Knowledge and Heuristic Based Modeling of Laser-Scanned Trees

Hui Xu
and
Nathan Gossett
and
Baoquan Chen
University of Minnesota

We present a semi-automatic and efficient method for producing full polygonal models of range scanned trees, which are initially represented as sparse point clouds. First, a skeleton of the trunk and main branches of the tree is produced based on the scanned point clouds. Due to the unavoidable incompleteness of the point clouds produced by range scans of trees, steps are taken to synthesize additional branches to produce plausible support for the tree crown. Appropriate dimensions for each branch section are estimated using allometric theory. Using this information, a mesh is produced around the full skeleton. Finally, leaves are positioned, oriented and connected to nearby branches. Our process requires only minimal user interaction, and the full process including scanning and modeling can be completed within minutes.

Categories and Subject Descriptors: I.3.3 [**Computer Graphics**]: Picture/Image Generation—*Digitizing and scanning*

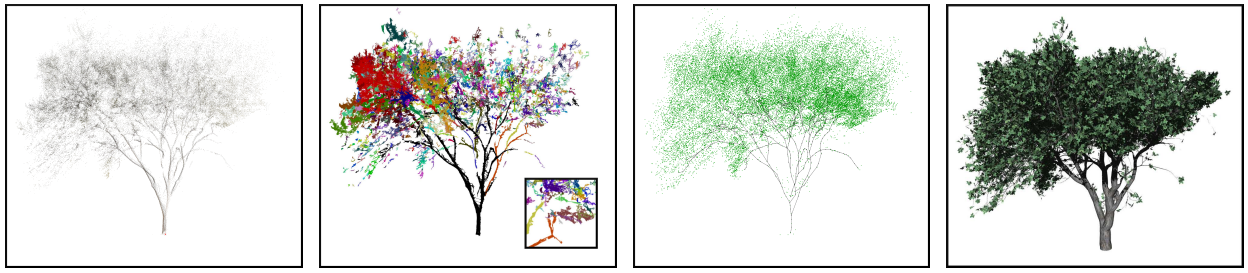
Additional Key Words and Phrases: Digitizing and Scanning, knowledge-based modeling

1. INTRODUCTION

The increasing availability and power of range scanners has enabled us to scan larger and more complex objects. Trees, however, pose special problems for such scanning. Unlike buildings, which have relatively simple geometry (which is to say their macro-structure is usually simple, not that they do not have detailed meso- or micro-structure) and remain stationary during scanning, trees are complex and make inconvenient movements during scanning due to wind. In addition, the small branches that make up the crown of the tree are often missing from the scan because of limited scanning resolution and occlusion from leaves and other branches. When scanning outdoors, it is often infeasible to obtain multiple scans of each object, especially if the purpose of the scan is to acquire architectural data rather than the trees themselves. In our experience, unless the scans are taken specifically to capture a particular tree, most trees appear in only a single scan. Standard mesh generation techniques often fail because of this inadequate data. In applications where the goal is to authentically reproduce a specific environment, inaccurate reproductions of significant objects such as trees can be distracting to a viewer, particularly if they are familiar with the real environment that was scanned.

Our purpose in this work is to use knowledge about the structure of trees to produce full polygonal meshes from the point clouds obtained from outdoor scans. We aim to produce a mesh that plausibly recreates the tree that was

Author's address: Hui Xu (now at Google, Inc.), 1600 Amphitheatre Parkway, Building #43, Mountain View, CA 94043. Nathan Gossett, Baoquan Chen, 499 Walter Library, 117 Pleasant St SE, Minneapolis, MN 55455



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

Fig. 1. Modeling a real tree: (a) A point cloud produced by a single scan of a real tree with the root point marked. (b) A graph is formed by connecting neighboring points. Each connected subgraph is shown here using a distinct color. (c) A skeleton is produced from the graph, and leaf locations (green dots) are identified. (d) The skeleton is extended by synthesizing new small branches growing into the tree crown. A mesh is reconstructed based on the skeleton and leaves are added.

originally scanned, but with extrapolated details that were missing in the point cloud. These surrogate models can then be used in place of the original point clouds for rendering.

The remainder of the paper is organized as follows. After a brief discussion of previous work (Section 2), we provide an overview of our tree modeling pipeline (Section 3). We then provide details of our techniques for producing skeletons from point clouds (Section 4), producing meshes from skeletons (Section 5), and attaching leaves to branches (Section 6). Some implementation details are given in Section 7. Results (Section 8) are followed by a discussion of limitations and plans for future work (Section 9), and finally a conclusion (Section 10).

2. PREVIOUS WORK

The modeling of trees has a long history in computer graphics and remains an active area of research. The L-System, introduced by Lindenmayer [1968] as the foundation of an axiomatic theory of biological development, has been widely used for tree modeling in computer graphics [Prusinkiewicz and Lindenmayer 1990]. However, while L-System-based methods are often quite good at producing examples of a particular species of plant, they have seldom been used to reproduce individual existing plants. An exception to this are the modeling tools developed by Prusinkiewicz et al. [2001]. However, while their tools are capable of reproducing the appearance of an individual photographed plant, this duplication seems to be dependant on significant skilled user interaction.

Since the demands of realism make it desirable to model real trees, many image-based methods, usually taking photographs of trees as input, have been proposed. Among these methods, billboard-type modeling and rendering methods are favored in real-time systems for their efficiency [Ervin and Hasbrouck 2001]. Shlyakhter et al. [2001] have proposed a method for constructing 3D tree models from instrumented photographs. Their approach first approximates a tree skeleton based on the tree shape as constructed from the instrumented photographs, and then fits an L-system to the skeleton. The remainder of the tree model is grown by the L-system. Recently Reche et al. [2004] have developed a volumetric tree modeling and rendering method from photographs. The above methods emphasize appearance and do not place a priority on modeling tree structures that conform to real trees. Our method differs from these methods in that we target applications where the trees are not necessarily the focus during the data acquisition process. Instead, we seek methods to work with data that might be found in pre-existing data sets. However, in doing so, we still intend to generate tree structures as faithful as possible.

Advances in laser scanning technology are making 3D acquisition feasible for large and complex objects like trees.

The tree data acquired through 3D scanning is normally in the form of point clouds. Due to the complexity of trees, constructing 3D models from these point clouds is a challenging task. Current research has mainly focused on extracting high level structures and features of scanned tree data for forestry and ecological purposes [Gorte and Pfeifer 2004; Pfeifer et al. 2004; Pyysalo and Hyypp 2002]. Pfeifer et al. [2004] have utilized a volumetric method for extracting skeletons from points and built meshes for stems using cylinders, but their method assumes that the trees are thin and without leaves (i.e., scanned in winter). Moreover, their method aims to find only prominent structures of trees rather than to construct a tree model with fine details. In this paper, we propose a novel method that takes advantage of the current knowledge of tree structures to generate detailed tree models from point clouds. Our process requires only minimal user interaction, and the full process including scanning and modeling can be completed within minutes.

3. OVERVIEW

Our task is to build a plausible tree model from the point cloud of a laser-scanned tree. As stated earlier, our strategy is to construct a tree model by using knowledge of tree structures. First, we produce the skeleton of the trunk and main branches (Section 4.1). Next, we extend this skeleton to include all main branches that are not yet structurally connected to the base of the tree (Section 4.2). To complete the model, we identify and locate the tree's leaves and synthesize new branches to ensure that all leaves are connected to the tree. To ensure the property of self-similarity, we synthesize new structures based on the existing skeleton structure (Section 4.3). Once a full skeleton has been produced, we determine the thickness of each branch section and construct a mesh along the skeleton (Section 5). Finally, leaves, modeled using alpha textured quads, are oriented and connected to the nearest branches (Section 6).

Our algorithm is able to handle two broad categories of trees: those that grow predominantly upwards, and those that feature significant horizontal growth. We have found that trees from the same category react similarly to our algorithm. For instance, the Elm, Ash and Cottonwood trees commonly found in our area all share a similar basic structure of upwards growth. We use a point cloud scanned from a 20-meter tall American Elm tree (*Ulmus Americana*) as an example input to illustrate our algorithm and demonstrate our results on trees of this type. We also demonstrate our method on a mature Crab Apple tree in Section 8 as an example of an alternate crown architecture featuring more prominent horizontal growth found in shorter species such as fruit trees. Finally, we discuss limitations of our approach and species that lie outside the scope of our algorithm in Section 9.

Our pipeline is demonstrated using a single scan. A comparison between a model produced from a single scan and one produced from multiple registered scans can be found in Figure 11. All parameter values suggested in later sections are based on trees of this size and species. Reproducing a tree of a different size or species would simply require a user to obtain the appropriate values for these parameters from reference literature, or if that is not available, from measurement. Selected parameters are gathered in Table I and general guidance is given for their derivation.

4. POINTS TO SKELETON

Given an under-sampled point cloud of a laser-scanned tree, we first find the skeleton of the tree. To do that, we connect the points to form a graph and utilize a single-source shortest path algorithm to produce the skeleton, which is then grown to the crown.

4.1 Main Skeleton Production

Figure 1(a) shows a point cloud of a laser-scanned tree. We assume that the root point of the tree, marked in the figure, has been located (e.g., selected manually or by taking the lowest point in the scan). In this step, each point is connected to all other points within its local neighborhood, forming a weighted graph as shown in Figure 1(b), where the weight of each edge in the graph is the length of the edge. The local neighborhood of each point is defined by a 3D distance (0.2m in our case) that is dependent on the scanning resolution, precision and distance. Limited scanning resolution and occlusion from leaves and branches often causes an incomplete point cloud. Therefore, the graph is

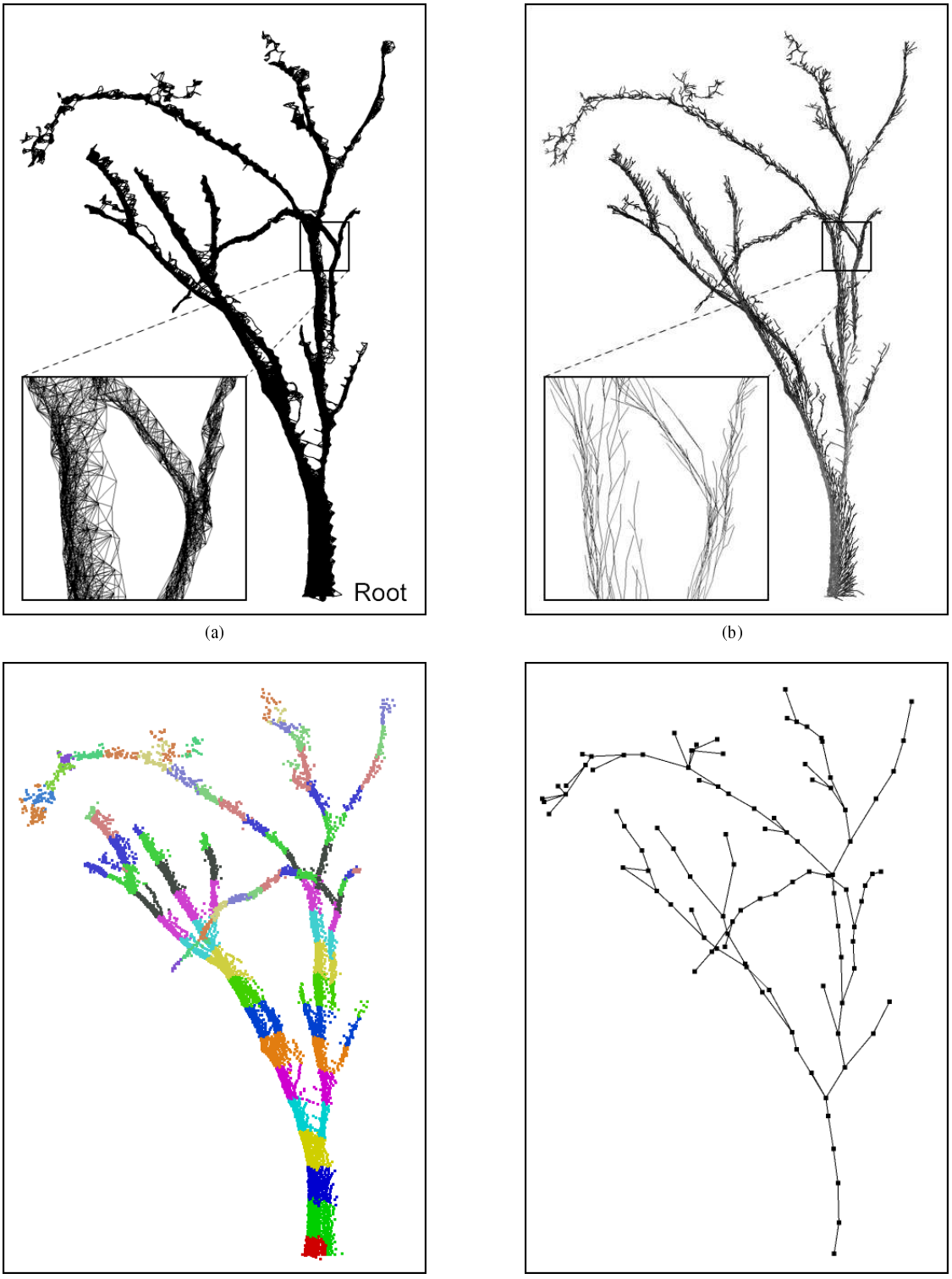
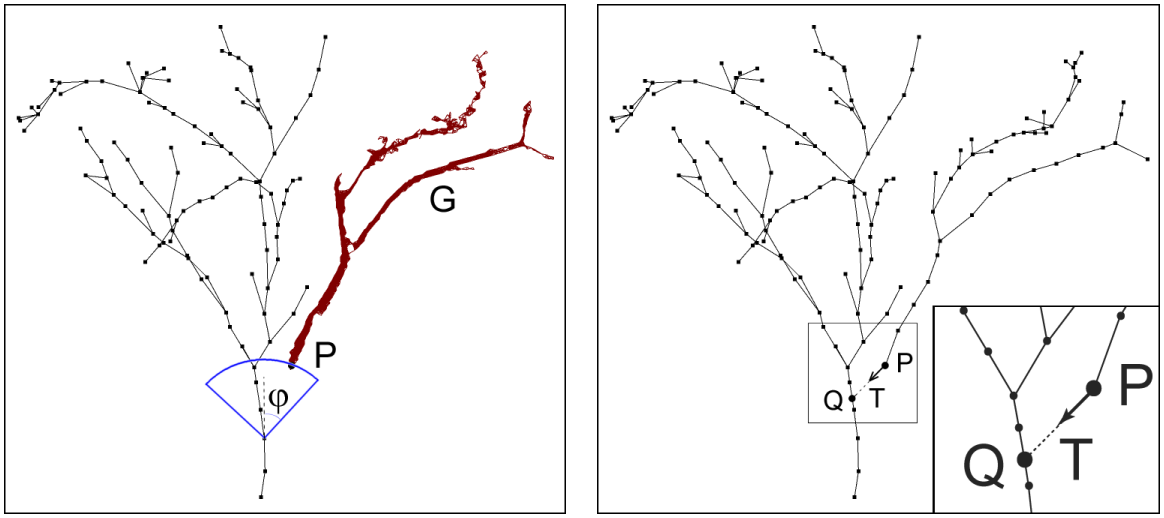
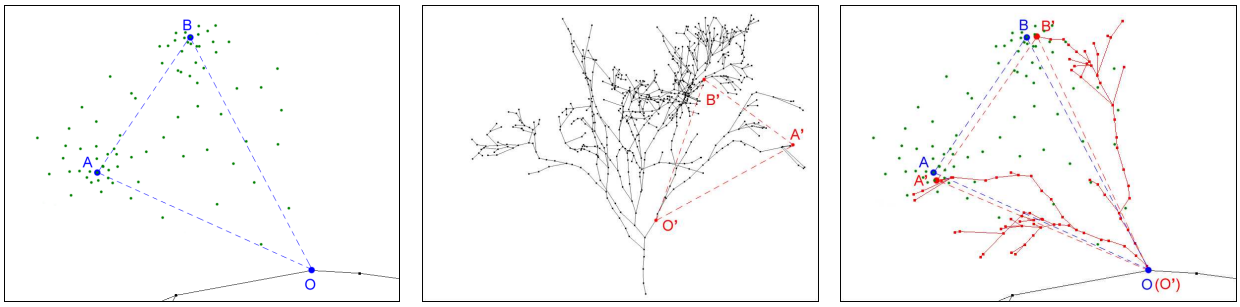


Fig. 2. Main skeleton production: (a) The connected subgraph that contains the root point is selected from the graph shown in Figure 1(b). (b) Shortest paths from the root to all other points are calculated. (c) The lengths of the shortest paths are quantized and the points are clustered into bins. (d) A skeleton is formed by connecting the centroid points of the adjacent bins.



(a) (b)

Fig. 3. Main skeleton extension: (a) A candidate subgraph is located by traversing the main skeleton. (b) The skeleton of the candidate subgraph is produced and attached to the main skeleton.



(a) (b) (c)

Fig. 4. Skeleton synthesis: (a) Skeleton node O along with unsupported leaf locations A and B. (b) $\triangle O'A'B'$ is located within the existing skeleton. (c) $\triangle O'A'B'$ is copied, transformed and attached at node O.

usually disconnected, but has many connected subgraphs as seen in Figure 1(b).

We start producing the skeleton from the connected subgraph that contains the root point, as shown in Figure 2(a). To do so, a single-source shortest path algorithm is first applied to find the shortest paths from the root to all other points. Figure 2(b) demonstrates the shortest paths calculated by Dijkstra's shortest path algorithm. Once the shortest path is calculated for every point, the lengths of the shortest paths are quantized and the points are clustered into bins based on these lengths and graph adjacency, similar to the method used in [Brostow et al. 2004]. Figure 2(c) shows an example of this quantization in which bins for each level are shown with unique colors. The centroid of the points in each bin is then calculated. Finally, the skeleton is formed by connecting the adjacent centroid points as shown in Figure 2(d). The centroid points are referred to as *skeleton nodes*, and this skeleton is referred to as the main skeleton. All of the points in a bin are declared to be *associated* with the bin's corresponding skeleton node.

The data structures used in our algorithms and some implementation details are presented in Section 7.

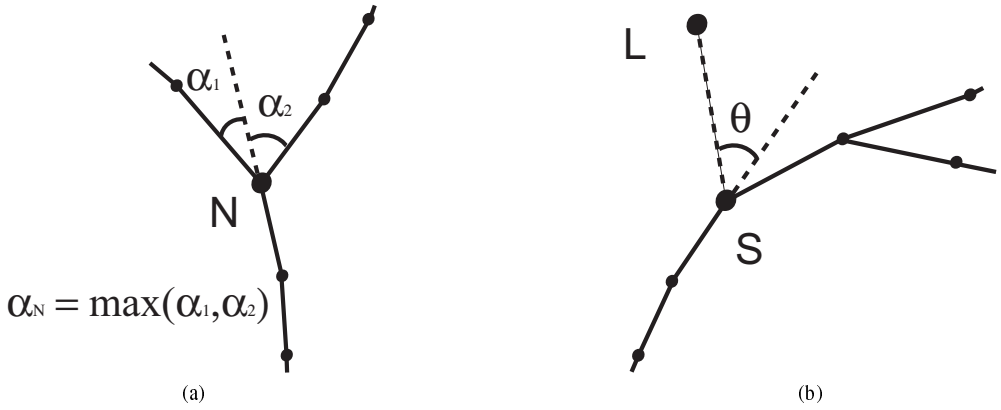


Fig. 5. Definitions of (a) axil and (b) a connection angle.

4.2 Main Skeleton Extension

As shown in Figure 1(b), some other connected subgraphs also represent additional main branches of the tree. Therefore, in order to better approximate the tree's structure, we extend the main skeleton by connecting it to those subgraphs. We use a bi-directional probing process to find connection points, as illustrated in Figure 3. A breadth-first traversal is performed on the main skeleton. During the traversal, at each skeleton node a cone of angle φ (e.g. 45^o) is projected along the direction from its parent to the node in the skeleton. When the cone first intersects with a connected subgraph G and the intersection point P is within a certain range r (e.g. 2.0m), we consider connecting G to the main skeleton (Figure 3(a)). First, the skeleton of G is produced using the same algorithm described in the previous section, treating the intersection point P as the root point in the algorithm (Figure 3(b)). Second, a tangent direction T is calculated at P by fitting a spline to the skeleton nodes close to P. Next, we project a cone from P along direction T. If the cone intersects with the main skeleton at Q within range r, we extend the main skeleton by connecting the skeleton of the subgraph with the line segment $\overline{PQ}$. In this case, Q is inserted as a skeleton node in the main skeleton if it does not already exist. Finally, the newly added part of the main skeleton is inserted at the end of the traversal queue.

Due to limited scanning resolution and occlusion from leaves and other branches, the connected subgraphs representing small branches and leaves in the crown are often irregular and chaotic. In order to prevent such subgraphs from being connected to the main skeleton, we process only nodes that have a shortest path that is less than a threshold (around $\frac{2}{3}$ of the tree height) during the traversal of the main skeleton. Figure 1(c) shows the resulting skeleton after connecting all acceptable connected subgraphs.

High scanning density and accuracy should produce a good main skeleton through this process. However, given the realities of outdoor scanning, we have found it useful to offer the user a chance to intervene at this stage, eliminating mistaken connections, adding missed connections, or correcting misguided links.

4.3 Skeleton Synthesis

We next need to extend the main skeleton obtained in the previous section to support the entire crown. As mentioned earlier, the small branches and twigs in the crown are usually inadequately sampled because of limited scanning resolution and natural occlusion. Therefore, it is impractical to construct such fine branches from the point cloud. Instead, we solve this problem by synthesizing the small branches based on the existing skeleton.

To guide the synthesis and ensure a match between the synthesized branches and the real appearance of the tree, we need to identify and locate the tree's leaves. The points in the subgraphs that are not connected to the main skeleton are considered to be leaves. These points are grouped based on a predefined neighborhood (say, a sphere of a certain

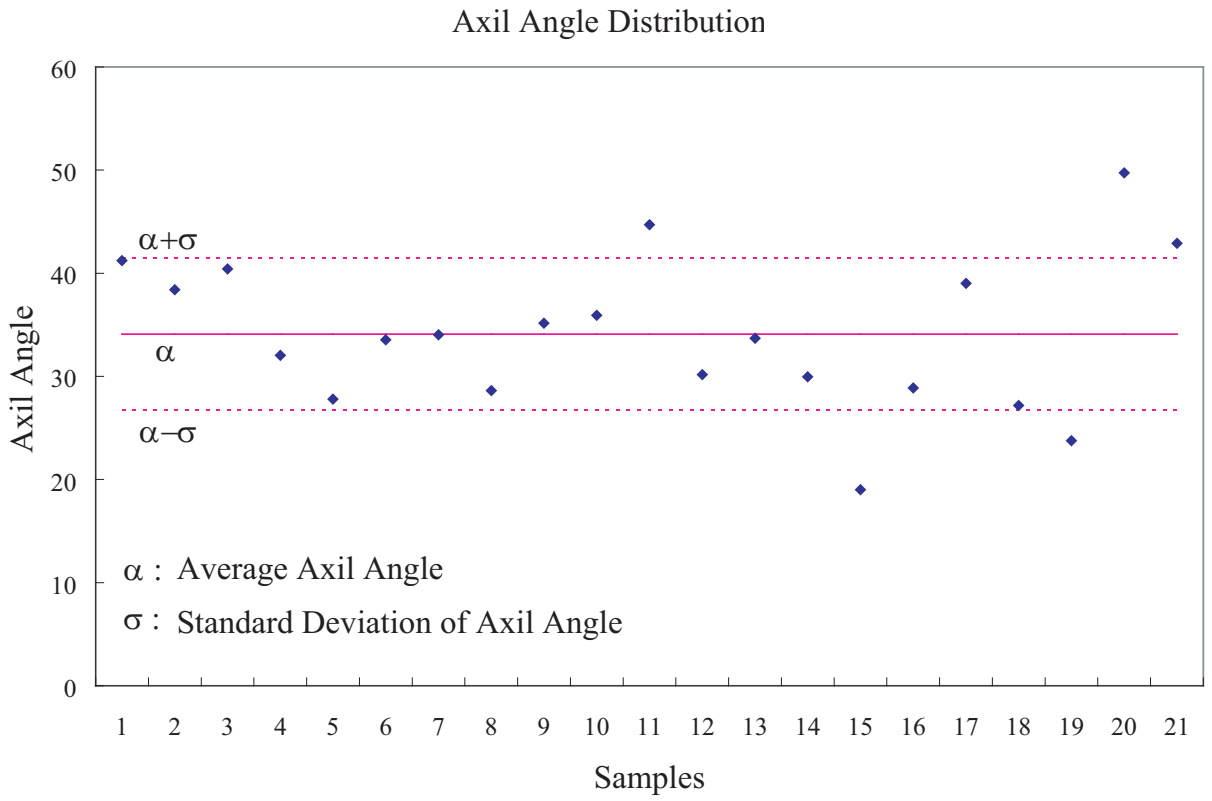


Fig. 6. Axil angle distribution of the skeleton in Figure 1(c). $\alpha = 34.1^o$. $\sigma = 7.4^o$.

radius based on leaf or leaf cluster size for the tree species in question). Each group defines a *leaf location* for placing leaves. These leaf locations, indicated as green dots in Figure 1(c), are used to guide the synthesis. It is worth noting that the groups obtained above may also include small branches, but these are referred to as leaf locations as well.

To ensure that all leaf locations are connected to the tree, we synthesize new branches that grow to reach them. While doing so, we must maintain the main characteristics of the existing skeleton, one of which is the angle formed between branches. We need to ensure that the angles formed when synthesized branches split from one to another are statistically consistent with angles in the existing skeleton.

For each skeleton node (except for the root), we define a direction pointing to it from its parent node. When the skeleton splits at a node N, the *axil angle* α_N is defined as the largest of the angles between the directions of its child nodes and that of its own, as shown in Figure 5(a). Let α be the average axil angle of the main skeleton, and σ be the standard deviation of the axil angles. With a normal distribution, the probability that a sample will fall into the intervals defined by

$$I_1 = [\alpha - \sigma, \alpha + \sigma] \tag{1}$$

and

$$I_2 = [\alpha - 2\sigma, \alpha + 2\sigma] \tag{2}$$

are around 68.27% and 95.45% respectively according to statistics [Weisstein 2005]. For the tree species we are

interested in, our experiments have also verified that the majority of the axil angles are indeed located in the interval I_1 defined by Equation 1. As an example, Figure 6 shows the axil angle distribution of the skeleton in Figure 1(c), where 16 out of 21 angles are from $\alpha - \sigma$ to $\alpha + \sigma$. Based on this knowledge, we give preference to synthesized angles in the interval I_1 (see definition of *feasible* below), and enforce that the synthesized angles must be in the interval I_2. This prevents any abnormally shaped structures from being produced during the synthesis process.

A *connection angle* between a leaf location L and a skeleton node S is defined as the angle between the skeleton node direction and direction $\overrightarrow{SL}$, as shown in Figure 5(b). A skeleton node is called *feasible* to a leaf location if the connection angle is in the interval I_1. A leaf location is called *supported* if it has a feasible skeleton node within a certain distance d_0, where d_0 is a predefined constant (e.g. 0.1m). Otherwise, it is called *unsupported*. The *density* of each leaf location is defined as the total number of other leaf locations within a constant distance range (e.g. 0.5m).

We perform branch synthesis iteratively. In each iteration, we first scan through all of the unsupported leaf locations and associate each of them with its *nearest* feasible skeleton node. For those skeleton nodes with associated unsupported leaf locations, we synthesize branches to support as many leaf locations as possible. Our synthesis algorithm works as follows. As illustrated in Figure 4(a), for a given skeleton node O, we select from the associated leaf locations two separate locations A and B with high density. The skeleton node and these two locations form a triangle $\triangle OAB$. We then search the main skeleton for the most similar triangle $\triangle O'A'B'$ formed by a skeleton node and two of its offspring as shown in Figure 4(b) (see Section 7.2 for implementation details). If more than one candidate triangle is found, we choose the one with more fine details (i.e., more skeleton nodes within the section defined by O', A' and B'). The skeleton structure from O' to A' and B' is copied, scaled, transformed, and attached to the skeleton node O, as shown in Figure 4(c). In this way, besides leaf locations close to A and B, many other leaf locations are likely to be supported by the synthesized substructure. Since the synthesized branches are transformed from the existing tree structure, the self-similarity property of the tree is ensured. The iteration process stops when there are no more unsupported leaf locations or a maximal number of iterations is reached.

4.4 Skeleton Refinement

In the skeleton production algorithm introduced in Section 4.1, recall that the skeleton is formed by connecting adjacent skeleton nodes. This does not guarantee a smooth transition between connections. Since sharp transitions do not resemble the real appearance of a tree in most cases, they should be smoothed as also discussed in [Bloomenthal 1985]. Figure 7(a) shows an example of a sharp transition, where the angle between $\overline{AO}$ and $\overline{OB}$ is large. To improve the smoothness, the skeleton nodes A and B are interpolated by a Hermite curve, where the directions defined by $\overline{AO}$ and $\overline{OB}$ are used as the tangent directions at A and B respectively. The Hermite curve is sampled depending on the angle of a sharp transition. In general, a larger angle requires more sample points on a Hermite curve to smooth. Figure 7(b) shows the result after the smoothing operations, where the section AOB is replaced by a Hermite curve sampled using eight points (red) while the smoother section AOC is replaced by a Hermite curve sampled using only six points (blue).

Our branch synthesis is driven only by the elimination of unsupported leaf locations, so unnecessary twigs can be produced by the synthesis process. Although they do not produce any major appearance problems thanks to obscuring leaves, a pruning process can reduce the geometric complexity and produce a consistent appearance for the crown. If some twigs in a synthesized branch do not support any leaf locations, they are considered as unnecessary. These unnecessary twigs can be removed. However, to allow for the existence of some twigs without leaves, as appear in the real world, a small percentage of the unnecessary twigs are randomly chosen and kept. Also, some bare twigs are populated with leaves as discussed in Section 6 in order to account for lack of scanning in the interior of the crown.

5. SKELETON TO MESH

Once a full skeleton for a tree has been produced, the thickness of each branch section must be determined in order to produce a mesh fitted around the skeleton. Ideally, the thickness of each branch would be determined from the range

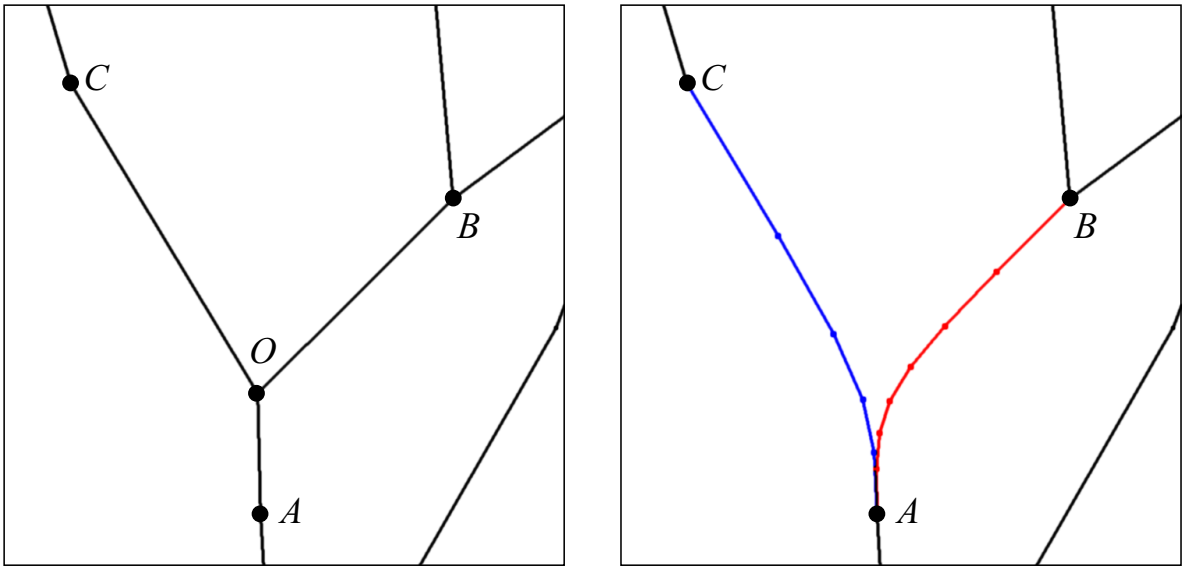


Fig. 7. Smoothing sharp transitions on a skeleton using Hermite curves.

scan data. However, several factors work against using the range scan information directly. First, those branches that are synthesized rather than scanned will not have any scanned points associated with them. Second, many branches have missing portions or are under-sampled in the scan because of occlusion and limited scanner resolution. Only the base of the trunk and possibly the larger sections of the lower branches will be scanned in sufficient detail to determine thickness. Therefore, some other technique must be used to produce plausible dimensions for the rest of the branches.

5.1 Tree Allometry

Many attempts have been made to formalize the size and structure of trees. As early as the 15th century, Leonardo da Vinci theorized that "all the branches of a tree at every stage of its height when put together are equal in thickness to the trunk" [Richter 1970].

This theory was justified and elaborated by the Pipe Model [Shinozaki et al. 1964a; 1964b]. In this model, trees are assumed to be composed of bundles of unit pipes, where each unit pipe supplies nutrients to a certain area of leaf surface. Thus, each time the tree branches, the sum of the cross-sectional areas of all child branches is equal to the cross-sectional area of the parent branch. The cross-sectional area of the tree at any given height remains constant because branches at lower levels exist to supply nutrients to leaves located at higher levels.

Recently the Pipe Model has been further refined in models such as the WBE model [West et al. 1999]. Here, branch thickness is dependent not only on fluid flow considerations, but also on mechanical considerations. The more mass a branch is supporting, the thicker it must be.

In work done by Murray[1927], the total cross sectional area becomes greater with each split such that

$$r_p{}^{2.49} = r_{c_1}{}^{2.49} + r_{c_2}{}^{2.49}$$

where r_p is the radius of the parent branch, and r_{c_1} and r_{c_2} are the radii of the two children in a binary split. The radius of a branch is taken to be proportional to the weight supported by that branch. Furthermore, Murray proposed [Murray 1926] that the angle formed between the directions of the parent branch and each child branch would follow the form in Equation 3.

$$cos\theta = \frac{r_p^4 + r_c^4 - (r_p^3 - r_c^3)^{\frac{4}{3}}}{2r_p^2 r_c^2} \tag{3}$$

We use a combination of Murray's law for branch splitting nodes, and a simplified WBE model for non-splitting nodes.

5.2 Radius Estimation and Branching Angle Adjustment

By using the allometric models outlined above, we can extrapolate believable dimensions for all branches in the skeleton regardless of their appearance in the scan. Using the scan data to estimate the radius of the base of the tree, several rules are used to obtain plausible estimates for the rest of the tree skeleton without further reliance on estimated measurements from the scan.

This operation works in two passes. First, the branch length supported by each node is calculated using a tip-to-root pass. Then, in a root-to-tip pass, two cases are considered for each parent node to estimate a radius for each child node. If the parent node has only one child node, the ratio between the child and parent radii is taken to be proportional to the ratio of the supported lengths for the child and parent nodes raised to the $\frac{3}{2}$ power as in the WBE model. Let r_p be the radius of the parent, r_c the radius of the child, l_p the total branch length supported by the parent, and l_c the total branch length supported by the child. The radius of the child node is defined by Equation 4.

$$r_c = r_p(\frac{l_c}{l_p})^{\frac{3}{2}} \tag{4}$$

If the parent node has more than one child, each child is assigned a radius based on an extension of Murray's rule. The child branch radii are scaled based on a crude estimate the supported weight for each child such that children supporting more weight are thicker. Let r_{c_i} be the radius of the ith child and l_{c_i} the total branch length supported by the ith child. The radius of each child node is defined by Equation 5.

$$r_{c_i} = r_p(\frac{l_{c_i}}{\sum_j l_{c_j}})^{\frac{1}{2.49}} \tag{5}$$

During the skeleton construction process, branching angles are largely based on shortest path considerations (reference Figures 2 and 3). At this point we examine all branching angles within the skeleton to determine how closely they match the angles predicted by Equation 3 given the radii just calculated. In the event that an angle differs from the predicted angle, we shift the intersection point at which a child branch splits from a parent branch such that the angle formed is closer to the predicted angle.

It should be noted, however, that biological models are not taken as the final word in determining branching angles. While they may describe ideal behavior, many factors can cause a tree to deviate from ideal growth paths, and old branches that affected the growth of other parts of the tree may have fallen off prior to scanning, obscuring them from our system. In particular, trees in urban environments are often pruned and shaped artificially and may not conform to the shapes that the above models dictate. We provide the ability for a user to specify that they wish to disregard biological rules in favor of trusting the detected structure in the event that a scan is considered to be trustworthy. In addition, when deciding whether or not to adjust a branch angle, we give top priority to any angle that the user has explicitly dictated (via an editing interface).

5.3 Mesh Production

Once each node has an estimated radius, producing a mesh is a relatively simple task. A ring is placed around each node perpendicular to the direction to the node's parent. Connecting these rings with triangles produces a mesh.

Splines may be used to create a smooth shape. At the same time, appropriate texture coordinates are assigned to each vertex. At rendering time a cyclic bark texture can be applied to the trunk and branches.

6. LEAVES

As discussed in Section 4.3, we identify the points representing leaves and find positions to place leaves. Since our branch synthesis process is driven by the elimination of unsupported leaf locations, after several iterations of synthesis most leaf locations are supported. We model leaves only at supported leaf locations. In this section the term *leaf locations* refers to supported leaf locations.

Leaves are modeled using quads with alpha textures. A quad is assigned at each leaf location, where a normal direction needs to be calculated to orient the quad. According to our definition of a supported leaf location, a feasible skeleton node exists within a certain distance d_0 (see Section 4.3). Therefore, for each leaf location L, we connect the quad to its *nearest* feasible skeleton node S. Since the skeleton node is feasible for L, the connection angle is guaranteed to be consistent with angles in the existing skeleton. Assume that the direction from S's parent to S is $\vec{d}$. The normal of the quad is then defined by $(\overrightarrow{SL} \times \vec{d}) \times \overrightarrow{SL}$. To ensure a natural appearance, we also perturb the normal slightly to introduce some randomness.

Placing leaves at detected leaf locations works for areas of the crown that are sufficiently scanned, but does not produce enough rendered leaves in areas that the scanner could not reach in detail, such as the interior of the crown. To compensate for this, we select leafless locations along branches that are within a certain distance of the branch tip to place additional leaves. This process takes place after the pruning process.

Recall that a density is defined at every leaf location (see Section 4.3). We assume that the leaf density in a real tree is more or less uniform. In our experience, a leaf location with low density is normally under-sampled or occluded by other branches and leaves. To compensate for this, a leaf location with lower density is assigned a texture with a larger number of leaves. In contrast, we assign a texture with a smaller number of leaves to leaf locations with higher densities. To achieve this, we sort all leaf locations from high to low density and divide them evenly into several bins. As shown in Figure 8, leaf locations in the first bin are assigned a texture of a single leaf, those in the second bin are assigned a texture of two leaves, and so on. Extra leaves that were placed on previously bare branches as discussed above are assigned a random sized leaf cluster. Quad size is proportionally adjusted based on the number of leaves.

Fig. 8. Leaves are modeled using quads with various alpha textures. (The alpha value is 0 in the white area.)

7. IMPLEMENTATION DETAILS

7.1 Data Structures

Many operations such as forming a graph by connecting adjacent points require searching for all neighboring points of a given point. To accelerate such searches, a point cloud is organized in a Kd-Tree in our system. In addition to the traditional range-based searching operations supported by a Kd-Tree, we have also enhanced the searching to support the intersection of a cone ray and the point cloud. With this enhancement, the bi-directional probing process discussed in Section 4.2 is made very efficient.

Parameter (Section of first appearance)	Derivation	20m Elm
Local neighborhood (4.1)	Scanner resolution, precision, distance	$0.2m$
Bi-directional probing cone angle (4.2)	Slightly greater than average axil angle	45^o
Max gap when connecting subgraphs (4.2)	Just larger than largest occluder (trunk)	$2.0m$
Max distance from "supported" leaf to branch (4.3)	Average twig length	$0.1m$
Leaf density range (4.3)	Average leaf cluster size (subjective)	$0.5m$
Max skeleton synthesis iterations (4.3)	Subjective	3
Percentage of bare twigs kept (4.4)	Subjective	25%

Table I. Selected Parameters, including suggested values to use for the 20m Elm tree used as our primary example.

The graph formed from the point cloud is represented by a sparse matrix, where a nonzero element indicates an edge existing between two points and its value is the length of the edge. The nonzero entries are stored in a compact row scheme, which is dynamically maintained. With this storage scheme, the operation of connecting two subgraphs is made easy by inserting two nonzero entries in the sparse matrix. Furthermore, with the matrix representation of the graph, the Dijkstra's shortest path algorithm used for finding the skeleton of a subgraph can be efficiently implemented.

It is natural to store the skeleton and the skeleton nodes in a tree structure. However, when determining if a leaf location is supported, it requires searching for nearest skeleton nodes, where the tree structure is inefficient. For this purpose, a Kd-Tree is also built for the skeleton nodes in addition to the tree structure.

7.2 Implementation of Skeleton Synthesis

When synthesizing a branch at a given skeleton node O, recall that we need to select from two locations A and B from its associated leaf locations (see Section 4.3). We intend to synthesize a branch not only being able to support as many leaf locations as possible, but also long enough to support leaf locations far away from the node O. Therefore, in our implementation, we always select one location A with the highest density among the associated leaf locations. For the other location, we consider both the density and the distance to the node O. Specifically, we first find a group of associated leaf locations whose distances to the node O are large relative to the other potential locations. Then, the other location is chosen as the one with the highest density among the group. If this location is identical or very close to A, another candidate is evaluated until an appropriate location is chosen as B.

Once the leaf locations A and B are determined, a triangle $\triangle OAB$ is formed. In the next step of branch synthesis, we need to search the main skeleton for a skeleton node O' and two of its offspring A' and B', such that the triangle $\triangle O'A'B'$ formed by these three skeleton nodes is most similar to the triangle $\triangle OAB$ (see Figure 4). The similarity is defined by the sum of the differences between the corresponding angles in these two triangles. In other words, O', A', and B' are chosen so that the following equation is satisfied,

$$\min_{O',A',B'} \sqrt{|\angle O - \angle O'|^2 + |\angle A - \angle A'|^2 + |\angle B - \angle B'|^2}. \tag{6}$$

For efficiency, all the possible triangles on the skeleton are pre-calculated before the synthesis process. One strategy we used for efficiently determining the triangle satisfying Equation 6 is as follows. For each triangle, assume that its angles at O', A', and B' are α, β, and γ respectively. We store these angles in the three components of a 3D point (α,β,γ). Considering symmetry, we also store them in another 3D point (α,γ,β). In this way, all the angles of all possible triangles on the skeleton are represented by 3D points. Thus, the problem of Equation 6 becomes finding a nearest neighbor in a 3D space for a given point whose coordinates are $(\angle O, \angle A, \angle B)$. Therefore, as also discussed in Section 7.1, this can be trivially solved by building a Kd-Tree for all the points storing the angles.

T : execution time in seconds;
n_S : the number of nodes in the produced skeleton;
n_T : the number of triangles in the generated mesh;

	Skeleton Production	Branch Synthesis		
		Iteration 1	Iteration 2	Iteration 3
T	5.3	1.5	8.2	22.3
n_S	615	4,258	8,972	14,653
n_T	23,790	91,746	271,590	573,396
Leaf Generation				Total Time (s)
T	24.5	n_T	64,632	61.8

Table II. Statistics for the tree models shown in Figure 10.

8. RESULTS

Our knowledge and heuristic based tree modeling pipeline has been implemented on a PC with an Pentium 2.4GHz processor and 640MB of main memory. The constructed tree models are rendered using DirectX 9.0 on an nVidia GeForce FX 5200 graphics card with 128MB video memory (see below for exceptions). The point clouds are scanned by Riegl Inc's LMS-Z360i 3D imaging sensor. Bark and leaf textures shown here are synthesized from photographs of the scanned trees.

The focus of our technique is tree modeling. Since our modeling method produces a polygonal mesh, rendering can be handled by many different applications. The figures shown here are rendered by our in-house modeling tools with the exception of Figures 1d, 10e and 10f, which were rendered using POV-Ray [POV-Team 2006].

Figure 10 demonstrates the most difficult case of a tree model constructed from a single point cloud (114,997 points) segmented from a single scan as shown in Figure 1(a). It is clear that the data from only one scan is very incomplete. Even still, our system is capable of generating a full polygonal model that plausibly recreates the tree. Figure 10(b)(c)(d) show models generated by one, two, and three iterations of branch synthesis. The effect of pruning can be seen between iterations 2 and 3. The leaf textures used in (e) and (f) are shown in Figure 8. Although we have not defined a quantitative way of evaluating the fidelity of the constructed trees, they visually resemble the main structure of the real trees, as seen in Figure 11 (c) and (d). The amount of data in the models, time for producing the main skeleton, time for synthesizing the branches, and time for generating the leaves are summarized in Table II. As shown in the table, it takes around a minute of computation time to generate the tree model. In general, even considering the interaction time of selecting reasonable modeling parameters by quick trial and error, it takes a novice user less than ten minutes to create a plausible tree model directly from the point cloud produced by a laser scanner. Rendering of the models can be done in real time to provide timely feedback to the user.

Sometimes, a point cloud from only a single scan does not include enough details to recreate a plausible tree model. In these cases, using points from multiple registered scans of the same tree often helps to create a more accurate tree model. As an example, Figure 11(a) and (b) show tree models constructed from a single scan and from three registered scans respectively. In this case, the tree constructed from multiple scans clearly has a more faithful appearance to a photograph of the tree shown in Figure 11(c), thanks to the better sampling rate and coverage of the multiple scans.

Figure 12(a) and (b) show some other examples of models constructed from laser scans of trees. To match the appearance of the real trees, bark and leaf textures generated or synthesized from photographs are desirable. However, various appearances may be achieved by applying different textures or specifying different leaf sparsity (choosing a subset of the leaf locations), as demonstrated in Figure 12(a). Figure 12(b) is a 6 meter tall mature Crab Apple Tree, shown here defoliated with dense winter fruit. This tree demonstrates a much squatter shape, with much more horizontal growth and spreading. This structure (different from the primarily upwards growth seen in the Elm examples) can be found in many smaller fruit trees.

Figure 13 demonstrates tree models constructed by our knowledge-based method in place of their original point

Fig. 9. Tree models are inserted in the Ephesos excavation site scanned in Turkey.

clouds in a scanned environment. Our tree models can also be inserted in other 3D scenes, as shown in Figure 9, where shadow volumes are used for rendering the tree's shadow.

9. LIMITATIONS AND FUTURE WORK

Although many of the parameters can be computed or specified using the guidelines given in Table I, one current limitation of this process remains the difficulty in obtaining detailed information regarding a wide variety of tree species. Many dendrology manuals and other sources only contain data regarding rough dimensions and leaf patterns for most species. Information regarding crown architecture, axil angle ranges and other structural considerations is not widely available for many species. As our algorithm is dependent on this knowledge to accurately recreate trees, we hope to see this type of information catalogued and distributed for more tree species in the future. As the situation stands now, we are forced to conduct our own limited surveys for those species that have not been studied extensively by the forestry and biology community.

Second, there is considerable room for expansion of our general tree model. Although we are already able to obtain results that are quite close to the original scanned tree, certain assumptions (such as the assumption of a cylindrical trunk) create visual differences when photographs of a tree are compared to our models (Figure 11(b-c). Also, our discussions with forestry experts have revealed room for improvement in how we handle new growth vs old growth in certain species of trees. In particular, the Bur Oaks found in our region develop much different branching structures late in life compared to younger trees of the same species. A defoliated maturing Oak tree can be seen in Figure 12(c). Note that the structure of the lower part of the tree is dominated by straight branches joined to an almost entirely vertical trunk at close to right angles, which become much shallower in the upper part of the tree. This produces a challenging situation if a user tries to synthesize the upper small branches based on the larger structures below. Other species, such as Weeping Willows, have large, drooping leaves which are likely to obscure any actual branching structure. Again, as this type of data is cataloged and distributed, we will have more opportunities to refine our techniques.

For future work, we would like to investigate strategies to overcome the existing limitations outlined above. We would like to increase our use of biological and physical constraints in guiding and validating our generated models. In particular, since our scans contain information regarding the environment surrounding the tree, we feel there is an opportunity to incorporate this information in our model, as seen in [Hart and Baker 2003]. We also plan to incorporate advanced rendering techniques including efficient shadowing [Max and Ohsaki 1995], displacement mapping of the tree bark [Wang et al. 2003], and enhanced rendering of leaves [Wang et al. 2005]. These rendering improvements will have to be tempered against speed requirements, as our ultimate goal is not to render a single tree, but to render many trees within a larger natural environment at interactive speeds.

10. CONCLUSION

We have presented a method for taking point clouds produced by range scans of trees and using knowledge of tree structures to produce a polygonal mesh with plausible details not present in the scan. We produce a tree skeleton from the scan, synthesize branches to ensure that all parts of the tree's crown are supported by the base of the tree, use allometric theory to produce a mesh of appropriate dimensions, and place textured leaves at detected leaf locations. Our process requires only minimal user interaction, and the full process including scanning and modeling can be completed within minutes.

REFERENCES

BLOOMENTHAL, J. 1985. Modeling the mighty maple. In *Proc. of ACM SIGGRAPH 85*. ACM Press, 305–311.

BROSTOW, G. J., ESSA, I., STEEDLY, D., AND KWATRA, V. 2004. Novel skeletal representation for articulated creatures. In *Proc. of ECCV04*. Vol III: 66–78.

ERVIN, S. M. AND HASBROUCK, H. H. 2001. *LANDSCAPE MODELING: Digital Techniques for Landscape Visualization*. McGraw-Hill Professional Publishing.

GORTE, B. AND PFEIFER, N. 2004. Structuring laser-scanned trees using 3D mathematical morphology. In *Proc. of XXth ISPRS Congress*. 929–933.

HART, J. C. AND BAKER, B. 2003. Structural simulation of tree growth and response. *The Visual Computer 19*, 2–3 (May), 151–163.

LINDENMAYER, A. 1968. Mathematical models for cellular interaction in development. *J. of Theoretical Biology 18*, 280–315.

MAX, N. L. AND OHSAKI, K. 1995. Rendering trees from precomputed z-buffer views. In *Proc. of Eurographics Workshop on Rendering 95*. 74–81.

MURRAY, C. D. 1926. The physiological principle of minimum work applied to the angle of branching of arteries. *Journal of General Physiology 9*, 835–841.

MURRAY, C. D. 1927. A relationship between circumference and weight in trees and its bearing on branching angles. *Journal of General Physiology 10*, 725–729.

PFEIFER, N., GORTE, B., AND WINTERHALDER, D. 2004. Automatic reconstruction of single trees from terrestrial laser scanner data. In *Proc. of XXth ISPRS Congress*. 114–119.

POV-TEAM. 2006. Pov-ray: The persistence of vision raytracer. `http://www.povray.org/`.

PRUSINKIEWICZ, P. AND LINDENMAYER, A. 1990. *The algorithmic beauty of plants*. Springer-Verlag New York, Inc.

PRUSINKIEWICZ, P., MÜNDERMANN, L., KARWOWSKI, R., AND LANE, B. 2001. The use of positional information in the modeling of plants. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM Press, New York, NY, USA, 289–300.

PYYSALO, U. AND HYYPP, H. 2002. Reconstructing tree crowns from laser scanner data for feature extraction. In *Proc. of ISPRS Commission III*. B–218–221.

RECHE, A., MARTIN, I., AND DRETTAKIS, G. 2004. Volumetric reconstruction and interactive rendering of trees from photographs. In *Proc. of ACM SIGGRAPH 04*. 720–727.

RICHTER, J. P. 1970. *The Notebooks of Leonardo da Vinci*. General Publishing Company, Ltd.

SHINOZAKI, K., YODA, K., HOZUMI, K., AND KIRA, T. 1964a. A quantitative analysis of plant form - the pipe model theory I. Basic analyses. *Japanese J. of Ecology 14*, 3, 97–105.

SHINOZAKI, K., YODA, K., HOZUMI, K., AND KIRA, T. 1964b. A quantitative analysis of plant form - the pipe model theory II. Further evidence of the theory and its application in forest ecology. *Japanese J. of Ecology 14*, 4, 133–139.

SHLYAKHTER, I., ROZENOER, M., DORSEY, J., AND TELLER, S. 2001. Reconstructing 3D tree models from instrumented photographs. *IEEE Compututer Graphics & Appl. 21,* 3, 53–61.

WANG, L., WANG, W., DORSEY, J., YANG, X., GUO, B., AND SHUM, H.-Y. 2005. Real-time rendering of plant leaves. *ACM Trans. Graph. 24,* 3, 712–719.

WANG, L., WANG, X., TONG, X., LIN, S., HU, S., GUO, B., AND SHUM, H.-Y. 2003. View-dependent displacement mapping. *ACM Trans. Graph. 22,* 3, 334–339.

WEISSTEIN, E. W. 2005. Standard deviation. From MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/StandardDeviation.html`.

WEST, G. B., BROWN, J. H., AND ENQUIST, B. J. 1999. A general model for the structure and allometry of plant vascular systems. *Nature 400,* 664–667.

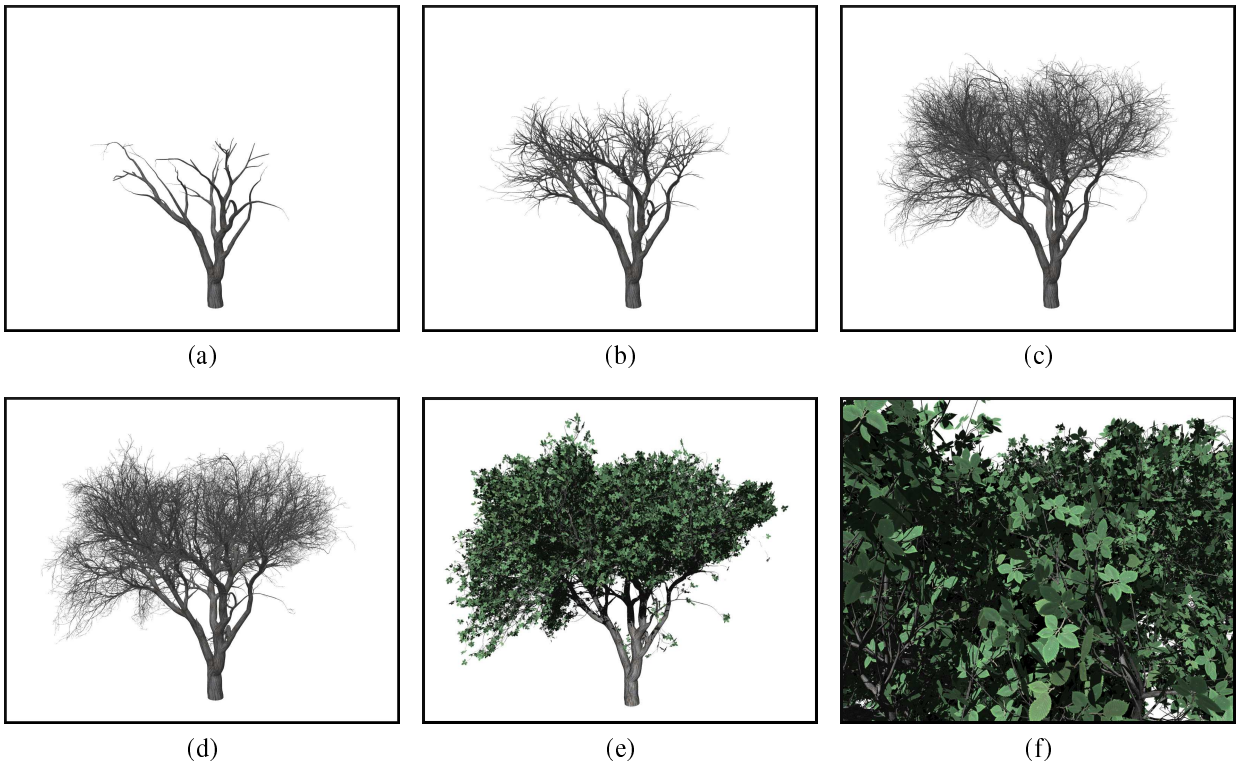
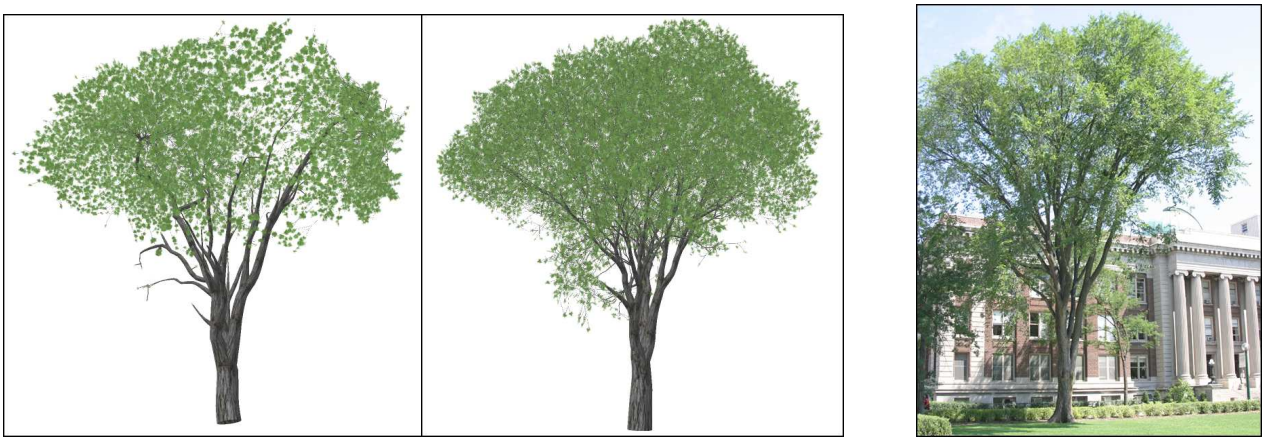
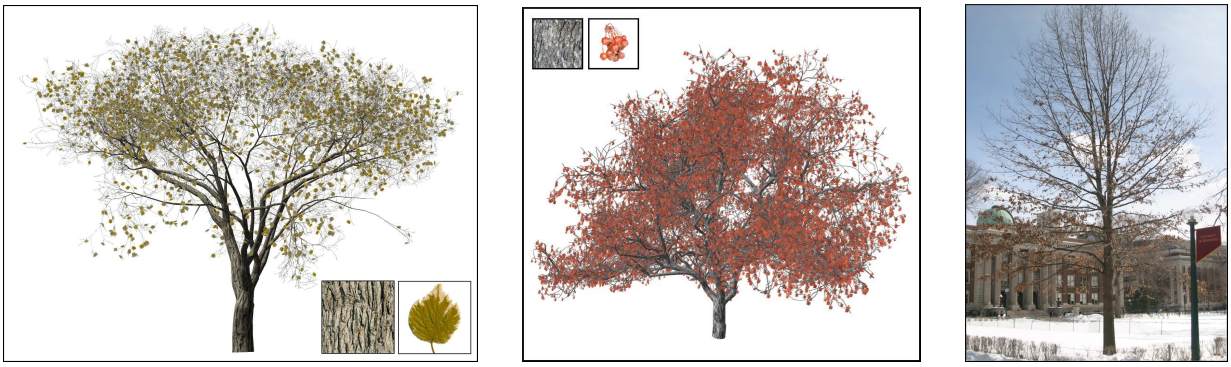


Fig. 10. Renderings of (a) a mesh produced from the main skeleton in Figure 1(c); (b)(c)(d) meshes produced from skeletons synthesized in one, two, and three iterations with pruning respectively; (e) the mesh in (d) with leaves attached; (f) a close view of (e).



(a) (b) (c)

Fig. 11. (a)(b)(c) Comparison of other tree models generated from (a) only one scan (67,157 points), (b) three scans (151,956 points), and (c) a photograph of the tree modeled in (a) and (b) for comparison.



(a) (b) (c)

Fig. 12. (a) The tree model in Figure 10 rendered using alternate bark and leaf textures and reduced leaf density. (b) A winter model of a scanned Crab Apple tree. (c) A photo of a maturing Oak tree.

Fig. 13. Below, constructed tree models are used in place of their original point clouds (shown above) in a scanned environment.