Generating 3D Tree Skeletons From Video

Lei Zeng

Bachelor of Science in Computer Science with Honours
The University of Bath
May 2015

Generating 3D Tree Skeletons From Video

Submitted by: Lei Zeng

COPYRIGHT

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Bachelor of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Signed:

Abstract

We present a probabilistic approach for automatically generating three-dimensional (3D) tree skeletons from videos. The system generates a 3D skeleton from a two-dimensional (2D) skeleton which is created based on information collecting from the input video. Our approach combines global and local constrains to ensure an optimal 3D structure. Further more, Von Mises distribution is used while generating new 3D skeletons from initial model. This means users could create potentially infinite similar but not identical 3D skeletons based on the initial video.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to express my gratitude and appreciation to all those who gave me the possibility to complete this project. A special thanks to my final year project supervisor, Dr. Peter Hall, for offering me an opportunity to take part in this project. Thanks for his help, stimulating suggestions and encouragement.

I would also like to thank Dr. Chuan Li for his valuable suggestions and tips on this project. It helped to coordinate my project in right path.

For the past 32 weeks, I have enjoyed the final year project so much. I dreamed to be a computer game developer since I was a child. With the invaluable experience that I have gained from this project, I am one step closer.

Chapter 1

Introduction

Trees are general objects which appear frequently among movies and video games. There are variety tools available for modelling 3D trees. Most of existing tools require users to have knowledge of 3D modelling to build tree skeleton and apply the suitable texture (*issue 1*). Usually two or more experts will be needed to model a 3D tree (from Li et al. (2011)). This also involves massive user interactions (*issue 2*). On the other hand, people have different requirements for various usages of trees. Sometimes we need to model exactly the same tree (*issue 3*) when we want to collect the accurate data about tree growth in a period of time; if we are trying to create a forest, we then will need a large amount of trees from the same species, which means they are similar but not identical to each other (*issue 4*). Overall, tree modelling remains a time consuming process (*issue 5*).

In this paper we propose a system for creating 3D tree skeletons using an approach that is almost entirely automatic. In order to create a model, users only need to outline the tree in the first frame of initial video (solves *issue 2*), the system then produce a full 3D model (solves *issue 1*). The data collected from video could be served as an example to automatically generate new 3D tree skeletons of the same species, potentially an infinite number of unique trees with similar appearance can be created (solves *issue 4*). In practice, the whole process takes less than one minute to produce an plausible 3D skeleton (solves *issue 5*).

1.1 Previous work

There are currently three categories of tree modelling summarized by Neubert et al. (2007): rules-based generation, interactive modelling and image-based production.

Rules-based generation such as L-systems (Lindenmayer, 1968) is used to describe the behaviour of plant cells and to model the growth processes of plant development. The core concept of L-systems is rewriting. The process is successively replacing parts of an initial object using a set of rewriting rules. Deussen and Lintermann (2006) proposed a

procedural approach to generate new trees from an initial state. System use computer-assisted methods to simulate the natural growth processes. Talton et al. (2011) developed an optimization method to produce L-system rules. Given a high-level specification, the algorithm computes a production from the grammar that conforms to the specification, and then generate a 3D model by optimizing over the space of possible productions from the grammar.

Rule-based systems could help to produce a various species of realistic trees (solves *issue 3*). However, it requires users to understand how the rules are applied (*issue 1*, *issue 2*). Usually high performance computers are required to produce a plausible 3D tree (*issue 5*).

Interactive modelling uses interaction to sketch a model in 2D and then create a 3D model from it. Okabe et al. (2005) proposes a system for designing 3D botanical trees based on a sketching interface and example-based control. Users only need to input 2D sketches (solves *issue 1*) based on the simple assumption that botanical trees tend to spread, system outputs a 3D tree. Wang et al. (2014) proposed a variational tree modelling approach, which targets on generating 3D trees in shapes that specifies by users.

Compare with rule-based systems, interactive modelling systems free users from complicated rules or parameters (solves *issue 2*). Trees can be produced quickly by ignoring some natural principles, thus the final outputs sometimes exhibit artefacts (*issue 3*). On the other hand, most systems are designed to construct a single tree (*issue 4*).

Image-based production models trees from image data with the advantage of increasing realism (*issue 3*). Reche-Martinez et al. (2004*a*) use a set of carefully registered photographs to compute the volumetric shape of a given tree with opacity. This approach produces impressive trees but does not recover explicit geometries of the branches. Neubert et al. (2007) proposed a technique for modelling 3D tree from loosely coupled images by computing a voxel-model of the tree volume with each voxel containing a density. This technique could product 3D tree model automatically (*issue 2*). It also allows user interaction to guide to a desired result, but it is not able to create all tree species. Pirk et al.'s (2012a) system needs users to input a tree model, which then will be converted into a graph-based representation. The structure of output tree can be modified according to the change of the environment.

Image-based modelling approaches have the best potential for producing plausible 3D tree since they rely on images of real trees. Limited user interaction is needed to improve the result.

1.2 Our system

We takes advantage of image-based production by using videos as input source to minimize user interaction. In order to reduce processing time, we ignore some details such as interactive modelling. However our system differs from these methods in the following aspects:

Figure 1.1: System components

- We use von Mises distribution (Forbes et al. (2011)) while generating 2D skeleton. This helps to keep the branch shape plausible and allows us to generate new trees that are similar but not identical. It is beneficial for users to create a forest from a small set of reconstructed trees.
- We use a probabilistic approach while converting 2D skeleton to 3D. Our method combines a bottom-up construction with the local cost and global cost. These two terms make sure the local branch pattern is natural after pushing, and simultaneously keeping the overall volumetric shape. This allows users to control the shape of the generated trees.

Figure 1.1 shows our system components. Firstly, users input a video and outline the tree in first frame. The background will be removed using alpha matting. Secondly we detect Harris interest points (Harris and Stephens, 1988) on the leafy part of the tree and track those points from frame to frame. Next, our system uses those data to construct a 2D skeleton: begin with dividing the whole feature sets to several clusters, and then use the centroid of each cluster as an initial approximation of nodes to find skeleton. Finally we need to generate 3D skeleton by two steps: copy 2D skeleton and rotate those copies, and following by pushing the branches away in a perpendicular direction.

Chapter 2

Literature Survey

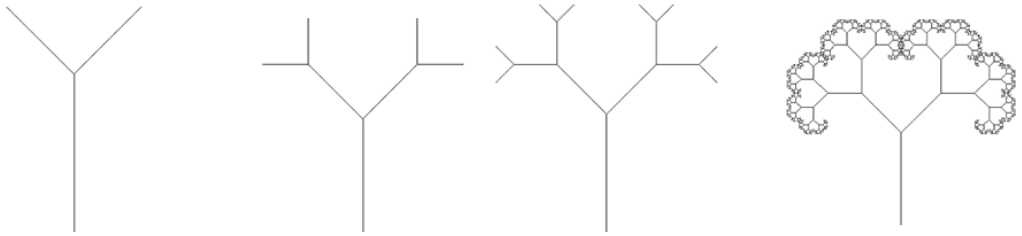
2.1 Different tree modelling approaches

There are currently three categories of tree modelling summarized by Neubert et al. (2007): rules-based generation, interactive modelling and image-based production.

2.1.1 Rules-based generation

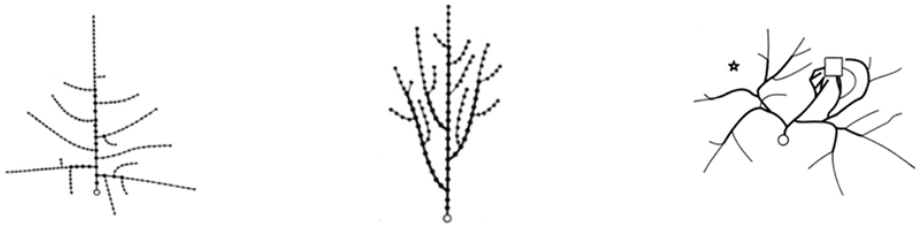
The modelling process in rule-based systems can be seen as a deductive process, and the final 3D tree is derived from initiator and abstract production rules.

Lindenmayer (1968) proposed L-systems and Prusinkiewicz et al. (1990) improved them. The core concept of L-systems is rewriting. The process is successively replacing parts of an initial object using a set of rewriting rules (Figure 2.1). The construction process applies the central concept of L-systems. In the very first step, we create the simplest tree which is one trunk and two branches, this is the initiator. Then we regard each branch as a tree, replacing each of them by the initiator, this is the rewriting rule. We keep repeating the rewriting process until we are satisfied with the results.



(a) initiator (b) replacing each branch with initiator (c) results

Figure 2.1: Construction of tree using L-systems (from Prusinkiewicz et al. (2003))



(a) (b) (c)

Figure 2.2: Continuous branching structures (from Deussen and Lintermann (2006)): (a) branching restriction introduced by minimal distance to end of branch; (b) vertical alignment of growth; (c) use of an attractor (square) and of an inhibitor (star)

Deussen and Lintermann (2006) proposed a procedural approach to generate new trees from an initial state. The system use computer-assisted methods to simulate of natural growth processes. With a high performance computer, users are able to model complex branching structures therefore producing a faithful tree (Figure 2.2). Procedural creation processes are parameterized algorithms that are designed for the production of certain species of trees.

Talton et al. (2011) used a Metropolis-based approach to determine which rules and parameter values to apply in order to generate a desired 3D output. The system firstly achieves different growth patterns including both sympodial and monopodial growth from a known large set, and then simulates the effects of growth using L-system. Several tree grammars are developed afterwards. To generate a 3D tree, users have to input a high-level specification of the desired production (e.g., a sketch), the system then computes a production from the grammar that conforms to the specification. The final output production is generated by optimizing over the space of possible productions from the grammar. Figure 2.3 shows that the inference technique successfully models details in the sketches, such as silhouettes and the low-density hole in the oak.

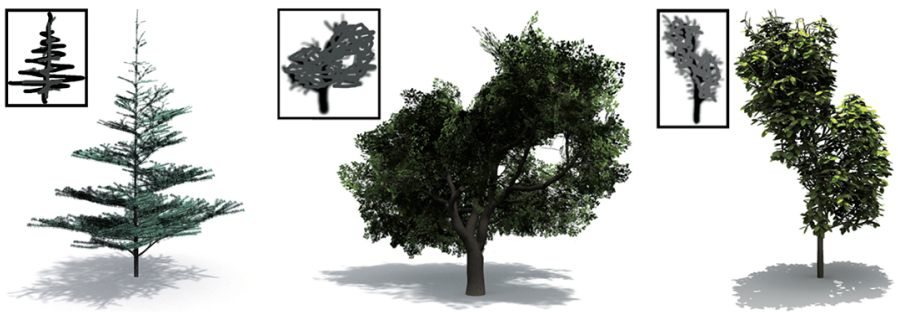


Figure 2.3: Conifer, old oak, and poplar grammars targeted to sketches (from Talton et al. (2011)

(a) (b) (c)

Figure 2.4: Construction of 3D Tree (from Okabe et al. (2005)): (a) user draws a 2D sketch of a tree; (b) system generates a 3D tree; (c) a denser 3D tree model is obtained.

Rule-based systems could help to produce a various species of realistic trees. However, it is difficult for non-technical users to design trees of desired appearance because users need to understand how the rules are applied, while the input of initiator and rewriting rules are in 2D which are very different from the 3D geometry output (such as L-systems). Moreover, rewriting rules work locally, thus small changes of values might cause large changes in the overall shape, which makes it even harder for novice users. On the other hand, high performance computers are required to produce a plausible 3D tree (such as procedural modelling). It is possible to limit the amount of adjustable parameters for the user to reduce computational complexity. However, with increasing model complexity, this amount also increases.

2.1.2 Interactive modelling

Interactive modelling system uses interaction to sketch a model in 2D and then create a 3D model. The process can be seen as an inductive process, in that the user specifies the 2D appearance of the model directly and then the system generates a 3D structure by inferring hidden parameters.

Okabe et al. (2005) proposed a system for designing 3D botanical trees based on a sketching interface and example-based control (Figure 2.4). Users only need to input 2D sketches based on the simple assumption that botanical trees tend to spread, system outputs a 3D tree. Compare with rule-based systems, the whole process free the user from complicated rules or parameters. The assumption is an overly simplistic description of the growth process of real trees, but it is fast and general enough for quickly designing reasonable-looking trees from typical sketches.

Chen et al. (2008) described a system (Figure 2.5) for converting a user's freehand sketch of a tree into a full 3D model using Markov random field. Users only need to provide a few strokes of branches and the outline of the tree, and then the system uses probabilistic optimization based on parameters that obtained from a database of tree models. The best matching model is selected by comparing its 2D projections with the sketch. Next, branch iteration is modelled by a Markov random field. The system then propagates branches

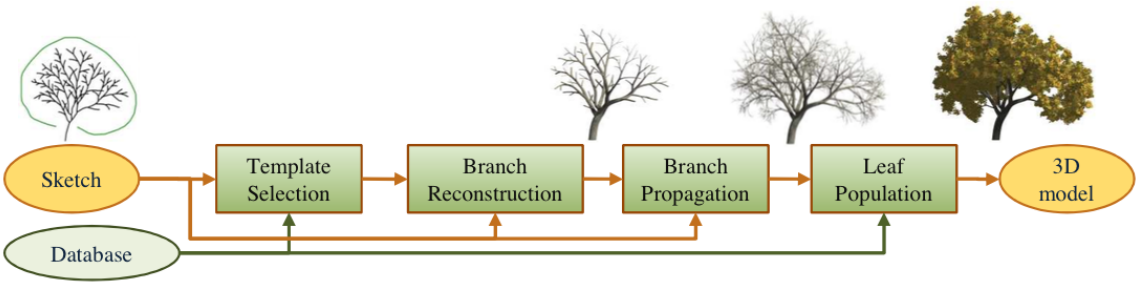


Figure 2.5: Overview of sketch-based tree modelling system (from Chen et al. (2008))

using the principle of self-similarity.

Some methods combine rules and interaction. Ijiri et al.'s (2006) system is based on L-systems. The user draws the stroke, the system gradually advances the growth simulation and creates a tree model along the stroke. The change in the shape of the stroke is used as a graphical metaphor for modifying the L-system parameters.

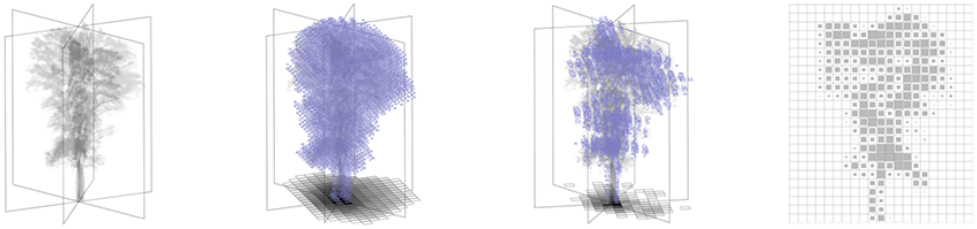
Interactive modelling systems could produce various interesting tree-like shapes quickly by ignoring some natural principles, therefore the final outputs sometimes exhibit artefacts that never appear in conventional tree modelling systems. On the other hand, most systems are designed to construct a single tree, which makes it impossible to create a forest where most trees are similar but not identical to each others.

2.1.3 Image-based production

Rather than requiring the user to manually specify the plant model, they model trees from image data.

A purely image-based modelling approach is described by Reche-Martinez et al. (2004*b*). The system models a 3D tree from photographs in three steps. Firstly, we use alpha matting to extract a tree from a set of photographs inputted by users. This step also generates the appropriate foreground colour to be used for rendering and requires limited user intervention. Subsequently an automatic reconstruction step creates a recursive grid which contains the opacity values for each cell. Finally we generate view-dependent textures for each cell corresponding to the input views, which are attached to billboards placed at the centres of the cells. Rendering is performed using view-dependent texture mapping.

Neubert et al. (2007) proposed a method to model a 3D tree from a set of photographs based on limited user interaction. The system combines image-based modelling and sketch-based modelling. An approximate voxel-based tree volume with density values is estimated from loosely registered input image set (Figure 2.6). The density values of the voxels are used to produce initial positions for a set of particles. A 3D flow simulation is performed to trace the particles downward to the tree basis and is combined to form twigs and branches. Then the geometry of the tree skeleton is produced using botanical rules for branch thickness



(a) (b) (c) (d)

Figure 2.6: Estimating the tree density (from Neubert et al. (2007)): (a) Initial density values for three input images; (b) voxel grid by back-projection; (c) refined voxel grid; (d) density values for one image plane.

and angle. The user is required to draw some branches if there is not enough information about branching in the images. As a result, the final 3D tree does not have the exactly same shape with the drawn branches.

Tan et al. (2007) proposed another method to reconstruct 3D tree with images and user interaction. The system consists of three main parts: image capture and 3D point recovery, branch recovery, and leaf population. The user can also manually edit the branch structures to improve the result. However, the system requires a significant amount of preprocessing work such as segmentation of branches, leaves, and background. User intervention may be required to correct errors in the 3D branch extraction.

Pirk et al. (2012a) presented a method which uses a static tree model instead of images. Given a tree model, the system converts it into a graph-based representation. The structure of the tree then can be modified according to the change of the environment. Also, Pirk et al. (2012b) proposed an automatic method, which analyses developmental stages that approximate the tree's natural growth. Users could add or prune branches, as well as explore all intermediate stages of tree growth.

Image-based modelling approaches have the best potential for producing plausible 3D trees since they rely on images of real trees and limited user interaction is needed to improve the result. In addition, it is also easy for novice users to produce a fairly realistic 3D tree.

2.2 Methods applicable to our system

2.2.1 Video tracking

Alpha matting

First of all, users need to outline the tree in frame one because automatic segmentation is not solved.

Extracting a foreground object from the background involves determining both full and

partial pixel coverage. This problem was mathematically established by Porter and Duff (1984), who introduced the alpha channel as the means to control the linear interpolation of foreground and background colours over an arbitrary background. Mathematically, the input image I, which is assumed to be a composite of a foreground image F and a background image B. The colour of the ith pixel is assumed to be a linear combination of the corresponding foreground and background colours:

$$I_i = \alpha_i F_i + (1 - \alpha_i) B_i \tag{2.1}$$

where α_i is the pixel's foreground opacity, it can be any value in $[0, 1]$. If $\alpha_i = 1$ or 0, we call pixel i definite foreground or definite background respectively. Otherwise it is called pixel z mixed. Given only a single input image, all three values α, F and B are unknown and need to be determined at every pixel location.

Levin et al. (2008) present a closed-form solution for extracting the alpha matte from a natural image. The derivation starts from a closed-form solution for grey-scale images. Here we will need some assumptions on F and B: assuming both F and B are approximately constant over a small window around each pixel. Therefore we could get α:

$$\alpha_i \approx a I_i + b, \forall i \in w \tag{2.2}$$

where $a = \frac{1}{F-B}$, $b = -\frac{B}{F-B}$ and w is a small image window. This relation suggests finding α, a, and b that minimize the cost function:

$$J(\alpha, a, b) = \sum_{j \in I} \left(\sum_{i \in w_j} (\alpha_i - a_j I_i - b_j)^2 + \epsilon a_j^2 \right) \tag{2.3}$$

where w_j is a small window around pixel j. If we define $J(\alpha) = \min_{a,b} J(\alpha, a, b)$, we get $J(\alpha) = \alpha^T L \alpha$, where L is an $N \times N$ matrix, whose (i, j)-th entry is:

$$\sum_{k|(i,j) \in w_k} \left(\delta_{ij} - \frac{1}{|w_k|} \left(1 + \frac{1}{\frac{\epsilon}{|w_k|} + \sigma_k^2} (I_i - \mu_k)(I_j - \mu_k) \right) \right) \tag{2.4}$$

Here δ is the Kronecker delta, μ_k and σ_k^2 are the mean and variance of the intensities in the window w_k around k, and $|wk|$ is the number of pixels in this window.

A simple way to apply the cost function to colour images is to apply the grey-level cost to each channel separately:

$$\alpha_i \approx \sum_c a^c I_i^c + b, \forall i \in w \tag{2.5}$$

where c sums over colour channels. Using above equation to define the following cost function for the matting of RGB images:

$$J(\alpha, a, b) = \sum_{j \in I} \left(\sum_{i \in w_j} \left(\alpha_i - \sum_c a_j^c I_i^c - b_j \right)^2 + \epsilon \sum_c a_j^{c^2} \right) \tag{2.6}$$

Similarly to the grayscale case, a^c and b can be eliminated from the cost function, yielding a quadratic cost in the α unknowns alone, we get $J(\alpha) = \alpha^T L \alpha$.

Harris Interest Points

Given an outline of how we detect Harris interest points(1988) on the leafy part of the tree.

Without loss of generality, we will assume a grayscale 2-dimensional image is used. Let this image be given by I. Consider taking an image patch over the area (u, v) and shifting it by (x, y). The weighted sum of squared differences (SSD) between these two patches, denoted S, is given by:

$$S(x, y) = \sum_u \sum_v w(u, v)(I(u + x, v + y) - I(u, v))^2 \tag{2.7}$$

Let I_x and I_y be the partial derivatives of I:

$$I(u + x, v + y) \approx I(u, v) + I_x(u, v)x + I_y(u, v)y \tag{2.8}$$

So Equation 2.7 becomes:

$$S(x, y) \approx \sum_u \sum_v w(u, v)(I_x(u, v)x + I_y(u, v)y)^2 \tag{2.9}$$

Which can be written in matrix form:

$$S(x, y) \approx \begin{pmatrix} x & y \end{pmatrix} A \begin{pmatrix} x \\ y \end{pmatrix} \tag{2.10}$$

Where A is the structure tensor:

$$A = \sum_u \sum_v w(u, v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = \begin{bmatrix} \langle I_x^2 \rangle & \langle I_x I_y \rangle \\ \langle I_x I_y \rangle & \langle I_y^2 \rangle \end{bmatrix} \tag{2.11}$$

This matrix (Equation 2.11) is a Harris matrix, and angle brackets denote averaging (i.e. summation over (u, v)).

An interest point is characterized by a large variation of S in all directions of the vector $\begin{pmatrix} x & y \end{pmatrix}$. By analysing the eigenvalues of A.

Tracking points

A simple technique is using exhaustive searching. System calculates a measure of differences between images at all possible values of the disparity vector. This is a very time consuming process: if the size of the picture $G(x)$ is $N \times N$ and the region of possible of h is of size $M \times N$, then this method requires $O(M^2N^2)$ time to compute. By using hill-climbing technique, we could find a better h to speed up the process, however this will involve the risk of possible failure.

Another technique introduced by Barnea and Silverman (1972), known as the sequential similarity detection (SSDA), which estimates the error disparity vector h. The error function must be cumulative such as the L_1 norm (Equation 2.12) or L_2 norm (Equation 2.13). One stops accumulating the error for the current h under investigation when it becomes apparent that the current h is not likely to give the best match.

$$L_1 \text{ norm} = \sum_{x \in R} |F(x+h) - G(x)| \tag{2.12}$$

$$L_2 \text{ norm} = \sqrt{\sum_{x \in R} [F(x+h) - G(x)]^2} \tag{2.13}$$

Most traditional image registration techniques are generally costly. Lucas et al. (1981) presented a new image registration technique that uses spatial intensity gradient information to direct the search for the position that yields the best match, known as Kanade-Lucas-Tomasi (KLT) feature tracker. It is faster than traditional techniques for examining far fewer potential matches between the images. In one-dimensional case, if h is the displacement between two images $F(x)$ and $G(x) = F(x+h)$, then the approximation is made such that:

$$F'(x) \approx \frac{F(x+h) - F(x)}{h} = \frac{G(x) - F(x)}{h} \tag{2.14}$$

so that

$$h \approx \frac{G(x) - F(x)}{F'(x)} \tag{2.15}$$

The derivation above cannot be generalized will to two dimensions for the 2D linear approximation occurs differently. However, this can be corrected by applying the linear approximation in the form:

$$F(x+h) \approx F(x) + hF'(x) \tag{2.16}$$

to find the h which minimizes the L_2 norm measure of the difference (or error) between the curves, where the error can be expressed as:

$$E = \sum_x [F(x+h) - G(x)]^2 \tag{2.17}$$

To minimize the error with respect to h, partially differentiate E and set it to zero:

$$\begin{aligned} 0 &= \frac{\partial E}{\partial h} \\ &\approx \frac{\partial}{\partial h} \sum_x [F(x) + hF'(x) - G(x)]^2 \\ &= \sum_x 2F'(x)[F(x) + hF'(x) - G(x)] \end{aligned} \tag{2.18}$$

so that:

$$h \approx \frac{\sum_x F'(x)[G(x) - F(x)]}{\sum_x F'(x)^2} \tag{2.19}$$

2.2.2 2D acquisition

Motion Magnification

The Harris points are tracked from frame to frame. We store the trajectories of each points, and calculate likelihoods between each point. To achieve this, we introduce a method from Liu et al.'s (2005) paper. We cluster points by normalized correlation between the trajectories as a measure of their similarity. Composing the x and y components of the velocities into a complex motion vector, the correlation index $\rho_{n,m}$ between the trajectories of two feature points n and m is:

$$\rho_{n,m} = \frac{\sum_k (v_{nk}^x + j v_{nk}^y)(v_{mk}^x + j v_{mk}^y)}{\sqrt{(\sum_k (v_{nk}^x)^2 + (v_{nk}^y)^2)(\sum_k (v_{mk}^x)^2 + (v_{mk}^y)^2)}} \tag{2.20}$$

with $j = \sqrt{-1}$. The normalized correlation between complex velocities is invariant to both the direction of the motion trajectories, and their magnitudes. The normalized correlation is close to zero for feature points that belong to objects with independent motions.

Normalized Cut Algorithm

Using the normalized correlation, we first construct a similarity matrix between all feature tracks, and then use normalized cut algorithm introduced by Shi and Malik (2000).

A graph $G = (V, E)$ can be partitioned into two disjoint sets, $A, B, A \cup B = V, A \cap B = \emptyset$, by simply removing edges connecting the two parts. The degree of dissimilarity between these two pieces can be computed as total weight of the edges that have been removed. In graph theoretic language, it is called the *cut*:

$$cut(A, B) = \sum_{u \in A, v \in B} w(u, v) \tag{2.21}$$

The optimal bipartitioning of a graph is the one that minimizes this cut value. Although there are an exponential number of such partitions, finding the minimum cut of a graph is a well-studied problem and there exist efficient algorithms for solving it. However, *cut* defined in Equation 2.21, increases with the number of edges going across the two partitioned parts. To avoid this unnatural bias for partitioning, we propose a new measure of disassociation between two groups. Instead of looking at the value of total edge weight connecting the two partitions, our measure computes the cut cost as a fraction of the total edge connections to all the nodes in the graph. We call this disassociation measure the *normalized cut (Ncut)*:

$$Ncut(A, B) = \frac{cut(A, B)}{assoc(A, V)} + \frac{cut(A, B)}{assoc(B, V)} \tag{2.22}$$

where $assoc(A, V) = \sum_{u \in A, t \in V} w(u, t)$ is the total connection from nodes in A to all nodes in the graph and $assoc(B, V)$ is similarly defined. With this definition of the disassociation between the groups, the cut that partitions out small isolated points will no longer have small *Ncut* value, since the cut value will almost certainly be a large percentage of the total connection from that small set to all other nodes.

In the same spirit, we can define a measure for total normalized association within groups for a given partition:

$$Nassoc(A, B) = \frac{assoc(A, A)}{assoc(A, V)} + \frac{assoc(B, B)}{assoc(B, V)} \tag{2.23}$$

where $assoc(A, A)$ and $assoc(B, B)$ are total weights of edges connecting nodes within A and B, respectively. We see again this is an unbiased measure, which reflects how tightly on average nodes within the group are connected to each other.

Another important property of this definition of association and disassociation of a partition is that they are naturally related:

$$\begin{aligned} Ncut(A, B) &= \frac{cut(A, B)}{assoc(A, V)} + \frac{cut(A, B)}{assoc(B, V)} \\ &= \frac{assoc(A, V) - assoc(A, A)}{assoc(A, V)} + \frac{assoc(B, V) - assoc(B, B)}{assoc(B, V)} \\ &= 2 - \left(\frac{assoc(A, A)}{assoc(A, V)} + \frac{assoc(B, B)}{assoc(B, V)} \right) \\ &= 2 - Nassoc(A, B) \end{aligned} \tag{2.24}$$

Hence, the two partition criteria that we seek in our grouping algorithm, minimizing the disassociation between the groups and maximizing the association within the groups, are in fact identical and can be satisfied simultaneously. In our algorithm, we will use this normalized cut as the partition criterion.

von Mises distribution

In order to produce similar but not identical 2D skeleton, we try to randomize the angle between branches. All angles need to be chosen carefully to ensure a plausible branch patterns. Since angles are only limited from 0 to 360 degrees, we found that using von Mises distribution suits this the best. It is a continuous probability distribution on the circle. We will find all angles of branches from the data gathering from video, divide them into different groups. Then using von Mises distribution to choose angles which have probability density higher then a certain value.

The von Mises distribution (Forbes et al. (2011)) can be regarded as the circular analogue of the normal distribution on the line. The distribution is unimodal, symmetric, and infinitely divisible. The probability density function for the von Mises variate with location parameter and for selected values of the scale parameter. The von Mises probability density function for the angle x is given by:

$$f(x|\mu,\kappa) = \frac{e^{\kappa \cdot \cos(x-\mu)}}{2\pi I_0(\kappa)} \tag{2.25}$$

where $I_0(\kappa)$ is the modified Bessel function (Abramowitz and Stegun (1964)) of order 0 (Equation 2.26).

$$x^2\frac{d^2y}{dx^2} + x\frac{dy}{dx} + (x^2 - \alpha^2)y = 0 \tag{2.26}$$

The parameters μ and $1/\kappa$ are analogous to μ and σ^2 (the mean and variance) in the normal distribution, where μ is a measure of location and κ is a measure of concentration.

2.2.3 3D skeleton modelling

Tan et al. (2007) presents a method based on image capture and 3D point recovery. A hand-held camera is used to capture the appearance of the tree from a number of different overlapping views. There are about 10 to 20 images taken for each tree, with coverage between 120° and 200° around the tree. Prior to any user-assisted geometry reconstruction, system extract point correspondences and ran structure from motion on them to recover camera parameters and a 3D point cloud. The system uses a quasi-dense approach (Lhuillier and Quan, 2005) to compute the camera poses and a semi-dense cloud of reliable 3D points in space.

Okabe et al.'s (2005) system creating 3D Tree from a 2D sketch by adjusting the orientation of a branch so that the distances between them are as large as possible. Branches are added

one by one and the order of processing sibling branches is random. Ideally, the system would construct a 3D volume whose voxels contain the distance to the nearest branch and then place the branch whose voxels have maximum distance values.

Li et al. (2011) proposed a method to build 3D model from a 2D tree skeleton which owes much to Tan et al. (2007) and Okabe et al. (2005). The basic approach consists of two steps: a copy-operation and "pushing" the resulting structures in the right form. For the first step, the system copies 2D skeleton, which lies on the xy-plane to yz-plane. Then for the second step, the branches need to be pushed away from the initial xy-plane and yz-plane in a perpendicular direction. This "pushing" step is performed from root to tip. At each step i, all the descendants of x_i are pushed by the same distance. In order to fill a volume and make branches plausible, Li et al. (2011) proposed a probabilistic solution (Equation 2.27), which maximizes the posterior probability defined by the Bayes' rule.

$$p(x_i|\Omega, X_{i-1}) \propto p(\Omega|X_{i-1}, x_i)p(X_{i-1}, x_i) \tag{2.27}$$

The posterior is factorized into two terms: the local term (Equation 2.28) keeps the branch shape plausible, while the global term (Equation 2.29) keeps the overall volumetric shape.

$$p(X_{i-1}, x_i) = p(X_{i-1}|x_i)p(x_i) \tag{2.28}$$

$$p(\Omega|X_{i-1}, x_i^l) = \frac{E(\Omega, X_{k-1}, x_i^l)}{\sum_{j=1}^{L} E(\Omega, X_{i-1}, x_i^l)} \tag{2.29}$$

Chapter 3

Requirements

3.1 User requirements

By reviewing interviews with project supervisor, we summarized user requirements as following:

- The system should involves minimum user interaction;
- Users should be able to use videos or images as the input source;
- The system should generate 3D tree in relatively short amount of time;
- The system should produce plausible 3D skeletons;
- Users could produce several similar but not identical 3D trees based on the same initial data;
- The system should be easy to reuse;
- Users could control some parameters according to different requirements of the 3D tree.

3.2 Requirements specification

3.2.1 Functional requirements

- Essential requirements (Priority: HIGH):
  1. Read a video or images as an input source;
  2. Collect necessary information from the input source;
  3. Generate 2D skeleton;

4. Able to create several new 2D skeletons based on the initial data.

- Ideal requirements (Priority: MEDIUM):

1. Create 3D skeleton based on 2D skeleton;
2. Able to control the shape of 3D model.

- Advanced requirements (Priority: LOW):

1. Add motions to 3D skeleton;
2. Add leaves to branches;
3. Add motions to leaves.

3.2.2 Non-functional requirements

1. The project should be completed by the end of March 2015 and start writing dissertation (final deadline is 1st May 2015).
2. All code must be well documented.
3. Meet project supervisor weekly to report current progress.
4. Project must have a version control, a git repository is needed.

3.3 Functional requirements dependencies

The dependencies table (Table 3.1) shows dependencies between different requirements within the system. Dependencies tables are drawn to give a visual overview to system designers of the relationships between a set of requirements and to remind them that changing one requirement may affect the others.

The dependencies table can also help to prioritize the implementation of requirements when scheduling different system development iterations. Requirements with no dependencies should be implemented at the early stages while the others can be scheduled for later iterations.

Req. ID	Requirement title	Dependencies
1.1	Read a video or images as an input source	None
1.2	Collect necessary information from the input source	1.1
2.1	Generate a 2D skeleton	1.1
2.2	Create new 2D skeletons	2.1
3.1	Create a 3D skeleton	2.1
3.2	Control the shape of 3D skeleton	3.1
3.3	Add motions to the 3D skeleton	3.1
4.1	Add leaves to branches	1.1, 3.1
4.2	Add motions to leaves	4.1
5.1	Testing	None
6.1	Code reuse	None

Table 3.1: Functional requirements dependencies

Chapter 4

Design

4.1 System decomposition

System decomposition allows us to break down a complex system into a collection of several small parts that are easier to manage. Our system could be decomposed into the following components:

- Read input data;
- Generate 2D skeleton;
- Generate 3D model;
- Add leaves;
- Add motions.

Based on the above 5 main components, we are able to schedule the processing of developing system in detail.

4.2 Software process

A software process model is an abstract representation of a software process. After comparing several well-known process models, we decide to choose the waterfall model (Figure 4.1). This model takes the fundamental process activities of specification, development, validation, evaluation and represents them as separate processing phases:

1. *Requirements analysis and definition*: each of our system components can be regard as a requirement. We need to analysis the constrains and goals, and then defined in detail and serve as a system specification;

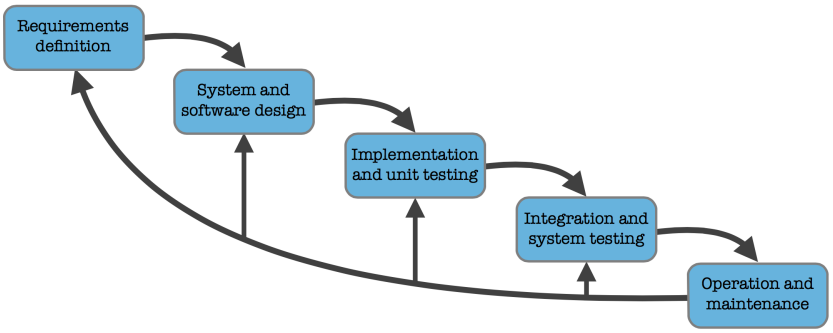


Figure 4.1: The waterfall model

2. *System and software design*: involves identifying and describing the fundamental software system abstractions and their relationships;

3. *Implementation and unit testing*: the software design is recognised as a set of program units. Each function has to be tested to ensure it meets the specification;

4. *Integration and system testing*: the individual program units are integrated and tested as a complete system to ensure that the software requirements have been met.

5. *Operation and maintenance*: the system is established and put into practical use, but there will be errors that were not discovered in earlier stage or we need to improve the performance of the system.

4.3 Project management

This project will be conducted over a 32 week period. There are a number of key requirements were set and were agreed upon with the project supervisor. Figure 4.2 shows the details about project management.

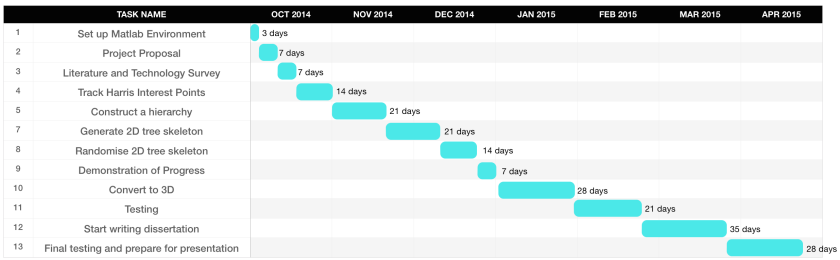


Figure 4.2: Gantt Chart

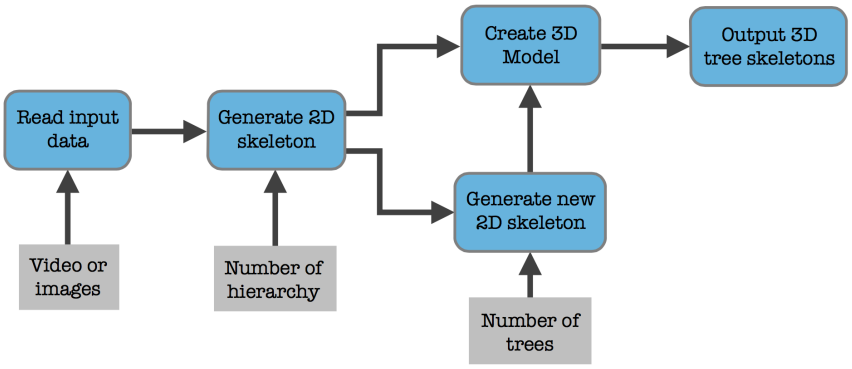


Figure 4.3: The function-oriented pipeline model

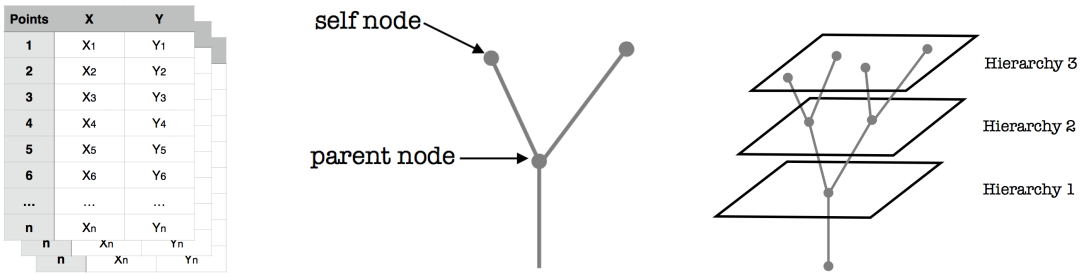
4.4 Architectural design

Architectural design is a process to establish a system organisation that will satisfy the functional and non-functional system requirements. It is very clear in our specification that we need several functions to produce a 3D model using our system, therefore a function-oriented pipeline model (Figure 4.3) suits our system naturally. In the system, data flows from one to another and is transformed as it moves through the sequence. Each processing step is implemented as a transformation. Input data flows through these transformations until converted to output: input information is used to generate 2D skeleton; data of the 2D skeleton could be used to create 3D model, or to generate new 2D skeleton; data of each 2D skeleton will be used to create new 3D model; data of all 3D models will be used to produce the finial result.

4.5 Data structures

Our system uses *matrix* as basic data type (Figure 4.4 (a)): each frame will be stored as a $n * 2$ matrix which contains x and y positions of each points (n is the number of total points); all operations that we are using to create 3D model will be applied to matrices; the results of 2D skeletons and 3D models will be stored in matrices. With the help of MATLAB, operations on matrices become very convenient.

For each point of branches, we create a new structure called *node* (Figure 4.4 (b)): each node contains two coordinates, *self* and *parent*. We create this new structure because we will use each node's parent coordinate quite frequently in the system, e.g., plotting trees; calculating angle between parent node. There is one place requires special attention, which is we only store the coordinate of parent node. Hence we can not use code such as `node.parent.parent` to get to coordinate of node's grandparent. Also, while system updating nodes, make sure to updates both `node.self` and `node.parent`.



(a) Matrices (b) Node (c) Hierarchy

Figure 4.4: Data structures

Figure 4.4 (c) illustrates another data structure called *hierarchy*. Tree is a special object, each branch of tree comes from its parent branch. Group nodes into different hierarchies can make things easier to manage: we could set different line widths to different hierarchies since branches in same hierarchy normally have similar thickness, and child branches are always getting thinner; we could also apply different algorithms to different hierarchy to produce a more plausible shape. On the other hand, users could specify the number of hierarchies to control density of branches: a larger number will result in denser tree.

Chapter 5

Implementation

5.1 Programming language

The choice of programming language for the development of the system has to take into account many factors that could influence the outcome of the project. A good choice could help to develop our system in an efficient way. It is always much easier to find out a solution when you encounter any problems for a popular programming language. Also, there will be a lot excellent libraries available to use.

After some researches and discussions with project supervisor, we agree to use MATLAB to develop our system for the following reasons:

- Its basic data element is the matrix: we need to deal with a large amount of images, which each could be read as matrices;
- Vectorized operations: our system involves a lot operations on matrix, such as rotations and translations;
- The graphical output is optimized for interaction. You can plot your data very easily, and then change colours, sizes, scales, etc., by using the graphical interactive tools;
- Most functions we need for developing our system are already there: we do not need to rewrite code for tracking points, plotting lines, etc.;
- The functionality can be greatly expanded by the addition of toolboxes: there are quite a lot toolboxes available for us if we have any special requirement for images.

Overall, choosing MATLAB seems to be a wise choice for us to develop a system that deals with graphics. It could help us to save time on development, hence we could focus on the basic concepts. It also helps to produce a fast prototype so we could test our new ideas really quick to see if it is worth keeping on in that direction.

The major disadvantage is that MATLAB is often slower at execution time. Also it uses a large amount of memory. In practice, we do not care much because we do not know in advance what method is going to be successful, we need to try many things, so our bottleneck is programming time.

5.2 Preparing videos

At very beginning, we need to shoot a video of a moving tree. There are some requirements to the trees in order to produce a better 3D model:

1. Trees with denser leaves are preferred:
   In the feature points detection process, we are looking for the corners between each leaf and branch. Figure 5.1 shows the feature points detected in two different trees. Red dots are the corners between outline and background, which we will talk more about how to remove them later. Yellow dots could be removed simply by choosing trees with denser leaves since there are less empty spaces in the inner tree. So most of the feature points are valid. While for the trees with sparse leaves, our system detects corners between leaves and background or between branches and background, those are the points introduce noises and need to be removed.

2. The motion of trees should be moderate:
   We need to track feature points from frame to frame to collect useful information. All points then will be grouped into different clusters base on the distances between each points and their movements. The tree should not in a violent movement because in this case, all feature points will move in relative large distance, which results in points moves in the neighbour clusters. No movement is also not good because then we could only use the distances between points to group them, this will reduce the accuracy.

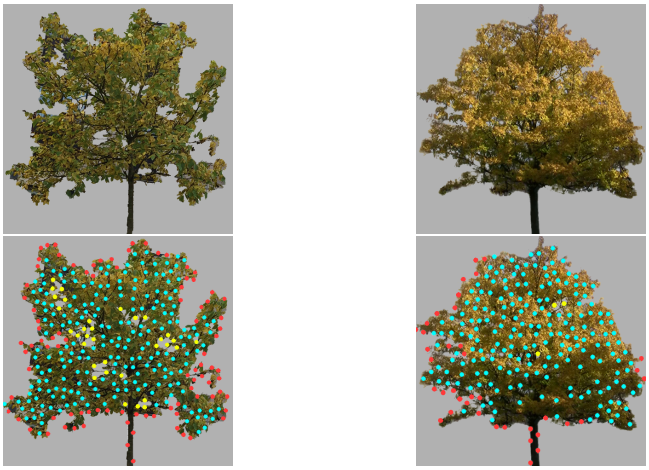
3. Make sure the video recorder is steady while recording videos: shakes will confuse our system since there is no way to distinguish whether it is the movement of a tree or the video recorder.

4. Make sure the whole tree is recorded not only a part, and tree is in correct position, e.g., not upside down.

5.3 Tracking video

5.3.1 Removing background

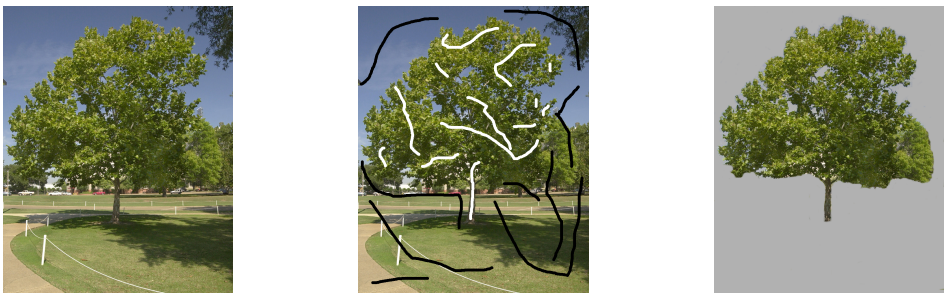
We remove background of the first frame using alpha matting. Rhemann et al. (2009) provides a online benchmark for image matting based on Levin et al.'s (2008) closed form



(a) Sparse leaves (b) Dense leaves

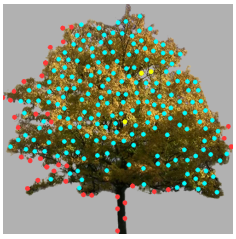
Figure 5.1: Feature points detection: Cyan dots are the points we want to track; red dots are the corners between outline and background; yellow dots are the corners between inner tree and background.

solution. Figure 5.2 shows how to remove background in our system. This is where the only user interaction is involved through the whole process. Users need to scribble the first frame of input video: draw white line on branches and leaves, and black line on any background we want to remove. Note that since the code uses a simple difference to find the scribbled pixels, user should not place scribbles with fuzzy boundaries. As you can see from the output, the boundary is not very accurate, it does not matter much since we will discard the feature points on boundary, which will be discussed later.



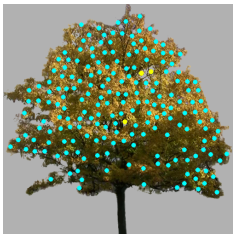
(a) (b) (c)

Figure 5.2: Removing background using alpha matting: (a) Original image; (b) Users scribble on original image with black and white where black indicates background to be removed and white is the leaves and branches we want to keep; (c) The output tree with background removed.



(a) Original result

(b) Alpha image



(c) Improved result

Figure 5.3: Removing inaccurate feature points (red dots)

5.3.2 Feature points detection

Our system uses OpenCV library to detect feature points in the first frame. Figure 5.3 (a) shows the result of detecting feature points. Users are allowed to use different parameters:

- `max`: the maximum number of feature points;
- `radius`: the minimum distance between feature points;
- `threshold`: feature distinguish threshold, determine the quality of feature points.

In practice, we keep threshold as default and change radius value to control the numbers of feature points. We found that number of feature points between 300 to 400 are sufficient for many trees species without introducing unnecessary complexity.

Improvement

We have mentioned before that we do not need to extract tree out of background with very accurate boundary. This is because no matter how accurate it is, feature points along the boundary will be regard as noises since we only need corners between leaves and branches not the background. So we need to remove the points along the boundary.

Firstly, we need to create a alpha image of the tree (see Figure 5.3 (b)) so that we could have a clear boundary. Note we also remove the main trunk because it is stable all the time, we can draw it at the end of the construction process. Then we create a mask base on the alpha image. We use this mask to filter out points which are at the boundary. Figure 5.3 (c) shows the final feature points that we are going to track throughout the video.

5.3.3 Tracking feature points

We track points in video using KLT algorithm (Lucas et al., 1981), which is available in MATLAB's tool box. In order to initialize the tracking process, we need to specify the initial locations of the points in first frame, which are the feature points we got from previous

step. All these points are tracked in subsequent video frames. We found that tracking once in ten frames rather than tracking each frame is sufficient without introducing unnecessary complexity.

At the end of each tracking, we need to store the positions of feature points. As we have discussed in Chapter 4.5 (Data structure), all points are stored by frame. After this, we need to store each point's trajectory, that is creating n matrices where n is the total number of points, and storing the positions of the same point in different frame into its corresponding matrix. So after this, we have a variable `Tree.trajectory`, which allows us to get the trajectory of any feature points simply by calling `Tree.trajectory{i}` where `i` is the index of a point.

5.4 2D acquisition

5.4.1 Constructing a hierarchy

Our system then constructs a hierarchy (see Figure 4.4 (c)) using recursive binary clustering: first dividing the whole feature set into halves, then each half into quarters and so on. We use the Normalized Cut algorithm (Shi and Malik, 2000) for clustering because it is designed to partition data based on affinities between pairs on nodes.

Calculating similarity matrix

In order to dividing feature points into different clusters, we need to find their similarities using motion magnification (Liu et al., 2005): calculate an affinity matrix from the spatial distances between the feature points and their similarity of motion. Distances between each feature points can be simply calculated by calling `dist(Tree.featurePoints)`, while the similarity of motion requires some careful calculations.

To get similarity of the motions, firstly we calculate velocity for each point: adding moving distance between each frame to get total distance; the total time is the same to all points so we just set it to a unit. Then we can get velocity for each point by subtracting the mean of its velocity. A normalized correlation is calculated using Equation 5.1 (from Liu et al. (2005)). The normalized correlation is close to zero for feature points that belong to objects with independent motions. Using the normalized correlation, we constructed a similarity matrix between all feature tracks.

$$\rho_{n,m} = \frac{\sum_k (v_{nk}^x + j v_{nk}^y)(v_{mk}^x + j v_{mk}^y)}{\sqrt{(\sum_k (v_{nk}^x)^2 + (v_{nk}^y)^2)(\sum_k (v_{mk}^x)^2 + (v_{mk}^y)^2)}} \tag{5.1}$$

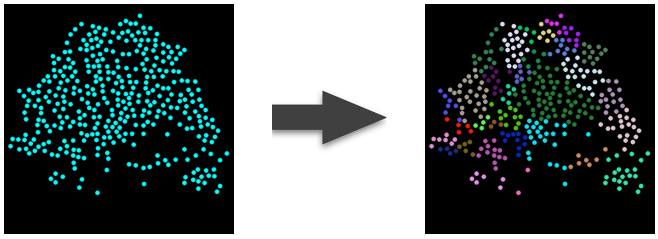


Figure 5.4: Clustering feature points

Clustering feature points

Now we need to split all feature points into different groups using Normalized Cut algorithm (Shi and Malik, 2000) based on the similarity matrix we get from previous step. Firstly we measure the normalized cut (see Equation 5.2) to get the similarity between different parts, and then measure the total normalized association within groups (see Equation 5.3). Since $Ncut = 2 - Nassoc$, a cut that minimize $Ncut$ also maximizes $Nassoc$. We will use this normalized cut as the partition criterion, recursively partition the segmented parts.

$$Ncut(A, B) = \frac{cut(A, B)}{assoc(A, V)} + \frac{cut(A, B)}{assoc(B, V)} \tag{5.2}$$

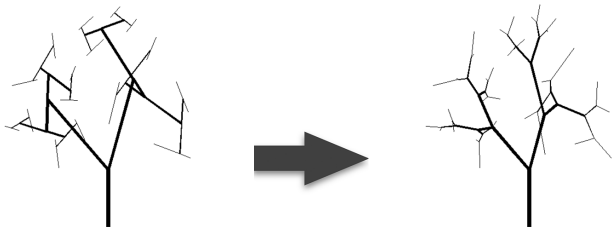
$$Nassoc(A, B) = \frac{assoc(A, A)}{assoc(A, V)} + \frac{assoc(B, B)}{assoc(B, V)} \tag{5.3}$$

The clustering process stops when the hierarchy reaches a certain level. We find $5-7$ levels of hierarchy is sufficient for many trees species without introducing unnecessary complexity. The output is a binary hierarchy: each node has two child nodes (except tip nodes); each node stores both its own position and its parent's position as we have shown in Figure 4.4 (b).

Figure 5.4 shows the result of clustering whole feature set with different colours indicate different clusters. It is a 5 levels hierarchy which contains $2^5 = 32$ different clusters. The hierarchy can be used for tracking the tree over long periods of time, which implies it is a plausible model of the branching structure.

5.4.2 Generating 2D skeleton

To find a skeleton, we use the centroid of each cluster as an initial approximation of nodes in X'. If we just connect those points from root to tip nodes recursively, we will get a skeleton like Figure 5.5 (a), which does not present a botanical shape. To solve this problem, we need to shift $x' \in X'$ to a better position: for every branch a_{ji}, we shift the child node along a line x'_i towards its parent: $x'_i \mapsto x'_i + \alpha(x'_j - x'_i)$. This gives a better result (see Figure 5.5 (b)).



(a) before (b) after

Figure 5.5: Shifting centroid of each cluster

Random sample 2D skeleton

As we can see in Figure 5.5, the skeleton generated after shifting does not look plausible in 2D: some branches are overlapped and some angles are not realistic. This is because our input source is a 3D tree, the distances between some nodes are along z axis not x axis (assume it is a xy-plane). The solution to the problem is using von Mises distribution, which is the circular analogue of the normal distribution on a line. We can achieve this by following steps:

1. Collecting all angles between each branches;
2. Calculating the location μ and concentration κ: μ and $1/\kappa$ are analogous to μ and σ^2 (the mean and variance) in the normal distribution;
3. Calculating $I_0(\kappa)$ which is the modified Bessel function of order 0:

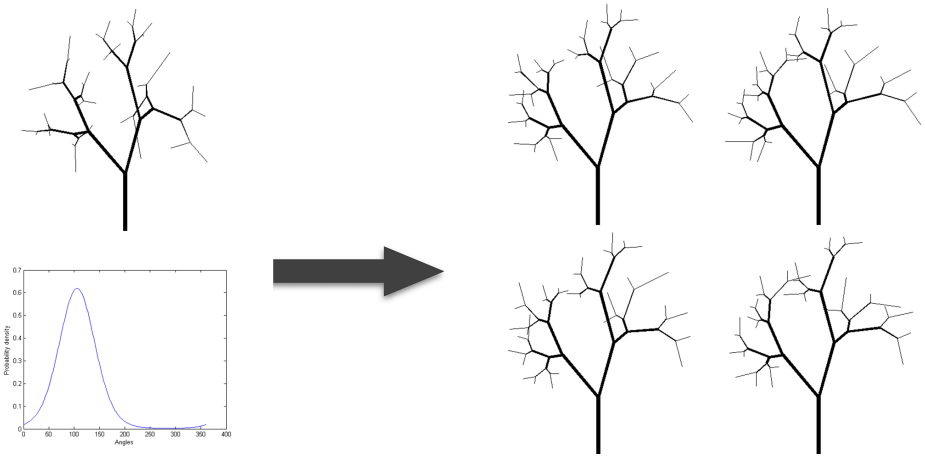
$$x^2\frac{d^2y}{dx^2} + x\frac{dy}{dx} + (x^2 - \alpha^2)y = 0 \tag{5.4}$$

4. Applying above parameters in von Mises probability density function (Equation 5.5), and then we get von Mises distribution (see Figure 5.6 (a));

$$f(x|\mu,\kappa) = \frac{e^{\kappa\cdot\cos(x-\mu)}}{2\pi I_0(\kappa)} \tag{5.5}$$

5. To generate new trees, we need to set the threshold, so only the angles with probability higher than the threshold will be chosen;
6. To keep the main structure identical, we only random sample angles start from level 3.

The new skeletons (see Figure 5.6 (b)) are similar but not identical to each other. Potentially an infinite number of unique skeletons can be generated.



(a) (b)

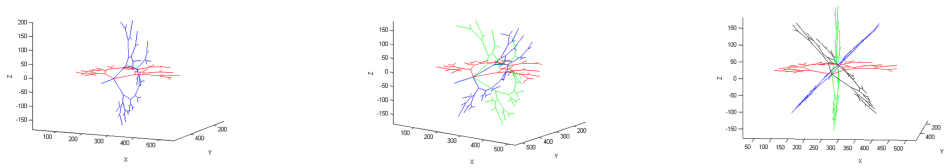
Figure 5.6: Random sample 2D skeleton: (a) Von Mises Probability density of angles between branches; (b) new 2d skeletons are generated.

5.5 3D skeleton modelling

There are several ways to achieve this, our system is using the method proposed by Li et al. (2011). The whole process consists of two steps: firstly we need to copy 2D skeletons, and then push branches to the right directions.

5.5.1 Copy operation

Our system generates 2D skeleton in xy-plane, we need to make several copies. We start making one duplicate skeleton in yz-plane (see Figure 5.7 (a)). Users are allowed to specify the number of copies to control the density of branches (see Figure 5.7 (b), (c)). In practice, we found $3 - 5$ copies are sufficient.



(a) 2 copies (b) 3 copies (c) 4 copies

Figure 5.7: Copy operation: duplicate the original 2D skeleton.

5.5.2 Pushing operation

The 2D skeletons look flat, we need to push branches away to create a botanically plausible structure. Each branch is pushed in a perpendicular direction (e.g., all branches in xy-plane are being pushed along z direction). The problem now is how far should we push each branch. To solve this problem, Li et al. (2011) introduced two terms: the *local term* $p(X_{i-1}, x_i)$, which keeps the local branch pattern looks natural after pushing the current node x_i; and the *global term* $p(\Omega|X_{i-1}, x_i)$ keeps the overall volumetric shape. These two terms provide a probabilistic solution, which maximizes the posterior probability defined by the Bayes' rule:

$$p(x_i|\Omega, X_{i-1}) \propto p(\Omega|X_{i-1}, x_i)p(X_{i-1}, x_i) \tag{5.6}$$

The pushing is performed following a root-to-leaf traversal. At each step i, all the sub-branches of x_i are pushed by the same distance that we get from above.

Local term

Local term can be formulated as:

$$p(X_{i-1}, x_i) = p(X_{i-1}|x_i)p(x_i) \tag{5.7}$$

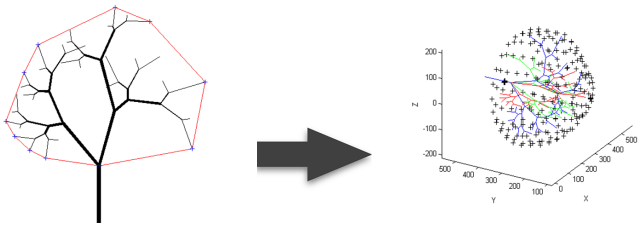
where the prior $p(x_i)$ prevents node x_i from being over-stretched when its tip is pushed, and the conditional probability $p(X_{i-1}|x_i)$ constrains the shape of each branch by examining the angle between the branch and its parent branch.

We begin with defining $p(x_i)$ as a uniform distribution over a range of value $[x_i - \delta, x_i + \delta]$ along the node's pushing direction. In practice, we set $\delta = \frac{1}{5} \times$ (tree width).

Then we calculate $p(X_{i-1}|x_i)$:

$$p(X_{i-1}|x_i) \triangleq exp\left(-\frac{(\alpha_i - \mu)^2}{2\sigma_i^2}\right) \tag{5.8}$$

Here α_i is the angle between the branch a_{ji} and its parent, μ is the expected angle. In practice, μ can be the average angle of all angles between branches, or we could get an angle using the von Mises distribution that we defined before (randomly get any angle with probability higher than a threshold). σ_i controls the width of the distribution: smaller values keep the actual push closer to the expected angle μ. In practice, we set $\sigma_i = \pi(h_{max} - h_i)/h_{max}$ where h is the level of hierarchy.



(a) 2D Envelope (b) 3D Envelope

Figure 5.8: Create a envelope of tree

Global term

To calculate the global term, we need to create the envelope Ω for our model. We are doing this by passing all points of a 2D skeleton to function `K = convhull(points)`, then we get `K`, which contains the out most points of the 2D model. We connect points in `K` to get the 2D envelope, which has been presented in Figure 5.8 (a). Then we evenly plant m attractors on the 2D envelope. For each point, we then evenly plant n attractors to form a circle on xz-plane (assume the 2D skeleton is in xy-plane). So now we have a 3D envelope with $m \times n$ attractors (see Figure 5.8 (b)).

Next, we need to calculate the total distance D from the envelope to the 3D skeleton:

$$D(\Omega, X) = \sum_{k=1}^{m} \min\left(|\omega_k - X|\right) \tag{5.9}$$

where $\omega_k = 1 : m \times n$.

The density of the attractors on the envelope can be controlled by the user. We keep the total number of the attractors to be $200 - 300$, which provide a good balance between the accuracy and computational efficiency.

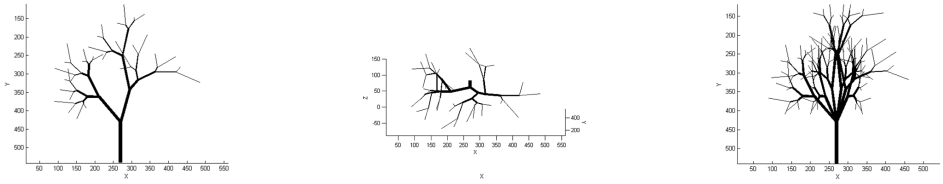
With the distance function, we formulate the global term as:

$$p(\Omega | X_{i-1}, x_i^l) = \frac{E(\Omega, X_{k-1}, x_i^l)}{\sum_{j=1}^{L} E(\Omega, X_{i-1}, x_i^l)} \tag{5.10}$$

Here $E(\Omega, X_{k-1}, x_i) = D(\Omega, X_{i-1}) - D(\Omega, X_i)$, x_i^l is one of the L possible solutions for x_i. We set the probability of x_i to 0 if it is outside of the envelope.

Choosing pushing distance

For each node x_{ji}, our system exhaustively tries all possible values within $[x_i - \delta, x_i + \delta]$: the quality of the branch angle is measured using Equation 5.8; the quality of the space fill



(a) (b) (c)

Figure 5.9: Pushing branches: (a) One copy, front view; (b) One copy, bottom view; (c) Three copies, front view.

is measured using Equation 5.10. We choose the distance that maximized Equation 5.6 as the best pushing distance. Figure 5.9 shows the result of pushing branches away to form a 3D skeleton.

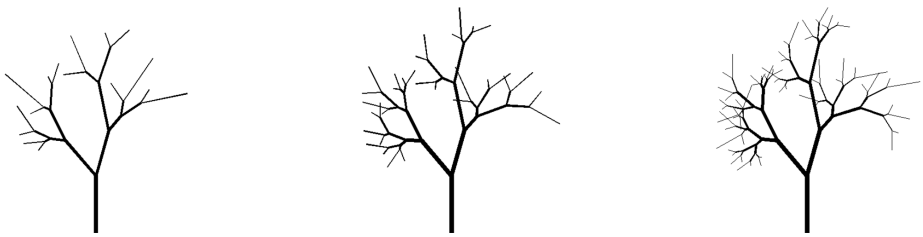
Chapter 6

Testing and Results

Our system provides flexible user controls for editing the output models. In this chapter, we present side-by-side comparisons of controlling available parameters to produce different trees, which could be used in various circumstances.

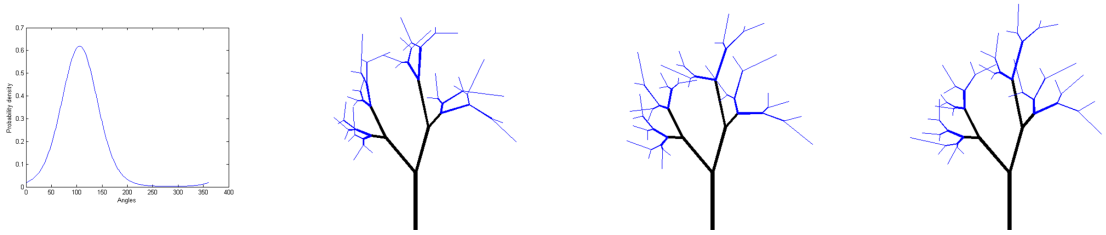
6.1 Number of hierarchy levels

Figure 6.1 shows a side-by-side comparisons of 2D skeletons with different levels of hierarchies. 4-level hierarchy does not provide a plausible look since it is too sparse, but this maybe suitable for trees with less sub-branches such as pine tree. 6-level hierarchy gives a better shape with much more branches, but the tip branches are too crowded especially in 3D model, it also doubles the time needed for a 5-level hierarchy. 5 levels are sufficient for many tree species. For the following tests in this chapter, we will use 5 levels hierarchy unless specified.



(a) 4 levels (b) 5 levels (c) 6 levels

Figure 6.1: 2D skeletons with different levels of hierarchies



(a) von Mises (b) threshold = 0.2 (c) threshold = 0.4 (d) threshold = 0.6

Figure 6.2: 2D skeletons with random angles: branches with blue colour use random angles.

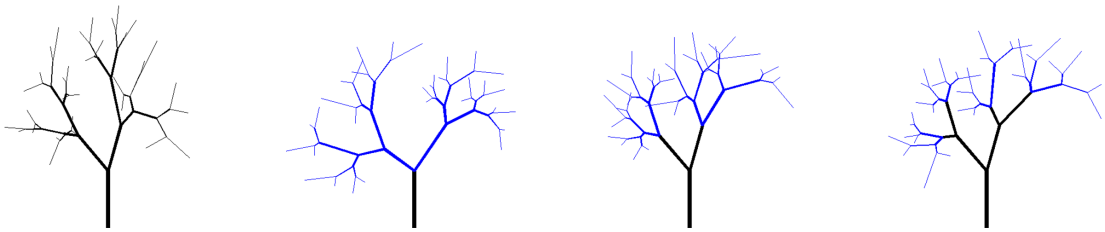
6.2 Random angle threshold

When using von Mises distribution to random sample angles between branches, a threshold value (same as probability density in Figure 6.2 (a)) can be set by users. System will evaluate each random result, if the angle give a probability value less than the threshold, system will then random a new angle, and it will keep repeating until a valid angle is found.

If we set threshold too low, the angles would vary too much (see Figure 6.2 (b)), so the overall shape of whole tree looks implausible. On the other hand, if we see the threshold too high, all angles would be almost the same (see Figure 6.2 (d), it looks artificial. We need to test different threshold values to find a optimized one, otherwise we actually lost the advantage of using the probability density function. For our example tree, values between $0.35 - 0.45$ give the best results (see Figure 6.2 (c)). Again, the following tests in this chapter, we will set $threshold = 0.4$ unless specified.

6.3 Random angle starting level

The random angle starting level decides the overall shape of tree model. Higher starting level results in more similar outputs. Figure 6.3 shows the outputs with different random angle starting level.



(a) Original tree (b) Starting level: 1 (c) Starting level: 2 (d) Starting level: 3

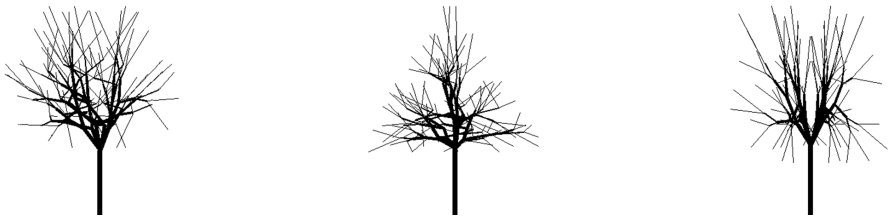
Figure 6.3: 2D skeletons with different random starting level

There is no better one without specify the environment. If users want to create a forest where trees grow wildly, it is better to start from low level to generate various structures. On the other hand, if users want to present a garden where trees are trimmed regularly, then a higher starting level will required to produce neat trees.

6.4 Control volumetric shape

Our system allows user to control the volumetric shape of output models. In Figure 6.4, we present how 3D tree branches fill an unusual shape volume (a vase). All user have to do is inputting an alpha image (see Figure 6.4 (b)). Note that users need to make sure the alpha image is not larger than original envelope, otherwise the output will not changes since all points are inside the white area. A new envelope will be created base on the alpha image. All branches outside the envelope will be cut, and branches inside will be pushed to fill the new envelope. Users could use this feature to control the shapes of output 3D skeletons.

(a) Vase (b) Alpha image (c) 3D skeleton

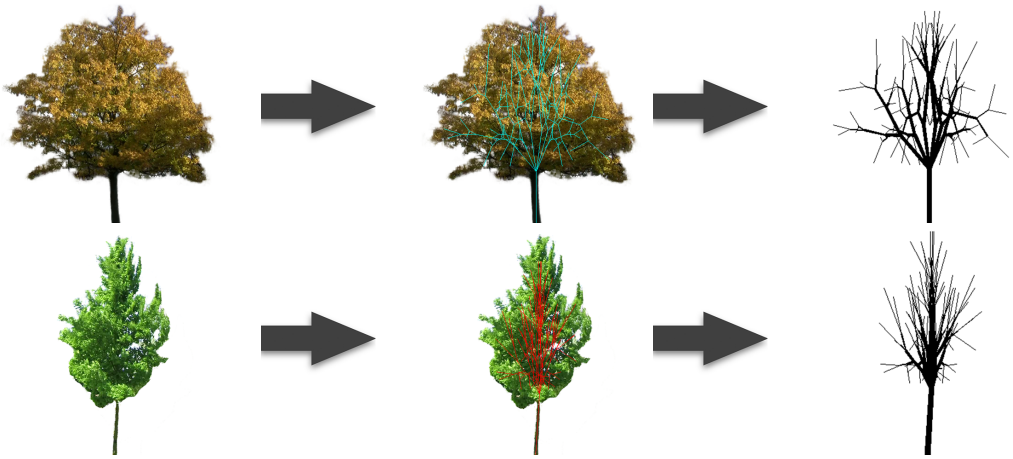


(d) Circle (e) Triangle (f) Rectangle

Figure 6.4: Control the shape of 3D skeleton

6.5 Results

We have tested our system using different input videos. Figure 6.5 shows the outputs of 3D skeletons of two different trees generate by our system. It takes less than 20 seconds to produce a 3D skeleton and less than 10 seconds to random sample a new 3D skeleton. Indeed, we could not provide a perfect reconstruction of the initial tree from video, but our



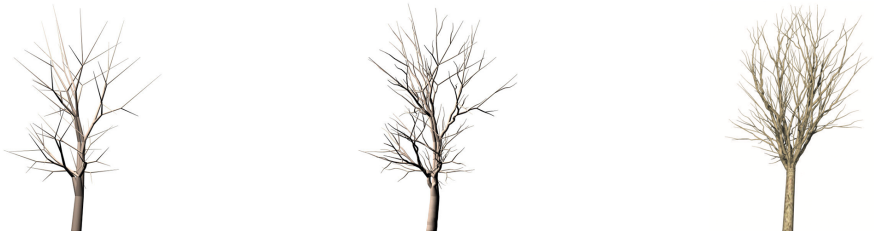
(a) Input trees (b) Comparisons (c) Output skeletons

Figure 6.5: Output of 3D skeletons

system could be used in some areas, which do not require a 3D tree skeleton that identical to the real tree, such as creating trees along highway or a forest.

Details of output skeletons are showed in Appendix A. A video of 3D skeleton is also available on attached CD.

There are several things we could do to improve the outputs comparing to Li et al.'s (2011) system (see Figure 6.6), such as curling branches, adding texture and lightings.



(a) 3D skeleton (b) Curliness (c) Adding texture

Figure 6.6: 3D skeletons from Li et al. (2011)

Chapter 7

Conclusions

7.1 Overview

At beginning of this project, we aimed to address the 3D tree modelling problem using entirely automatic approach. This is a coding intensive project, which requires strong mathematical skills. We could not manage to add motions and leaves to 3D skeleton due to the limited of time. At the early stage, we spent too much time on setting up development environment such as building `.mex` files so we can use OpenCV interface and installing decoders for MATLAB to read input videos. On the other hand, our mathematical background are not strong enough to support us finishing the whole project: we need to spend a large amount of time on understanding relevant research papers rather than implementation and testing. Overall, the process of development followed our plan well, we achieved most of our milestones. Most importantly, we are able to present a fully functional system to generate 3D tree skeleton in an almost entirely automatic approach.

7.2 Project achievements

In this paper we described a method for generating 3D tree skeletons from an input video and showed how to automatically create various similar skeletons. Our approach uses video as an input resource, this reduces user interaction to outlining tree in first frame. Users can control the shape of skeleton by inputting a new alpha image. The density of branches can be controlled by specifying the number of levels.

The von Mises distribution is used to generate more similar but not identical tree models. Our probability formulation takes account of local and global constraints to ensure an optimal structure. Another probability method is also used to control the angles of bifurcations to produce plausible branches while generating similar skeletons.

We have provided side-by-side comparisons to show different output models by controlling

different parameters. Therefore, users are able to create different 3D tree skeletons for various circumstances.

7.3 Future work

A restriction is the use of binary skeleton. All branches have only two bifurcations, which is not realistic, but this does not significantly affect the final appearance. Another reason is that there are various methods and algorithms applicable while using binary skeleton. As a result, our system is not able to create species such as willows, palm trees, algae and any other trees species whose branches have special shapes or even no branch at all. However, this could be an interesting path for future work.

Another area for further development is to model the motion of 3D skeletons. This can be done in two different ways: (1) using the tracking data from the video, and then transform the tracked 2D motion to 3D that moves the model in a realistic way described by Li et al. (2011); (2) using multiple views to tracking motion in 3D used in Habel et al. (2009), this could also improve the realism of our output models.

Bibliography

Abramowitz, M. and Stegun, I. A. (1964), *Handbook of mathematical functions: with formulas, graphs, and mathematical tables*, number 55, Courier Corporation.

Barnea, D. I. and Silverman, H. F. (1972), 'A class of algorithms for fast digital image registration', *Computers, IEEE Transactions on* **100**(2), 179–186.

Chen, X., Neubert, B., Xu, Y.-Q., Deussen, O. and Kang, S. B. (2008), *Sketch-based tree modeling using Markov random field*, Vol. 27, 5, ACM.

Deussen, O. and Lintermann, B. (2006), *Digital design of nature: computer generated plants and organics*, Springer Science & Business Media.

Forbes, C., Evans, M., Hastings, N. and Peacock, B. (2011), *Statistical distributions*, John Wiley & Sons.

Habel, R., Kusternig, A. and Wimmer, M. (2009), Physically guided animation of trees, *in* 'Computer Graphics Forum', Vol. 28, Wiley Online Library, pp. 523–532.

Harris, C. and Stephens, M. (1988), A combined corner and edge detector., *in* '4th Alvey vision conference', Vol. 15, Manchester, UK, pp. 189–192.

Ijiri, T., Owada, S. and Igarashi, T. (2006), The sketch l-system: Global control of tree modeling using free-form strokes, *in* 'Smart Graphics', Springer, pp. 138–146.

Levin, A., Lischinski, D. and Weiss, Y. (2008), 'A closed-form solution to natural image matting', *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **30**(2), 228–242.

Lhuillier, M. and Quan, L. (2005), 'A quasi-dense approach to surface reconstruction from uncalibrated images', *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **27**(3), 418–433.

Li, C., Deussen, O., Song, Y.-Z., Willis, P. and Hall, P. (2011), 'Modeling and generating moving trees from video', *ACM Trans. Graph.* **30**(6), 127:1–127:12.
URL: *http://doi.acm.org/10.1145/2070781.2024161*

Lindenmayer, A. (1968), 'Mathematical models for cellular interactions in development i. filaments with one-sided inputs', *Journal of theoretical biology* **18**(3), 280–299.

Liu, C., Torralba, A., Freeman, W. T., Durand, F. and Adelson, E. H. (2005), 'Motion magnification', *ACM Trans. Graph.* **24**(3), 519–526.
URL: *http://doi.acm.org/10.1145/1073204.1073223*

Lucas, B. D., Kanade, T. et al. (1981), An iterative image registration technique with an application to stereo vision., *in* 'IJCAI', Vol. 81, pp. 674–679.

Neubert, B., Franken, T. and Deussen, O. (2007), Approximate image-based tree-modeling using particle flows, *in* 'ACM SIGGRAPH 2007 Papers', SIGGRAPH '07, ACM, New York, NY, USA.
URL: *http://doi.acm.org/10.1145/1275808.1276487*

Okabe, M., Owada, S. and Igarash, T. (2005), Interactive design of botanical trees using freehand sketches and example-based editing, *in* 'Computer Graphics Forum', Vol. 24, 3, Wiley Online Library, pp. 487–496.

Pirk, S., Niese, T., Deussen, O. and Neubert, B. (2012b), 'Capturing and animating the morphogenesis of polygonal tree models', *ACM Transactions on Graphics (TOG)* **31**(6), 169.

Pirk, S., Stava, O., Kratt, J., Massih Said, M. A., Neubert, B., Mech, R., Benes, B. and Deussen, O. (2012a), 'Plastic trees: interactive self-adapting botanical tree models'.

Porter, T. and Duff, T. (1984), Compositing digital images, *in* 'ACM Siggraph Computer Graphics', Vol. 18, 3, ACM, pp. 253–259.

Prusinkiewicz, A. L. P., Lindenmayer, A., Hanan, J. S., Fracchia, F. D. and Fowler, D. (1990), 'The algorithmic beauty of plants with'.

Prusinkiewicz, P., Samavati, F., Smith, C. and Karwowski, R. (2003), 'L-system description of subdivision curves', *International Journal of Shape Modeling* **9**(01), 41–59.

Reche-Martinez, A., Martin, I. and Drettakis, G. (2004*a*), 'Volumetric reconstruction and interactive rendering of trees from photographs', *ACM Trans. Graph.* **23**(3), 720–727.
URL: *http://doi.acm.org/10.1145/1015706.1015785*

Reche-Martinez, A., Martin, I. and Drettakis, G. (2004*b*), Volumetric reconstruction and interactive rendering of trees from photographs, *in* 'ACM Transactions on Graphics (TOG)', Vol. 23, 3, ACM, pp. 720–727.

Rhemann, C., Rother, C., Wang, J., Gelautz, M., Kohli, P. and Rott, P. (2009), A perceptually motivated online benchmark for image matting, *in* 'Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on', IEEE, pp. 1826–1833.

Shi, J. and Malik, J. (2000), 'Normalized cuts and image segmentation', *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **22**(8), 888–905.

Talton, J. O., Lou, Y., Lesser, S., Duke, J., Měch, R. and Koltun, V. (2011), 'Metropolis procedural modeling', *ACM Transactions on Graphics (TOG)* **30**(2), 11.

Tan, P., Zeng, G., Wang, J., Kang, S. B. and Quan, L. (2007), Image-based tree modeling, *in* 'ACM Transactions on Graphics (TOG)', Vol. 26, 3, ACM, p. 87.

Wang, R., Yang, Y., Zhang, H. and Bao, H. (2014), Variational tree synthesis, *in* 'Computer Graphics Forum', Vol. 33, 8, Wiley Online Library, pp. 82–94.

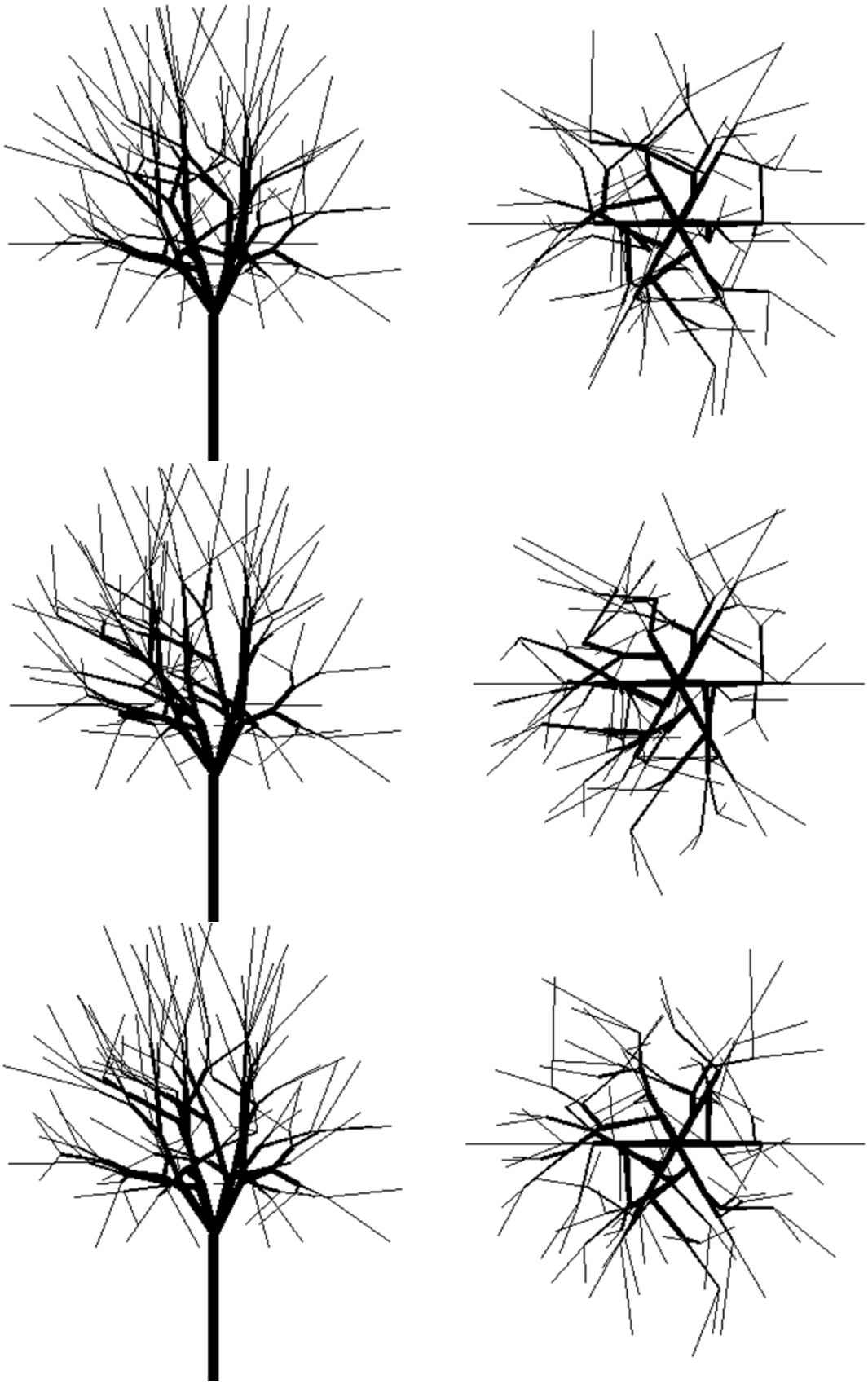
Appendix A

Raw results output

Figure A.1 shows the original trees from two different videos. We use our system to generate 2 sets of 3D skeletons (see Figure A.2, A.3). We are able to create several similar but not identical trees based on the original data within a few minutes.

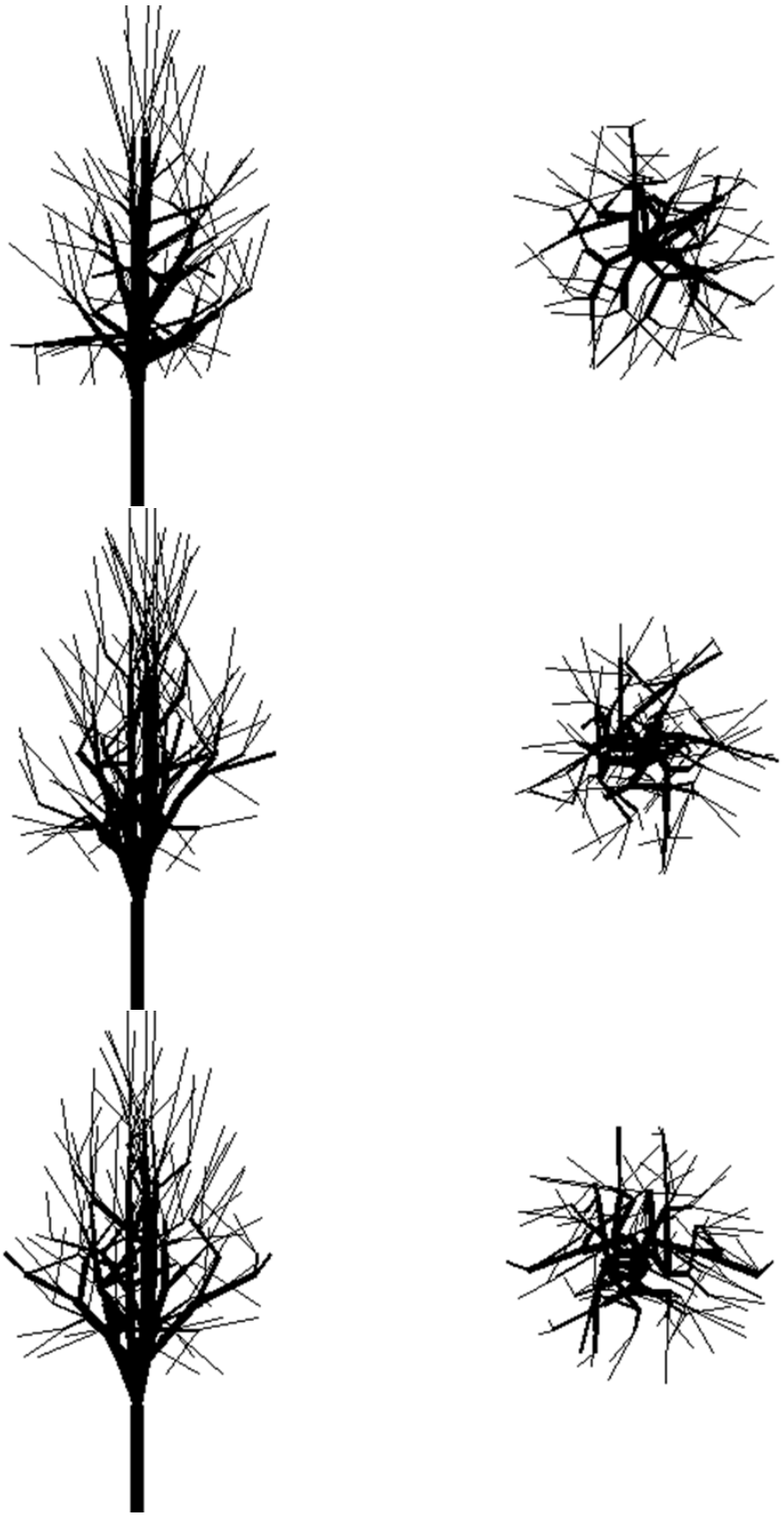
(a) Tree 1 (b) Tree 2

Figure A.1: The original trees



(a) Front view (b) Top view

Figure A.2: 3D skeletons of Tree 1



(a) Front view (b) Top view

Figure A.3: 3D skeletons of Tree 2

Appendix B

User Documentation

B.1 Working Environment

Our system is developed using MATLAB R2013a in Windows 7 Ultimate 64-bit.

We also tested in MATLAB R2013b and R2014a. Users need to be careful if using newer versions. Some functions in standard library may be removed. Our system also capable in OS X Yosemite and Ubuntu 13.10 64-bit.

B.2 Input Sources

Our system accepts both video and a set of consecutive images of a moving tree as input source. The output of those two input sources are identical. The information we need from input sources are the positions of feature points in each frame. Users could save those information (e.g., save as `.mat` file in MATLAB), so for the next time, users could use this `.mat` file as input source, this could help to save massive time spend on reading videos.

For users who want to use video as an input source, you need to make sure the video decoder of your video type has been installed successfully. Otherwise MATLAB is not able to read any information from the video.

We recommend to use a set of consecutive images. Users could extract images out of the video. In addition, we do not need each frame, one out of ten frames will be enough in practice. Since the moving distance of branches between each frame is tiny, this could also help to reduce unnecessary noise.

High resolution videos or images are preferred but not necessary. We recommend users to chose higher resolution to create a more accurate 3D model. The trade off is, higher resolution requires more disk spaces and reading time.

Users need to outline tree in first frame, which is the only user interaction required in the

whole process. This helps our system to remove background of the image. Users do not have to draw the exactly outline since the system could handle this by ignoring feature points in the outline.