Deep Convolutional Priors for Indoor Scene Synthesis

KAI WANG, Brown University
MANOLIS SAVVA, Princeton University
ANGEL X. CHANG, Princeton University
DANIEL RITCHIE, Brown University

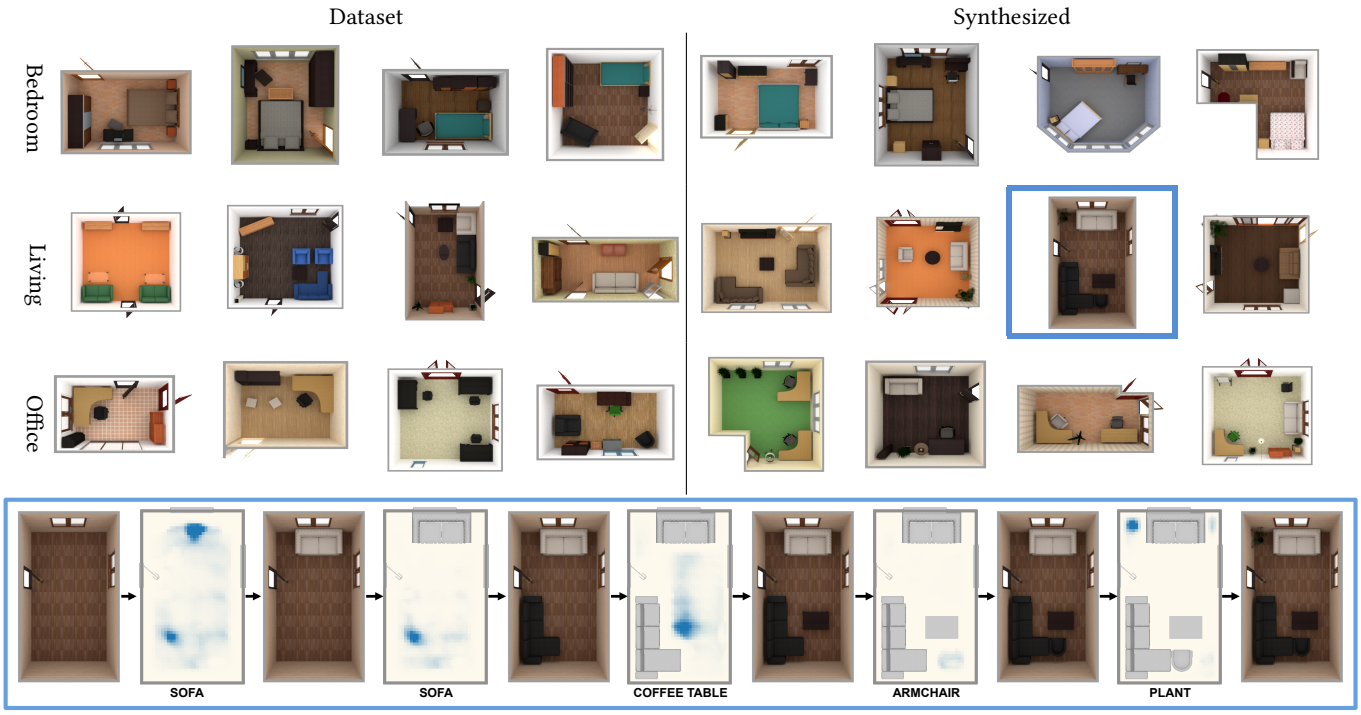

Fig. 1. We present a method to synthesize indoor scenes using deep convolutional network priors trained on top-down views of scenes from a scene dataset. *Top:* Bedroom, living room, and office scenes from our dataset, along with scenes synthesized from our model. *Bottom:* Our model generates scenes by iteratively inserting one object at a time. Here we show one such object insertion sequence for the living room scene outlined in blue. Renders of intermediate scenes are interleaved with a visualization of our model's predicted spatial probability distribution for the type of object about to be inserted.

We present a convolutional neural network based approach for indoor scene synthesis. By representing 3D scenes with a semantically-enriched image-based representation based on orthographic top-down views, we learn convolutional object placement priors from the entire context of a room. Our approach iteratively generates rooms from scratch, given only the room architecture as input. Through a series of perceptual studies we compare the plausibility of scenes generated using our method against baselines for object selection and object arrangement, as well as scenes modeled by people. We find that our method generates scenes that are preferred over the baselines, and in some cases are equally preferred to human-created scenes.

Authors' addresses: Kai Wang, Brown University; Manolis Savva, Princeton University; Angel X. Chang, Princeton University; Daniel Ritchie, Brown University.

CCS Concepts: • **Computing methodologies** → **Computer graphics**; **Neural networks**; **Probabilistic reasoning**;

Additional Key Words and Phrases: scene synthesis, object layout, neural networks, convolutional networks, deep learning

ACM Reference Format:
Kai Wang, Manolis Savva, Angel X. Chang, and Daniel Ritchie. 2018. Deep Convolutional Priors for Indoor Scene Synthesis. *ACM Trans. Graph.* 37, 4, Article 70 (August 2018), 14 pages. https://doi.org/10.1145/3197517.3201362

1 INTRODUCTION

People spend a large percentage of their lives indoors: in offices, living rooms, bedrooms, and other interior spaces. A number of industries use virtual reproductions of such indoor scenes: interior design, architecture, gaming and virtual reality are a few. A computer model that understood the structure of such scenes well enough to generate new ones could support such industries by enabling fully or semi-automatic population of indoor environments.

Prior work has studied this problem of indoor *scene synthesis*: given the architectural geometry of a room (i.e. its walls, floor, and ceiling), select a set of objects to place and arrange within the room. Various approaches have been proposed, including constraint satisfaction, optimization of hand-crafted interior design principles, statistical priors on pairwise relationships between objects, and human-centric relationship priors [Fisher et al. 2012, 2015; Fu et al. 2017; Merrell et al. 2011; Xu et al. 2002; Yu et al. 2011]. However, unconstrained synthesis at room scale is still a challenging problem. Prior work has either focused on modeling smaller, functional regions within a larger room (i.e. a working desk) [Fisher et al. 2012], or introduced additional inputs to constrain the problem: a fixed set of objects and manually-annotated important relationships [Yu et al. 2011], a sketch [Xu et al. 2013], a natural language description [Chang et al. 2014], or a 3D scan of a room [Chen et al. 2014; Fisher et al. 2015].

Convolutional neural networks (CNNs) hold some promise in addressing the scene synthesis problem, as they have been demonstrated to reliably learn to recognize and generate visual patterns and relationships in other graphics domains. Deep networks require a large amount of training data, which would have prohibited their use with the small scene datasets considered by early work on scene synthesis. Recently, however, large datasets of 3D scenes have become available [Song et al. 2017]. In this paper, we use such data to train a CNN-based model to select and place objects in a room given only the type of room desired and its wall structure as input.

To apply convolutional networks to the scene synthesis problem, we leverage the insight that while an indoor room is a 3D space, most objects that characterize a room are arranged on its 2D ground plane. As such, we represent a room via an orthographic top-down view, a representation on which a convolutional network can operate—just as architects are trained to think of rooms as patterns of objects on a floor plan, so too do we train our neural networks. Our top-down view representation is a semantically-enriched, multi-channel image capturing multiple scene features per pixel, such as the type of object present and its orientation. Applying layers of learnable convolutions to this representation allows our model to process the entire state of a room as context for its sampling decisions, capturing patterns and relationships between objects . Our proposed model generates a room by iteratively adding one object at a time, first deciding whether to add another object before deciding which type of object to place and where. Each of these decisions is governed by a convolutional neural network.

Our contributions are:

(1) We present (to our knowledge) the first convolutional network-based system for synthesizing indoor scenes from scratch: our system takes as input only the geometry of the room in which it should synthesize a scene.
(2) We introduce a new semantically-enriched, image-based representation of scenes based on orthographic top-down views.
(3) Through a series of ablations, we analyze our learned convolutional priors and show they capture common-sense patterns of object arrangement given the state of a room.
(4) We perform forced-choice perceptual studies to compare the plausibility of rooms generated using our method against baselines for object selection and arrangement inspired by prior work, as well as against rooms modeled by people. Scenes generated by our method are preferred over the baselines, and in some cases are equally preferred to human-created scenes.

After reviewing related work in Section 2, we give a high-level overview of our approach in Section 3. We then introduce our dataset and our image-based scene representation in Section 4 before describing the details of our generative model in Section 5. Finally, Section 6 presents the results of our perceptual studies. Our code, data, and pre-trained models are available at https://kwang-ether.github.io/deep-synth/

2 RELATED WORK

Indoor Scene Synthesis and Furniture Layout. There is a large body of existing work on synthesizing virtual interior scenes. Some of the early work in this space is concerned with creating plausible arrangements of a pre-specified set of objects according to interior design principles [Merrell et al. 2011] and simple statistical relationships between objects learned from examples [Yu et al. 2011]. Later work showed how to arrange objects in scenes where the set of objects to be used can change, using a hand-written program to specify priors over which objects may occur and their spatial relationships [Yeh et al. 2012]. Follow-on work from this has focused on *data-driven scene synthesis*: rather than using a hand-written program, learning priors over object occurrence and arrangement from examples. The first such method learned separate priors for occurrence and arrangement, using a directed graphical model of object co-occurrence relationships and a Gaussian mixture model of pairwise spatial relationships between objects [Fisher et al. 2012]. Various related methods have been proposed, modeling object occurrence and/or arrangement with undirected factor graphs [Kermani et al. 2016], topic models [Liang et al. 2017], directed graphical models combined with Gaussian mixture arrangement patterns [Henderson and Ferrari 2017], human-centric stochastic grammars [Qi et al. 2018], and activity-based object relation graphs [Fu et al. 2017]. In contrast, our method uses deep convolutional networks to learn priors over which objects should be in a scene and how they should be arranged. These deep network priors use a view of the entire scene as context, rather than attempting to build context from pairwise object relationships. They also couple the occurrence of objects with their arrangement, as the model decides which objects to add to the scene next based on the complete arrangement of objects that have been added thus far.

Other related work has focused on generating virtual indoor scenes given some sparse or low-fidelity input representation. Methods have been proposed for generating scenes from sketches [Xu et al. 2013], from noisy 3D scans [Fisher et al. 2015], or given text descriptions [Chang et al. 2014], all of which extend scene synthesis methods from the list above [Fisher et al. 2012]. We believe that our models could be used as the underlying synthesis models for these types of tasks, as well.

Deep Generative Models. Generative models based on deep neural networks have become popular in recent years. Deep latent

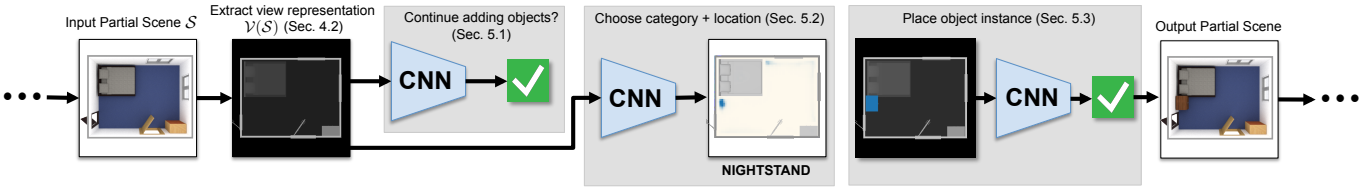

Fig. 2. Overview of our scene synthesis pipeline. Our generative model synthesizes scenes one object at a time. Given an input scene $\mathcal{S}$, it first computes a top-down view of the scene $\mathcal{V}(\mathcal{S})$ with multiple features per pixel (Section 4.1). Next, it analyzes this image to determine whether to add another object (Section 5.1). Then, it chooses a location at which to add the next object, and what category of object to add (Section 5.2). Given a location and category, it selects a instance of that category from a model database and adds it to the scene at an appropriate orientation (Section 5.3). These model components are implemented using deep convolutional networks.

variable models, such as variational autoencoders (VAEs) and generative adversarial networks (GANs), are perhaps the most well-known [Goodfellow et al. 2014; Kingma and Welling 2014]. These models can jointly generate an entire image at once from a latent random code; conditional versions can take an image as input [Isola et al. 2017]. While one could cast scene synthesis as an image-to-image problem (i.e. map an image of an empty room to an image of a room with objects), conditional GANs struggle to capture multiple modes of output variation. Recent work helps with generating local appearance variation [Zhu et al. 2017], but generating geometric variety (as our setting requires) remains an open problem. Instead, we synthesize indoor scenes one object at a time. This better matches the iterative object placement process for constructing real-world scenes and allows using the same model in multiple applications: auto-completing partial rooms, suggesting likely next objects to place, or suggesting likely objects at a given location.

Many similar methods for sequentially generating 2D or 3D visual content have also been proposed recently. Several are based on recurrent neural networks: DRAW [Gregor et al. 2015] and AIR [Eslami et al. 2016] are both recurrent variational models that generate and reconstruct images, and sketch-rnn uses a recurrent model to generate vector drawings as a sequence of strokes [Ha and Eck 2017]. There are also recurrent models of 3D shapes, decomposing shapes into either a sequence of primitive boxes [Zou et al. 2017] or constructive solid geometry operations [Sharma et al. 2017].

Rather than use a recurrent architecture, our model autoregressively predicts its next decision based on the cumulative result of all decisions it has made thus far (in the form of the current partial scene). This "predict distribution, sample, repeat" approach is similar in spirit to the well-known PixelCNN [van den Oord et al. 2016] and WaveNet [Van Den Oord et al. 2016]. More closely-related works also adopt this approach, including methods for controlling the output of procedural models [Ritchie et al. 2016], inferring drawing programs from hand-drawn sketches [Ellis et al. 2017], or suggesting components to add to a partially-complete 3D shape [Sung et al. 2017].

View-based CNNs for 3D Content. Our model uses deep convolutional networks which operate on a view of a 3D scene. Many other recent works use view-based convolutional networks to analyze 3D objects, to perform object classification [Su et al. 2015], perform semantic segmentation [Kalogerakis et al. 2017], learn local shape descriptors from correspondences [Huang et al. 2017], or learn stylistic similarity between objects [Lim et al. 2016].

These methods use multiple views of a single 3D object, whereas as our method uses a top-down view of a larger 3D scene. In that respect, it bears resemblance to the high-level character controller from the DeepLoco system, which uses a convolutional network on a top-down depth map view of the surrounding environment to predict a footstep plan for a bipedal character [Peng et al. 2017].

3 APPROACH

This paper focuses on learning generative models of indoor scenes, specifically, rooms that contain one or more objects. The input to our method is a room, i.e. geometry describing floor, walls, and ceiling. In this work, we focus on generating and arranging major functional objects such as furniture which are placed on the floor. We do not address wall-mounted or second-tier objects (i.e. objects placed on top of other objects), though we discuss straightforward extensions to support these features in Section 7.

Our goal is to build a generative model of scenes that can encode complex dependencies and relationships involving objects and the room geometry. We use deep networks as the major building block of our model, as they have been shown to reliably recognize and generate complex visual patterns in other visual domains. To do this, we exploit the fact that while indoor scenes exist in 3D space, gravity dictates that most objects are arranged on the 2D floor plane. Thus, we can apply 2D convolutional networks to a top-down view representation $\mathcal{V}(\mathcal{S})$ of a scene $\mathcal{S}$, where the networks learn to recognize the presence of and relationships between objects.

Since scenes are composed of a set of discrete objects, we use a model architecture that generates a scene iteratively, adding objects one-by-one. This sequential paradigm has been frequently used to model other visual content that decomposes into discrete components [Ellis et al. 2017; Ha and Eck 2017; Ritchie et al. 2016; Sharma et al. 2017; Sung et al. 2017; Zou et al. 2017]. Figure 2 shows a schematic overview of one iteration our method, somewhere in the middle of a synthesis sequence. Our model first takes the scene constructed thus far and renders a top-down view of it. This top-down view is augmented with multiple semantic features, such as the category of the object at each pixel (Section 4.1). This view $\mathcal{V}(\mathcal{S})$ of a scene $\mathcal{S}$ is an image-based representation of the scene that can

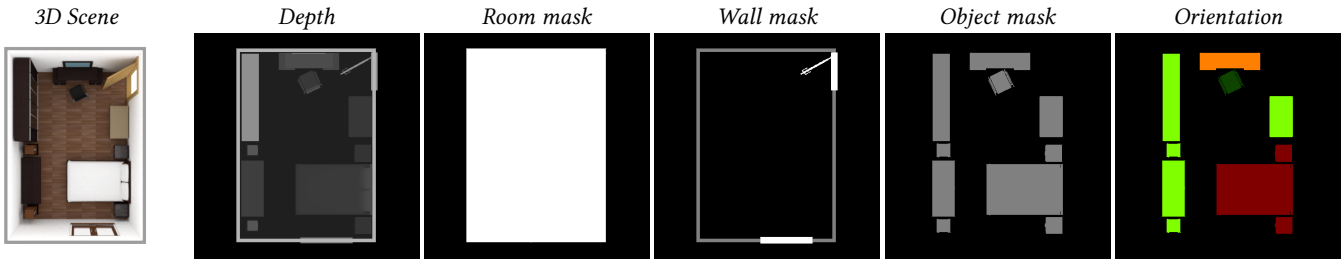

Fig. 3. Image channels included in our top-down view representation $\mathcal{V}(\mathcal{S})$ of a scene $\mathcal{S}$. Object category channels not shown, for brevity. Orientation is visualized by normalizing $\cos\theta$ and $\sin\theta$ to $[0, 1]$ and mapping them to the red and green channels of an RGB image.

be analyzed by convolutional networks (CNNs); it is used as input by several CNN-based components of our model.

The first of these components predicts whether the model should keep adding objects to the scene (Section 5.1). If this component returns true, the next model component chooses a category of object to add and a location at which to add it. This component works by building a probability distribution of possible object categories across many possible scene locations, then sampling from that distribution (Section 5.2). Given an object category and location, the model must then instantiate a particular object that belongs to that category at that location, orienting the object appropriately in the process (Section 5.3). These three model components all use deep convolutional networks which take the scene view as input. These networks are trained on scenes sampled from a large database of 3D scenes; Section 4 describes our method of constructing training data in detail. Figure 1 Bottom shows a scene synthesized by executing multiple iterations of this process.

4 DATASET

Learning a deep network-based model requires a large amount of training data. To train our model, we use the SUNCG dataset [Song et al. 2017], a large database of virtual 3D scenes created by users of the online Planner5d interior design tool [Planner5d 2017]. The dataset contains over 45,000 3D scenes, with each scene segmented into individual rooms labeled with a room type (e.g. bedroom, living room) or multiple room types. In this work, we consider three different types of rooms that occur frequently both in residential interior environments and in the SUNCG dataset: bedrooms, living rooms, and offices. As the SUNCG dataset is large and noisy, we first do some preprocessing by filtering out certain rooms and certain types of objects from rooms. Appendix A provides more details on these filters. After filtering, we are left with 8,398 bedrooms, 1,238 offices, and 1,452 living rooms. For each room type, we hold out 250 rooms for validation and testing and use the rest for training.

4.1 Top-down View Representation

As mentioned previously, indoor rooms are largely characterized by the 2D layout of objects on the floor; much prior work on scene synthesis phrases the problem in terms of determining 2D positions and 1D orientations of objects on a supporting surface [Fisher et al. 2012; Merrell et al. 2011; Yu et al. 2011]. Likewise, we convert 3D rooms into a 2D layout representation that our model operates on using deep convolutional networks. Specifically, we represent the scene $\mathcal{S}$ as a multi-channel top-down view $\mathcal{V}(\mathcal{S})$. The base of this representation is an orthographic top-down depth render of the room, i.e. a heightmap. This render maps a $6\,\text{m} \times 6\,\text{m}$ region into a 512×512 image. Discrete-sampled heightmaps have been used for scene synthesis before, as a coarse representation of the geometry of a scanned 3D scene that synthesis should match [Fisher et al. 2015].

With this height map alone, the deep networks in our model would be forced to use some of their representational capacity to learn how to detect semantic features in the height map, such as the boundaries, orientations and categories of objects (and this may or may not succeed). Instead, we encode such semantic features as additional channels in the top-down view image. The final image has the following information at each pixel, which defaults to 0:

Depth: Depth from the camera.
Room mask: Takes a value of 1 when inside the room.
Wall mask: Takes a value of 0.5 for walls and 1 for doors/windows.
Object mask: Indicates the number of objects present. Each object adds 0.5 to the pixel value.
Orientation: The orientation of the object contained by the pixel, represented by an angle θ about the world-up vector. θ is expressed in a local coordinate frame that is consistent across all objects. We encode this information into the image using two channels, one for $\sin\theta$ and one for $\cos\theta$ (i.e. a polar-to-rectangular transform).
Category: The category of the object(s) contained by the pixel. For each category, including doors and windows, we add a channel that stores the number of objects of that category.

Figure 3 shows what some of these channels look like for a typical room. This representation has $C + 6$ channels total, where C is the number of possible categories for the type of room. In our dataset, C ranges from 21 (living room, office) to 31 (bedroom). Though we do not experience hardware or training problems with this representation, for significantly larger C, it may become necessary to instead learn a fixed-dimensional representation for object categories, serving a similar function as word embeddings in natural language data [Mikolov et al. 2013]. We leave such an effort for future work.

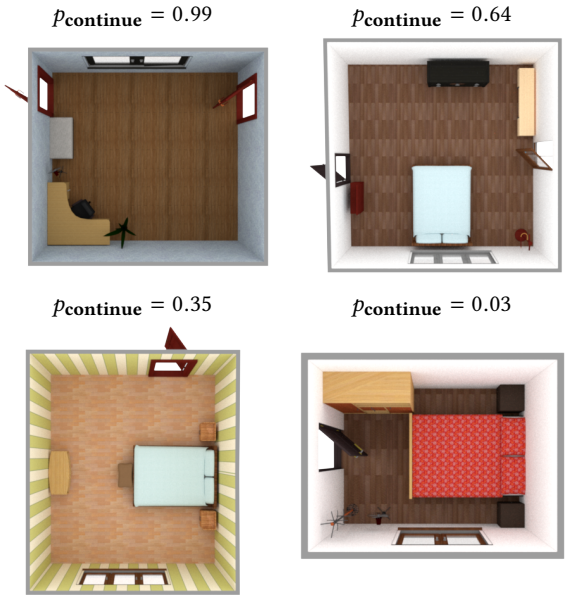

Fig. 4. Predicted p_{continue} values for partial bedroom scenes from our dataset. Clockwise from top left: the absence of a bed leads to continue probability near 1.0, a scene with a bed has lower continue probability but still above 0.5, a scene that is complete enough to terminate but could still have more objects added, a scene that is completely full and thus has nearly 0 continue probability.

5 GENERATIVE MODEL

Our model generates a scene $\mathcal{S}$ by iteratively placing one object at a time, where the existence and configuration of the object being placed is conditioned on the state of the scene thus far. Each iteration of our model decomposes into three steps: deciding whether to add another object (`Continue?`), deciding what category of object to add and where (`CategoryLocation`), and inserting an instance of that object category into the scene (`InstanceOrientation`).

5.1 When to stop adding objects

The first component of our model, `Continue?`, decides whether the model should add another object to $\mathcal{S}$ or should terminate. It is a function which takes as input the current scene $\mathcal{S}$ and outputs a Bernoulli distribution $p_{\text{continue}}(\top|\mathcal{S})$, i.e. the probability that 'continue adding objects' is true (i.e. logical $\top$) given the current scene $\mathcal{S}$. We encapsulate the current state of the scene by two sets of features. First, we use a vector of counts of each category of object already present in the scene, counts($\mathcal{S}$). This provides *global* information about the state of the scene that may be important for making continue/terminate decisions, e.g. a bedroom with zero beds must continue adding objects. Second, we extract high-level features from the top-down view representation of the scene $\mathcal{V}(\mathcal{S})$ using a deep convolutional network. These features provide information about whether the current set of scene objects represents a complete scene *for the given room*, e.g. while a single bed and nightstand may be sufficient for a tiny, narrow room, a large bedroom requires more objects. We then apply a multilayer feed-forward neural network (MLP) to compute the probability of continuing:

$$p_{\text{continue}}(\top|\mathcal{S}) = \sigma(\text{MLP}([\text{counts}(\mathcal{S}), \text{CNN}(\mathcal{V}(\mathcal{S}))]))$$

where $[\cdot]$ is vector concatenation and σ is the logistic sigmoid.

We build a training set for `Continue?` with 50% negative examples (complete rooms from our dataset) and 50% positive examples (rooms with a random number of random objects removed). Figure 4 shows some partial bedroom scenes from our dataset with their predicted p_{continue} values. In the upper left scene, the room has no bed and thus p_{continue} is near 1. The upper right scene has a bed but is otherwise somewhat sparse; p_{continue} is lower but still above 0.5. The lower left scene has enough appropriate objects to be reasonably considered a 'complete' bedroom, and thus has $p_{\text{continue}} < 0.5$, but it also has space for more objects. Finally, the lower right scene is completely packed with objects, leading to a p_{continue} near 0.

We train `Continue?` using the standard binary cross entropy loss. At synthesis time, we terminate synthesis when p_{continue} falls below 0.5 (i.e. the classification decision boundary).

If `Continue?` decides to add another object, our model must then decide both which category of object to add and also where in the room to add it. These two decisions are strongly correlated: some locations only make sense for certain types of objects, and vice versa. Ideally, we would like our model to learn the joint distribution $p(c, x, y|\mathcal{S})$ over all categories and all possible locations in the scene. To make this problem amenable to the use of convolutional networks, we instead learn a conditional distribution. Specifically, our next model component `CategoryLocation` computes $p_{\text{cat}}(c|\mathcal{S}, x, y)$, the probability that c is the category which should be added to scene $\mathcal{S}$ at location (x, y). This structure is similar to the 'action map' representation used in prior work to encode probabilities of different human activities taking place across different parts of a scene [Savva et al. 2014]. Later, we describe how we construct the joint distribution from this conditional and then sample from it.

5.2 What category of object to add next (and where)

To compute these probabilities, `CategoryLocation` uses a similar structure as `Continue?`: a convolutional network to extract spatial features from the current state of the room, along with the room's current category counts. Since `CategoryLocation` computes probabilities for a particular location (x, y), it must incorporate additional information to instruct the CNN to attend to that location. For this purpose, we add an additional *attention mask* channel to the top-down view image $\mathcal{V}(\mathcal{S})$. This is an image $\mathcal{M}_{\text{attn}}(x, y)$ containing a small (9×9) mask centered about (x, y); Figure 5 shows some examples. Category probabilities are then computed as

$$f_{\text{cat}}(c|\mathcal{S}, x, y) = \text{MLP}([\text{counts}(\mathcal{S}), \text{CNN}([\mathcal{V}(\mathcal{S}), \mathcal{M}_{\text{attn}}(x, y)])])$$

$$p_{\text{cat}}(c|\mathcal{S}, x, y) = \frac{\exp(f_{\text{cat}}(c|\mathcal{S}, x, y))}{\sum_{c'} \exp(f_{\text{cat}}(c'|\mathcal{S}, x, y))}$$

i.e. we compute $f_{\text{cat}}(c|\mathcal{S}, x, y)$ and then normalize it using a softmax activation. Figure 6 Top visualizes a few representative examples of these probabilities evaluated across all locations in a scene.

$\mathbf{p}_{\text{cat}}(c|\mathcal{S}, x, y)$ gives a probability distribution over object categories with centroid at location (x, y). As it is a distribution, it sums to one, which would imply that there is *some* object centroid at every

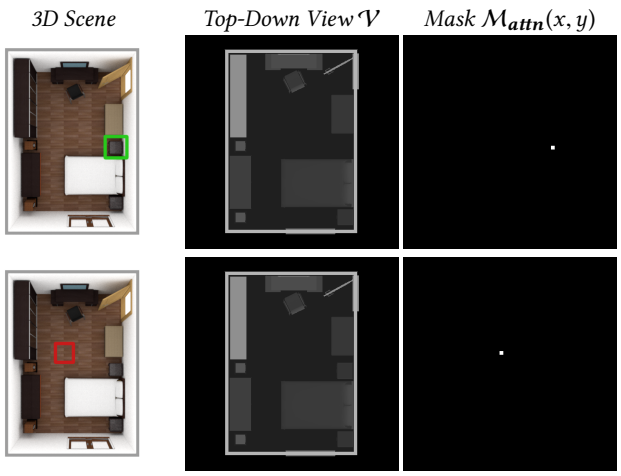

Fig. 5. Illustrating the training examples (x, y, c) used to train `CategoryLocation`. The colored boxes are centered around the location (x, y). *Top:* an example where c = 'nightstand'. The third column shows the attention mask $\mathcal{M}_{\mathbf{attn}}(x, y)$. *Bottom:* an example where c = 'no object'.

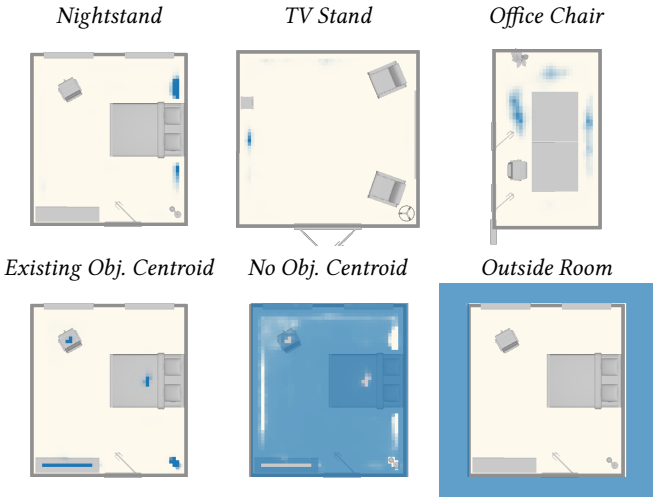

Fig. 6. Examples of predicted category probabilities for all locations in a scene. *Top:* predicted probabilities for several object categories in different scenes. *Bottom:* predicted probabilities for auxiliary categories for the same bedroom scene.

location. This is obviously implausible: a large fraction of locations in a finished room remain unoccupied by objects, to permit human movement. Thus, we augment the set of object categories with an *auxiliary category* for 'no object centroid.' We also add auxiliary categories for 'existing object centroid' (i.e. a location in the scene already occupied by an object centroid) and for 'outside room,' to allow `CategoryLocation` to learn how such locations differ from those in which object centroids may be placed. Figure 6 Bottom shows probabilities for these categories across an example scene.

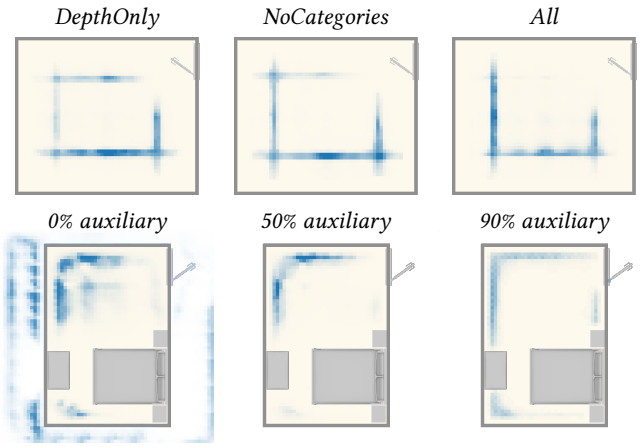

Fig. 7. Example object category location distributions predicted by the `CategoryLocation` network with certain features omitted. *Top:* Predicted distributions for the centroid of a bed with scene view features excluded. Only when all features are present does the distribution avoid parts of the scene that would block the door in the upper-right. *Bottom:* Predicted distributions for a wardrobe, varying the percentage of auxiliary category examples in the training data. A high percentage is needed to learn a distribution that is flush against the walls.

Training. To generate training examples for `CategoryLocation`, we take rooms from our dataset and randomly remove objects from them. To generate examples for object categories, we choose a random removed object and generate an attention mask $\mathcal{M}_{\mathbf{attn}}(x, y)$ about that object's centroid (x, y). To generate examples for auxiliary categories, we choose a random point that is outside the room, in empty space within the room, or within an existing object, and we generate an attention mask about that point. Figure 5 illustrates two examples of this process. We generate training sets with 95% auxiliary category examples and 5% object category examples. This balance makes intuitive sense, given the sparsity of object centroid locations in a scene. The low proportion of object category examples can cause the network to train slowly; to speed this up, we start training with only object category examples and gradually increase the percentage of auxiliary category examples to 95%.

Given the large number of possible conditioning inputs $(\mathcal{S}, x, y)$, the network may not see enough cases to learn rarer rules, e.g. not to place a third bed in a bedroom that already has two. To help with this situation, we train `CategoryLocation` with the standard categorical cross entropy loss along with an additional loss $\mathcal{L}_{\mathbf{global}}$. This loss provides additional global context by penalizing the model for assigning probability to categories not in the set $C_{\mathbf{removed}}$ of object categories removed from the current partial training scene. We enforce this loss more strictly for nearly-complete partial training scenes, linearly scaling it based on the ratio of the number of objects in the partial scene to that of the complete scene:

$$\mathcal{L}_{\mathbf{global}} = \frac{N_{\mathbf{partial}}}{N_{\mathbf{complete}}} \sum p(c) \; \forall c \notin C_{\mathbf{removed}}$$

Ablations. We also trained versions of `CategoryLocation` with features omitted from the scene view images $\mathcal{V}(\mathcal{S})$. In addition to

the full set of image features (*All*), we considered scenarios with the category mask channels removed (*NoCategories*) and with all channels except depth removed (*DepthOnly*). We also experimented with the percentage of auxiliary category examples (*NoObject*, *ExistingObject*, *OutsideRoom*) in the dataset used to train the `CategoryLocation` network. Figure 7 shows typical predicted distributions for each setting. In the top row, we show how all the features are needed to prevent the predicted location distribution for beds to place probability mass in locations that would block the door (upper right of the room). In the bottom row, we show how increasing the percentage of auxiliary category examples in the training data removes spurious probability mass from the distribution for wardrobe locations, leading to a distribution that plausibly concentrates its mass along available walls.

Sampling. To sample a new category and location at synthesis time, we need to construct a joint distribution $p(c, x, y|\mathcal{S})$ using the values of $p_{\mathbf{cat}}(c|\mathcal{S}, x, y)$. To make this problem tractable, we consider candidate locations on a $N \times N$ regular grid defined over the scene. We then need to define a prior distribution $p(x, y|\mathcal{S})$ over grid locations. Since we have already taken into account the probability of objects being present/absent at a particular location through the use of auxiliary categories, we can use a uniform prior $p(x, y|\mathcal{S}) = \frac{1}{NN}$. Our joint distribution is then, by the chain rule:

$$p(c, x, y|\mathcal{S}) = p_{\mathbf{cat}}(c|\mathcal{S}, x, y) \cdot \frac{1}{NN}$$

For brevity, we drop the conditioning on the current scene $\mathcal{S}$ for the remainder of this section.

$p(c, x, y)$ is a three dimensional discrete distribution of size $C \times N \times N$, where C is the number of categories. When sampling from large generative models such as this one trained on a high volume of noisy data, it is common practice to suppress noise in the distribution by adjusting the model's *temperature*, i.e. taking $p^{1/\tau}$ for some temperature value τ (see e.g. [Andrej Karpathy 2015; Ha and Eck 2017]). We also temper the joint category-location distribution $p(c, x, y)$, though we have found a two-step tempering scheme to perform better. We first temper the spatial probability distributions for each category c:

$$p(c) = \sum_{x=1}^{N} \sum_{y=1}^{N} p(c, x, y)$$

$$p^{1/\tau_1}(x, y|c) = \frac{p(c, x, y)^{1/\tau_1}}{\sum_{x=1}^{N} \sum_{y=1}^{N} p(c, x, y)^{1/\tau_1}}$$

$$f_1(c, x, y) = p^{1/\tau_1}(x, y|c) \cdot p(c)$$

Here, $f_1(c, x, y)$ is a density with the same shape as $p(c, x, y)$, but with noise suppressed and high-probability modes highlighted for each category c. We then temper the overall spatial density $f_1(x, y)$:

$$f_1(x, y) = \sum_{c=1}^{C} f_1(c, x, y)$$

$$f_1(c|x, y) = f_1(c, x, y)/f_1(x, y)$$

$$f_2(c, x, y) = f_1(c|x, y) \cdot f_1(x, y)^{1/\tau_2}$$

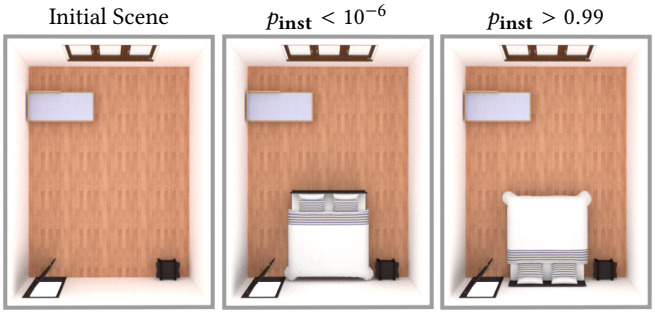

Fig. 8. Using the `InstanceOrientation` network to predict the validity of inserting model instances at different orientations. Inserting the double bed at the correct orientation with respect to the wall has probability near 1, whereas the opposite orientation is clearly recognized as being incorrect with probability near 0.

After this step, $f_2(c, x, y)$ has suppressed overall low-probability locations, but without changing the relative probability of categories at any location. We then normalize f_2 to produce the final distribution p^* from which our model samples:

$$p^*(c, x, y) = \frac{f_2(c, x, y)}{\sum_{c=1}^{C} \sum_{x=1}^{N} \sum_{y=1}^{N} f_2(c, x, y)}$$

We set $\tau_1 = 0.25$ and $\tau_2 = 0.4$ for all the results shown in this paper.

At synthesis time, we use a 32×32 grid (i.e. $N = 32$) to sample a location (x, y) and a category c. To obtain a more fine-tuned location, we then use a 16×16 grid within the previously-sampled grid cell, constrain the category to c, and choose the location which maximizes $p^*(x, y|c)$).

5.3 Placing an object instance

Given a location (x, y) and an object category c, our model's final step is to insert an instance of that category at that location. Here, 'instance' refers to a specific 3D model. In our work, we draw these 3D models from the set of models used in the SUNCG dataset.

There can be many possible instances for a given category (for example, there are nearly 300 chair models in SUNCG), not all of which are visually compatible with one another. Thus, we first restrict the set of instances considered to those likely to be compatible with existing objects in the room. SUNCG, having originated from an interior design tool, contains many 3D models drawn from stylistically-consistent *collections* of furniture. We annotated all SUNCG 3D models with their collection. At synthesis time, we allow the insertion of instances that come from the same collection as another object already in the room. We relax this constraint slightly by also allowing instances that co-occur in the same SUNCG scene with another object already in the room. Prior work has developed more sophisticated approaches for measuring the style compatibility of 3D furniture models in terms of their geometry [Liu et al. 2015] or materials [Chen et al. 2015]; such methods could also be applied here.

To insert an instance into the room, our model must also choose an orientation for that instance, where orientation is defined as a single angle θ about the gravity vector. To do this, we use a third and

final convolutional network component, `InstanceOrientation`. This component inserts a candidate instance i into the scene at a candidate orientation θ, and then evaluates a network which returns $p_{\mathbf{inst}}(\top|\mathcal{S}, i, x, y, \theta)$, the probability that inserting instance i at (x, y, θ) is a valid addition to the scene $\mathcal{S}$. `InstanceOrientation` augments the top-down scene view $\mathcal{V}(\mathcal{S})$ with a mask for the geometry of the inserted instance, $\mathcal{M}_{\mathbf{geo}}(i, x, y, \theta)$. It computes the probability of the insertion as

$$p_{\mathbf{inst}}(\top|\mathcal{S}, i, x, y, \theta) = \sigma(\text{MLP}(\text{CNN}([\mathcal{V}(\mathcal{S}), \mathcal{M}_{\mathbf{geo}}(i, x, y, \theta)]))$$

Figure 8 shows an example of $p_{\mathbf{inst}}$ discriminating between a valid and an invalid insertion.

We train `InstanceOrientation` by taking rooms from our dataset, again removing a random set of objects from each room, and then selecting one removed object to be re-inserted. We generate 50% positive training examples by inserting the object at its original orientation, and 50% negative training examples by inserting the object at a different orientation. For these negative example orientations, we quantize $[0, 2\pi]$ into 16 discrete orientations with the object's original orientation at 0, and we select from among the 15 other orientations. Training uses the standard binary cross entropy loss.

At synthesis time, our model tries to insert all of the allowed instances at each of 16 discrete orientations and chooses the collision-free insertion with highest probability. If there is no collision free insertion or no insertion with probability higher than 50%, our model resamples a different (x, y, c) tuple from `CategoryLocation`. We automatically reject any synthesized room that uses more than 20 such resampling steps. In our experiments, this was $\sim$ 20% of synthesized bedrooms, $\sim$ 15% of offices, and $<$ 1% of living rooms. Rooms that are not rejected use on average less than three resampling steps.

5.4 Details and timings

We implement our neural networks in PyTorch [Paszke et al. 2017]. All the convolutional networks use the same architecture, the Resnet-101 architecture [He et al. 2016] (modified to use 512×512 images as input). The set of features $f_{\mathbf{cnn}}$ output by these networks has size 2048. The MLP networks contain three fully-connected layers with input sizes $2048 + C$, 2048, and 1024, respectively. We use ReLU activation and batch normalization between these layers.

We train our networks on NVIDIA GeForce GTX 1080 Ti GPUs, using the Adam optimizer [Kingma and Ba 2015]. Training takes 6 hours for `Continue?` and 2 days for `CategoryLocation` and `InstanceOrientation`. `Continue?` and `InstanceOrientation` can both be interpreted as binary classifiers, so we can evaluate their classification accuracy: `Continue?` reaches $\sim$ 85% validation accuracy on average across all room types, and `InstanceOrientation` reaches $\sim$ 94%.

At synthesis time, evaluating `CategoryLocation` for a 32×32 grid of scene locations takes 25 seconds, and placing an object instance takes on average 20 seconds. This lead to an average overall room synthesis time of 4 minutes. In Section 7, we discuss possible approaches to reduce this computation time.

6 RESULTS AND EVALUATION

In this section, we evaluate our model's ability to generate plausible rooms with characteristics similar to the SUNCG training scenes. To do this, we compare scenes synthesized by our model to several baselines using perceptual studies on Amazon Mechanical Turk. All of these comparisons use the same study design, which extends that of prior work on evaluating automatic image colorization methods [Zhang et al. 2016]. Study participants were given a series of forced-choice image comparison tasks. Each task displayed a pair of images, one depicting a scene synthesized by our model and the other from a baseline source. Both scenes used the same room geometry. The order of images within each pair was randomized. The objects in each scene image were colored according to their category, so that participants would make judgments based on the types of objects present and their arrangement, and not based on extraneous factors such as object materials. The task instructed participants to choose the image from the pair which they think depicts a more plausible arrangement of objects in the room. Each participant performed 55 comparison tasks. One out of each 11 tasks was a 'vigilance test': a comparison with an obviously wrong answer (specifically, one image depicted a randomized, jumbled arrangement of random objects). For each study on each room type, we collected responses from 10 AMT Master workers (workers AMT identifies as consistently high-performing) and discarded all responses from workers who did not achieve 100% accuracy on the vigilance tests.

Comparison against independently selecting scene objects. Most prior scene synthesis methods first generate a set of objects for the scene, then arrange those objects. In contrast, our method adds each object based on the existence and arrangement of all objects added before it. In theory, our method can more reliably produce higher-quality scenes, since it is sensitive to the size and shape of the room and what objects can plausibly be used in it, as opposed to independently generating a set of objects that is plausible 'on average.' We conduct an experiment to determine whether this advantage is visible in practice.

As a baseline method for independently generating a set of object categories, we use a Neural Autoregressive Distribution Estimator (NADE) [Uria et al. 2016]. A NADE is a learnable model for probability distributions over a sequence of variables, where the distribution over variable i is a function of the values of variables 1 through $i-1$. We model distribution over the number of instances of each object category occurring in a scene, which can be represented as a sequence of categorical variables $(x_1, x_2, \ldots)$, with one variable for each category of object that may occur. If x_i is a variable with D_i possible count values (including zero), then its probability distribution is a categorical distribution computed as follows:

$$\begin{aligned} p(x_i|\mathbf{x}_{<i}) &= \text{softmax}(\mathbf{V}_i\mathbf{h}_i + \mathbf{b}_i) \\ \mathbf{h}_i &= \text{sigmoid}(\mathbf{a}_i) \\ \mathbf{a}_i &= x_{i-1}\mathbf{w}_i + \mathbf{a}_{i-1} \ , \text{ where } \mathbf{a}_1 = \mathbf{c} \end{aligned}$$

where $\mathbf{V}_i \in \mathbb{R}^{D_i \times H}$, $\mathbf{b}_i \in \mathbb{R}^{D_i}$, $\mathbf{w}_i \in \mathbb{R}^H$, $\mathbf{c} \in \mathbb{R}^H$ are the learnable parameters of the NADE, with H being the NADE's hidden layer size. We use this model as a stand-in for Bayesian networks, topic models, and other approaches that have been used to model the distribution of objects in a scene. We train the NADE with the same set of training scenes used to train our model.

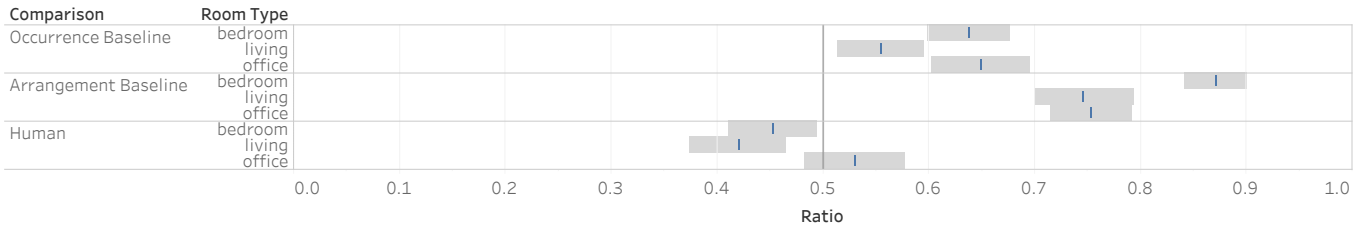

Fig. 9. Results of Mechanical Turk forced-choice perceptual studies comparing scenes synthesized by our method to those from several baselines. Blue lines show the percentage of time that our method was chosen, with gray bars showing 95% confidence intervals computed by bootstrap [Efron and Tibshirani 1986]. Over all room types, scenes generated by our method are significantly preferred to those generated by first sampling a set of objects and then arranging them using our model (*Occurrence Baseline*). When arranging a fixed set of objects, our model is also significantly preferred over a model based on pairwise object relationship statistics (*Arrangement Baseline*). Compared to human-created scenes from our test dataset, our generated scenes are equally preferred for office scenes, and slightly less preferred for bedroom and living room scenes.

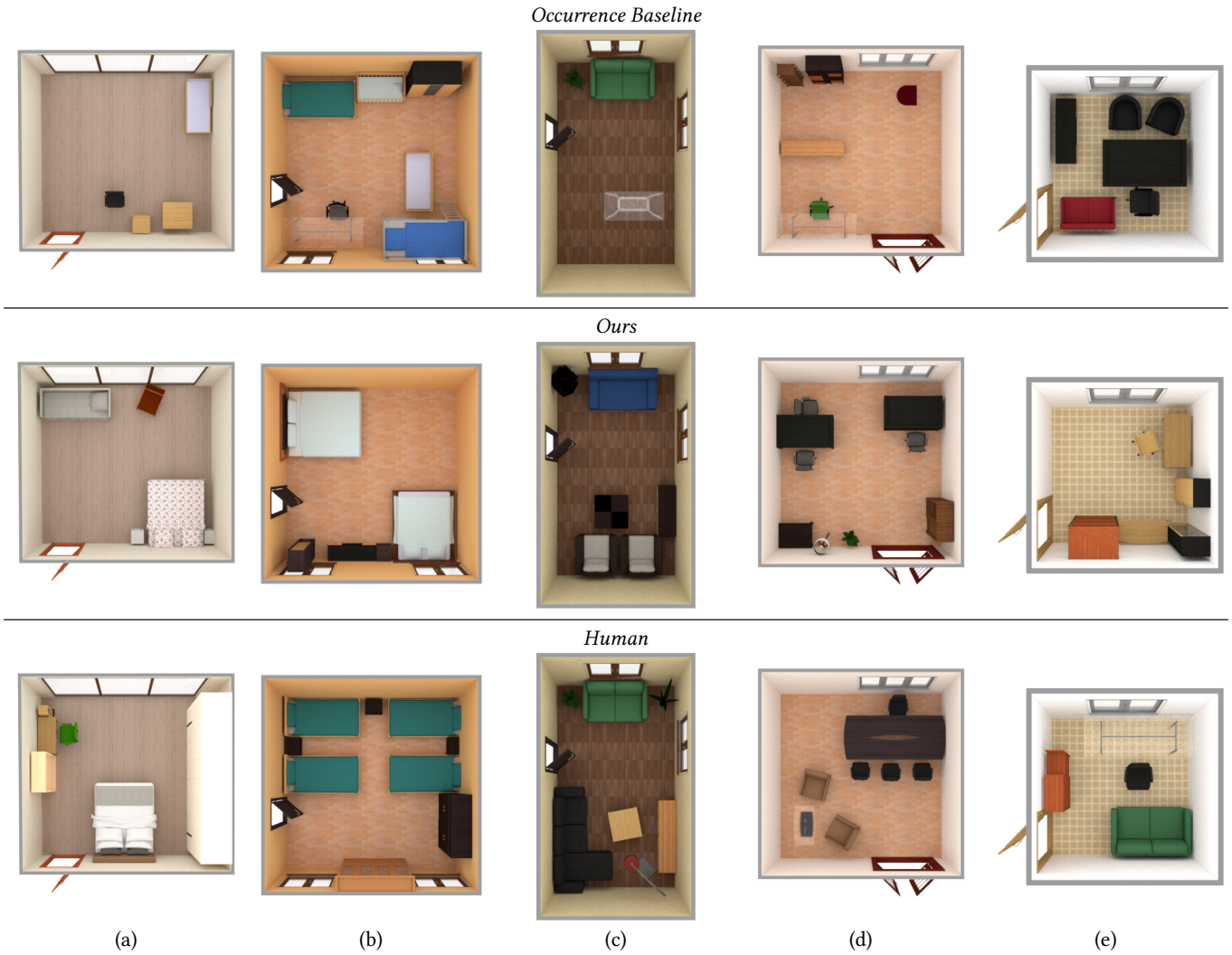

Fig. 10. Comparison of outputs using baseline occurrence model (top) and our full model (middle), with original human-designed scenes (bottom). Columns (a), (c) and (d) show examples where the baseline model results in too sparsely populated rooms. In column (b), the baseline selects different bed types for the four beds. In column (e) the baseline selects objects that are too large for the space. In contrast, our model is informed by the current arrangement and more plausibly populates the space of the room with consistent object types.

In our experiment, we synthesize scenes using the NADE-based object occurrence model as follows: We first use the NADE to sample occurrence counts for all possible object categories in the scene. We then select object instances for these categories and arrange them using our model—this isolates the specific object occurrence behavior we are interested in comparing. To do this, we rejection sample from our model until it chooses to add one of the categories for which the NADE sampled a nonzero count, and we repeat this process until all of the sampled category counts have been satisfied.

In a perceptual study, participants preferred scenes generated by our model over those generated by the baseline across all types of rooms (Figure 9, row *Occurrence Baseline*). Figure 10 shows some representative scenes generated by each method, along with a human-created scene from the dataset that uses the same room geometry. In many cases, the baseline occurrence model samples a set of objects for the room that is plausible, but too sparse for the particular room geometry (columns (a), (c), and (d)). The opposite phenomenon can also occur, with the baseline model sampling a set of objects that causes the room to be too crowded (column (e) Top, the sofa is forced into a position which blocks the door). Our model, by basing its decision of whether to add more objects and which to add on the current state of the scene, generally avoids these issues.

Comparison against arrangement using pairwise relationship priors. We also conducted an experiment to see whether our model suggests better arrangements of objects than models based on simpler, pairwise relationship statistics. In our experiment, we compare the ability of the two approaches to arrange the same, fixed set of objects—this isolates the specific object arrangement behavior we are interested in evaluating. For the fixed set of objects, we use the objects from a scene in our dataset which was not used for training. This also allows us to compare against the arrangement for these objects chosen by the original human author of the scene. This is a challenging task for any method like ours which attempts to greedily add objects one a time into the scene: many scenes in the dataset are tightly packed and precisely arranged, making it difficult to produce an alternative plausible arrangement. We are interested in whether our priors allow our model more frequently to find a reasonable re-arrangement of the scene objects.

As a baseline method for arranging objects, we use a set of learned pairwise priors based on prior work. In constructing this baseline, our goal was to select a set of commonly used elements in prior approaches for object layout optimization. We learn priors for the pairwise relative position and orientation of objects, and distance and angle to the closest wall for each object category. Variants of these priors are used by much prior work in 3D interior arrangement [Fisher et al. 2012; Kermani et al. 2016; Merrell et al. 2011; Yu et al. 2011]. Though priors on other properties such as symmetry [Kermani et al. 2016], conversation [Merrell et al. 2011], human activities [Fisher et al. 2015; Fu et al. 2017], and door-to-door navigability [Yu et al. 2011] have been used in optimizing object arrangement, our goal is to contrast our method against a representative purely data-driven approach using only learned pairwise object placement priors. We represent these pairwise priors as mixtures of Gaussian distributions encoding the offset between a pair of objects, and a discrete distribution encoding the angle in the 2D plane between the front orientations of the two objects (including walls). During synthesis, like our method, the baseline adds objects one at a time, in order of decreasing physical size. It evaluates the probability of many randomly-sampled candidate placements of the object using the pairwise priors of that object with respect to all existing objects in the scene, and it chooses the highest-probability position and orientation. Appendix B provides more details.

In the perceptual study, participants preferred re-arrangements generated by our method to those generated by the baseline (Figure 9, row *Arrangement Baseline*). Figure 11 shows representative scenes compared by participants in this study. While both methods struggle to re-arrange especially tightly-packed scenes, ours performs better in most cases. Columns (a) and (b) show two bedrooms that our model manages to re-arrange plausibly, while the baseline model resorts to implausible, non-navigable layouts to pack everything into the available space. Column (e) shows re-arrangement of a crowded office; our model places the sofa against a wall, whereas the baseline places it in an unusable configuration close to the desk. In column (c), the L-shape of the room makes it difficult to arrange the sofa, table, and chair around each other as is typical. Our method packs the table too tightly into the sofa's concavity, but this is preferable to putting the chair there instead, as the baseline does. Finally, column (d) illustrates a known failure mode of pairwise priors: conflict between multiple strong pairings. In this case, both chairs cluster around one of the two desks in the room. Our method tends to handle this case better, as this result and the other synthesized offices in Figure 1 illustrate. Some of the baseline's shortcomings could be remedied via explicit human-factors penalty terms, such as those mentioned in the previous paragraph. Such terms could also improve our method, and we find it interesting and encouraging that our model performs as well as it does given that it is purely data-driven.

Comparison against human-created scenes. We then compare scenes synthesized by our model with held-out test scenes from our dataset. Figure 1 shows examples of such scenes for each room type. In the perceptual study, participants showed no preference between the two types of scenes for offices, and they slightly preferred the human-created scenes for bedrooms and living rooms. While our model generates plausible scenes on average, it does exhibit a few failure modes, some typical instances of which are shown in Figure 14. In the top-left living room scene, the model has placed plausible objects in plausible locations, but it did not place any seating objects such as sofas or chairs. This is partly a product of some training set living rooms having this characteristic, and partly that our model is more sensitive to local scene plausibility than global plausibility. The top-right bedroom has too many nightstands next to bed, as beds are a strong cue for nightstands, and the presence of one does not always fully suppress the probability of adding another in the same or nearby location. In the bottom-left office scene, `CategoryLocation` sampled a reasonable centroid location for a wardrobe cabinet, but the model inserted by `InstanceOrientation` is very long and partially blocks the door. Finally, the lower-right bedroom scene contains two beds that are plausibly arranged when considered independently but whose geometry does not leave room between them to traverse the room. Improving global consistency

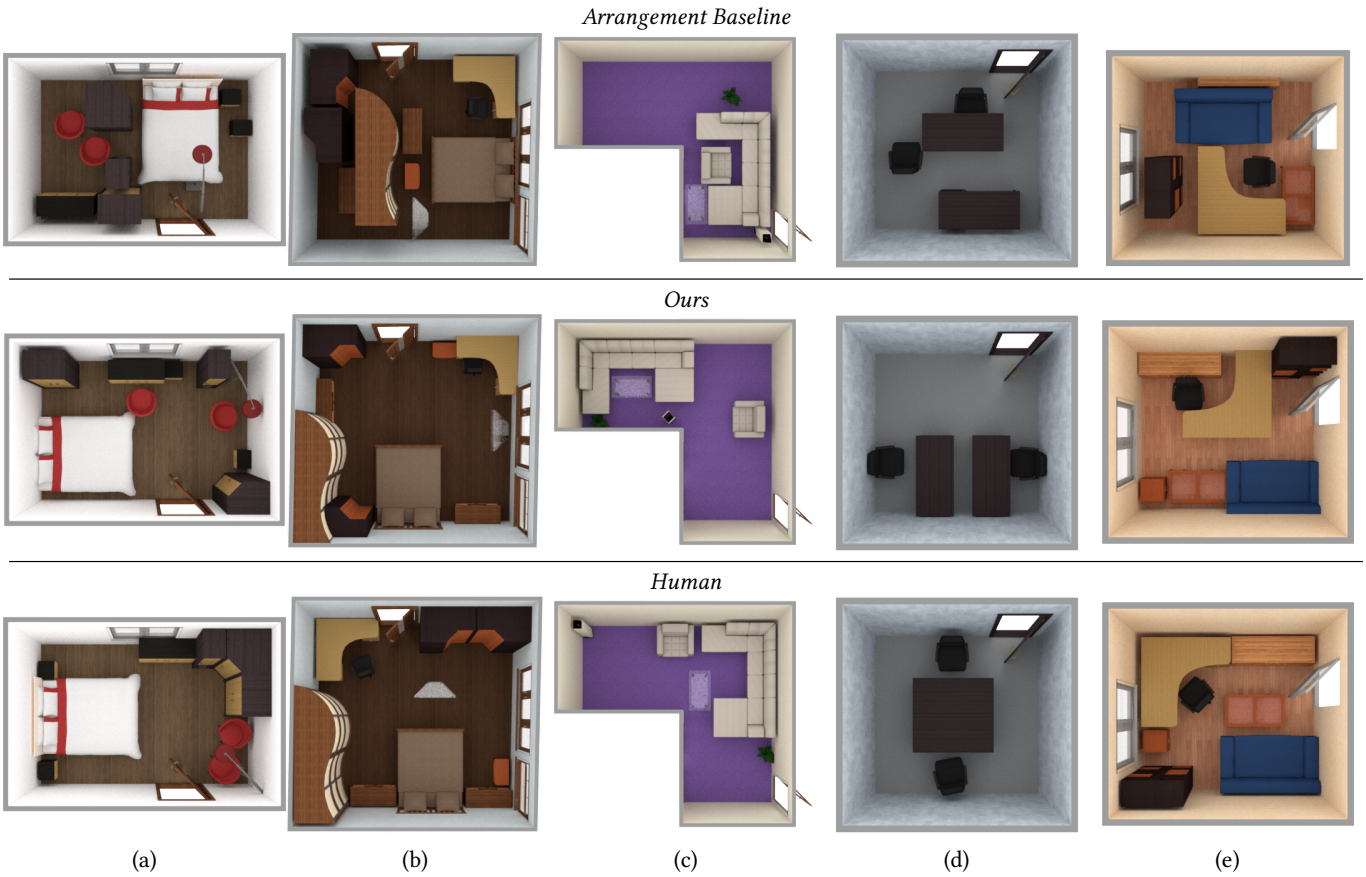

Fig. 11. Comparison of outputs using baseline arrangement model (top) and our full model (middle), with original human-designed scenes (bottom). In columns (a) and (b), the baseline arranges objects packed too close together and not fully making use of space against the room walls. In column (c), the space in front of the L-shaped couch is taken up by a chair instead of the glass coffee table. In column (d), both chairs are placed close to one of the tables, while in column (e), the blue couch is too close to the desk. In contrast, our model generates arrangements that more plausibly make use of free space.

and enforcing a stronger coupling between object location and geometry are important avenues for future work, and challenging problems for scene synthesis in general.

Scene variety and generalization. Finally, we examine our model's ability to generate a variety of scenes and generalize beyond its training dataset. Figure 12 shows multiple scenes synthesized in the same input room. Our model explores different possible positions for key objects (such as beds and sofas), chooses different combinations of objects, and generates rooms with varying functionality (e.g. living rooms with/without televisions, offices accommodating different numbers of people). Figure 13 shows four bedrooms synthesized by a model trained on a small dataset of 320 rooms. We also show the nearest neighbor of each room in the training set, using a distance function adapted from prior work [Henderson and Ferrari 2017]. Even with this smaller training set, our model generalizes and generates reasonable layouts that differ from rooms in the training set. This training set also contains no L-shaped rooms, yet our model can still generate layouts for them that respect the room shape.

7 DISCUSSION AND FUTURE WORK

We presented the first deep convolutional neural network-based system for generating object arrangements of entire rooms, given only the room wall architecture as input. Our multi-channel top-down view representation encodes the context of an entire room and enables the application of convolutional networks to the domain of 3D scene synthesis. We demonstrated that our system learns to iteratively condition object selection and placement on the global state of the room during synthesis. Finally, we evaluated the plausibility of rooms generated using our approach against representative object selection and arrangement baselines based on prior work.

Our system has several limitations that suggest future work. First, we represent scenes as flat collections of objects. However, indoor scenes exhibit hierarchy, with sets of objects forming 'functional groups' (e.g. *bed-and-nightstands*, *table-and-chairs*). Such groups also often exhibit symmetries, e.g. chairs being symmetrically arranged around a table. Our model currently struggles with such carefully-coordinated groupings of many objects (Figure 15). Explicitly incorporating hierarchy and symmetry into our model could

Fig. 12. Synthesizing multiple scenes in the same room. *Bedroom:* three different orientations of the bed and different wall-adjacent objects. *Living Room:* one room with a TV stand and three without, each with distinct sofa/table layouts. *Office:* synthesized rooms accommodate different numbers of occupants and place sofas and bookshelves where space is available.

Fig. 13. Examining our model's generalization capability. *Top row:* Bedrooms synthesized by a model trained on 320 rooms. *Bottom row:* For each synthesized room, we show its nearest neighbor in the training set. *From left to right:* adding a second bed to a common layout in the dataset; different desk/cabinet placement, given a similar bed position; generalizing to L-shaped rooms, which do not occur in the training set.

Fig. 14. Typical failure cases of our model. Clockwise from top-left: the living room contains no seats (global inconsistency), the bedroom has too many nightstands (local/global inconsistency), the wardrobe cabinet blocks the door (conflict between object location and geometry), the two beds don't leave enough room to walk between (global inconsistency, conflict between object location and geometry).

Fig. 15. Room types that our model currently does not handle well. *Top:* A human-designed kitchen and dining room. *Bottom:* Our approach struggles with generating carefully coordinated functional groups of objects, such as the contiguous placement of separate kitchen countertop sections and the symmetric arrangement of chairs around dining tables.

help address these problems, along with allowing us to tackle larger-scale scenes, such as classrooms, restaurants, and corporate offices (which the SUNCG dataset contains). Including latent variables for functional object groups, perhaps in a similar manner to recent work on learning generative models of object part hierarchies [Li et al. 2017], could prove fruitful here.

As mentioned in Section 5.4, sampling a scene from our model takes several minutes on average. Much of this computation time is spent in evaluating the `CategoryLocation` network for many candidate locations in the scene. This phase could be sped up if the computations for multiple locations could be shared. One possible approach would be to draw inspiration from fast region proposal networks for object detection, which use a common feature map for the whole image when evaluating candidate object regions [Ren et al. 2015]. Or, we could design the `CategoryLocation` network as an image-to-image translation function, predicting category probabilities across all pixels of the output image in one forward pass [Isola et al. 2017]. Predicting category probabilities across an entire room at training could also facilitate better global consistency, reducing the need for `CategoryLocation`'s global loss $\mathcal{L}_{\textbf{global}}$ and for tempering during sampling.

The model as described considers only floor-supported (i.e. 'first-tier') objects. It would be straightforward to incorporate 'second-tier' objects (i.e. those supported by other objects) using a second pass of our model, after all first-tier objects have been placed. It is also possible to include wall-mounted objects such as clocks and paintings by using multiple orthographic views as input to our networks, i.e. a top-down view along with a wall-facing view.

More broadly, our model makes no major assumptions specific to the domain of interior room design. We believe that deep convolutional priors extracted from multi-channel view representations such as the ones we have presented open up new possibilities for many application domains that involve reasoning over 3D scene structure. We hope that others can build on this first step we have taken into learning deep convolutional scene generation priors.

ACKNOWLEDGMENTS

Scene renderings shown in this paper were created using the Mitsuba physically-based renderer. This work is supported in part by Google, Intel, and with the support of the Technical University of Munich-Institute for Advanced Study, funded by the German Excellence Initiative and the European Union Seventh Framework Programme under grant agreement no 291763.

REFERENCES

Andrej Karpathy. 2015. char-rnn. https://github.com/karpathy/char-rnn. (2015). Accessed: 2018-01-20.

Angel X Chang, Manolis Savva, and Christopher D Manning. 2014. Learning Spatial Knowledge for Text to 3D Scene Generation. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Kang Chen, Yukun Lai, Yu-Xin Wu, Ralph Robert Martin, and Shi-Min Hu. 2014. Automatic semantic modeling of indoor scenes from low-quality RGB-D data using contextual information. *ACM Transactions on Graphics* 33, 6 (2014).

Kang Chen, Kun Xu, Yizhou Yu, Tian-Yi Wang, and Shi-Min Hu. 2015. Magic Decorator: Automatic Material Suggestion for Indoor Digital Scenes. In *SIGGRAPH Asia 2015*.

B. Efron and R. Tibshirani. 1986. Bootstrap Methods for Standard Errors, Confidence Intervals, and Other Measures of Statistical Accuracy. *Statist. Sci.* 1, 1 (02 1986), 54–75.

Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, and Joshua B. Tenenbaum. 2017. Learning to Infer Graphics Programs from Hand-Drawn Images. *CoRR* arXiv:1707.09627 (2017).

S. M. Ali Eslami, Nicolas Heess, Theophane Weber, Yuval Tassa, David Szepesvari, Koray Kavukcuoglu, and Geoffrey E. Hinton. 2016. Attend, Infer, Repeat: Fast Scene Understanding with Generative Models. In *NIPS 2016*.

Matthew Fisher, Daniel Ritchie, Manolis Savva, Thomas Funkhouser, and Pat Hanrahan. 2012. Example-based Synthesis of 3D Object Arrangements. In *SIGGRAPH Asia 2012*.

Matthew Fisher, Manolis Savva, Yangyan Li, Pat Hanrahan, and Matthias Nießner. 2015. Activity-centric Scene Synthesis for Functional 3D Scene Modeling. (2015).

Qiang Fu, Xiaowu Chen, Xiaotian Wang, Sijia Wen, Bin Zhou, and Hongbo Fu. 2017. Adaptive Synthesis of Indoor Scenes via Activity-associated Object Relation Graphs. In *SIGGRAPH Asia 2017*.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *NIPS 2014*.

Karol Gregor, Ivo Danihelka, Alex Graves, and Daan Wierstra. 2015. DRAW: A Recurrent Neural Network For Image Generation. In *ICML 2015*.

David Ha and Douglas Eck. 2017. A Neural Representation of Sketch Drawings. *CoRR* arXiv:1704.03477 (2017).

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *CVPR 2016*.

Paul Henderson and Vittorio Ferrari. 2017. A Generative Model of 3D Object Layouts in Apartments. *CoRR* arXiv:1711.10939 (2017).

H. Huang, E. Kalogerakis, S. Chaudhuri, D. Ceylan, V. Kim, and E. Yumer. 2017. Learning Local Shape Descriptors with View-based Convolutional Neural Networks. *ACM Transactions on Graphics* (2017).

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. 2017. Image-to-Image Translation with Conditional Adversarial Networks. In *CVPR 2017*.

Evangelos Kalogerakis, Melinos Averkiou, Subhransu Maji, and Siddhartha Chaudhuri. 2017. 3D Shape Segmentation with Projective Convolutional Networks. In *CVPR 2017*.

Z. Sadeghipour Kermani, Z. Liao, P. Tan, and H. Zhang. 2016. Learning 3D Scene Synthesis from Annotated RGB-D Images. In *Eurographics Symposium on Geometry Processing*.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR 2015*.

Diederik P. Kingma and Max Welling. 2014. Auto-Encoding Variational Bayes. In *ICLR 2014*.

Jun Li, Kai Xu, Siddhartha Chaudhuri, Ersin Yumer, Hao Zhang, and Leonidas Guibas. 2017. GRASS: Generative Recursive Autoencoders for Shape Structures. In *SIGGRAPH 2017*.

Yuan Liang, Song-Hai Zhang, and Ralph Robert Martin. 2017. Automatic Data-Driven Room Design Generation. In *Next Generation Computer Animation Techniques: Third International Workshop (AniNex 2017)*, Jiana Chang, Jian Jun Zhang, Nadia Magnenat Thalmann, Shi-Min Hu, Ruofeng Tong, and Wencheng Wang (Eds.).

Isaak Lim, Anne Gehre, and Leif Kobbelt. 2016. Identifying Style of 3D Shapes using Deep Metric Learning. In *Eurographics Symposium on Geometry Processing*.

Tianqiang Liu, Aaron Hertzmann, Wilmot Li, and Thomas Funkhouser. 2015. Style Compatibility for 3D Furniture Models. In *SIGGRAPH 2015*.

Paul Merrell, Eric Schkufza, Zeyang Li, Maneesh Agrawala, and Vladlen Koltun. 2011. Interactive Furniture Layout Using Interior Design Guidelines. In *SIGGRAPH 2011*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *CoRR* arXiv:1301.3781 (2013).

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. (2017).

Xue Bin Peng, Glen Berseth, Kangkang Yin, and Michiel Van De Panne. 2017. DeepLoco: Dynamic Locomotion Skills Using Hierarchical Deep Reinforcement Learning. In *SIGGRAPH 2017*.

Planner5d. 2017. Home Design Software and Interior Design Tool ONLINE for home and floor plans in 2D and 3D. https://planner5d.com. (2017). Accessed: 2017-10-20.

Siyuan Qi, Yixin Zhu, Siyuan Huang, Chenfanfu Jiang, and Song-Chun Zhu. 2018. Human-centric Indoor Scene Synthesis Using Stochastic Grammar. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *NIPS 2015*.

Daniel Ritchie, Anna Thomas, Pat Hanrahan, and Noah D. Goodman. 2016. Neurally-Guided Procedural Models: Amortized Inference for Procedural Graphics Programs using Neural Networks. In *NIPS 2016*.

Manolis Savva, Angel X. Chang, Pat Hanrahan, Matthew Fisher, and Matthias Nießner. 2014. SceneGrok: Inferring Action Maps in 3D Environments. In *SIGGRAPH Asia 2014*.

Gopal Sharma, Rishabh Goyal, Difan Liu, Evangelos Kalogerakis, and Subhransu Maji. 2017. CSGNet: Neural Shape Parser for Constructive Solid Geometry. *CoRR* arXiv:1712.08290 (2017).

Shuran Song, Fisher Yu, Andy Zeng, Angel X Chang, Manolis Savva, and Thomas Funkhouser. 2017. Semantic Scene Completion from a Single Depth Image. *CVPR 2017*.

Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik G. Learned-Miller. 2015. Multi-view convolutional neural networks for 3d shape recognition. In *ICCV 2015*.

Minhyuk Sung, Hao Su, Vladimir G. Kim, Siddhartha Chaudhuri, and Leonidas J. Guibas. 2017. ComplementMe: Weakly-Supervised Component Suggestions for 3D Modeling. In *SIGGRAPH Asia 2017*.

Benigno Uria, Marc-Alexandre Cote, Karol Gregor, Iain Murray, and Hugo Larochelle. 2016. Neural Autoregressive Distribution Estimation. *CoRR* arXiv:1605.02226 (2016).
Aaron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. 2016. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499* (2016).
Aaron van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. 2016. Conditional image generation with pixelcnn decoders. In *Advances in Neural Information Processing Systems*. 4790–4798.
Kun Xu, Kang Chen, Hongbo Fu, Wei-Lun Sun, and Shi-Min Hu. 2013. Sketch2Scene: Sketch-based Co-retrieval and Co-placement of 3D Models. In *SIGGRAPH 2013*.
Ken Xu, James Stewart, and Eugene Fiume. 2002. Constraint-based automatic placement for scene composition. In *Graphics Interface*, Vol. 2. 25–34.
Yi-Ting Yeh, Lingfeng Yang, Matthew Watson, Noah D. Goodman, and Pat Hanrahan. 2012. Synthesizing Open Worlds with Constraints Using Locally Annealed Reversible Jump MCMC. In *SIGGRAPH 2012*.
Lap-Fai Yu, Sai-Kit Yeung, Chi-Keung Tang, Demetri Terzopoulos, Tony F. Chan, and Stanley J. Osher. 2011. Make It Home: Automatic Optimization of Furniture Arrangement. In *SIGGRAPH 2011*.
Richard Zhang, Phillip Isola, and Alexei A Efros. 2016. Colorful Image Colorization. In *ECCV 2016*.
Jun-Yan Zhu, Richard Zhang, Deepak Pathak, Trevor Darrell, Alexei A Efros, Oliver Wang, and Eli Shechtman. 2017. Toward multimodal image-to-image translation. In *Advances in Neural Information Processing Systems*. 465–476.
C. Zou, E. Yumer, J. Yang, D. Ceylan, and D. Hoiem. 2017. 3D-PRNN: Generating Shape Primitives with Recurrent Neural Networks. In *ICCV 2017*.

A DATASET FILTERING

We apply the following filters to SUNCG scenes to produce our training/test datasets:

Low-quality rooms: Each SUNCG scene has been quality-rated by three human raters. We discard all rooms that do not have the highest rating, as well as any remaining rooms with large object interpenetrations.

Aggregate objects: We remove rooms containing objects that are actually sets of multiple objects, e.g. *chair_set* and *double_desk*.

Architectural features: We filter out rooms containing objects that are actually architectural features which alter the room geometry, such as *partition*, *column*, *stairs* and *arch*.

Inhabitants: We remove objects representing people and pets.

Infrequent objects: We filter out rooms containing unusual, infrequently occuring object categories. Specifically, we filter for categories that occur in less than 5% of rooms and are not functionally important, e.g. *vacuum_cleaner*, *fish_tank*, *cart*, *tripod*.

Outlier room sizes: We are interested in modeling residential-scale rooms, and our image-based scene representation must fit the extents of all training rooms. Thus, we filter out rooms larger than 6 m × 6 m in floor plan size and taller than 4 m. We also filter outliers in terms of object density by removing rooms with fewer than 4 or greater than 20 objects.

Rugs: We remove floor rugs, as they are very frequently non-uniformly (and thus non-physically) scaled to a large range of sizes and aspect ratios (up to the size of an entire room), making them inappropriate for category-based analysis.

B BASELINE OBJECT ARRANGEMENT MODEL

Existing object arrangement methods differ in terms of input/output assumptions, datasets, and user interactivity, making fair comparison hard. We select data-driven model elements common to multiple approaches to construct a representative baseline.

We learn our object arrangement baseline model by extracting pairwise observations for all objects in each room type. The pairwise observations in the training set are used to estimate parameters for three types of priors:

- Relative 2D offset between closest points on object o_i and point on reference object o_r in coordinate frame of o_r.
- Relative 2D clearance of object o_i from closest point on a wall in coordinate frame of o_i, and angle between front vector of o_i and offset to closest point on wall in frame of o_i.
- Relative orientation of front vector of o_i relative to front vector of o_r in coordinate frame of o_r.

Parameter estimation. The 2D offsets and 2D clearances between objects, and between an object and a wall point are used to estimate the parameters of a Gaussian mixture model $p(\theta) = \sum_{i=1}^{K} \phi_i \mathcal{N}(\mu_i, \Sigma_i)$. We use variational Bayesian estimation to infer the weights ϕ_i of up to $K = 5$ components, with a tied covariance matrix between components, and a Dirichlet process concentration prior set to $1/K$. The angles characterizing relative front orientation between objects are used to estimate the PDF over the angular domain as a histogram discretized into 16 equal bins of width $\pi/8$.

Synthesis procedure. At synthesis time, we order all objects to be arranged by decreasing bounding box volume. For each object, we uniformly sample 2D positions on the floor of the room, and assign one of eight cardinal orientations at random. The sampled position and orientation is checked for collisions with other objects or room wall geometry. If collisions exist, we take a new random sample.

We sample $n = 2000$ times and take up to $m = 200$ non-colliding candidate placements. We evaluate the probability of each candidate placement by treating the object to be placed as a reference object and computing the pairwise offset and orientation of all present objects and the closest point on a wall. The pairwise offset, clearance from wall, and relative orientations are evaluated against all pairwise priors to obtain the relative offset probability $p_{\mathrm{o}}(o_i, o_r)$, relative front angle probability $p_{\mathrm{a}}(o_i, o_r)$, and wall clearance and orientation probability $p_{\mathrm{w}}(o_i, p_w)$ where p_{w} is a point on a room wall. The overall log probability of a candidate placement x is then:

$$\log(p(x)) = p_{\mathrm{cooc}}(w_{\mathrm{o}} \log(p_{\mathrm{o}}(x)) + w_{\mathrm{a}} \log(p_{\mathrm{a}}(x)) + w_{\mathrm{w}} \log(p_{\mathrm{w}}(x)))$$

where p_{cooc} is the co-occurrence probability of the object category pair, estimated from the ratio of observations involving the pair out of all pairwise observations in the training set. We sum this pairwise log probability over all pairs involving the object to be placed, to estimate the likelihood of the candidate placement. We then take the placement with the maximum probability, and proceed to the next object in order of decreasing size. Arrangement ends when the smallest object has been placed. As this synthesis procedure has no mechanism for automatically rejecting questionable scenes, we disable that feature in our system in comparisons between the two.