Shape Synthesis from Sketches via Procedural Models and Convolutional Networks

Haibin Huang, Evangelos Kalogerakis, Ersin Yumer, Radomir Mech

Abstract—
Procedural modeling techniques can produce high quality visual content through complex rule sets. However, controlling the outputs of these techniques for design purposes is often notoriously difficult for users due to the large number of parameters involved in these rule sets and also their non-linear relationship to the resulting content. To circumvent this problem, we present a sketch-based approach to procedural modeling. Given an approximate and abstract hand-drawn 2D sketch provided by a user, our algorithm automatically computes a set of procedural model parameters, which in turn yield multiple, detailed output shapes that resemble the user's input sketch. The user can then select an output shape, or further modify the sketch to explore alternative ones. At the heart of our approach is a deep Convolutional Neural Network (CNN) that is trained to map sketches to procedural model parameters. The network is trained by large amounts of automatically generated synthetic line drawings. By using an intuitive medium i.e., freehand sketching as input, users are set free from manually adjusting procedural model parameters, yet they are still able to create high quality content. We demonstrate the accuracy and efficacy of our method in a variety of procedural modeling scenarios including design of man-made and organic shapes.

Index Terms—shape synthesis, convolutional neural networks, procedural modeling, sketch-based modeling

1 Introduction

Procedural Modeling (PM) allows synthesis of complex and non-linear phenomena using conditional or stochastic rules [1], [2], [3]. A wide variety of 2D or 3D models can be created with PM e.g., vases, jewelry, buildings, trees, to name a few [4]. PM frees users from direct geometry editing and helps them to create a rich set of unique instances by manipulating various parameters in the rule set. However, due to the complexity and stochastic nature of rule sets, the underlying parametric space of PM is often very high-dimensional and nonlinear, making outputs difficult to control through direct parameter editing. PM is therefore not easily approachable by non-expert users, who face various problems such as where to start in the parameter space and how to adjust the parameters to reach outputs that match their intent. We address this problem by allowing users to perform PM through freehand sketching rather than directly manipulating high-dimensional parameter spaces. Sketching is often a more natural, intuitive and simpler way for users to communicate their intent.

We introduce a technique that takes 2D freehand sketches as input and translates them to corresponding PM parameters, which in turn yield detailed shapes. Examples of input sketches and output shapes are shown in Figure 1. The users of our technique are not required to have artistic and professional skills in drawing. We aim to synthesize PM parameters from approximate, abstract sketches drawn by casual modelers who are interested in quickly conveying design ideas. A main challenge in recognizing such sketches and converting them to high quality visual content is the fact that humans often perform dramatic abstractions, simplifications and exaggerations to convey the shape of objects [5]. Developing an algorithm that factors out these exaggerations, is robust to simplifications and approximate line drawing, and captures the variability of all possible abstractions of an object is far from a trivial task. Hand-designing an algorithm and manually tuning all its internal parameters or thresholds to translate sketch patterns to PM outputs seems extremely hard and unlikely to handle the enormous variability of sketch inputs.

We resort to a *machine learning* approach that automatically learns the mapping from sketches to PM parameters from a large corpus of training data. Collecting training human sketches relevant to given PM rule sets is hard and time-consuming. We instead automatically generate *synthetic training data* to train our algorithm. We exploit key properties of PM rule sets to generate the synthetic datasets. We simplify PM output shapes based on structure, density, repeated patterns and symmetries to simulate abstractions and simplifications found in human sketches. Given the training data, the mapping from sketches to PM parameters is also far from trivial to learn. We found that common classifiers and regression functions used in sketch classification and sketch-based retrieval [6], [7], such as Support Vector Machines, Nearest Neighbors or Radial Basis Function interpolation, are largely inadequate to reliably predict PM parameters. We instead utilize a *deep Convolutional Neural Network* (CNN) architecture to map sketches to PM parameters. CNNs trained on large datasets have demonstrated large success in object detection, recognition, and classification tasks [8], [9], [10], [11], [12]. Our key insight is that CNNs are able to capture the complex and non-linear relationships between sketches and PM parameters through their hierarchical network structure and learned,

• *Ersin Yumer and Radomir Mech are with Adobe Research. Haibin Huang and Evangelos Kalogerakis are with University of Massachusetts Amherst.*

Manuscript received February 29, 2016; accepted July 29, 2016.

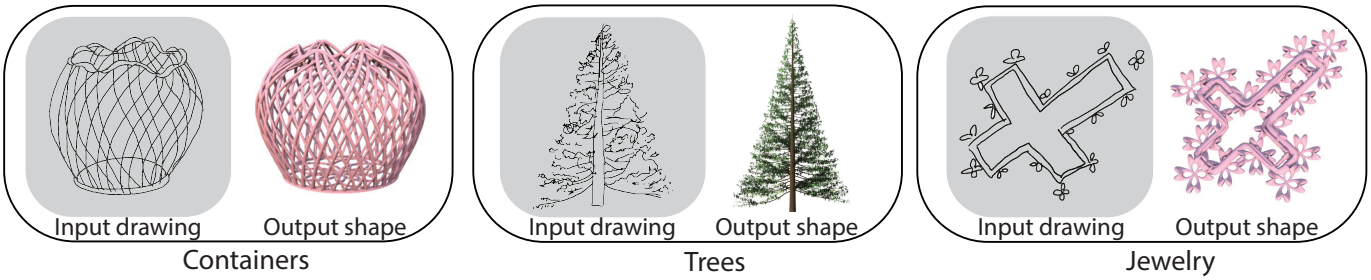

Fig. 1: Given freehand sketches drawn by casual modelers, our method learns to synthesize procedural model parameters that yield detailed output shapes.

multi-resolution image filters that can be optimized for PM parameter synthesis.

Since the input sketches represent abstractions and simplifications of shapes, they often cannot be unambiguously mapped to a single design output. Through the CNN, our algorithm provides *ranked, probabilistic outputs*, or in other words *suggestions* of shapes, that users can browse and select the ones they prefer most.

Contributions. Our main contribution is two-fold. First, we introduce a procedural modeling technique for 2D/3D shape design using human sketches as input, without the need for direct, manual parameter editing. Second, we provide a method to generate input sketches from procedural models simulating aspects of simplifications and exaggerations found in human sketches, making our system scalable and improving the generalization ability of machine learning algorithms to process human sketches. Our qualitative and quantitative evaluation demonstrates that our approach outperforms state-of-the-art methods on sketch-based retrieval in the context of procedural modeling.

2 Related Work

Our work is related to prior work in procedural modeling, in particular targeted design of procedural models and exploratory procedural modeling techniques, as well as sketch-based shape retrieval and convolutional neural networks we discuss in the following paragraphs.

Procedural Modeling. Procedural models were used as early as in sixties for biological modeling based on L-systems [13]. L-systems were later extended to add geometric representations [14], parameters, context, or environmental sensitivity to capture a wide variety of plants and biological structures [15], [16]. Procedural modeling systems were also used to generate shapes with shape grammars [17], [18], for modeling cities [1], buildings [3], furniture arrangements [19], building layouts [20], and lighting design [21].

Procedural models often expose a set of parameters that can control the resulting visual content. Due to the recursive nature of the procedural model, some of those parameters often have complex, aggregate effects on the resulting geometry. On one hand, this is an advantage of procedural models i.e., an unexpected result emerges form a given set of parameters. On another hand, if a user has a particular design in mind, recreating it using these parameters results in a tedious modeling experience. To address this problem, there has been some research focused on targeted design and exploratory systems to circumvent the direct interaction with PM parameters.

Targeted design of procedural models. Targeted design platforms free users from interacting with PM parameters. Lintermann and Deussen [22] presented an interactive PM system where conventional PM modeling is combined with free-form geometric modeling for plants. McCrae and Singh [23] introduced an approach for converting arbitrary sketch strokes to 3D roads that are automatically fit to a terrain. Vanegas et al. [24] perform inverse procedural modeling by using Monte Carlo Markov Chains (MCMC) to guide PM parameters to satisfy urban design criteria. Stava et al. [25] also use a MCMC approach to tree design. Their method optimizes trees, generated by L-systems, to match a target tree polygonal model. Talton et al. [26] presented a more general method for achieving high-level design goals (e.g., city skyline profile for city procedural modeling) using inverse optimization of PM parameters based on a Reversible Jump MCMC formulation so that the resulting model conforms to design constraints. MCMC-based approaches receive control feedback from the completely generated models, hence suffer from significantly higher computational cost at run-time. Alternative approaches incrementally receive control feedback from intermediate states of the PM based on Sequential Monte Carlo, allowing them to reallocate computational resources and converge more quickly [27].

In the above kinds of systems, users prescribe target models, indicator functions, silhouettes or strokes, but their control over the rest of the shape or its details is very limited. In the case of inverse PM parameter optimization (e.g., [26]) producing a result often requires significant amount of computation (i.e., several minutes or hours). In contrast, users of our method have direct control over the output shape and its details based on their input sketch. Our approach trivially requires a single forward pass in a CNN network at run-time, hence resulting in significantly faster computation for complex procedural models, and producing results at near interactive rates.

Concurrently to our work, Nishida et al. [28] introduced a CNN-based urban procedural model generation from sketches. However, instead of solving directly for the final shape, their method suggests potentially incomplete parts that require further user input to produce the final shape. In contrast, our approach is end-to-end, requiring users to provide only an approximate sketch of the whole shape.

Exploratory systems for procedural models. Exploratory systems provide the user with previously computed and sorted exemplars that help users study the variety of models and select seed models they wish to further explore. Talton et al. [29] organized a set of precomputed

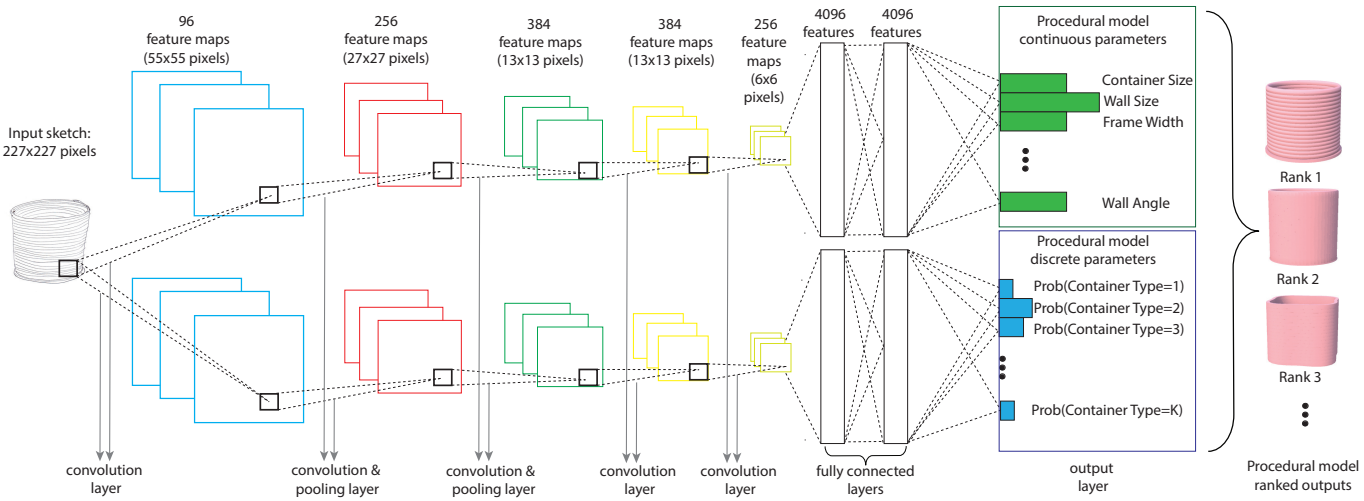

Fig. 2: Convolutional Neural Network (CNN) architecture used in our method. The CNN takes as input a sketch image and produces a set of PM parameters, which in turn yield ranked design outputs.

samples in a 2D map. The model distribution in the map is established by a set of landmark examples placed by expert users of the system. Lienhard et al. [30] sorted precomputed sample models based on a set of automatically computed views and geometric similarity. They presented the results with rotational and linear thumbnail galleries. Yumer et al. [31] used autoencoder networks to encode the high dimensional parameter spaces into a lower dimensional parameter space that captures a range of geometric features of models.

A problem with these exploratory techniques is that users need to have an exact understanding of the procedural model space to find a good starting point. As a result, it is often hard for users to create new models with these techniques.

Sketch-based shape retrieval. Our work is related to previous methods for retrieving 3D models from a database using sketches as input using various matching strategies to compare the similarity of sketches to database 3D models [6], [32], [33], [34], [35]. However, these systems only allow retrieval of existing 3D models and provide no means to create new 3D models. Our method is able to synthesize new outputs through PM.

Convolutional Neural Networks. Our work is based on recent advances in object recognition with deep CNNs [36]. CNNs are able to learn hierarchical image representations optimized for image processing performance. CNNs have demonstrated large success in many computer vision tasks, such as object detection, scene recognition, texture recognition and fine-grained classification [8], [9], [10], [11], [36]. Sketch-based 3D model retrieval has also been recently demonstrated through CNNs. Su et al. [12] performed sketch-based shape retrieval by adopting a CNN architecture pre-trained on images, then fine-tuning it on a dataset of sketches collected by human volunteers [5]. Wang et al. [37] used a Siamese CNN architecture to learn a similarity metric to compare human and computer generated line drawings. In contrast to these techniques, we introduce a CNN architecture capable of generating PM parameters, which in turn yield new 2D or 3D shapes.

3 Overview

Our algorithm aims to learn a mapping from input approximate, abstract 2D sketches to the PM parameters of a given rule set, which in turn yield 2D or 3D shape suggestions. For example, given a rule set that generates trees, our algorithm produces a set of discrete (categorical) parameters, such as tree family, and continuous parameters, such as trunk width, height, size of leaves and so on. Our algorithm has two main stages: a *training stage*, which is performed offline and involves training a CNN architecture that maps from sketches to PM parameters, and a *runtime synthesis stage*, during which a user provides a sketch, and the CNN predicts PM parameters to synthesize shape suggestions presented back to the user. We outline the key components of these stages below.

CNN architecture. During the training stage, a CNN is trained to capture the highly non-linear relationship between the input sketch and the PM parameter space. A CNN consists of several inter-connected "layers" (Figure 2) that process the input sketch hierarchically. Each layer produces a set of feature representations maps, given the maps produced in the previous layer, or in the case of the first layer, the sketch image. The CNN layers are "convolutional", "pooling', or "fully connected layers". A convolutional layer consists of learned filters that are convolved with the input feature maps of the previous layer (or in the case of the first convolutional layer, the input sketch image itself). A pooling layer performs subsampling on each feature map produced in the previous layer. Subsampling is performed by computing the max value of each feature map over spatial regions, making the feature representations invariant to small sketch perturbations. A fully connected layer consists of non-linear functions that take as input all the local feature representations produced in the previous layer, and non-linearly transforms them to a global sketch feature representation.

In our implementation, we adopt a CNN architecture widely used in computer vision for object recognition, called AlexNet [36]. AlexNet contains five convolutional layers, two pooling layers applied after the first and second convolutional layer, and two fully connected layers. Our CNN

architecture is composed of two processing paths (or sub-networks), each following a distinct AlexNet CNN architecture (Figure 2). The first sub-network uses the AlexNet set of layers, followed by a regression layer, to produce the continuous parameters of the PM. The second sub-network uses the AlexNet set of layers, followed by a classification layer, to produce probabilities over discrete parameter values of the PM. The motivation behind using these two sub-networks is that the the continuous and discrete PM parameters are predicted more effectively when the CNN feature representations are optimized for classification and regression separately, as opposed to using the same feature representations for both the discrete and continuous PM parameters. Using a CNN versus other alternatives that process the input in one stage (i.e., "shallow" classifiers or regressors, such as SVMs, nearest neighbors, RBF interpolation and so on) also proved to be much more effective in our experiments. We describe the CNN processing layers in Section 4.1 in more detail, and quantitative comparisons with alternatives in Section 5.

CNN training. As in the case of deep architectures for object recognition in computer vision, training the CNN requires optimizing millions of weights (110 million weights in our case). Training a CNN with fewer number of layers decreases the PM parameter synthesis performance. We leverage available massive image datasets widely used in computer vision, as well as synthetic sketch data that we generated automatically. Specifically, the convolutional and fully connected layers of both sub-networks are first pre-trained on ImageNet [38] (a publically available image dataset containing 1.2 million photos) to perform generic object recognition. Then each sub-network is further fine-tuned for PM discrete and continuous parameter synthesis based on our synthetic sketch dataset. We note that adapting a network trained for one task (object recognition from images) to another task (PM-based shape synthesis from sketches) can be seen as an instance of transfer learning [39]. We describe the CNN training procedure in Section 4.2.

Synthetic training sketch generation. To train our CNN, we could generate representative shapes by sampling the parameters of the PM rule set, then ask human volunteers to draw sketches of these shapes. This approach would provide us training data with sketches and corresponding PM parameters. However, such approach would require intensive human labor and would not be scalable. Instead, we followed an automatic approach. We conducted an informal user study to gain insight how people tend to draw shapes generated by PMs. We randomly sampled a few shapes generated by PM rules for containers, jewelry and trees, then asked a number of people to provide freehand drawings of them. Representative sketches and corresponding shapes are shown in Figure 3. The sketches are abstract, approximate with noisy contours, as also observed in previous studies on how humans draw sketches [5]. We also found that users tend to draw repetitive patterns only partially. For example, they do not draw all the frame struts in containers, but a subset of them and with varying thickness (i.e., spacing between the outlines of struts). We took into account this observation while generating synthetic sketches. We sampled thousands of shapes for each PM rule set, then for each of them, we generated automatically several different line drawings, each with progressively subsampled repeated patterns and varying thickness for their components. The synthetic dataset generation is described in Section 4.2.

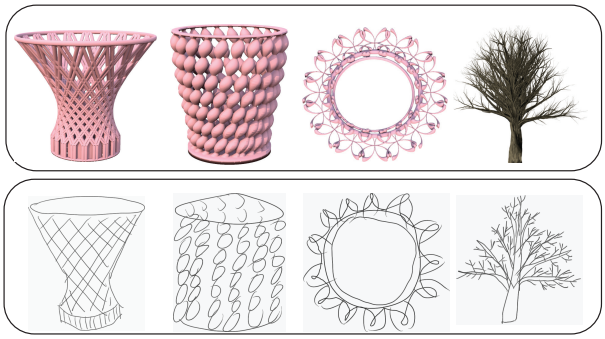

Fig. 3: Freehand drawings (bottom row) created by users in our user study. The users were shown the corresponding shapes of the top row.

4 METHOD

We now describe the CNN architecture we used for the PM parameter synthesis, then we explain the procedure for training the CNN, and finally our runtime stage.

4.1 CNN architecture

Given an input sketch image, our method processes it through a neural network composed of convolutional and pooling layers, fully connected layers, and finally a regression layer to produce PM continuous parameter values. The same image is also processed by a second neural network with the same sequence of layers, yet with different learned filters and weights, and a classification (instead of regression) layer to produce PM discrete parameter values. We now describe the functionality of each type of layer. Implementation details are provided in the Appendix.

Convolutional layers. Each convolutional layer yields a stack of feature maps by applying a set of learned filters that are convolved with the feature representations produced in the previous layer. As discussed in previous work in computer vision [8], [9], [10], [11], [36], after training, each filter often becomes sensitive to certain patterns observed in the input image, or in other words yield high responses in their presence.

Pooling layers. Each pooling layer subsamples each feature map produced in the previous layer. Subsampling is performed by extracting the maximum value within regions of each input feature map, making the resulting output feature representation invariant to small sketch perturbations. Subsampling also allows subsequent convolutional layers to efficiently capture information originating from larger regions of the input sketch.

Fully Connected Layers. Each fully connected layer is composed of a set of learned functions (known as "neurons" or "nodes" in the context of neural networks) that take as input all the features produced in the previous layer and non-linearly transforms them in order to produce a global sketch representation. The first fully connected layer following a convolutional layer concatenates ("unwraps")

the entries of all its feature maps into a single feature vector. Subsequent fully connected layers operate on the feature vector produced in their previous fully connected layer. Each processing function k of a fully connected layer l performs a non-linear transformation of the input feature vector as follows:

$$h_{k,l} = \max(\mathbf{w}_{k,l} \cdot \mathbf{h}_{l-1} + b_{k,l}, 0) \quad (1)$$

where $\mathbf{w}_{k,l}$ is a learned weight vector, $\mathbf{h}_{l-1}$ is the feature vector originating from the previous layer, $b_{k,l}$ is a learned bias weight, and $\cdot$ denotes dot product here. Concatenating the outputs from all processing functions of a fully connected layer produces a new feature vector that is used as input to the next layer.

Regression and Classification Layer. The feature vector produced in the last fully connected layer summarizes the captured local or global patterns in the input image. As shown in prior work in computer vision, the final feature vector can be used for image classification, object detection, or texture recognition [8], [9], [10], [11], [36]. In our case, we use this feature vector to predict continuous or discrete PM parameters.

To predict continuous PM parameters, the top sub-network of Figure 2 uses a regression layer following the last fully connected layer. The regression layer consists of processing functions, each taking as input the feature vector of the last fully connected layer and non-linearly transforming it to predict each continuous PM parameter. We use a sigmoid function to perform this non-linear transformation, which worked well in our case:

$$O_c = \frac{1}{1 + \exp(-\mathbf{w}_c \cdot \mathbf{h}_L - b_c)} \quad (2)$$

where O_c is the predicted value for the PM continuous parameter c, $\mathbf{w}_c$ is a vector of learned weights, $\mathbf{h}_L$ is the feature vector of the last fully connected layer, and b_c is the learned bias for the regression. We note that all our continuous parameters are normalized within the $[0, 1]$ interval.

To predict discrete parameters, the bottom sub-network of Figure 2 uses a classification layer after the last fully connected layer. The classification layer similarly consists of processing functions, each taking as input the feature vector of the last fully connected layer and non-linearly transforming it towards a probability for each possible value d of each discrete parameter D_r ($r = 1...R$, where R is the total number of discrete parameters). We use a softmax function to predict these probabilities, as commonly used in multi-class classification methods:

$$Prob(D_r = d) = \frac{\exp(\mathbf{w}_{d,r} \cdot \mathbf{h}_L + b_{d,r})}{\exp(\sum_{d'} \mathbf{w}_{d',r} \cdot \mathbf{h}_L + b_{d',r})} \quad (3)$$

where d' represent the rest of the discrete values of that parameter, $\mathbf{w}_{d,r}$ is a vector of learned weights, and $b_{d,r}$ is the learned bias for classification.

4.2 Training

Given a training dataset of sketches, the goal of our training stage is to estimate the internal parameters (weights) of the convolutional, fully connected, regression and classification layers of our network such that they reliably synthesize PM parameters of a given rule set. There are two sets of trainable weights in our architecture. First, we have the set of weights for the sub-network used for regression $\boldsymbol{\theta}_1$, which includes the regression weights $\{\mathbf{w}_c, b_c\}$ (Equation 2) per each PM continuous parameter and the weights used in each convolutional and fully connected layer. Similarly, we have a second set of weights for the sub-network used in classification $\boldsymbol{\theta}_2$, which includes the classification weights $\{\mathbf{w}_{d,r}, b_{d,r}\}$ (Equation 3) per each PM discrete parameter value, and the weights of its own convolutional and fully connected layers. We first describe the learning of the weights $\boldsymbol{\theta}_1$, $\boldsymbol{\theta}_2$ given a training sketch dataset, then we discuss how such dataset was generated automatically in our case.

CNN learning. Given a training dataset of S synthetic sketches with reference ("ground-truth") PM parameters for each sketch, we estimate the weights $\boldsymbol{\theta}_1$ such that the deviation between the reference and predicted continuous parameters from the CNN is minimized. Similarly, we estimate the weights $\boldsymbol{\theta}_2$ such that the disagreement between the reference and predicted discrete parameter values from the CNN is minimized. In addition, we want our CNN architecture to generalize to sketches not included in the training dataset. To prevent over-fitting our CNN to our training dataset, we "pre-train" the CNN in a massive image dataset for generic object recognition, then we also regularize all the CNN weights such that their resulting values are not arbitrarily large. Arbitrarily large weights in classification and regression problems usually yield poor predictions for data not used in training [40].

The cost function we used to penalize deviation of the reference and predicted continuous parameters as well as arbitrarily large weights is the following:

$$E_r(\boldsymbol{\theta}_1) = \sum_{s=1}^{S} \sum_{c=1}^{C} [\delta_{c,s} == 1] \|O_{c,s}(\boldsymbol{\theta}_1) - \hat{O}_{c,s}\|^2 + \lambda_1 ||\boldsymbol{\theta}_1||^2 \quad (4)$$

where C is the number of the PM continuous parameters, $O_{c,s}$ is the predicted continuous parameter c for the training sketch s based on the CNN, $\hat{O}_{c,s}$ is the corresponding reference parameter value, and $[\delta_{c,s} == 1]$ is a binary indicator function which is equal to 1 when the parameter c is available for the training sketch s, and 0 otherwise. The reason for having this indicator function is that not all continuous parameters are shared across different types (classes) of shapes generated by the PM. The regularization weight λ_1 (also known as weight decay in the context of CNN training) controls the importance of the second term in our cost function, which serves as a regularization term. We set $\lambda_1 = 0.0005$ through grid search within a validation subset of our training dataset.

We use the logistic loss function [40] to penalize predictions of probabilities for discrete parameter values that disagree with the reference values, along with a regularization term as above:

$$E_c(\boldsymbol{\theta}_2) = -\sum_{s=1}^{S} \sum_{r=1}^{R} ln(Prob(D_{s,r} = \hat{d}_{s,r}; \boldsymbol{\theta}_2)) + \lambda_2 ||\boldsymbol{\theta}_2||^2 \quad (5)$$

where R is the number of the PM discrete parameters, $Prob(D_{s,r} = \hat{d}_{s,r}; \boldsymbol{\theta}_2)$ is the output probability of the CNN

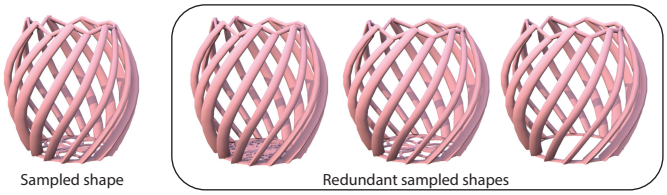

Fig. 4: Examples of highly redundant shapes removed from our training dataset.

for a discrete parameter r for a training sketch s, and $\hat{d}_{s,r}$ is the reference value for that parameter. We also set $\lambda_2 = 0.0005$ through grid search.

We minimize the above objective functions to estimate the weights $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ through stochastic gradient descent with step rate 0.0001 for $\boldsymbol{\theta}_1$ updates, step rate 0.01 for $\boldsymbol{\theta}_2$, and batch size of 64 training examples. The step rates are set empirically such that we achieve smoother convergence (for regression, we found that the step size should be much smaller to ensure convergence). We also use the dropout technique [41] during training that randomly excludes nodes along with its connections in the CNN with a probability 50% per each gradient descent iteration. Dropout has been found to prevent co-adaptation of the functions used in the CNN (i.e., prevents filters taking same values) and improves generalization [41].

Pre-training and fine-tuning. To initialize the weight optimization, one option is to start with random values for all weights. However, this strategy seems extremely prone to local undesired minima as well as over-fitting. We instead initialize all the weights of the convolutional and fully connected layers from the AlexNet weights [36] trained in the ImageNet1K dataset [38] (1000 object categories and 1.2M images). Even if the weights in AlexNet were trained for a different task (object classification in images), they already capture patterns (e.g., edges, circles etc) that are useful for recognizing sketches. Starting from the AlexNet weights is an initialization strategy that has also been shown to work effectively in other tasks as well (e.g., 3D shape recognition, sketch classification [12], [39]). We initialize the rest of the weights in our classification and regression layers randomly according to a zero-mean Gaussian distribution with standard deviation 0.01. Subsequently, all weights across all layers of our architecture are trained (i.e., fine-tuned) on our synthetic dataset. Specifically, we first fine-tune the sub-network for classification, then we fine-tune the sub-network for regression using the fine-tuned parameters of the convolutional and fully connected layers of the classification sub-network as a starting point. The difference in PM parameter prediction performance between using a random initialization for all weights versus starting with the AlexNet weights is significant (see experiments in Section 5).

Synthetic training sketch generation. To train the weights of our architecture, we need a training dataset of sketches, along with reference PM parameters per sketch. We generate such dataset automatically as follows. We start by generating a collection of shapes based on the PM rule set. To generate a collection that is representative enough of the shape variation that can be created through the PM set, we sample the PM continuous parameter space through Poisson disk sampling for each combination of PM discrete parameter values. We note that the number of discrete parameters representing types or styles of shapes in PMs is usually limited (no more than 2 in our rule sets), allowing us to try each such combination. This sampling procedure can still yield shapes that are visually too similar to each other. This is because large parameter changes can still yield almost visually indistinguishable PM outputs. We remove redundant shapes in the collection that do not contribute significantly to the CNN weight learning and unnecessarily increase its training time. To do this, we extract image-based features from rendered views of the shapes using the last fully connected layer of the AlexNet [36], then we remove shapes whose nearest neighbors based on their image features in any view is smaller than a conservative threshold we chose empirically (we calculate the average feature distance between all pairs of nearest neighboring samples, and set the threshold to 3 times of this distance.). Figure 4 shows examples of highly redundant shapes removed from our collection.

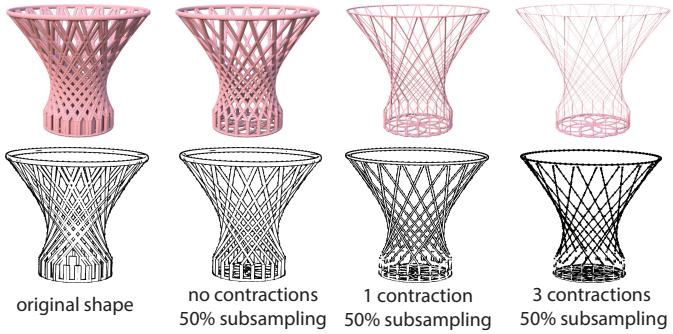

Fig. 5: Synthetic sketch variations (bottom row) generated for a container training shape. The sketches are created based on symmetric pattern sub-sampling and mesh contractions on the container shape shown on the top.

For each remaining shape in our collection, we generate a set of line drawings. We first generate a 2D line drawing using contours and suggestive contours [42] from a set of views per 3D shape. For shapes that are rotationally symmetric along the upright orientation axis, we only use a single view (we assume that all shapes generated by the PM rule set have a consistent upright orientation). In the case of 2D shapes, we generate line drawings through a Canny edge detector. Then we generate additional line drawings by creating variations of each 2D/3D shape as follows. We detect groups of symmetric components in the shape, then we uniformly remove half of the components per group. Removing more than half of the components degraded the performance of our system, since an increasingly larger number of training shapes tended to have too similar, sparse drawings. For the original and decimated shape, we also perform mesh contractions through the skeleton extraction method described in [43] using 1, 2 and 3 constrained Laplacian smoothing iterations. As demonstrated in Figure 5, the contractions yield patterns with varying thickness (spacing between contours of the same component).

The above procedure generate shapes with sub-sampled symmetric patterns and varying thickness for its components. The resulting line drawings simulate aspects of human line drawings of PM shapes. As demonstrated in Figure 3, humans tend to draw repetitive components only

partially and with varying thickness, sometimes using only a skeleton. Figure 5 shows the shape variations generated with the above procedure for the leftmost container of Figure 3, along with the corresponding line drawings. Statistics for our training dataset per rule set is shown in Table 1.

4.3 Runtime stage

The trained CNN acts as a mapping between a given sketch and PM parameters. Given a new user input sketch, we estimate the PM continuous parameters and probabilities for the PM discrete parameters through the CNN. We present the user with a set of shapes generated from the predicted PM continuous parameters, and discrete parameter values ranked by their probability. Our implementation is executed on the GPU. Predicting the PM parameters from a given sketch takes 1 to 2 seconds in all our datasets using a NVidia Tesla K40 GPU. Responses are not real-time in our current implementation based on the above GPU. Yet, users can edit or change their sketch, explore different sketches, and still get visual feedback reasonably fast at near-interactive rates.

5 Results

We now discuss the qualitative and quantitative evaluation of our method. We first describe our datasets used in the evaluation, then discuss a user study we conducted in order to evaluate how well our method maps human-drawn freehand sketches to PM outputs.

5.1 Datasets

We used three PM rule sets in our experiments: (a) 3D containers, (b) 3D jewelry, (c) 2D trees. All rule sets are built using the Deco framework [44]. The PM rule set for containers generates 3D shapes using vertical rods, generalized cylinders and trapezoids on the side walls, and a fractal geometry at the base. The PM rule set for jewelry generates shapes in two passes: one pass generates the main jewelry shape and the next one decorates the outline of the shape. The PM rule set for trees is currently available in Adobe Photoshop CC 2015. Photoshop includes a procedural engine that generates tree shapes whose parameters are controlled as any other Photoshop's filter.

For each PM rule set, we sample training shapes, then generate multiple training sketches per shape according to the procedure described in Section 4.2. The number of training shapes and line drawings in our synthetic sketch dataset per PM rule set is summarized in Table 1. We also report the number of continuous parameters, the number of discrete parameters, and total number of different discrete parameter values (i.e., number of classes or PM grammar variations) for each dataset. As shown in the table, all three rule sets contain several parameters. Tuning these parameters by hand (e.g., with a slider per each parameter) would not be ideal especially for novice users. In the same table, we also report the total time to train our architecture and time to process a sketch during the runtime stage.

Statistics	Containers	Trees	Jewelry
# training shapes	$30k$	$60k$	$15k$
# training sketches	$120k$	$240k$	$60k$
# continuous parameters	24	20	15
# discrete parameters	1	1	2
# discrete parameter values/classes	27	34	13
training time (hours)	12	20	9
runtime stage time (sec)	1.5	1.6	1.2

TABLE 1: Dataset statistics

5.2 User study

We conducted a user study to evaluate the performance of our method on human line drawings. We presented 15 volunteers a gallery of example shapes generated by each of our PM rule sets (not included in the training datasets), then asked them to provide us with a freehand drawing of a given container, jewelry piece, and tree. None of the volunteers had professional training in arts or 3D modeling. We collected total 45 drawings (15 sketches per dataset). Each drawing had associated PM continuous and discrete parameters based on the used reference shape from the gallery. In this manner, we can evaluate quantitatively how reliably our method or alternative methods are able to synthesize shapes that users intend to convey through their line drawing. We compare the following methods:

Nearest neighbors. The simplest technique to predict PM parameters is nearest neighbors. For each input human drawing, we extract a popular feature representation, retrieve the nearest synthetic sketch in that feature space using Euclidean distances, then generate a shape based on this nearest retrieved sketch PM parameters. We used the Fisher vector representation to represent sketches, which has recently been shown to outperform several other baseline feature representations in sketch-based shape retrieval [7].

SVM classification and RBF interpolation. Instead of nearest neighbors, SVMs can be used for sketch classification based on the same Fisher vector representations, as suggested in [7]. To predict continuous parameters, we used RBF interpolation again on Fisher vector representations. Here we use LIBSVM [45] for SVM classification and Matlab's RBF interpolation.

Standard CNNs. Instead of using Fisher vector representations, an alternative approach is to use a standard CNN trained on an image database as-is, extract a feature representation from the last fully connected layer, then use it in nearest neighbor or SVM classification, and RBF interpolation for regression. We used the features from the last fully connected layer of the AlexNet CNN trained on ImageNet1K.

Single CNN. Instead of using two sub-networks with distinct weights, one can alternatively train a single CNN network with shared filters and weights followed by a regression and a classification layer (i.e., classification and regression rely on the same feature representation extracted by the last fully connected layer).

No pre-training. We tested our proposed architecture of Figure 2 without the pre-training procedure. Instead of pre-training, we initialized all the CNN weights based on the random initialization procedure of [36], then trained the whole network from scratch.

Our method. We tested our proposed architecture including the pre-training procedure described in Section 4.2.

In Table 2, we report the classification accuracy i.e., percentage of times that the reference discrete parameter value (class) agrees with the top ranked discrete value predicted

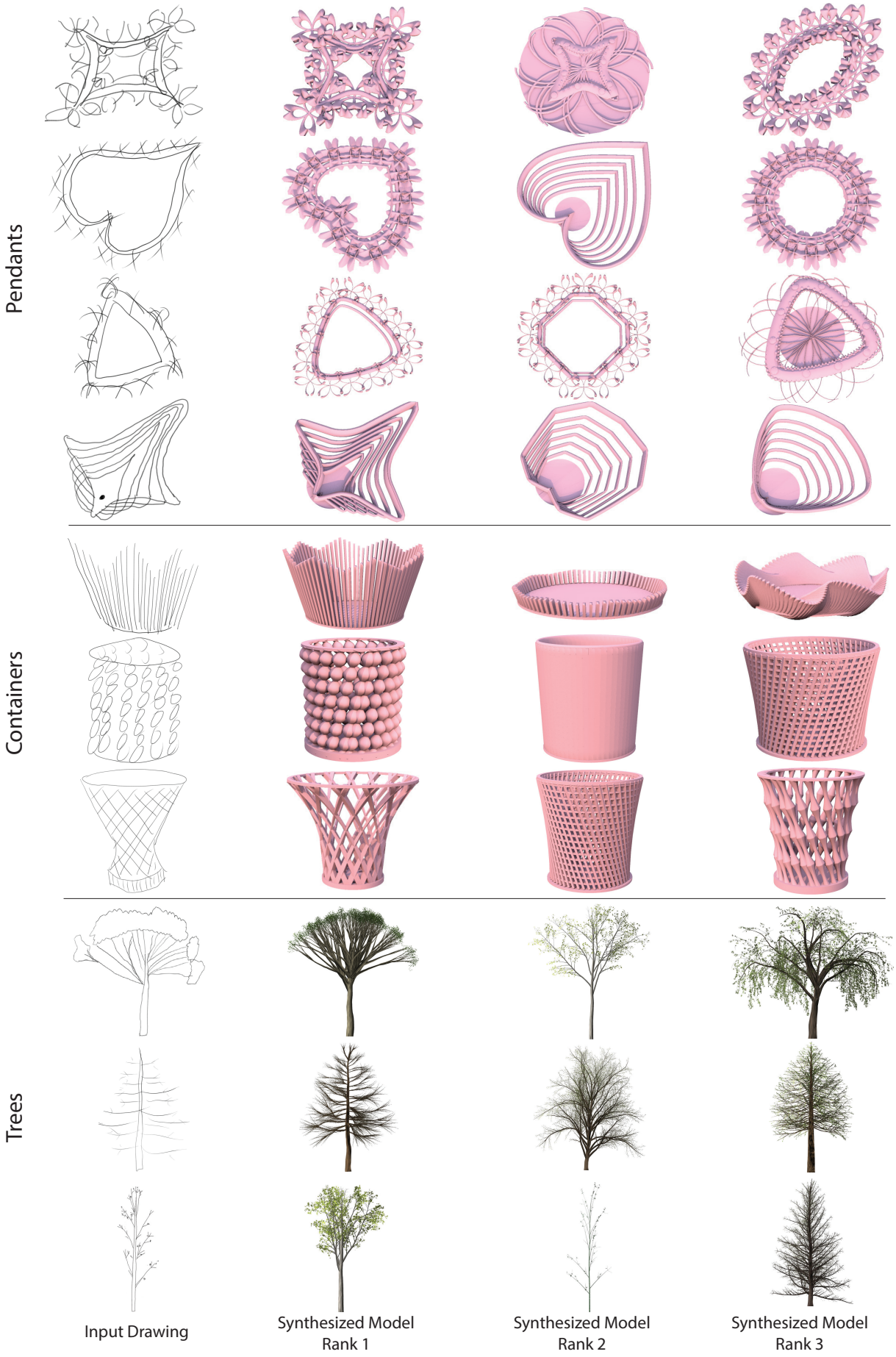

Fig. 6: Input user line drawings along with the top three ranked output shapes generated by our method.

Method	Containers	Trees	Jewelry	Average
Nearest neighbors (Fisher)	20.0%	13.3%	26.7%	20.0%
Nearest neighbors (CNN)	20.0%	20.0%	26.7%	22.2%
SVM (Fisher)	26.7%	26.7%	46.7%	33.3%
SVM (CNN)	26.7%	20.0%	40.0%	28.9%
Single CNN	46.7%	40.0%	60.0%	48.9%
No pretraining	6.7%	6.7%	13.3%	8.9%
Our method	53.3%	53.3%	73.3%	60.0%

TABLE 2: Classification accuracy (top-1) for PM discrete parameters predicted by the examined methods on our user study line drawings.

Method	Containers	Trees	Jewelry	Average
Nearest neighbors (Fisher)	33.3%	26.7%	46.7%	35.6%
Nearest neighbors (CNN)	26.7%	33.3%	40.0%	33.3%
SVM (Fisher)	33.3%	46.7%	60.0%	46.7%
SVM (CNN)	40.0%	33.3%	53.3%	42.2%
Single CNN	66.7%	60.0%	80.0%	68.9%
No pretraining	20.0%	13.3%	20.0%	17.8%
Our method	**80.0%**	**73.3%**	**86.7%**	**80.0%**

TABLE 3: Top-3 classification accuracy for PM discrete parameters predicted by the examined methods on our user study line drawings.

Method	Containers	Trees	Jewelry	Average
Nearest neighbors (Fisher)	32.1%	36.3%	29.1%	32.5%
Nearest neighbors (CNN)	29.3%	34.7%	27.5%	30.3%
RBF (Fisher)	30.5%	35.6%	28.9%	37.1%
RBF (CNN)	31.4%	34.2%	27.6%	31.1%
Single CNN	15.2%	17.3%	11.6%	14.7%
No pretraining	35.2%	40.3%	30.5%	35.3%
Our method	**12.7%**	**15.6%**	**8.7%**	**12.3%**

TABLE 4: PM continuous parameter error (regression error) of the examined methods on our user study line drawings.

by each examined method for our user study drawings. Our method largely outperforms all alternatives. Yet, since the input sketches often represent significant abstractions and simplifications of shapes, it is often the case that an input drawing cannot be unambiguously mapped to a single output shape. If the shape that the user intended to convey with his approximate line drawing is similar to at least one of the highest ranked shapes returned by a method, we can still consider that the method succeeded. Thus, in Table 3 we also report the top-3 classification accuracy i.e., percentage of times that the reference discrete parameter value (class) is contained within the top three ranked discrete values (classes) predicted by each method. Our method again outperforms all alternatives. On average, it can return the desired class of the shape correctly within the top-3 results around 80% of the time versus 69% using a single CNN (a variant of our method), and 47% using SVM and the Fisher vector representation proposed in a recent state-of-the-art work in sketch-based shape retrieval [7]. In Table 4, we report regression error i.e., the relative difference between the predicted and reference values averaged over all continuous parameters for each method. Again, our method outperforms all the alternatives.

5.3 Evaluation on synthetic sketches

We also evaluate our method on how well it predicts PM parameters when the input is a very precise sketch, such as a computer-synthesized line drawing. Evaluating the performance of our method on synthetic sketches is not useful in practice, since we expect human line drawings as input. Yet, it is still interesting to compare the performance of our algorithm on synthetic sketches versus human line drawings to get an idea of how much important is to have a precise input line drawings as input. To evaluate the performance on synthetic sketches, we performed hold-out validation: we trained our CNN on a randomly picked subset of our original training dataset containing 80% of our shapes, and tested the performance on the remaining subset. The classification accuracy (top-1) was 92.5% for containers, 90.8% for trees, 96.7% for pendants, while the regression error was 1.7%, 2.3%, 1.2% respectively. As expected, performance is improved when input drawings are very precise.

5.4 Qualitative evaluation

We demonstrate human line drawings along with the top three ranked outputs generated by the predicted parameters using our method in Figure 6. In Figure 1, we show human line drawings together with the top ranked shape generated by our method. We include additional results in the supplementary material.

6 Limitations and Discussion

We introduced a method that helps users to create and explore visual content with the help of procedural modeling and sketches. We demonstrated an approach based on a deep Convolutional Network that maps sketches to procedural model outputs. We showed that our approach is robust and effective in generating detailed, output shapes through a parametric rule set and human line drawings as input.

Our method has a number of limitations. First, if the input drawings depict shapes that are different from the shapes that can be generated by the rule set, our method will produce unsatisfactory results (Figure 7). The user must have some prior knowledge of what outputs the rule set can produce. When the input drawings become too noisy, contain lots simplifications, exaggerations, or hatching strokes, then again our method will generate outputs that are not likely to be relevant to the users' intentions. We believe that generating synthetic line drawings that reliably simulate such features found in human line drawings would improve the performance of our system. Alternatively, users could be guided to provide more accurate sketches, potentially in 3D, through advanced sketching UIs [46], [47]. Another limitation is that during training we can only support limited number of PM discrete parameters, since we consider each possible combination of discrete values (shape types) to generate our training data (i.e., the number of training shapes grows exponentially with the the number of discrete parameters). In the current implementation of our method, we used a single viewpoint for training the CNN and ask users to draw from the same viewpoint, which can limit their drawing perspective. Predicting PM parameters from multiple different viewpoints e.g., using view-pooling layers [12] would be a useful extension of our system. Finally, our method does not provide real-time feedback to the user while drawing. An interesting future direction would be to

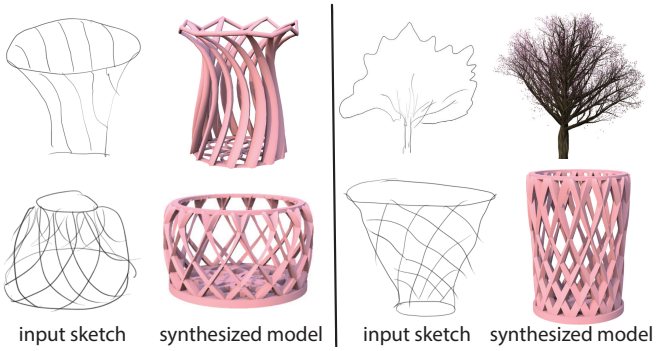

Fig. 7: When the input sketch is extremely abstract, noisy or does not match well any shape that can be generated by the PM, then our method fails to generate a result that resembles the input drawing.

provide such feedback and interactively guide users to draw sketches according to underlying shape variation learned from the procedural models.

APPENDIX

We provide here details about our CNN implementation and the transformations used in its convolutional and pooling layers.

Architecture implementation. Each of our two subnetworks follow the structure of AlexNet [36]. In general, any deep convolutional neural network, reasonably pretrained on image datasets, could be used instead. We summarize the details of AlexNet structure for completeness. The first convolutional layer processes the 227x227 input image with 96 filters of size 11x11. Each filter is applied to each image window with a separation (stride) of 4 pixels. In our case, the input image has a single intensity channel (instead of the three RGB channels used in computer vision pipelines). The second convolutional layer takes as input the max-pooled output of the first convolutional layer and processes it with 256 filters of size 5x5x48. The third convolutional layer processes the max-pooled output of the second convolutional layer with 384 filters of size 3x3x256. The fourth and fifth convolutional layers process the output of the third and fourth convolutional layer respectively with 384 filters of size 3x3x192. There are two fully connected layers contain 4096 processing functions (nodes) each. Finally, the regression layer contains as many regression functions as the number of the PM continuous parameters, and the classification layer contains as many softmax functions as the number of PM discrete parameters. The number of the PM discrete and continuous parameters depend on the rule set (statistics are described in Section 5). The architecture is implemented using the Caffe [48] library.

Convolutional layer formulation. Mathematically, each convolution filter k in a layer l produces a feature map (i.e., a 2D array of values) $h_{k,l}$ based on the following transformation:

$$h_{k,l}[i,j] = f\Big(\sum_{u=1}^{N}\sum_{v=1}^{N}\sum_{m\in\mathcal{M}} w_{k,l}[u,v,m]\cdot h_{m,l-1}[i+u,j+v]+b_{k,l}\Big) \quad (6)$$

where i,j are array (pixel) indices of the output feature map h, $\mathcal{M}$ is a set of feature maps produced in the previous layer (with index $l-1$), m is an index for each such input feature map $h_{m,l-1}$ produced in the previous layer, NxN is the filter size, u and v are array indices for the filter. Each filter is three-dimensional, defined by NxNx$|\mathcal{M}|$ learned weights stored in the 3D array $w_{k,l}$ as well as a bias weight $b_{k,l}$. In the case of the first convolutional layer, the input is the image itself (a single intensity channel), thus its filters are two-dimensional (i.e., $|\mathcal{M}|=1$ for the first convolutional layer). Following [36], the response of each filter is non-linearly transformed and normalized through a function f. Let $x = h_{k,l}[i,j]$ be a filter response at a particular pixel position i,j. The response is first non-linearly transformed through a rectifier activation function that prunes negative responses $f_1(x) = max(0,x)$, and a contrast normalization function that normalizes the rectified response according to the outputs $x_{k'}$ of other filters in the same layer and in the same pixel position: $f_2(x) = [x/(\alpha+\beta\sum_{k'\in\mathcal{K}} x_{k'}^2)]^\gamma$ [36]. The parameters α, β, γ, and the filters $\mathcal{K}$ used in contrast normalization are set according to the cross-validation procedure of [36] ($\alpha=2.0$, $\beta=10^{-4}$, $\gamma=0.75$, $|\mathcal{K}|=5$).

Pooling layer formulation. The transformations used in max-pooling are expressed as follows:

$$h_{k,l}[i,j] = \max\Big\{h_{k,l-1}[u,v]\Big\}_{i<u<i+N, j<v<j+N}$$

where k is an index for both the input and output feature map, l is a layer index, i,j represent output pixel positions, and u,v represent pixel positions in the input feature map.

ACKNOWLEDGMENTS

Kalogerakis gratefully acknowledges support from NSF (CHS-1422441,CHS-1617333) and NVidia for GPU donations. We thank Daichi Ito for providing us with procedural models, and Olga Vesselova for proofreading. We finally thank the anonymous reviewers for their feedback.

REFERENCES

[1] Y. I. Parish and P. Müller, "Procedural modeling of cities," in *Proc. SIGGRAPH*, 2001.
[2] C. White, "King kong: the building of 1933 new york city," in *Proc. SIGGRAPH*, 2006.
[3] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. Van Gool, "Procedural modeling of buildings," *ACM Trans. Graph.*, vol. 25, no. 3, 2006.
[4] R. M. Smelik, T. Tutenel, R. Bidarra, and B. Benes, "A survey on procedural modelling for virtual worlds," *Comp. Graph. Forum*, vol. 33, no. 6, 2014.
[5] M. Eitz, J. Hays, and M. Alexa, "How do humans sketch objects?" *ACM Trans. Graph.*, vol. 31, no. 4, 2012.
[6] M. Eitz, R. Richter, T. Boubekeur, K. Hildebrand, and M. Alexa, "Sketch-based shape retrieval," *ACM Trans. Graph.*, vol. 31, no. 4, 2012.
[7] R. G. Schneider and T. Tuytelaars, "Sketch classification and classification-driven analysis using fisher vectors," *ACM Trans. Graph.*, vol. 33, no. 6, 2014.
[8] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "Decaf: A deep convolutional activation feature for generic visual recognition," arXiv:1310.1531, http://arxiv.org/abs/1310.1531, 2013.
[9] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. CVPR*, 2014.
[10] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "Cnn features off-the-shelf: an astounding baseline for recognition," in *Proc. CVPR Workshops*, 2014.

[11] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi, "Describing textures in the wild," in *Proc. CVPR*, 2014.
[12] H. Su, S. Maji, E. Kalogerakis, and E. G. Learned-Miller, "Multi-view convolutional neural networks for 3d shape recognition," in *Proc. Int. Conf. Comput. Vision*, 2015, pp. 945–953.
[13] A. Lindenmayer, "Mathematical models for cellular interactions in development i. filaments with one-sided inputs," *J. Theoretical biology*, 1968.
[14] P. Prusinkiewicz, "Graphical applications of l-systems," in *Proc. Graphics interface*, 1986.
[15] P. Prusinkiewicz, M. James, and R. Měch, "Synthetic topiary," in *Proc. SIGGRAPH*, 1994.
[16] R. Měch and P. Prusinkiewicz, "Visual models of plants interacting with their environment," in *Proc. SIGGRAPH*, 1996.
[17] G. Stiny and J. Gips, "Shape grammars and the generative specification of painting and sculpture." in *IFIP Congress*, 1971.
[18] M. Lipp, P. Wonka, and M. Wimmer, "Interactive visual editing of grammars for procedural architecture," *ACM Trans. Graph..*, vol. 27, no. 3, 2008.
[19] T. Germer and M. Schwarz, "Procedural arrangement of furniture for real-time walkthroughs," *Comp. Graph. Forum*, vol. 28, no. 8, 2009.
[20] P. Merrell, E. Schkufza, and V. Koltun, "Computer-generated residential building layouts," in *ACM Trans. Graph.*, vol. 29, no. 6, 2010.
[21] M. Schwarz and P. Wonka, "Procedural design of exterior lighting for buildings with complex constraints," *ACM Trans. Graph.*, vol. 33, no. 5, 2014.
[22] B. Lintermann and O. Deussen, "Interactive modeling of plants," *IEEE Computer Graphics and Applications*, 1999.
[23] J. McCrae and K. Singh, "Sketch-based path design," in *Proc. Graphics Interface*, 2009.
[24] C. A. Vanegas, I. Garcia-Dorado, D. G. Aliaga, B. Benes, and P. Waddell, "Inverse design of urban procedural models," *ACM Trans. Graph.*, vol. 31, no. 6, 2012.
[25] O. Stava, S. Pirk, J. Kratt, B. Chen, R. Měch, O. Deussen, and B. Benes, "Inverse procedural modelling of trees," *Comp. Graph. Forum*, vol. 33, no. 6, 2014.
[26] J. O. Talton, Y. Lou, S. Lesser, J. Duke, R. Měch, and V. Koltun, "Metropolis procedural modeling," *ACM Trans. Graph.*, vol. 30, no. 2, 2011.
[27] D. Ritchie, B. Mildenhall, N. D. Goodman, and P. Hanrahan, "Controlling procedural modeling programs with stochastically-ordered sequential monte carlo," *ACM Trans. Graph.*, vol. 34, no. 4, 2015.
[28] G. Nishida, I. Garcia-Dorado, D. G. Aliaga, B. Benes, and A. Bousseau, "Interactive sketching of urban procedural models," *ACM Trans. Graph., to appear*, 2016.
[29] J. O. Talton, D. Gibson, L. Yang, P. Hanrahan, and V. Koltun, "Exploratory modeling with collaborative design spaces," *ACM Trans. Graph.*, vol. 28, no. 5, 2009.
[30] S. Lienhard, M. Specht, B. Neubert, M. Pauly, and P. Mller, "Thumbnail galleries for procedural models," *Comp. Graph. Forum*, vol. 33, no. 2, 2014.
[31] M. E. Yumer, P. Asente, R. Mech, and L. B. Kara, "Procedural modeling using autoencoder networks," in *Proc. ACM UIST*, 2015.
[32] T. Funkhouser, P. Min, M. Kazhdan, J. Chen, A. Halderman, D. Dobkin, and D. Jacobs, "A search engine for 3d models," *ACM Trans. Graph.*, vol. 22, no. 1, 2003.
[33] S. Hou and K. Ramani, "Sketch-based 3d engineering part class browsing and retrieval," in *Proc. SBIM*, 2006.
[34] J. Pu, K. Lou, and K. Ramani, "A 2d sketch-based user interface for 3d cad model retrieval," *Computer-Aided Design and Applications*, vol. 2, no. 6, 2005.
[35] R. G. Schneider and T. Tuytelaars, "Sketch classification and classification-driven analysis using fisher vectors," *ACM Trans. Graph.*, vol. 33, no. 6, 2014.
[36] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. NIPS*, 2012.
[37] F. Wang, L. Kang, and Y. Li, "Sketch-based 3d shape retrieval using convolutional neural networks," in *Proc. Comput. Vision Pattern Recognition*, 2015, pp. 1875–1883.
[38] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision*, 2015.
[39] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *Proc. NIPS*, 2014.
[40] C. M. Bishop, *Pattern recognition and machine learning*. Springer, 2006.
[41] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Machine Learning Research*, vol. 15, no. 1, 2014.
[42] D. DeCarlo, A. Finkelstein, S. Rusinkiewicz, and A. Santella, "Suggestive contours for conveying shape," *ACM Trans. Graph.*, vol. 22, no. 3, 2003.
[43] O. K.-C. Au, C.-L. Tai, H.-K. Chu, D. Cohen-Or, and T.-Y. Lee, "Skeleton extraction by mesh contraction," *ACM Trans. Graph.*, vol. 27, no. 3, 2008.
[44] R. Mech and G. Miller, "The deco framework for interactive procedural modeling," *J. Computer Graphics Techniques*, 2012.
[45] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Trans. Intell. Syst. Technol.*, vol. 2, 2011.
[46] S.-H. Bae, R. Balakrishnan, and K. Singh, "Ilovesketch: As-natural-as-possible sketching system for creating 3d curve models," in *Proc. ACM UIST*, 2008.
[47] Y. Zheng, H. Liu, J. Dorsey, and N. J. Mitra, "Smartcanvas: Context-inferred interpretation of sketches for preparatory design studies," *Computer Graphics Forum*, vol. 35, no. 2, 2016.
[48] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," arXiv:1408.5093, https://arxiv.org/abs/1408.5093, 2014.

Haibin Huang received his BSc and MSc degrees in Mathematics in 2009 and 2011 respectively from Zhejiang University. He is currently a PhD student at the College of Information and Computer Sciences, University of Massachusetts Amherst. His research mainly focuses on three-dimensional shape analysis, modeling and synthesis, and in particular machine learning techniques for geometry processing.

Evangelos Kalogerakis is an assistant professor in computer science at the University of Massachusetts Amherst. His research deals with the development of computational techniques that intelligently analyze and synthesize visual content. He is particularly interested in developing machine learning algorithms that help people to easily create and process 3D models. He obtained his PhD from the University of Toronto in 2010. He was a postdoctoral researcher at Stanford University from 2010 to 2012.

Ersin Yumer is a research scientist at Adobe Research. He graduated from CMU in 2015. His PhD dealt with computer graphics applications of machine learning and data driven approaches. He focused on frameworks where the end user is abstracted from directly interacting with the content, but is enabled with 'natural language like' controllers to manipulate 3D geometry and its appearance. His research has recently focused on bringing 2D and 3D worlds together by bridging the gap between different media modalities.

Radomir Mech leads a Procedural Imaging Group at Adobe Research. His group currently consists of 5 researchers, 1 tech transfer engineer and 1 technical artist. The researchers span areas of 2D and 3D design, image processing, modeling, natural media simulation, and HCI. His areas of research are procedural modeling, with a particular focus on interaction with procedural models and casual modeling, rendering, 3D printing, and imaging algorithms. He obtained his PhD from the University of Calgary.