Automatic Generation of Constrained Furniture Layouts

PAUL HENDERSON, University of Edinburgh
KARTIC SUBR, University of Edinburgh
VITTORIO FERRARI, University of Edinburgh & Google Research

Efficient authoring of vast virtual environments hinges on algorithms that are able to automatically generate content while also being controllable. We propose a method to automatically generate furniture layouts for indoor environments. Our method is simple, efficient, human-interpretable and amenable to a wide variety of constraints. We model the composition of rooms into classes of objects and learn joint (co-occurrence) statistics from a database of training layouts. We generate new layouts by performing a sequence of conditional sampling steps, exploiting the statistics learned from the database. The generated layouts are specified as 3D object models, along with their positions and orientations. We incorporate constraints using a general mechanism – rejection sampling – which provides great flexibility at the cost of extra computation. We demonstrate the versatility of our method by accommodating a wide variety of constraints.

1 INTRODUCTION

Large scale virtual environments are an important feature across a plethora of applications such as massively-multiplayer-online computer games, movies, and driving simulators [Dosovitskiy et al. 2017] used to train self-driving cars. Manual authoring of environments at these scales is impractical and there is a need for algorithms that can automatically generate realistic virtual environments. To be considered useful, it is important that automatic algorithms are able to accommodate constraints stipulated by domain experts. In this paper, we address the problem of automatically generating furniture layouts for indoor scenes. We propose a simple algorithm to automatically populate empty rooms with realistic arrangements of 3D models of furniture and other embellishments using occurrence and placement statistics learned from an input database of layouts of CAD models. Our algorithm is amenable to a variety of a priori constraints. Our model can be used in conjunction with layout generators [Ma et al. 2014] to automatically generate complete indoor virtual environments.

The computer graphics literature is rich with methods that enable 3D content creation, from landscapes [Cordonnier et al. 2017] and urban sprawls [Parish and Müller 2001] to individual objects such as furniture [Li et al. 2017], buildings [Müller et al. 2006; Nishida et al. 2016], etc. These methods involve varying degrees of user-interaction to achieve realism and/or aesthetic appeal. Procedural approaches rely on parametric controllability while methods that are posed as optimization rely on constraint-driven controllability. A third class of methods adopts a data-driven approach to generate or edit models based on features learned from training examples [Emilien et al. 2015; Funkhouser et al. 2004; Li et al. 2017].

We propose a new data-driven, probabilistic, generative model for 3D room layouts. Our model learns statistics from a database [Song et al. 2017; Zhang et al. 2017] containing over 250,000 rooms that are designed by humans. We categorize over 2,500 3D models from the database and learn the conditional statistics across these categories. The output space of this learned model is a high-dimensional combination of discrete and continuous variables. Our model is
Generating realistic furniture layouts: A common approach is to adopt an energy-based formulation [Handa et al. 2016; Merrell et al. 2011; Sadeghipour Kermani et al. 2016; Yu et al. 2011], with potentials between objects to impose constraints and preferences. The method of Handa et al. [2016] samples room layouts probabilistically by optimizing a pairwise energy term using simulated annealing, with random initialization. Sadeghipour Kermani et al. [2016] propose a similar method, but separate the sampling of classes and counts from the spatial arrangement. Fisher et al. [2012] propose a method that learns from a dataset of 130 layouts, to embellish parts of rooms by adding relevant objects around (e.g., chairs around a table) as well as on (e.g., books on a table) furniture. It does not model entire layouts of rooms containing complex arrangements of furniture. Another class of methods suggests improvements to layouts based on partial input [Merrell et al. 2011; Yu et al. 2011]. A recent method [Fu et al. 2017] exploits knowledge of how human agents interact with indoor environments to synthesize entire room layouts from partial input such as room shape and a few classes of objects that are to be placed in the rooms. Liang et al. [2017] proposed a two-step probabilistic generative model. Objects are first generated based on statistics learned from a database and then placed in the room using a Bayesian model. The former step does not consider inter-object relationships and the latter is realized using MCMC sampling.

Concurrent work: Two new approaches, which are yet to be published, propose data-driven strategies to generate indoor furniture layouts. One of them [Qi et al. 2018] represents indoor scenes using a probabilistic grammar, using a conditional Gibbs distribution as prior on its parse graphs. The conditioning parameter is learned from a large database. Their approach requires considerable manual modeling including specification of potential functions and grouping relations between objects such as chairs and tables. Novel layouts are generated using MCMC sampling, along with simulated annealing. A second concurrent approach to generating furniture layouts [Wang et al. 2018] exploits the power of convolutional neural networks (CNNs). This method generates layouts by using three different CNNs to decide whether to add furniture, what furniture to add, and where to place it. Both these approaches are trained with the same dataset that we use in this paper. While both methods are fully automatic, neither of them supports inputting user-specified constraints.

1.1 Related work

Priors for understanding structure in indoor scenes: Choi et al. [2013] performed 3D object detection and layout estimation by inferring spatial relations between objects using a discriminative, energy-based formulation. They do not present a generative model over layouts. Zhao and Zhu [2011; 2013] built a probabilistic grammar model, using specifically engineered production rules, over cube-based 3D structures constituting parts of rooms. This grammar generates arrangements of coarse blocks and does not result in layouts of entire rooms. Similarly, treating objects as cuboids, Del Pero et al. [2012] proposed a generative model over room size and layout using learned distributions for the dimensions of the cuboids. The model does not learn inter-object relationships such as co-occurrence or relative locations. Although the importance of such relationships was discussed in follow-up work [Del Pero et al. 2013], inter-object relationships were not incorporated in the generative model.

Generating realistic furniture layouts: A common approach is to adopt an energy-based formulation [Handa et al. 2016; Merrell et al. 2011; Sadeghipour Kermani et al. 2016; Yu et al. 2011] with potentials between objects to impose constraints and preferences. The method of Handa et al. [2016] samples room layouts probabilistically by optimizing a pairwise energy term using simulated annealing, with random initialization. Sadeghipour Kermani et al. [2016] propose a similar method, but separate the sampling of classes and counts from the spatial arrangement. Fisher et al. [2012] propose a method that learns from a dataset of 130 layouts, to embellish parts of rooms by adding relevant objects around (e.g., chairs around a table) as well as on (e.g., books on a table) furniture. It does not model entire layouts of rooms containing complex arrangements of furniture. Another class of methods suggests improvements to
We begin by sampling a room type (e.g. kitchen, living room), then sequentially sample furniture instances, conditioned on the room type and instances already sampled. We partition rooms into cells and sample objects and their positions within these cells (Fig. 3) such that geometric intersections will not occur. We begin with furniture objects (Sec. 2.2), which define the structure of the room. Then, in an embellishment step (Sec. 2.3), we sample instances of other categories, given the furniture items and their locations. The overall sampling process is illustrated in Fig. 2.

The parameters used for sampling instances of objects, for cell assignments and positioning within a cell are learned from the dataset (Sec. 3). These are simple parametric models, which are human-interpretable and modifiable.

2.1 Classes and categorization

The dataset contains CAD models, where each model \( m \) is assigned an object class \( \phi_1(m) \) such as “television”, “bathtub”, “armchair”, etc. There is a total of 2500 CAD models and 170 object classes in the collection. We introduce an additional set of labels called categories: furniture, small objects such as books and laptops, wall objects such as picture frames, and ceiling objects such as lamps. We manually specified a second mapping \( \phi_2 \) from models to categories (Fig. 4).

2.2 Sampling furniture

We sample counts of furniture instances both as singletons (i.e. individual objects), and in motifs and abutments, which are spatially-related groups learnt from the training data. For each instance, we sample an orientation and padding, and a cell in the layout it is assigned to. After all furniture counts and instance parameters have been sampled, we position the resulting objects deterministically. The room structure is finalized when cells (and hence the room) are sized to accommodate the objects and their paddings.

Singletons: We sample singletons using algorithm 1, where functions SampleNumInstances, SampleCell, SampleOrientation and SamplePadding sample from distributions whose parameters are learned from training data. The arguments to these functions signify what the underlying distributions are conditional on. At each iteration (line 6 of algorithm 1), cells expand to fit the sampled objects, ensuring no intersections between objects.

Motifs: Motifs are groups of items that are present together in many examples of the dataset, such as a table with chairs around it. We sample counts and instance parameters for each motif following lines 5–10 of algorithm 1, but in this case \( m \) represents a motif rather than a singleton CAD model. Then, we set the relative offsets and orientations between items within a motif as observed in a randomly selected instance of the motif in the training database. This non-parametric strategy for determining relative placement eliminates the need for testing if there are geometric collisions/intersections.

Abutments: Abutments are groups of items that appear in rows, abutting one another, with variations in their sequence, e.g. a row of cabinets in a kitchen along with a dishwasher, sink, refrigerator or a washing machine. Again, we sample counts and instance parameters for each abutment using lines 5–10 of algorithm 1, where \( m \) now represents a class of abutment. The furniture items within an abutment are modeled as a Markov chain with a terminal state; for each instance of the abutment, we sample from this Markov chain to obtain a specific sequence and number of CAD models. The transition probabilities of the Markov chains are learned during training.
Algorithm 1: Sampling singleton furniture instances

Input: \( r \) is the room type

1: function SampleFurniture\( (r) \)
2: for each object class \( c \) do
3: \( n_c \leftarrow 0 \)
4: for each model \( m \) with \( (\phi_1(m) = c) \) \& (\phi_2(m) = \text{furniture}) \) do
5: \( n_m \leftarrow \text{SampleNumInstances}(m, n_c, r) \)
6: for \( j \leftarrow 1 \) to \( n_m \) do \( \triangleright \text{n}_m \text{ instances of } m \]
7: \( k_j \leftarrow \text{SampleCell}(m) \)
8: \( \theta_j \leftarrow \text{SampleOrientation}(m, k) \)
9: \( p_j \leftarrow \text{SamplePadding}(m) \)
10: end for
11: \( n_c \leftarrow n_c + n_m \) \( \triangleright \text{accumulate count} \]
12: end for
13: end function

Output: for each CAD model \( m \):
(i) number of instances \( n_m \) that we place;
(ii) parameters \( \{ k_j, \theta_j, p_j \}_{j=1}^{n_m} \) of each instance of \( m \).

2.3 Embellishment

**Ceiling objects:** Given a room type \( r \), we draw a single CAD model \( m \) randomly according to a discrete probability mass function (pmf) \( \theta_{c,a} \) over models in this category. The number of instances of \( m \) is determined using \( \text{SampleNumInstances}(m, 0) \). This number is rounded up so that it can be factorized into a product of integers and the objects are positioned on a grid.

**Wall objects:** For each CAD model \( m \) that is a wall object, we determine the number of instances in similar fashion to furniture in lines 2–5 of algorithm 1. Each instance is then assigned to a wall uniformly randomly and its position on the wall is a combination of a Normal distribution along the \( Y \) axis and Uniform distributions in \( X \) and \( Z \). The parameters of the Normal distribution are learned (conditioned on \( m \)). If this results in a geometric conflict (intersection with other wall objects, doors, etc.) we reject the sampled location and repeat the process until there are no conflicts.

**Small objects:** For each furniture instance with CAD model \( m \), we sample small objects non-parametrically conditioned on \( m \) and \( r \). We choose a random instance of \( m \) in a room of type \( r \) from the database, and replicate the configuration of small objects associated with that instance.

2.4 Algorithm summary

To summarize, we first randomly sample the type of room \( r \) from a discrete distribution over 9 room types found in the database. The distribution (pmf) of \( r \) is learned during training. Then, we sample furniture items conditioned on \( r \): first singletons, then motifs and finally abutters. The numbers and instances of each item are determined by parameters learned during training. Once all furniture items are sampled, and assigned to cells, we use a deterministic placement algorithm that calculates their final positions in the room. Then, we sample ceiling objects and wall objects conditioned on \( r \) and the furniture placed. Finally, we sample small objects conditional on the furniture in the room and \( r \).

3 Training

**Dataset:** We use a large dataset of ground-truth room layouts to learn parameters that are then used for automatically generating layouts. SUNCG [Song et al. 2017; Zhang et al. 2017] is a new dataset of 45000 apartment layouts, created by humans, and separately verified as reasonable by humans. Each apartment has an average of 8.1 rooms; the majority are annotated with the room type. The apartments are designed with an online tool, and contain objects of 170 classes, represented by around 2500 CAD models. There are 4.5M object instances; each consists of a reference to a CAD model, and its location and orientation in the room.

**Number of instances:** We model the number of instances \( n_m \) of each CAD model as being conditional on the model \( m \), the room type \( r \), and on the number \( n_c \) of furniture instances already sampled for the class \( \phi_1(m) \). The distribution (pmf) \( \theta_{n,m} \) over count bins \( \{0, 1, 2, 3, 4, > 4\} \) is calculated as a histogram (normalized) over all scenes in the database of type \( r \). Further, a Poisson distribution is fitted to the observed \( n_m \) in all scenes with \( n_m > 4 \). \( \text{SampleNumInstances} \) (line 5 of algorithm 1) is implemented in two steps. First, we draw an indicator variable according to \( \theta_{n,m} \). If this variable is less than or equal to 4, then we return it as the number of instances. Otherwise, we return a sample from the Poisson distribution.

**Instance attributes:** For each model \( m \), during training, we calculate a pmf over 9 cells (4 corners, 4 edges and internal) by normalizing the histogram of occurrences. We implement \( \text{SampleCell} \) (line 7 of algorithm 1) by returning a cell according to this pmf. For models in internal cells, we count the number occurrences where they are aligned (positively or negatively) with respect to any axis and the number of "non-aligned" instances, and use this to learn a pmf. We implement \( \text{SampleOrientation} \) (line 8 of algorithm 1) by sampling an indicator variable according to this pmf for orientations. If this variable indicates non-alignment, we sample an orientation uniformly at random. Finally, we model padding around CAD models as a 4D diagonal-covariance Normal distribution conditioned on the CAD model \( m \). The dimensions correspond to padding on each side of the object: in-front-of, behind, to-the-left-of and to-the-right-of. \( \text{SamplePadding} \) (line 9 of algorithm 1) returns a sample from this 4D Normal distribution. The knowledge learnt during training is interpretable: Fig. 12b,c show values from the pmfs, indicating typical placements of objects, while Fig. 12a shows typical locations where we place various classes. In both cases, these agree well with human intuition on interior design.

**Motif discovery:** We search the training set for all joint occurrences of a given \( K \)-tuple of classes (e.g. table, chair, chair) within a room. For every occurrence of one of these classes – designated as the base object – we calculate displacements of the centres of the other objects relative to the base object. We model these displacements as points in a \( 2(K - 1) \)-dimensional space and cluster them with a
Dirichlet process mixture model (DPMM) [Rasmussen 2000], fitted by variational inference. We use Gaussian clusters with diagonal covariance and fit a DPMM per K-tuple of classes. We calculate the area inside the 2d contour of the location for each element in the motif; if all of these are less than a threshold, then the cluster is accepted as a motif. We store the CAD models, relative locations, and orientations for every occurrence assigned to the cluster; one of these will be selected when instantiating the pattern. Some examples of motifs we discover are given in Fig. 5.

**Abutment discovery:** We discover abutments in two stages. First, we gather sets $S_i$ of sequences of CAD models, where each set will ultimately become an abutment pattern. Each sequence of CAD models represents an abutting series of instances in the training set. Then, for each $S_i$, we calculate the transition probabilities for the corresponding Markov chain, as maximum-likelihood estimates given the CAD model sequences $s \in S_i$. More precisely, we collect such sets $S_i$ in a collection $T$, initialising $T$ to be empty. While doing so, we maintain the invariant that $\forall i \neq j, S_i$ and $S_j$ do not contain any sequences that share CAD models. For each room in the training set, we find all pairs of objects that abut, based on their rotated bounding-boxes touching at an edge. These pairs are combined transitively to form full sequences of objects $s_j$, each being a row of abutting objects of some orientation. For each object-sequence $s_j$, ignoring those with just two objects, we check if any of its CAD models appears in any sequence in a set $S_i \in T$ already created. If so, we add the object-sequence to $S_i$; if not, we create a new one storing just $s_j$ and add it to $T$. In the first case, we also check that adding the sequence to $S_i$ has broken the invariant that sets do not share CAD models; if it has, we merge sets until the invariant holds again. At the end of the above process, each $S_i \in T$ contains many sequences of CAD models, each of which we will treat as a sample from the Markov chain $M_i$. It is then straightforward to learn the transition probabilities for $M_i$ by maximising the likelihood of all the sequences $s_j \in S_i$. Some examples of abutments we discover are given in Fig. 6.

![Example abutments](image)

**Fig. 5. Motif discovery. Left:** We use DPMM clustering for K tuples of object occurrences and identify motifs as those tuples for which displacements from a base object in the tuple is within some threshold. **Right:** Examples of motifs that we automatically discover in SUNCG. Each colour corresponds to a different object in the pattern; we overlay 200 occurrences of each pattern to illustrate its variability. The red objects are the base objects of the patterns.

**Fig. 6. Statistics of abutments observed in the training database are recorded in a transition matrix, along with a terminal state (blue circle). We synthesize abutments by generating Markov chains using the learned transition probabilities.**

### 4 CONSTRAINTS

Our generative model accommodates diverse constraints using rejection sampling as a general mechanism, i.e. we sample layouts until we obtain one that satisfies all constraints. We demonstrate the versatility of our generative model using some example constraints. Incorporating other constraints can be achieved similarly as long as a given layout can be verified to satisfy them. Since our sampling process is fast (tens of milliseconds per room), any inefficiency due to rejection sampling is outweighed by its ability to serve as a common mechanism to impose a wide range of constraints (Tab. 2). In some special cases, we can avoid rejection sampling by allowing users to explicitly manipulate parameters of distributions learned (Fig. 11).

**Room type and size:** As the room type $r$ has no ancestors in our model, it can directly be assigned a constrained value, avoiding rejection sampling entirely. Since room size is a continuous value, and the probability of any sample satisfying this is zero, we allow a small tolerance on each dimension (2% in all our examples).

**Traversability:** A layout is traversable if there exists a path between all points in free space (regions with no furniture) and from all points in free space to all doors in the room. To verify this, we first rasterise an orthographic projection of the furniture onto the floor plane at a fixed resolution and identify free space as the complement of this footprint. We calculate $P$ (areas where people can stand or pass) via morphological erosion of the free space using a circular kernel of radius 0.25m and also add regions on doors to $P$. Similarly we calculate regions $A$ that require access using morphological erosion, but with a larger kernel. Then we verify traversability by checking whether $x$ is reachable from $y$, $\forall x, y \in A$ via some $\{z\} \subset P$.

**Gap placement:** Ensuring there is a gap at a particular location allows users to augment layouts with their own 3D models, that are not part of our system. In order to make rejection sampling efficient, rather than just discarding layouts until one that satisfies the constraint is found, we directly place a ‘gap instance’ in a suitable cell, ensuring that no object will occupy the relevant space. Note that some rejections will still occur, as cell locations are not known precisely until all furniture items are placed.

**Object placement:** We allow users to place instances of CAD models known to our system, at specific locations – e.g. a bed against a particular wall. Similarly to placing gaps, we ensure that a suitable
instance is placed in the relevant cell, thereby greatly reducing the chances of rejection.

**Doors, windows and refurnishing**: We model door and window specification using a combination of gap-placement, at edges of rooms, and traversability (the relevant area is included in $P$). This allows us to support refurnishing existing rooms from SUNCG – that is, generating new furniture, while retaining the existing size/shape, doors, and windows. This is valuable for generating complete, realistic rooms without any user input.

5 RESULTS

In this section, we present qualitative results to highlight the samples (with and without constraints) generated using our model, and quantitative results measuring performance. We also assess the quality of our generated layouts via a simple user study. All rendered images were obtained using path-tracing [Jakob 2010]. The execution times reported in this paper were obtained using our unoptimized and sequential Python implementation, on an Intel Xeon E5-2620v3 processor, using less than 1GB of RAM.

5.1 Generating layouts

**Unconstrained output**: We show some output examples from our generative model, without any constraints specified, in Figure 7. Our model produce results without objects intersecting and mimics the diversity found in the training dataset – both in terms of the types of rooms as well as the objects in them. The co-occurrence and relative placements of objects are also realistic and natural. Unconstrained sample layouts are generated in 0.04s on average.

**Examples with constraints**: Figure 8 shows examples of layouts where the room size and placement of one object was specified by the user. Note that the other sampled objects in the room are automatically chosen, and placed harmoniously. e.g. for the first image, the constraint was “place a bed near the top right corner”. Our method automatically places nightstands on either side.

Figure 9 shows sample layouts where the shapes of the rooms and the locations of doors where specified as constraints. Note that doors are unobstructed.

Figure 11 shows examples of layouts where a user has specified particular clearances to be respected around specific object classes. The bar plots on the first column (solid blue) show the ranges of clearances learned during training on the left (L), front (F), right (R) and back (B) of the models sampled from four chosen classes (sofa, double bed, sink and bath). The pink squares on the bar plots depict user modifications of the learned parameter (dashed blue rectangles). For each specified constraint (rows), four sample outputs are visualized (columns), and the impact of the user specification is shown using pink arrows as annotation. In this particular example, the constraints are imposed by directly editing learned parameters rather than using rejection sampling, which leads to faster runtime.

**Constraints producing uncharacteristic layouts**: Large generative models run the risk of over-fitting their training set. However, a benefit of training a constrainable generative model is that we can generate arbitrary numbers of rooms fulfilling constraints, that are never (or very rarely) fulfilled in the training dataset. We demonstrate this using random sets of reasonable constraints and identify those sets of constraints which are not jointly satisfied by any room in the SUNCG dataset. Then, we use our model to sample a room that does satisfy the constraint. Examples are given in Fig. 10.

**Runtime with constraints**: We measured the runtime penalty due to constraints as the ratio of the time taken to generate a valid,
5.2 User study

We assessed the realism of layouts generated using our model via a user study comparing its output to human-designed rooms from the SUNCG database. We presented 1400 pairs of images to eight non-expert users and asked them to identify the image with a more realistic, or natural, layout of objects. In each case, one image was a ground-truth (human-designed) layout from SUNCG, and the other was a sample from our model; the order of the two images was randomised for each pair.

Unconstrained: We sampled several hundred random layouts from our model without constraints, and a similar number of ground-truth layouts from SUNCG. We presented images in the form of either overhead renderings or first-person camera views from inside the room. The observed user preferences are given in Table 3a. In first-person views, users preferred our layouts; in overhead views, our layouts are indistinguishable from ground-truth up to statistical significance.
we randomly selected rooms from SUNCG, and used their size and with constraints as we did layouts without constraints, but using object constrained, our layouts are indistinguishable from ground-truth.

Constrained: We assessed room layouts generated by our model with constraints as we did layouts without constraints, but using only overhead renderings. We considered two representative settings for constrained generation: (i) fixing the room size and placement of one object (indicated by a pink box), choosing a combination of constraints that is not satisfied by any layout in SUNCG. Our model is able to sample rooms, fulfilling the constraints, despite not having seen such examples at training time.

| Constraint        | Runtime penalty |
|-------------------|-----------------|
| room type         | 1×               |
| direct parameter editing | 1×               |
| object class exclusion | 1×               |
| traversability    | 1.2×             |
| object placement  | 35×              |
| gap placement     | 45×              |
| room size         | 170×             |
| size + doors + windows | 2800×            |

Table 2. Average runtime penalties for different constraints, compared to an unconstrained sample. Our unoptimized Python implementation generates unconstrained samples at about 0.04s per room layout. With a 50× penalty, layouts are generated in 2s which is still reasonably fast.

truth up to statistical significance. With room size and the positions of doors constrained, users preferred human-designed layouts.

Comparison with [Wang et al. 2018]: We compared randomly generated unconstrained samples from our model with images obtained from a preprint of concurrent work [Wang et al. 2018]. We presented 4 users with 192 pairs, each using one of 44 images from their preprint and a randomly drawn layout from ours. For 57.8 ± 7.0% of pairs presented, users preferred our image. Thus, our unconstrained layouts are competitive with state of the art in layout generation.

6 DISCUSSION

Comparison with related/concurrent work: Probabilistic generative methods for room layouts are challenging to sample from. Often the sampling is not guaranteed to converge to a valid layout, especially when many objects are present. e.g. the model proposed by Handa et al [2016]. This particular model also requires that the number of objects, and size of the room, be specified manually. Our model performs comparably with concurrent, unpublished work [Wang et al. 2018] that learns millions of parameters over days of training. In 57.8 ± 7.0% of pairs presented, users preferred our layouts to theirs. In addition to accommodating constraints easily, our model has another advantage in that the parameters learned are over semantically meaningful concepts (categories such as furniture) allowing direct modulation of learned parameters as shown in Figure 11. Although we manually specified padding constraints, they could be calculated from alternatives such as human-centric affordances [Qi et al. 2018].

Inter-object relationships: We explicitly discover and encode relationships across classes of objects using patterns such as motifs and abutments. These patterns capture higher order relationships (not just pairwise); in the case of abutments, they are able to model sequences of variable lengths which may not be present in the training database. Figure 13 shows unnatural layouts generated when inter-object relationships such as patterns and abutments are not modeled. Additionally, implicit relationships are captured between different CAD models of the same class in a given layout. The generative process does not favor a large item from a class if multiple small items from that class have been sampled.

Efficient implementation of constraints: For many of the constraints listed in Section 4, rejection sampling can be avoided using

| Viewpoint          | Ours pref. | Constraints | Ours pref. |
|--------------------|------------|-------------|------------|
| overhead           | 48.1 ± 6.6%| size + object | 45.2 ± 6.8% |
| 1st person         | 58.1 ± 6.0%| size + door   | 35.2 ± 5.4% |

Table 3. Percentage of image-pairs where non-expert users preferred (i.e. deemed more realistic) a layout sampled from our model, as opposed to a ground-truth layout from SUNCG (‘Ours pref.’). Higher is better, with 50% indicating that our samples are indistinguishable from ground-truth. Ranges are the 95% confidence interval [Efron and Tibshirani 1986]. (a) Unconstrained layouts; (b) Constrained layouts.
Fig. 11. Samples from our model, with user-specified clearance constraints. The left column shows the default (blue) and user-specified (pink) padding ranges in meters for each side (left/front/right/back) of the indicated object; the remaining columns show samples drawn from our model with the constraint applied, with the specified padding regions indicated.

Fig. 12. Many parameters that are learned during training are human-interpretable. (a) Heat-maps showing locations where our model places different objects. Clockwise from top-left: shower, cabinet, sofa, double bed, dining table and toilet. (b) Furniture classes with highest (top) and lowest (bottom) probability $p_{\text{edge}}$ of being at the edge of a room rather than the interior (c) furniture classes highest (top) and lowest (bottom) probability $p_{\pi/2}$ of being at an angle that is a multiple of $\pi/2$.

Fig. 13. Samples from our model, but without patterns. Left: the kitchen cabinets and appliances are scattered, rather than placed adjacent to one another (as enabled by abutments). Centre: the chairs are scattered, rather than placed around the dining table (as enabled by motifs). Right: the two night-stands (lower left) are not at the expected location near the bed (as enabled by motifs).

alternative implementations. For example, space constraints may be tailored at the class level by biasing the 4D Normal distribution learned for padding. Figure 11 shows direct editability of learned parameters. Example layouts produced by the modified distribution are shown on the right, along with the effects of the user manipulation.
We have presented an efficient, probabilistic, data-driven, generative model for indoor furniture layouts. The algorithm used to generate layouts is simple and the parameters learned from training data are human-interpretable. We demonstrated that our model is able to accommodate a variety of constraints using rejection sampling as well as editing of learned parameters. We presented qualitative and quantitative results through rendered layouts, performance measurements and a user study.

7 CONCLUSION

We have presented an efficient, probabilistic, data-driven, generative model for indoor furniture layouts. The algorithm used to generate layouts is simple and the parameters learned from training data are human-interpretable. We demonstrated that our model is able to accommodate a variety of constraints using rejection sampling as well as editing of learned parameters. We presented qualitative and quantitative results through rendered layouts, performance measurements and a user study.

REFERENCES

Wongun Choi, Yu-Wei Chao, Caroline Pantofaru, and Silvio Savarese. 2013. Understanding Indoor Scenes using 3D Geometric Phrasers. In CVPR.
Guillaume Cordonnier, Eric Galin, James Gain, Bedrich Benes, Eric Guérin, Adrien Peytavie, and Marie-Paule Cani. 2017. Authoring Landscapes by Combining Ecosystem and Terrain Erosion Simulation. ACM Trans. Graph., 36, 4, Article 134 (2017).
Luca Del Pero, Joshua Bowdish, Daniel Fried, Bonnie Kermerg, Emily Hartley, and Kobus Barnard. 2012. Bayesian geometric modeling of indoor scenes. In CVPR.
Luca Del Pero, Joshua Bowdish, Daniel Fried, Bonnie Kermerg, Emily Hartley, and Kobus Barnard. 2013. Understanding Bayesian rooms using composite 3D object models. In CVPR.
Aleksey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. 2017. CARLA: An Open Urban Driving Simulator. In Proceedings of the 1st Annual Conference on Robot Learning. 1–16.
Bradley Efron and Robert Tibshirani. 1986. Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy. Statistical science (1986), 54–75.
Arnaud Emilius, Ulisse Vimont, Marie-Paule Cani, Pierre Poulin, and Bedrich Benes. 2015. WorldBrush. Interactive Example-based Synthesis of Procedural Virtual Worlds. ACM Trans. Graph. 34, 4 (2015).
Matthew Fisher, Daniel Ritchie, Manolis Savva, Thomas Funkhouse, and Pat Hanrahan. 2012. Example-based Synthesis of 3D Object Arrangements. In SIGGRAPH.
Yibiao Zhao and Song-Chun Zhu. 2013. Scene Parsing by Integrating Function, Geometry and Appearance Models. In CVPR.
Yibiao Zhao and Song-Chun Zhu. 2011. Image Parsing with Stochastic Scene Grammar. In NIPS.
Siyuan Qi, Yixin Zhu, Siyuan Huang, Chenfanfu Jiang, and Song-Chun Zhu. 2018. Deep Understanding of 3D Indoor Scenes from RGB-D Images. ACM Trans. Graph. 37, 4, Article 140 (2018), to appear.

ACKNOWLEDGMENTS

This research was supported by a Google Faculty Research Award.

ACM Trans. Graph., Vol. –, No. –, Article –. Publication date: August 2018. 2018-08-24 01:49. Page 10 of 1–10.