Pacific Graphics 2010
P. Alliez, K. Bala, and K. Zhou
(Guest Editors)

Volume 29 (2010), Number 7

Modeling Complex Unfoliaged Trees from a Sparse Set of Images

Luis D. Lopez, Yuanyuan Ding, and Jingyi Yu

University of Delaware, USA

Abstract

We present a novel image-based technique for modeling complex unfoliaged trees. Existing tree modeling tools either require capturing a large number of views for dense 3D reconstruction or rely on user inputs and botanic rules to synthesize natural-looking tree geometry. In this paper, we focus on faithfully recovering real instead of realistically-looking tree geometry from a sparse set of images. Our solution directly integrates 2D/3D tree topology as shape priors into the modeling process. For each input view, we first estimate a 2D skeleton graph from its matte image and then find a 2D skeleton tree from the graph by imposing tree topology. We develop a simple but effective technique for computing the optimal 3D skeleton tree most consistent with the 2D skeletons. For each edge in the 3D skeleton tree, we further apply volumetric reconstruction to recover its corresponding curved branch. Finally, we use piecewise cylinders to approximate each branch from the volumetric results. We demonstrate our framework on a variety of trees to illustrate the robustness and usefulness of our technique.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Modeling Applications

1. Introduction

The problem of modeling and generating realistic trees has attracted much attention from many research areas in recent years. Applications are numerous, from the computer graphics perspective as well as from the plant sciences perspective. For example, realistic 3*D* tree models can be integrated into film footage to reproduce a special season, or with architectural designs to add the realism to virtual urban environments. Recovering geometric structures of real trees can also reveal new botanic rules, and help plant scientists to study the influence of genetic and environmental factors on tree growth.

In computer graphics, tremendous efforts have been focused on developing easy-to-use tools for *synthesizing* naturally-looking tree models. Rule-based systems, such as L-system, apply botanic rules or geometric patterns to guide tree generation. Sketch-based approaches allow users to draw coarse tree geometry and then synthesize structural details via physical-based simulation [OOI06], probabilistic optimization [CNX*08], or texture synthesis [TFX*08]. These methods can produce visually pleasing results, although the synthesized tree geometry may differ greatly from the real ones. For example, synthesized tree branches have homogenous shapes whereas real branches often appear irregular and exhibit complex occlusions between each other, as shown in Fig 1.

In this paper we present a new image-based technique to faithfully reproduce real instead of realistically-looking geometry of unfoliaged trees. Previous approaches have used a 3*D* scanner to obtain the range data of the tree [XGC05]. However, they require elaborate point cloud registrations and hole filling [BF05, EL99] and are less suitable for portable, on-site acquisition. Our work is inspired by the recent image-based approach by Tan et al. [TZW*07] that recovers 3*D* tree geometry from a set of images. The major difference is that their method requires capturing a large number of images over a wide range of view angles for conducting reliable 3*D* reconstruction whereas we only use a very small number (4∼6) of images over a moderate range of view angles. Therefore, our method is more suitable for on-site acquisition with a hand-held (e.g., a digital SLR) camera.

Published by Blackwell Publishing, 9600 Garsington Road, Oxford OX4 2DQ, UK and 350 Main Street, Malden, MA 02148, USA.

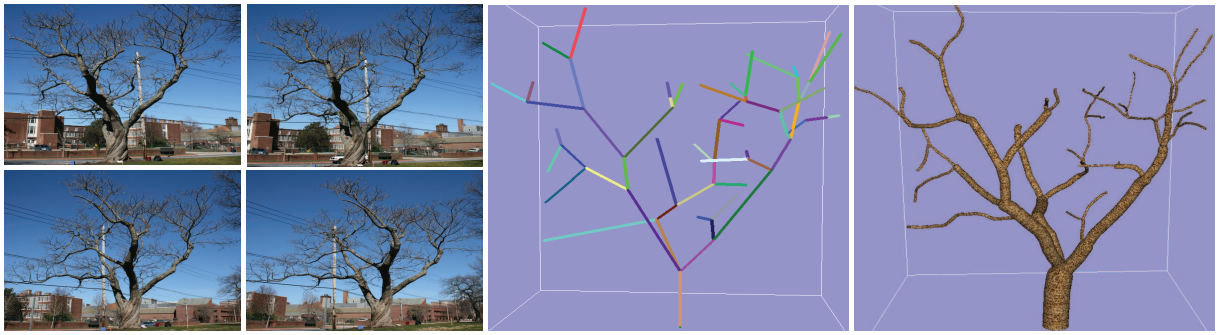

Figure 1: *Recovering an Unfoliaged Eml Tree Using Our Approach. Left: the 4 input images; Middle: our recovered 3D skeleton tree; Right: our reconstructed 3D tree model.*

For each input view, we first estimate a 2D skeleton graph from its matte image and then estimate a 2D skeleton tree from the graph. Notice that real tree branches often obscure each other in 3*D*, and thus exhibit complex partial occlusions in images. Therefore, we impose 2*D* tree topology for recovering 2D skeleton trees from the images. We then develop a simple but effective technique to find the optimal 3*D* skeleton tree that is most consistent with the 2D skeletons. Our solution is based on matching the nodes between the 2D skeleton trees using a hypothesis tree, where we apply geometric and topological cues for efficient pruning. For each edge in the recovered 3*D* skeleton tree, we further apply volumetric reconstruction to recover its corresponding curved branch. Finally, we use piecewise cylinders to approximate the branch, where the cylinder radii are estimated from the volumetric results. We demonstrate our framework on a variety of trees, and show that our method can not only faithfully reproduce complex 3D geometry but also minimize user efforts by using much fewer images.

2. Previous Work

Most existing tree modeling methods can be classified into two categories: synthesis-based and reconstruction-based.

Synthesis Methods: Approaches in this category focus on synthesizing tree geometry from heuristics or knowledge such as botanic rules [NRS01, WP95], shape grammar [PJM94], physical simulations [OOI06], or probabilistic optimization [CNX*08]. Rule-based methods assume that the growth of trees follows specific patterns that can be derived from botanic science [DREF*88]. [MP96] integrates a small set of generative rules/grammar with the L-system for synthesizing both branch and leaves. These rules or grammar are commonly associated with a large number of parameters. Therefore, most rule-based systems rely on user's familiarity with the model to construct specific types of trees. Recently, Chen et al. [CNX*08] build a tree database that contains typical tree exemplars along with their associated parameters. With their modeling system, the user only needs to draw a simple 2D sketch of the tree and the best matching exemplar in the database is used to generate its complete 3*D* geometry.

Image-based methods have also been developed for synthesizing 3*D* trees [HZ03, Sak98]. Reche-Martinez et al. [RMMD04] compute a volumetric representation of the tree from only three images. They estimate variable opacity for the voxels and apply volume rendering for synthesizing new views. Their volumetric results, however, do not reveal the actual geometric structures of trees. Neubert et al. [CNX*08] allow the user to draw skeleton geometry on top of tree images and use particle flow simulation to synthesize 3*D* tree structure. Most recently, Tan et al. [TFX*08] developed a simple sketching method that can synthesize realistic 3*D* tree models from a single image. In their system, the user only needs to mark up the leaf region, the main trunk, and a few branches, and their algorithm automatically synthesizes 3*D* tree geometry from a library of elementary trees. It is important to note that all these approaches aim to generate realistically-looking trees, and the synthesized tree geometry may differ greatly from the real one.

Reconstruction Methods: In contrast, reconstruction-based methods aim to faithfully recover 3*D* tree geometry. Xu et al. [XGC05] use a laser scanner to acquire the range data of the tree. Their method is able to produce very high-quality 3*D* models. However, like most range data, their tree model contains holes, and it relies on robust geometric algorithms for filling in the missing information. The use of a laser scanner also makes it less appealing for portable, on-site acquisition. Instead of using the scanner, Tan et al. [QTZ*06] capture a large number of images surrounding the tree for dense 3*D* reconstruction. Their method is able to recover highly accurate tree models. A downside of their technique, however, is that it requires the user to segment the tree out from the back in each image. Even with advanced segmentation [RKB04, KZ02] and matting tools [WC07, LSTS04], segmenting a large number of images is time consuming and tedious. Therefore, we pursue a solution that uses much fewer images. Classical stereo matching and volumetric reconstruction algorithms use a small number of images, and, in theory, could also be used to recover unfoliaged trees. In practice, these algorithms perform poorly on tree-shaped objects, as tree geometry violates many common assumptions in these methods: trees are highly concave,

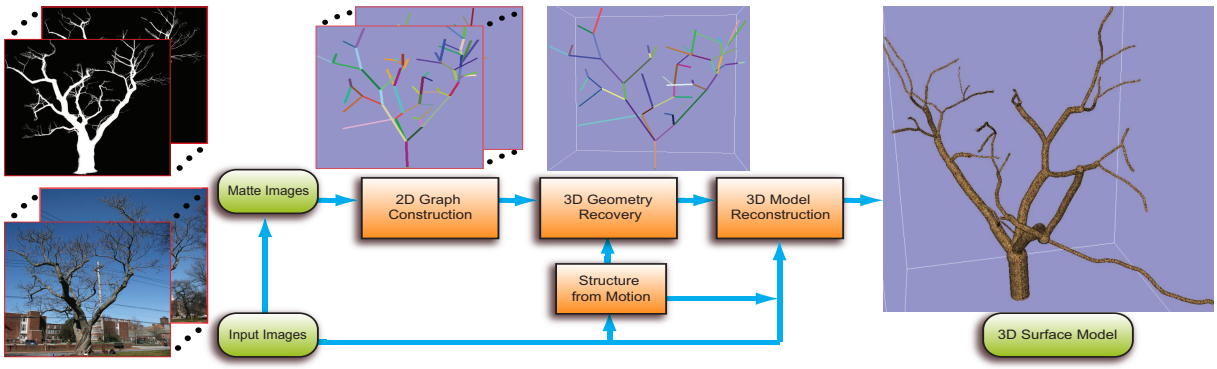

Figure 2: *Overview of Our Image-based Tree Modeling System.*

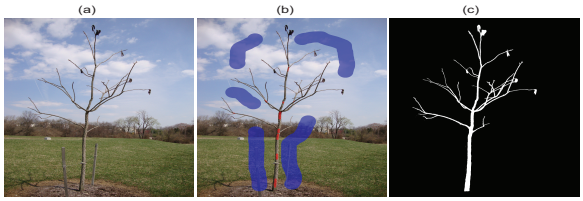

Figure 3: *Matte Estimation using algorithm [WC07] (A) Source Image; (B) User Input Strokes; (C) Estimated Alpha Matte.*

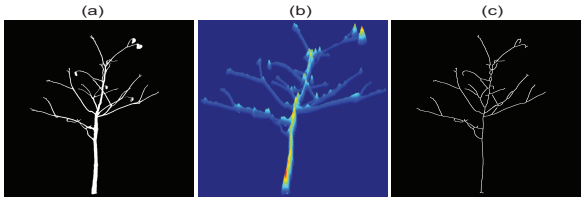

Figure 4: *Skeleton Estimation from Alpha Matte: (a) input matte image; (b) distance transform on (a); (c) the estimated Median Axis of (a).*

tree branches have very similar appearance and they obscure each other; branch occlusion also appear significantly different across the views. In line with our method, Shlyakhter et al. [SRDT01] reconstructs a visual hull according to image silhouette and compute the medial axis of the volume to fit a simple L-system for branch generation. Sakaguchi et al. [Sak98] use simple branching rules in voxel space. However, their recovered tree geometry is in general very coarse and lacks realism due to low quality volumetric reconstructions. We also use the medial axis in our approach; the difference is that we apply it to approximate 2D skeleton trees and use prior-based multi-view reconstruction to recover the 3*D* skeleton tree.

3. Recovering 2D Skeleton Trees

3.1. Matte Estimation and Camera Calibration

Similar to the image-based tree modeling method [QTZ*06], we start with segmenting the foreground tree from the background environment. While [QTZ*06] uses hard segmentation, we choose to estimate the alpha matte of the tree. This is because, unlike bushes or heavily leaved trees, unfoliaged trees contain thin branches that blend into the background and are difficult to segment. The alpha matte, instead, can capture these details and is consistent with our approach for finding the medial-axis representation (Section 3.2). To compute the alpha matte, we directly use the Robust Matting [WC07] algorithm that allows the user to draw foreground and background strokes and progressively refine the alpha matte. Figure 3 shows the recovered alpha matte of a sample input image. For trees with many small twigs, we further post-process the matte images to remove these fine details. Specifically, we threshold the matte image and then apply a median filter.

Our method requires obtaining the camera calibration matrix for each view. In [QTZ*06], structure-from-motion (SfM) technique was used for conducting both camera calibration and dense point cloud reconstructions. In their case, plants/trees contain many heterogenous features (e.g., leaves vs. branches) that appear coherently across the views. In our case, branches of unfoliaged trees have very similar colors and textures and the occlusion patterns vary significantly across views due to sparse sampling.

To resolve this problem, we put special calibration targets on ground and onto tree trunks at the acquisition stage, and then treat them as feature points in the standard point-based calibration. In case that we could not find sufficient calibration features across the views, we allow the user to hand mark correspondences, e.g., using the tip of recognizable branches. It is important to note that by using a small set of images, we significantly reduce the user efforts even counting for manual calibrations.

3.2. 2D Skeleton Graph

Next, we construct a 2D skeleton graph for each matte image. Our solution consist on finding the medial axis geometry of the matte by using the gradient vector diffusion technique [XP98]. As follows, we briefly reiterate the main steps for computing the skeleton geometry.

From the matte image, we first compute a distance field image DT [Bor86]: for each pixel p, we compute its shortest distance to the background pixels (i.e., whose alpha value is zero). We further apply spatially variant Gaussians to smooth DT [LLBL07]. We use $\tilde{DT}$ to represent the smoothed distance field. Next, we compute *gradient vector field* GVF of $\tilde{DT}$ as $GVF = \nabla\tilde{DT} = \left(\frac{\partial\tilde{DT}}{\partial x}, \frac{\partial\tilde{DT}}{\partial y}\right)$ and apply isotropic diffusion to further smooth GVF as $G\tilde{V}F$. We then construct a Skeleton Strength Map (SSM) [XP98]: for every pixel p, we compute the consistency of its gradient vector along all possible directions as:

$$SSM(p) = max\left(0, \sum_{p' \in N(p)} \frac{G\tilde{V}F(p') \cdot (p' - p)}{\|p' - p\|}\right) \quad (1)$$

where $N(p)$ is the set of eight neighbors of p.

Finally, we identify the critical points (i.e., the ones lying on the medial axis) as local maxima on the SSM. It is commonly observed that the critical points form disconnected regions [LLBL07]. Therefore, for each region, we find two points inside the region with the minimal $||G\tilde{V}F||$, where $||\cdot||$ represents the magnitude of the gradient, and connect the points by finding the shortest distance on $G\tilde{V}F$. We also connect the regions in the similar way. Figure 4(c) shows the medial axis image computed from 4(a).

3.3. From Graph to Tree

Once we recover the 2D skeleton graph, we construct its corresponding 2D skeleton tree. To do so, we first identify the root node and the leaf nodes in the graph. The leaf nodes are detected by finding the endpoints on the skeleton graph and the lowest leaf node corresponds to the root. Next, for each leaf node, we trace its shortest path to the root on the skeleton graph.

To further identify the bifurcation/multi-furcation nodes, we apply a simple labeling scheme. We sequentially traverse the shortest paths for all leaf nodes. For each shortest path, we assign a unique label to all pixels lying the path. As we traverse along a specific path, we detect the first pixel on it that has been previously labeled and mark it as a junction node. Notice that there are two types of junction nodes: Y-junction nodes that correspond to the real bi- or multi-furcation points and should be included in the tree and X-junction nodes that are caused by occlusions between the branches and should be discarded as shown in Fig 5. For X-junctions, we also need to fix the child-parent relationship.

To do so, once we finish labeling all shortest paths, we check whether the detected junction node is attached to any unlabeled path. If this is the case, the junction node is an X-junction. Otherwise, it is a Y-junction. For a X-junction node, we need to splice the node and fix the connectivity. Therefore, we check the angles between the edges and apply a Greedy algorithm to find the optimal connectivity

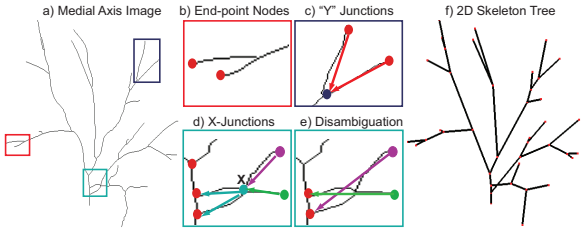

Figure 5: *Reconstructing 2D Skeleton Trees from 2D Graphs. (a) A 2D skeleton graph. (b) to (d): We identify the endpoints and the junction nodes. (e): We fix the X-junction nodes. f): Our final recovered 2D tree.*

that yields to the highest angle consistency. In the rare case when multiple branches occlude each other by two or more branches at the same point, it is highly challenging to automatically correct the connectivity between the nodes. Therefore, our framework allows the user to manually fix the connectivity in such cases.

4. 3D Skeleton Tree Reconstruction

In this section, we present a simple but efficient algorithm to reconstruct the $3D$ skeleton tree from the recovered $2D$ skeleton trees. Our solution is based on using geometric and topological cues for robust 2D tree matching.

We use a 4-tuple $\mathbf{t} = (\mathbf{v}, \mathbf{e}, r, \theta)$ to denote the $2D$ skeleton trees, where $\mathbf{v} = \{v_i, i = 1, 2, \cdots, N\}$ represents the set of $2D$ nodes in $\mathbf{t}$, $\mathbf{e} \subset \mathbf{v} \times \mathbf{v}$ is the set of edges contained by $\mathbf{t}$, r is the root, and $\theta = (K, [R|T])$ is the camera parameters.

We start by matching a pair of 2D skeleton trees. The matching result corresponds to a bijective correspondence set between the nodes on the pair of trees. Given two trees $\mathbf{t}^l$ and $\mathbf{t}^r$, our first step is to compute the matching cost between the nodes in both trees. For clarity, we use l_i to represent the i-th node in $\mathbf{t}^l$ and r_j to represent the j-th node in $\mathbf{t}^r$.

Matching Cost Table We compute the matching cost C_{ij} by first finding the corresponding rays (lines) passing through l_i and r_j in their cameras. In theory, if they match, they should correspond to the same 3D point and intersect in 3D space. In reality, since the tree nodes are obtained via skeletonization and the camera calibrations contain errors, the rays are generally oblique. Therefore, we compute the distance d_{ij} between the rays and find the corresponding midpoint $\dot{Q}_{ij}$ of the minimal distance segment. We also identify the "impossible" matching node-pairs: if d_{ij} is too large or $\dot{Q}_{ij}$ lies outside our pre-defined frustum, we simply mark $C_{ij} = +\infty$. Otherwise, we set $C_{ij} = d_{ij}$. We further assign a penalty cost for resolving occlusion matches.

Pair-wise Tree Matching. Next, we find the optimal matching between $\mathbf{t}^l$ and $\mathbf{t}^r$ that would yield to the minimal total matching cost. It is well known that 2D tree matching is an NP hard problem. In this paper, we adopt a simple exhaustive search approach by constructing a hypothesis

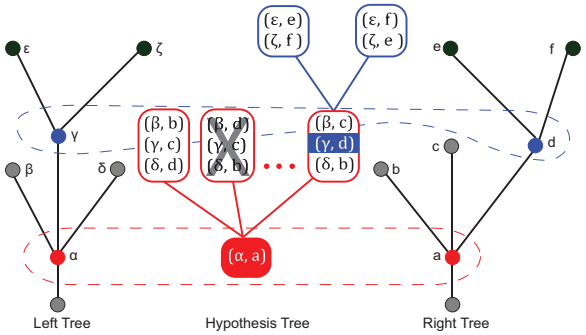

Figure 6: *Pair-wise 2D Tree Matching. Based on the hypothesis that α matches a, we generate a sequence of new hypothesis nodes and apply pruning to reduce the search space.*

tree. To prevent the hypothesis tree growing too fast, we use the cost matrix to efficiently prune the hypothesis. We assume that the root nodes l_0 and r_0 always match and make the matching pair as the root of the hypothesis tree. Given two matched nodes l_i and r_j, we check all possible bijective mappings between the child nodes $CH(l_i)$ and $CH(r_j)$ as hypothesis, where we allow a node to match to *nil*. If a specific bijective mapping contains a pair of nodes with ∞ matching cost, we simply discard the hypothesis. We recursively grow the hypothesis tree via breadth first search and propagate the average matching cost to the next level, as shown in Fig 6.

Since each 2D skeleton tree is constructed separately, a child node directly connected to its parent node in one view may connect to some intermediate node before its parent node in the other. Therefore, we need to consider potential "splicing" between the nodes. To resolve this issue, we also consider the hypothesis of matching grandchild nodes $GCH(l_i)$ and $GCH(r_j)$ against the child nodes $CH(r_j)$ and $CH(l_i)$ respectively when constructing the hypothesis tree. Fig 7 shows the pseudo code of our pair-wise tree matching algorithm.

3D Skeleton Trees. Next, we apply the pairwise 2D tree matching results to reconstruct the 3D skeleton tree. In theory, we should conduct our matching algorithm between all possible pairs between the K 2D skeleton trees. In practice, we adopt a simpler approach: we pick the skeleton tree $\mathbf{t}^f$ that contains the most visible branches as the reference view and match it with the rest of the skeleton trees. For each node v_i^f, we find its matching nodes in the rest of the skeleton trees. Each matching pair has already stored its corresponding 3D point $\dot{Q}$ (i.e., the midpoint of the short segment between the corresponding rays). We map all these points to the 3D space and apply K-means cluster to remove the outliers. Finally, we compute their centroid as the corresponding 3D node for f_i. We apply our technique to all nodes in $\mathbf{t}^f$ to find their corresponding 3D nodes and impose the topology of $\mathbf{t}^f$ for connecting them. Fig 9 shows example 3D skeleton trees recovered by our method.

```
double matchingTrees(TreeNode &l0, TreeNode &r0)
{   compute CH(l0), CH(r0), GCH(l0), and GCH(r0);
    list the set M of combinations of bijective matchings between
          CH(l0), GCH(l0), GCH(r0) and CH(r0), CH(r0), CH(l0);
    min_cost = +∞ ;
    for Mk ∈ M
    {   bDiscard = false; C̄ = 0; acc_recurr = 0;
        for (li, rj) ∈ Mk
        {   Look up C, get the cost Cij;
            if (Cij == +∞) {
                bDiscard = true; break ;
            }
            C̄ += Cij ;
            if (CH(li)!=∅ or CH(rj)!=∅) {
                acc_recurr += matchingTrees(ui, vi);
            }
        }
        if (bDiscard==true) {
            discard Mk ; continue ;
        }
        tot_cost = C̄ + acc_recurr ;
        if (tot_cost < min_cost) {
            Mmin = Mk; min_cost = tot_cost ;
        }
    }
    return min_cost ;
}
```

Figure 7: *Pseudo Code of Our Tree Matching Algorithm.*

5. Branch Recovery

Once we extracted the 3*D* skeleton tree from the images, we construct the complete 3*D* tree geometry. Our approach first applies per-branch-based volumetric reconstruction and then estimates the branch geometry (e.g., shape and size) from the volumetric result. The major difference between our reconstruction method and the classical visual hull approach [SRDT01] is that we use the recovered 3*D* skeleton tree as priors to effectively reduce the initial volume size. Furthermore we show that, even with highly noisy volumetric reconstruction results, our technique is still able to faithfully and robustly recover high quality tree geometry by using the 3D skeleton tree prior.

5.1. Per-Branch Volumetric Reconstruction.

Classical volumetric reconstruction first bounds the scene with a 3*D* volume by intersecting the view frustums of all cameras. The volume is then discretized into voxels with a size coherent with the input image resolution. Notice that in order to recover fine branches from the input images, full image resolution needs to be used for discretization (e.g., 2Kx2K in our case) and the resulting volume size would be too large to process or store. Therefore, existing volumetric-based tree modeling methods have trade off volume resolution for processing speed, and they are only able to recover very coarse tree geometry [SRDT01].

Our technique resolves the resolution issue by directly using the recovered 3*D* skeleton tree to guide volume discretization. Specifically, we use a 3*D* cylinder to bound each branch. The radius of the cylinder is chosen large enough to

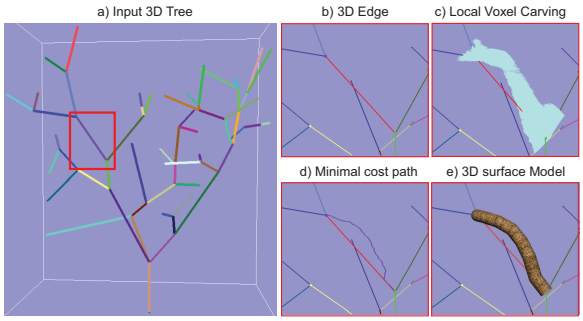

Figure 8: *Per-Branch Volumetric Reconstruction.*

cover all pixels of the branch when back-projected onto the views. We then discretize each cylinder into voxels at a resolution of twice the image resolution to avoid undersampling. Since each branch only covers a small amount of pixels, our method significantly reduces the initial volume size. In our experiments, the size of our cylinder volume representation is usually only 5 percent or less of the standard view frustum discretization.

Next, we apply local space carving on each cylinder volume. For each voxel inside the cylinder, we project it back to all input images. To further reduce the computational overhead, we simply carve out the voxel if its corresponding pixel lies in the background of the matte image in any views. For the remaining voxels, we compute their color consistencies across the input images and store the mean and the variance of each voxel.

To recover the actual branch geometry, standard volumetric reconstruction [KS00] could be applied to refine the volume. However, since we only use very few views, the volumetric results are usually very noisy, as shown in Fig 8(C). Therefore, we set out to find a curve that best matches the shape of the branch and then use primitive geometry such as cylinders along the curve to approximate the branch. Specifically, we first locate two voxels that correspond to the endpoints of the branch edge in the 3*D* skeleton tree. We then use Fast March algorithm [TvW02] to compute the optimal path between the two nodes that minimizes the inconsistency cost. Finally, we approximate the branch geometry by concatenating cylinders along the pass. To estimate the cylinder radius, at every voxel V on the optimal path, we estimate a local cut plane and find all uncarved voxels lying on the plane. We compute the weighted distance of the voxels using their stored consistency values. We further average the radii over all voxels on the path and use it as the radius of the curved cylinder to approximate the 3*D* branch. Since the peripheral voxels of a branch are often less reliable and hence are assigned with a smaller weight, our estimated cylinder radius tends to be smaller than its actual value. Fig 8(E) shows our reconstruction result of a sample branch on an Elm tree. We refer the readers to the supplementary video to view the complete tree.

Sequence	Images	Resolution	Branches
Eml	4	1408×940	58
CoffeeTree	5	2448×3264	48
Oak	4	1880×2816	80

Table 1: *Tree/Image Information in Our Experiments.*

6. Results and Discussions

We demonstrate our framework on reconstructing 3D tree models using captured images of real trees. In all our examples, we only use 4-6 images that span 100° to 160° angles. The images were captured using a Canon EOS Digital Rebel Xti SLR camera.

We conduct our reconstruction algorithms on a DELL PC with Intel DualCore 2.4Ghz CPU and 4GB of RAM. We apply the Robust Matting algorithm [WC07] for alpha matte estimation. It takes about 2 minutes per image for the user to specify the strokes and obtain the mattes. Our 2*D* skeleton tree reconstruction algorithm takes about 5 seconds for each image. As was discussed in Section 3, our system occasionally requires manual separation of the overlapping branches in the 2*D* skeleton graph. In our experiments, such user interventions occur only four times on average for each image. The 3*D* Skeleton Tree is then automatically reconstructed using the algorithm described in Section 4 in less than 2 seconds. Branch geometry reconstruction takes about 10 to 25 seconds depending on the number of branches in the tree. Table 1 summarizes the image resolution (after cropping) and the estimated number of branches of the input trees.

Elm. The first row of Figure 9 shows an Elm tree reconstructed by our algorithm. For camera calibration, we stick 8 calibration targets onto the tree trunk and to the ground and our algorithm automatically detects and matches these targets via SfM. We capture 4 images towards the tree that span about 100 degree view angle. The tree contains many tiny branches on top of the major branches, making it challenging for the segmentation and skeletonization process. We thus post-process the matte images to remove these small branches while preserving the main branches. Column (a) shows the view that we pick for 3*D* reconstruction. Column (b), (c), and (d) show our recovered 3D skeleton geometry and branch geometry. Our method accurately captures the shape characteristics of the tree although it misses a few branches that are invisible to the view. We refer the reviewers to the supplementary video for the complete 3*D* tree geometry recovered by our algorithm.

Kentucky Coffee Tree. The second row of Fig 9 shows our reconstruction result of a coffee tree. The tree has only a few large leaves and appears significantly different from the background. These features make the matte estimation much easier. In our experiment, the user only needs to draw four to six strokes to obtain the high quality alpha mattes. However, the branches of the tree are very thin and we were unable to attach the calibration patterns to the tree trunk. Therefore, we

Figure 9: *Our Recovered 3D Tree Models. (a): The reference view used for tree reconstruction. (b) The recovered skeleton tree. (c) and (d): Two views of our final reconstructed 3D tree model. Each tree is reconstructed using no more than 6 input images.*

require that user manually mark correspondences on the five input images. Our recovered 3*D* tree geometry contains few discrepancies in the diameter of the branches. This is due to errors in the camera calibration results that later lead to less accurate volumetric reconstruction. Our technique, however, is still able to recover a reasonably good 3*D* reconstruction to faithfully represent the overall tree geometry.

Oak. In the bottom row of Fig 9, we apply our reconstruction method on a complex Oak. We capture four images towards the tree that span about 120 degree angle views. Similar to the Elm tree, it contains hundreds of tiny branches. To avoid reconstruction errors, we remove these branches in the matte images via median filters. Notice that although the tree geometry is highly irregular and several branches are strongly curved, our technique is able to faithfully recover most branches of the trees. In all three examples, our reconstructed tree models are not only consistent with the input images but also appear more realistic compared with the synthesis-based approaches.

7. Limitations and Future Work

We have presented a new image-based technique for modeling complex unfoliaged trees. Our method is able to faithfully capture, rather than synthesize, 3*D* tree geometry by only using a sparse set of images. The core of our algorithm is to actively integrate 2*D* and 3*D* tree topology as shape priors into the modeling process. Specifically, we have applied 2*D* tree topology for constructing the 2*D* skeleton trees from the matte images and further imposed 3*D* tree topology for matching 2*D* skeleton tree pairs. We have also used the resulting 3*D* skeleton tree to guide per-branch volumetric reconstruction for recovering the complete tree geometry. We have demonstrated our framework on a variety of trees. Our results on real tree images have shown that our method is robust and reliable.

An apparent limitation of our technique is that it can only handle unfiolaged trees. In our framework, we rely on accurate recovery of 2*D* skeleton trees for robust 3*D* reconstruction. If the tree is covered by leaves, it would be very challenging to construct the 2*D* skeleton tree from the image. One possible solution is to combine the 3*D* scanning technique [XGC05] or the dense multi-view reconstruction method [QTZ*06] to first obtain the 3*D* point cloud and then apply color/texture based segmentation to separate the branches from the leaves. The resulting branches, however will contain many holes due to leave occlusions. Rule-based methods may help to infer and complete the 2*D* skeleton trees.

Another limitation of our approach is that we pick the im-

age that contains most visible branches as the reference view for recovering the 3*D* skeleton tree. In other words, we do not equally treat all input images, and if there is a branch missing in the reference view, we will not be able to recover it. In the future we plan to study more sophisticated 3*D* skeleton tree reconstruction algorithm that can recover the branches observed from any view. Our method is also unable to reconstruct detailed tree geometry such as twigs. This is because our algorithm relies on coherent image matte and 2D skeleton estimation across the views to robustly recover the 3D geometry. For trees with fine twigs, it is challenging to maintain consistent topology in all views when applying the matting and skeletonization tools to individually process each image.

Our algorithm is purely image-based and does not use the botanic rules. In the future, we also plan to explore combining the Markov Random Field approach [CNX*08] to integrate these rules into $2D/3D$ skeleton tree reconstructions. For example, we could directly query the tree database to find the most consistent 3*D* skeleton tree from the 2*D* ones for further improving reconstruction quality. Finally, since our method provides a convenient way for recovering 3*D* tree geometry, we plan to capture a wide range of trees and contribute to the tree geometry database for studying new botanic rules and generating more realistic model templates.

Acknowledgements. We would like to thank the anonymous PG reviewers for their valuable comments and suggestions. The authors are supported in part by NSF grants MSPAMCS-0625931, IIS-CAREER-0845268 and CONACYT 171570.

References

[BF05] BRECKON T. P., FISHER R. B.: Non-parametric 3d surface completion. In *Proc. of the 5th Int. Conf. on 3D Digital Imaging and Modeling* (2005), pp. 573–580.

[Bor86] BORGEFORS G.: Distance transformations in digital images. *Comput. Vision Graph. Image Process. 34*, 3 (June 1986), 344–371.

[CNX*08] CHEN X., NEUBERT B., XU Y.-Q., DEUSSEN O., KANG S. B.: Sketch-based tree modeling using markov random field. *ACM Trans. Graph. 27*, 5 (2008), 1–9.

[DREF*88] DE REFFYE P., EDELIN C., FRANÇON J., JAEGER M., PUECH C.: Plant models faithful to botanical structure and development. In *SIGGRAPH '88* (1988), pp. 151–158.

[EL99] EFROS A. A., LEUNG T. K.: Texture synthesis by non-parametric sampling. *Computer Vision, IEEE International Conference on 2* (1999), 1033.

[HZ03] HAN F., ZHU S.-C.: Bayesian reconstruction of 3d shapes and scenes from a single image. In *HLK '03: Proc. of the 1st IEEE International Workshop on Higher-Level Knowledge in 3D Modeling and Motion Analysis* (2003), p. 12.

[KS00] KUTULAKOS K., SEITZ S.: A theory of shape by space carving. *IJCV 38*, 3 (July 2000), 199–218.

[KZ02] KOLMOGOROV V., ZABIH R.: What energy functions can be minimized via graph cuts? In *ECCV '02: Proceedings of the 7th European Conference on Computer Vision-Part III* (London, UK, 2002), pp. 65–81.

[LLBL07] LONGIN J. L., LI Q., BAI X., LIU W.: Skeletonization using ssm of the distance transform. In *ICIP (5)* (2007), pp. 349–352.

[LSTS04] LI Y., SUN J., TANG C.-K., SHUM H.-Y.: Lazy snapping. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers* (New York, NY, USA, 2004), ACM, pp. 303–308.

[MP96] MĚCH R., PRUSINKIEWICZ P.: Visual models of plants interacting with their environment. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (NY, USA, 1996), ACM, pp. 397–410.

[NRS01] NOSER H., RUDOLPH S., STUCKI P.: Physics-enhanced l-systems. In *WSCG* (2001), pp. 214–221.

[OOI06] OKABE M., OWADA S., IGARASHI T.: Interactive design of botanical trees using freehand sketches and example-based editing. In *ACM SIGGRAPH 2006 Courses* (2006).

[PJM94] PRUSINKIEWICZ P., JAMES M., MĚCH R.: Synthetic topiary. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1994), ACM, pp. 351–358.

[QTZ*06] QUAN L., TAN P., ZENG G., YUAN L., WANG J., KANG S. B.: Image-based plant modeling. *ACM Trans. Graph. 25*, 3 (2006), 599–604.

[RKB04] ROTHER C., KOLMOGOROV V., BLAKE A.: "grabcut": interactive foreground extraction using iterated graph cuts. In *SIGGRAPH '04* (2004), pp. 309–314.

[RMMD04] RECHE-MARTINEZ A., MARTIN I., DRETTAKIS G.: Volumetric reconstruction and interactive rendering of trees from photographs. *ACM Trans. Graph. 23*, 3 (2004), 720–727.

[Sak98] SAKAGUCHI T.: Botanical tree structure modeling based on real image set. In *SIGGRAPH '98* (1998), p. 272.

[SRDT01] SHLYAKHTER I., ROZENOER M., DORSEY J., TELLER S.: Reconstructing 3d tree models from instrumented photographs. *IEEE Comput. Graph. Appl. 21*, 3 (2001), 53–61.

[TFX*08] TAN P., FANG T., XIAO J., ZHAO P., QUAN L.: Single image tree modeling. *ACM Trans. Graph. 27*, 5 (2008), 108.

[TvW02] TELEA A., VAN WIJK J. J.: An augmented fast marching method for computing skeletons and centerlines. In *VISSYM '02: Proc. of the symposium on Data Visualisation* (Aire-la-Ville, Switzerland, 2002), Eurographics Association, pp. 251–ff.

[TZW*07] TAN P., ZENG G., WANG J., KANG S. B., QUAN L.: Image-based tree modeling. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers* (New York, NY, USA, 2007), ACM, p. 87.

[WC07] WANG J., COHEN M. F.: Optimized color sampling for robust matting. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on* (2007), pp. 1–8.

[WP95] WEBER J., PENN J.: Creation and rendering of realistic trees. In *SIGGRAPH '95* (1995), pp. 119–128.

[XGC05] XU H., GOSSETT N., CHEN B.: Knowledge-based modeling of laser-scanned trees. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Sketches* (New York, USA, 2005), ACM, p. 124.

[XP98] XU C., PRINCE J. L.: Snakes, shapes, and gradient vector flow. *Image Processing, IEEE Transactions on 7*, 3 (1998), 359–369.