TreeSketchNet: From Sketch to 3D Tree Parameters Generation

GILDA MANFREDI, NICOLA CAPECE, UGO ERRA, and MONICA GRUOSSO, University of Basilicata, Italy

Fig. 1. Reconstructed 3D trees are shown as images captured from different camera angles. The image on the left shows a front-view maple tree reconstruction starting from a synthetically-generated (SG) input. The top row of the middle column shows input, reconstruction and Ground Truth (GT) for the left-camera view of a palm, starting from the SG input. The middle row of the same column shows a front-camera view of a cherry tree, starting from the SG input. The bottom row of the middle column shows a right-camera view reconstruction of a bonsai, starting from SG input. Finally, the image in the right-hand column shows a left-camera view reconstruction of a pine tree, starting from the SG input.

3D modeling of non-linear objects from stylized sketches is a challenge even for Computer Graphics (CG) experts. The extrapolation of object parameters from a stylized sketch is a very complex and cumbersome task. In the present study, we propose a broker system that can transform a stylized sketch of a tree into a complete 3D model by mediating between a modeler and a 3D modeling software. The input sketches do not need to be accurate or detailed: they must only contain a rudimentary outline of the tree that the modeler wishes to 3D-model. Our approach is based on a well-defined Deep Neural Network (DNN) architecture, called TreeSketchNet (TSN), based on convolutions and capable of generating Weber and Penn [1995] parameters from a simple sketch of a tree. These parameters are then interpreted by the modeling software, which generates the 3D model of the tree pictured in the sketch. The training dataset consists of Synthetically-Generated (SG) sketches that are associated with Weber-Penn parameters, generated by a dedicated Blender modeling software add-on. The accuracy of the proposed method is demonstrated by testing the TSN with synthetic and hand-made sketches.

*All authors contributed equally to this research.

Authors’ address: Gilda Manfredi, gilda.manfredi@unibas.it; Nicola Capece, nicola.capece@unibas.it; Ugo Erra, ugo.erra@unibas.it; Monica Gruosso, monica.gruosso@unibas.it, University of Basilicata, Via dell’Ateneo Lucano 10, Potenza, Italy, 85100.
Finally, we provide a qualitative analysis of our results, by evaluating the coherence of the predicted parameters with several distinguishing features.

CCS Concepts: Computing methodologies → 3D imaging; Mesh models.

Additional Key Words and Phrases: procedural modeling, 3D trees, 3D generation, datasets, neural networks, deep learning

1 INTRODUCTION

Manual mesh modeling of objects that are characterized by non-linear complex 3D structures remains a challenge even for experts in CG. Typically, objects such as trees are designed using procedural modeling [Xie et al. 2016], which allows users to manipulate the specific parameters [Deussen and Lintzmann 2005; Huang et al. 2017; Nishida et al. 2016] that characterize them, avoiding direct editing of their geometries. However, these objects are affected by complex rules characterized by a set of parameters not only very large but also non-linear. Drawing a 3D object is often easier and more intuitive than manually or parametrically modeling its geometry.

In the literature, there are several image-based approaches to reconstructing real objects as 3D models, e.g., those based on photogrammetry [Gatzios et al. 2015; Remondino et al. 2009] and new technologies such as Depth Cameras [Nguyen et al. 2018; Zhou and Koltun 2014], LiDAR [Hu et al. 2017; Tachella et al. 2019], Laser Scanning [Lu-Xingchang and Liu-Xianlin 2006], etc. Although there are several methods that use images to predict directly a coarse 3D mesh [Kanazawa et al. 2018; Lei et al. 2020; Pan et al. 2019; Wang et al. 2018d], many produce inaccurate results, with smoothed or very high poly meshes, wrongly-closed holes, non-manifold edges and vertices, and isolated vertices. To overcome these problems, several approaches aim to automatize procedural modeling using images for predicting parameters [Liu et al. 2021b; Smelik et al. 2014; Wang et al. 2018a]. Along the same lines as previous approaches, we introduce a broker system that sits between 3D modelers and the specialized 3D modeling software used to build 3D trees. Users only need to generate a hand-made sketch (HM) to be provided to the broker as input so that it predicts a set of parameters that can be read by the 3D modeling software and used to build a 3D tree that is as similar as possible to the sketch. Moreover, our broker can determine the appropriate texture for the rendering of the 3D tree. The key element in our broker is a convolution-based neural network that is able to learn parameter mappings and analyze the sketch based on a set of training data, having been trained using supervised learning paradigms. As collecting a sufficiently large number of HM sketches is a very expensive operation, we have developed a specific Blender add-on. This is called Render Tree (RT) and is based on the existing Blender Sapling Tree Gen1, which adopts the Weber and Penn [1995] procedural modeling methodology to create 3D trees from a set of parameters. The benefit of this approach is that it makes it possible to create a consistent training dataset of synthetically-generated (SG) sketches, as reported in Figure 1 (e.g., the maple tree). Although Weber and Penn is a fairly old model, its stability combined with the wide availability of implementation code make it a widely used method for the parametric generation of 3D models of plants and more [Delalieux et al. 2014; Frisk et al. 2022; Richter and Maas 2022; Rutzinger et al. 2010]. Our experiment evaluated five trees species: maple, pine, bonsai, palm, and cherry. Four camera views were gathered for each: front, back, left, and right. In the final step, we generated 250 randomly-controlled versions of each species of the tree by varying the input parameters to the RT Blender add-on. This resulted in 5000 SG. It should be noted that our dataset could be extended to include other species of trees.

The main contributions of our paper can be summarized as follows:

(1) An approach to quickly generate a training dataset consisting of synthetic and realistic sketches of 3D trees, starting from randomly-controlled Weber-Penn parameters.

(2) A specific DNN architecture with multiple outputs based on the set of parameter values.

1https://docs.blender.org/manual/en/latest/addons/add_curve/sapling.html
The RT Blender add-on and the trained DNN are available online and can be freely used for similar predictions: https://github.com/Unibas3D/TreeSketchNet.

2 RELATED WORK
In past years, the best way to reconstruct an object in 3D relied upon the manual skills of a human modeler, especially for objects with very complex geometries, such as trees. Later, the procedural modeling technique began to take hold. This technique facilitated the modeling phase by allowing the user to manipulate the values of a parameters series to which the object’s geometry was bound. However, procedural modeling requires a user training phase to understand how the object parameters work. In recent years, thanks to the rapid growth of artificial intelligence technologies, 3D modeling has broken new ground and has become accessible to non-expert modelers. In this section, we report the State-Of-The-Art (SOTA) in the domain of 3D mesh generation, starting from sketches or images.

2.1 3D mesh generation
Among the fastest methods for the generation of a 3D object, it is necessary to mention those that use new sensors, such as LiDAR or Depth Cameras, to reconstruct meshes through photogrammetry. In this context, Cheng et al. [2013] propose an approach for the automatic reconstruction of 3D roofs using airborne LiDAR data and optical multi-view aerial imagery. On the same line, Tachella et al. [2019] present a computational framework for real-time 3D scene reconstruction using single-photon LiDAR data, acquired in broad daylight from distances up to 320 meters. LiDAR point clouds can be also used to model plausible trees in fine-grained detail, as shown by Hu et al. [2017]. Indeed, they were able to reconstruct the tree skeleton by using the nearest neighbors algorithm on each point of a segmented point cloud, integrated with some trunk points. The branch thickness is determined by a pipe model while the leaves arrangement follows the botanical rules. Other interesting approaches use a tree point cloud, obtained by scanning a real tree, as input for a procedural modeling algorithm ([Guo et al. 2020b]) or for a DNN ([Liu et al. 2021a]) to reconstruct a realistic 3D tree geometry. As an alternative to LiDAR, there are many other studies incorporating a Kinect Depth Camera sensor ([Haque et al. 2014; Izadi et al. 2011]) or one depth camera and two mirrors ([Nguyen et al. 2018]) for 3D reconstruction. Another method for 3D generation is Laser scanning [Lu-Xingchang and Liu-Xianlin 2006]. An example is the work of Rutzinger et al. [2010], characterized by the detection of trees from mobile laser scanner point clouds and the modeling of a single tree. The aim of this is to model single trees in 3D city models for visualization purposes. In addition, Wang et al. [2018b] propose a data-driven mechanism to automatically create 3D trees from existing ones through structure and geometrical blending. The initial models can be created using existing 3D modeling tools or internet repositories. Quigley et al. [2021] try to reconstruct trees with geometrical and topological accuracy for physical simulation starting from RGB images to create point clouds, textured meshes and skinned cylindrical articulated rigid body models.

With respect to these approaches, we do not use ad-hoc devices such as the laser scanner or the LiDAR to create the input data. Indeed, these instruments often suffer from real-world scanned object problems due e.g., to high solar angles or huge reflections since the laser pulses depend on the reflection principle. One of the main differences concerns the amount of information that feeds our system. We use raw data (sketches) that contain less information than existing multiple RGB images, point clouds, or polygonal meshes.

More generally, as noted in Hu et al. [2017], 3D tree modeling techniques can be classified into procedural [Stava et al. 2014], sketch [Liu et al. 2010; Okabe et al. 2007], and image-based [Kim and Jeong 2014; Tan et al. 2007] methods.

Approaches for generating 3D trees [Palubicki et al. 2009] are often generalized to plant simulations and vegetation modeling. Hädrich et al. [2021] propose a method to capture the combustion process of plants. One
of their contributions is the wildfire simulation on more than 100K individual plants represented with detailed geometry. Similarly, Palubiki et al. [2022] present a method to simulate ectoclimates to model tree growth interactively based on temperature, light, and gradients of water. Plant growth simulation and visualization are also faced by Golla et al. [2020], which define a semantic segmentation-based method to allow temporal upsampling of acquired point cloud sequences of the growth process, including topological changes.

Although these methods provided impressive results in terms of quality and correctness, some of them are based on a self-organizing process to generate consistent trees and plants [Makowski et al. 2019]. Contrary to the related work, our approach considers SG sketches to create training and validation datasets and HM sketches for testing purposes.

2.2 From images to 3D reconstruction

Digital photogrammetry using RGB images is one of the most popular techniques used to reconstruct 3D models. However, software tools such as Reality Capture require a large amount of images to build a 3D reconstruction of a single real object in a well-defined method [ingsland 2020]. In particular, several extensive learning-based studies have attempted to generate 3D objects from simple single RGB views [Hane et al. 2017; Lei et al. 2020; Yang et al. 2018; Zhang et al. 2019; Zou et al. 2017]. Xu et al. [2019] propose a deep implicit surface network to generate 3D meshes from 2D images, combining local and global image features to improve the accuracy of distance field prediction. Tatarchenko et al. [2017] present a volumetric 3D generating network based on a convolutional decoder. This approach predicts the octree structure and individual cell occupancy values, producing high-resolution output even with limited memory. Furthermore, it can generate shapes from a single image and objects or entire scenes from high-level representations. Deep learning is also used to generate point clouds. For example, Fan et al. [2017] investigate generative networks for 3D geometry based on a point cloud representation. The main focus of the latter work is a well-designed pipeline that is used to infer point positions in the 3D frame from the input image, taking into account the viewpoint. Niu et al. [2018] propose a convolutional-recursive autoencoder architecture to retrieve cuboid, connectivity, and symmetry aspects of objects using a single 3D image. The encoder is trained on a shape contour estimation task, while a decoder focuses on the structure features by parsing the original image and the network and recursively decoding the cuboid structure. Xie et al. [2019] investigate a deep learning approach based on single and multiple views to reconstruct voxel 3D representations. The authors propose an encoder-decoder-based framework called Pix2Vox that can be used for 3D reconstruction from real-world and synthetic images.

Most of the mentioned 3D reconstruction methods based on images and deep learning are related to 3D shapes based on voxels or point clouds. They are limited to reconstructing simple and smooth shapes, such as chairs, furniture, or cars. The main difference from our approach concerns the object to be reconstructed, that is, the tree, which has more complex and detailed tiny structures, such as branches and leaves. In this context, making a direct prediction of 3D tree models is much more difficult. Furthermore, these methods are based on single or multiple RGB images, which provide more information than sketches.

Our work is similar to Liu et al. [2021c] in that it also uses a DNN to procedurally generate 3D tree models. Indeed, they use a conditional Generative Adversarial Network (cGAN) to extrapolate two depth images, which are useful to reconstruct the trees silhouette and skeleton as 3D point clouds using the edges and strokes obtained from an input RGB image. Finally, they create a final 3D tree model from the point clouds using a procedural modeling approach based on the self-organization concept [Palubicki et al. 2009]. Although this approach seems similar to ours, it differs substantially in methodology, input processing, and final results. Indeed, their input consists of RGB images from which the tree edges are extracted using the Canny [1986] method. For the approach to work, a user must draw the skeleton of the tree. Both images are used to feed and train a cGAN to create the depth images mentioned above, casting the problem into an image-to-image translation task. In contrast, we feed
a DNN using only sketches without any external intervention, thus keeping the information used low. In this way, our approach is more extensible with other types of sketch and tree species, and it is usable even without real tree pictures.

2.3 From sketches to 3D reconstruction

As reported by Ding and Liu [2016], using sketches to reconstruct a 3D model is intuitive for a human being. In this context, deep learning approaches can be helpful in generating a 3D mesh using minimal data, such as a sketch, as input information. Guillard et al. [2021] provide an encoder-decoder architecture to translate a 2D sketch into a 3D mesh. The method uses latent parameters to represent and refine a 3D mesh that can match the external contours of the sketch. Delanoy et al. [2018] propose a data-driven learning approach that can reconstruct 3D shapes from one or more drawings. This Convolutional Neural Network (CNN) based method predicts voxel grid occupancy from a line drawing and outputs an initial 3D reconstruction, while users can complete the drawing of the desired shape. There are also studies that adopt unsupervised learning methods for 3D object modeling. Wang et al. [2018c] propose a learning paradigm to reconstruct 3D objects from HM sketches that do not rely on labeled HM sketch data during training. Their paradigm takes advantage of adaptation network training, notably autoencoder and adversarial loss, and combines an unpaired 2D rendered image and an HM sketch in a shared latent vector space. In the second step, nearest neighbors are retrieved from the embedded latent space, and each sketch in the training set is used in a 3D Generative Adversarial Network. An interesting approach that is not based on deep learning is proposed in [Li et al. 2017]. This study presents a tool that can model complex free-form shapes by sketching sparse 2D strokes. The proposed framework combines multi-view inputs to model a complex shape that can be occluded. Compared to our approach, most of these methods fail in terms of faith in the GT. Indeed, their results show unbalanced predicted meshes for simple structures such as chairs and cars. Furthermore, they have problems with thin and layered structures like those of the tree trunk and branches, of which no test example is also shown. Some of these approaches also require continuous user intervention to improve faith in the prediction. On the contrary, we used only a single sketch as input for our pipeline, and the prediction is reliable for the thin and layered structures to be reconstructed.

2.4 Procedural modeling

Rule-based methods [Ebert et al. 2002; Schwarz and Müller 2015] can also take advantage of deep learning and overcome the need for user intervention in manipulating a large number of parameters in rule sets. Thus, Huang et al. [2017] present an HM sketch approach to automatically compute a set of procedural model parameters that can be used to generate 2D/3D shapes that resemble the initial input HM sketches. An interesting aspect of this study is that it focuses on three sets of procedural modeling rules, namely 3D containers (e.g., vases), 3D jewelry, and 2D trees. The authors did not consider 3D tree shape generation due to the high complexity of the task. However, several other approaches can overcome this problem by controlling the output of procedural modeling algorithms. Methods include [Haubenwallner et al. 2017], which uses a genetics-based algorithm, and [Polasek et al. 2021; Stava et al. 2014], which is based on a Monte Carlo Markov Chain. Unlike our approach, these methods require a parameter learning phase by the users to understand their functionalities. To further facilitate the modeler work, Longay et al. [2012] developed a procedural modeling-based application for generating 3D tree models with a tablet. The modeler can define the tree’s profile and the direction of the branch’s growth with a brush. Many other tree details need to be defined by working on the parameters related to the procedural modeling algorithms used. Although this approach is more intuitive than the previous ones, the presence of parameters implies a preliminary study by the user that is not necessary with our methodology. An important rule-based method is L-system, proposed by Lindenmayer [1968], which is widely used for plant generation. For example, there are several algorithms based on the L-system model [Liu et al. 2021a; Prusinkiewicz and
In particular, an interesting inverse procedural modeling approach was introduced by Guo et al. [2020a], which uses deep learning to detect an L-system from an input image with branching structures. Because this method can predict only parameters for generating linear branching structures, both the input image and the resulting grammar have no information about the crown of a tree. Li et al. [2021] provide an approach to overcome this limitation. They used a combination of deep learning and procedural modeling algorithms to extract a 3D tree model from a single photograph of a tree. This method is based on three DNNs that, starting from an input photograph of a tree, mask out the tree, identify the tree species, and extract the Radial Bounding Volume (RBV) of the tree, respectively. The RBV is then used as an input for a procedural modeling algorithm that generates the 3D model of the tree. Hence it follows that this neural architecture is much more complex than ours. Furthermore, their input RGB images have much more 3D hints than sketches, such as shadows and information about textures. Interesting are also newer approaches for generating 3D models from HM sketches. Wang et al. [2018a] use a multiple encoder-decoder network architecture to create a final draped 3D garment from a single sketch. The complexity of the network architecture is strongly linked to the necessity of predicting not only the parameters of the 2D garment pattern but also the body shape parameters, which must be related in order to obtain a final 3D garment that resembles the input sketch. The idea of using multiple neural networks to predict a different set of parameters that have to be related is followed by Unlu et al. [2022]. Indeed, they use two neural networks to predict 2D silhouettes and 2D joints from an input mannequin sketch. These intermediate representations are then used to generate the mannequin 3D model. Unlike the two previous approaches, ours can generate 3D tree models with a single neural network without using an intermediate representation, although tree geometry is linked to a complex and non-linear set of rules.

3 GENERATION OF 3D TREE PARAMETERS

In this section, we discuss the details of our approach, notably: (i) the dataset creation pipeline; (ii) the configuration of the TSN, and; (iii) the training procedure. The RT Blender add-on is a key element in our pipeline as it quickly allows the generation of the training dataset and the visualization of the 3D tree mesh. The latter is generated from Weber-Penn [1995] parameters predicted by the TSN from SG sketches created by the add-on. To define our parameters, we adopted the current standard notation in CG vegetation simulation as used in [Pradal et al. 2009; Štava et al. 2014; Tang et al. 2019; Yang et al. 2019].

3.1 The dataset creation pipeline

One of the weaknesses of supervised learning methods is finding a way to arrange a large set of labeled data, which also have to be well-structured to obtain the expected results [Fredriksson et al. 2020; Mohri et al. 2018; Yang et al. 2021]. Our RT Blender add-on overcomes this drawback by creating, for each species, 250 3D tree meshes and storing parameters in a dedicated dictionary file for each tree (see Details in our GitHub website [Manfredi et al. 2021]). We divide the Weber-Penn parameters into two subsets: fixed and unfixed. Fixed parameters always have the same values for each tree of the same species, while unfixed parameters have randomly-controlled values that vary the shape and detailed visual features of the tree. The TSN could be trained to understand the tree species shown in the input sketch by adding a classification branch. However, this would increase its complexity and, consequently, degrade prediction performance. Moreover, adding a classification branch is unnecessary because it is possible to identify the tree species associated with the input sketch using fixed parameters. This task was implemented as a robust algorithm 1 and is discussed in Section 4. Starting from the pre-existing Sapling Tree Generator Blender plugin, we manually define a dictionary of fixed and unfixed parameters for each of the 5 tree species. Here, the aim is to obtain consistent 3D tree meshes in terms of visual features. The tree species were chosen in order to introduce as much differentiation as possible in their shape and visual features. The consistent mesh dictionaries produced by our RT plugin were used as a starting point to generate the other 3D...
meshes and their respective parameter dictionaries. The unfixed parameters were randomly varied with respect to their order of magnitude, as shown in Table 1.

Table 1. Unfixed parameters and their range (minimum and maximum). The Sign parameter is binary.

| Unfixed Parameters                  | Min Values | Max Values |
|-------------------------------------|------------|------------|
| Sign (binary)                       | -1         | 1          |
| Tree Forks Number                   | 0          | fixed parameter |
| Parent Branch Roll Angle            | -360       | 360        |
| First Half Internodes Branching Angle | -360     | 360        |
| Second Half Internodes Branching Angle | -360   | 360        |
| Internode Branching Angle Variance | -360       | 360        |
| Sibling Angle                       | -360       | 360        |
| Sibling Angle Variance              | -360       | 360        |
| Branch Roll Angle                   | -360       | 360        |
| Branch Roll Angle Variance          | -360       | 360        |
| Leaf Roll Angle                     | -360       | 360        |
| Leaf Angle Variance                 | -360       | 360        |
| Leaf Scaling Factor                 | 0          | ∞          |
| Leaf Scaling Factor Variance        | 0          | 1          |
| Parent Branch Angle Variance        | -360       | 360        |
| Number of Branch Whorls             | 0          | ∞          |

In this way, we obtained 250 dictionaries for each tree species, which are used individually in our RT plugin to generate the corresponding Blender Curve [Hughes et al. 2014] object. This object is converted into two meshes: the skeleton (trunk and branches) of the tree, excluding the foliage, and the foliage alone.

The obtained meshes are loaded into a well-designed Blender scene integrated with 4 cameras corresponding to the 4 tree views: front, back, left, and right as shown in Figures 2. These views have been chosen because they are easy to recognize and also because is easier to make visual comparisons with the 3D model predicted by the TSN. Furthermore, each of the 4 views is completely different from the others: a characteristic that can reduce the TSN overfitting, granting a great level of generalization, as shown in Figure 7. We also tried to train our TSN with other views, but we did not get significant improvement. In addition, our scene contains a directional light, without shadows, to better illuminate the tree. The global scene illumination consists of Blender path tracing and ambient occlusion to increase rendering realism. After setting the scene, the SG sketch for each view has to be rendered. The first step of the SG sketch generation pipeline consists in adding two black materials to the tree, one for the trunk and one for the foliage, as shown in Figures 2a and 2b. The first is a simple diffuse material with zero roughness. The second consists of a simple leaf-shaped black texture based on the tree species.

The next step is to render the skeleton and the foliage meshes of the tree individually with the previously defined black materials. To obtain the skeleton edges, we define several steps as shown in Figure 2c: (i) the first step is to slightly thin the skeleton mesh to remove the thinnest branches; (ii) the second step consists of the following chain of compositing operations [Porter and Duff 1984]:

(a) remove the background using a 0.5 threshold value;
(b) apply the Intel® Open Image Denoise-based filter to delete the residual background artifact;
(c) apply the Sobel filter [Kanopoulos et al. 1988] to trace rough outlines;
(d) apply a color ramp that sets pixel values less than 0.9 to 0 and the remainder as 1 to highlight all branch edges;

ACM Trans. Intell. Syst. Technol.
Fig. 2. Generation of the SG sketch. For each camera view, the tree’s skeleton and foliage are individually rendered after applying the black and white materials and after a set of pre-processing operations specific to the two species of meshes. In the final step, these renderings are mixed to obtain the final SG sketch.

(e) apply the inversion color filter to obtain a negative, with white background and dark edges;
(f) apply the erosion morphological operation to delete scattered points and produce the final skeleton sketch, as seen in Figure 2c.

Similarly, to obtain the foliage edges (see Figure 2d), we use a simpler compositing chain of operations consisting of:

(a) apply the background removal filter using a 0.9 threshold value;
(b) apply the strong Gaussian filter to uniformise and emphasize the shape of the foliage;
(c) apply a color ramp that sets values less than 0.01 to 0 pixel and the remainder 1, to include blurred foliage edges;
(d) apply the Sobel filter to trace the edges;
(e) apply the same inversion color filter used for the skeleton sketch to obtain the foliage sketch, as shown in Figure 2d.

The final synthetic sketch is obtained by multiplying the skeleton and foliage images (the mixed sketch), as shown in Figure 2e. The drawing style chosen for the SG sketch aims to resemble a hand-made sketch drawn by a typical user, as shown in [Eitz et al. 2012; Zou et al. 2018].

For each camera view in the scene, we generate an SG sketch. The final dataset, therefore, consists of 250 trees × 4 views × 5 tree species = 5000 SG sketches of trees (1000 sketches for each species [Liu et al. 2021c]), together with their parameter dictionaries and ground truth (GT) images. The resolution was 608 × 608, which is the maximum dimension supported by our tested core net (see Section 5.2). This initial resolution can be resized based on the required input of each tested TSN, including those chosen by us (see Section 3.2). The dataset is split into training and validation sets, with examples of the 5 tree species, together with their GT parameters, and SG sketches, as reported in Figure 3. The validation set consists of a single viewpoint for each tree species, and the training set consists of the other three viewpoints [Bishop 1995]. A specific test dataset was created separately, as described in Section 4.

3.2 The TSN architecture and training procedure

To address the regression problem, we define a specific TSN architecture based on EfficientNet-B7 [Tan and Le 2019] as the core net. This network was the best core net for our architecture after a series of empirical experiments with other well-known networks, as reported in Section 5.2. EfficientNet-B7 belongs to a family of networks created starting from an initial network architecture called EfficientNet-B0, subsequently scaled...
up using the method proposed by Tan and Le [2019]. This method aims to an improvement in performance by uniformly scaling the EfficientNet-B0 width, depth, and image resolution. All the EfficientNet architectures are characterized by a convolutional layer followed by 7 macro-blocks, each containing \(L_i\) Inverted Residual Blocks, sometimes called MBConv blocks [Sandler et al. 2018]. EfficientNet-B7 is 8.4× smaller, 6.1× faster, and much more accurate on ImageNet than the best existing CNN.

Our TSN input consists of 224 × 224 reshaped sketches (see Sec. 3.1). This is because the resolution for training CNNs is generally between 64 × 64 and 256 × 256, considering that a CNN model performs better with lower input image resolutions [Thambawita et al. 2021]. Furthermore, Tan and Le [2019] show that the EfficientNet base model accuracy gain saturates after reaching 80%, also for resolutions higher than 224 × 224. Each sketch has no \(a\) channel to stay coherent with the images contained in the dataset used to train and test EfficientNet [Tan and Le 2019]. The background of each sketch is white to make the TSN more efficient in extracting the essential features of the image [Kc et al. 2021]. As we use a supervised learning paradigm, each input image is associated with the corresponding tree parameters, structured as a matrix we call the target matrix. Each target matrix row represents a single parameter, and column values are parameter values. Since the number of values of each parameter item can be 4 or 1, our target matrix has a dimension of 4 \(\times\) \(n_p\), where \(n_p = 62\) is the number of parameters (see Details in our GitHub website [Manfredi et al. 2021]). To generate a consistent matrix, 1-valued parameters are repeated in each column. The matrix contains all fixed and unfixed parameters reported in our GitHub website [Manfredi et al. 2021], without considering the pruning, armature, and animation parameters, which are left as default values where they are mandatory. To avoid overfitting and underfitting problems (see Sec. 3.2.1), based on their order of magnitude, we divide this target matrix into six sub-matrices that represent the target for each final TSN branch. The dimension of each sub-matrix is 4 \(\times\) \(n_t\), where \(n_t \in [1; n_p]\), which represents the number of parameters for a specific order of magnitude (see details in Sec. 3.2.1). Our TSN was trained using Adam [Kingma and Ba 2014], with default parameters and \(1e^{-5}\) as the initial Learning Rate and the Mean Squared Error as the loss function. To assess performance accuracy, we used \((1 - RMSE) \times 100\) (Root Mean Square Error) during the training phase. Because the training dataset is not so large, we used the EfficientNet-B7 layers pre-trained with ImageNet [Deng et al. 2009; Goodfellow et al. 2016] dataset of images. This technique is largely used to have good Network performance with a not very large dataset [Weiss et al. 2016]. We conducted some tests to select the best batch size value, which is equal to 8 for our approach. Our TSN was trained using an NVIDIA Titan XP GPU with a 12 GB G5X frame buffer.
Fig. 4. The architecture of our TSN, with input in the form of a 224 × 224 sketch. The initial block is a simple convolutional layer. The following blocks contain $L_i$ MBConv blocks. The last part of the network is made up of six branches, one for each parameter order of magnitude. Each branch has a fully-connected layer with a linear activation function and a reshape layer.

3.2.1 Overfitting and underfitting. The optimal dataset and the TSN configuration were identified after several attempts, in which we addressed overfitting and underfitting problems. Our first “toy” approach used a basic VGG-16 [Simonyan and Zisserman 2015] DNN as the core net that represents 608 × 608 rendered images (sketches and GT). This configuration was found to lead to general overfitting with higher intensity in some branches and very high losses. Therefore, we inserted dropout layers [Srivastava et al. 2014] before the layers of each DNN output branch with $\alpha = 0.2$ for branches that were less affected by overfitting and $\alpha = 0.5$ for branches that were more affected. High loss during training could indicate the vanishing gradient problem and, consequently, underfitting [Capece et al. 2019; He and Sun 2015; Kolen and Kremer 2001]. To reduce this problem, we introduced a residual learning approach using skip connections [He et al. 2016] placed among consecutive layers in the VGG-16 core net. Although this approach resulted in acceptable performance, we used this experiment as a baseline for our final configuration. This configuration use a more recent version of EfficientNet which can better manage the problems described above. Four other attempts used ResNet [He et al. 2016] (variant 50), AlexNet [Krizhevsky et al. 2012], Inception V3 [Szegedy et al. 2016] and CoAtNet [Dai et al. 2021] as core nets. ResNet and Inception V3 are classic neural networks [Khan et al. 2020] that are used in many computer vision tasks, AlexNet was used in a comparative study with a similar sketch-to-mesh-parameter approach [Huang et al. 2017], while CoAtNet and EfficientNet represent the SOTA for image recognition. Section 5.2 reports quantitative performance comparisons between VGG-16 with skip connections and EfficientNet-B7. Using VGG-16 as the core net, we experimented with using the entire target matrix as the output of the DNN, with only one output branch. However, the results were very poor and the DNN was completely underfitted. We, therefore, defined 6 DNN
output branches based on the order of magnitude of parameter values. This was possible because the Weber-Penn Blender parameter dictionary consists of heterogeneous data, as reported in our GitHub website [Manfredi et al. 2021]. Therefore, we were able to carry out conversions and resize many of them as a function of their order of magnitude. Specifically, we defined several sub-matrices, one for each DNN output branch, as shown in Figure 4. Discrete integers and string parameter values were parsed as floats; boolean parameter values were parsed as integers; non-numeric string parameter values were parsed as floats using the Labeled Encoding method (e.g., Leaf Shape described in our GitHub website [Manfredi et al. 2021]; and binary [-1, 1] parameters were converted to [0; 1]. Although it is not possible to consider [−inf, inf] values, we normalized them and [-360, 360] values to a [-1, 1] range using the Max Abs Scaling method. This scales the data and preserves sparsity. Maximum values were stored in normalization matrices and reused in the testing step. Tests of Inception V3 identified significant overfitting on [−inf, +inf] and [0, 1] DNN output branches, which were reduced by adding a $L_2$ regularization [Ng 2004] on the first two fully-connected layers, followed by a dropout layer with $\alpha = 0.2$. We obtained little overfitting on the [-360, 360] DNN output branch; in this case, we only used a dropout layer with $\alpha = 0.5$ to reduce the problem (see Figure 4). EfficientNet-B7 performs well on each branch, with slightly better performance on $[0, \infty]$, $[-1, 1]$, $[-360, 360]$, and $[\min, \max]$ branches. For the sake of simplicity, Figure 5 shows the general training and validation loss calculated by averaging the losses of the various branches.

![Training Loss vs Validation Loss of TSN.](image)

### 4 RESULTS

The tree sketch is an input to our TSN, which produces six $4 \times n_t$ sub-matrices containing predicted parameter values (see Sec. 3.2). These sub-matrices, which need to be de-normalized, are multiplied from their corresponding normalization matrices stored after the training phase. This operation makes it possible to adopt predicted TSN values as valid Blender inputs. The sub-matrices that are obtained from a single sketch prediction are rearranged as a Blender-like parameters dictionary. In particular, each sub-matrix column is associated with a dictionary key, representing a specific parameter. When the expected parameter is represented by a single value rather than a vector of four elements, only the first value is taken. The dictionary is also used to identify the species of the predicted tree (see Algorithm 1) and assign the correct textures, based on the tree species, to the 3D Blender mesh from the same dictionary. Specifically, Algorithm 1 takes two data structures as input. The first is a well-designed dictionary-based data structure with characteristic parameters $CP$ that are organized for each tree species. Each element in $cp \in CP$ contains three items of information: the parameter name, its value, and the tree species. A parameter is defined as $cp$ for a specific tree species if it respects two properties: (i) it is always present in all samples of that specific tree species, and; (ii) its value is in a specific range for that tree species. The second input is $P$, which is a dictionary of predicted parameters. We compare each parameter $p \in P$ with the corresponding $cp$.
Fig. 6. The results of the SG test set obtained from our RT add-on taken from the same camera view, for each tree-species. The first row shows the input SG sketches; the second row shows the reconstructed 3D meshes using the corresponding predicted parameters dictionaries; the last row shows the GT.

Fig. 7. Results of TSN network given images with different camera views: in the left column the camera was right tilted; in the middle was botom placed and tilted upward; in the last column the camera was left tilted.

for each tree species using the parameter name and verify that the value of $p$ respects the following condition:

$$p.value \in [c.p.value - \epsilon, c.p.value + \epsilon]$$

(1)
where $[cp.value - \epsilon, cp.value + \epsilon]$ is the range of acceptable values for that tree species. Let $eligibles$ be a dictionary in which the keys are the tree species, and the values are counters (one for each tree species). If the predicted parameter respects the condition (1), then this parameter is eligible for that tree species, and its counter is increased by one. Finally, Algorithm 1 returns the tree species of the predicted dictionary based on the highest percentage of counters. The rearranged predicted parameters dictionary is then loaded from Blender and interpreted using a load utility method in our RT plugin. This utility makes it possible to generate the 3D tree mesh from the input dictionary and assigns the correct texture based on the tree species detected by Algorithm 1. The texture is chosen from a set of predefined textures for each tree species. To test our approach, an appropriate dataset was defined based on our tree species. Figure 6 shows the input SG sketches similar to the training set that was generated by our RT add-on, the reconstructed 3D tree mesh, and the GT. As Figure 6 shows, the tree
structure of the test sketches was correctly predicted by our TSN, notably with respect to, for example, the number of splits for each trunk segment, curve angles, the number and distribution of branches, the ratio of the trunk to its height, and the overall height. As can be seen from Figures 6, our results are visually consistent with the input sketches and are consistent with the GT parameters with regard to SG sketches. To demonstrate the TSN generalization capability, we also created a test set containing sketch images rendered in different views than the 4 views used to generate the training data. The results of this test, shown in Figures 7, confirm the generalization ability of TSN. Figure 8 shows some SG sketches and the 3D meshes generated by Blender based on the predicted parameters dictionaries. Figure 8 also shows the applied materials as a function of the tree species and the normal mapping of each 3D mesh.

5 COMPARISONS
To the best of our knowledge, this approach is the first attempt to generate 3D tree shapes from well-defined predicted parameters. Consequently, we try to do our best to compare our results with other approaches. Inspired by [Huang et al. 2017], we assessed and validated our approach in a user study based on HM sketches provided by 15 different users, as reported in Sec. 5.1. In addition, we provide quantitative comparisons using different core nets to validate our choice of EfficientNet-B7 as optimal (see Sec. 5.2 and Sec. 5.3). Finally, we provide an extensive qualitative analysis of our TSN predicted parameters, as reported in Sec. 5.4.

5.1 Controlled experiment
We conducted a controlled experiment with 15 non-experts in drawing to test our approach in a real-life context. Each person was asked to provide us with a single sketch for each species of the tree. The goal was to assess the confidence level of our TSN parameters prediction using HM sketches provided by the participants. Participants were free to choose any software package to draw their trees. For each participant, we randomly selected 5 SG sketches, one for each tree species. Participants were asked to look at each SG sketch for two minutes and then draw the tree sketch according to their style. In this way, participants can understand what species of trees the system could handle without being constrained too much on the drawings.

To have another significant proof of the generalization ability of TSN, we adopt the approach of Wang et al. [2018a], which consists in finding the Nearest Neighbour Sketch (NNS) of the SG sketch extracted from the 3D tree model generated by TSN, given an HM sketch as input. The NNS is obtained from the training set using the TSN features of the sketches. Figure 9 shows the results of the comparison between the HM sketch, the reconstructed SG sketch described above, and the NNS. The first column of Figure 9 shows that the reconstructed SG resembles the HM sketch more than the NNS. This can be noticed in the width of the trunk (green box) and in the presence of an additional branch (blue box) in NNS that in the HM sketch and in the reconstructed SG is not present. The second column shows an NNS with a different orientation and trunk bending compared to the HM and the SG reconstructed sketch. Furthermore, in the NNS, we can count one less side branch (green box), both for the left and the right side, than the HM and the SG reconstructed sketch. The blue box identifies the tip of the tree. In the third column, it can be noticed an NNS quite different from the HM sketch. By contrast, the reconstructed SG sketch looks more like the HM one, both in the shape of the trunk (blue box) and in the bending (green box) and in the general arrangement of branches. Also, in the fourth column, the bending and the general arrangement (blue box) of branches are more similar between the HM and the SG reconstructed sketches rather than between the HM sketch and the NNS. In the last column, the NNS has an extra Branch Whorl compared to the reconstructed SG and the HM sketch. In addition, the general branch arrangement of the HM sketch is more similar to the reconstructed SG sketch than the NNS.
Fig. 9. Generalization ability of TSN. The first row shows the HM sketches. The second row shows the 3D model generated by the TSN from the HM sketch. The third row contains the SG sketches obtained from the reconstructed 3D model. The last row shows the nearest sketch retrieved from the training set.

5.2 Core net testing

The choice of the EfficientNet-B7 core can greatly influence the outcome of the approach. Therefore, we ran tests with different core models and assessed their performance. In particular, for each tested core model, we analyzed the $1 - RMSE$ for each specific branch and the overall average for all branches. The first comparison was between EfficientNet-B7 and VGG-16 with skip connections (see Sec. 3.2.1). The second comparison was performed with ResNet and Inception V3, which are used as core models in several computer vision tasks [Szegedy et al. 2016]. For ResNet, we used variant 50 because its depth is sufficient to extract the essential low-level features needed for our task. For the test made with Inception V3, we used the first 2 convolutional blocks and the first 5 Inception Modules to reduce the DNN complexity and consequently the overfitting problem caused by the overparametrization [Salman and Liu 2019]. We also compared EfficientNet-B7 with AlexNet [Krizhevsky et al. 2012] to evaluate the validity of procedural modeling-based approaches for 3D mesh generation that are proposed in [Huang et al. 2017]. For the sake of completeness, we finally compared EfficientNet-B7 with another SOTA
network called CoAtNet. Each test was performed using 30 sketches, of which 15 were SG, and 15 were HM. For each core model, we trained our architecture for 2k epochs, and the assessment was based on the following factors: (i) possible overfitting of branches (low and high) based on the validation curve accuracy with respect to the training curve accuracy; (ii) the accuracy of each branch, evaluated by observing the validation curve, and; (iii) the generalization level, evaluated by observing the testing curve.

Table 2 shows the results of the comparison with the SG-based test set. These sketches were obtained using our pipeline, as described in Section 3.1. The SG tests were run using all parameters and the prediction dictionaries were compared with GT parameters using $1 - \text{RMSE}$. EfficientNet-B7 performs better than the other core models. However, CoAtNet performs slightly better on the $[-360, 360]$ branch. Inception V3 also performs well, but there are no branches where Inception V3 outperforms EfficientNet-B7. The overall performance suggests that EfficientNet-B7 performs better than the other networks.

| Core Nets      | 1-RMSE       |
|---------------|--------------|
|                | $[\inf, \inf]$ | $[-360, 360]$ | $[0, 1]$ | $[0, \inf]$ | $[\min, \max]$ | $[-1, 1]$ | Overall     |
| VGG-16        | 0.8022       | 0.9552       | 0.7988   | 0.9454       | 0.9772         | 0.9899     | 0.9115       |
| ResNet-50     | 0.7885       | 0.9568       | 0.7803   | 0.937        | 0.9579         | 0.9646     | 0.8975       |
| AlexNet       | 0.7791       | 0.9552       | 0.7963   | 0.9688       | 0.9792         | 0.9895     | 0.9113       |
| Inception V3  | 0.7957       | 0.959        | 0.802    | 0.9657       | 0.9811         | 0.988      | 0.9152       |
| CoAtNet       | 0.8073       | 0.9607       | 0.8047   | 0.9556       | 0.9769         | 0.9848     | 0.915       |
| **EfficientNet-B7 (our)** | **0.8093** | **0.9595** | **0.8129** | **0.9798** | **0.9936** | **0.9964** | **0.9253**   |

Table 2. Core net comparisons of SG sketches. Observations of overall $1 - \text{RMSE}$ suggest that EfficientNet-B7 has higher accuracy. However, CoAtNet performs slightly better on the $[-360, 360]$ branch. The best values are highlighted in bold.

For HM testing, we prepared some hand-made sketches inspired by those generated by the SG (see Sec. 3.1 and Figure 2) approach and considered their corresponding parameters as GT. The comparisons reported in Table 3 show that both SOTA networks (EfficientNet-B7 and CoAtNet) and Inception V3 achieved better results than the other three core models for HM sketches with regard to training sketches, demonstrating a higher level of generalization. In particular, we can notice that EfficientNet-B7 has higher accuracy with respect to CoAtNet and Inception V3. Furthermore, a comparison of the results reported in Table 2 and Table 3, which have similar features, and an analysis of the training report, identified that AlexNet is slightly overfitted on $[0, \inf]$ and $[\min, \max]$. By applying the same analysis to the other core models, we identified that for VGG-16 with skip connections, the branches $[\inf, \inf]$ and $[0, \inf]$ are slightly overfitted, while for ResNet-50, Inception V3, EfficientNet, and CoAtNet there is no evidence of overfitting. However, EfficientNet-B7 was selected as the best core model due to its higher accuracy compared to the other DNNs and its generalization ability with HM sketches significantly different from training data.

5.3 Hausdorff Distance

Since in the previous section, we provided a quantitative analysis of the DNN’s performances in terms of accuracy, in this section, we assess the quality of the reconstructed 3D trees. For this purpose, we used a metric called Hausdorff Distance (HDD) [Cignoni et al. 1998]. In particular, we compare the distances calculated for EfficientNet-B7, CoatNet, and Inception V3, because they represent the best-performing DNNs, as reported in the previous section. The HDD represents the maximum distance in a set of distances between two meshes. This set contains all minimum distances between the first and second mesh vertices. In our case, we calculate, for each DNN, the HDD between 15 tree meshes of different species reconstructed from the DNN predictions and their ground-truth...
Core Nets 1-RMSE

| Core Nets      | $[-\infty, \infty]$ | $[-360, 360]$ | $[0, 1]$ | $[0, \infty]$ | $[\min, \max]$ | $[-1, 1]$ | Overall |
|----------------|----------------------|----------------|---------|---------------|----------------|-----------|---------|
| VGG-16         | 0.7577               | 0.9485         | 0.7931  | 0.9176        | 0.931          | 0.9795    | 0.8879  |
| ResNet-50      | 0.7585               | 0.9605         | 0.8278  | 0.9356        | 0.952          | 0.9614    | 0.8993  |
| AlexNet        | 0.75                 | 0.955          | 0.8057  | 0.9136        | 0.9124         | 0.975     | 0.8853  |
| Inception V3   | 0.7873               | 0.9618         | 0.8122  | 0.9607        | 0.9746         | 0.9862    | 0.9138  |
| CoAtNet        | 0.7712               | 0.9646         | 0.8192  | 0.9455        | 0.9608         | 0.9815    | 0.9071  |
| EfficientNet-B7 (our) | 0.7688 | 0.9628 | 0.8658 | 0.979 | 0.9883 | 0.9957 | 0.9267 |

Table 3. The results of HM sketches clearly show the difference between the different core nets. Based on overall performance, EfficientNet-B7 has higher accuracy. Despite the differences between hand-drawn sketches, EfficientNet-B7 is demonstrated to have a greater ability to generalize. The best values are highlighted in bold.

meshes. The 15 ground-truth trees are the same for all DNNs to grant the results coherence. Table 4 shows, for each network, the average of the calculated HDDs. The results in Table 4 show that EfficientNet-B7 is still the better choice because its HDD tends more to 0. It means that the meshes from EfficientNet-B7 are more similar to their ground truth than those from Inception V3 or CoAtNet. Additionally, our approach provides better results in calculating this metric than the SOTA [Liu et al. 2021c].

| Core Nets       | Hausdorff Distance |
|-----------------|---------------------|
| Inception V3    | 0.047               |
| CoAtNet         | 0.041               |
| EfficientNet-B7 (our) | **0.031** |

Table 4. Hausdorff distances of the best performing DNNs. Based on the results, the meshes resulting from EfficientNet-B7 are more similar to their ground truth. The best values are highlighted in bold.

To further test the TSN accuracy, we provide an experiment. It consists of generating the 3D model of a tree that TSN has never seen and rotating it by 5 degrees until the starting position is reached. For each rotation, a new SG sketch has to be rendered and given as input to the TSN. Finally, the HDD has to be calculated for each TSN prediction. As shown in Figure 10, for each degree of rotation, the HDD value is approximately 0.03, in line with the results in Table 4. Figure 11 presents the experiment’s worst, median and better case. Each column contains the SG sketch given as input to our TSN, the 3D model predicted by the TSN, the 3D model of the ground truth with the realistic textures, and the 3D model of the ground truth whose vertices have the color corresponding to their HDD value.

5.4 Qualitative analysis

In this section, we provide an empirical analysis of our TSN-predicted parameters using HM sketches for which GT parameters were unknown. To do this, we manually extracted some easily-visible parameters for each tree species based on the input sketches and compared them with the TSN predictions. Table 5 describe the symbolism adopted in this section.

Figure 12 shows the HM maple tree input sketch and some of its simple visual parameters. This table shows some of the TSN’s predicted parameters and performance accuracy. In particular, the first GBS value, highlighted in dark orange, shows that the trunk (the first branch level) is attracted upwards (positive sign) and the branches

ACM Trans. Intell. Syst. Technol.
Fig. 10. HDD distances for each degree of rotation. Because the HDD values are near 0 we narrowed the y-axis range to [0, 0.1]. In this way, the minimal differences between trees are highlighted.

Fig. 11. Worst, median, and the better case of the experiment with different degrees of rotation. For each case are presented the SG input sketch, the reconstructed 3D model, and the 3D model of the ground truth both with realistic texture (Ground Truth) and with its vertices having the color corresponding to the HDD value (GT HDD). At the bottom of the figure, the HDD Mean Value is shown.

(second branch level) are attracted downwards (negative sign) with a different attraction coefficient, as correctly predicted by our TSN. CBH indicates that about 30% of the maple tree height has no branches. The trunk in the sketch is also split into two parts, confirmed by the predicted $N_{TF}$ parameter, highlighted in green, which
Table 5. Summary of symbolism used in this section.

| Parameter                                | Symbol | Parameter                                | Symbol |
|------------------------------------------|--------|------------------------------------------|--------|
| Gravity-bending Strength                 | GBS    | Crown Base Height                        | CBH    |
| Number of Tree Forks                     | NTF    | Branch Distribution                      | BD     |
| Number of Branches                       | NB     | First Half Internodes Branching Angle    | φFIB   |
| Second Half Internodes Branching Angle   | φSIB   | Number of Levels                         | NL     |
| Number of Branch Whorls                  | NBW    |                                          |        |

Fig. 12. Simple visual parameters of the maple tree. The corresponding table reports TSN predicted values. Indices of elements in array-structured parameters represent the branch level.

indicates one trunk subdivision. The NB parameter indicates the number of branches for each branch level. There are 0 first-level branches in the case of maple tree because the trunk is not considered a branch, and 54 second-level branches. φFIB and φSIB trunk signs are positive, which means that the trunk and its tip are curved backward the front of the tree. Second-level branches and their tips are curved in the opposite direction to the trunk, as shown in the related sketch. The remaining parameters are also correctly predicted by our TSN. Notably, the e.g., NL parameters indicate that the sketch is characterized by two branch levels.

Figure 13 is a visual comparison of some simple parameters and their corresponding TSN-predicted values. In this case, CBH indicates that only the 1% of the palm tree has no branches, but the BD parameter gathers all of the branches towards the top of the tree and attenuates the CBH effect. This aspect is also confirmed in the maple tree example 12, where these two parameters are balanced. In the sketch, there are no subdivisions of the trunk, indicated by the NTF parameter. Since φFIB and φSIB trunk signs are discordant, the trunk is S-shaped and curved backward (positive sign for φFIB), and its tip (negative sign for φSIB) is curved forwards. The first-level branches follow the curvature of the trunk. Finally, this example shows the NBW parameter, which indicates how many rings the branches are distributed on around the trunk.

The most evident parameter in Figure 14 is NBW. Noteworthy are also the sign and number of GBS, φFIB, and φSIB parameters. The red arrows show the sign of the φFIB parameter and indicate that all branch levels are
Fig. 13. An example of a palm tree SG sketch with predicted parameters. The correctness of the prediction is mainly demonstrated by the parameters $BD$ and $CBH$, which indicate that the foliage is concentrated towards the top of the tree.

Fig. 14. This figure and the corresponding table show that the number of $N_{BW}$ is correctly predicted by our system, as are other parameters.

curved forward. The tips of the last two branch levels follow the same direction of curvature as their base, and the trunk tip is curved backward from its base, as can be seen from the $\phi_{SIB}$ parameter. In this case, the branches of all levels are attracted upward, as correctly predicted by our TSN.

The cherry tree shown in Figure 15 is characterized by four branch levels ($N_L = 4$). The coefficients of $\phi_{FIB}$ and $\phi_{SIB}$ indicate the steepness of the curve. In this case, the first is curved backward, and the other branch levels are curved in the opposite direction. The curvature of the tip of the first three branch levels follows the curvature of the base of the trunk, and the tips of the last branch levels follow their base curvature. In this case, the trunk is
TreeSketchNet: From Sketch to 3D Tree Parameters Generation

Fig. 15. An example of a cherry tree. In this case, the $N_{TF}$ parameter is 1, indicating that the main trunk is divided into two sub-trunks. The branch attraction points upward for all levels. Finally, the $CBH$ indicates that about the 50% of the cherry tree has no branches, which is consistent with the $BD$ parameter.

Fig. 16. An example of the bonsai tree SG sketch. The S-shape of the trunk and first-level branches is indicated by the discordant signs of $\phi_{FIB}$ and $\phi_{SIB}$ parameters. The bonsai tree shown in Figure 16 also have four branch levels ($N_L = 4$). The trunk of the bonsai has no splits, as indicated by the $N_{TF}$ parameter, and is also S-shaped, as showed by the discordant signs of the $\phi_{FIB}$ and $\phi_{SIB}$ parameters. The first-level branches are also S-shaped but specular in their trunk orientation. Indeed, the $\phi_{FIB}$ and $\phi_{SIB}$ of the other branches’ levels are all negative, so the bases and the tips of these branches are curved in the forward direction. In addition, the 20% of the cherry tree has no branches, and all the branch levels are attracted upward, as confirmed by the $CBH$ and $GBS$ parameters, respectively.
CONCLUSION AND FUTURE WORK

Our proposed approach can predict parameters for 3D tree mesh generation using a DNN method. The main goal is to automate procedural modeling by introducing a broker system between the modeler and the modeling software used to build 3D trees. The core of our broker consists of a DNN based on convolutions that we call TreeSketchNet, which is trained using supervised methods to learn the mapping between well-known Weber-Penn tree parameters and 2D sketches of trees. As a large amount of data is needed for training and validation datasets, we developed a dedicated RT Blender add-on. This add-on makes it possible to automate the generation of realistic SG sketches starting from 3D tree meshes that are generated a priori using a set of fixed and unfixed randomly-controlled parameters. This approach is implemented to overcome the problem of generating an expensive HM dataset of drawings. For our experimental purpose, we consider 5 tree species (maple, pine, bonsai, palm, and cherry) and create corresponding sketches from the front, back, left, and right camera angles. We assessed our system and the obtained results using a controlled experiment. Specifically, we asked participants to provide HM sketches based on reference SG examples shown for two minutes and used them to test the system. The results were promising and we believe may be a starting point for future research. Our experiment also highlighted the high level of generalization and validated the accuracy of our approach. Furthermore, we experimented with several other core nets to identify the most suitable and performant option. Notably, we tested AlexNet, which is widely used in computer vision tasks, and won the ILSVRC competition [Russakovsky et al. 2015], becoming established as the SOTA in the deep learning field. AlexNet is the only DNN that has been used in a similar sketch-to-mesh parameter approach [Huang et al. 2017]. However, it did not perform as well as EfficientNet-B7 and was therefore discarded. Finally, we provide a qualitative analysis of our results, specifically, a visual comparison of the predicted parameters with their corresponding input sketches. Our results show that our procedural modeling-based approach is better compared to image-to-mesh or voxel, as the latter could generate rough and smooth surfaces, artifacts, holes, and deformed or unnecessary polygons. As our results are promising, we plan to continue to investigate the sketch-to-3D mesh approach based on procedural modeling and deep learning. Indeed, our approach is a baseline for the generation of a 3D tree model from a hand-made sketch. We proved that by giving the user some guidelines for drawing the sketch, the TSN obtained results consistent with the input sketch. Obviously, as also affirmed by Unlu et al. [2022], the approach has limitations once the user draws a tree very different from the sketches in the training set. We foresaw exploring the use of the different styles of sketches to make our approach more generalizable. In future work, we could expand the number of tree species and/or generate new 3D meshes by adapting our approach to other contexts where procedural modeling can be used, such as vases, chairs, buildings, furniture, etc. [Huang et al. 2017]. Also, in future work, we would like to define a method for texture generation, which would avoid the use of the tree species identification algorithm 1 that is used to select the most suitable texture from a predefined set. A possible method could be to consider colored input sketches and extract the texture or color from them. Furthermore, this approach could be used as a starting point for implementing an application for generating 3D trees or 3D environments in real-time. Another future work could use our approach to generate a 3D model from a sketch and use it to produce a video in cartoon style, making a mesh-to-image synthesis.

6.1 Discussion and Limitations

There are several reasons why our approach, which automates procedural modeling by predicting parameters from sketches, proves to be the best choice for generating complex 3D models, such as trees. One of the reasons is that approaches that directly predict mesh using images as DNN input often present qualitative problems [Gkioxari et al. 2019; Xie et al. 2019; Xu et al. 2019], such as rough and smooth surfaces, artifacts, holes, and deformed and unnecessary polygons, especially for a thin and layered structure like those of tree trunk and branches. Furthermore, the 3D meshes that are generated from direct methods often poorly resemble the input RGB images.
or sketches. Procedural modeling approaches for the generation of 3D meshes can overcome these problems [Liu et al. 2021b; Smelik et al. 2014; Wang et al. 2018a]. Another advantage is that the 3D mesh is always correct, as our robust RT Blender add-on correctly interprets the parameters and uses them, avoiding artifacts and error generation. The latter is demonstrated by the predictive accuracy of the parameters of our TSN, even in difficult conditions, as reported in Section 5.1. Although procedural modeling has only recently been explored in the field of deep learning and artificial intelligence [Liu et al. 2021b; Park et al. 2019; Yumer et al. 2015], there are, as yet, few specific approaches that examine prediction parameters for 3D non-linear content generation, either plant, in general, or trees [Li et al. 2021; Liu et al. 2021c], in particular. We compared our baseline (see Section 5.2) with better-known core nets, to assess the effectiveness of our work. To further assess the performance of TSN in terms of accuracy, we provided a quantitative analysis using the HDD metric. We also evaluated the coherence of the predicted parameters with sketches provided as input to the TSN through a qualitative visual analysis (see Section 5.4) of some parameters that are easier to understand and visually identified. In addition, we compared the 3D tree meshes generated from predicted parameters with ground-truth meshes using the Hausdorff distance, as reported in Section 5.3. To test the robustness of our method, we performed some additional experiments on challenging cases. In detail, we tested our TSN with sketch images that contain broken segments. Figure 17 shows two examples of tree sketches with deleted parts. In the first row of Figure 17 it can be seen that the reconstructed tree has fewer branches in the cropped region of the sketch than the ground truth. The row in Figure 17 shows the sketch of a pine with a missing ring. For this reason, TSN predicts a pine with one less ring than the ground truth. Figure 18 reports two outliers caused by the poor TSN accuracy concerning the provided input sketches drawn by the experiment participants (see Section 5.1). The first two images of Figure 18 represent an HM sketch of a maple tree with the relative 3D model. The sketch represents a tree with a few branches drawn with a very thin line. As a result, the TSN predicts a maple tree with few branches, placing leaves on them. For this reason, the crown of the output tree is barer than the sketch one, while the predicted branches are similar to the sketch ones. Another interesting behavior is shown in the last two images of Figure 18. Here, the tree branches are not well drawn, so the TSN tries to predict the tree's shape based on the information provided by the trunk and the crown. The result is a 3D mesh not very similar to the input sketch but consistent with the tree species and the main visible elements.

![Fig. 17. Results of TSN given sketches with missing regions.](image)

To get a more visible indication of the details of sketches that the TSN considers mandatory, we propose another experiment. This consists of testing the TSN robustness in relation to HM sketches with various Levels
Of Completeness (LOC). To further test our TSN, we decided to draw five maples with different LOCs because, with this species, the TSN returns less accurate results. Figure 19 shows the results of this experiment. Comparing the first two columns, TSN has more difficulty predicting the second-level branches if the crown is not present in the sketch and if the trunk and the branches have no thickness (second column). This is because the training set contains only sketches of crown trees whose branches are represented with different thicknesses concerning their level. For this reason, when the sketch in the second column is given as input to TSN, this does not understand at what level the branches belong. On the other hand, the sketches in the third and fourth columns have more information than the sketch in the second column. In particular, the sketch in the third column also contains the crown information, and the sketch in the fourth column has information about the thickness of the trunk and the branches. As a result, the TSN generates trees similar to the input sketch. From all of this, it follows that TSN is able to obtain better results with sketches that have the same drawing style as the sketch in the last column. Observing the results in Figure 19, it can be noticed that if the sketch contains a branch much more bent than the others, the TSN tends to even the curvature of the other branches.

Through the information obtained from the previous experiments, it is clear that it is necessary to provide participants with some guidelines for drawing HM sketches. Although our work provides a pipeline for the generation of 3D trees based on well-defined parameters, one of the limitations is the small number of tree species considered. However, this limitation could be easily overcome by considering more species in future scenarios. The proposed approach is based on the Weber-Penn method because it is stable and was implemented as a Blender add-on. Despite this, it could be adapted to other newer procedural modeling methods [Stava et al. 2014], thanks to the structure of the last part of TSN. It would be enough to sort each parameter of the new
methodology into the branch of the last layer of the network corresponding to the relative range of values and then retrain the network using the previously learned weights as a starting point for the new training.

7 ONLINE RESOURCES

We have shared the RT Blender plugin, DNN source code, and the trained weights through a GitHub repository: https://github.com/Unibas3D/TreeSketchNet. Furthermore, the authors are happy to share the training dataset upon request by sending an email to them. Finally, we have included an illustrative video on the GitHub page.

REFERENCES

Christopher M. Bishop. 1995. *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., USA.

John Canny. 1986. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-8*, 6 (1986), 679–698. https://doi.org/10.1109/TPAMI.1986.4787581

Nicola Capece, Francesco Banterle, Paolo Cignoni, Fabio Ganovelli, Roberto Scopigno, and Ugo Erra. 2019. Deepflash: Turning a flash selfie into a studio portrait. *Signal Processing: Image Communication* 77 (2019), 28–39.

Liang Cheng, Lihua Tong, Yanning Chen, Wen Zhang, Jie Shan, Yongxue Liu, and Manchun Li. 2013. Integration of LiDAR data and optical multi-view images for 3D reconstruction of building roofs. *Optics and Lasers in Engineering* 51, 4 (2013), 493–502.

Paolo Cignoni, Claudio Rocchini, and Roberto Scopigno. 1998. Metro: Measuring Error on Simplified Surfaces. *Computer Graphics Forum* 17, 2 (1998), 167–174. https://doi.org/10.1111/1467-8659.00236 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/1467-8659.00236

Zihang Dai, Hanxiao Liu, Quoc V Le, and Mingxing Tan. 2021. CoAtNet: Marrying Convolution and Attention for All Data Sizes. In *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (Eds.), Vol. 34. Curran Associates, Inc., On-line, 3965–3977. https://proceedings.neurips.cc/paper/2021/file/20568602db622456cc42a2e853ac2160-Paper.pdf

Stephanie Delalieux, Pablo J. Zarco-Tejada, Laurent Titis, Miguel Ángel Jiménez Bello, Diego S. Intrigliolo, and Ben Somers. 2014. Unmixing-Based Fusion of Hyperspatial and Hyperspectral Airborne Imagery for Early Detection of Vegetation Stress. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 7, 6 (2014), 2571–2582. https://doi.org/10.1109/JSTARS.2014.2330352

Johanna Delanoy, Mathieu Aubry, Philipp Isola, Alexei A Efros, and Adrien Boussau. 2018. 3D sketching using multi-view deep volumetric prediction. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 1, 1 (2018), 1–22.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. 2009. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. Ieee, Miami, FL, 248–255. https://doi.org/10.1109/CVPR.2009.5206848

Oliver Deussen and Bernd Lintermann. 2005. *Digital design of nature* computer generated plants and organsics. Springer Science & Business Media, Berlin.

Chao Ding and Ligang Liu. 2016. A survey of sketch based modeling systems. *Frontiers of Computer Science* 10, 6 (2016), 985–999.

David S Ebert, F Kenton Musgrave, Darwyn Peachey, Ken Perlin, and Steven Worley. 2002. *Texturing and Modeling: A Procedural Approach* (3rd ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Mathias Eitz, James Hays, and Marc Alexa. 2012. How Do Humans Sketch Objects? *ACM Trans. Graph.* 31, 4, Article 44 (jul 2012), 10 pages. https://doi.org/10.1145/2155330.2155340

Haoqiang Fan, Hao Su, and Leonidas J Guibas. 2017. A point set generation network for 3d object reconstruction from a single image. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. Ieee, Honolulu, HI, USA, 605–613.

Teodor Fredriksson, David Isa Mattos, Jan Bosch, and Helena Holmström Olsson. 2020. Data labeling: An empirical investigation into industrial challenges and mitigation strategies. In *International Conference on Product-Focused Software Process Improvement*. Springer, 202–216.

Sarah Frisken, Nazim Haouchine, Rose Du, and Alexandra J. Golby. 2022. Using temporal and structural data to reconstruct 3D cerebral vasculature from a pair of 2D digital subtraction angiography sequences. *Computerized Medical Imaging and Graphics* 99 (2022), 102076. https://doi.org/10.1016/j.compmedimag.2022.102076

Demetrios Gatzios, Jean F. Lienard, Andre Vogs, and Nikolay S. Strigul. 2015. 3D Tree Dimensionality Assessment Using Photogrammetry and Small Unmanned Aerial Vehicles. *PLOS ONE* 10, 9 (09 2015), 1–21. https://doi.org/10.1371/journal.pone.0137765

Georgia Gkioxari, Jitendra Malik, and Justin Johnson. 2019. Mesh R-CNN. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. Ieee, Seoul, Korea, 9784–9794. https://doi.org/10.1109/ICCV.2019.00988

Tim Golla, Tom Kneiphof, Heiner Kuhlmann, Michael Weinmann, and Reinhard Klein. 2020. Temporal upsampling of point cloud sequences by optimal transport for plant growth visualization. *Comput. Graph. Forum* 39, 6 (2020), 167–179.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press, USA. http://www.deeplearningbook.org.

Benoit Guillard, Edoardo Remelli, Pierre Yvernay, and Pascal Fua. 2021. Sketch2Mesh: Reconstructing and Editing 3D Shapes from Sketches. (2021).

ACM Trans. Intell. Syst. Technol.
Zhihao Liu, Kai Wu, Jianwei Guo, Yunhai Wang, Oliver Deussen, and Zhanglin Cheng. 2021c. Single Image Tree Reconstruction via Adversarial

Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. 2018. Foundations of machine learning

Fabio Remondino, Stefano Girardi, Alessandro Rizzi, and Lorenzo Conzo. 2009. 3D Modeling of Complex and Detailed Cultural Heritage

Ed Quigley, Winnie Lin, Yilin Zhu, and Ronald Fedkiw. 2021. Three Dimensional Reconstruction of Botanical Trees with Simulatable Geometry.
The algorithmic beauty of plants

Przemyslaw Prusinkiewicz and Aristid Lindenmayer. 2012.

Christophe Pradal, Frédéric Bondon, Christophe Nouguier, Jérôme Chopard, and Christophe Godin. 2009. PlantGL: A Python-based geometric library for 3D plant modelling at different scales. Graphical Models 71, 1 (2009), 1–21. https://doi.org/10.1016/j.gmod.2008.10.001

Przemyslaw Prusinkiewicz and Aristid Lindenmayer. 2012. The algorithmic beauty of plants. Springer Science & Business Media, New York.

Ed Quigley, Lin, Yilin Zhu, and Ronald Fedkiw. 2021. Three Dimensional Reconstruction of Botanical Trees with Simulatable Geometry.

ACM Trans. Intell. Syst. Technol.
Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. 2016. A survey of transfer learning. *Journal of Big Data* 3, 1 (28 May 2016), 9. https://doi.org/10.1186/s40537-016-0043-6

Haozhe Xie, Hongxun Yao, Xiaoshuai Sun, Shangchen Zhou, and Shengping Zhang. 2019. P2V2Vox: Context-Aware 3D Reconstruction From Single and Multi-View Images. In 2019 IEEE/CVF International Conference on Computer Vision (ICCV). IEEE Computer Society, Los Alamitos, CA, USA, 2690–2698. https://doi.org/10.1109/ICCV.2019.00278

Ke Xie, Feilong Yan, Andrei Sharf, Oliver Deussen, Hui Huang, and Baoquan Chen. 2016. Tree Modeling with Real Tree-Parts Examples. *IEEE Transactions on Visualization and Computer Graphics* 22, 12 (2016), 2608–2618. https://doi.org/10.1109/TVCG.2015.2513409

Qiangeng Xu, Weiyue Wang, Duygu Ceylan, Radomir Mech, and Ulrich Neumann. 2019. Disn: Deep implicit surface network for high-quality single-view 3d reconstruction. (2019).

Guandao Yang, Yin Cui, Serge Belongie, and Bharath Hariharan. 2018. Learning single-view 3d reconstruction with limited pose supervision. In *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer, Munich, Germany, 86–101.

Liuming Yang, Meng Yang, and Gang Yang. 2019. Modeling fractures and cracks on tree branches. *Computers & Graphics* 80 (2019), 63–72. https://doi.org/10.1016/j.cag.2019.03.006

Xiangli Yang, Zixing Song, Irwin King, and Zenglin Xu. 2021. A Survey on Deep Semi-supervised Learning. (2021).

Mehmet Ersin Yumer, Paul Asente, Radomir Mech, and Levent Burak Kara. 2015. Procedural Modeling Using Autoencoder Networks. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology* (Charlotte, NC, USA) (UIST ’15). Association for Computing Machinery, New York, NY, USA, 109–118. https://doi.org/10.1145/2807442.2807448

Yang Zhang, Zhen Liu, Tianpeng Liu, Bo Peng, and Xiang Li. 2019. RealPoint3D: An Efficient Generation Network for 3D Object Reconstruction From a Single Image. *IEEE Access* 7 (2019), 57539–57549. https://doi.org/10.1109/ACCESS.2019.2914150

Qian-Yi Zhou and Vladlen Koltun. 2014. Color Map Optimization for 3D Reconstruction with Consumer Depth Cameras. *ACM Trans. Graph.* 33, 4, Article 155 (July 2014), 10 pages. https://doi.org/10.1145/2601097.2601134

Changqing Zou, Qian Yu, Ruofei Du, Haoran Mo, Yi-Zhe Song, Tao Xiang, Chengying Gao, Baoquan Chen, and Hao Zhang. 2018. SketchyScene: Richly-Annotated Scene Sketches. In *Computer Vision – ECCV 2018*, Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss (Eds.). Springer International Publishing, Cham, 438–454

Chuhang Zou, Ersin Yumer, Jimei Yang, Duygu Ceylan, and Derek Hoiem. 2017. 3D-PRNN: Generating Shape Primitives with Recurrent Neural Networks. In *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE Computer Society, Los Alamitos, CA, USA, 900–909. https://doi.org/10.1109/ICCV.2017.103

ACM Trans. Intell. Syst. Technol.