) pair. If a crossing is still detected when the smallest available (T, α) is used, we abandon that position and move to the next position.

As we observed from Figure 7.3, the curve with largest value of T and α occupies the largest space. If the curve with T_1 and α_1 fails, we restart the particle at its initial position with T_2 and α_2. If T_2 does not fit, we use T_3 and α_3 and then T_4 and α_4. The intent behind doing so is to fill the space with the largest possible curves, while still being able to use the small curves to fill the gaps.

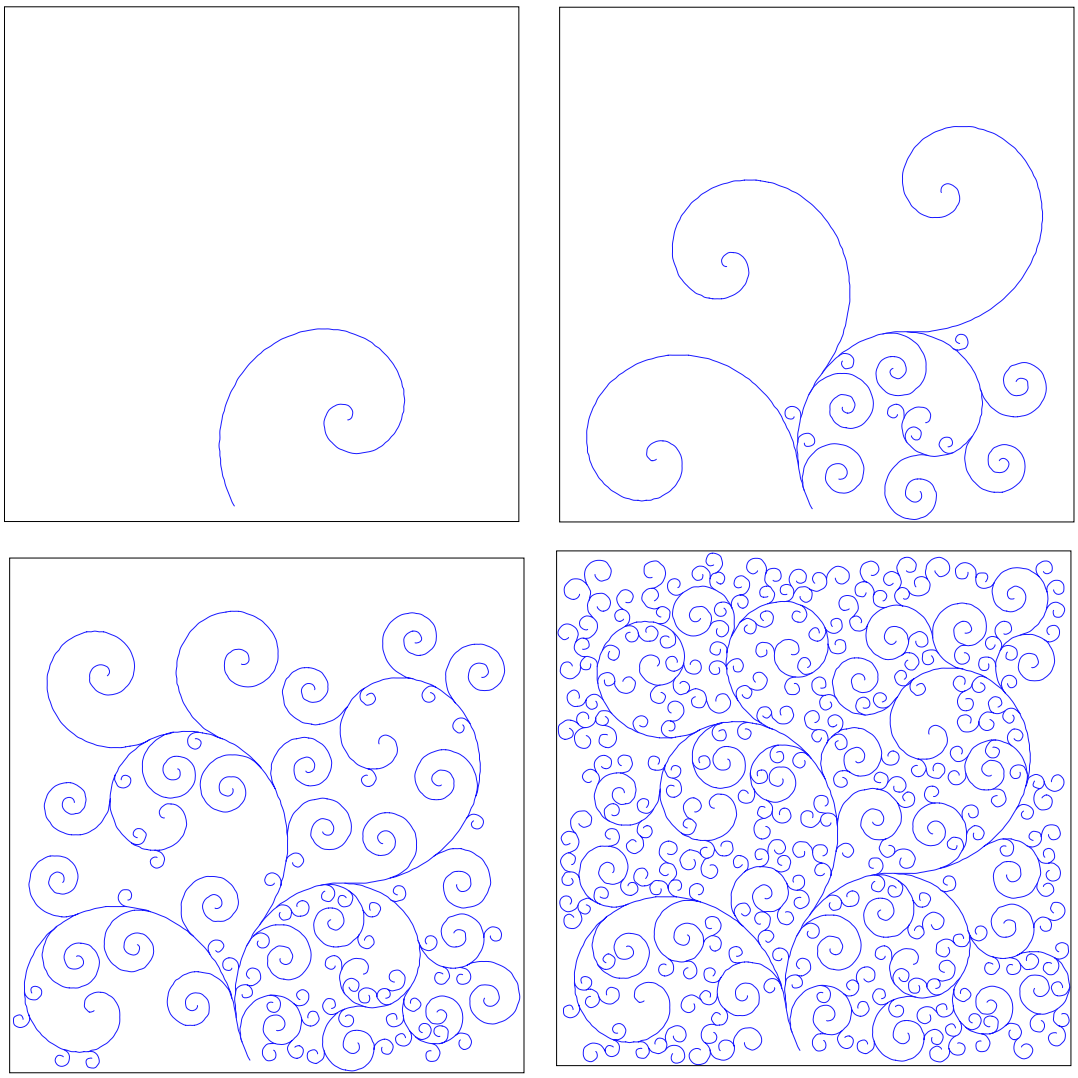


Figure 7.4: Iterations in creating the space-filling curve.

If we release new particles successively, we can obtain a space filled with different shapes of curves. The process is shown in Figure 7.4. The top left image shows the initial curve. The top right image shows different sizes of curves growing from the initial curve. Some curves are relegated to small size to fit in small spaces. The lower left image shows more curves growing from new curves in the top right image. The lower right image shows the final outcome, in which the curves fill the whole space. Note that while the process was demonstrated with a particular set of $q(t)$ equations, it is a general process that can use any set of equations or distributions of parameters.

We also have the option of varying the magnetic field instead of using a constant magnetic field. Next, we show the effect of using Perlin noise to vary B spatially; in particular we take B_z from the noise, leaving the other two components at zero. We first build a lattice of size $m \times m$. For each node in the lattice we obtain a magnetic field value: $B_z = 0.5 * (1 + noise(x/50, y/50))$, where (x,y) is the position of lattice points. We then choose a position in the lattice to release a particle and create a space-filling branching curve as described previously. At each time step, we update the velocity of our particle according to the magnetic field at its current position. The irregular values of B give irregular curves. In Figure 7.5 we visualize the magnetic field and the resulting curves. From this figure, we can observe that curves are much more irregular than the curves in Figure 7.4, while still possessing the same level of continuity of curvature, leading to a more visually appealing structure.

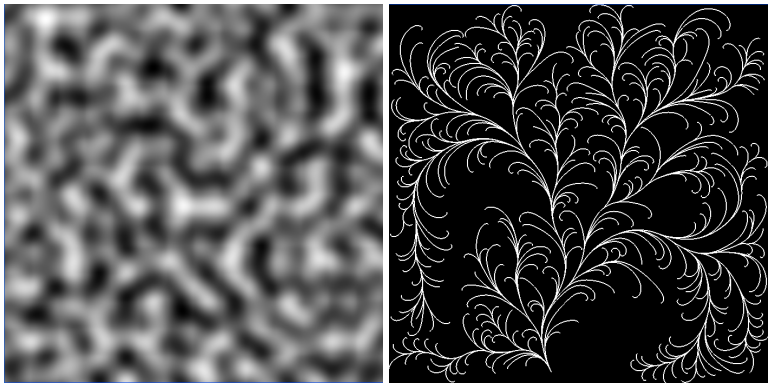


Figure 7.5: Left: the magnetic field with value of Perlin noise; right: the resulting irregular curves

7.3 Results and Discussion

Magnetic curves are well suited to creating stylized depictions of certain classes of objects, including trees, hair, water, and fire. In addition, they can be used to create decoration and appealing abstract forms. In this section, we will show some examples of both representational and abstract structures created with magnetic curves.

Figure 7.6 shows two abstract trees created by varying B spatially using Perlin noise. Each point on the curve is colored by taking one RGB color channel from the local noise value and setting the other two to zero. These are space-filling curves with irregular spatial variation in their curvature, yielding a subjectively pleasing structure.

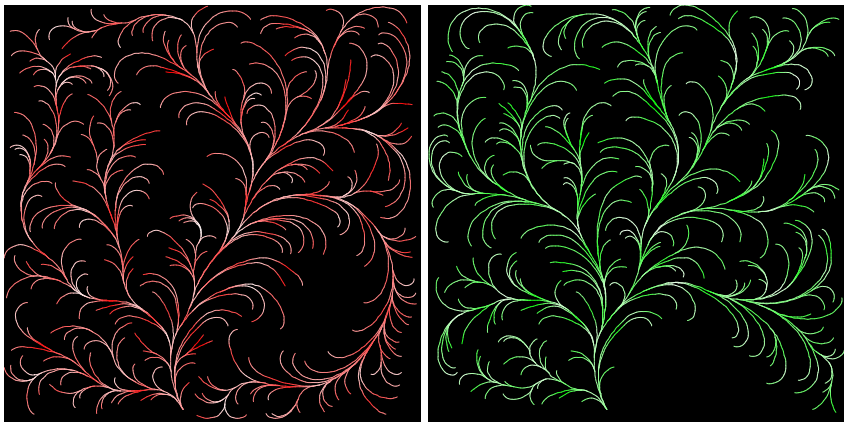


Figure 7.6: Two abstract curvilinear forms.

In Figure 7.7 we show an abstract tree in a similar style to the tree in Figure 1.3. The tree has our desired characteristics: it is composed of spiral branches; it has a hierarchical structure where the trunk and the lower primary branches split into many smaller branches; the space is compactly filled with different sizes of spirals. To achieve the result, we release a particle to grow the main curve. Then we release new particles on the main curve and repeat the process until the space (top 3/4 of the image) is filled with different sizes of curves, while avoiding the curves crossing over each other. We are not intending to exactly reproduce Klimt's tree, but to demonstrate that this kind of stylization can be captured by the magnetic curves. The spirals in "Tree of Life", and present in some of Klimt's other work, are easy to create with magnetic curves.

In addition to abstract trees, other forms have similar curves that can be created using our method. Architectural ornamentation on doors, windows, and arches is common; some

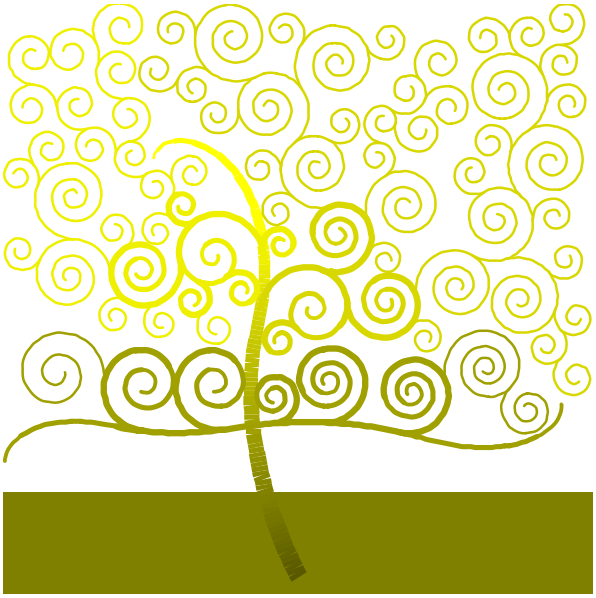


Figure 7.7: Our stylized tree created with magnetic curves.

Figure 7.8: Architectural ornamentation. The window image comes from Richard Marcin and John Wahlert's "The city of Kosice". The door image comes from Cambridge 2000 Gallery.

examples appear in Figure 7.8. Here we use our method to generate two window frames as shown in Figure 7.9. In our results, all patterns are composed of different curves with a background taken from a real glass image. Our results have similar aesthetic symmetric patterns composed of curves with smoothly varying curvature. The crossing of curves increases the beauty of complexity. Since we wanted a symmetrical pattern, we only built the right side of the frame and mirrored it. The difficult part is the design of pattern and finding proper T and α. We use existing functions of $q(t)$ to get different shapes of curves. We tentatively select T and α to make the curve fit in the window area. If the curve from selected T and α is too long and out of the window boundary, we decrease T or change α.

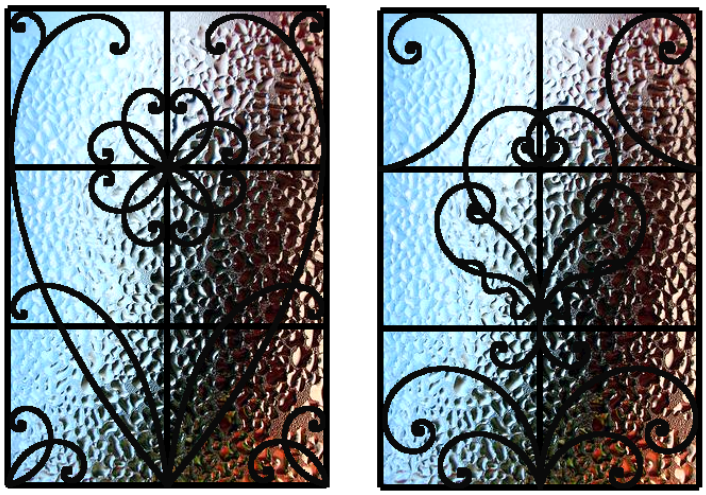


Figure 7.9: Two window frames adorned with curves.

Figure 7.10 shows three different hair styles created by our method, overlayed on a hand-drawn image (also shown). The hair is inspired by the art by Alfons Mucha [18] (as shown in Figure 7.11), and has similar features. The hair is composed of curves with smoothly varying curvature. Each curve has a thickness and the outer contour has a different color, making the overlapping and crossing of curves have a 3D effect. To create each hair style, we grow the curves by releasing particles from points on a hand-drawn "hairline" curve. To obtain the upper right image, we set $q(t) = (T-t)^{-\alpha}$ with large T and small α values; we used different T and α values for different strands, and reversed the sign of q partway along the curve so that the hair would look wavy. In the lower left image, we set $q(t)$ to the sine function to achieve the curve. In the lower right image, we set the curves

growing with different time length T and different α, so the hair is composed of curves with different lengths and curvatures. A closer look at yet another hairstyle can be seen in Figure 7.12; this was also produced using $q(t) = (T-t)^{-\alpha}$.

Figure 7.10: Three hair styles by our method.

We use our method to create the fire image shown in Figure 7.13. We release an initial particle with a charge profile of $q(t) = (T-t)^{\alpha}$ and reverse its charge partway along its trajectory (at $t = T/2$), causing the visible point of inflection. Subsequently, we terminate the particle and spawn a new one when a condition is met; in the depicted fire, the condition was that the x component of velocity was 0. The new particle is given a random velocity with a small uniform angular distribution about the positive x direction. When a sufficient number of repetitions have been performed (say 3 or 4) we start moving down the left side. The overall result is a boundary composed of piecewise continuous curves; the positions

Figure 7.11: Art by Alfons Mucha: inspiration for the hair styles in Figure 7.10.

Figure 7.12: Another magnetic hairstyle.

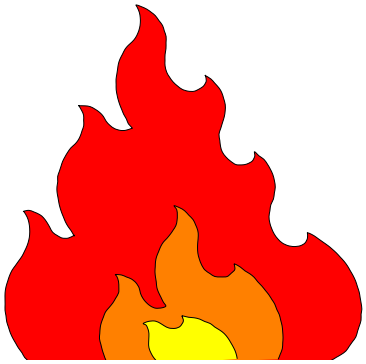


Figure 7.13: Cartoon flames created with magnetic curves.

where the velocity changes happen are the tips of the flames. The resulting flame image is a conventional cartoon flame. A similar process could be used to produce conventional cartoon water waves.

Figure 7.14 shows curves grown in a magnetic field by setting B according to Perlin noise. To create this figure, we tracked a single particle with a fixed charge as it wandered within the image area. Whenever it reached the image boundary, we reversed its velocity, ensuring that it remains within the image area. Every time it reentered the image area, we assigned it a different random color, determined by assigning each RGB channel an independently chosen value from a uniform random distribution. From the figure we can see the different colors of curves which represent different periods of time spent continuously within the image area. While we did not find this image as straightforwardly attractive as the space-filling curve, we confess to a certain fascination with the unpredictable meanderings of the particle and the resulting tangle of curves.

Table 7.1 gives timing results for a 1.8GHz P4 with 512 MB RAM. The process is not particularly demanding, and computer times are only a few seconds per image. Note that the hair timing includes the time needed to load files (the face image and an image containing the hairline to spawn hair particles from). In terms of human time, the process can be time-consuming; we expect that most users lack intuition for controlling curves by modifying charges and magnetic fields. It took on the order of two minutes to find the appropriate parameters for each curve on the window frames in Figure 7.9.

Figure 7.14: Meanderings of a charge in a varying magnetic field.

Image	number of curves	processing time
Space-filling example	312	1.5s
Stylized Tree	72	0.7s
Perlin curves	272	1.0s
Hair	376	4.3s
Window	7	0.05s

Table 7.1: Timing results for some of the magnetic curves.

The main strength of the method lies in its ability to produce constantly and smoothly varying curvature within the well-understood framework of particle systems. The ability to create branching structures by tracing particles released from previous particle's trajectory is also an asset.

One limitation of the method is the difficulty to control the exact curve shapes. This is a limitation held in common with other particle tracing methods. Another limitation of the method is the lack of communication between particles as well as the environment. As we have found in the creation of space-filling curves, we have to design some strategies to avoid the crossing of curves and to prevent curves from growing out of a region. Our current method of tentatively adjusting parameters is not efficient. An obvious limitation of our present implementation is its restriction to 2D. The extension to 3D applications is possible, such as allowing particles to move on a 3D surface.

7.4 Summary

In this chapter, we introduced the modeling of abstract trees in the art style of Gustav Klimt, which has never been explored in computer graphics before. We proposed simulating magnetic forces to create curves with continuously varying curvature. The generation process is fast, and the resulting curves can be used to create appealing abstract forms. Magnetic curves are also a potential tool for creating stylized depictions of classes of objects and phenomena that are well described by curves.

This work has been published as follows:

– L. Xu and D. Mould. Magnetic curves: Curvature-controlled aesthetic curves using magnetic fields. *Proceedings of Computational Aesthetics*, pages 1–8, 2009

Chapter 8

Conclusion

Tree modeling is a popular topic in computer graphics. This thesis presented procedural techniques for modeling realistic trees and abstract trees – the method with iterated graphs (MIG), the method with guiding vectors (MGV), and magnetic curves. Using these methods, we created a variety of tree models that can be used in computer graphic applications such as 3D movies, games, and artistic decorations.

For realistic tree modeling, it is a challenge for previous procedural methods to control the global tree shape and branch shapes at the same time. In this thesis, we presented a realistic tree modeling system based on the idea of path planning: least-cost paths between selected endpoints and a root point form a branching structure. Our objective is to make these structures resemble real trees by controlling their global shapes and intermediate-scale architectures. Towards this objective, we focus on three modules in the system: creating a graph, setting edge weights, and placing endpoints. We tend to achieve the desired controls through designing the modules.

We presented two designs of the modules: MIG and MGV. MIG emphasizes the graph creation, and MGV emphasizes the setting of edge weights. MIG creates a tree in a hierarchical graph. The global shape and the intermediate-scale architecture of the tree can be controlled by specifying the graph properties, such as graph volume and orientations, and by changing the endpoint distribution mechanisms. MGV creates a tree in a unified graph, in which each node has a guiding vector. We set edge weights according to whether the edge aligns with the guiding vector direction, thus controlling the intermediate-scale tree architecture. Both the design in MIG and the design in MGV can create a wide range of tree models. MGV, especially, can create branches that curve in a natural way resembling real tree branches. In our judgement, the resulting models are more realistic than MIG's results, and also comparable to the results from some notable methods such as Neubert et al.'s particle tracing method [48] and self-organizing method [52].

For abstract tree modeling, we presented magnetic curves, a particle tracing method for generating curves with constantly changing curvature. The curves are used to create abstract trees. We also described how to create aesthetic patterns with different sizes of noncrossing spiral curves. Our work is the first exploration of abstract trees in the art style of Gustav Klimt in computer graphics.

In conclusion, the thesis makes the following contributions.

- It proposes using the Yao graph [93] for graph-based tree modeling. We are the first to use the Yao graph for the purpose of tree modeling.
- It proposes a graph construction method and a graph-based tree construction method with iterated graphs. These methods are more elaborate than previous graph-based tree modeling methods.
- It proposes guiding vectors for setting edge weights. Guiding vectors provide effective control over branch shapes, which is a challenge for previous methods.
- It suggests a mechanism for assigning guiding vectors. It is flexible and controllable, allowing users to specify intermediate-scale tree architectures with a few simple rules.
- It contributes magnetic curves, a particle tracing method, to create curvature-controlled curves, which are difficult for previous particle tracing methods.
- It contributes a method for abstract tree modeling. It is the first exploration of Gustav Klimt's art style in computer graphics.

8.1 Future Work

Our future work on realistic tree modeling includes the following possible directions.

– ***Interface to increase user control:*** Adding sketch-based and example-based control over the endpoint distribution and guiding vector field creation would be useful. Alternatively, simulation could be used to create vector fields.

– ***Improvements to the algorithm:*** At present, guiding vectors alter the edge weights unimodally; multimodal edge weight adjustments are possible, so that there can be multiple favored directions. Also, we would like to reduce the memory requirements

of our approach. The fixed resolution of our graph simplifies the implementation, but a hierarchical or variable-resolution graph would reduce the memory usage.

– ***Landscapes:*** For realistic tree modeling, we have concentrated on creating individual trees; it is worth investigating constructing multiple trees simultaneously. Environmental factors, such as daylight, wind direction, and space restrictions could be considered in the tree modeling process.

– ***More applications:*** Graph-based methods have been used primarily for reconstruction, and enlisting guiding vectors to help in that domain would be productive. Our model currently only treats the main structure of the tree, and we would like to investigate phenomena including leaves, bark, fruit, and flowers. Also, using guiding vectors to help with other modeling tasks, such as creating rivers or cracks, is an obvious direction.

For abstract tree modeling, the following directions for future work are possible.

– ***Further exploration:*** We have not yet much explored the possibility of allowing q to vary according to other elements of the particle's state vector. For example, we could have q depend on position, or on the direction of the particle's travel. We demonstrated some position-dependent effects by varying B spatially; more investigation is warranted, considering the success of the results obtained so far. We have showed one method for creating branching curves, but we have not much considered the aesthetics of curve thickness as it relates to branching. Varying curve thickness will add richness to magnetic curves.

– ***More applications:*** Our present implementation is restricted to 2D. The idea of magnetic curves extends straightforwardly to three dimensions, and we hope both to create curves in three dimensions and to create 2D curves on the surfaces of complex structures. Magnetic fields could vary spatially or could be textures on the surfaces of objects. We showed only one stylized tree (after Klimt) but other vegetation, including trees but also vines and leaves, can be depicted also. Indeed, a variety of leaf outlines can be created by adapting our approach for cartoon flames. We showed a few hairstyles; we envision possibilities for also drawing stylized fur, feathers, cloth, and smoke.

In the long term, we will consider creating a complicated virtual world. It may contain various branching natural phenomena such as trees, veins, rivers, cracks, and lightning. Animations such as the growth of trees and the development of cracks would be interesting. We believe this work will contribute to computer-generated movies and games.

Bibliography

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

[2] B.Benes, O.Stava, R.Mech, and G.Miller. Guided procedural modeling. *Computer Graphics Forum*, 30:325–334.

[3] P. Bénard, J. Lu, F. Cole, A. Finkelstein, and J. Thollot. Active strokes: Coherent line stylization for animated 3d models. In *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering*, NPAR '12, pages 37–46. Eurographics Association, 2012.

[4] F. Boudon, P. Prusinkiewicz, P. Federl, C. Godin, and R. Karwowski. Interactive design of bonsai tree models. In *Proceedings of Comput. Graph. Forum*, pages 591–600, 2003.

[5] A. Bucksch, R. C. Lindenbergh, and M. Menenti. Skeltre - fast skeletonisation for imperfect point cloud data of botanic trees. In *3DOR'09*, pages 13–20, 2009.

[6] T. Capizzi. *3D Modeling And Texture Mapping*. Premier Press, 1982.

[7] X. Chen, B. Neubert, Y.-Q. Xu, O. Deussen, and S. B. Kang. Sketch-based tree modeling using markov random field. In *ACM SIGGRAPH Asia 2008 papers*, pages 109:1–109:9, 2008.

[8] S.-W. Cheng, T. K. Dey, and J. Shewchuk. *Delaunay Mesh Generation*. Chapman & Hall/CRC, 1st edition, 2012.

[9] P. de Reffye, C. Edelin, J. Françon, M. Jaeger, and C. Puech. Plant models faithful to botanical structure and development. *SIGGRAPH Comput. Graph.*, 22(4):151–158, 1988.

[10] O. Deussen, P. Hanrahan, B. Lintermann, R. Měch, M. Pharr, and P. Prusinkiewicz. Realistic modeling and rendering of plant ecosystems. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, pages 275–286. ACM.

[11] O. Deussen and T. Strothotte. Computer-generated pen-and-ink illustration of trees. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 13–18, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[12] F. Dietrich. Visual intelligence: The first decade of computer art (1965-1975). *IEEE Computer Graphics and Applications*, 5(7):33–45, 1985.

[13] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[14] P. Eichhorst and W. J. Savitch. Growth functions of stochastic lindenmayer systems. *Information and Control*, 45:217 – 228, 1980.

[15] P. Ferraro and C. Godin. A distance measure between plant architectures. *Annals of Forest Science*, 57:445–461, June 2000.

[16] E. Fiume. *An Introduction to Scientific, Symbolic, and Graphical Computation*. A K Peters/CRC Press, 1995.

[17] F. Han and S.-C. Zhu. Bayesian reconstruction of 3d shapes and scenes from a single image. *HLK '03: Proceedings of the First IEEE International Workshop on Higher-Level Knowledge in 3D Modeling and Motion Analysis*, page 12, 2003.

[18] H. H. Hofstatter. *Art Nouveau: Prints, Illustrations And Posters*. Greenwich House, 1984.

[19] 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(2):331 – 338, 1971.

[20] T. Ijiri, S. Owada, and T. Igarashi. The sketch l-system: Global control of tree modeling using free-form strokes. In *Smart Graphics*, pages 138–146, 2006.

[21] S. S. John. *Journeys To Abstraction: 100 Paintings and Their Secrets Revealed*. North Light Books, 2012.

[22] M. Johnson. *Gustav Klimt: 130 Paintings in Close Up*. CreateSpace Independent Publishing Platform, 2015.

[23] R. D. Kalnins, P. L. Davidson, L. Markosian, and A. Finkelstein. Coherent stylized silhouettes. *ACM Trans. Graph.*, 22(3):856–861, 2003.

[24] M. Kaplan and E. Cohen. Computer generated celtic design. In *Proceedings of the 14th Eurographics Workshop on Rendering*, EGRW '03, pages 9–19. Eurographics Association, 2003.

[25] M. A. Kowalski, L. Markosian, J. D. Northrup, L. Bourdev, R. Barzel, L. S. Holden, and J. F. Hughes. Art-based rendering of fur, grass, and trees. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, pages 433–438, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[26] K. D. Lee and S. Hubbard. *Data Structures and Algorithms with Python*. Springer, 2015.

[27] Y. Li, X. Fan, N. J. Mitra, D. Chamovitz, D. Cohen-Or, and B. Chen. Analyzing growing plants from 4d point cloud data. *ACM Trans. Graph.*, 32(6):157:1–157:10, 2013.

[28] A. Lindenmayer. Mathematical models for cellular interaction in development, parts i and ii. *Journal of Theoretical Biology*, 18:180–315, 1968.

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

[30] Y. Livny, S. Pirk, Z. Cheng, F. Yan, O. Deussen, D. Cohen-Or, and B. Chen. Texture-lobes for tree modelling. *ACM Trans. Graph.*, 30(4):53:1–53:10, 2011.

[31] Y. Livny, F. Yan, M. Olson, B. Chen, H. Zhang, and J. El-Sana. Automatic reconstruction of tree skeletal structures from point clouds. *ACM Trans. Graph.*, 29(6):151:1–151:8, 2010.

[32] F. Lobkowicz and A. C. Melissinos. *Physics for Scientists and Engineers*. W.B.Saunders, 1975.

[33] J. Long and D. Mould. Dendritic stylization. *Vis. Comput.*, 25(3):241–253, 2009.

[34] S. Longay, A. Runions, F. Boudon, and P. Prusinkiewicz. Treesketch: Interactive procedural modeling of trees on a tablet. In *Proceedings of the International Symposium on Sketch-Based Interfaces and Modeling*, pages 107–120, 2012.

[35] T. Luft, M. Balzer, and O. Deussen. Expressive illumination of foliage based on implicit surfaces. In *Proceedings of the Third Eurographics Conference on Natural Phenomena*, NPH'07, pages 71–78, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[36] R. Mech and P. Prusinkiewicz. Visual models of plants interacting with their environment. In *SIGGRAPH96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 397–410, 1996.

[37] I. Millington and J. Funge. *Artificial Intelligence for Games, Second Edition*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2009.

[38] D. Mould. Authorial subjective evaluation of non-photorealistic images. In *Proceedings of the Workshop on Non-Photorealistic Animation and Rendering*, NPAR '14, pages 49–56. ACM, 2014.

[39] Natural Resources Canada. American beech. http://tidcf.nrcan.gc.ca/en/trees/factsheet/25.

[40] Natural Resources Canada. American mountain-ash. http://tidcf.nrcan.gc.ca/en/trees/factsheet/71.

[41] Natural Resources Canada. Black-gum. http://tidcf.nrcan.gc.ca/en/trees/factsheet/329.

[42] Natural Resources Canada. Red mulberry. http://tidcf.nrcan.gc.ca/en/trees/factsheet/439.

[43] Natural Resources Canada. Tulip-tree. http://tidcf.nrcan.gc.ca/en/trees/factsheet/328.

[44] Natural Resources Canada. Water birch. http://tidcf.nrcan.gc.ca/en/trees/factsheet/317.

[45] Natural Resources Canada. White elm. http://tidcf.nrcan.gc.ca/en/trees/factsheet/76.

[46] Natural Resources Canada. White oak. http://tidcf.nrcan.gc.ca/en/trees/factsheet/63.

[47] Natural Resources Canada. White spruce. http://tidcf.nrcan.gc.ca/en/trees/factsheet/38.

[48] B. Neubert, T. Franken, and O. Deussen. Approximate image-based tree-modeling using particle flows. *ACM Transactions on Graphics (Proc. of SIGGRAPH 2007)*, 26(3), 2007.

[49] M. Okabe, S. Owada, and T. Igarashi. Interactive design of botanical trees using freehand sketches and example-based editing. In *ACM SIGGRAPH 2006 Courses*, SIGGRAPH '06, 2006.

[50] P. E. Oppenheimer. Real time design and animation of fractal plants and trees. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '86, pages 55–64, New York, NY, USA, 1986. ACM.

[51] J. O'Rourke. *Computational Geometry in C (Second Edition)*. Cambridge University Press, 1998.

[52] W. Palubicki, K. Horel, S. Longay, A. Runions, B. Lane, R. Měch, and P. Prusinkiewicz. Self-organizing tree models for image synthesis. *ACM Trans. Graph.*, 28:58:1–58:10, 2009.

[53] S. Pirk, T. Niese, O. Deussen, and B. Neubert. Capturing and animating the morphogenesis of polygonal tree models. *ACM Trans. Graph.*, 31(6):169:1–169:10, 2012.

[54] S. Pirk, T. Niese, T. Hädrich, B. Benes, and O. Deussen. Windy trees: Computing stress response for developmental tree models. *ACM Trans. Graph.*, 33(6):204:1–204:11, 2014.

[55] S. Pirk, O. Stava, J. Kratt, M. A. M. Said, B. Neubert, R. Měch, B. Benes, and O. Deussen. Plastic trees: Interactive self-adapting botanical tree models. *ACM Trans. Graph.*, 31(4):50:1–50:10, 2012.

[56] J. L. Power, A. J. B. Brush, P. Prusinkiewicz, and D. H. Salesin. Interactive arrangement of botanical l-system models. In *Proceedings of the 1999 symposium on Interactive 3D graphics*, pages 175–182, New York, NY, USA, 1999. ACM.

[57] P. Prusinkiewicz. Graphical applications of L-systems. In *Proceedings of Graphical Interface 86*, pages 247 – 253, 1986.

[58] P. Prusinkiewicz. Modeling and visualization of biological structures. In *In Proceeding of Graphics Interface*, pages 128–137, 1993.

[59] P. Prusinkiewicz, M. Hammel, R. Mech, and J. Hanan. The artificial life of plants. *Artificial life for graphics, animation, and virtual reality,SIGGRAPH 95 Course Notes*, 7:1:1 – 38, 1995.

[60] P. Prusinkiewicz and J. Hanan. Lindenmayer systems, fractals and plants. *Lecture Notes on Biomathematics*, 1989.

[61] P. Prusinkiewicz, M. James, and R. Měch. Synthetic topiary. *Computer Graphics*, 28:351–358, 1994.

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

[63] L. Quan, P. Tan, G. Zeng, L. Yuan, J. Wang, and S. B. Kang. Image-based plant modeling. *ACM Trans. Graph.*, 25:599–604, 2006.

[64] A. Reche, I. Martin, and G. Drettakis. Volumetric reconstruction and interactive rendering of trees from photographs. *ACM Transactions on Graphics (SIGGRAPH Conference Proceedings)*, 23:720–727, 2004.

[65] W. T. Reeves. Particle systems - a technique for modeling a class of fuzzy objects. *ACM Trans. Graph.*, 2:91–108, 1983.

[66] A. Runions. Modeling biological patterns using the space colonization algorithm, 2008.

[67] A. Runions, B. Lane, and P. Prusinkiewicz. Modeling trees with a space colonization algorithm. In *Eurographics Workshop on Natural Phenomena*, pages 63–70, 2007.

[68] J. Ruskin. *The Elements of Drawing*. Dover Publications, 1971.

[69] T. T. Sasada. Drawing natural scenery by computer graphics. *Comput. Aided Des.*, 19(4):212–218, 1987.

[70] D. S.Ebert, F. Musgrave, D. Peachey, K. Perlin, and S. Worley. *Texturing and Modeling, A Procedural Approach (Third Edition)*. Elsevier Science,USA, 2005.

[71] F. Y. Shih. *Image Processing and Pattern Recognition: Fundamentals and Techniques*. Wiley-IEEE Press, 2010.

[72] O. Stava, S. Pirk, J. Kratt, B. Chen, R. Mch, O. Deussen, and B. Benes. In search of the right abstraction: The synergy between art, science, and information technology in the modeling of natural phenomena. *Art @ Science*, pages 60–68, 1998.

[73] O. Stava, S. Pirk, J. Kratt, B. Chen, R. Mch, O. Deussen, and B. Benes. Inverse procedural modelling of trees. *Computer Graphics Forum*, 33(6):118–131, 2014.

[74] T. Strothotte, B. Preim, A. Raab, J. Schumann, and D. R. Forsey. How to render frames and influence people. *Comput. Graph. Forum*, 13(3):455–466, 1994.

[75] T. Strothotte and S. Schlechtweg. *Non-photorealistic Computer Graphics: Modeling, Rendering, and Animation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[76] J. O. Talton, Y. Lou, S. Lesser, J. Duke, R. Měch, and V. Koltun. Metropolis procedural modeling. *ACM Trans. Graph.*, 30:11:1–11:14, 2011.

[77] P. Tan, T. Fang, J. Xiao, P. Zhao, and L. Quan. Single image tree modeling. *ACM Trans. Graph.*, 27:108:1–108:7, 2008.

[78] P. Tan, G. Zeng, J. Wang, S. B. Kang, and L. Quan. Image-based tree modeling. In *International Conference on Computer Graphics and Interactive Techniques archive, ACM SIGGRAPH 2007*, 2007.

[79] B. van Wyk and P. van Wyk. *How to Identify Trees in Southern Africa*. Random House Struik, 2007.

[80] R. Wang, Y. Yang, H. Zhang, and H. Bao. Variational tree synthesis. *Computer Graphics Forum*, 33(8):82–94, 2014.

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

[82] S. S. Weeks, H. P. Weeks, and G. R. Parker. *Native Trees of the Midwest: Identification, Wildlife Values, & Landscaping Use*. Purdue University Press, 2005.

[83] J. Wejchert and D. Haumann. Animation aerodynamics. *SIGGRAPH Comput. Graph.*, 25(4):19–22, 1991.

[84] J. Wither, F. Boudon, M.-P. Cani, and C. Godin. Structure from silhouettes: a new paradigm for fast sketch-based design of trees. *Comput. Graph. Forum*, 28(2):541–550, 2009.

[85] M. T. Wong, D. E. Zongker, and D. H. Salesin. Computer-generated floral ornament. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, pages 423–434, New York, NY, USA, 1998. ACM.

[86] H. Xu, N. Gossett, and B. Chen. Knowledge and heuristic-based modeling of laser-scanned trees. *ACM Trans. Graph.*, 26, 2007.

[87] L. Xu and D. Mould. Modeling dendritic shapes - using path planning. In *GRAPP (GM/R)*, pages 29–36, 2007.

[88] L. Xu and D. Mould. Constructive path planning for natural phenomena modeling. In *3IA 11th International Conference on Computer Graphics and Artificial Intelligence*, pages 59–69, 2008.

[89] L. Xu and D. Mould. Magnetic curves: Curvature-controlled aesthetic curves using magnetic fields. *Proceedings of Computational Aesthetics*, pages 1–8, 2009.

[90] L. Xu and D. Mould. A procedural method for irregular tree models. *Computers and Graphics*, 36(8):1036–1047, Dec. 2012.

[91] L. Xu and D. Mould. Synthetic tree models from iterated discrete graphs. In *Proceedings of Graphics Interface*, pages 149–156, 2012.

[92] L. Xu and D. Mould. Procedural tree modeling with guiding vectors. *Computer Graphics Forum*, 34(7), 2015.

[93] A. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. Technical report, Stanford University, 1977.

[94] C. I. Yessios. Computer drafting of stones, wood, plant and ground materials. In *Proceedings of the 6th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '79, pages 190–198. ACM, 1979.

[95] R. C. Zeleznik, K. P. Herndon, and J. F. Hughes. Sketch: An interface for sketching 3d scenes. In *ACM SIGGRAPH 2006 Courses*, SIGGRAPH '06. ACM, 2006.

[96] X. Zhang, H. Li, M. Dai, W. Ma, and L. Quan. Data-driven synthetic modeling of trees. *Visualization and Computer Graphics, IEEE Transactions on*, 20(9):1214–1226, Sept 2014.