Example-based Synthesis of 3D Object Arrangements

Matthew Fisher* Stanford University
Daniel Ritchie* Stanford University
Manolis Savva* Stanford University
Thomas Funkhouser† Princeton University
Pat Hanrahan* Stanford University

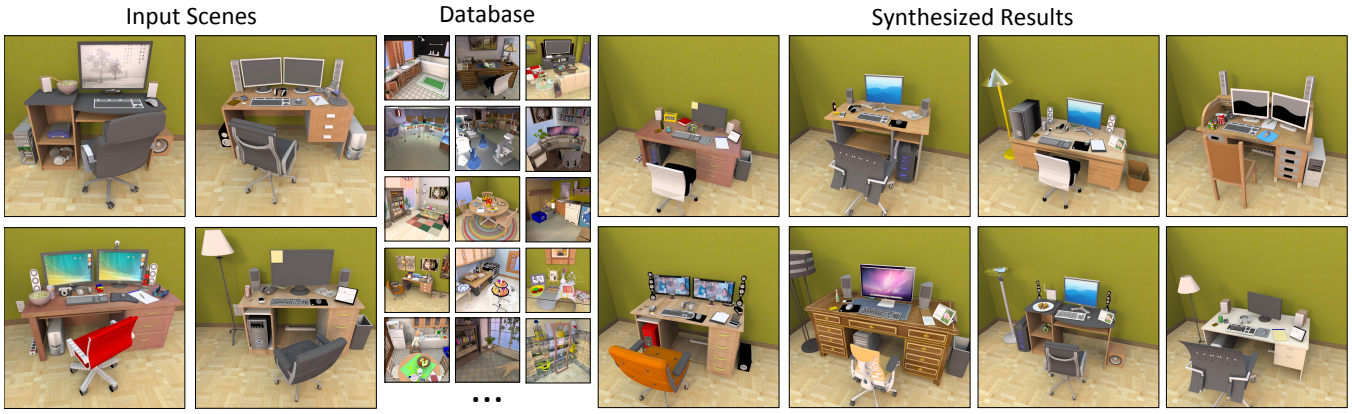

Figure 1: *Example-based scene synthesis. Left: four computer desks modeled by hand and used as input to our algorithm. Right: scenes synthesized using our algorithm. Our algorithm generates plausible results from few examples, incorporating object composition and arrangement information from a database of 3D scenes to increase the variety of its results.*

Abstract

We present a method for synthesizing 3D object arrangements from examples. Given a few user-provided examples, our system can synthesize a diverse set of plausible new scenes by learning from a larger scene database. We rely on three novel contributions. First, we introduce a *probabilistic model for scenes* based on Bayesian networks and Gaussian mixtures that can be trained from a small number of input examples. Second, we develop a clustering algorithm that groups objects occurring in a database of scenes according to their local scene neighborhoods. These *contextual categories* allow the synthesis process to treat a wider variety of objects as interchangeable. Third, we train our probabilistic model on a mix of user-provided examples and relevant scenes retrieved from the database. This *mixed model* learning process can be controlled to introduce additional variety into the synthesized scenes. We evaluate our algorithm through qualitative results and a perceptual study in which participants judged synthesized scenes to be highly plausible, as compared to hand-created scenes.

Keywords: 3D scenes, procedural modeling, automatic layout, probabilistic modeling, data-driven methods

Links: DL PDF WEB DATA

*e-mail: {mdfisher, dritchie, msavva, hanrahan}@stanford.edu
†e-mail: funk@cs.princeton.edu

ACM Reference Format
Fisher, M., Ritchie, D., Savva, M., Funkhouser, T., Hanrahan, P. 2012. Example-based Synthesis of 3D Object Arrangements. *ACM Trans. Graph. 31* 6, Article 135 (November 2012), 11 pages.
DOI = 10.1145/2366145.2366154 http://doi.acm.org/10.1145/2366145.2366154.

1 Introduction

Large-scale games and virtual worlds demand detailed 3D environments. Creating this content requires a lot of time from many artists. In this paper, we focus on scenes: environments composed of arrangements of 3D objects. They include many typical environments in games, animation, and virtual worlds, including most indoor and human-made environments.

A promising solution to this content-creation bottleneck is example-based synthesis: algorithms that can generate new environments similar to a set of input examples. A user provides the system with examples illustrating the desired type of scene, and the system returns a large set of synthesized results. The user then selects the results that she likes best from a ranked list. The user should not have to browse for a long time to find good scenes; as a rule of thumb, at least one out of every three synthesized results should be usable.

To support a tool such as the one described above, there are several criteria an example-based synthesis algorithm should meet. First, it should generate plausible scenes; they should look believable to a casual observer. Second, it should generate a variety of scenes; they should not be copies or small perturbations of the examples. Third, users should only have to provide a few examples, since they are time-consuming to create.

These goals are challenging to meet, and some stand in conflict with one another. Generating a variety of scenes is difficult when the system can only draw data from a few examples. Scenes can contain a large number of different objects, the full range of which can be difficult to specify in a few examples. Just because the user omitted some type of object in the examples, does that mean she does not want it in her scenes? Some objects are connected via precise functional and geometric relationships, while others are more loosely coupled. To generate plausible scenes, an example-based algorithm must infer these relationships from data without additional user guidance.

In this paper, we present an example-based scene synthesis method that meets the above challenges through three main contributions.

Our first contribution is a *probabilistic model for scenes*. It consists of an *occurrence model*, which specifies what objects should be in the generated scenes, and an *arrangement model*, which specifies where those objects should be placed. The occurrence model uses a Bayesian network, drawing on recent work in example-based shape synthesis [Kalogerakis et al. 2012]. The arrangement model uses a novel mixture of Gaussians formulation.

Our second contribution is a clustering algorithm that automatically discovers interchangeable objects in a database of scenes and forms them into groups. We call these groups *contextual categories*. A contextual category can contain a greater variety of objects than the basic categories (i.e. "table," "chair," "lamp") used by most applications. To find these categories, our algorithm exploits the insight that objects that occur in similar local neighborhoods in scenes (i.e. "on a plate," "beside a keyboard") are likely to be considered interchangeable when building plausible scenes. Using contextual categories, our algorithm can incorporate a wider variety of objects in synthesized scenes.

Our third contribution is a method for learning the probabilistic models from a mix of example scenes and scenes from a large database. In doing so, we treat the database as a "prior" over possible scenes. This provides a wide variety of scene content, and the examples guide that content toward a particular desired target. We allow the user to control the strength of the database prior through simple parameters, trading similarity to examples for increased diversity. Using these *mixed models*, our algorithm can synthesize scenes with a greater variety of both objects and arrangements.

Our results demonstrate the utility of our method for synthesizing a variety of scenes from as few as one sparsely populated example. Through a judgement study with people recruited online, we found that approximately 80% of synthesized scenes are of adequate quality to replace manually-created ones. This level of quality is more than sufficient for a user to browse a sorted list and find synthesized scenes of interest.

2 Background

Related work has tackled problems similar to example-based scene synthesis. While our algorithm uses some of the same underlying techniques, none of these methods alone are sufficient to meet the goals of this paper.

Component-based object modeling Recent work has demonstrated a graphical model for individual objects (such as chairs, planes, and boats) that encodes the cardinality, style, and adjacencies of object sub-components [Kalogerakis et al. 2012; Chaudhuri et al. 2011]. This model can be trained on input objects of a particular class and sampled to generate new objects that are recombinations of input object components. This approach is well-suited for object synthesis, since objects in a given class typically have few functional sub-components whose placement is well-specified by mesh adjacency information. Scenes do not share these properties: they can contain dozens of different, loosely-related objects, and each object is free to move continuously on its supporting surface. Our algorithm uses a separate *arrangement model* to capture the continuous spatial relationships between many possible objects. The *occurrence model* determines which objects should go in a scene and is based on the Bayesian network formalization used in Chaudhuri et al. [2011], with several modifications to better support scene synthesis.

Evolutionary object modeling Another approach to object synthesis evolves a set of diverse objects that is iteratively fit to a user's preferences [Xu et al. 2012]. With this scheme, generated objects are always an interpolation of the initial input set of objects. Thus, the algorithm cannot introduce any new object sub-components that were not present in the input set. This restriction is acceptable for objects, since they typically have a handful of functional subcomponents. Scenes can contain dozens of loosely-related objects, however, so using this method on a few input scenes will generate repetitive content. Since our algorithm extracts *contextual categories* and learns *mixed probabilistic models* from a database of scenes, it can incorporate new types of objects not found in the examples and increase the variety of synthesized results.

Inverse procedural modeling Researchers have also synthesized objects from examples by inferring a procedural model that might have generated those examples [Bokeloh et al. 2010]. This system searches for partial symmetries in the example object geometry, cuts the objects into interlocking parts based on those symmetries, and then generates a grammar that can synthesize new objects by stitching compatible parts together. This method works well with objects that are defined by repeated, regular sub-structures—a property that most scenes do not exhibit. Salient relationships in scenes cut across levels in the scene graph hierarchy, so context-free grammars are unlikely to model them well. In contrast, our probabilistic models for object *occurrence* and *arrangement* can learn salient existence and placement relationships between any pair of objects.

Automatic furniture layout Outside the domain of object synthesis, methods have recently been developed for optimizing the layout of furniture objects in interior environments [Merrell et al. 2011; Yu et al. 2011]. These methods define an energy functional representing the 'goodness' of a layout, and then use stochastic optimization techniques such as simulated annealing to iteratively improve an initial layout. While they can generate plausible and aesthetically pleasing furniture arrangements, these methods do not completely synthesize scenes, since they require a user to specify the set of furniture objects to be arranged. In contrast, our algorithm chooses what objects exist in the scene using its *occurrence model*. These methods also require non-example input: Yu et al. [2011] uses example layouts but requires important object relationships to be marked, and Merrell et al. [2011] uses domain-specific principles from interior design and requires a user in the loop. Our *arrangement model*, based on Gaussian mixtures and Gaussian kernel density estimation, can be trained entirely from data. It uses a heuristic for relationship salience to automate arrangement of scenes with dozens of objects.

Open-world layout Closely-related recent work seeks to automatically generate 'open-world' layouts, which are layouts with an unspecified number of objects [Yeh et al. 2012]. The underlying generative model is a probabilistic program, which is compiled into a factor graph representation and sampled using a variant of Markov Chain Monte Carlo. The algorithm succeeds at synthesizing interior environments, such as coffee shops, with varying shapes and scales. However, the underlying probabilistic program is written by hand and can only generate patterns that were expressed in the code. In contrast, our algorithm learns its generative model from data and does not require any programming ability on the part of the user.

3 Approach

The goal of this paper is to develop a method for synthesizing scenes from a few input examples. The fundamental challenge is that scene synthesis is hard due to the number of possible configu-

rations. Even for a single type of scene with a modest number of objects, such as an office desk scene, the restricted set of plausible configurations that we would like to synthesize is large. Furthermore, a user who provides a few examples can only express a small number of desired configurations, but the assumption that they would like synthesized results to conform to only the objects and arrangements in the examples is usually false. We need to address this combination of a large output domain and a restricted set of inputs with an approach that avoids generating repetitive results while retaining plausibility and similarity to the examples.

Our insight is that, much like users have diverse background knowledge from which they draw to construct examples, we can turn to a large database of scenes in order to "fill in the gaps" between the examples. The scene database provides many models instantiated in plausible configurations which we exploit in two ways to improve the diversity of our synthesized scenes. Firstly, we compute contextual categories by using object neighborhoods in scenes to group together objects that are likely to be considered interchangeable. These contextual categories then allow the synthesis algorithm to use more objects in any given role. We describe a simple algorithm to compute these categories using bipartite matching and hierarchical agglomerative clustering. Secondly, we treat the scene database as a prior over scenes and train our probabilistic model on both the examples and relevant scenes from the database. We use a recently-developed similar-scene retrieval technique to choose scenes relevant to the examples and introduce diversity without impacting plausibility. The user can control the degree of mixing between the two data sources via intuitive blending parameters.

For our investigation, we recruited participants to construct a database of interior scenes. Participants assembled scenes using a simple interface that allows placement, rotation, and scaling of models from Google 3D Warehouse. The scenes are segmented cleanly into individual objects, and each object is tagged with a *basic category label* such as "clock" or "bed." Each object is also rooted to a point on the surface of another object; the set of all such "parent-child" relationships defines a *static support hierarchy* over the scene that our algorithm exploits. Section 9 describes the results of our database construction effort in more detail.

Our system begins by retrieving contextual categories from a large database of scenes (Section 4). Given these categories, it then learns mixed models from both the user provided examples and the scene database (Section 5). It first trains the occurrence model, which describes what objects can be in synthesized scenes (Section 6). It then trains the arrangement model, which describes where those objects can be placed (Section 7). Finally, it samples from these probabilistic models to synthesize new scenes (Section 8).

4 Contextual Categories

We would like our synthesis algorithm to be able to draw from a wide variety of objects in order to increase the diversity of our scenes. Previous work addressing furniture layout used predefined sets of basic functional categories, such as "chairs" [Merrell et al. 2011; Yu et al. 2011]. For scene synthesis, such a categorization can be restrictive, leading to repetitive scenes. We note that many objects are interchangeable with other objects not necessarily from the same basic category. For example, a contextual category of "objects that belong on a plate in a kitchen" may contain many different basic categories such as "apples," "bread," "cookies," etc.

Our insight is that such interchangeable objects are frequently surrounded by objects that are similar both in type and arrangement. We use the term *neighborhood* to refer to the arranged collection of models around an object. This insight suggests that neighborhood similarity can predict interchangeability to automatically construct contextual categories.

Neighborhood similarity To group objects using the similarity of their neighborhoods, we must first quantify neighborhood similarity. We reduce each object x to its centroid point plus its basic category label L^x. Comparing two neighborhoods then reduces to matching two labeled point sets, which is a well-studied problem in computer vision and structural bioinformatics. Popular methods include techniques based on geometric hashing [Wolfson and Rigoutsos 1997], bipartite matching [Diez and Sellarès 2007], and finding maximum-weight cliques in association graphs [Torsello et al. 2007].

Our approach is based on bipartite matching. To compare objects A and B, we transform the database scenes in which they occur such that A and B are at the origin and the normals of their supporting surfaces are aligned with $\vec{z} = [0, 0, 1]^T$. We form the complete bipartite graph between the objects in these two scenes, where the weight of each edge in the graph is:

$$k(a, b) = \mathbf{1}\{L^a = L^b\} \cdot G(|\vec{a} - \vec{b}|, \sigma_d) \cdot G(\min(|\vec{a}|, |\vec{b}|), \sigma_n)$$

Here, $\mathbf{1}\{\}$ is the indicator function, $\vec{o}$ is the vector-valued position of object o and $G(x, \sigma) = e^{-x^2/2\sigma^2}$ is the unnormalized Gaussian function. This equation states that objects should only match if they have the same basic category and if the distance between them is small. The third term decreases the significance of matches that occur far away from the objects that we are comparing (A and B). In our implementation, $\sigma_d = 15\,\text{cm}$ and $\sigma_n = 90\,\text{cm}$.

We solve for the maximum-weight matching $\mathbf{M}$ in the graph using the Hungarian algorithm [Kuhn 1955]. Our neighborhood similarity function is then:

$$n(A, B) = \mathbf{1}\{\text{isLeaf}(A) = \text{isLeaf}(B)\} \cdot G\left(\frac{|A| - |B|}{\min(|A|, |B|)}, \sigma_s\right) \cdot \sum_{(a,b)\in\mathbf{M}} k(a, b)$$

where isLeaf(o) = true if object o is a leaf node in its scene's static support hierarchy, and $|o|$ is the diagonal length of the bounding-box of object o. The first term states that two objects are similar if they serve the same support role (i.e. they either statically support other objects or do not). The second term compares the sizes of the objects themselves, and the third term compares the similarity of their neighborhoods. We use $\sigma_s = 1.5$.

We then discretize all possible rotations of B's neighborhood about $\vec{z}$ and find the orientation for which $n(A, B)$ is strongest

$$\bar{n}(A, B) = \max_{\theta} n(A, \text{rot}(B, \vec{z}, \theta))$$

and then normalize the result to be in the range $[0, 1]$

$$\hat{n}(A, B) = \frac{\bar{n}(A, B)}{\max(\bar{n}(A, A), \bar{n}(B, B))}$$

Clustering We cluster all objects in all database scenes using hierarchical agglomerative clustering (HAC) [Steinbach et al. 2000], where the merge score between two clusters C_1 and C_2 is

$$\textit{Merge}(C_1, C_2) = \max_{A\in C_1, B\in C_2} (1-\lambda_{\textbf{cat}}) \cdot \mathbf{1}\{L^A = L^B\} + \lambda_{\textbf{cat}} \cdot \hat{n}(A, B)$$

The influence of neighborhood similarity is controlled by $\lambda_{\textbf{cat}}$. When $\lambda_{\textbf{cat}} = 0$, clustering only considers basic category labels and will recover these basic categories exactly. As $\lambda_{\textbf{cat}}$ increases,

objects from different basic categories that occur in similar neighborhoods are encouraged to merge together. For our database, $\lambda_{\textbf{cat}} = 0.7$ yielded many useful categories; see Section 9 for examples. The merge score above is the single linkage criterion for HAC [Murtagh 1984], which states that two clusters are as similar as their most similar objects. Since it is susceptible to outliers, in practice we use the 90^{th} percentile rather than the maximum. We stop merging when the best cluster merge score is less than a similarity threshold $\tau = 0.1$.

Alignment Our synthesis algorithm requires all objects in a category to be aligned in a common coordinate frame. We align our objects using both their neighborhood similarity and their geometric features. We first choose an anchor object at random and align the neighborhoods of all other objects to the anchor's neighborhood using the neighborhood similarity function $\hat{n}$. This process alone can successfully align objects with strong neighborhood cues, such as keyboards. To further refine an alignment, we compute the sum of squared distances for all possible rotations, which has been shown to be effective at aligning certain categories of 3D models [Kazhdan 2007]. We snap our neighborhood alignment to the nearest local minima of this inter-object distance function. For some categories that are sensitive to changes in rotation, such as computer speakers and alarm clocks, we manually inspect and correct the inter-object alignments.

5 Learning Mixed Models

Creating many diverse scenes from a small number of examples is difficult even if we draw upon contextual categories. Frequently, users intend examples they provide to be rough guides of the type of scene they would like and may be missing many details, which the user may not have recalled but has seen in the past. Our insight is that a database of scenes can act as a prior over scenes and can be used to fill in missing details or enrich the synthesized scenes. The degree to which this is a desired behavior may vary depending on the user and task, so we would also like to control how strong we want the influence of the prior to be by using intuitive mixing parameters.

In Sections 6 and 7, we learn two models from a mixture of data provided by the few examples and a larger scene database. We define these models here and describe how to mix separate input data sources but defer discussion of the details of each model to the relevant sections. The occurrence model $\mathcal{O}(S)$ is a function which takes a scene S as input and returns a probability for the static support hierarchy of the objects in the scene. The arrangement model $\mathcal{A}(o, S)$ is a function which takes an object o positioned within a scene S and returns an unnormalized probability of its current placement and orientation.

During synthesis, we cannot blindly draw information from the database: if the examples depict bedrooms, we don't want to draw from bathrooms introducing toilets in our synthesized scenes. The need to isolate 'semantically similar' content from a large database has been recognized in prior work on data-driven content generation in computer graphics [Hays and Efros 2007]. During model training, we retrieve scenes similar to the example scenes $\mathfrak{E}$ using the graph kernel-based scene comparison operator described in Fisher et al. [2011]. To threshold the number of retrieved scenes we sort them by similarity value and use only the results with more than 80% of the maximum value, forming the relevant set of scenes $\mathfrak{R}$.

Our system then learns its probabilistic models from the scenes in both the examples $\mathfrak{E}$ and the relevant set $\mathfrak{R}$. The extent to which each data source is emphasized is determined by two mixing parameters $\lambda_{\textbf{occur}}$ and $\lambda_{\textbf{arrange}}$. For these parameters, a value of 0 denotes 100% emphasis on $\mathfrak{E}$ and a value of 1 denotes 100% emphasis on $\mathfrak{R}$.

Mixing the arrangement model is straightforward: we train one model on $\mathfrak{R}$, another model on $\mathfrak{E}$, and the linearly interpolate between them using $\lambda_{\textbf{arrange}}$. Section 7 describes the procedure for training a single model.

Mixing the occurrence model is less trivial since it uses Bayesian networks which cannot simply be interpolated. Instead, we use an enrichment approach where we replicate each input scene many times to create a larger training set containing N observations. Each scene in $\mathfrak{E}$ is replicated $\lceil \frac{N(1-\lambda_{\textbf{occur}})}{|\mathfrak{E}|} \rceil$ times and equivalently each scene in $\mathfrak{R}$ is replicated $\lceil \frac{N\lambda_{\textbf{occur}}}{|\mathfrak{R}|} \rceil$ times. The results in this paper use $N = 500$. The model is then learned as described in Section 6.

6 Occurrence Model

We learn from the replicated set of scenes described in Section 5 a model $\mathcal{O}(S)$ over the static support hierarchy of a scene S. Our support hierarchy model is broken down into two parts. First, we use a Bayesian network $\mathcal{B}(S)$ to model the distribution over the set of objects that occur in a scene. Second, given a fixed set of objects we use a simple parent probability table to define a function $\mathcal{T}(S)$ that gives the probability of the parent-child connections between objects in a scene.

6.1 Object Distribution

Following previous work on learning probabilistic models for part suggestion in 3D object modeling, we use a Bayesian network to represent our learned probability distribution [Chaudhuri et al. 2011]. We can sample from this Bayes net to produce plausible sets of objects that do not occur in the examples. Prior to performing structure learning, we transform the network to reduce the number of free parameters introduced, which is important when learning from a small number of examples. We also constrain our Bayesian network to guarantee that the set of objects generated can be connected together into a valid static support hierarchy.

We start by representing each input scene with a *category frequency vector* that counts the number of times each category occurs in the scene. We consider these vectors to be observations of discrete random variables over which we want to learn a Bayesian network. To guarantee that our network generates plausible scenes, we will define below a set of constraints that specify edges which must be present in the learned network. Given these edge constraints, we use a standard algorithm to learn the structure of our Bayesian network following prior work on object modeling by Chaudhuri et al. [2011] with the following modifications:

Support constraints Our Bayesian network should guarantee that for every generated object, there is another object that can support it. Thus we require that for every category C, the variable representing that category must be conditional on the variable for each parent category C' it has been observed on.

Booleanization Prior to learning, we transform our problem to better handle the case where we have few input scenes. Each random variable in our network has a maximum value that can be large if that category occurs many times. This makes Bayesian network learning challenging, as it introduces many free parameters. We address this problem by breaking each discrete random variable into a set of boolean variables. If a category C occurred

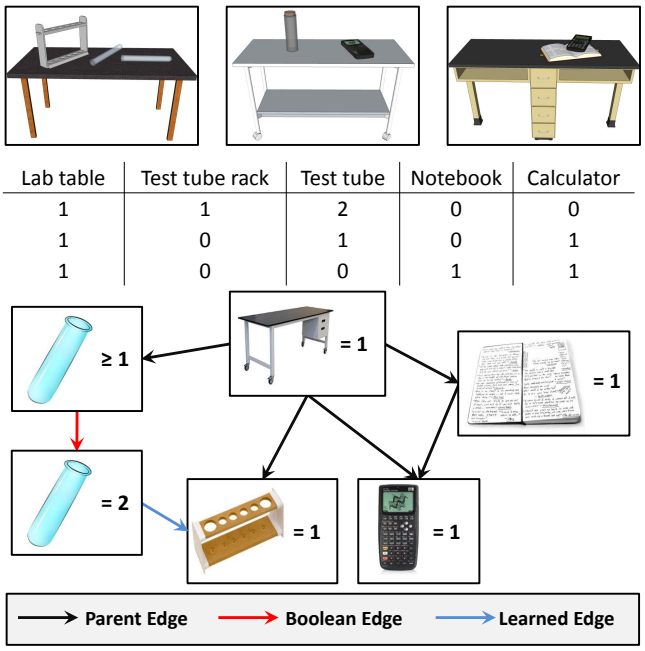

Figure 2: *Bayesian structure learning example. We start with three input scenes and their corresponding category frequency vectors. At the bottom we show a Bayesian network learned from these vectors. The black edges are enforced by observed static support relationships. Network booleanization splits the test tube variable into two nodes and enforces the edge shown in red. In blue, we show one edge learned by our structure inference algorithm that captures the positive correlation between multiple test tubes and a test tube rack.*

at most M times in an input scene, we introduce M boolean variables $(C \geq 1), (C \geq 2), ...(C = M)$. Higher count boolean variables can only be true if lower counts are true, so we constrain our learning algorithm by requiring that each node $(C \geq a)$ is a parent of $(C \geq a + 1)$. After transforming our problem into a set of boolean variables, we combine the set of booleanization-enforced edges with the support constraints defined above and apply our structure learning algorithm. Figure 2 shows an example of this learning process.

Input enrichment Bayesian structure learning algorithms can result in undesired correlations when few unique examples are provided. To help combat this problem, we use a perturbation method to add variety to our replicated scenes. We form new scenes by selectively removing objects from each input scene. We define a *decay coefficient* for each category $e^{-\alpha R(C)}$ where $R(C)$ is the fraction of input scenes that contain C (the results in this paper use $\alpha = 4$). Ubiquitous categories have small coefficients and are likely to be preserved, while infrequent categories have large coefficients and are more likely to be removed. Previous work has used similar perturbation methods to improve the robustness of learning algorithms for OCR and for speech recognition [Varga and Bunke 2003; Lawson et al. 2009].

6.2 Parent Support

To generate a static support hierarchy over a set of objects, we must also define a probabilistic model over possible parent-child static support connections. Let **Parents**(C) denote the set of categories that have been observed supporting a given category C in any input scene. We make the assumption that an object's support parent depends only on the existence of each category in this set. We build a *parent probability table* for each of the $2^{|\textbf{Parents}(C)|}$ different states of existence of the parent categories. For each observation of C in the input scenes, we look at the existence of each possible parent and record the actual parent in the corresponding parent probability table. We use this table to define the probability of any given parent-child support relationship. The probability $\mathcal{T}(S)$ of a given support hierarchy arrangement in a scene S is taken as the product of all its constituent support relationships according to this table.

6.3 Final Model

Given the components above, we can define the final probability of a given support hierarchy as the product of its object occurrence model and parent support model:

$$\mathcal{O}(S) = \mathcal{B}(S)\mathcal{T}(S)$$

where $\mathcal{B}(S)$ is the probability our learned Bayesian network assigns to the set of objects in the scene, and $\mathcal{T}(S)$ is the probability of the parent-child support hierarchy given the set of objects in the scene.

7 Arrangement Model

We must use the input scenes to learn, for each object category, the kinds of surfaces it can be placed on, spatial locations where it can go, and directions it can face. This is a challenging task because objects can have many valid configurations which are determined by functional relationships with other objects. People can easily identify which relationships are salient, but an example-based algorithm must infer this from data.

Previous object arrangement models do not meet our goals. [Yu et al. 2011] represents each position and orientation relationship with a single Gaussian. However, a single Gaussian does not account for multiple valid configurations. Also, users must specify which relationships are salient. Finally, their system does not address likely supporting surfaces for objects, since it arranged furniture on a flat floor.

We represent position and orientation relationships between all pairs of object categories using Gaussian mixture models. Gaussian mixtures can model multimodal distributions, thus handling objects with multiple valid configurations. We also describe a simple but effective heuristic for determining relationship saliency using object co-occurrence frequencies. Finally, we learn likely placement surfaces for objects using a simple Gaussian kernel density estimation approach.

7.1 Spatial Placement

From a set of input scenes, we learn a model of how objects are spatially arranged with respect to one another. Formally, we learn a probability distribution $\mathcal{P}_{C|C'}$ for every pair of categories C and C'. $\mathcal{P}_{C|C'}$ describes where category C objects tend to occur in the coordinate frame of category C' objects. It is a four-dimensional distribution over (x, y, z, θ), where $[x, y, z]$ defines an object's spatial position and θ defines its rotation about the normal of its supporting surface.

To learn these distributions, we first extract training data from the input scenes. To build robust models we need a large number of (x, y, z, θ) tuples, but the set of input scenes may be very small. Consequently, we extract many jittered (x, y, z, θ) samples from each object; 200 is more than enough, in our experience. We jitter

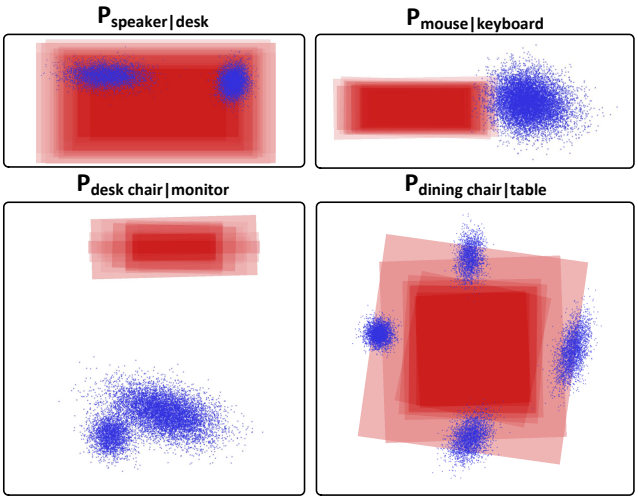

Figure 3: *Pairwise spatial distributions for object arrangement. Distributions are visualized as points drawn from a learned mixture of Gaussians. The bounding boxes for objects in the category the model is conditioned on are shown in red. Points have been projected into the xy plane; the z and θ dimensions are not shown.*

object positions by $\alpha \sim \mathcal{N}(0, 25\,\text{cm} \cdot \mathbb{I}_3)$ and orientations by $\omega \sim \mathcal{N}(0, 5^\circ)$.

We represent $\mathcal{P}_{C|C'}$ with a Gaussian mixture model, trained using the expectation-maximization algorithm on the data described above. The number of Gaussians in the mixture is a latent variable; we choose the value that maximizes the Akaike information criterion [Akaike 1973]. This combats overfitting by favoring a low-complexity model unless the added complexity significantly boosts the model's likelihood. Figure 3 visualizes some of these learned models.

These distributions should not all be treated equally; we would like to emphasize 'reliable' relationships that occur more frequently in the input scenes. Thus, we also compute a set of pairwise weights $w_{C|C'}$, which we use in synthesis to indicate the relative importance of each distribution. $w_{C|C'} = f(C, C')^{30/n}$, where $f(C, C')$ is the frequency with which categories C and C' co-occur in the input scenes, and n is the number of input scenes. As desired, this weighting scheme emphasizes frequent relationships, where the definition of 'frequent' becomes more stringent with fewer input scenes.

7.2 Surface Placement

Different objects are supported by surfaces with different physical properties, often reflecting functional constraints: light switches and mice are found at particular elevations above the ground, and keyboards are unlikely to be placed inside a desk drawer. To capture this knowledge for each category, we observe how objects in that category are supported by other objects in a set of input scenes. For each support observation, we record the height above ground and the area of the supporting surface in a *surface descriptor*. We then treat these descriptors as independently sampled vectors in $\mathbb{R}^2$ and use kernel density estimation to construct a probability distribution over possible supporting surfaces.

First, we perform mesh segmentation to extract planar support surfaces for all objects in the scene database that statically support other objects. We use a simple greedy region growth algorithm which handles non-manifold input models and robustly extracts planar and near-planar surfaces [Kalvin and Taylor 1996].

After segmentation, we compute surface descriptors for all segments that support any other objects. Each descriptor is a point $(\sqrt{area}, height) \in \mathbb{R}^2$, where the square root of area enforces consistent units between dimensions. We can estimate the underlying probability density function using any kernel density estimation method [Silverman 1986]. For each object category C, we approximate the function by summing Gaussian kernels centered at the $\mathbb{R}^2$ point for each surface descriptor. The Gaussian bandwidth is set using the normal distribution approximation: $h_C = 1.06\hat{\sigma} n_C^{(-1/5)}$, where $\hat{\sigma}$ is the standard deviation of all surface descriptor observations and n_C is the number of observations in the category. This approximates the variability over surface descriptor space using $\hat{\sigma}$ while enlarging the bandwidth to account for less certainty when fewer observations are available.

We call the estimated density functions $\mathcal{U}_C(s)$; for a surface s, this function returns the probability under our model that an object of category C should occur on s. Figure 4 visualizes this function for a few object categories.

7.3 Final Model

Given the components described above, we can define the final arrangement model as

$$\mathcal{A}(o, S) = \mathcal{U}_C(\text{surf}(o, S)) \cdot \sum_{o' \in S, o' \neq o} w_{C^o|C^{o'}} \cdot \mathcal{P}_{C^o|C^{o'}}(o)$$

where $\text{surf}(o, S)$ is the supporting surface of object o in scene S. Intuitively, this distribution combines the probability of o's surface placement with the probability of its spatial placement according to all other objects in the scene.

8 Synthesis

Synthesizing a new scene is straightforward given the models learned in the previous sections. First, we generate a new static support hierarchy that defines the objects in the scene and their parent-child relationships. Then, we determine a plausible spatial layout for the objects.

8.1 Static Support Hierarchy

The occurrence model $\mathcal{O}$ admits a very efficient sampling approach. We first use forward sampling to sample from the Bayesian network learned in Section 6, which generates a set of objects for the scene. To determine their parent-child support relationships, we independently assign parents to the objects in each category by sampling from the appropriate parent probability table. If the scene contains multiple instances of the sampled parent category, we choose an instance at random. This procedure samples a valid parent for each object, but the overall set of parent-child relationships may contain cycles. We use rejection sampling to generate a valid configuration, and repeatedly sampling parent-child relationships until an acyclic assignment is found.

Next, we assign a specific model to each object in the generated hierarchy. For each category, we compute a *consistency probability*: the frequency with which a category occurs two or more times in an input scene with all objects using the same model. We decide whether all objects from that category in our scene should use the same model according to this probability. If not, we choose models from the category at random.

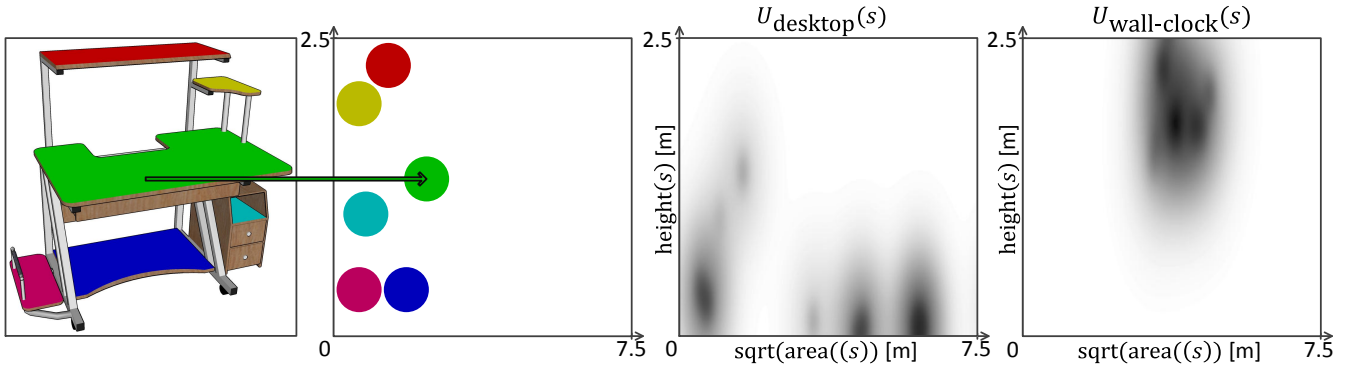

Figure 4: *Left: Map from supporting surfaces on a computer desk onto the 2D surface placement probability density function. Right: probability density functions for desktop computers and wall clocks.*

This procedure can sometimes produce implausible scenes because some objects might be asked to support more objects than can fit comfortably on their supporting surfaces. To avoid this problem, for each object that is supporting children, we compute the total surface area (after projection into the plane of their contact surface) of all the supported children. If the total supported surface area is greater than the total surface area supported by an instance of that object in the scene database, we reject the scene and resample from the Bayesian network.

8.2 Object Layout

Given a synthesized support hierarchy, we must determine the exact position and orientation of its constituent objects. This is challenging because we want the generated layout to respect both guidelines implicitly expressed in the examples as well as physical constraints, such as objects not colliding. To arrange our objects, for each object o in the scene we define a density function that describes o's preferred configurations:

$$\mathcal{D}(o) = \mathcal{A}(o) \cdot \mathcal{L}(o) \cdot \mathcal{X}(o) \cdot \mathcal{H}(o)$$

$\mathcal{A}$ is the arrangement model from Section 7. The other terms are defined as follows:

Collision Penalty ($\mathcal{L}$) Objects in physically plausible arrangements do not interpenetrate. $\mathcal{L}(o) = 1$ if o does not collide with any other objects in the scene, and 0 otherwise.

Proximity Penalty ($\mathcal{X}$) The arrangement model $\mathcal{A}$ is often multimodal. The modes may represent multiple configurations for a single object (such as a left or right-handed computer mouse) or locations for multiple instances of the same type of object (such as stereo speakers). In the latter case, multiple instances should not concentrate around the same mode. We prevent this behavior with $\mathcal{X}(o) = 1 - G(d, \mu_d)$, where d is the distance from o to the nearest o' such that $C^{o'} = C^o$, and μ_d is the average distance between the instances of C^o observed in the examples.

Overhang Penalty ($\mathcal{H}$) In some cases, o's most likely location according to the terms defined thus far may leave it hanging off the edge of its supporting surface. This is not physically plausible. We address this problem with $\mathcal{H}(o)$, which returns the percentage of o's projected bounding box that is contained by its supporting surface.

The density function $\mathcal{D}$ we have defined typically has a small number of isolated modes and is near zero almost everywhere else. To find a good initial layout, our algorithm places each object one at a time by sampling from $\mathcal{D}$. Objects are placed by order of decreasing size, since large objects often constrain the placement of others.

Finally, the algorithm iteratively improves this initial layout via hill climbing. Each iteration makes a small perturbation to the object configurations, as in prior work on automatic furniture layout [Merrell et al. 2011; Yu et al. 2011]. Proposed perturbations are accepted if they increase the total *layout score*,

$$\sum_{o \in S} \mathcal{D}(o)$$

For all the results in this paper, the algorithm stabilized within 100 iterations.

9 Results and Evaluation

In this section, we first describe the results of our scene database construction effort. To investigate the effectiveness of our method, we then synthesized several types of scenes under varying input conditions. We also conducted an experiment in which people provided subjective judgments on the plausibility of synthesized scenes.

9.1 Scene Database

Scenes in our database are constructed using a simple scene modeling interface. Unlike most modeling software, it does not allow individual object modeling. Instead, users populate scenes by placing, rotating, and scaling objects drawn from a collection of 12490 Google 3D Warehouse models.

We distributed our scene modeling program and asked participants in our database-building effort to model indoor scenes such as studies, kitchens, and living rooms. The majority of our participants were colleagues, or their friends and family. In total our participants generated 130 scenes, containing 3461 model instances and using 1723 distinct models. We provide this dataset as a resource to the research community; it can be found on the project web page: http://graphics.stanford.edu/projects/scenesynth.

9.2 Synthesis Results

Figure 1 shows scenes synthesized from an input set of four computer desks using $\lambda_{\text{occur}} = \lambda_{\text{arrange}} = 0.5$. The scenes are similar in style to the original four examples without being repetitive. The

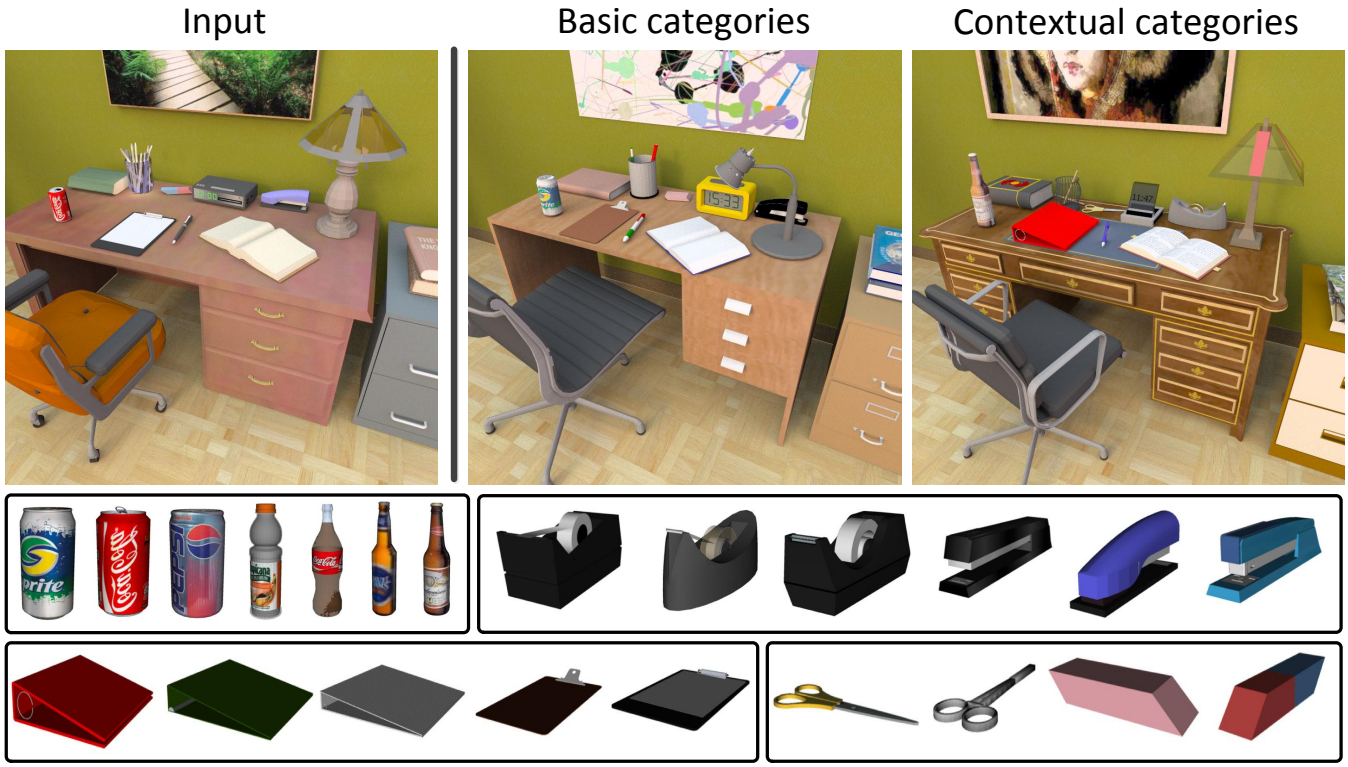

Figure 5: *Comparing basic and contextual categories. We show an input scene and the synthesized results using basic and contextual categories. On the bottom we show four of the relevant contextual categories generated by our clustering algorithm. Contextual categories allow synthesis with a greater variety of plausible models.*

synthesized scenes use a wide variety of models not found in the examples by drawing from the scene database. By using contextual categories and mixing information from a related set of scenes, our algorithm can insert plausible models not found in the input examples. For instance, the left two generated scenes both contain an alarm clock in a plausible location and orientation on the desk even though an alarm clock was not present in any of the input scenes.

In Figure 5, we compare the results of synthesizing using basic and contextual categories. In both cases we set $\lambda_{\mathbf{occur}}$ and $\lambda_{\mathbf{arrange}}$ to zero. When synthesizing using basic categories, our algorithm can only introduce variety by replacing each object with another model from the same basic category. Synthesizing using contextual categories can increase the variety of results by replacing objects in one basic category with objects from related basic categories. For example, using basic categories the soda can on the desk will only be replaced with other soda cans. In contrast, using contextual categories we draw from a broader set of object types including bottles.

In Figure 6 we show how the $\lambda_{\mathbf{occur}}$ term controls mixing in the occurrence model. The input is a single example with only a couple of objects. When $\lambda_{\mathbf{occur}} = 0$ we can only replace objects with other objects of the same category, resulting in equally simple scenes. As we increase $\lambda_{\mathbf{occur}}$ we incorporate more diversity from similar scenes in the database. Relevant new object categories are integrated in consistent arrangements with respect to the existing objects. At $\lambda_{\mathbf{occur}} = 1$, the only contribution of the objects in the input scene is to determine the set of relevant scenes. In the case of the two desk scenes, a difference of only two objects had a significant impact on the type of environment that was synthesized. Note that the scenes in the database are not categorized, nor was a 'desired type of scene' label provided with the input example.

Figure 7 demonstrates how the $\lambda_{\mathbf{arrange}}$ term can be used to control mixing of the arrangement model. The single input scene uses a desk with only one supporting surface. Without mixing, the synthesized results do not use other valid supporting surfaces present on the chosen desk models, leading to cluttering of some surfaces and barrenness of others. By increasing the value of the $\lambda_{\mathbf{arrange}}$ term, we leverage observations of model placements from the database to appropriately arrange objects on all available supporting surfaces of the parent model, even if similar models were not used in the user provided examples.

9.3 Human Evaluation

To evaluate whether our system consistently generates plausible scenes we ran a online judgment study on three types of scenes: *Gamer Desks*, *Study Desks*, and *Dining Tables*. Our hypothesis is that a human observer will consider a significant fraction of the synthesized scenes to be as plausible as scenes created by hand.

For each of the three scene types, we created scenes under the following experimental conditions:

1. *Manually Created (Manual)*: We manually created four scenes; building each scene took approximately 15-20 minutes for an experienced user of our scene modeling tool.
2. *Synthesized (Synth)*: Scenes generated by our system using a mixed model, trained on the four *Manual* scenes plus relevant scenes retrieved from the database. $\lambda_{\mathbf{occur}} = \lambda_{\mathbf{arrange}} = 0.25$. We generated 50 scenes in this condition.

We then rendered images of all of these scenes. Within a given scene type, scenes were rendered against the same background and using the same camera configuration.

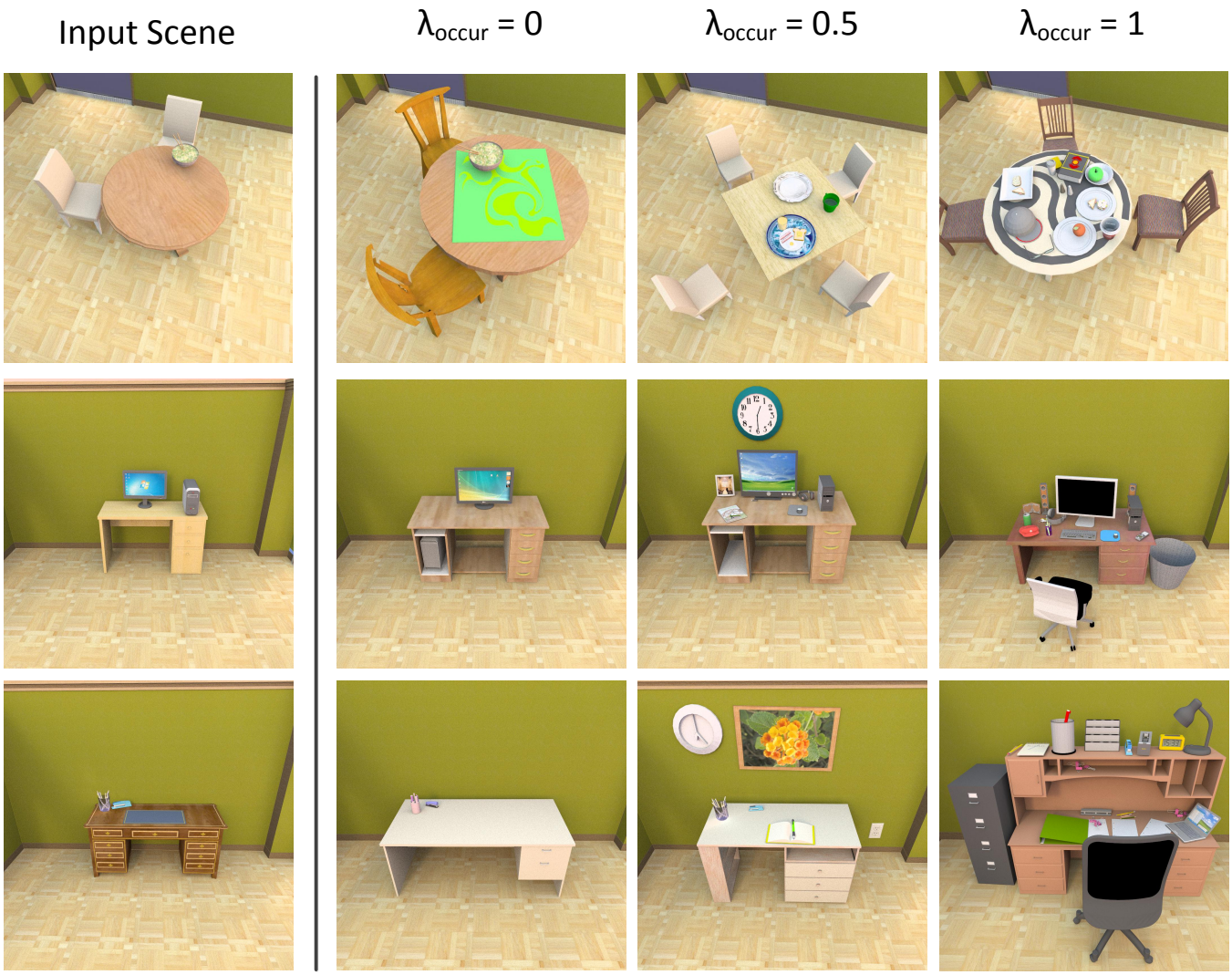

Figure 6: *Effects of varying the λ_{occur} term. Left: a manually created input scene. Right: results generated at three different values of λ_{occur}. Even a sparsely populated example can direct the algorithm to retrieve and incorporate relevant content from the database.*

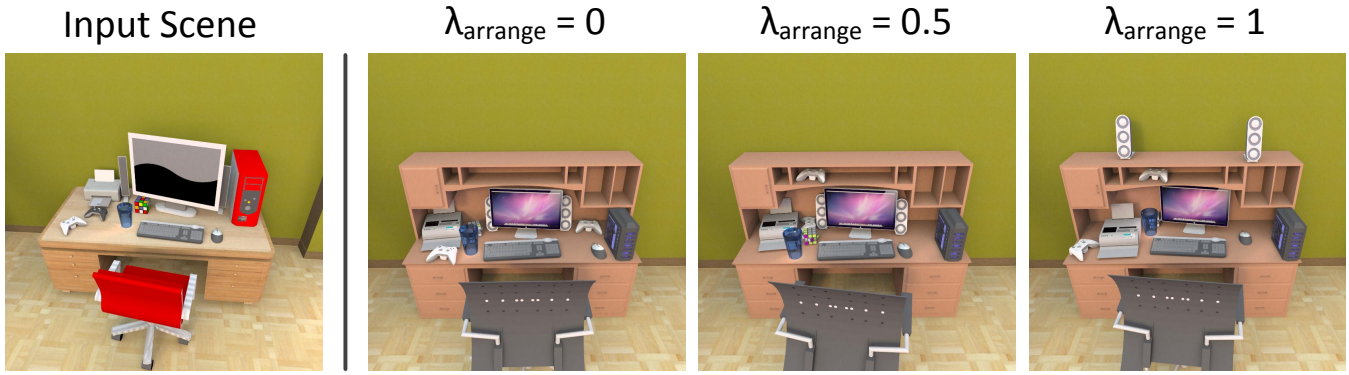

Figure 7: *Effects of varying the $\lambda_{arrange}$ term. Left: a manually created input scene. Right: results generated at three different values of $\lambda_{arrange}$. With $\lambda_{arrange} = 0$, the objects are only placed on the main desk surface, as in the desk from the input scene.*

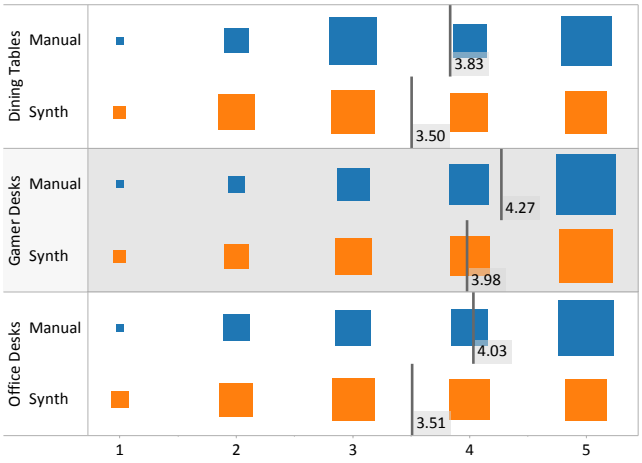

Figure 8: *Results of an online judgment study in which people evaluated the plausibility of synthesized scenes (*Synth*) and manually created scenes (*Manual*) for three scene types. Each row is a histogram for one scene type/condition pair; the area of each box corresponds to the percentage of responses with that rating. Average ratings are plotted as vertical lines.*

We recruited 30 participants via Amazon Mechanical Turk. Participants were required to be US residents to mitigate cultural influences on familiarity with different scene types. Through a web interface, each participant was shown 51 scene images: four from the *Manual* condition for each scene type, and 13 drawn at random from the *Synth* condition for each scene type. Participants were shown images one at a time in randomized order. Participants were asked to specify, on a 5-point Likert scale, the plausibility of the scene (1 = "Completely random, implausible scene," 3 = "Somewhat plausible scene," 5 = "Very plausible scene"). Rating a large set of more than 50 images helped participants calibrate their responses.

Figure 8 shows a summary of the ratings obtained through this experiment. Manual inspection of the data revealed no evidence of 'freeloading' or misbehaving workers, so we did not filter the data prior to anaylsis. Eliciting subjective responses from workers on Mechanical Turk is an inherently noisy process, but as expected, the responses for *Manual* scenes are concentrated toward the high end of the rating scale. In addition, the distribution of responses for *Synth* scenes closely matches the distribution for *Manual* scenes. *Manual* scenes were rated higher, on average, than *Synth* scenes for all scene types, and this difference is statistically significant (Mann-Whitney U test, $p < 0.05$). However, the difference is a small one: across all scene types, ratings for the top 80% of *Synth* scenes are *not* statistically distinguishable from ratings for *Manual* scenes. This result suggests that on average, at least 80% of synthesized scenes are not distinguishable from hand-created scenes by a casual observer. It also far exceeded our initial goal of having one third of the synthesis results be of comparable plausibility to manually-created scenes. The images used in this study, as well as the data obtained, can be found in the supplemental materials.

10 Discussion and Future Work

In this paper, we present a method for synthesizing arrangements of 3D objects from examples. We develop a probabilistic model for scenes composed of 3D objects, and we describe how to learn it from data. We introduce the concept of contextual categories and present an algorithm for computing them from a database of scenes. We also describe a procedure for learning our probabilistic model from a mix of specific example scenes and relevant scenes retrieved from the database. Using the above techniques, our system can synthesize a variety of plausible new scenes, requiring few examples as input from a user. Finally, we validate the quality of our synthesized results with an evaluation using human participants.

Our algorithm has several limitations that suggest areas for future work. First, scenes are often constructed with a particular style in mind, such as "antique" or "modern," and our algorithm does not capture these properties. This can result in synthesized scenes that might be functionally viable but stylistically objectionable. Second, our hill-climbing layout algorithm cannot escape local optima, nor can it add or remove objects from the scene to resolve overcrowding. A more sophisticated sampler, such as a transdimensional MCMC method, could alleviate these problems [Yeh et al. 2012]. Third, since its underlying models are probabilistic, our method cannot represent 'hard constraints' such as rigid grid layouts or exact alignment relationships. Future work might allow the user to edit model components and mark them rigid if needed, then synthesize scenes using a combination of probabilistic sampling and a constraint satisfaction approach such as the one in Xu et al. [2002]. Finally, we have only evaluated our method on small-scale scenes. Moving forward, we would like to address example-based synthesis of large-scale environments such as amusement parks or military bases. One way to make synthesis of these environments tractable is to represent them as a hierarchy of arrangements at different scales.

We introduced contextual categories and showed that they can add object variety to synthesized scenes. This is only a first step in exploring alternative categorization schemes for 3D objects. Our contextual categories are still a flat, disjoint list of object groups. In contrast, researchers in computer vision have shown that *hierarchical* categories can improve performance on challenging tasks such as content-based image retrieval [Deng et al. 2011]. Exploring techniques for automatically discovering such categorizations from collections of 3D models is an important direction for future work.

Embedding automatic scene synthesis technologies into interactive modeling tools presents another opportunity for future work. Users could manipulate higher-level scene primitives—such as 'fully decorated kitchen table'—using synthesized content. The interface could also suggest 'auto-completions' for the user's work-in-progress when he or she runs out of ideas. We believe that these types of tools can fundamentally change how people model 3D environments.

Acknowledgments

Support for this research was provided by the Fannie and John Hertz Foundation, a Stanford Graduate Fellowship, the Akiko Yamazaki and Jerry Yang Engineering Fellowship Fund, the NSF (CCF-0937139, CNS-0831374), Intel (ISTC-VC), Adobe, and Google. We would also like to thank all the 3D Warehouse users who created the models found in our scene database.

References

AKAIKE, H. 1973. Information theory and an extension of the maximum likelihood principle. In *Second International Symposium on Information Theory*, vol. 1, 267–281.

BOKELOH, M., WAND, M., AND SEIDEL, H.-P. 2010. A connection between partial symmetry and inverse procedural modeling. In *ACM SIGGRAPH 2010 papers*, ACM, New York, NY, USA, SIGGRAPH '10, 104:1–104:10.

CHAUDHURI, S., KALOGERAKIS, E., GUIBAS, L., AND KOLTUN, V. 2011. Probabilistic reasoning for assembly-based 3D modeling. *ACM Transactions on Graphics 30* (December).

DENG, J., BERG, A. C., AND FEI-FEI, L. 2011. Hierarchical semantic indexing for large scale image retrieval. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on 0*, 785–792.

DIEZ, Y., AND SELLARÈS, J. A. 2007. Efficient colored point set matching under noise. In *Proceedings of the 2007 international conference on Computational science and its applications - Volume Part I*, Springer-Verlag, Berlin, Heidelberg, ICCSA'07, 26–40.

FISHER, M., SAVVA, M., AND HANRAHAN, P. 2011. Characterizing structural relationships in scenes using graph kernels. In *ACM SIGGRAPH 2011 papers*, 34:1–34:12.

HAYS, J., AND EFROS, A. A. 2007. Scene completion using millions of photographs. *ACM Transactions on Graphics (SIGGRAPH 2007) 26*, 3.

KALOGERAKIS, E., CHAUDHURI, S., KOLLER, D., AND KOLTUN, V. 2012. A probabilistic model for component-based shape synthesis. *ACM Transactions on Graphics 31*, 4.

KALVIN, A., AND TAYLOR, R. 1996. Superfaces: Polygonal mesh simplification with bounded error. *Computer Graphics and Applications, IEEE 16*, 3, 64–77.

KAZHDAN, M. 2007. An approximate and efficient method for optimal rotation alignment of 3d models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 29*, 7 (july), 1221 –1229.

KUHN, H. 1955. The hungarian method for the assignment problem. *Naval research logistics quarterly 2*, 1-2, 83–97.

LAWSON, A., LINDERMAN, M., LEONARD, M., STAUFFER, A., POKINES, B., AND CARLIN, M. 2009. Perturbation and pitch normalization as enhancements to speaker recognition. In *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*, IEEE, 4533–4536.

MERRELL, P., SCHKUFZA, E., LI, Z., AGRAWALA, M., AND KOLTUN, V. 2011. Interactive furniture layout using interior design guidelines. In *ACM SIGGRAPH 2011 papers*, 87:1–87:10.

MURTAGH, F. 1984. Complexities of hierarchic clustering algorithms: state of the art. *Computational Statistics Quarterly 1*, 2, 101–113.

SILVERMAN, B. 1986. *Density estimation for statistics and data analysis*, vol. 26. Chapman & Hall/CRC.

STEINBACH, M., KARYPIS, G., AND KUMAR, V. 2000. A comparison of document clustering techniques. In *KDD workshop on text mining*, vol. 400, 525–526.

TORSELLO, A., ALBARELLI, A., AND PELILLO, M. 2007. Matching relational structures using the edge-association graph. In *Image Analysis and Processing, 2007. ICIAP 2007. 14th International Conference on*, 775 –780.

VARGA, T., AND BUNKE, H. 2003. Generation of synthetic training data for an hmm-based handwriting recognition system. In *Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on*, IEEE, 618–622.

WOLFSON, H., AND RIGOUTSOS, I. 1997. Geometric hashing: an overview. *Computational Science Engineering, IEEE 4*, 4 (oct-dec), 10 –21.

XU, K., STEWART, J., AND FIUME, E. 2002. Constraint-based automatic placement for scene composition. In *Graphics Interface 2002*, 25–34.

XU, K., ZHANG, H., COHEN-OR, D., AND CHEN, B. 2012. Fit and diverse: Set evolution for inspiring 3d shape galleries. *ACM Transactions on Graphics 31*, 4.

YEH, Y.-T., YANG, L., WATSON, M., GOODMAN, N. D., AND HANRAHAN, P. 2012. Synthesizing open worlds with constraints using locally annealed reversible jump mcmc. *ACM Transactions on Graphics 31*, 4.

YU, L.-F., YEUNG, S.-K., TANG, C.-K., TERZOPOULOS, D., CHAN, T. F., AND OSHER, S. J. 2011. Make it home: automatic optimization of furniture arrangement. In *ACM SIGGRAPH 2011 papers*, 86:1–86:12.