Procedural Modeling Using Autoencoder Networks

Mehmet Ersin Yumer
Adobe Research
yumer@adobe.com

Paul Asente
Adobe Research
asente@adobe.com

Radomir Mech
Adobe Research
rmech@adobe.com

Levent Burak Kara
Carnegie Mellon University
lkara@cmu.edu

ABSTRACT

Procedural modeling systems allow users to create high quality content through parametric, conditional or stochastic rule sets. While such approaches create an abstraction layer by freeing the user from direct geometry editing, the nonlinear nature and the high number of parameters associated with such design spaces result in arduous modeling experiences for non-expert users. We propose a method to enable intuitive exploration of such high dimensional procedural modeling spaces within a lower dimensional space learned through autoencoder network training. Our method automatically generates a representative training dataset from the procedural modeling rule set based on shape similarity features. We then leverage the samples in this dataset to train an autoencoder neural network, while also structuring the learned lower dimensional space for continuous exploration with respect to shape features. We demonstrate the efficacy our method with user studies where designers create content with more than 10-fold faster speeds using our system compared to the classic procedural modeling interface.

ACM Classification Keywords

I.3.7 Computer Graphics: Three-Dimensional Graphics and RealismApplications

Author Keywords

procedural modeling; parametric shape design; autoencoders; neural networks

INTRODUCTION

Procedural modeling (PM) allows complex phenomena and geometry to be created using a set of parametric, conditional or stochastic rules [31, 50, 29]. As such, procedural approaches in geometry modeling create an abstraction layer between the user and the geometry that alleviates the need for tedious direct geometric editing. A wide variety of object categories can be modeled using PM including organic shapes such as trees and animated characters as well as man-made shapes such as buildings, furniture, and jewelry [40]. Once a procedural model for an object category is available, it can be used to create a rich set of unique instances by varying the parameters of the procedural rule set. However, often times the underlying parametric space is very high-dimensional and its mapping to geometry is complex, thus making the resulting geometry difficult to control and explore using direct parameter editing.

We propose a method to enable an intuitive exploration of such high-dimensional PM spaces using only a small number of parameters generated by an autoencoder learning algorithm. Based on the underlying PM rules, we first use a categorization tree [14] to generate a set of representative shapes to be used in training. This allows us to generate a training dataset uniform in shape variety rather than in the rule set parameters. We then use the generated samples to train an autoencoder neural network. Our insight is that the conventional autoencoder bottleneck [13] suitably reduces the large number of original parameters into a much smaller set, which we have found to greatly enhance shape control in PM.

A dimensionality reduction exclusively in the parametric space, however, does not ensure a meaningful organization with respect to shape similarity. In particular, a uniform sampling of the reduced parametric space rarely produces geometrically continuous variations; a problem similarly inherent to sampling in the original parameter space [40]. To address this issue, we augment our autoencoders with additional nodes capable of encoding and reproducing geometric shape features. This augmentation allows our autoencoders to generate a continuous geometric space with respect to the shape features, resulting in an intuitive and predictable shape navigation experience driven primarily by shape similarity (Figure 1).

Our main contributions are:

- A method to create a lower-dimensional and generative representation of high-dimensional procedural models using autoencoders, continuous with respect to shape features.
- A shape design system for generating novel procedural models using an explore-and-select interaction, without the need for direct parameter editing.

Our evaluations show that using our approach, end users can complete targeted as well as open-ended modeling tasks on average six to twelve times faster compared to a conventional PM system.

UIST 2015, November 8–11, 2015, Charlotte, NC, USA.

http://dx.doi.org/10.1145/2807442.2807448

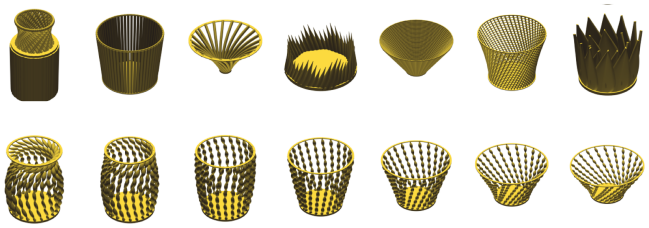

Figure 1. Uniform sampling in a random direction from a procedural rule set. Top: Original parameter space, Bottom: Our low-dimensional space.

RELATED WORK

Procedural modeling. L-systems introduced by Lindenmayer *et al.* [21] mark the first use of procedural models and targets biological modeling. Current geometric applications of PM [34] have been enabled by extended L-systems involving symbol strings that are interpreted as a sequence of commands for higher topological structures [32]. Shape grammars [43] are also used in PM with the recent introduction of split grammars for urban modeling [52]. Applications of PM in digital design span a wide variety of categories including plants [33, 27], cities [31], buildings [29], furniture arrangements [11], building layouts [28] and lighting design [36]. For an extensive survey of procedural modeling in computer graphics refer to the recent survey by Smelik *et al.* [40].

Conventional PM interfaces require end users to adjust a set of parameters that typically have complex, aggregate effects on the resulting geometry. Solutions to this problem have primarily focused on targeted design and exploratory systems.

Targeted design with procedural models. There exist a body of PM work that circumvent direct interaction with PM parameters by enabling a targeted design platform. Lintermann *et al.* [22] present an interactive PM system where conventional PM modeling is combined with free-form geometric modeling for plants. McCrae and Singh [25] introduce an approach for converting arbitrary sketch strokes to 3D roads that are automatically fit to a terrain. Talton *et al.* [45] present a method for achieving high-level design goals (*e.g.,* city skyline profile for city procedural modeling) with PM by inverse optimization of PM parameters that conform to the design constraints. Inverse PM systems have also been shown to be effective for urban facade design [47] and vegetation design [42].

Although the methods for targeted design free the user from directly adjusting PM parameters, the primary goal is to realize high-level design goals. This choice results in a less user control over the generated models. For instance, while the user can prescribe the skyline silhouette in [45], their control over the building geometries is at a minimum.

Exploration of procedural models. High number of parametric spaces that are difficult and often non-intuitive to explore are a common problem in computer graphics. Marks *et al.* [24] introduced one of the earlier works in abstracting the user from directly manipulating high-dimensional parametric spaces. Their interface provides the user automatically organized, perceptually different options that can be created with a given input-parameter vector. Koyama *et al.* [18] introduced a method where they learn *better* (*i.e.,* human-preferred) parameters of a procedural model using pairwise parameter set comparisons resulting from a crowd-sourced user study. They represent this preference as a *goodness* measure, which is then used to interactively guide users towards better parameter sets during design exploration.

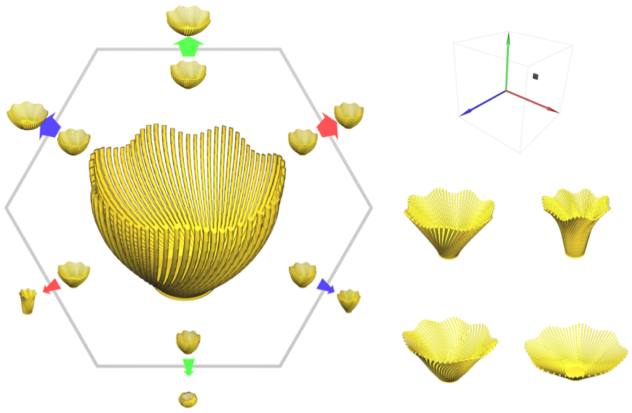

Figure 2. Our procedural modeling exploration interface. Left: Navigation. Bottom-right: saved configurations and shape interpolation. Top-right: design space visualization.

Exploratory systems such as [44, 20] provide the user with previously computed and sorted exemplars that helps the user study the variety of models and select the seed models they wish to further explore. Talton *et al.* [44] organize a set of precomputed samples on a 2D map, where the model distribution in the map is established by a set of landmark examples created by the expert users of the system. Lienhard *et al.* [20] sort precomputed sample models based on a set of automatically computed views and geometric similarity. They present the results as rotational and linear thumbnail galleries. In both systems, if the user wants to create a new model based on a precomputed sample, they have to revert to the conventional PM approach and tweak the parameters of the original procedural model.

Generative shape and image design. Previous works that studied generative shape [15, 16, 53, 46, 6, 54, 2, 1] and image design [9, 38] systems have introduced modeling interfaces similar to ours that help the user explore and synthesize models more intuitively. These methods target non-parametric models (*e.g.,* polygonal shapes) and cannot be readily extended to parametric models. Our method introduces a generative dimensionality reduction method that automatically organizes the lower dimensional space tailoring the learner towards intuitive user-interaction. This enables us to learn complex parametric design spaces, independent of the underlying geometry, while serving specific user centric tasks (such as intuitive shape modeling as demonstrated in this paper), whereas the methods mentioned above are restricted to learn from and process the shape geometry directly.

Deep neural networks and autoencoders. Deep neural networks model high-level abstractions in data using model architectures composed of multiple non-linear transformations

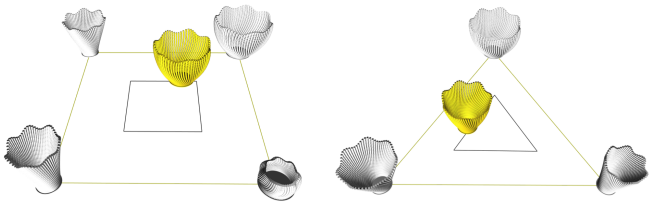

Figure 3. Users can generate new models by interpolating between an arbitrary number of existing (left: four, right three) models.

[10, 19]. Recent advances in training methods involving layer by layer unsupervised training followed by supervised fine tuning have made these approaches highly successful on large datasets [12, 4]. These advances have led to applications ranging from segmentation and object detection in images and video [30, 37], to speech recognition [8].

An autoencoder is a special form of a neural network where the goal is to reproduce the input layer at the output layer. While autoencoders have been used primarily for unsupervised deep feature learning [35], they have also been shown to be superior to other dimensionality reduction methods [13]. Inspired by this outcome, we employ autoencoder networks to reduce the dimensionality of the original parameter space. Our formulation additionally organizes the distribution of the objects in the reduced-dimensional space based on geometric proximity using shape features.

END USER EXPERIENCE

Our user interface shown in Figure 2 consists of a shape navigation, a shape interpolation, and a parametric positioning pane.

Navigation. The model corresponding to the current parameter configuration is shown in the center. The user can explore the proximate models by pressing the arrows, each increasing or decreasing one of the reduced dimensional procedural parameters. In our approach, we use a 3-dimensional reduced parameter space (coded in red, green, and blue in Figure 2) which we have empirically found to balance geometric richness with ease of navigation. Two small 3D thumbnails are also shown associated with each arrow that reveal the immediate next and the last models, respectively, along that parametric direction.

Shape interpolation. Users can save as many models as they wish and generate new models by interpolating between the set of saved models (Figure 3). As the user drags the current model in the interpolation widget, a new model is computed as a barycentric combination of the saved models. The interpolation is linear in the learned low-dimensional parametric space and produces results that are geometrically continuous.

Parametric positioning. The reduced dimensional parametric coordinates of the current model are presented within a 3D frame to allow users to track the current model's global configuration.

LEARNING THE PROCEDURAL MODEL SPACE

Our approach to learning the procedural model space involves first generating a robust model sample set followed by training a generative autoencoder network for dimensionality reduction.

Sampling the Training Data

Because learning a reduced dimensional representation of the original PM space ultimately depends on the training data, the quality of this data becomes an important consideration. Specifically, it is desirable that the training set provides a uniform coverage of the input space with respect to the resulting *shapes* rather than the underlying *parameters*. This is a non-trivial problem as there exists no immediate mechanism to ensure a uniform sampling of the shapes. To address this issue, we use categorization trees [14] to sample shapes with respect to a set of shape features followed by a uniform subsampling from the resulting trees. We iterate between these two steps until the final samples uniformly sample the shape space.

Categorization tree. To generate the initial hierarchical categorization of samples, we use the categorization tree (C-tree) approach introduced by Huang *et al.* [14]. A C-tree, inspired by the evolutionary trees in bioinformatics [51], is a hierarchical tree structure (Figure 4) where each leaf corresponds to a unique shape. The tree is constructed by assembling and reorganizing *quartets* of shapes subject to topological constraints. A quartet is a tree of four leaves which forms the smallest bundle that can express meaningful topological relations. C-tree construction begins by computing a set of quartets from the input shapes which form the edges of a graph. We apply the Quartets MaxCut algorithm [41] to this graph to compute the resulting C-tree. However we use a set of shape features that are different than those used in [14].

Subsampling. The next step is to sample the C-tree to obtain samples uniformly distributed with respect to shape. We originally utilized uniform subtree sampling methods introduced for evolutionary trees. However, the polynomial time complexity[17, 5] renders them prohibitive when the desired sample size is in the thousands. We therefore adopt an approximate but fast method that uniformly samples from a probability distribution based on individual selection probabilities computed for each leaf node as follows:

$$P_i = \frac{(d_i \times s_i)^{-1}}{\sum_i (d_i \times s_i)^{-1}} \quad (1)$$

where d_i is the depth of the node starting from the root node (center node in Figure 4) and s_i is the number of siblings (*i.e.,* shapes in the two-ring neighborhood). The probability of selection decreases if: (1) the number of siblings increases, or (2) the node is deeper in the tree. The first condition helps distribute the probability of selection equally among the nodes at the same depth between the immediate siblings. The second condition favors the nodes at shallower depths thereby concentrating on parts of the shape space that have not been deeply explored.

Iterative C-tree subsampling. To construct a training set that reliably captures the shape variation we iteratively: (1)

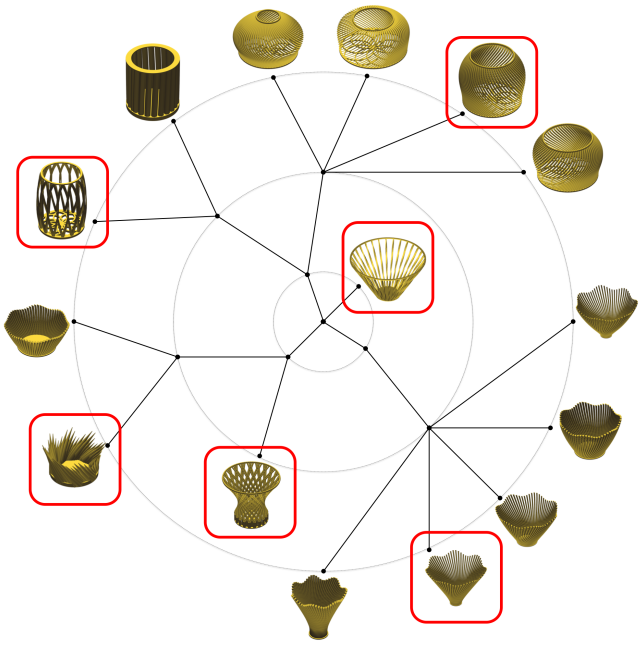

Figure 4. (a) Categorization tree computed for a small set of models sampled from the *Containers* set. Subsampling performed on the categorization tree are shown in red.

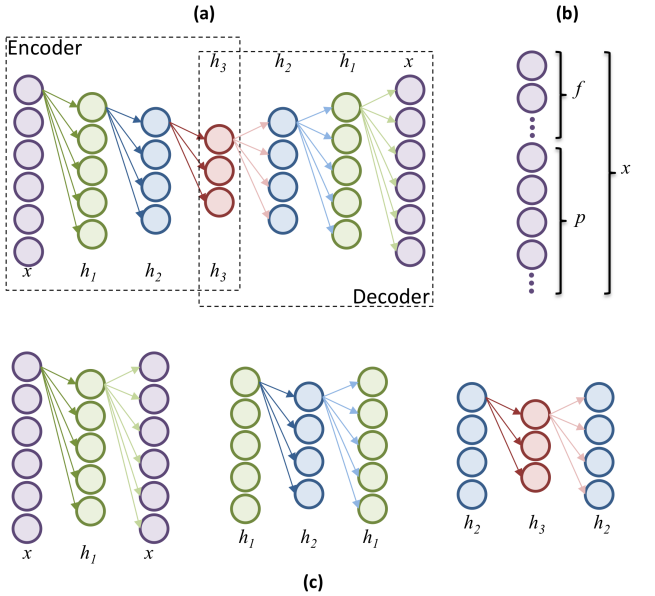

Figure 5. (a) Illustration of our symmetric autoencoder network. (b) Our input/output vectors consist of parametric - p as well as shape descriptor - f information. (c) Training is performed layer by layer, followed by a fine tuning of the complete structure. While fully connected, network edge weights are illustrated only for the first neuron of each layer.

add new samples to the models forming the C-tree, (2) compute a new C-tree, (3) perform subsampling. We continue this process until the following criterion is satisfied:

$$\mathbb{E}_t - \mathbb{E}_{t-1} < w(\mathbb{E}_{t-1} - \mathbb{E}_{t-2}) \tag{2}$$

where we define $\mathbb{E}_t$ to be the potential energy in the C-tree at iteration t, and $0 < w < 1$ is a weighting constant. We define C-tree potential as

$$\mathbb{E} = \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} e_i \tag{3}$$

where $\mathcal{S}$ is the set of shapes in the tree and e_i is the potential of shape i defined as:

$$e_i = \min |x_i - x_j| \qquad i, j \in \mathcal{S}, i \neq j \tag{4}$$

where x_i and x_j are the shape descriptors of shapes i and j respectively. This formulation penalizes large geometric differences between a shape and the one most similar to it thereby ensuring a homogeneous sampling of the design space.

Organized Low-Dimensional Shape Space

Autoencoder architecture. We use a five hidden layer symmetric autoencoder network illustrated in Figure 5. We use three bottleneck neurons to facilitate design space visualization. We determine the number of neurons for the hidden layers h_1 and h_2 based on the number of input layer neurons.

Conventional autoencoders work to replicate input vectors at the output layer. In the context of PM this means if the input layer consists of the original PM parameters, the autoencoder will aim to reproduce these parameters at the output layer, albeit with a reduced set of bottleneck neurons. However, using this approach shape navigation in the reduced dimensional space would remain cumbersome as no provision would exist to ensure geometric continuity.

To mitigate this issue we augment our autoencoder with additional nodes (vector f in Figure 5) that incorporate shape features, thereby simultaneously helping to organize the learned reduced dimensional space. This structuring takes place with respect to shape similarity and ensures geometrically similar shapes to appear close together in the low-dimensional space.

Shape features. Our approach is applicable to both 3D and 2D geometries. For 3D shapes we utilize the light field descriptor [7] with shape context [3]. In this formulation, a 3D model is placed in a virtual sphere and multiple 2D renders of the shape is computed using a set of camera placements on this sphere. From each render, a silhouette-based histogram is extracted and the results are concatenated to form the shape's feature vector. This formulation is size invariant but is sensitive to rotations. However, because PM enables the generation of consistently oriented shapes, rotational sensitivity does not impact our autoencoders. When the models are 2D shapes (images) we similarly use shape contexts to encode shape [3] and additionally augment the shape's feature vector with texton histograms [39] to encode appearance.

Shape feature weighted reconstruction error. We utilize the popular layer by layer training scheme [12] (Figure 5(c)) followed by a fine-tuning pass over the network. We back-propagate an error function designed to equalize the importance of the procedural model parameters and the shape descriptors. We apply a nonuniform weighting scheme to the conventional back-propagation function (mean-squared reconstruction error) of an output neuron, n, over all examples as follows:

$$E_n = \frac{1}{I} w_n \sum_{i=1}^{I} (t_i - r_i)^2 \quad \text{where } w_n = \begin{cases} 1 & \text{if } n \in p \\ c\frac{|p|}{|f|} & \text{if } n \in f \end{cases} \quad (5)$$

where I is the number of samples and t_i and r_i are the target and the reconstructed values at the output neuron, respectively. p is the set of output neurons that correspond to procedural model parameters and f is the set of output neurons encoding the shape features. The total network error is then:

$$E_{total} = \sum_{n=1}^{|p|+|f|} E_n \quad (6)$$

Parameter c enables control over the influence of parameters versus shape features. As c increases, the formulation becomes more sensitive to feature nodes. For a given PM rule set, we choose c that results in the smallest network error E_n using a greedy search after a training of $1K$ epochs[1]. Note that, c can be manually designed to force the network put even more emphasis on the features but in our experiments this resulted in relatively more error in PM parameter reconstruction. We therefore opted to choose c based on total network error minimization. This weighting scheme is required only for the first hidden layer. For the remaining layers, the training is carried out in the conventional way where the first hidden layer's neurons readily transmit the necessary weights to the subsequent layers.

Denoising. For autoencoders, the *denoising* training scheme introduced by Vincent *et al.* [48] has proven to be superior to conventional training methods. The idea behind denoising is to make learning robust to partial corruption in the input. For each training sample, denoising works by providing as input a set noise-added variations of the sample and engineer the network to reconstruct the original, noiseless sample. This works particularly well for feature learning in vision applications as the input image is often inherently noisy. Based on this observation, we experimented with three alternatives: (1) denoising applied to both to the PM parameters and shape features, (2) denoising applied only to the PM parameters, (3) denoising applied only to the shape features.

We have found the third option to work best for our application. We evaluated all three options using separate training and validation sets. Table 1 shows the average reconstruction errors obtained for a 10-fold cross-validation study and normalized against the reconstruction error in *parameter denoising* setting. As shown, denoising applied only to the shape features produces the best results. This outcome is due to the fact that PM parameters p are generative and thus fully control the resulting shape (*i.e.,* no hidden processes that introduce noise) but the feature vector f is computed from the resulting shape and is not generative (hence, sensitive to image-level processing details and thus noise).

[1]In this paper, $c = 3.1$ for the *Containers* and $c = 2.7$ for the *Trees*.

Table 1. 10-fold cross-validation reconstruction error normalized by the largest error in each category.

	Containers	*Trees*
Parameter + Shape feature denoising	0.94	0.91
Parameter denoising	1.00	1.00
Shape feature denoising	0.83	0.81

NAVIGATING THE PROCEDURAL MODEL SPACE

Once an autoencoder is trained, we use the decoding part of the network to generate new models. Specifically, we use the nodes at the bottleneck as input, generate the procedural modeling parameters at the output layer, and compute the corresponding model using the procedural modeling rule set (Figure 5(a)).

While the number of bottleneck nodes can be arbitrary, these nodes dictate the dimensionality of the new parametric space. For ease of end-user visualization and navigation in the user interface (Figure 2), we have considered two- and three-bottleneck neuron configurations. Based on our evaluations with the autoencoder performance and the replication capability (Figure 12), we choose three bottleneck neurons for our demonstrations and user evaluations.

It should be noted that we do not claim three degrees of freedom (DOF) are enough for reducing the dimensionality of any procedural model. Our primary goal is to make the interaction and understating of the interface and the modeling space easier and intuitive for novice users. This is different than a traditional dimensionality reduction problem where the number of DOF of the reduced space can be optimized to gracefully represent the original space. We fix the number of DOF of the reduced space similar to other procedural model exploration systems (*e.g.,* [44]) to be able to create a convenient and intuitive user experience. Although we are not optimizing for a maximum representation of the original space in the reduced one, note that our replication (Figure 12, Figure 8) user study shows the reduced dimensional space enabled the users to replicate their models designed in the original space gracefully. For more complex shapes where higher dimensions for the learned space is necessary, the interface can provide slices of the higher dimensional space in 3D with our widget and an additional widget might be programmed to jump between these slices.

In shape interpolation mode, the saved models are presented at the corners of a multi-sided regular polygon. New models are computed as the barycentric combinations within the convex space spanned by the saved models [23]. The resulting barycentric combination constitutes the low-dimensional

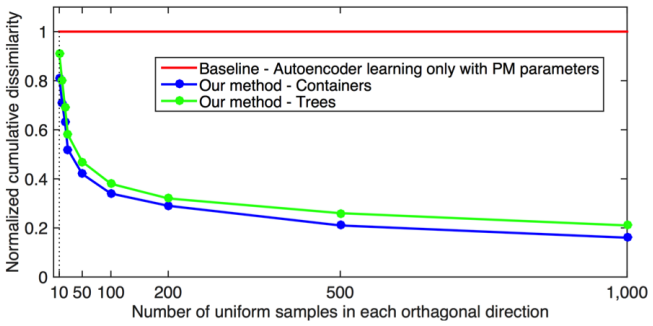

Figure 6. Normalized cumulative dissimilarity (D_N) as a function of sampling density. $D_N = \frac{D}{D_b}$, where D_b is the baseline cumulative dissimilarity.

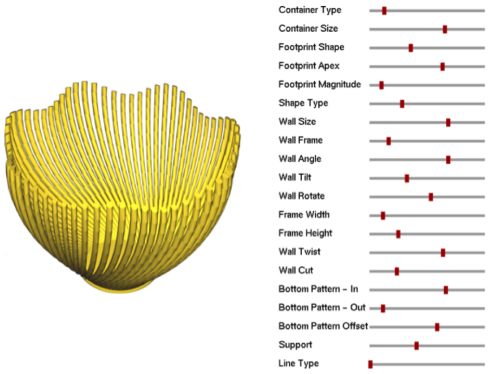

Figure 7. Conventional procedural modeling interface.

parametric coordinates of the new shape which is then geometrically realized through the autoencoder as explained above. Figure 11 illustrates several interpolations. Please see our supplementary video for further examples and user interaction.

EVALUATION AND RESULTS

Procedural models used in the experiments. We demonstrate our method using two PM rule sets: (1) containers, (2) trees. Both rule sets are built using the Deco framework [26]. *Containers* set consists of 72 independent parameters whereas *Trees* set consist of 100 parameters. The *Containers* rule set generates 3D shapes using vertical rods, generalized cylinders and trapezoids on the side walls, and a fractal geometry at the base. Our supplementary video shows the interaction with this model's parameters. The *Trees* set is similar to [49] in spirit and is currently available in Adobe Photoshop©. It generates 2D tree renders which are processed similar to the 3D shapes.

For the container rule set, Figure 9 compares our method (dimensionality reduction + space organization) against a baseline method involving dimensionality reduction only on the PM parameters. Our approach produces a shape space that organizes the reduced dimensional space more favorably with respect to shape similarity. For the *Trees* set, Figure 10 similarly compares our method against the baseline method. While the autoencoder for the trees included three bottleneck neurons, Figure 10 shows shapes uniformly sampled along the vector $\sqrt{.25}\mathbf{i}$, $\sqrt{.40}\mathbf{j}$, $\sqrt{.35}\mathbf{k}$, in the low-dimensional space with the left- and right-most shapes corresponding to the two extremes along this vector. In both examples, we quantify the difference between our method and the baseline method using a cumulative shape *dissimilarity* measure:

Table 2. Times and satisfaction outcomes for a design / replication user study. (A): Users who designed using the conventional system, followed by replication in our system. (B): Users who designed using our system, followed by replication in the conventional system.

	Group A	Group B
Time to Design (min.)	$\mu = 25, \sigma = 5.8$	$\mu = 3.1, \sigma = 0.9$
Time to Replicate (min.)	$\mu = 4.1, \sigma = 1.2$	$\mu = 37, \sigma = 9.6$
Satisfied with replica (%)	91.1	42.2

$$D = \sum_{i \in \mathcal{S}} \sum_{j \in N_i} \left| f_i - f_j \right| \tag{7}$$

where $\mathcal{S}$ are the samples generated in the low-dimensional space. N_i is the set of neighbors of i along the axes of the low-dimensional space[2]. f_i and f_j are the feature vectors of sample i and j, respectively. The resulting dissimilarity measure depends on the sampling density of the low-dimensional space. Figure 6 reports the cumulative dissimilarity measure as a function of sampling density using our method, versus the baseline method of dimensionality reduction only on the PM parameters.

User Study: Comparative Evaluation

To compare our system with the conventional PM approach, we conducted a user study with the *Containers* dataset involving 90 users. All users of the system were non-experts: they did not have educational or professional experience with shape design, 3D modeling, or procedural modeling. First, to familiarize the users with the variety of models in this space, we provided them with a catalog of 1000 models randomly sampled from the original parameter space. We then asked each user to design a container of their liking using our system and the conventional procedural modeling system

[2]The number of neighbors is six for the *Containers* and two for the *Trees* except for the boundary models.

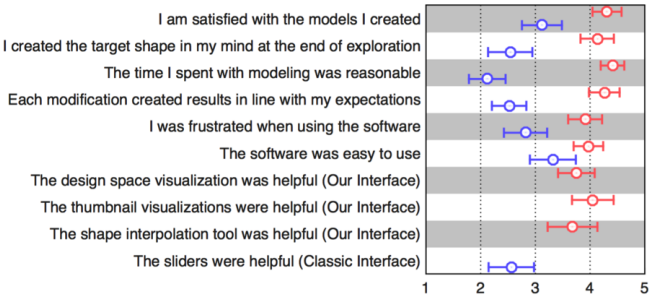

Figure 8. Survey response collected from all 90 participants of the user study (1: strongly disagree, 3: neutral, 5: strongly agree). Red: responses about our system. Blue: responses about the conventional procedural modeling system. Error bars indicate one standard deviation.

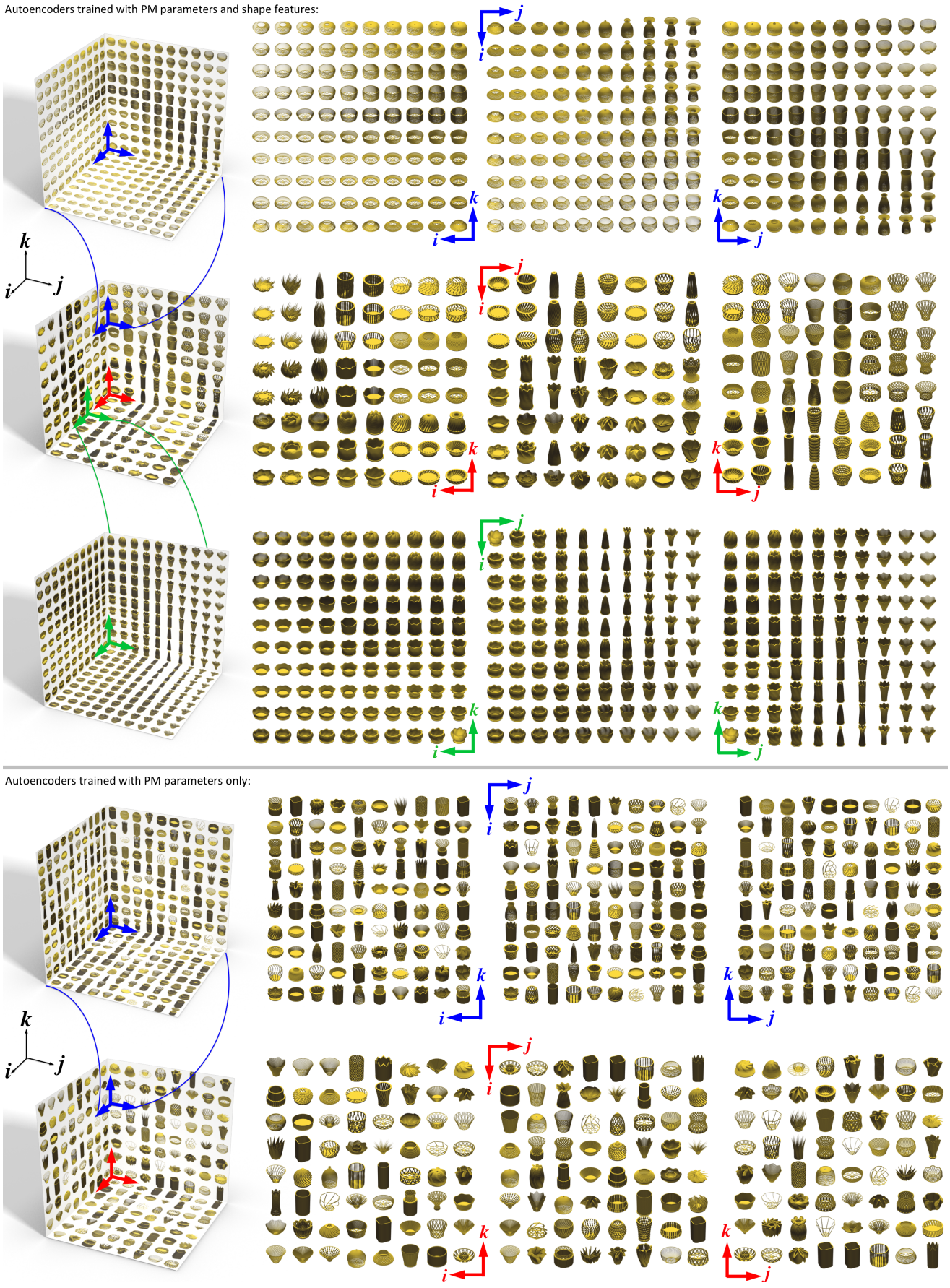

Figure 9. **Top: Shape sampling and the resulting container space using our approach. Both PM parameters and shape features are encoded in the autoencoder. Bottom: Same approach, without using shape features. Navigation is much more challenging without the shape features.**

Figure 10. Uniform sampling of trees. Top: Uniform sampling in the low-dimensional space learned with autoencoders using only the PM parameters. Bottom: Uniform sampling in our organized low-dimensional space.

(Figure 7). Half of the users started with our interface, and the other half started with the conventional interface. Both groups were given the following instructions:

You will be using two applications for creating a container model similar to the ones you saw in the catalog. Please design a container to your liking with the first application. Once finished, please try to replicate your first model using the second application.

To preserve neutrality, we referred to the two applications as #1 and #2 based on the order they were presented to the users. Table 2 summarizes the results. Users who started with the conventional modeling system (Group A) took on average 25 minutes to arrive at a model of their liking. They were then able to replicate it to their satisfaction using our system in 4 minutes on average. 91% of the users in this group reported that they were satisfied with their replication. On the other hand, users who started with our system (Group B) took 3 minutes on average to arrive at a model of their liking while taking 37 minutes to replicate it with the conventional modeling system. Only 42% of the users in this group reported that they were satisfied with their replication results. This mismatch is also reflected in the visual differences between the designed and the replica models (Figure 12). We believe these results highlight the efficacy of our approach.

Following the modeling session, we asked each user to complete a survey about their experience with the two systems on a Likert scale (Figure 8). The results indicate that the users seem to have had an overall better modeling experience using our system.

Limitations and Future Work. Our method aims to enhance ease of shape exploration by transforming the original parametric space into a potentially much smaller set of parameters. Such a transformation will invariably result in certain parts of the original shape space to be eradicated, making some of the shapes enabled by the original parameters to be no longer accessible. While this can be mitigated by increasing the number of bottleneck neurons, such an alteration will have an impact on user experience as exploration may become cumbersome therefore, future work will require more studies in the design space visualization techniques.

CONCLUSION

We introduced a method to enable an intuitive exploration of procedural modeling spaces within a low-dimensional space using autoencoder neural networks. We demonstrated that a combination of shape features with PM parameters as the input to autoencoder networks can automatically generate and organize a low-dimensional shape space primarily driven by shape similarity. We showed that users can create content with our system six to twelve times faster compared a conventional procedural modeling system.

REFERENCES

1. I. Alhashim, H. Li, K. Xu, J. Cao, R. Ma, and H. Zhang. 2014. Topology-varying 3d shape creation via structural blending. *ACM Trans. Graph.* 33, 4 (2014), 158.
2. Melinos Averkiou, Vladimir G Kim, Youyi Zheng, and Niloy J Mitra. 2014. Shapesynth: Parameterizing model collections for coupled shape exploration and synthesis. In *Computer Graphics Forum*, Vol. 33. 125–134.
3. Serge Belongie, Jitendra Malik, and Jan Puzicha. 2000. Shape context: A new descriptor for shape matching and object recognition. In *NIPS*, Vol. 2. 3.
4. Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. 2007. Greedy layer-wise training of deep networks. *Advances in neural information processing systems* 19 (2007), 153.
5. A. Bhattacharjee, Z. Shams, and K. Z. Sultana. 2006. New Constraints on Generation of Uniform Random Samples from Evolutionary Trees. In *CCECE*. 115–118.
6. S. Chaudhuri, E. Kalogerakis, S. Giguere, and T. Funkhouser. 2013. Attribit: content creation with semantic attributes. In *UIST*. 193–202.
7. D. Chen, X. Tian, Y. Shen, and M. Ouhyoung. 2003. On visual similarity based 3D model retrieval. In *Computer graphics forum*, Vol. 22. 223–232.
8. George Dahl, Abdel-rahman Mohamed, Geoffrey E Hinton, and others. 2010. Phone recognition with the mean-covariance restricted Boltzmann machine. In *Advances in neural information processing systems*. 469–477.

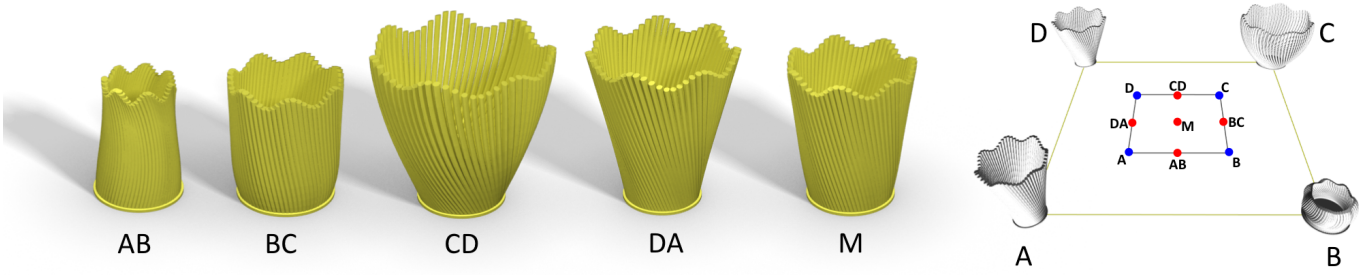

Figure 11. Interpolation results from our system.

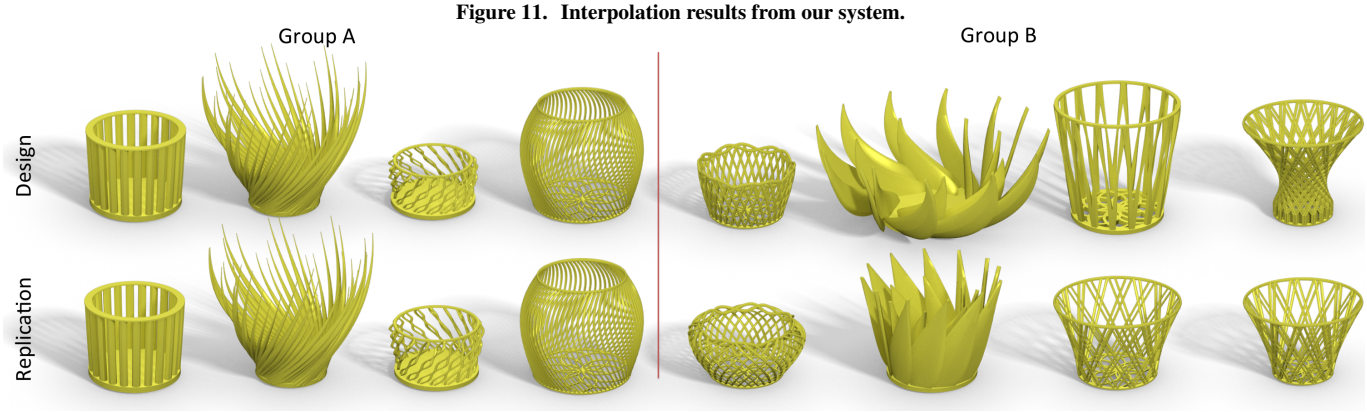

Figure 12. Group A: Users first designed models using the conventional system system, followed by a replication of their models using our system. Note the similarity of the replicas to the original models. Group B: Users first designed models using the our system, followed by a replication of their models using the conventional system. Note that the replicas are markedly different than the originals.

9. Alexey Dosovitskiy, Jost Tobias Springenberg, and Thomas Brox. 2014. Learning to generate chairs with convolutional neural networks. *arXiv preprint arXiv:1411.5928* (2014).
10. Kunihiko Fukushima. 1980. Neocognitron: A self-organizing network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics* 36, 4 (1980), 193–202.
11. T Germer and M Schwarz. 2009. Procedural Arrangement of Furniture for Real-Time Walkthroughs. In *Computer Graphics Forum*, Vol. 28. 2068–2078.
12. Geoffrey Hinton, Simon Osindero, and Yee-Whye Teh. 2006. A fast learning algorithm for deep belief nets. *Neural computation* 18, 7 (2006), 1527–1554.
13. Geoffrey E Hinton and Ruslan R Salakhutdinov. 2006. Reducing the dimensionality of data with neural networks. *Science* 313, 5786 (2006), 504–507.
14. S. Huang, A. Shamir, C. Shen, H. Zhang, A. Sheffer, S. Hu, and D. Cohen-Or. 2013. Qualitative organization of collections of shapes via quartet analysis. *ACM Trans. Graph.* 32, 4 (2013), 71.
15. Arjun Jain, Thorsten Thormählen, Tobias Ritschel, and Hans-Peter Seidel. 2012. Exploring Shape Variations by 3D-Model Decomposition and Part-based Recombination. In *Computer Graphics Forum*, Vol. 31. 631–640.
16. Evangelos Kalogerakis, Siddhartha Chaudhuri, Daphne Koller, and Vladlen Koltun. 2012. A probabilistic model for component-based shape synthesis. *ACM Trans. Graph.* 31, 4 (2012), 55.
17. P. Kearney, J. I. Munro, and D. Phillips. 2003. Efficient generation of uniform samples from phylogenetic trees. In *Algorithms in Bioinformatics*. 177–189.
18. Yuki Koyama, Daisuke Sakamoto, and Takeo Igarashi. 2014. Crowd-powered parameter analysis for visual design exploration. In *UIST*. 65–74.
19. Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. 1989. Backpropagation applied to handwritten zip code recognition. *Neural computation* 1, 4 (1989), 541–551.
20. Stefan Lienhard, Matthias Specht, Boris Neubert, Mark Pauly, and Pascal Müller. 2014. Thumbnail galleries for procedural models. In *Computer Graphics Forum*, Vol. 33. 361–370.
21. Aristid Lindenmayer. 1968. Mathematical models for cellular interactions in development. *Journal of theoretical biology* 18, 3 (1968), 280–299.
22. Bernd Lintermann and Oliver Deussen. 1999. Interactive modeling of plants. *Computer Graphics and Applications* 19, 1 (1999), 56–65.
23. Charles T Loop and Tony D DeRose. 1989. A multisided generalization of Bézier surfaces. *ACM Trans. Graph.* 8, 3 (1989), 204–234.
24. Joe Marks, Brad Andalman, and others. 1997. Design galleries: A general approach to setting parameters for

computer graphics and animation. In *ACM SIGGRAPH*. 389–400.

25. James McCrae and Karan Singh. 2009. Sketch-based Path Design. In *Proceedings of Graphics Interface 2009*. 95–102.
26. Radomír Měch and Gavin Miller. 2012. The *Deco* Framework for Interactive Procedural Modeling. *JCGT* 1, 1 (2012), 43–99.
27. Radomír Měch and Przemyslaw Prusinkiewicz. 1996. Visual models of plants interacting with their environment. In *ACM SIGGRAPH*. 397–410.
28. Paul Merrell, Eric Schkufza, and Vladlen Koltun. 2010. Computer-generated residential building layouts. In *ACM Trans. Graph.*, Vol. 29. 181.
29. Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. 2006. Procedural modeling of buildings. *ACM Trans. Graph.* 25, 3 (2006), 614–623.
30. Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y Ng. 2011. Multimodal deep learning. In *ICML*. 689–696.
31. Yoav IH Parish and Pascal Müller. 2001. Procedural modeling of cities. In *ACM SIGGRAPH*. ACM, 301–308.
32. Przemyslaw Prusinkiewicz. 1986. Graphical applications of L-systems. In *Proceedings of graphics interface*, Vol. 86. 247–253.
33. Przemyslaw Prusinkiewicz, Mark James, and Radomír Měch. 1994. Synthetic topiary. In *ACM SIGGRAPH*. 351–358.
34. Przemyslaw Prusinkiewicz and Aristid Lindenmayer. 1990. *The Algorithmic Beauty of Plants*. Springer-Verlag, New York.
35. M Ranzato, Fu Jie Huang, Y-L Boureau, and Yann LeCun. 2007. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *IEEE CVPR*. 1–8.
36. Michael Schwarz and Peter Wonka. 2014. Procedural Design of Exterior Lighting for Buildings with Complex Constraints. *ACM Trans. Graph.* 33, 5 (2014), 166.
37. Pierre Sermanet, Koray Kavukcuoglu, Soumith Chintala, and Yann LeCun. 2013. Pedestrian detection with unsupervised multi-stage feature learning. In *IEEE CVPR*. 3626–3633.
38. Lior Shapira, Ariel Shamir, and Daniel Cohen-Or. 2009. Image Appearance Exploration by Model-Based Navigation. In *Computer Graphics Forum*, Vol. 28. 629–638.
39. J. Shotton, J. Winn, C. Rother, and A. Criminisi. 2009. Textonboost for image understanding: Multi-class object recognition and segmentation by jointly modeling texture, layout, and context. *IJCV* 81, 1 (2009), 2–23.
40. Ruben M. Smelik, Tim Tutenel, Rafael Bidarra, and Bedrich Benes. 2014. A Survey on Procedural Modelling for Virtual Worlds. *Computer Graphics Forum* 33, 6 (2014), 31–50.
41. Sagi Snir and Satish Rao. 2010. Quartets MaxCut: a divide and conquer quartets algorithm. *IEEE TCBB* 7, 4 (2010), 704–718.
42. O Stava, S Pirk, J Kratt, B Chen, R Měch, O Deussen, and B Benes. 2014. Inverse Procedural Modelling of Trees. In *Computer Graphics Forum*.
43. George Stiny and James Gips. 1971. Shape Grammars and the Generative Specification of Painting and Sculpture.. In *IFIP Congress (2)*. 1460–1465.
44. Jerry O Talton, Daniel Gibson, Lingfeng Yang, Pat Hanrahan, and Vladlen Koltun. 2009. Exploratory modeling with collaborative design spaces. *ACM Trans. Graph.* 28, 5 (2009), 167.
45. Jerry O Talton, Yu Lou, Steve Lesser, Jared Duke, Radomír Měch, and Vladlen Koltun. 2011. Metropolis procedural modeling. *ACM Trans. Graph.* 30, 2 (2011), 11.
46. Nobuyuki Umetani, Takeo Igarashi, and Niloy J Mitra. 2012. Guided exploration of physically valid shapes for furniture design. *ACM Trans. Graph.* 31, 4 (2012), 86.
47. Carlos A Vanegas, Ignacio Garcia-Dorado, Daniel G Aliaga, Bedrich Benes, and Paul Waddell. 2012. Inverse design of urban procedural models. *ACM Trans. Graph.* 31, 6 (2012), 168.
48. Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. 2008. Extracting and composing robust features with denoising autoencoders. In *ICML*. 1096–1103.
49. Jason Weber and Joseph Penn. 1995. Creation and rendering of realistic trees. In *ACM SIGGRAPH*. 119–128.
50. Chris White. 2006. King Kong: the building of 1933 New York City. In *ACM SIGGRAPH Sketches*. 96.
51. S. J. Willson. 1999. Building phylogenetic trees from quartets by using local inconsistency measures. *Molecular Biology and Evolution* 16, 5 (1999), 685.
52. Peter Wonka, Michael Wimmer, François Sillion, and William Ribarsky. 2003. Instant architecture. In *ACM Trans. Graph.*, Vol. 22. 669–667.
53. Kai Xu, Hao Zhang, Daniel Cohen-Or, and Baoquan Chen. 2012. Fit and diverse: set evolution for inspiring 3D shape galleries. *ACM Trans. Graph.* 31, 4 (2012), 57.
54. M. E. Yumer, S. Chaudhuri, J. K. Hodgins, and L. B. Kara. 2015. Semantic Shape Editing Using Deformation Handles. *ACM Trans. Graph.* 34 (2015). Issue 4.