CEIG - Spanish Computer Graphics Conference (2017)
F. J. Melero and N. Pelechano (Editors)

Tree Variations

Oscar Argudo, Carlos Andújar and Antoni Chica

Computer Science Department, Universitat Politècnica de Catalunya, Spain

Abstract

The cost-effective generation of realistic vegetation is still a challenging topic in computer graphics. The simplest representation of a tree consists of a single texture-mapped billboard. Although a tree billboard does not support top views, this is the most common representation for still image generation in areas such as architecture rendering. In this paper we present a new approach to generate new tree models from a small collection of RGBA images of trees. Key ingredients of our method are the representation of the tree contour space with a small set of basis vectors, the automatic crown/trunk segmentation, and the continuous transfer of RGBA color from the exemplar images to the synthetic target. Our algorithm allows the efficient generation of an arbitrary number of tree variations and thus provides a fast solution to add variety among trees in outdoor scenes.

1. Introduction

Vegetation (trees, bushes and grass) is an essential part of natural outdoor scenes. Many different plant representations have been proposed in the literature, including polygonal-based, image-based [BCF*05], point-based [DCSD02], volume-based [DN04], relief mapping-based [ACA16], and procedural [ACV*14] approaches. Here we focus exclusively on image-based representations of trees.

Billboards are the simplest tree representation and as such they are extensively used today in many applications including architecture rendering and video games. A billboard consists of a single texture-mapped quad that is rotated around its vertical axis so that it always faces the camera, thus exploiting the apparent axial symmetry of trees. Billboard textures are RGBA, where the alpha (opacity) channel segments foreground (tree) from background pixels, allowing non-tree parts of the texture to be easily discarded.

Billboards suffer from some well-known limitations. They do not support top views of the trees, since the image only shows a side view, and the billboard is allowed to rotate only around its vertical axis. Since billboards are planar, shading effects are limited, even when equipped with normal maps. Cast shadows correspond to that of the tree silhouette and these shadows are received as if the trees were planar. Concerning real-time rendering, billboards do not support view-motion parallax, and the fact that they always show the same side of the tree might become apparent e.g. when the camera rotates around the tree.

Despite the limitations above, the simplicity, speed and compactness of billboards, the realism provided by actual photographs of trees, and the availability of large collections of ready-to-use tree textures make tree billboards ubiquitous in many CG applications.

In this paper we explore different strategies for the automatic generation of tree variations from a collection of RGBA images (this image collection will be referred to as the tree library). We consider two main usage scenarios. The first one is oriented towards the *authoring* of new tree images. Given a user-defined subset of exemplars from the tree library, the algorithm creates plausible trees through random combinations of the chosen exemplars. These variations can be used to enrich existing plant libraries.

The second scenario aims at avoiding the perception of repeated copies of trees. If the same tree image is instanced multiple times, users will easily detect the repeated copies, resulting in poor, overly repetitive vegetation. Straightforward per-instance variations operating on shape (e.g. scaling) or appearance (e.g. HSL perturbations) do not suffice to overcome the impressive pattern-matching ability of the human visual system. Given an input tree image, the algorithm creates an arbitrary number of variations of the tree image by perturbing its shape and appearance using features extracted from the library.

A key ingredient of our approach is the combination of multiple tree shapes and color textures so that the resulting images have a realistic tree appearance. Regarding the overall tree shape, we encode tree contours by a fixed-length high-dimensional vector that enables high-quality morphing between two or more tree shapes. Indeed, we use contours from library exemplars to build a set of basis vectors that represents the contour space and facilitates the computation of tree variations. Concerning the tree appearance, we transfer color (and opacity values) between tree images through the use of Mean Value Coordinates [HF06] computed over simplified contours. We also present a fast and efficient method to segment tree images into crown/trunk pixels, so that their respective contours and appearances are combined and transferred properly.

DOI: 10.2312/ceig.20171218

The main contributions of the paper are: (a) a completely-automatic pipeline for creating tree variations from existing RGBA images of trees; (b) a robust method to extract the overall shape of a tree, including the automatic trunk segmentation; (c) the encoding of the overall external contour of a tree as a fixed-length high-dimensional vector; arbitrary contours are supported, not being limited to convex nor star-shaped polygons; highly fractal contours are supported; (d) the representation of the contour space as a relatively small set of basis vectors, in the spirit of EigenFaces [KBG11]; (e) the synthesis of random contours from convex combinations of contours from the library; and (f) methods for transferring color and opacity values from one or more source exemplars to a target image; the use of Mean Value Coordinates along with pinned contour points found through trunk segmentation avoids excessive area distortion during color transfer.

The rest of the paper is organized as follows. Section 2 reviews relevant previous work on tree creation through procedural, inverse procedural, and contour-based techniques. Section 3 provides an overview of our preprocessing and synthesis algorithms, which are further developed in Sections 4 and 5. Results with a large collection of tree exemplars are presented in Section 6. Conclusions are presented in Section 7.

2. Previous work

2.1. Procedural generation

The problem of generating plausible vegetation has received constant attention in computer graphics research. The first techniques used to model trees were based on procedural generation due to the lack of proper scanning devices. These models can be used trivially to create image-based tree representations. Honda introduced in [Hon71] a model that focused on the effect of certain parameters (such as branching angle and branch length) on tree shapes. These findings were applied to recursive algorithms which produced the first tree structures resembling their real counterparts [AK84]. Additional research was devoted to transform the generated branching hierarchy into a 3D model. Bloomenthal et al. [Blo85] examined the transformation process proposing techniques to represent the trunk, branches, and bark of a tree more faithfully.

Another set of techniques were introduced by Lindenmayer [Lin68] that, exploiting the capabilities of formal languages, managed to imitate plant development. L-systems have been widely used for modeling all types of plants and have been extended to support most of its peculiarities. These extensions include the integration of production rules and differential equations to represent the development of the plant over time [PHM93], the interaction with the environment [MP96], the expression of plant attributes based on their spatial location [PMKL01], and many others [PL96].

A different possibility is to exploit the factors that influence the final shape of a tree to be able to generate several species from a relatively small set of parameters. In particular, competition for resources (sunlight, space) by different branches of a tree seems to be critical for the general shape of temperate-climate trees. The space colonization algorithm presented by Runions et al [RLP07] uses this fact. In order to represent how branches compete for space first it generates a set of attractors in the volume defined by the tree's foliage. These attractors are then iteratively conquered by the branches as they occupy the available space. Palubicki et al. [PHL*09] extended this algorithm using a signaling mechanism to mimic different types of growth. In [XM15] Xu and Mould improved the performance of tree generation algorithms based on space competition by using pregenerated guiding vector fields and Yao graphs.

Kohek and Strnad [KS15] on the other hand used competition for incoming light as the main competition resource. They computed on the GPU how the shadows projected by the growing branches propagated through a regular grid. Then branches could grow on directions that maximized the amount of received light while the shadow grid was being updated.

Wind and surrounding space also influence a tree's growth. Pirk et al. [PSK*12] presented a technique that made it possible for content creators to change a tree's position inside a scene and see its shape adapt to changes in light distribution and the occupation of surrounding space. In [PNH*14] it was extended to include the effect of wind.

2.2. Tree generation from contours

Generating a tree from its contour is already possible applying the space colonization algorithm [RLP07], but there are other contributions that compute a tree from its silhouette. It is possible to provide the overall crown shape using a gesture based system that guides the resulting tree's growth [LRBP12], reducing modeling time while maintaining the artist's ability to obtain the desired result.

Okabe et al [OOI05] were able to generate 3D tree models from 2D sketches by taking the assumption that trees spread their branches far apart from each other. Such a system also allows users to apply gesture-based editing operations and manually generate trees from given examples. Wither et al. [WBCG09] made use of successive silhouettes sketched at different zoom levels to create a 3D tree.

Other techniques generate trees from a single photograph. These algorithms extract the silhouette of the crown and use it to generate the crown and its foliage. In [TFX*08] the user draws strokes in the photograph to identify the crown and the branches. The crown is segmented and the visible branches are converted to 3D using the approach proposed in [OOI05]. The initial skeleton is extended into the crown by iteratively substituting an existing branch by a subtree from a database. Guenard et al. [GMBC13], on the other hand, extract the foliage and compute a vector field from the segmented shape that is used to obtain a skeleton. This base skeleton is populated by leaves and branchlets via an L-system.

For dense trees it is possible to extract more compact representations. Argudo et al [ACA16] present a complete pipeline for synthesizing and rendering detailed trees with minimal effort. A rough estimate of the crown shape is created by solving a thin-plate energy minimization problem. Then detail is added through a simplified shape-from-shading approach. The final models are encoded into radial relief maps that may be rendered directly or used to populate the crown with billboards or a branch skeleton.

All these approaches use an input contour provided by the user or extracted from a photograph, which they preserve. Thus the silhouette of the generated trees is always the same.

2.3. Inverse procedural modeling

One way of generating variations from a single tree could be to identify the input parameters of a given procedural modeling algorithm, then alter those parameters slightly. Despite the difficulty of predicting the outcome produced by procedural modeling grammars, it is possible to control the process and produce the desired output. One possibility is to let the user explore the space of all possible models while guiding this search [TGY*09]. In order to help users explore the space of possible models it is interesting to provide a relatively small set of parameters that control the generation. These may be provided by the procedural algorithm designer but they may also be obtained semi-automatically. In [YAMK15] Yumer et al. use autoencoder networks to find a lower dimensional space that can be used as the set of parameters for users to play with. A weak point of these systems is that the resulting trees follow the intention of the artist more closely but the cost of generating a great quantity of similar models is very high.

Consequently, to automate the process Talton et al. [TLL*11] presented an approach using Markov chain Montecarlo, but its convergence is affected by the fact that the algorithm receives its feedback from completely-generated models only. Ritchie et al [RMGH15] improved upon it by developing a new sequential Montecarlo algorithm capable of using incremental feedback provided by incomplete models. Stava et al [SPK*14] developed an inverse procedural modeling algorithm specifically tailored for trees. An input tree is used to estimate the parameters of a recursive algorithm similar to that of [AK84], then new trees resembling the input one may be generated. Still, the needed optimization step is quite expensive.

2.4. Dimensionality reduction for variation synthesis

The main disadvantage of inverse procedural modeling techniques is the computation time required to extract the parameters of an input tree. Instead, our approach finds a base of vectors that represents the tree contour space. In this our algorithm is similar to how Eigenfaces represent faces for their recognition. The Eigenfaces algorithm [KBG11] reduces the high-dimensional space of all face images to a subset of the eigenvectors (called eigenfaces) of the associated covariance matrix. Any face can then be projected onto the span generated by the set of eigenfaces and thus be represented as a linear combination.

A similar method is used by Blanz et al. [BBPV03] to reanimate faces from input images. The animations are transferred by defining a vector space of 3D shapes and textures. Given the complexity of computing an eigendecomposition of a large set of exemplar 3D scans, these models are used as the base for the vector space of faces. The consequence is that the exemplars have to be chosen carefully so that they are representative of any faces we wish to reanimate.

As far as we can tell our contribution is the first one that applies this approach to the characterization of tree shapes. An advantage of its application to tree contours is the fact that the span generated by the computed eigenvectors always results in plausible tree shapes. The same cannot be said about its use for face representation. Many of the linear combinations of the eigenfaces result in invalid faces. This allows us to apply this algorithm to generate variations from a single photograph of a tree.

3. Overview

We assume a *library* $\mathcal{L}$ of tree images (RGBA side views of trees) is available. Tree images from the library (or a user-defined subset of it) will be referred to as *exemplars*. Our approach has two main stages: exemplar processing and tree synthesis.

An overview of the exemplar processing stage is shown in Figure 1. We start by resizing all exemplars to a fixed resolution (Section 4.1). A transparent border is added if necessary to preserve the aspect ratio. Each non-transparent exemplar pixel is then segmented into crown and trunk classes using a trained Convolutional Neural Network (Section 4.3). We then extract all contours of the alpha channel using a border following algorithm, and take the largest external contour as the overall tree contour C (Section 4.2).

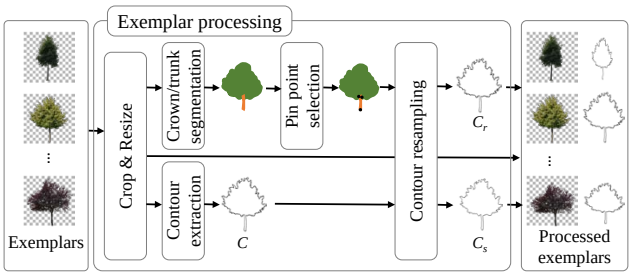

Figure 1: *Overview of the exemplar processing algorithm. Given a collection of RGBA images of trees, we generate fixed-length contours representing the overall shape of the trees.*

Then we re-sample C to create a new contour C_r with a fixed number N of 2D points (Section 4.5). The resampling uses three pin points (tree bottom, trunk top-left, trunk top-right) that are computed from the intersection of the tree contour with the segmented crown/trunk components (Section 4.4).

We then concatenate the (mean-subtracted) (x,y) coordinates of the N contour points onto a $2N$ vector $\boldsymbol{u}$ that represents the overall tree shape in $\mathbb{R}^{2N}$ space. The resampling above guarantees that a fixed range of $\boldsymbol{u}$ components correspond to crown contour points and another fixed range of $\boldsymbol{u}$ components to left/right trunk contour points. Although we could use PCA to extract a suitable base for the contour space, in practice we just take $\boldsymbol{U} = \{\boldsymbol{u_0}, \ldots \boldsymbol{u_m}\}$ as such a base. We then re-sample again the contour C_r (Section 4.6) to obtain a simplified contour C_s that will be used later to encode interior points of the tree as Mean Value Coordinates.

The output of the preprocessing step is, for each tree exemplar, a couple of contours C_r and C_s, along with a binary mask representing the crown/trunk segmentation.

Regarding tree image synthesis, we propose two variants. The

first one creates tree variations from scratch, using exemplars from the library. A new overall contour is created by computing a random convex combination of $\boldsymbol{u} \in \boldsymbol{U}$ vectors (Section 5.1). This contour already defines a preliminary segmentation of the output alpha channel. Then, we transfer the RGBA color from some exemplars to the target tree, using Mean Value Coordinates (Section 5.2). This allows us to associate each point of the target image to a matching point on the source(s) images from which we will compute the final RGBA color. Since we also transfer alpha values, the final shape of the tree is richer than the original contour.

The second synthesis variant requires the user to specify a tree image T. This image will be combined with features from the library to create variations of T. In this case, we apply to T all the processing steps we apply to exemplars, and then apply a similar synthesis procedure but giving higher weights to T features.

4. Exemplar Processing

We now describe the preprocessing steps to be performed for each exemplar RGBA image from the library. The main outcome of these steps is a suitable encoding of the overall tree shape through a fixed-length contour.

4.1. Image normalization

We start by normalizing the images so that all exemplars have the same size. This normalization is beneficial for subsequent steps, specially for the segmentation through FCN. In particular, we perform the following normalization steps. We first compute a binary version of the alpha channel through alpha thresholding, and set all transparent pixels to black to simplify the classifier task. We then crop the image to the minimum axis-aligned rectangle that encloses all opaque pixels, and add a padding black (and transparent) border to make the image square without distorting the tree. Finally, we resize the image to a fixed size (we used 1024×1024 pixel images for the experiments). Figure 8 shows the normalized versions of a collection of exemplars.

4.2. Contour extraction

We extract the overall (external) tree contour from the alpha channel A. We first threshold the alpha channel (we used 0.5 as threshold, assuming normalized values) to get a binary alpha mask A_t. Then we apply a border-following algorithm [S*85] to A_t to extract all the contours separating opaque regions from transparent ones. Each contour is represented as a collection of 2D point coordinates.

Figure 2 shows the contours extracted from some exemplars. Typical tree images include multiple contours; trees with sparse foliage have multiple see-through parts (holes in A_t) and even multiple connected components (due e.g. to thin branches not appearing in A_t).

We classify the extracted contours as exterior (not inside any other contour) and interior (inside another enclosing contour). We take as the overall contour the longest exterior contour C (Figure 2). Notice that $C = \{(x_i, y_i)\}$ will have a variable number of points depending on, among other factors, the fractal nature of the tree silhouette. We guarantee that external contour vertices are given in counter-clockwise order.

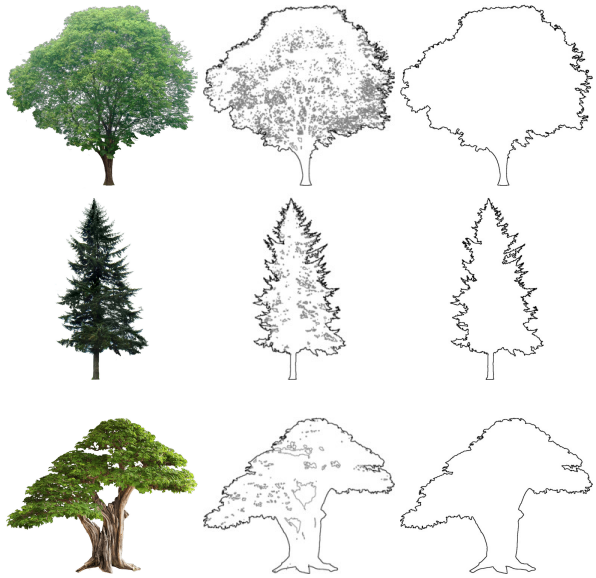

Figure 2: *Some RGBA images (left), all contours extracted from their alpha channel (middle), and longest external contour (right). Internal contours are shown in light gray, and external contours in black.*

4.3. Crown/Trunk segmentation

The contour contains both the tree crown as well as the trunk. For better results, we would like to have approximately the same number of contour points belonging to the trunk independently of how big each trunk is. Therefore, we need to segment the crown from the trunk.

Several image segmentation approaches exist. In the past few years, Deep Learning approaches, in particular Convolutional Neural Networks, have been shown to offer remarkable results for image classification tasks. Long et al. [LSD15] explain how classification networks can be converted into fully convolutional networks (FCNs) such that a per-pixel segmentation can be learned end-to-end. They extend various networks into their respective fully convolutional form. We decided to use the network they refer to as FCN-8s-atonce, which they obtain by extending into a FCN and training the VGG16 classification net [SZ14] - which, in turn, had been trained using the ImageNet database. The authors provide the network implementation and weights for the Deep Learning framework Caffe [JSD*14]. We downloaded it, modified the output layer to produce three classes - background, crown and trunk - and fine-tuned it.

From our tree dataset, we segmented manually 55 exemplars (Figure 9), trained the net for 200 epochs (21 s/epoch), and obtained around 95% accuracy and 86% mean IoU (Intersection over Union) on the same training set. Since our number of exemplars was lim-

ited, we did not split them into train and validation sets. Moreover, we expect new exemplars to be very similar to those already provided, and obtaining a rough segmentation suffices for our needs as we shall see in the next section. Figure 3 shows the segmentation output of our net on new inputs not seen during the training phase. Segmenting each of these 1024×1024 images takes on average 15.5 s. Both training and segmentation times could be improved by leveraging the batch input capabilities of the network, as shown in [LSD15].

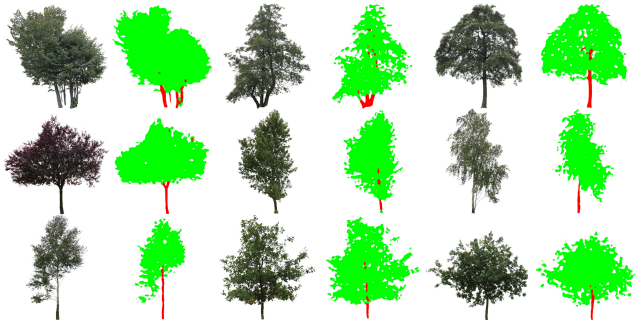

Figure 3: *Trees segmented automatically using our FCN.*

4.4. Pin point selection

The creation of new contours through linear combinations of existing contours requires the definition of a minimum set of matching points across all contours, so that e.g. the first contour point always refers to the point at the tree bottom.

For each exemplar, we select three pin points on the extracted contour C: tree bottom (B), trunk top-left (L) and trunk top-right (R), see Figure 4. The tree bottom is set to the index of the point with minimum y. If multiple points share such a minimum, we set B to the index of the median point. For the trunk L and R points, we traverse the points in C, starting from B, in both forward and backward directions. We stop the traversal as soon as the current contour segment intersects the crown baseline, i.e. the lowest height on the segmented crown. In the rare case that the segmented image contains no trunk pixels, we set L=R=B.

4.5. Contour resampling

Extracted contours C are variable-length and thus not suitable for generating variations through linear combinations of exemplars. We thus resample the overall contour C of each exemplar to include exactly N points. Our resampling strategy considers three different segments on C: the crown segment (R to L), the left truck segment (L to B) and the right trunk segment (B to R). Each segment is assigned a fixed number of samples in the resampled contour C_r. We used N =2,000 points, allocating 1,600, 200 and 200 points for each of the three segments above. Resampling within each segment is performed as in chord-length parameterization, i.e. attempting to generate uniform chord lengths between samples. The output contour C_r has thus N points, all of them uniformly distributed (in a chord-length sense) within each segment. Figure 4 shows the resampled contours (in cyan) for a few exemplars.

4.6. Contour simplification

The resampled contour C_r is detailed enough to be used for contour synthesis, but too complex for the generation of Mean Value Coordinates. We thus further resample C_r to $M = 100$ points to generate a simpler contour C_s. We observed that the distortions on the color image produced by the Mean Value Coordinates after warping the contour are more acute for those pixels near the contour or outside of it. Therefore, we actually generate C_s by first rendering the mask of the interior of C_r, dilating this mask for some iterations (8 in our tests) and resampling to M points the contour of this mask.

Figure 4 shows the simplified contours (in red) for a few exemplars.

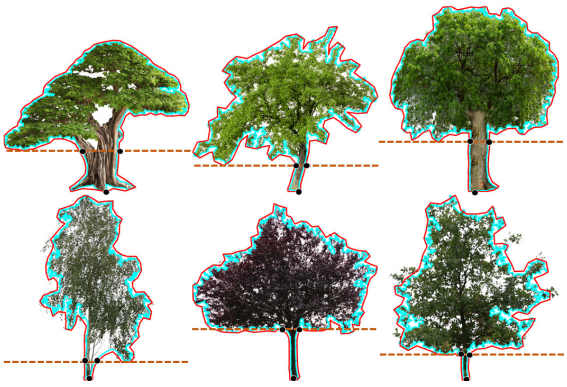

Figure 4: *Resampled contour C_r (in cyan), simplified contour C_s (in red), crown baseline (in orange) and pin points (in black). Top row: manually segmented trees, bottom row: automatically segmented trees.*

4.7. Preprocessing output

The output of the above processing steps is a collection of processed exemplars along with their resampled C_r and simplified contours C_s.

5. Tree Synthesis

We now discuss different strategies to generate new tree images through random combinations of exemplars. We first define the overall tree shape by synthesizing a new contour through linear combinations of contours (Section 5.1). Then the contour is filled by transferring RGBA color from the exemplars (Section 5.2).

5.1. Contour synthesis

Let $\boldsymbol{u_j}$ be the vector $\in \mathbb{R}^{2N}$ that results from flattening the (x,y) coordinates of the resampled contour C_r from the j-th exemplar.

We can linearly interpolate two contours $\boldsymbol{u'} = (1-t)\boldsymbol{u_0} + t\boldsymbol{u_1}$ with $t \in [0,1]$ to produce a continuous morphing between them. Figure 5 shows several snapshots of the interpolation between two contours. The quality of the interpolated contours is highly dependent on the matching contour points; the use of the B, L, R pin points prevents excessive rotations during morphing.

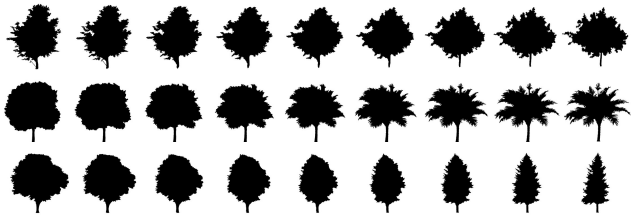

Figure 5: *Morphing two contours through convex combinations.*

We can extend this idea to incorporate additional contours through convex combinations of existing contours, i.e., $\boldsymbol{u'} = \sum w_i \boldsymbol{u_i}$, with $w_i \geq 0$ and $\sum w_i = 1$. We avoid negative weights to prevent contours from being reflected (e.g. $-\boldsymbol{u_i}$ would result in an upside-down contour).

The generated contour provides a preliminary version of the output alpha channel (to be refined later, see next section). We do this by simply drawing the contour onto a blank alpha channel (with alpha set to 1.0 for all pixels), and then using a region fill algorithm from any seed outside the contour to clear the alpha values of the pixels outside the contour. This method is robust against potential self-intersections of the combined contours.

5.2. Color transfer

In the previous subsections we generated new contours C'_r and their associated alpha masks. We now explain how to fill non-transparent pixels of the output image E' with color.

We address this problem by transferring color from one or more exemplars (*source* images) to the image being synthesized (*target*). We pose this color transfer problem as an *image warping* problem. Let $w \times h$ be the resolution of the (processed) exemplar images, and let Ω be their $w \times h$ rectangular domain. Given a source contour C_s and a target contour C'_s, both with vertices in Ω, we aim at defining a smooth warp function $f : \Omega \mapsto \Omega$ mapping each vertex $(x_i, y_i) \in C_s$ to the corresponding vertex $(x'_i, y'_i) \in C'_s$. Such a warping function can be used to deform any source image E defined on Ω to a target image E' by simply letting $E' = E \circ f^{-1}$ [HF06].

We define the mapping above through barycentric coordinates with respect to (a simplified version of) the source and target contours. In particular, we use mean value coordinates [HF06], which are well-defined for arbitrary planar polygons.

When transferring RGBA color from a single source exemplar E, the algorithm proceeds as follows. For each non-transparent pixel $p' = (x', y')$ of the target image E', we first compute the mean value coordinates λ'_i of p' with respect to the target contour C'_s. We then find the corresponding point on the source image, $p = f^{-1}(p')$, by simply using the resulting coordinates λ'_i with the vertices $\{v_i\}$ of the chosen source contour C_s, i.e. $p = \sum \lambda'_i v_i$. The final RGBA color for pixel p' is just $E(p)$. As in [HF06], color sampling can be improved through bilinear interpolation on the 2×2 grid of pixels surrounding each source pixel.

The color transfer approach above can be extended to take colors from multiple exemplars. Let (c_j, a_j) be the RGB and A components of the color extracted (through mean value coordinates) from j-th exemplar. We compute the output alpha value as $a' = \max a_j$, i.e. the final pixel will take the highest opacity from the source pixels. We do this to avoid an excessive amount of transparent pixels to be transferred to the target image. The color is computed as a random convex combination of the colors, but considering only those colors with non-null opacity values.

Figure 6 shows the morphing of two exemplars using the RGBA color transfer to fill the interpolated contours.

Figure 6: *Morphing two trees through RGBA color transfer.*

5.3. Histogram transfer

As a result of the previous operations we have been able to generate new trees. We can add more variation by changing the color histogram of the generated image. Since we want the result to be plausible, we use a histogram transfer algorithm [GW07] so that the image we have generated has the same color distribution as another image provided as a reference.

In order to do this we compute the cumulative histograms for both images (source image and template image). We then interpolate linearly to find the unique pixel values in the template image that most closely match the quantiles of the unique pixel values in the source image. This process is performed for each RGB color channel separately and it always ignores transparent pixels.

Histogram matching is also useful both to improve the matching between the combined images, since they can be taken in varying lighting conditions, as well as to simulate the change of vegetation coloration during different seasons.

Figure 7 shows an RGB transition using histogram matching.

Figure 7: *Morphing two trees through RGBA color transfer; with histogram transfer (top) and without (bottom).*

6. Results

6.1. Test dataset

We tested the preprocessing and synthesis steps using as tree library a collection of 55 RGBA images from different sources, including the VTP Plant Library. The corresponding normalized exemplars are shown in Figure 8.

Figure 8: *Tree images used as exemplars.*

6.2. Segmentations

Figure 9 shows the result of segmenting crown and trunk pixels on the test exemplars, using our fine-tuned FCN. Notice that the trunk was correctly segmented from the crown in all images where the trunk was visible. Only two images had no visible trunk. For these images, pin points L and R were set to B.

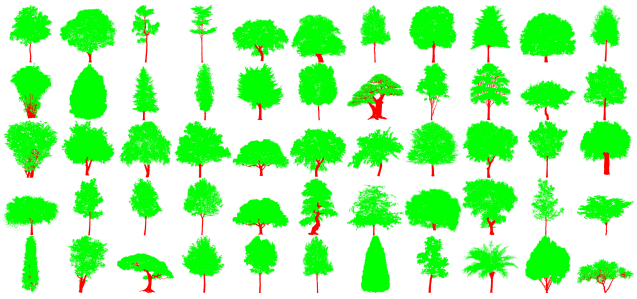

Figure 9: *Manual segmentations for the exemplars in Figure 8.*

6.3. Extracted contours

Figure 10 shows the thresholded alpha channels from the input exemplars, and Figure 11 shows all the contours detected by the border following algorithm. The number of contours varied from 1 (trees with dense foliage) up to 5,481 (very sparse tree). We use only the longest external contour C. The resulting alpha masks are shown in Figure 12. Notice that these alpha masks will be later refined during RGBA transfer.

As expected, major differences between the initial alpha mask and the one bounded by C correspond to see-through parts of the tree due to sparse foliage or empty space between main branches. Differences around the silhouette mostly depend on the alpha threshold value. We also explored the application of a few iterations of a morphological closing operation to the thresholded alpha mask, before contour extraction. This obviously tends to minimize the number of connected components in the mask, but at the expense of some loss of high-frequency details on the tree silhouette. We thus applied no closing iterations to our examples.

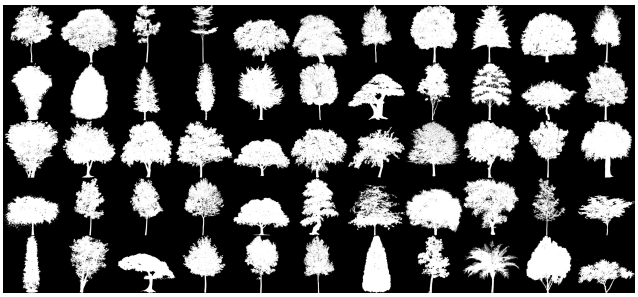

Figure 10: *Thresholded (0.5) alpha channels of the test exemplars.*

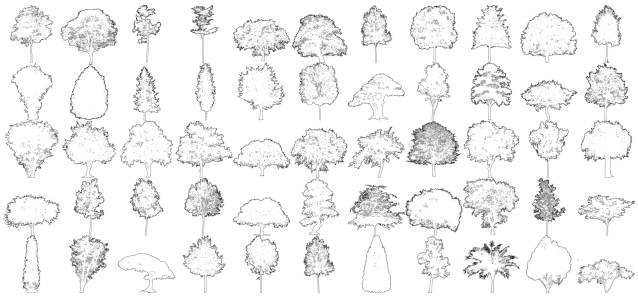

Figure 11: *All contours extracted from the alpha channel. External contours are shown in black and internal contours are shown in gray.*

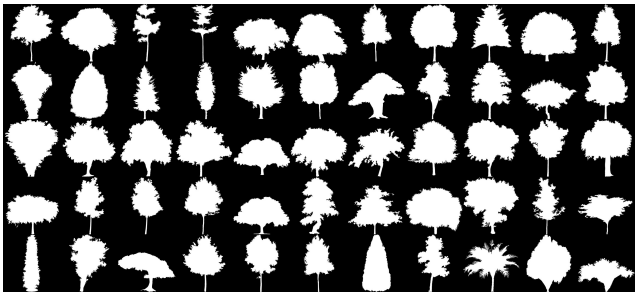

Figure 12: *Alpha channels defined by the largest external contour.*

6.4. Synthetic contours

Figure 13 shows some convex combinations of multiple contour pairs. Due to space limitations, here we only show convex combinations with weights $w_1 = 1/3$, $w_2 = 2/3$ (upper triangle of the table) and $w_1 = 2/3$, $w_2 = 1/3$ (lower triangular part). Despite trying to combine radically-different tree species, most combinations look plausible.

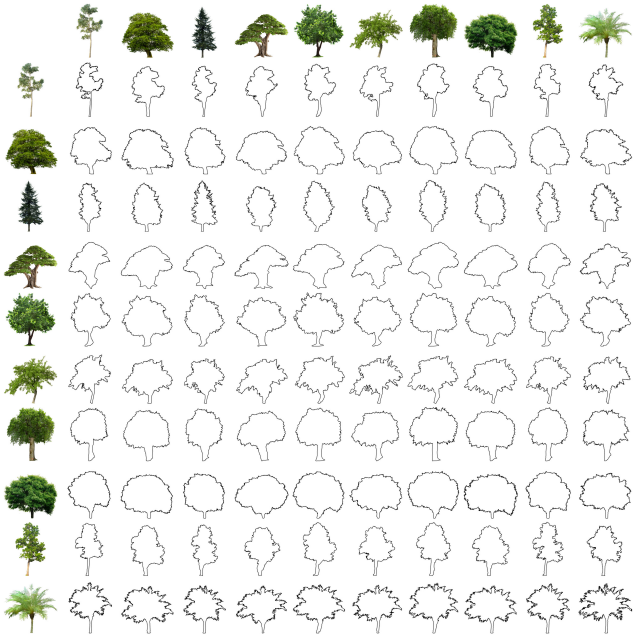

Figure 13: *Contour combination table. For each cell, the contour has been obtained as a combination of the row and column exemplars contours, with corresponding weights* $2/3$ *and* $1/3$.

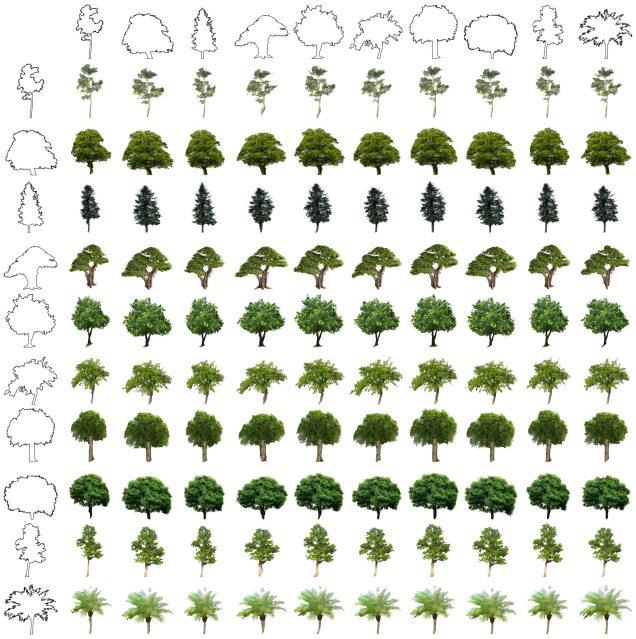

Figure 14: *Tree deformations table. For each cell, the tree image has been distorted to match the shape of the combined contours of Figure 13.*

6.5. Synthetic trees

Figure 14 shows some combinations of multiple tree pairs, distorting the image of each row towards the shape of the image in each column. Again, due to space limitations, here we only show convex combinations with weights $1/3$, $2/3$.

Figure 16 shows more examples obtained by combining two randomly selected exemplars, using a convex combination of the contour $w_1 \in [0.3, 0.7]$ and $w_2 = 1 - w_1$, and setting the color as either the convex combination with the same weights as the contour, or sampling one of the two images directly. Generating each of these new trees at 1024×1024 resolution takes between 8 and 12 seconds, mainly depending on the ratio of non-transparent pixels.

Although the synthetic tree images we create are not necessarily plausible from a botanical point-of-view (specially when combining exemplars from radically different tree species), these images are still suitable for mainstream applications such as video games, where indeed artists often look for fictional trees.

Limitations The color transfer approach works best when the two contours are not radically different. Otherwise, the source image needs to be severely distorted to fit the target contour, as shown in Figure 15.

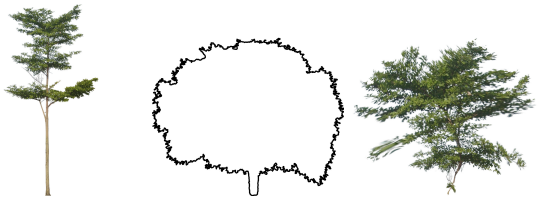

Figure 15: *The thin, sparse tree on the left has been used to fill the thick contour on the middle, obviously resulting in a large distortion.*

7. Conclusions and future work

We have presented an algorithm for the automatic generation of tree variations starting from a collection of example images. The set of exemplars is first used to train a neural network for pixel classification of tree images into canopy and trunk pixels. This allows us to extract pinpoints for the main sections of the tree contour, which facilitates the synthesis of new ones from the exemplars. Synthesis of new tree images is handled by first creating their overall external contour as a random convex combination of existing contours, and then transferring the color from other exemplars through mean value coordinates. The tree variations we generate can be used to author fictional tree images and hybrid speciments, as well as to create variations to prevent tree copies from being discovered by users.

Regarding contour sampling, the pin points we currently compute may introduce some distortion in the results. It would be interesting to research how to perform this sampling so that areas and angles are preserved as well as possible. In addition, Mean Value Coordinates allow multiple polygons as long as the source and target polygon-sets are topologically equivalent. Making use of this it should be possible to extract multiple contours instead of a single outer contour.

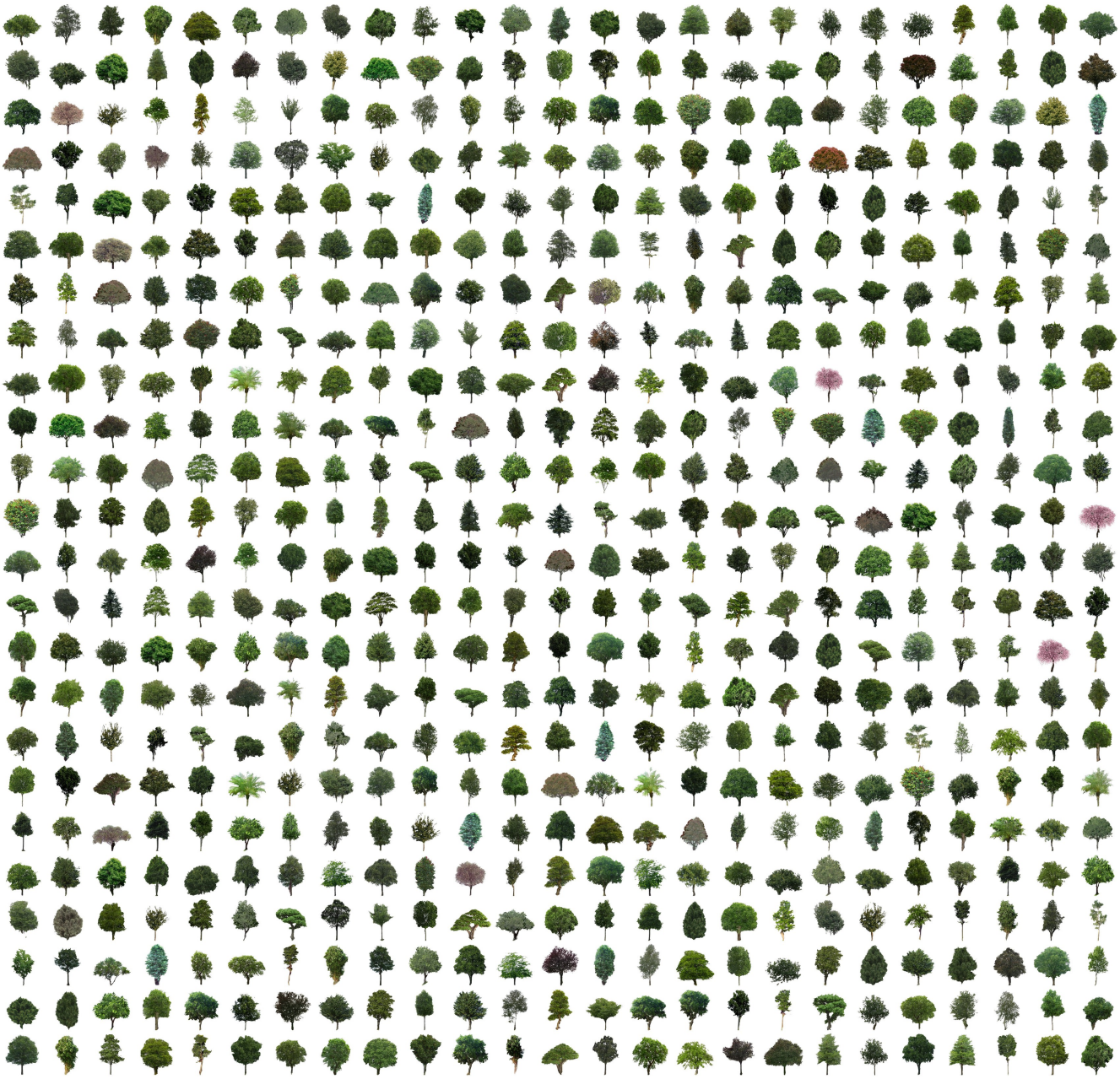

Figure 16: *Randomly generated tree variations from the test dataset.*

Another important point is the fact that the provided exemplars may be radically different. While the species are similar (e.g. similar temperate-climate trees) trees match and the distortions are tolerable. This leaves the user with the job of avoiding combinations between completely incompatible trees. A clear improvement of the proposed algorithm would be to compute which subset of trees are sufficiently compatible for their combination, storing this information in a graph. The generation of variations could then be guided by it. Finally, since there exist multiple barycentric coordinates generalizations, we would be interested in analyzing the effect of these on the color transfer step.

Acknowledgments

This work has been partially funded by the Spanish Ministry of Economy and Competitiveness and FEDER under grant TIN2014-52211-C2-1-R, and the Spanish Ministry of Education, Culture and Sports PhD grant FPU13/01079.

References

[ACA16] ARGUDO O., CHICA A., ANDÚJAR C.: Single-picture reconstruction and rendering of trees for plausible vegetation synthesis. *Computers & Graphics 57* (2016), 55–67. 1, 2

[ACV*14] ANDÚJAR C., CHICA A., VICO M. A., MOYA S., BRUNET P.: Inexpensive reconstruction and rendering of realistic roadside landscapes. *Computer Graphics Forum 33*, 6 (2014), 101–117. 1

[AK84] AONO M., KUNII T.: Botanical tree image generation. *IEEE Comput. Graph. Appl. 4*, 5 (May 1984), 10–34. 2, 3

[BBPV03] BLANZ V., BASSO C., POGGIO T., VETTER T.: Reanimating faces in images and video. In *Computer graphics forum* (2003), vol. 22(3), pp. 641–650. 3

[BCF*05] BEHRENDT S., COLDITZ C., FRANZKE O., KOPF J., DEUSSEN O.: Realistic real-time rendering of landscapes using billboard clouds. *Computer Graphics Forum (Proceedings of EUROGRAPHICS 2005), 24(3)* (2005). 1

[Blo85] BLOOMENTHAL J.: Modeling the mighty maple. *SIGGRAPH Comput. Graph. 19*, 3 (July 1985), 305–311. 2

[DCSD02] DEUSSEN O., COLDITZ C., STAMMINGER M., DRETTAKIS G.: Interactive visualization of complex plant ecosystems. In *Proceedings of the Conference on Visualization '02* (Washington, DC, USA, 2002), VIS '02, IEEE Computer Society, pp. 219–226. 1

[DN04] DECAUDIN P., NEYRET F.: Rendering forest scenes in real-time. In *Rendering Techniques* (2004), Keller A., Jensen H. W., (Eds.), Eurographics Association, pp. 93–102. 1

[GMBC13] GUÉNARD J., MORIN G., BOUDON F., CHARVILLAT V.: Reconstructing plants in 3d from a single image using analysis-by-synthesis. In *Advances in Visual Computing*. Springer, 2013, pp. 322–332. 2

[GW07] GONZALEZ R. C., WOODS R. E.: *Digital Image Processing (3rd Edition)*. Pearson, 2007. 6

[HF06] HORMANN K., FLOATER M. S.: Mean value coordinates for arbitrary planar polygons. *ACM Trans. Graph. 25*, 4 (Oct. 2006), 1424–1441. 1, 6

[Hon71] HONDA H.: Description of the form of trees by the parameters of the tree-like body: Effects of the branching angle and the branch length on the shape of the tree-like body. *Journal of Theoretical Biology 31*, 2 (1971), 331 – 338. 2

[JSD*14] JIA Y., SHELHAMER E., DONAHUE J., KARAYEV S., LONG J., GIRSHICK R., GUADARRAMA S., DARRELL T.: Caffe: Convolutional architecture for fast feature embedding. *arXiv:1408.5093* (2014). 4

[KBG11] KSHIRSAGAR V. P., BAVISKAR M. R., GAIKWAD M. E.: Face recognition using eigenfaces. In *2011 3rd International Conference on Computer Research and Development* (2011), vol. 2, pp. 302–306. 2, 3

[KS15] KOHEK Š., STRNAD D.: Interactive synthesis of self-organizing tree models on the gpu. *Computing 97*, 2 (2015), 145–169. 2

[Lin68] LINDENMAYER A.: Mathematical models for cellular interactions in development: Parts i and ii. *Journal of theoretical biology 18*, 3 (1968), 280–315. 2

[LRBP12] LONGAY S., RUNIONS A., BOUDON F., PRUSINKIEWICZ P.: Treesketch: Interactive procedural modeling of trees on a tablet. In *Proceedings of the International Symposium on Sketch-Based Interfaces and Modeling* (Aire-la-Ville, Switzerland, Switzerland, 2012), SBIM '12, Eurographics Association, pp. 107–120. 2

[LSD15] LONG J., SHELHAMER E., DARRELL T.: Fully convolutional networks for semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2015). 4, 5

[MP96] MĚCH R., PRUSINKIEWICZ P.: Visual models of plants interacting with their environment. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1996), SIGGRAPH '96, ACM, pp. 397–410. 2

[OOI05] OKABE M., OWADA S., IGARASHI T.: Interactive Design of Botanical Trees using Freehand Sketches and Example-based Editing. *Computer Graphics Forum 24*, 3 (2005), 487–496. 2

[PHL*09] PALUBICKI W., HOREL K., LONGAY S., RUNIONS A., LANE B., MĚCH R., PRUSINKIEWICZ P.: Self-organizing tree models for image synthesis. In *ACM SIGGRAPH 2009 Papers* (New York, NY, USA, 2009), SIGGRAPH '09, ACM, pp. 58:1–58:10. 2

[PHM93] PRUSINKIEWICZ P., HAMMEL M. S., MJOLSNESS E.: Animation of plant development. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1993), SIGGRAPH '93, ACM, pp. 351–360. 2

[PL96] PRUSINKIEWICZ P., LINDENMAYER A.: *The Algorithmic Beauty of Plants*. Springer-Verlag New York, Inc., 1996. 2

[PMKL01] PRUSINKIEWICZ P., MÜNDERMANN L., KARWOWSKI R., LANE B.: The use of positional information in the modeling of plants. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2001), SIGGRAPH '01, ACM, pp. 289–300. 2

[PNH*14] PIRK S., NIESE T., HÄDRICH T., BENES B., DEUSSEN O.: Windy trees: Computing stress response for developmental tree models. *ACM Trans. Graph. 33*, 6 (Nov. 2014), 204:1–204:11. 2

[PSK*12] PIRK S., STAVA O., KRATT J., SAID M. A. M., NEUBERT B., MĚCH R., BENES B., DEUSSEN O.: Plastic trees: Interactive self-adapting botanical tree models. *ACM Trans. Graph. 31*, 4 (July 2012), 50:1–50:10. 2

[RLP07] RUNIONS A., LANE B., PRUSINKIEWICZ P.: Modeling Trees with a Space Colonization Algorithm. In *Proceedings of the Third Eurographics Conference on Natural Phenomena* (Aire-la-Ville, Switzerland, Switzerland, 2007), NPH'07, Eurographics Association, pp. 63–70. 2

[RMGH15] RITCHIE D., MILDENHALL B., GOODMAN N. D., HANRAHAN P.: Controlling procedural modeling programs with stochastically-ordered sequential monte carlo. *ACM Transactions on Graphics (TOG) 34*, 4 (2015), 105. 3

[S*85] SUZUKI S., ET AL.: Topological structural analysis of digitized binary images by border following. *Computer vision, graphics, and image processing 30(1)* (1985), 32–46. 4

[SPK*14] STAVA O., PIRK S., KRATT J., CHEN B., MĚCH R., DEUSSEN O., BENES B.: Inverse procedural modeling of trees. *Computer Graphics Forum 33*, 6 (2014), 118–131. 3

[SZ14] SIMONYAN K., ZISSERMAN A.: Very deep convolutional networks for large-scale image recognition. *CoRR abs/1409.1556* (2014). 4

[TFX*08] TAN P., FANG T., XIAO J., ZHAO P., QUAN L.: Single Image Tree Modeling. *ACM Transactions on Graphics 27*, 5 (Dec. 2008), 108:1–108:7. 2

[TGY*09] TALTON J. O., GIBSON D., YANG L., HANRAHAN P., KOLTUN V.: Exploratory modeling with collaborative design spaces. In *ACM SIGGRAPH Asia 2009 Papers* (2009), pp. 167:1–167:10. 3

[TLL*11] TALTON J. O., LOU Y., LESSER S., DUKE J., MĚCH R., KOLTUN V.: Metropolis procedural modeling. *ACM Trans. Graph. 30*, 2 (Apr. 2011), 11:1–11:14. 3

[WBCG09] WITHER J., BOUDON F., CANI M.-P., GODIN C.: Structure from silhouettes: a new paradigm for fast sketch-based design of trees. *Computer Graphics Forum 28*, 2 (2009), 541–550. 2

[XM15] XU L., MOULD D.: Procedural tree modeling with guiding vectors. *Computer Graphics Forum 34*, 7 (2015), 47–56. 2

[YAMK15] YUMER M. E., ASENTE P., MECH R., KARA L. B.: Procedural modeling using autoencoder networks. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology* (2015), pp. 109–118. 3