

Image-based Tree Modeling

Ping Tan Gang Zeng* Jingdong Wang Sing Bing Kang¹ Long Quan

The Hong Kong University of Science and Technology ¹Microsoft Research, Redmond, WA, USA



Figure 1: Image-based modeling of a tree. From left to right: A source image (out of 18 images), reconstructed branch structure rendered at the same viewpoint, tree model rendered at the same viewpoint, and tree model rendered at a different viewpoint.

Abstract

In this paper, we propose an approach for generating 3D models of natural-looking trees from images that has the additional benefit of requiring little user intervention. While our approach is primarily image-based, we do not model each leaf directly from images due to the large leaf count, small image footprint, and widespread occlusions. Instead, we populate the tree with leaf replicas from segmented source images to reconstruct the overall tree shape. In addition, we use the shape patterns of visible branches to predict those of obscured branches. We demonstrate our approach on a variety of trees.

1 Introduction

Trees are hard to model in a realistic way because of their inherent geometric complexity. While progress has been made over the years in modeling trees, ease of model generation, model editability, and realism are difficult to achieve simultaneously. A tool with all these features could easily be part of a cost-effective solution to building realistic-looking environments for movie post-production, architectural designs, games, and web applications.

Our system uses images to model the tree. We chose an image-based approach because we believe such an approach has the best potential for producing a realistic tree model. The capture process is simple, as it involves only a hand-held camera. We use a structure from motion technique to recover the camera motion and 3D point

cloud of the tree. Rather than applying specific rules for branch generation, we use the local shapes of branches that are observed to interpolate those of obscured branches. Finally, the leaves are generated by segmenting the source images and computing their depths using the pre-computed 3D points or the recovered branches. One such result can be seen in Figure 1.

Note that in this paper, we differentiate between bushes and trees—we consider “bushes” as terrestrial flora with large discernible leaves (relative to the bush size), and “trees” as large terrestrial flora with small leaves (relative to the tree size). The spectrum of bushes and trees with varying leaf sizes is shown in Figure 2.

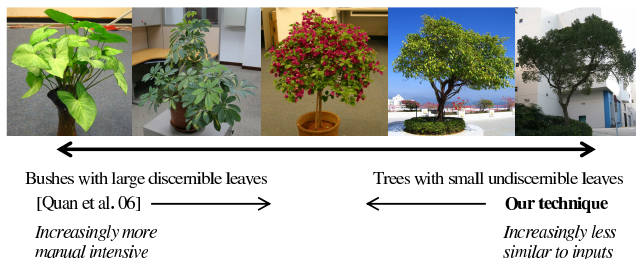


Figure 2: Spectrum of bushes and trees based on relative leaf size: On the left end of the spectrum, the size of the leaves relative to the bush is large. This is ideal for using the modeling system of Quan *et al.* [2006]. Our system, on the other hand, targets trees with small relative leaf sizes (compared to the entire tree).

2 Related work

The techniques for modeling bushes and trees can be roughly classified as being rule-based or image-based. Rule-based techniques make use of a small set of generative rules or grammar to create branches and leaves. Prusinkiewicz *et al.* [1994], for example, developed a series of approaches based on the idea of the generative L-system, while Weber and Penn [1995] used a series of geometric rules to produce realistic-looking trees. De Reffye *et al.* [1988] also used a collection of rules of plant growth. Such techniques provide some realism and editability, but they require expertise for effective use. They work on the (usually) reasonable assumption that the

*Gang Zeng is currently at ETH Zürich.

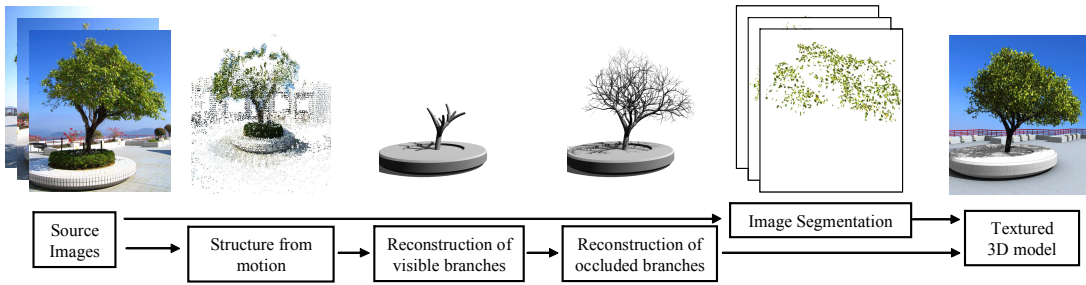


Figure 3: Overview of our tree modeling system.

branch shape and leaf arrangement follow a predictable pattern. On the other hand, they require considerable effort to replicate unexpected local structural modifications such as stunted growth due to disease or responses to external agents.

Image-based approaches use images of a tree for modeling. They range from the use of a single image and (limited) shape priors [Han and Zhu 2003] to multiple images [Sakaguchi 1998; Shlyakhter et al. 2001; Reche-Martinez et al. 2004; Quan et al. 2006]. A popular approach is to use the visual hull to aid the modeling process [Sakaguchi 1998; Shlyakhter et al. 2001; Reche-Martinez et al. 2004]. While Shlyakhter et al. [2001] refines the medial axis of the volume to a simple L-system fit for branch generation, Sakaguchi et al. [1998] use simple branching rules in voxel space for the same purpose. However, the models generated by these approaches are only approximate and have limited realism. Reche-Martinez et al. [2004], on the other hand, compute a volumetric representation with variable opacity. While realism is achieved, their models cannot be edited or animated easily.

The approaches closest to ours are those of Quan *et al.* [2006] and Xu *et al.* [2005]. Quan *et al.* [2006] showed that it is possible to model bushes well by explicitly modeling leaves from images and providing a simple user interface to generate branches. However, generating a tree with a large number of small overlapping leaves would be impractical. In our approach, the branches are generated automatically from the images, with pattern-based interpolation in occluded areas. This substantially reduces the amount of manual input required. In addition, Quan *et al.* usually require a 360° capture, which may not always be possible for outdoor trees. We require only a partial coverage of the tree and a small number of overlapping images (10-20 images for most examples in this paper). Figure 2 illustrates the types of bushes and trees that are appropriate for Quan *et al.*’s [2006] technique and ours. Both these techniques are complementary.

Xu *et al.* [2005] used a laser scanner to acquire the range data for modeling trees. Part of our work—the generation of initial visible branches—is inspired by their work. The major difference is that they use a 3D point cloud for modeling; no registered source images are used. It is not easy to generate complete tree models from just 3D points because of the difficulties in determining what is missing and in filling the missing information. Our experience has led us to believe that adapting models to images is a more intuitive means for realistic modeling. The image-based approach is also more flexible for modeling a wide variety of trees at different scales.

3 Overview of the system

Our tree modeling system consists of three main parts: image capture and 3D point recovery, branch recovery, and leaf population illustrated in Figure 3. It is designed to reduce the amount of user interaction required by using as much data from images as possible. The recovery of the visible branches is mostly automatic, with the user given the option of refining their shapes. The subsequent recovery of the occluded branches and leaves is automatic with only a few parameters to be set by the user.

As was done by researchers in the past, we capitalize on the structural regularity of trees, more specifically the self-similarity of structural patterns of branches and arrangement of leaves. However, rather than extracting rule parameters (which is very difficult to do in general), we use the extracted local arrangement of visible branches as building blocks to generate the occluded ones. This is done using the recovered 3D points as hard constraints and the matte silhouettes of trees in the source images as soft constraints.

To populate the tree with leaves, the user first provides the expected average image footprint of leaves. The system then segments each source image based on color. The 3D position of each leaf segment is determined either by its closest 3D point or by its closest branch segment. The orientation of each leaf is approximated from the shape of the region relative to the leaf model or the best-fit plane of leaf points in its vicinity.

4 Image capture and 3D point recovery

We use a hand-held camera to capture the appearance of the tree of interest from a number of different overlapping views. In all but one of our experiments, only 10 to 20 images were taken for each tree, with coverage between 120° and 200° around the tree. The exception is the potted flower tree shown in Figure 7, where 32 images covering 360° were taken.

Prior to any user-assisted geometry reconstruction, we extract point correspondences and ran structure from motion on them to recover camera parameters and a 3D point cloud. We also assume the matte for the tree has been extracted in each image, so that we know the extracted 3D point cloud is that of the tree and not the background. In our implementation, matting is achieved with automatic color-based segmentation and some user guidance.

Standard computer vision techniques have been developed to estimate the point correspondences across the images and the camera parameters. We used the approach described in [Lhuillier and Quan 2005] to compute the camera poses and a semi-dense cloud of reliable 3D points in space. Depending on the spatial distribution of the camera and the geometric complexity of the tree, there may be significant areas that are missing or sparse due to occlusion. One example of structure from motion is shown in Figure 3.

5 Branch recovery

Once the camera poses and 3D point cloud have been extracted, we next reconstruct the tree branches, starting with the visible ones. The local structures of the visible branches are subsequently used to reconstruct those that are occluded in a manner similar to non-parametric texture synthesis in 2D [Efros and Leung 1999] (and later 3D [Breckon and Fisher 2005]), using the 3D points as constraints.

To enable the branch recovery stage to be robust, we make three assumptions. First, we assume that the cloud of 3D points has been partitioned into points belonging to the branches and leaves (using color and position). Second, the tree trunk and its branches are assumed to be unoccluded. Finally, we expect the structures of the

visible branches to be highly representative of those that are occluded (modulo some scaled rigid transform).

5.1 Reconstruction of visible branches

The cloud of 3D points associated with the branches is used to guide the reconstruction of visible branches. Note that these 3D points can be in the form of multiple point clusters due to occlusion of branches. We call each cluster a *branch cluster*; each branch cluster has a primary branch with the rest being secondary branches.

The visible branches are reconstructed using a data-driven, bottom-up approach with a reasonable amount of user interaction. The reconstruction starts with graph construction, with each sub-graph representing a branch cluster. The user clicks on a 3D point of the primary branch to initiate the process. Once the reconstruction is done, the user iteratively selects another branch cluster to be reconstructed in very much the same way, until all the visible branches are accounted for. The very first branch cluster handled consists of the tree trunk (primary branch) and its branches (secondary branches).

There are two parts to the process of reconstructing visible branches: graph construction to find the branch clusters, followed by sub-graph refinement to extract structure from each branch cluster.

Graph construction. Given the 3D points and 3D-2D projection information, we build a graph G by taking each 3D point as a node and connecting to its neighboring points with edges. The neighboring points are all those points whose distance to a given point is smaller than a threshold set by the user. The weight associated with each edge between a pair of points is a combined distance $d(p, q) = (1 - \alpha)d_{3D} + \alpha d_{2D}$ with $\alpha = 0.5$ by default. The 3D distance d_{3D} is the 3D Euclidean distance between p and q normalized by its variance. For each image I_i that p and q project to, let l_i be the resulting line segment in the image joining their projections $\mathcal{P}_i(p)$ and $\mathcal{P}_i(q)$. Also, let n_i be the number of pixels in l_i and $\{x_{ij} | j = 1, \dots, n_i\}$ be the set of 2D points in l_i . We define a 2D distance function $d_{2D} = \sum_i \frac{1}{n_i} \sum_j |\nabla I_i(x_{ij})|$, normalized by its variance, with $\nabla I(x)$ being the gradient in image I at 2D location x . The 2D distance function accounts for the normalized intensity variation along the projected line segments over all observed views. If the branch in the source images has been identified and pre-segmented (e.g., using some semi-automatic segmentation technique), this function is set to infinity if any line segment is projected outside the branch area. Each connected component, or sub-graph, is considered as a branch cluster. We now describe how each branch cluster is processed to produce geometry, which consists of the skeleton and its thickness distribution.

Conversion of sub-graph into branches. We start with the branch cluster that contains the lowest 3D point (the “root” point), which we assume to be part of the primary branch. (For the first cluster, the primary branch is the tree trunk.) The shortest paths from the root point to all other points are computed by a standard shortest path algorithm. The edges of the sub-graph are kept if they are part of the shortest paths and discarded otherwise.

This step results in 3D points linked along the surface of branches. Next, to extract the skeleton, the lengths of the shortest paths are divided into segments of a pre-specified length. The centroid of the points in each segment is computed and is referred to as a skeleton node. The radius of this node (or the radius of the corresponding branch) is the standard deviation of the points in the same bin. This procedure is similar to those described in [Xu et al. 2005] and [Brostow et al. 2004].

User interface for branch refinement. Our system allows the user to refine the branches through simple operations that include adding or removing skeleton nodes, inserting a node between two adjacent nodes, and adjusting the radius of a node (which controls the local branch thickness). In addition, the user can also connect different

branch clusters by clicking on one skeleton node of one cluster and a root point of another cluster. The connection is used to guide the creation of occluded branches that link these two clusters (see Section 5.2). Another feature of our system is that all these operations can be specified at a view corresponding to any one of the source images; this allows user interaction to occur with the appropriate source image superimposed for reference.

A result of branch structure recovery is shown for the bare tree example in Figure 4. This tree was captured with 19 images covering about 120° . The model was automatically generated from only one branch cluster.



Figure 4: Bare tree example. From left to right: one of the source images, superimposed branch-only tree model, and branch-only tree model rendered at a different viewpoint.

5.2 Reconstruction of occluded branches

The recovery of visible branches serves two purposes: portions of the tree model is reconstructed, and *the reconstructed parts are used to replicate the occluded branches*. We make the important assumption that the tree branch structure is locally self-similar. In our current implementation, any subset, i.e., a subtree, of the recovered visible branches is a candidate replication block. This is illustrated in Figure 5 for the final branch results shown in Figure 1 and 7.

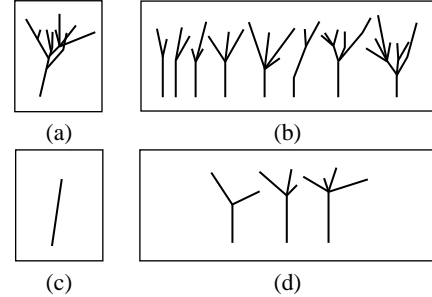


Figure 5: Branch reconstruction for two different trees. The left column shows the skeletons associated with visible branches while the right are representative replication blocks. (a,b) are for the fig tree in Figure 1, and (c,d) are for the potted flower tree in Figure 7. (Only the main branch of the flower tree is clearly visible. Three replication blocks were chosen from another tree and used for branch reconstruction.)

The next step is to recover the occluded branches given the visible branches and the library of replication blocks. We treat this problem as texture synthesis, with the visible branches providing the texture sample and boundary conditions. There is a major difference with conventional texture synthesis: the scaling of a replication block is spatially dependent. This is necessary to ensure that the generated branches are geometrically plausible with the visible branches.

In a typical tree with dense foliage, most of branches in the upper crown are occluded. To create plausible branches in this area, the system starts from the existing branches and “grows” to occupy part of the upper crown using our synthesis approach. The cut-off boundaries are specified by the tree silhouettes from the source images. The growth of the new branches can also be influenced by the reconstructed 3D points on the tree surface as branch endpoints. Depending on the availability of reconstructed 3D points, the “growth” of occluded branches can be unconstrained or constrained.

Unconstrained growth. In areas where 3D points are unavailable, the system randomly picks an endpoint or a node of a branch structure and attaches the endpoint or node to a random replication block. Although the branch selection is mostly random, priority is given to thicker branches or those closer to the tree trunk. In growing the branches, two parameters associated with the replication block are computed on the fly: a random rotation and a scale. The replication block is first rotated about its primary branch by the chosen random angle. Then, it is globally scaled right before it is attached to the tree such that the length of its scaled primary branch matches that of the end branch being replaced. Once scaled, the primary branch of the replication block replaces the end-branch. This growth is capped by the silhouettes of the source images to ensure that the reconstructed overall shape is as close as possible to that of the real tree.

Constrained growth. The reconstructed 3D points, by virtue of being visible, are considered to be very close to the branch endpoints. By branch endpoints, we mean the exposed endpoints of the last generation of the branches. These points are thus used to constrain the extents of the branch structure. By comparison, in the work of [Quan et al. 2006], the 3D points are primarily used to extract the shapes of leaves.

This constrained growth of branches (resulting in *Tree*) is computed by minimizing $\sum_i D(p_i, Tree)$ over all the 3D points $\{p_i | i = 1, \dots, n_{3D}\}$, with n_{3D} being the number of 3D points. $D(p, Tree)$ is the smallest distance between a given point p and the branch endpoints of *Tree*. Unfortunately, the space of all possible subtrees with a fixed number of generations to be added to a given tree is exponential. Instead, we solve this optimization in a greedy manner. For each node of the current tree, we define an *influence cone* with its axis along the current branch and an angular extent of 90° side to side. For that node, only the 3D points that fall within its influence cone are considered. This restricts the number of points and set of subtrees considered in the optimization.

Our problem reduces to minimizing $\sum_{p_i \in Cone} D(p_i, Subtree)$ for each subtree, with *Cone* being the set of points within the influence cone associated with *Subtree*. If *Cone* is empty, the branches for this node are created using the unconstrained growth procedure described earlier. The order in which subtrees are computed is in the same order of the size of *Cone*, and is done generation by generation. The number of generations of branches to be added at a time can be controlled. In our implementation, for speed considerations, we add one generation at a time and set a maximum number of 7 generations.

Once the skeleton and thickness distribution have been computed, the branch structure can be converted into a 3D mesh, as shown in Figures 1, 4, and 7. The user has the option to perform the same basic editing functions as described in Section 5.1.

6 Populating the tree with leaves

The previous section described how the extracted 3D point cloud is used to reconstruct the branches. Given the branches, one could always just add the leaves directly on the branches using simple guidelines such as making the leaves point away from branches. While this approach would have the advantage of not requiring the use of the source images, the result may deviate significantly from

the look of the real tree we wish to model. Instead, we chose to analyze the source images by segmenting and clustering and use the results of the analysis to guide the leaf population process.

6.1 Image segmentation and clustering

Since the leaves appear relatively repetitive, one could conceivably use texture analysis for image segmentation. Unfortunately, the spatially-varying amounts of foreshortening and mutual occlusion of leaves significantly complicate the use of texture analysis. However, we do not require very accurate leaf-by-leaf segmentation to produce models of realistic-looking trees.

We assume that the color for a leaf is homogeneous and there are intensity edges between adjacent leaves. We first apply the mean shift filter [Comaniciu and Meer 2002] to produce homogeneous regions, with each region tagged with a mean-shift color. These regions undergo a split or merge operation to produce new regions within a prescribed range of sizes. These new regions are then clustered based on the mean-shift colors. Each cluster is a set of new regions with similar color and size that are distributed in space, as can be seen in Figure 6(c,d). These three steps are detailed below.

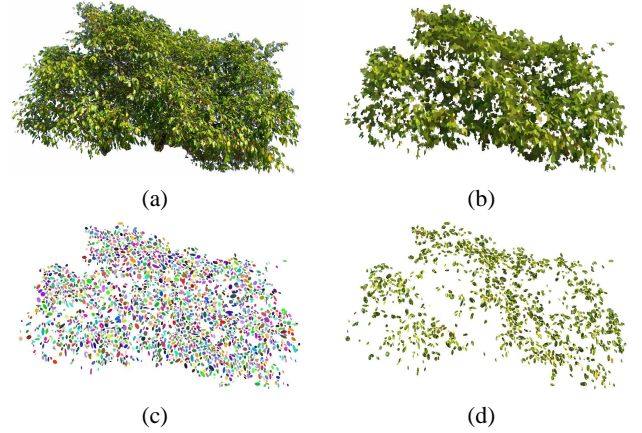


Figure 6: Segmentation and clustering. (a) Matted leaves from source image, (b) regions created after the mean shift filtering, (c) the first 30 clusters (color-coded by cluster), and (d) 17 textured clusters (textures from source images).

Mean shift filtering. The mean shift filter is performed on color and space jointly. We map the RGB color space into LUV space, which is more perceptually meaningful. We define our multi-variate kernel as the product of two radially symmetric kernels:

$K_{h_s, h_r}(\mathbf{x}) = \frac{C}{h_s^2 h_r^2} k_E \left(\left| \frac{\mathbf{x}_s}{h_s} \right|^2 \right) k_E \left(\left| \frac{\mathbf{x}_r}{h_r} \right|^2 \right)$, where \mathbf{x}_s is the spatial vector (2D coordinates), and \mathbf{x}_r is the color vector in LUV, and C is the normalization constant. $k_E(x)$ the profile of Epanechnikov kernel, $k_E(x) = 1 - x$ if $0 \leq x \leq 1$, and 0 for $x > 1$. The bandwidths parameters h_s and h_r are interactively set by the user. In our experiments, h_s ranged from 6 to 8 and h_r from 3 to 7. The segmentation results were reasonable as long as the values used were within the specified ranges.

Region split or merge. After applying mean-shift filtering, we build a graph on the image grid with each pixel as a node; edges are established between 8-neighboring nodes if their (mean-shift) color difference is below a threshold (1 in our implementation). Prior to the split or merge operation, the user specifies the range of valid leaf sizes. Connected regions that are too small are merged with neighboring ones until a valid size is reached. On the other hand, connected regions that are too large are split into smaller valid ones. Splitting is done by seeding and region growing; the seeds can be either automatic by even distribution or interactively set. This split

or merge operation produces a set of new regions.

Color-based clustering. Each new region is considered a candidate leaf. We use K -means clustering method to obtain about 20 to 30 clusters. We only keep about 10 clusters associated with the brightest colors, as they are much more likely to represent visible leaves. Each new region in the kept clusters is fitted to an ellipse through singular value decomposition (SVD).

User interaction. The user can click to add a seed for splitting and creating a new region, or click on a specific cluster to accept or reject it. With the exception of the fig tree shown in Figure 1, the leaves were fully automatically segmented. (For the fig tree, the user manually specified a few boundary regions and clusters.)

6.2 Adding leaves to branches

There are two types of leaves that are added to the tree model: leaves that are created from the source images using the results of segmentation and clustering (Section 6.1), and leaves that are synthesized to fill in areas that either are occluded or lack coverage from the source viewpoints.

Creating leaves from segmentation. Once we have produced the clusters, we now proceed to compute their 3D locations. Recall that each region in a cluster represents a leaf. We also have a user-specified generic leaf model for each tree example (usually an ellipse, but a more complicated model is possible). For each region in each source image, we first find the closest pre-computed 3D point (Section 4) or branch (Section 5) along the line of sight of the region’s centroid. The 3D location of the leaf is then snapped to the closest pre-computed 3D point or nearest 3D position on the branch. Using branches to create leaves is necessary to make up for the possible lack of pre-computed 3D points (say, due to using a small number of source images).

The orientation of the generic leaf model is initially set to be parallel to the source image plane. In the case where more than three pre-computed 3D points project onto a region, SVD is applied to all these points to compute the leaf’s orientation. Otherwise, its orientation is such that its projection is closest to the region shape.

This approach of leaf generation is simple and fast, and is applied to each of source images. However, since we do not compute the explicit correspondences of regions in different views, it typically results in multiple leaves for a given corresponding leaf region. (Correspondence is not computed because our automatic segmentation technique does not guarantee consistent segmentation across the source images.) We just use a distance threshold (half the width of a leaf) to remove redundant leaves.

Synthesizing missing leaves. Because of lack of coverage by the source images and occlusion, the tree model that has been reconstructed thus far may be missing a significant number of leaves. To overcome this limitation, we synthesize leaves on the branch structure to produce a more evenly distributed leaf density.

The leaf density on a branch is computed as the ratio of the number of leaves on the branch to the length of the branch. We synthesize leaves on branches with the lowest leaf densities (bottom third) using the branches with the highest leaf densities (top third) as exemplars.

7 Results

In this section, we show reconstruction results for a variety of trees. The recovered models have leaves numbering from about 3,000 to 140,000. We used MayaTM for rendering; note that we did not model complex phenomena such as inter-reflection and subsurface scattering of leaves. In our experiments, image acquisition using an off-the-shelf digital camera took about 10 minutes. The computation time depends on the complexity of the tree. Automatic visible branch reconstruction took 1-3 minutes while manual editing took about 5 minutes. Invisible branches were reconstructed in about 5 minutes while leaf segmentation took about 1 minute per image.

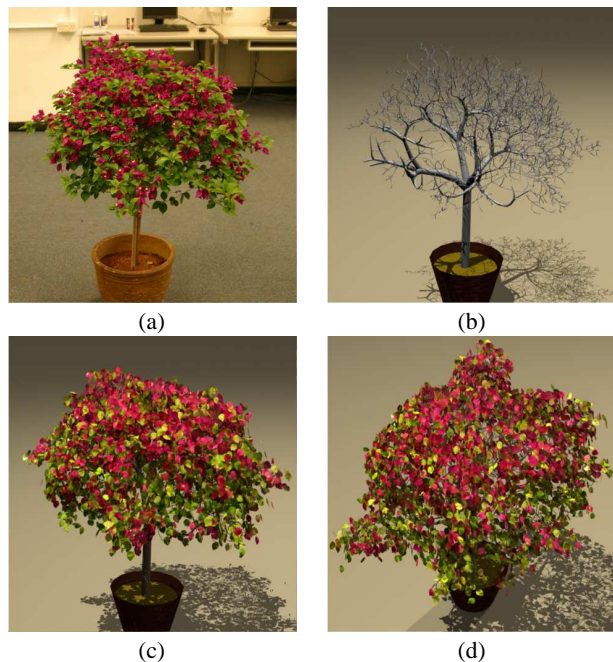


Figure 7: Potted flower tree example. (a) One of the source images, (b) reconstructed branches, (c) complete tree model, and (d) model seen from a different viewpoint.

The final stage of leaf population took 3-5 minutes.

The fig tree shown in Figure 1 was captured using 18 images covering about 180° . It is a typical but challenging example as there are substantial missing points in the crown. Nevertheless, its shape has been recovered reasonably well, with a plausible-looking branch structure. The process was automatic, with the exceptions of manual addition of a branch and a few adjustments to the thickness of branches.

The potted flower tree shown in Figure 7 is an interesting example: the leaf size relative to the entire tree is moderate and not small as in the other examples. Here, 32 source images were taken along a complete 360° path around the tree. Its leaves were discernable enough that our automatic leaf generation technique produced only moderately realistic leaves, since larger leaves require more accurate segmentation. The other challenge is the very dense foliage—dense to the extent that only the trunk is clearly visible. In this case, the user supplied only three simple replication blocks shown in Figure 5(d); our system then automatically produced a very plausible-looking model. About 60% of the reconstructed leaves relied on the recovered branches for placement. Based on leaf/tree size ratio, this example falls in the middle of the bush/tree spectrum shown in Figure 2.

Figure 8 shows a medium-sized tree, which was captured with 16 images covering about 135° . The branches took 10 minutes to modify, and the leaf segmentation was fully automatic. The rightmost image in Figure 8 shows a view not covered by the source images; here, synthesized leaves are shown as well.

The tree in Figure 9 is large with relatively tiny leaves. It was captured with 16 images covering about 120° . We spent five minutes editing the branches after automatic reconstruction to clean up the appearance of the tree. Since the branches are extracted by connecting nearby points, branches that are close to each other may be merged. The rightmost visible branch in Figure 9 is an example of a merged branch.



Figure 8: Medium-sized tree example. From left to right: One of the source images, reconstructed tree model, and model rendered at a different viewpoint.



Figure 9: Large tree example. Two of the source images on the left column. Reconstructed model rendered at the same viewpoint on the right column.

8 Concluding remarks

We have described a system for constructing realistic-looking tree models from images. Our system was designed with the goal of minimizing user interaction in mind. To this end, we devised automatic techniques for recovering visible branches, generating plausible-looking branches that have been originally occluded, and populating the tree with leaves.

There are certainly ways for improving our system. For one, the replication block needs not necessarily be restricted to being part of the visible branches in the same tree. It is possible to generate a much larger and tree-specific database of replication blocks. The observed set of replication blocks can be used to fetch the appropriate database for branch generation, thus providing a richer set of branch shapes.

Our system currently requires that the images be pre-segmented into tree branches, tree leaves, and background. Although there are many automatic techniques for segmentation, the degree of automation for *reliable* segmentation is highly dependent on the complexity of the tree and background. We currently do not see an adequate solution to this issue.

Acknowledgements

We would like to thank Tian Fang for rendering all results in this paper. The work is supported by Hong Kong RGC projects 619005 and 619006.

References

- BRECKON, T. P., AND FISHER, R. B. 2005. Non-parametric 3D surface completion. In *Proc. Int'l Conf. on 3-D Digital Imaging and Modeling*, 573–580.
- BROSTOW, G. J., ESSA, I., STEEDLY, D., AND KWATRA, V. 2004. Novel skeletal representation for articulated creatures. In *Proc. European Conf. on Computer Vision*, 66–78.
- COMANICIU, D., AND MEER, P. 2002. Mean shift: A robust approach toward feature space analysis. *IEEE Trans. on Pattern Analysis and Machine Intelligence* 24, 5, 603–619.
- DE REFFYE, P., EDELIN, C., FRANCON, J., JAEGER, M., AND PUECH, C. 1988. Plant models faithful to botanical structure and development. In *ACM SIGGRAPH*, 151–158.
- EFROS, A. A., AND LEUNG, T. K. 1999. Texture synthesis by non-parametric sampling. In *Proc. of IEEE Int'l Conf. on Computer Vision*, 1033–1038.
- HAN, F., AND ZHU, S.-C. 2003. Bayesian reconstruction of 3D shapes and scenes from a single image. In *Proc. IEEE Workshop on Higher-Level Knowledge in 3D Modeling and Motion Analysis*, 12–20.
- IJIRI, T., OWADA, O., OKABE, M., AND IGARASHI, T. 2005. Floral diagrams and inflorescences: Interactive flower modeling using botanical structural constraints. *ACM Trans. on Graphics (SIGGRAPH)* 24, 3 (July), 720–726.
- LHULLIER, M., AND QUAN, L. 2005. A quasi-dense approach to surface reconstruction from uncalibrated images. *IEEE Trans. on Pattern Analysis and Machine Intelligence* 27, 3, 418–433.
- MECH, R., AND PRUSINKIEWICZ, P. 1996. Visual models of plants interacting with their environment. In *ACM SIGGRAPH*, 397–410.
- NOSER, H., RUDOLPH, S., AND STUCKI, P. 2001. Physics-enhanced L-systems. In *Proc. Int'l Conf. in Central Europe on Computer Graphics, Visualization and Computer Vision*, vol. 2, 214–221.
- PRUSINKIEWICZ, P., JAMES, M., AND MECH, R. 1994. Synthetic topiary. In *ACM SIGGRAPH*, 351–358.
- OKABE, M., OWADA, S., AND IGARASHI, T. 2005. Interactive Design of Botanical Trees using Freehand Sketches and Example-based Editing. *Eurographics*, 487–496.
- PRUSINKIEWICZ, P., MUENDERMANN, L., KARWOWSKI, R., AND LANE, B. 2001. The use of positional information in the modeling of plants. In *ACM SIGGRAPH*, 289–300.
- QUAN, L., TAN, P., ZENG, G., YUAN, L., WANG, J., AND KANG, S. B. 2006. Image-based plant modeling. *ACM Trans. on Graphics (SIGGRAPH)* 25, 3 (August), 772–778.
- RECHE-MARTINEZ, A., MARTIN, I., AND DRETTAKIS, G. 2004. Volumetric reconstruction and interactive rendering of trees from photographs. *ACM Trans. on Graphics (SIGGRAPH)* 23, 3 (August), 720–727.

- SAKAGUCHI, T. 1998. Botanical tree structure modeling based on real image set. In *ACM SIGGRAPH 1998 (Tech. Sketch)*, 272.
- SHLYAKHTER, I., ROZENOER, M., DORSEY, J., AND TELLER, S. 2001. Reconstructing 3D tree models from instrumented photographs. *IEEE Computer Graphics and Applications* 21, 3 (May/June), 53–61.
- VAN HAEVRE, W., AND BEKAERT, P. 2003. A simple but effective algorithm to model the competition of virtual plants for light and space. In *Proc. Int'l Conf. in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*.
- WEBER, J., AND PENN, J. 1995. Creation and rendering of realistic trees. In *ACM SIGGRAPH*, 119–127.
- XU, H., GOSSETT, N., AND CHEN, B. 2005. Knowledge-Based Modeling of Laser-Scanned Trees. In *SIGGRAPH'05 Sketches*.