Interactive Design of Probability Density Functions for Shape Grammars

Minh Dang* EPFL
Stefan Lienhard† EPFL
Duygu Ceylan‡ Adobe Research
Boris Neubert§ KIT, EPFL
Peter Wonka¶ KAUST
Mark Pauly‖ EPFL

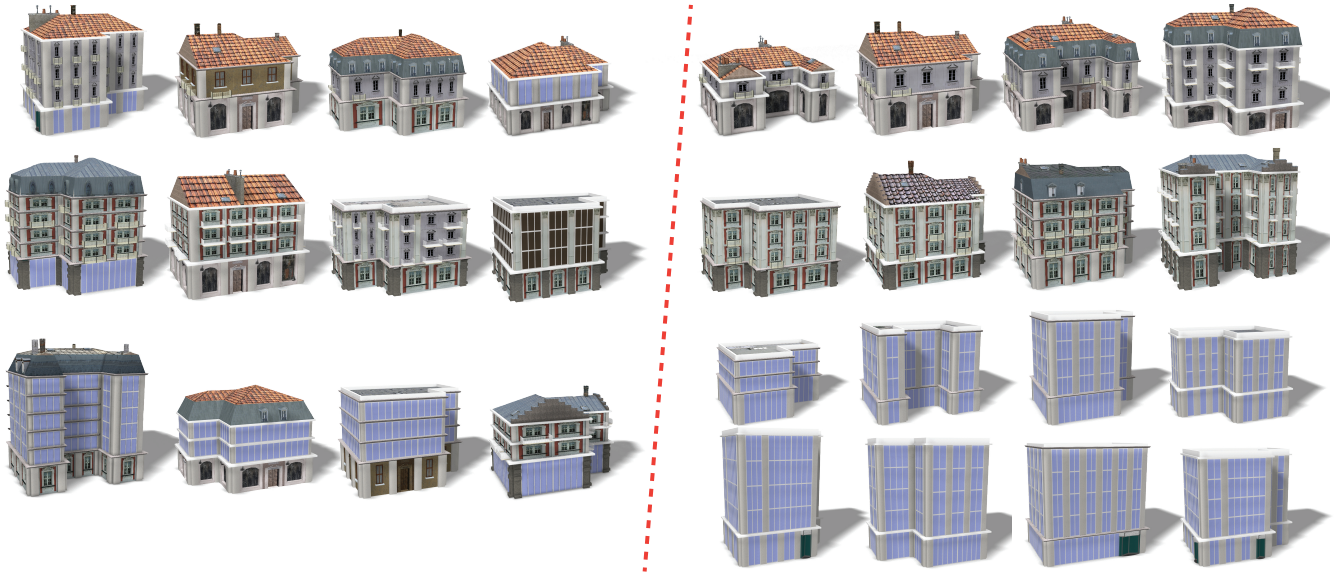

Figure 1: *(Left) Random models generated from a probabilistic building grammar. Although these models are visually plausible, they do not comply with a design scenario which also requires architectural plausibility, i.e. matching styles of ground floors, upper floors, and roofs (B1, see Table 2). (Right) Our framework takes user specified preference scores as input and learns a new model probability density function (pdf) which samples models (with consistent style) proportionally to their predicted preference scores. In this design scenario, office buildings received a higher preference score.*

Abstract

A shape grammar defines a procedural *shape space* containing a variety of models of the same class, e.g. buildings, trees, furniture, airplanes, bikes, etc. We present a framework that enables a user to interactively design a probability density function (pdf) over such a shape space and to sample models according to the designed pdf. First, we propose a user interface that enables a user to quickly provide preference scores for selected shapes and suggest sampling strategies to decide which models to present to the user to evaluate. Second, we propose a novel kernel function to encode the similarity between two procedural models. Third, we propose a framework to interpolate user preference scores by combining multiple techniques: function factorization, Gaussian process regression, auto-relevance detection, and l_1 regularization. Fourth, we modify the original grammars to generate models with a pdf proportional to the user preference scores. Finally, we provide evaluations of our user interface and framework parameters and a comparison to other exploratory modeling techniques using modeling tasks in five example shape spaces: furniture, low-rise buildings, skyscrapers, airplanes, and vegetation.

*e-mail:minh.dang@epfl.ch
†e-mail:stefan.lienhard@epfl.ch
‡e-mail:duygu.ceylan@gmail.com
§e-mail:boris.neubert@gmail.com
¶e-mail:pwonka@gmail.com
‖e-mail:mark.pauly@epfl.ch

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling; I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques

Keywords: procedural modeling, interactive design, regression, preference elicitation, active learning

1 Introduction

Procedural modeling using grammars is a very effective tool to generate a large variety of similar models. Grammars have been successfully used for modeling vegetation, buildings, building interiors, streets, sea shells, furniture, feathers, etc. Even though grammars are a great tool for modeling, generating good grammars is difficult and requires some programming skills. After talking to

many users of the procedural modeling software CityEngine[1], we can identify two groups of users of grammar-based procedural modeling. Non-expert users such as architects, urban planners, and regular artists without training in computer science, mainly use existing grammars, possibly with minor customizations. Expert users, typically technical artists or computer scientists with some art background, can generate new grammars from scratch. A common problem that both groups of users face is the difficulty to control the distribution of models generated by a grammar. In our paper we use the term *shape space* to describe the set of all models that can be generated by a grammar. Further, the *probability density function* (pdf) of a grammar determines how likely a model is to be generated. We propose new interactive interfaces and techniques to design the pdf of a given grammar, without editing the grammar rules directly. To motivate our research we discuss three example workflows of non-expert and expert users.

The goal of the first workflow is to generate a single interesting model. A user, e.g. an architect, might like to navigate the shape space of a grammar to get a design idea for a new building. Currently, the main solution is to randomly regenerate a model many times. Following pioneering work in exploratory modeling, e.g. [Marks et al. 1997; Talton et al. 2009], we would like to give the user the ability to navigate the shape space by designing a suitable pdf with high density in the neighborhood of the previously liked models.

The goal of the second workflow is to customize the pdf of an existing grammar (for non-expert users) or to fine tune a grammar written by an expert user for large-scale modeling, e.g. a complete city. For example, a user might want to control the distribution of building heights, building styles, building functions (e.g. residential vs. commercial), and the assets (meshes and textures) used for windows, doors, and ornaments. In simple cases these distributions can be controlled by a single parameter in the grammar, but in many cases the user will be interested in influencing aspects of the distribution that involve more than one parameter. Due to the context-free nature of most shape grammars, this is difficult to encode and would need major changes to the grammar rules. For example, it requires some work to encode that tall buildings should be predominantly gray and small buildings should be mainly brown.

The goal of the third workflow is to write a grammar that can generate a large variety of models. An expert user might start with a deterministic grammar that is able to generate a single high quality model. In a subsequent step, the user can introduce initial randomness to generate some minor variations, most of them having high quality. As the user adds more and more rules and randomness to the grammar, the probability of generating low quality models increases due to the trade-off between model variety and model quality. The reason for this degradation is again a limitation of context-free rules. The rules make design choices without global context, and it is difficult to coordinate design choices made by different rules. In Fig. 1 we illustrate two example problems: matching the style of ground floor and upper floors, and matching the roof style of a building with its facade style. The price for having a large shape space to choose from will typically be a larger percentage of undesirable models. Our work enables an expert user to focus on writing simpler rules encoding the structure of models and then using our proposed framework to model a pdf that eliminates most undesirable models.

This paper makes the following contributions to the state of the art:

- On the application side, we are the first to extend the concept of exploratory modeling from selecting a single model to interactively designing a pdf for a procedural shape space.
- On the user interface side, we propose strategies to display, sort, and sample models to enable the user to quickly provide preference scores.
- On the technical side, we integrate concepts from machine learning with context-free shape grammars. First, we propose a novel kernel function to encode the similarity between two procedural models. Second, we propose a framework to interpolate preference scores given by a user combining multiple techniques: function factorization, Gaussian process regression, autorelevance detection, and l_1 regularization. We show that our framework leads to better results than the kernel density estimation framework proposed by Talton et al. [2009]. Third, we propose the first algorithm to automatically generate a new grammar that approximates the target pdf well.

[1] http://www.esri.com/software/cityengine

2 Related Work

Procedural modeling. Our work is applicable to rule-based procedural modeling using grammars, L-systems, or production systems in general. The original shape grammars were proposed by Stiny [1975], but they are too complicated for most modeling tasks. Therefore, most work in computer graphics is based on a simplified version of shape grammars, set grammars [Stiny 1982]. L-systems, are very similar to grammars, but they use a parallel replacement strategy. They have been successfully used for modeling plants [Prusinkiewicz 1986; Prusinkiewicz and Lindenmayer 1990] and have been extended to query and interact with their environment during derivation to tackle more challenging plant modeling problems [Prusinkiewicz et al. 1994; Měch and Prusinkiewicz 1996]. Several grammars have been proposed for modeling streets and buildings, e.g. [Parish and Müller 2001; Wonka et al. 2003; Müller et al. 2006]. In our paper we build on CGA-shape [Müller et al. 2006] because there is commercial software available to author the grammars and to generate models. Other grammar extensions include the question of interactive modeling of grammar rules and modifications [Lipp et al. 2008] and guiding procedural modeling for additional control [Beneš et al. 2011; Talton et al. 2011].

In general, there is a trade-off between grammar complexity and expressiveness. Algorithms that try to learn grammar structure or grammar parameters tend to build on very simple, typically context-free grammars or only learn rule parameters [Müller et al. 2007; Bokeloh et al. 2010; Šťava et al. 2010; Simon et al. 2011; Wu et al. 2014]. Similarly, since we would like to learn a grammar pdf based on user preferences, we mainly use context-free rules in our input grammars. For more details about procedural modeling we refer the reader to a recent survey [Smelik et al. 2014].

Grammar induction. Currently, high-quality grammars are predominately written by hand. Grammar induction is a tool to automatically create grammars by generalizing a set of training models. A common approach to induce shape grammars is based on Bayesian inference using a minimum description length prior on the grammar structure [Talton et al. 2013; Martinovic and Van Gool 2013]. The grammar structure can be optimized using randomized algorithms (such as Markov chain Monte Carlo) with local moves including both *splitting* and *merging* operations. While our approach to grammar generation shares the use of the splitting operation, the goals as well as the techniques involved are fundamentally different.

Design exploration. We have recently seen several research efforts for exploratory modeling in the graphics community. An early inspiration for these efforts is the concept of design galleries introduced by Marks et al. [1997].

An important category of exploratory modeling efforts focus on ex-

ploring a pre-defined, discrete design space such as a collection of websites [Lee et al. 2010] or 3D shapes [Kleiman et al. 2013]. Several papers have proposed to use high-level feature attributes and utilized crowdsourcing tools to learn the relevance of such attributes [O'Donovan et al. 2014; Chaudhuri et al. 2013]. Averkiou et al. [2014] compute a hierarchical embedding of a large shape collection. In addition to exploring the given shape collection, they also compute the most dominant variation modes in this embedding as basis vectors. They utilize these basis vectors to define an inverse mapping from the embedding to the shape space and to generate new shapes. Our algorithm, on the other hand, operates on large design spaces defined by production grammars or other generative processes.

One approach to explore large design spaces is to provide the users with a discrete set of samples in an interactive framework. Samples can be generated by utilizing probabilistic models [Merrell et al. 2011] or evolutionary algorithms [Xu et al. 2012]. In another thread, researchers have proposed to locally explore the design space in the neighborhood of an optimized sample with respect to an energy function [Umetani et al. 2012; Yang et al. 2011; Deng et al. 2013]. Bao et al. [2013] have extended this idea to enable global exploration by extracting good pathways between different local spaces. In the context of procedural modeling, Lienhard et al. [2014] propose a strategy to sample a procedural space to generate representative thumbnail images.

Several researchers have adopted a parametric model to explore large design spaces assuming a direct mapping between the changes in the parameters and the output design. For example, Kerr and Pellacini [2010] have proposed to use sliders for the parameters of the design space to help the users select different materials. Koyama et al. [2014] fit a goodness function by defining a goodness value on a set of discrete samples based on crowdsource data and interpolating these values in the corresponding parameter space using radial basis functions. Kovar and Gleicher [2001] aim to construct the *legal* space of PostScript drawings by requiring user feedback on a set of initial samples and generate new samples by interpolation in the corresponding parameter space. Talton et al. [2009] explore parametric design spaces of trees and human shapes where they focus on avoiding invalid shapes by defining a density function based on manually selected valid models. Shapira et al. [2009] present an exploratory interface for recoloring images by parameterizing the design space using Gaussian Mixture Models. Brochu et al. [2010] present a Bayesian optimization approach to explore parametric animation spaces.

Our work is inspired by these research efforts focusing on exploration of large design spaces. In contrast, however, our main goal is to model the output of a production process such as a grammar as a density function and manipulate this function based on the user's preferences. This framework not only enables the user to navigate to the desired regions of the design space as most of the related work, but also to model complex distributions of the output designs. Moreover, since we do not assume an explicit parameterization of the design space, our approach is general and applicable to a variety of production systems.

3 Overview

3.1 Framework overview

The input to our framework is a shape grammar. The default values of the rule probabilities and rule parameters defined by the grammar impose a default density function over the procedural space S from which output shapes can be sampled. Our goal is to model a new density function that reflects the user preferences as closely as possible and then to generate a new grammar that samples shapes according to this new density function.

Our framework has the following major components: 1) A user interface that enables a user to provide preference scores for selected models in the shape space of a given input grammar (see Sec. 4). 2) A regression algorithm that interpolates the user defined preference scores (see Sec. 5). 3) An algorithm to generate new models according to the derived pdf (see Sec. 6).

3.2 Definitions

We use attributed, stochastic, context-free shape grammars that generate models by shape replacement. Each shape has a list of attributes and a label from the set of non-terminal symbols NT or from the set of terminal symbols T. While the number of attributes can vary depending on the grammar, we require at least the following: an oriented bounding box, called scope, a polyhedral mesh that is transformed to fit inside the scope, and a material identifier. We use an example grammar, called *toy grammar*, shown at the end of this subsection to illustrate various concepts in this paper. Random samples of this grammar are shown in Fig. 2. A shape grammar is defined as a tuple:

$$G = \langle NT, T, \omega, P, \Theta \rangle, \tag{1}$$

where $\omega \in NT$ is the starting symbol. The set of all symbols is denoted by $SYM = NT \cup T$. P is a set of context-free rules of the form

$$id_i \; prob_i : shape_i \rightarrow ShapeOp_i, \tag{2}$$

where $shape_i \in NT$ and $ShapeOp_i$ is a sequence of shape operations that generates replacement shapes $(succ_{i1}, \ldots, succ_{ik})$ with $succ_{ij} \in SYM$. The number j is called the *child index* and it can be used to identify a particular shape among the k replacement shapes. The probability of the rule being selected is $prob_i$ so that the probabilities for all rules that have $shape_i$ on the left hand side need to sum up to one. The rule id id_i is generated automatically and mainly used so we can refer to rules in the paper. The parameter vector Θ includes all information about probabilistic choices available in the grammar. These are a) the rule probabilities and b) all parameters of the distributions used to sample random values in shape operations, e.g. the minimum and maximum values of the uniform distributions to sample the heights h_1 and h_2 in the toy grammar. There are quite a few details about how shape operations could be specified, but these details are not important for the core contribution of the paper. In our examples we use grammars that are compatible with CityEngine and we use shape operations that modify the scope of a shape, e.g. via translation, rotation, and scaling, set the mesh of a shape (see the **i**() operation in rules id_2 to id_4), split shapes into multiple smaller shapes, or combine multiple shapes.

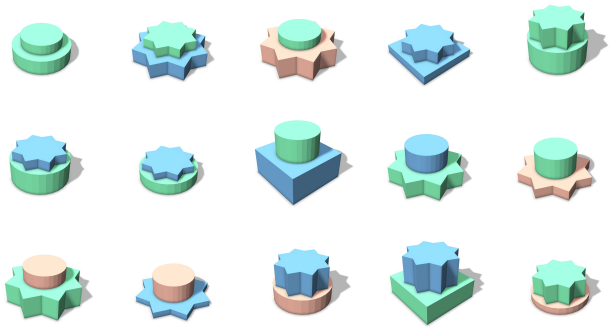

Figure 2: *Random samples of the toy grammar in Sec. 3.2.*

Figure 3: *Main components of the user interface. (a) Display: Models (sampled from the current pdf) are shown with their predicted preference scores (orange) and prediction uncertainties (blue, see Sec. 5.3) (b) Sorting: Models are sorted by the similarity to the selected model (green) (c) Selection and assignment: Multiple models can be selected and assigned the same preference score at once.*

In the toy grammar the rule id_1 stacks two shapes with height h_1 and height h_2 on top of each other to yield a top shape and a bottom shape. The rules id_2, id_3, and id_4 generate either a box, cylinder, or star geometry with equal probability $1/3$. id_5 is an example for a redundant rule. Finally id_6 to id_8 select a color for the generated geometry. This grammar already illustrates a limitation of context-free grammars. For example, it is not possible to modify the parameter vector Θ to ensure that the bottom shape has the color orange and the top shape has a random color. That requires changing the structure of the grammar (the rules in P) by duplicating and changing the rules id_2 to id_8. If there were 5 shapes stacked on top of each other it would already become unmanageable to encode the probabilities for various geometry and color combinations. Unfortunately, any context-free grammar requires an exponential number of rules to encode general joint probability distributions of multiple variables. Nevertheless, we can automatically generate context-free grammars approximating a given pdf. The grammar might be too complex to be human readable, but it is compatible with existing derivation engines and can be used for fast model generation.

Toy Grammar

NT = {MASS, MESH, MESH2}
T = {TerminalMesh}

attr h_1 = rand(1, 5)
attr h_2 = rand(1, 5)

id_1 1 : ω → **split**(y) { h_1 : MASS | h_2 : MASS }
id_2 1/3 : MASS → **i**("box") MESH
id_3 1/3 : MASS → **i**("cylinder") MESH
id_4 1/3 : MASS → **i**("star") MESH
id_5 1 : MESH → MESH2
id_6 1/3 : MESH2 → **setMaterial**("orange") TerminalMesh
id_7 1/3 : MESH2 → **setMaterial**("blue") TerminalMesh
id_8 1/3 : MESH2 → **setMaterial**("green") TerminalMesh

4 User Interface

In the following we will describe the user interface and the interaction possibilities and then give the details for the technical realization in Sec. 5. The main idea of the user interface is to enable the users to provide simple feedback about their preferences on a set of shapes and then use this feedback to compute a preference function that can assign a preference score to each model in the procedural shape space. The density function is the normalized preference function. Two important characteristics of the user interface are how quickly a user can provide a preference score and how much knowledge is gained by scoring the shown models.

The user interface is shown in Fig. 3. Models sampled from the procedural space are scaled and organized in one or two regular grids. Our framework not only predicts the preference score of each model but also can estimate the prediction uncertainty (see Sec. 5.3). Thus, for each model we optionally show its predicted preference score by an orange bar (in the range 0-100) and the uncertainty of the prediction in blue. Note that the range of scores will be normalized later so that only the ratio between scores is important and not their absolute values.

The user can choose how many models are shown in the grid(s). We typically use smaller grids, e.g. 4×4, to rate models and larger grids, e.g. 10×10, to check the results. The user can also zoom and pan to see individual models in more detail.

Further, the user can decide how the models are generated to populate the grid. There are five choices: a) uniform, b) grammar, c) density function, d) uncertainty, and e) already ranked. Uniform sampling selects candidates that are as dissimilar as possible using furthest point sampling among a set of candidates generated by the grammar. The grammar option simply samples models using the pdf of the original grammar. When the user selects the density function option we sample one grid from a pdf that is proportional to the preference function and a second grid to sample from the complementary pdf (100 minus the preference score and then normalized). Showing these two grids helps us to identify models that are undesirable, but have a predicted high preference value and models that are desirable, but have a predicted low preference value. The option d) allows sampling models according to their prediction uncertainty. The last option e) is used to review already ranked models to check for mistakes.

The user can also choose different sorting strategies. The models in the grid can be sorted according to their preference score, randomly, or according to an individual feature (see Sec. 5.1 for a description of features). The user can also select a set of models and sort the remaining models according to the highest similarity score when comparing to the selected models. The user is able to use standard

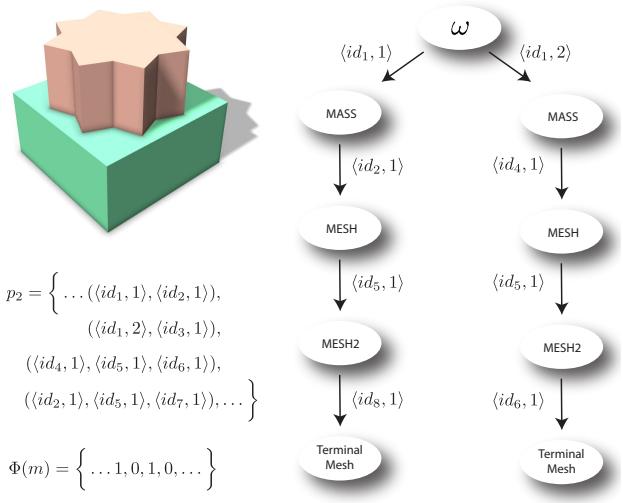

Figure 4: *Feature mapping. A model m generated from the toy grammar is shown together with its derivation tree. We show selected features as possible tree paths with effective length 2 (p_2) and the mapping of the shown model into feature space $\Phi(m)$. The mapping assigns a value v if the corresponding path exists v times in the derivation tree.*

selection techniques to provide preference scores for selected models. After changing scores for one or more models the user can trigger an update to re-estimate the density function and resample models shown in the visualization. This process is repeated until the user is satisfied with the modeled density function.

More complex density functions can be modeled as the normalized product of individual preference functions, also called *factors*. Typically, these factors describe the users preference about particular aspects of the shapes in the shape space. For example, for the simple grammar in Fig. 2 the user might want to specify her preference for the color and the geometry of the models separately. To enable the user to work with factors, we provide options to create, delete, and name preference functions (factors). Further, the user can set an active preference function to indicate which preference function she wants to work with.

5 Learning the Probability Density Function

The user interface described in the previous section enables the user to assign preference scores to selected models of the shape space. We now describe our technical solution to obtain a probability density function for the complete shape space using the following steps: 1) We map procedural models into a feature space (see Sec. 5.1) in which we assume preference scores are smoothly varying. 2) We define an overall preference function (see Sec. 5.2) by combining individual preference functions (factors). 3) We use a non-linear regression technique to interpolate the user preference scores specified for a set of models to the rest of the shape space. In this part we explain the regression model, the regularization, and the estimation of hyperparameters of the kernel function using automatic relevance detection (see Sec. 5.3).

5.1 Features

Finding a good function that maps a procedural model m to a feature space $\mathcal{X}$ is a challenging problem. We initially experimented with geometric features, but realized that such features need to be specifically designed for each grammar separately. For parametric models, e.g. Talton et al. [2009], one can simply use the model parameters as features directly and obtain good results. While grammar parameters have been used as features previously [Simon et al. 2011], this approach only works for simple grammars where all models have the same structure. For more interesting grammars, each generated model can be encoded by its derivation tree so that the fundamental problem of feature design is to encode the discriminative properties of the derivation tree (see Fig. 4 for an example). The nodes of the derivation tree represent shapes and they are labeled by the corresponding shape labels, i.e. terminal and nonterminal symbols of the grammar. The edges of the tree represent rules labeled by a tuple consisting of the rule id and the child index. The child index consistently numbers all shapes generated by the rule. In the following we propose to use the information encoded in the derivation tree to automatically extract features.

We can identify a path in the derivation tree by the sequence of labels on its edges. For example, for the derivation tree in Fig. 4 we can observe a path $(\langle id_1, 2\rangle, \langle id_4, 1\rangle)$. While there are many possible paths in the tree, in the following we only consider paths that follow the order of derivation, i.e. go from parents to children. We can observe that the paths encode the relationships between shapes and shape attributes. For example, if $(\langle id_4, 1\rangle, \langle id_5, 1\rangle, \langle id_6, 1\rangle)$ exists, there exists an orange star in m (either at the top or the bottom), while $(\langle id_1, 1\rangle, \langle id_2, 1\rangle, \langle id_5, 1\rangle, \langle id_8, 1\rangle)$ indicates that the bottom shape is a green box.

A model m generated by the grammar G is mapped into the feature space with a mapping function $\Phi(m)$ with the following components:

- the values θ_m assigned to all random parameters of shape operations $ShapeOp_i$ used to derive m (In a recursive grammar, we combine the parameters of all recursive rules up to a predefined level of recursion.)
- the number of times each shape label ($\in SYM$) occurs in the derivation tree
- the number of times a path occurs in the derivation tree.

Since there can be a large number of uniquely identifiable paths, we need to select a useful subset of such paths. We make the important observation that rule sequences of different lengths have varying discriminating power. While shorter sequences are capable of distinguishing models via the presence of more general properties (e.g. an orange box versus an orange cylinder), longer sequences separate more specific models (e.g. an orange box on the bottom versus an orange box on top). After analyzing many useful preference functions, we concluded that most basic concepts can be explained by short paths. Therefore, we start out using only paths of up to length k. We then gradually add longer paths as features if the current feature set becomes incapable of discriminating shapes that are assigned different preference scores by the user or the user requests more complex features.

There are several details to consider. First, some rules have no discriminative information. For example, in the toy grammar, if rule id_4 is executed, the child shape will always be replaced with rule id_5. We define the *effective length* of a path by discarding these non-informative rules. For example, the path $(\langle id_4, 1\rangle, \langle id_5, 1\rangle, \langle id_6, 1\rangle)$ has length 3, but effective length 2 since id_5 is not counted. We therefore categorize paths by their *effective length* instead of *length*. Second, since superfluous features increase the chance of overfitting, we typically start out setting path length $k = 1$. In Fig. 4 we show the extraction of selected features from a derivation tree.

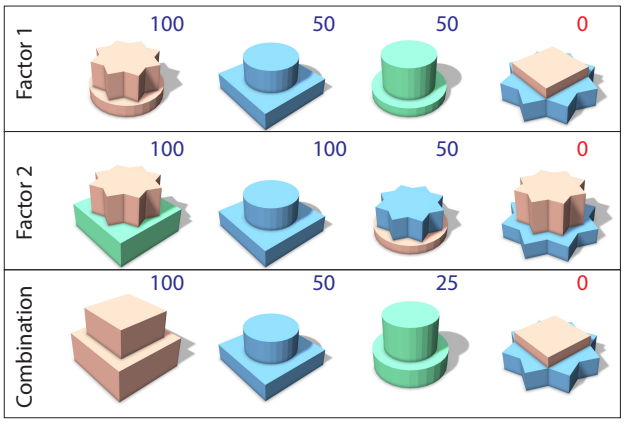

Figure 5: *A factorized preference function for the toy grammar. Two factors are trained separately and later combined to obtain the overall preference function. The factors are described in the text.*

5.2 Preference function factorization

In our system we allow the user to either model a single global preference function or to model the preference function as a product of individual preference functions, called *factors*. The idea of using multiple preference functions is that the user might want to specify preferences about particular aspects of the models instead of the models as a whole. Examples for factors would be the preference for certain materials, roof shapes, mass models, or window styles. A similar concept, visual attributes, has been employed in computer vision for various tasks such as object recognition [Farhadi et al. 2009].

As an example, consider the following two factors for the toy grammar from Sec. 3.2. 1) *Color*: The user prefers models that have the same color for the bottom and the top mesh with preference 100 for orange and 50 for green and blue. 2) *Geometry*: The user is interested in mass models where the bottom mesh is a box with preference 100, a cylinder with preference 50, and a star with preference 0. In Fig. 5 we show preference scores for the individual preference functions (factors) as well as their combination. A factor can involve one or multiple features. Most importantly, in most cases the sets of features necessary to evaluate different factors are not identical. This is what makes learning with individual factors more efficient compared to specifying a single preference function. Specifically, let m be a procedural shape which can be mapped into a D-dimensional feature space $\mathcal{X}$ by $\Phi(m) = \mathbf{x} = [x_1, x_2, \dots x_D]^T$ (in the rest of the paper we will use m and $\mathbf{x}$ interchangeably). We model the user preference function, $u(m) = u(\mathbf{x})$, in this space as the product of K factors:

$$u(\mathbf{x}) = \prod_{i=1}^{K} u^i(\mathbf{x}). \tag{3}$$

To train a factor u^i, the user provides preference scores for a set of models, considering only the corresponding semantic aspects of that factor. We then interpolate these scores to the rest of the shape space. We accomplish this task by using a kernel-based Bayesian regression technique called Gaussian Process Regression (GPR).

5.3 Gaussian Process Regression

GPR assumes a Gaussian Process prior over the preference function u^i. Thus, any finite set of n observations $[u^i(m_1), \dots, u^i(m_n)]^T$ can be considered as an n-dimensional point sampled from an n-variate Gaussian distribution. This method can work well with small training sets, which is necessary for our active learning framework where we expect reasonable results starting from the first iteration. For a more detailed discussion of GPR, we refer the reader to the work of Rasmussen and Williams [2005].

Similar to parameterizing a Gaussian distribution by its mean and the covariance matrix, a Gaussian Process prior over u^i is specified by a mean function indicating the prior bias of u^i and a kernel function $k(m, m')$ which specifies how similar $u^i(m)$ and $u^i(m')$ are. In our framework, we assume a zero-mean Gaussian Process prior and learn the kernel function from user input. This initially assumes that all models are not wanted, an assumption that was successfully employed in similar contexts, e.g. music ranking [Platt et al. 2001].

Prediction. Given a set of n models $\mathbf{m} = [m_1, m_2, \dots, m_n]^T$ and the corresponding user preference scores, $\mathbf{u}^i = [u^i(m_1), u^i(m_2), \dots, u^i(m_n)]^T$, our goal is to predict the preference $u^i(m^*)$ of any other model m^*.

Assuming $u^i(m)$ has a zero-mean Gaussian Process prior with the kernel function $k(m, m')$, $[\mathbf{u}^i, u^i(m^*)]^T$ has a joint $(n+1)$-variate zero mean Gaussian distribution with the following covariance matrix:

$$\mathbf{C}_{n+1} = \begin{bmatrix} k(m_1, m_1) & \dots & k(m_1, m_n) & k(m_1, m^*) \\ & & \vdots & \\ k(m_n, m_1) & \dots & k(m_n, m_n) & k(m_n, m^*) \\ k(m^*, m_1) & \dots & k(m^*, m_n) & k(m^*, m^*) \end{bmatrix}.$$

The conditional distribution $p(u^i(m^*)|\mathbf{u}^i)$ is also a Gaussian distribution, which leads to a closed form solution for $u^i(m^*)$ defined at the mean of this distribution as:

$$u^i(m^*) = \mathbf{k}^T \mathbf{C}_n^{-1} \mathbf{u}^i, \tag{4}$$

where $\mathbf{C}_n$ is the top-left $n \times n$ block of $\mathbf{C}_{n+1}$ and

$$\mathbf{k} = \begin{bmatrix} k(m_1, m^*) \\ \vdots \\ k(m_N, m^*) \end{bmatrix}.$$

In addition to predicting $u^i(m^*)$, GPR also provides the *prediction uncertainty* which measures the confidence of the prediction. This uncertainty is defined as the variance of $p(u^i(m^*)|\mathbf{u}^i)$ and calculated as follows:

$$\sigma^2(m^*) = k(m^*, m^*) - \mathbf{k}^T \mathbf{C}_n^{-1} \mathbf{k}. \tag{5}$$

We utilize the variance values in our sampling strategy based on prediction uncertainty (see Sec. 4) to determine the models presented to the user.

Kernel function. Given a kernel function, we interpolate the user preference scores from a set of shapes to the procedural space using GPR. The choice of the kernel function heavily influences the results of this regression process. Instead of using a fixed kernel function, we propose to use a family of parametrized kernel functions and learn the kernel parameters from user input.

To encode the similarity between procedural models m and m', we use an anisotropic kernel function k defined over the D-dimensional feature space $\mathcal{X}$ as:

$$k(m, m') = \theta_0 \exp\left(-\frac{1}{2} \sum_{d=1}^{D} \theta_d (x_d - x'_d)^2 \right) + \beta^{-1} \delta_{mm'},$$

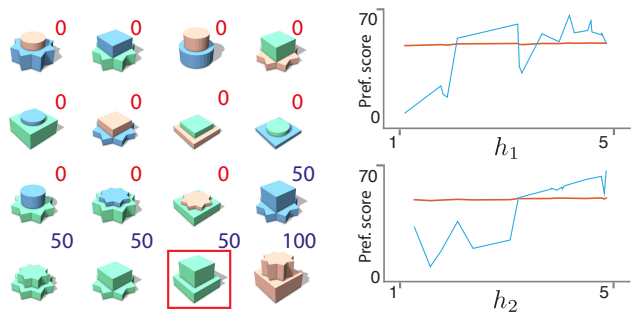

Figure 6: *Auto-relevance detection (ARD). We assign preference scores for the 16 models on the left to train the "Color" factor of the toy grammar (Sec. 5.2). Then we take the green box model highlighted by a red rectangle and create a lot of variations by changing the heights h_1 and h_2. The plots on the right show the predicted preference scores for h_1 and h_2 varying from 1 to 5. The preference scores learned with ARD are shown in orange and without ARD in cyan. As the orange lines remain almost constant when h_1 and h_2 change, this suggests that those two parameters are irrelevant to the preference scores.*

where x_d and x'_d are the features of m and m' respectively. $\beta^{-1}\delta_{mm'}$ is a small added noise to ensure positive-definite covariance matrix where $\delta_{mm'}$ is the Kronecker delta. The kernel hyperparameters $\theta_d, d \in [1, D]$ are adapted every time a new user preference score is specified. They are feature weights indicating the relevance of the corresponding features. Intuitively, features with small weights do not have a high influence on the preference scores. We learn these parameters by automatic relevance detection via maximizing the likelihood of the training data as explained next.

Automatic relevance detection. Typically, the influence of a feature depends on the preference function a user wants to model. Thus, instead of assigning a fixed θ_d to each feature, we adapt the hyperparameters of the kernel function based on user input. Additionally, our goal is to discard the irrelevant features by assigning them a 0-weight. We achieve this goal by optimizing for the hyperparameters $\boldsymbol{\theta} = (\theta_0, \theta_1, \ldots \theta_D)$ that maximize the log likelihood of the training samples:

$$\log p(\mathbf{u}^i|\boldsymbol{\theta}, \mathbf{m}) = -\frac{1}{2}\log|\mathbf{C}_n| - \frac{1}{2}\mathbf{u}^{iT}\mathbf{C}_n^{-1}\mathbf{u}^i - \frac{n}{2}\log(2\pi).$$

In order to avoid overfitting, we add a Lasso regularization term and minimize the following energy function:

$$E(\boldsymbol{\theta}) = -\log p(\mathbf{u}^i|\boldsymbol{\theta}, \mathbf{m}) + \lambda\sum_{d=1}^{D}|\theta_d|.$$

We define the hyperparameters to be non-negative, leading to $|\theta_d| = \theta_d$. Instead of defining constraints to avoid negative θ_d, we optimize for $\log(\theta_d)$ using the Conjugate Gradient method. The additional Lasso term also favors as few non-zero hyperparameters as possible resulting in the assignment of 0-weights to irrelevant features.

When training two attributes of the toy grammar (see Sec. 5.2), the block heights (h_1 and h_2) are irrelevant to the preference scores. This can be detected by our framework, as illustrated in Fig. 6.

6 Generating Models According to a PDF

After learning the preference function $u(m)$, the next step is to generate procedural models with a pdf proportional to $u(m)$. We provide three options for this task including 1) Parameter learning: modifying the grammar parameters without changing the rule structure. 2) Structure learning: generating a new context-free grammar by changing both rule structure and rule parameters. 3) Rejection sampling: standard rejection sampling as a post-process on generated models.

Parameter learning is very simple, but not very flexible. Thus the learned pdf does not match the target pdf well in most cases. Rejection sampling produces results exactly according to the target pdf, but it is slow and not compatible with existing grammar derivation engines. Structure learning is the default solution of our framework. It approximates $u(m)$ well and sampling can be done by the computed context-free grammar directly.

Parameter learning. This approach keeps the rule structure of the input grammar and only modifies rule parameters, specifically rule probabilities. We use the standard histogram-based estimation of rule probabilities of probabilistic context-free grammars (c.f. [Johnson 1998]). This naive solution does not perform well as shown in Table 3. The main problem is that it ignores the dependency between rule choices. For example, the choice of table bases may depend on how many legs this table has.

Structure learning. We propose this second option to deal with the aforementioned drawback. We use a *splitting operation* to split the original grammar and then train a mixture of grammars. While this generates a context-free grammar with more rules, it gives us the flexibility to encode conditional dependencies between rule choices.

We propose to use a splitting operation that is inspired by the concept of parent annotation in natural language processing [Johnson 1998]. The splitting operation splits a symbol in the original grammar into multiple different symbols by indexing it with the parent *edge labels* (see Sec. 5.1). The splitting operation therefore changes rules where the symbol appears on the right hand side and it duplicates all rules having the original symbol as left hand side. For example, in Fig. 7 a V-shape block at the ground level of the Skyscraper grammar can obtain a different symbol from a V-shape block in the upper level. The splitting operation does not change the shape space, but it increases the degrees of freedom to manipulate its pdf. Recursively splitting every symbol in the original grammar leads to an exponential growth in the number of rules. Thus, we only split the symbols involved in the relevant features detected by the learning process in Sec. 5.3.

After performing the described splitting operations we train a new grammar G' as the combination of K component grammars G'_i. (K is determined by the algorithm described later.) In this mixture, a component grammar G'_i is selected with a probability α_i:

$$\alpha_i : G' \rightarrow G'_i.$$

We start with a training set $T = (m_1, m_2, \ldots m_N)$ sampled from the target pdf $u(m)$ (using rejection sampling). We then find a set of hyperparameters $\boldsymbol{\eta}$ consisting of α_i and the grammar parameters of G'_i to maximize the log likelihood of the training set $\mathcal{L}(T : G') = \sum_{i=1}^{N}\log p(m_i|G')$. The value of $p(m|G')$ for generating a model m using G' is a weighted combination of the pdfs

from all component grammars G'_i:

$$p(m|G') = \sum_{i=1}^{K} \alpha_i p(m|G'_i).$$

Similar to learning Gaussian Mixture Models (GMMs), we optimize for $\boldsymbol{\eta}$ using an Expectation-Maximization approach. Both E-step and M-step (see Appx. A) are identical to those in the GMM case, except that in the M-step, we train grammar parameters instead of Gaussian parameters. To initialize the grammar mixture, we cluster the training set T using the kernel function learned previously, and train one grammar for each cluster using the aforementioned histogram-based method. To compute the clustering, we build a neighborhood graph, where two models are connected if their pairwise difference is smaller than a threshold. The pairwise difference between models is derived from the learned kernel. The threshold is set to 0.001 of the maximum difference. We then iteratively look for the model with the most neighbors, put it together will all its neighbors in a new cluster, and remove all elements of the new cluster from the graph.

Splitting symbols and clustering the training set significantly reduces the number of rules in our design scenarios. Nonetheless, for an artificially designed worst case preference function, an exponential growth of rules (see Sec. 3.2) cannot be avoided.

Rejection sampling. This algorithm does not compute a new grammar, but instead modifies the sampling as a post-process. For each generated model m, we calculate its generation pdf $p(m)$ by multiplying the associated rule probabilities and the pdf of parameters of shape operations (e.g. floor height). This model is accepted with the rate $\frac{u(m)}{Cp(m)}$, where the constant C is the upper bound of $\frac{u(m)}{p(m)}$ found empirically by sampling a large amount of models in a preprocessing step. Since rejection sampling is a post-process, it can be used in conjunction with the original grammar, as well as grammars generated by parameter learning or structure learning.

7 Evaluation

Example grammars. We designed four grammars for our tests: Furniture, Building, Skyscraper, and Airplane. To compare to previous work we also use the Weber & Penn parametric tree model implemented in Arbaro[2]. Table 1 shows some statistics describing the complexity of the grammars and the parametric model.

Implementation. Our framework is implemented in `C++` and we used a MacBook Pro with Intel i7 2.6 Ghz CPU for our tests.

Design scenarios. We generate a set of design scenarios with varying complexity shown in Table 2. For each scenario, we define the ground truth pdf by labeling a pre-generated set of up to 5000 models. In order to validate our results, we use Jensen-Shannon divergence (also known as information radius) [Manning and Schütze 1999] to compare the density functions designed using our interface and the ground truth density functions. We additionally measure the correlation between these two density functions and provide the correlation scores in the supplementary material. These values are computed based on the pre-sampled models.

[2] `http://arbaro.sourceforge.net`

	N_Θ	N_{SYM}	N_P
Furniture	50	38	75
Building	26	80	122
Skyscraper	39	27	61
Airplane	8	44	26
Tree	73	N/A	N/A

Table 1: *The table shows the number of parameters (N_Θ), the number of symbols (N_{SYM}) and the number of rules (N_P) for our input grammars and the parametric tree model.*

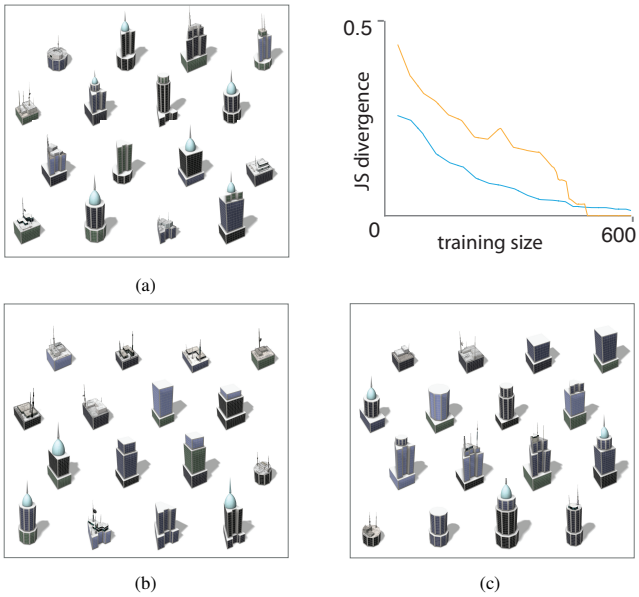

Figure 7: *(a) Random samples of the Skyscraper grammar. (b) Design scenario S1. (c) Design scenario S2. See Table 2 for the descriptions of S1 and S2. While S1 can be achieved using our initial set of features (random parameters of the shape operations, counts of the occurrence of shape labels in the derivation tree, and counts of paths, with effective length 1 in the derivation tree), S2 requires paths of length 2 as features. Jensen-Shannon divergence (S1 - blue, S2 - orange) are shown to evaluate the goodness of fit.*

We show the outputs of our design scenarios for the Skyscraper grammar in Fig. 7, the Airplane grammar in Fig. 8, and the Furniture and the Building grammar in Fig. 16. Each design task is represented by a 4×4 grid of models sampled from the designed density function. Bigger grids of size 10×10 can be found in the supplementary material.

Evaluation of the user interface. We provide multiple tests to evaluate the performance of our framework. In the first test, we evaluate the four different sampling strategies (see Sec. 4) used to sample the models to display. The results in Fig. 9 (a-b) empirically show that sampling from the current pdf gives good convergence rate and we choose this as the default strategy. We next evaluate the effect of the batch size, i.e. the number of models a user ranks (see Fig. 9 (c-d)). In each step, we show two grids of size $n \times n$, one sampled from the current pdf and the other from the complementary pdf, query for the preference scores, recompute the pdf and then resample the models to show in the grids. There are two aspects we consider when choosing a batch size. First, the learning efficiency: We can see from our experiments that smaller batch sizes lead to more informative samples. Second, the user interface speed: in our interface, the user can quickly rank multiple models without recomputing the pdf (using a combination of sorting and selection). This favors larger batch sizes. Considering both aspects, we propose that showing two 4×4 grids to the user is a good trade-off. In the third

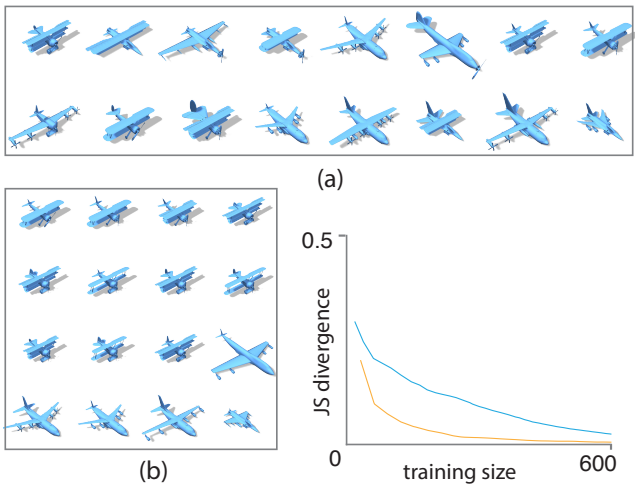

Figure 8: *(a) Random samples of the Airplane grammar. (b) Design scenario A1 (see Table 2). The JS divergence plot compares our method (orange) and Talton et al.'s method (cyan).*

test, we show the benefit of factorizing the preference function. We compare a factorized preference function and non-factorized preference function in Fig. 12. Using a factorized preference score leads to much better convergence. We also present an informal user study to test the speed and accuracy achieved with our interface in additional materials.

Evaluation of the feature design. In Fig. 10 we evaluate the parameter k (path length) (see Sec. 5.1) on the Furniture grammar

	Preference scores
F1	Valid tables with one leg (70), two legs (20) and four legs (10), non-standing tables (0)
F2	Round top & light wood tables (60), rectangular top & dark wood tables (40), others (0)
F3	Valid tables with steel materials (70) and wooden materials (30), non-standing tables (0)
F4	Valid tables with round top (70) and rectangular top (30), non-standing tables (0)
B1	Building styles: office (40), R1 (20), R2 (20), R3 (20), mixed styles (0)
B2	Big building (5-6 floors)& L-shape (50), small building (2-3 floors) & rectangular shape (50), others (0)
B3	Building shapes: L-shape (60), rectangular shape (40), others (0)
B4	Building size: big building (80), small building (20), others (0)
T1	Plausible tree (100), non-plausible tree (0)
S1	Skyscrapers with all rectangular blocks (60), V-blocks (20), cylindrical blocks (20), mixture of block types (0)
S2	Skyscrapers with rectangular base (100), cylindrical base (50), V-base (0)
A1	Old-style airplanes (Biplane or Fokker) (100), modern airplanes (commercial, transport and jet fighter) (50), airplanes with non-matching components (0)

Table 2: *Design scenarios for Furniture (Fi), Building (Bi), Skyscraper (Si), Airplane (A1) and Weber & Penn trees (T1). Valid tables are tables that stand by themselves. For aesthetic reasons, we also require legs and bases to match. Building styles are defined as follows. Office: glass windows, glass door and flat roof. R1: residential blocks with bright wall colors, Paris-like windows, ground floor shops. R2: residential blocks with simple windows and doors. R3: residential blocks with old-style windows and doors. Example models with their preference scores are given in the supplementary material.*

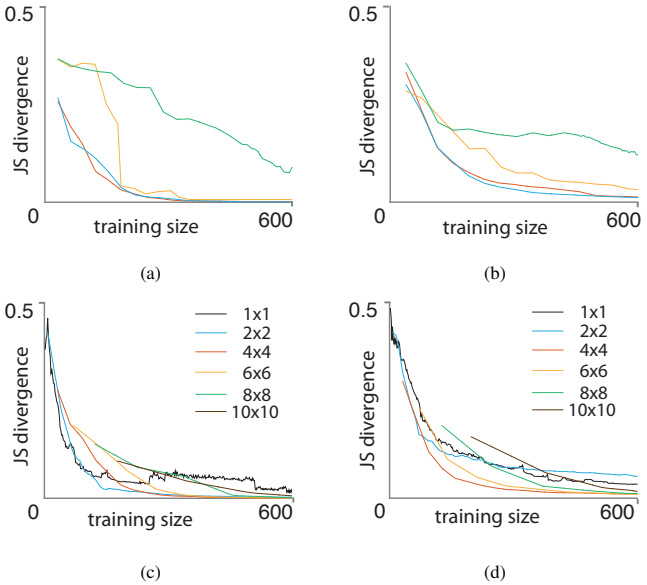

Figure 9: *(a) - (b) Comparison of different sampling strategies for the factor F1 (a) and B1 (b): sampling randomly (red), uniformly (green), by prediction uncertainty (orange) and based on the current pdf (cyan). (c) - (d) Comparison of sampling batch size for the factor F1 (c) and B1 (d) when using two grids sampled from the current pdf and the complementary pdf of size $n \times n$. Note that these curves start at different points due to different numbers of training data inserted in the first iteration.*

(Fig. 10a, scenario F4) and the Skyscraper grammar (Fig. 10b, scenario S2). For each scenario, we obtain 4 sets of features associated with paths in the production trees with maximum *effective* lengths of 1, 2, 3 and 4. While the convergence does not vary significantly in F4, the set of features with maximum length 4 (black) slows down the convergence in comparison to our proposed scheme (maximum length 2, orange). Note that the feature set of maximum length 1 (red) is not sufficient to learn the scenario S2.

Evaluation of framework parameters. We also show an evaluation of the noise level of the kernel function (Fig. 11a) and the weight for Lasso regularization (Fig. 11b). We empirically select the noise level of 0.01 and a weight of 0.1 for Lasso regularization. Our results are generated based on these parameters.

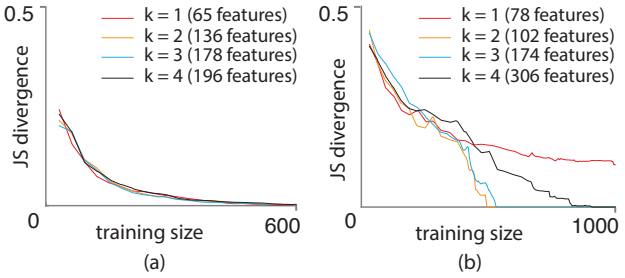

Figure 10: *Convergence of the pdf learning process when using sets of features associated with different maximum effective path lengths. We evaluate the convergence rate on the Furniture grammar (scenario F4, (a)) and the Skyscraper grammar (scenario S2, (b)). The feature sets with maximum effective path length 1 are shown in red, length 2 in orange, length 3 in cyan and length 4 in black.*

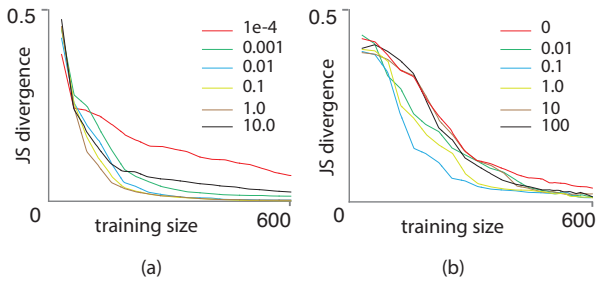

Figure 11: *(a) Evaluation of noise levels of the kernel function on the design scenario F1. Excessive noise (black) or limited noise (red) both result in undesirable convergence rate. (b) Evaluation of the lasso regularization factor λ on the design scenario B3. A suitable amount of lasso regularization improves the convergence in comparison to no regularization (red).*

Evaluation of generation strategies. Finally, in Table 3, we evaluate the performance of parameter learning and structure learning by comparing the JS divergence scores of their learned pdfs with the target pdf for all design scenarios. We also provide the correlation scores of the pdfs in the supplementary material. By modifying only the grammar parameters, parameter learning cannot approximate the target pdf well (around 0.20 divergence score). Much better results can be achieved with structure learning. One exception is B3, where the design scenario involves only one rule, which is simple enough to achieve by parameter learning. Nonetheless, structure learning performs equivalently well. To separately evaluate the effect of the split operation in structure learning, we train a mixture of original grammars (without any previous splitting operations). We perform this test on the design scenario S2 and observe a JS score of 0.18096 compared to 0.0816 for a full implementation of structure learning. This low-quality output is expected as S2 requires paths of effective length 2 as features to describe the geometry of the ground block.

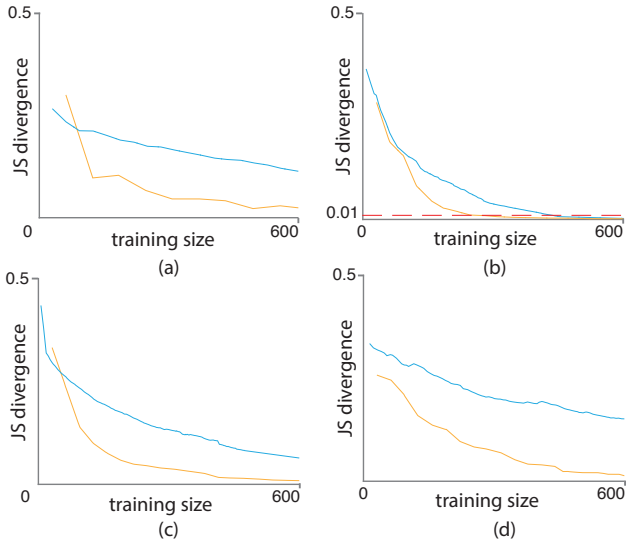

Figure 12: *(a) Benefit of factorizing the preference function. Training two factors F3 and F4 separately (orange) gives a better convergence than training their combination (cyan). (b - d) Our method (orange) converges faster than Talton et al.'s method (cyan) in the Furniture (F1) (b), Building (B1) (c) and Skyscraper (S1) (d) grammars.*

	F1	F2	F3	F4	S1	S2
Original	0.32824	0.44492	0.28037	0.28969	0.24646	0.31335
P-Learning	0.21976	0.25386	0.16915	0.18537	0.08854	0.19631
S-Learning	0.06850	0.06678	0.07820	0.08400	0.00090	0.08162
	B1	B2	B3	B4	A1	
Original	0.34923	0.34829	0.09858	0.11973	0.21040	
P-Learning	0.10541	0.28129	0.00710	0.12177	0.16748	
S-Learning	0.06661	0.08254	0.00712	0.07774	0.06269	

Table 3: *JS divergence score w.r.t the target pdf to compare parameter learning (P-Learning) and structure learning (S-Learning). The design scenarios (F1-F4, S1, S2, B1-B4, A1) are described in Table 2. Structure learning achieves significantly lower divergence scores. The JS scores from the original grammars (Original) are also included as a baseline for comparison.*

Comparison to kernel density estimation. Fig. 15 compares our framework to a kernel density estimation method based on Talton et al. [2009] using the same number of training examples. We modify the original kernel density estimation used by Talton et al. to account for models having different preference scores by using these scores as weights. This reduces the amount of training examples needed by the original method. Still, we can observe a better convergence using our framework. In Fig. 12 (b-d) and Fig. 8 we compare the convergence of our method and Talton et al.'s method for different design scenarios of our grammars. In the Furniture grammar (F1), our framework needs 245 models to achieve a divergence score of 0.01 while the density estimation requires 438 models. The convergence rate is clearly better with our method in the Building grammar (B1), the Skyscraper grammar (S1), and the Airplane grammar (A1). One practical difference between our framework and Talton et al. is that we can directly give feedback about models we do not want (by giving a preference score of 0).

8 Limitation and Future Work

Limitations. There are several limitations of our framework. First, our system starts sampling models using the pdf of the original grammar. As a result, models having very low initial probability are unlikely to be sampled. Second, in some design scenarios which involve discontinuities in the preference scores of continuous variables, our framework cannot learn the pdfs exactly with a finite number of training samples. Two examples are given in Fig. 13. This is a typical problem in Gaussian Process Regression as one cannot represent exactly a discontinuity using a finite number of gaussians. Nonetheless, our framework can learn these scenarios when the variables are discretized. The next limitation concerns the convergence of complex design scenarios involving features associated with long tree paths. Fig. 14 shows a slow convergence rate associated with a new design scenario of the Skyscraper grammar which requires a large feature set (maximum path length 4). Finally, as our features only contain branches in the production tree, our framework cannot handle complicated scenarios which require subtrees of the production tree as features. For example, in the Skyscraper grammar, one may prefer skyscrapers with a black cylinder block above a green box block but not a blue box block. As window colors and block geometries are in different branches, feature sets with only tree branches are not sufficient for this situation. Simply adding subtree features to our framework might not be a solution as this increases the number of features exponentially. This example also illustrates the fact that the success of the regression depends on how the initial grammar is written. We leave further improvement in our feature design for future research.

Future work. In future work we would like to extend our approach to the analysis of large shape repositories. While current approaches mainly use probabilistic models to encode shape vari-

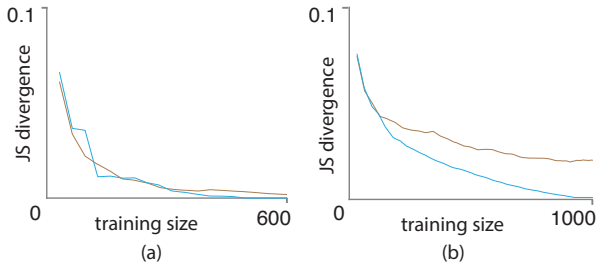

Figure 13: *We compare the learning accuracy of scenarios associated with continuous variables (orange) and discrete variables (with 100 discrete values each, cyan). We generate a box with three variables width, depth and height which take random values from 1.0 to 10.0. In (a), the scenario involves one variable (all boxes with height < 5.0 are preferred). In (b), the scenario requires a non-linear combination of these three variables (all boxes with volume < 125.0 are preferred). For continuous variables, it will require infinite number of training samples to learn the pdfs exactly. Nonetheless, our framework can learn these two scenarios with discrete variables.*

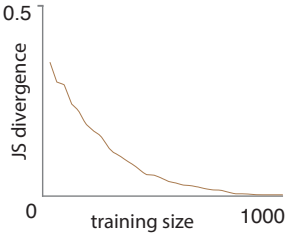

Figure 14: *We design an additional scenario for the Skyscraper grammar which requires the feature set with maximum path length 4. In this scenario only skyscrapers with 3 blocks are preferred and the preference score depends on the window color of the top block, in particular, skyscrapers with black windows in the top block receive a score of 50, green windows 30, blue windows 20 and other colors 0. The slow convergence is observed due to the size of the feature set.*

ations (e.g. [Kalogerakis et al. 2012]), we believe that it is more useful to use grammars and probabilistic models in combination. Grammars are great to encode variations in model topology (i.e. how parts are combined) and probabilistic models are great to encode part variations and part compatibility. Further, we would like to extend our framework to learn spatial distributions. For example, we could try to learn the distribution of residential, commercial, and industrial areas in a city or the distribution of tree species in a forest.

9 Conclusion

We presented a framework that enables a user to interactively design a probability density function for a shape grammar and to generate models according to the designed pdf. We thereby extended existing exploratory modeling tools that are suitable to select a single model from a shape space to modeling a distribution of shapes. We proposed a user interface to display, sort, and sample models to enable a user to quickly assign preference scores. To propagate user assigned preference scores to the complete procedural shape space, we proposed a novel kernel function to encode the similarity between two procedural models. This kernel function is then used for Gaussian process regression with auto-relevance detection and l_1 regularization. Finally, we introduced a structure learning method

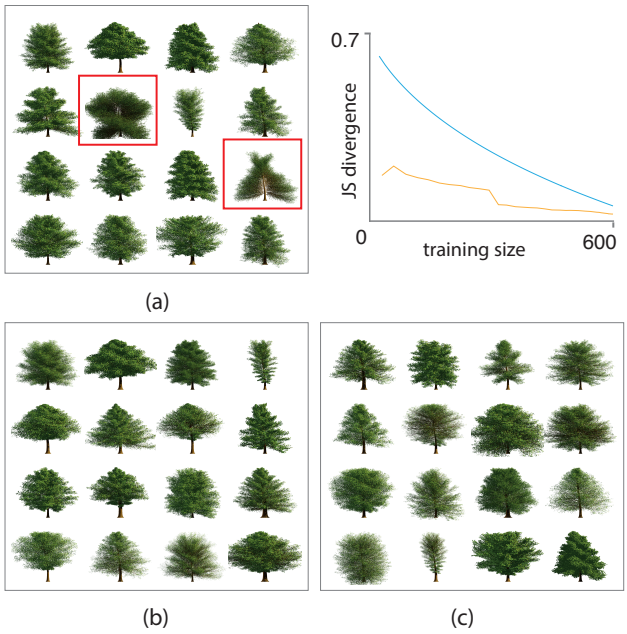

Figure 15: *(a) Weber & Penn trees with random parameters are often undesirable (red). Both Kernel Density Estimation [Talton et al. 2009](b) and our method (c) can bias the model distribution towards good tree samples. In verification with the ground truth using Jensen-Shannon divergence, our method (orange) converges faster than Talton et al.'s method (cyan).*

to automatically generate a new context-free grammar which approximates the learned pdf well. Our approach can benefit both non-expert and professional users to more effectively design with procedural and parametric models.

Acknowledgement

We thank the anonymous reviewers for their valuable comments. We also thank Cheryl Lau for assisting us in making the video. The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement n. 257453: COSYM.

A Training a Mixture of Grammars

E-step We find the affinity of G'_k to each model m_n. This affinity measures how likely m_n has been generated from G'_k

$$w_{kn} = p(G'_k|m_n, \boldsymbol{\eta}) = \frac{\alpha_k p(m_n|G'_k)}{\sum_{i=1}^{K} \alpha_i p(m_n|G'_i)}.$$

M-step The sum of affinity of each component grammar $N_k = \sum_{n=1}^{N} w_{kn}$ is the effective number of models that have been generated from G'_k. We can then update the mixture weights as follows:

$$\alpha_k^{new} = \frac{N_k}{N}.$$

To update the rule probabilities of G'_k, we count the number of appearance of each rule in every model m_n, weighted by the affinity w_{kn} and then normalize the rule counts as in the normal histogram-based approach.

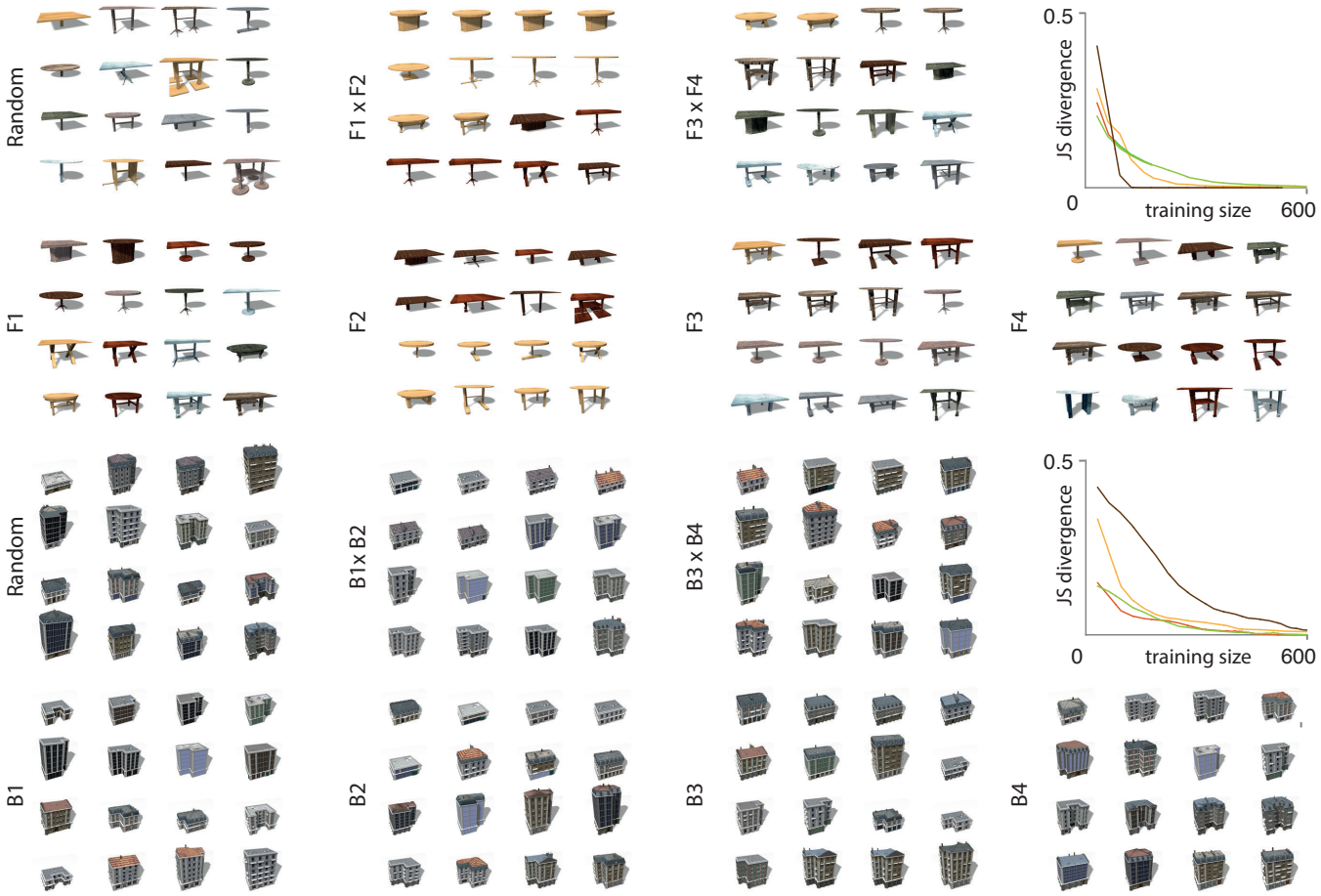

Figure 16: *Using our framework, a user can design different probability density functions to bias the procedural generation towards models with desired attributes (factors). These factors can then be mixed to obtain a combined density function. We show* 4×4 *samples for the initial distribution (random), the factors, and the factor combinations. See Table 2 for the descriptions of the factors. The designed pdfs are compared with ground truth using Jensen-Shannon divergence. Color codes: F1, B1 - orange, F2, B2 - brown, F3, B3 - red, F4, B4 - green.*

References

AVERKIOU, M., KIM, V., ZHENG, Y., AND MITRA, N. J. 2014. ShapeSynth: Parameterizing Model Collections for Coupled Shape Exploration and Synthesis. *Comp. Graph. Forum (Eurographics) 33*, 2, 125–134.

BAO, F., YAN, D.-M., MITRA, N. J., AND WONKA, P. 2013. Generating and Exploring Good Building Layouts. *ACM Trans. Graph. (Siggraph) 32*, 4, 122.

BENEŠ, B., ŠT'AVA, O., MĚCH, R., AND MILLER, G. 2011. Guided Procedural Modeling. *Comp. Graph. Forum (Eurographics) 30*, 2, 325–334.

BOKELOH, M., WAND, M., AND SEIDEL, H.-P. 2010. A Connection Between Partial Symmetry and Inverse Procedural Modeling. *ACM Trans. Graph. (Siggraph) 29*, 4, 104.

BROCHU, E., BROCHU, T., AND DE FREITAS, N. 2010. A Bayesian Interactive Optimization Approach to Procedural Animation Design. *ACM SIGGRAPH/Eurographics Symp. on Computer Animation*, 103–112.

CHAUDHURI, S., KALOGERAKIS, E., GIGUERE, S., AND FUNKHOUSER, T. 2013. Attribit: Content Creation with Semantic Attributes. *ACM Symp. User Interface Software and Technology*, 193–202.

DENG, B., BOUAZIZ, S., DEUSS, M., ZHANG, J., SCHWARTZBURG, Y., AND PAULY, M. 2013. Exploring Local Modifications for Constrained Meshes. *Comp. Graph. Forum (Eurographics) 32*, 2pt1, 11–20.

FARHADI, A., ENDRES, I., HOIEM, D., AND FORSYTH, D. 2009. Describing Objects By Their Attributes. *IEEE CVPR*, 1778–1785.

JOHNSON, M. 1998. Pcfg models of linguistic tree representations. *Comput. Linguist. 24*, 4, 613–632.

KALOGERAKIS, E., CHAUDHURI, S., KOLLER, D., AND KOLTUN, V. 2012. A probabilistic model for component-based shape synthesis. *ACM Trans. Graph. (Siggraph) 31*, 4, 55.

KERR, W. B., AND PELLACINI, F. 2010. Toward Evaluating Material Design Interface Paradigms for Novice Users. *ACM Trans. Graph. (Siggraph) 29*, 4, 35.

KLEIMAN, Y., FISH, N., LANIR, J., AND COHEN-OR, D. 2013. Dynamic Maps for Exploring and Browsing Shapes. *Comp. Graphics Forum (SGP) 32*, 5, 187–196.

KOVAR, L., AND GLEICHER, M. 2001. Simplicial Families of Drawings. *ACM Symp. User Interface Software and Technology*, 163–172.

KOYAMA, Y., SAKAMOTO, D., AND IGARASHI, T. 2014. Crowd-powered Parameter Analysis for Visual Design Exploration. *ACM Symp. User Interface Software and Technology*, 65–74.

LEE, B., SRIVASTAVA, S., KUMAR, R., BRAFMAN, R. I., AND KLEMMER, S. R. 2010. Designing with Interactive Example Galleries. *Proc. SIGCHI Conf. on Human Factors in Comp. Sys.*, 2257–2266.

LIENHARD, S., SPECHT, M., NEUBERT, B., PAULY, M., AND MÜLLER, P. 2014. Thumbnail Galleries for Procedural Models. *Comp. Graph. Forum (Eurographics) 33*, 2, 361–370.

LIPP, M., WONKA, P., AND WIMMER, M. 2008. Interactive Visual Editing of Grammars for Procedural Architecture. *ACM Trans. Graph. (Siggraph) 27*, 3, 102.

MANNING, C. D., AND SCHÜTZE, H. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press.

MARKS, J., ANDALMAN, B., BEARDSLEY, P. A., FREEMAN, W., GIBSON, S., HODGINS, J., KANG, T., MIRTICH, B., PFISTER, H., RUML, W., RYALL, K., SEIMS, J., AND SHIEBER, S. 1997. Design Galleries: A General Approach to Setting Parameters for Computer Graphics and Animation. *Proc. of SIGGRAPH*, 389–400.

MARTINOVIC, A., AND VAN GOOL, L. 2013. Bayesian grammar learning for inverse procedural modeling. *IEEE CVPR*, 201–208.

MERRELL, P., SCHKUFZA, E., LI, Z., AGRAWALA, M., AND KOLTUN, V. 2011. Interactive Furniture Layout Using Interior Design Guidelines. *ACM Trans. Graph. (Siggraph) 30*, 4, 87.

MÜLLER, P., WONKA, P., HAEGLER, S., ULMER, A., AND GOOL, L. V. 2006. Procedural Modeling of Buildings. *ACM Trans. Graph. (Siggraph) 25*, 3, 614–623.

MÜLLER, P., ZENG, G., WONKA, P., AND GOOL, L. V. 2007. Image-based Procedural Modeling of Facades. *ACM Trans. Graph. (Siggraph) 26*, 3, 85.

MĚCH, R., AND PRUSINKIEWICZ, P. 1996. Visual Models of Plants Interacting with Their Environment. *Proc. of SIGGRAPH*, 397–410.

O'DONOVAN, P., LĪBEKS, J., AGARWALA, A., AND HERTZMANN, A. 2014. Exploratory Font Selection Using Crowdsourced Attributes. *ACM Trans. Graph. (Siggraph) 33*, 4, 92.

PARISH, Y. I. H., AND MÜLLER, P. 2001. Procedural Modeling of Cities. *Proc. of SIGGRAPH*, 301–308.

PLATT, J. C., BURGES, C. J., SWENSON, S., WEARE, C., AND ZHENG, A. 2001. Learning a gaussian process prior for automatically generating music playlists. In *NIPS*, 1425–1432.

PRUSINKIEWICZ, P., AND LINDENMAYER, A. 1990. *The Algorithmic Beauty of Plants*. Springer.

PRUSINKIEWICZ, P., JAMES, M., AND MĚCH, R. 1994. Synthetic Topiary. *Proc. of SIGGRAPH*, 351–358.

PRUSINKIEWICZ, P. 1986. Graphical Applications of L-systems. *Proc. on Graphics Interface/Vision Interface*, 247–253.

RASMUSSEN, C. E., AND WILLIAMS, C. K. I. 2005. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. MIT Press.

SHAPIRA, L., SHAMIR, A., AND COHEN-OR, D. 2009. Image Appearance Exploration by Model-Based Navigation. *Comp. Graph. Forum (Eurographics) 28*, 2, 629–638.

SIMON, L., TEBOUL, O., KOUTSOURAKIS, P., AND PARAGIOS, N. 2011. Random Exploration of the Procedural Space for Single-View 3D Modeling of Buildings. *IJCV 93*, 2, 253–271.

SMELIK, R. M., TUTENEL, T., BIDARRA, R., AND BENES, B. 2014. A Survey on Procedural Modelling for Virtual Worlds. *Comp. Graph. Forum (Eurographics) 33*, 6, 31–50.

ŠT'AVA, O., BENEŠ, B., MĚCH, R., ALIAGA, D. G., AND KRIŠTOF, P. 2010. Inverse Procedural Modeling by Automatic Generation of L-systems. *Comp. Graph. Forum (Eurographics) 29*, 2, 665–674.

STINY, G. 1975. *Pictorial and Formal Aspects of Shape and Shape Grammars*. Birkhauser Verlag, Basel, Switzerland.

STINY, G. 1982. Spatial Relations and Grammars. *Environment and Planning B 5*, 1, 5–18.

TALTON, J. O., GIBSON, D., YANG, L., HANRAHAN, P., AND KOLTUN, V. 2009. Exploratory Modeling with Collaborative Design Spaces. *ACM Trans. Graph. (Siggraph Asia) 28*, 5, 167.

TALTON, J. O., LOU, Y., LESSER, S., DUKE, J., MĚCH, R., AND KOLTUN, V. 2011. Metropolis Procedural Modeling. *ACM Trans. Graph. (Siggraph) 30*, 2, 11.

TALTON, J., YANG, L., KUMAR, R., LIM, M., GOODMAN, N., AND MĚCH, R. 2013. Learning design patterns with bayesian grammar induction. *ACM Symp. User Interface Software and Technology*, 63–74.

UMETANI, N., IGARASHI, T., AND MITRA, N. J. 2012. Guided Exploration of Physically Valid Shapes for Furniture Design. *ACM Trans. Graph. (Siggraph) 58*, 9, 86.

WONKA, P., WIMMER, M., SILLION, F., AND RIBARSKY, W. 2003. Instant Architecture. *ACM Trans. Graph. (Siggraph) 22*, 3, 669–677.

WU, F., YAN, D.-M., DONG, W., ZHANG, X., AND WONKA, P. 2014. Inverse Procedural Modeling of Facade Layouts. *ACM Trans. Graph. (Siggraph) 33*, 4, 121.

XU, K., ZHANG, H., COHEN-OR, D., AND CHEN, B. 2012. Fit and Diverse: Set Evolution for Inspiring 3D Shape Galleries. *ACM Trans. Graph. (Siggraph) 31*, 4, 57.

YANG, Y.-L., YANG, Y.-J., POTTMANN, H., AND MITRA, N. J. 2011. Shape Space Exploration of Constrained Meshes. *ACM Trans. Graph. (Siggraph Asia) 30*, 6, 124.