Context-Based Search for 3D Models

Matthew Fisher* Pat Hanrahan†

Figure 1: *Scene modeling using a context search. Left: A user modeling a scene places the blue box in the scene and asks for models that belong at this location. Middle: Our algorithm selects models from the database that match the provided neighborhood. Right: The user selects a model from the list and it is inserted into the scene. All models pictured in this paper are used with permission from Google 3D Warehouse.*

Abstract

Large corpora of 3D models, such as Google 3D Warehouse, are now becoming available on the web. It is possible to search these databases using a keyword search. This makes it possible for designers to easily include existing content into new scenes. In this paper, we describe a method for context-based search of 3D scenes. We first downloaded a large set of scene graphs from Google 3D Warehouse. These scene graphs were segmented into individual objects. We also extracted tags from the names of the models. Given the object shape, tags, and spatial relationship between pairs of objects, we can predict the strength of a relationship between a candidate model and an existing object in the scene. Using this function, we can perform context-based queries. The user specifies a region in the scene they are modeling using a 3D bounding box, and the system returns a list of related objects. We show that context-based queries perform better than keyword queries alone, and that without any keywords our algorithm still returns a relevant set of models.

Keywords: 3D model search, scene modeling, shape retrieval, spatial context

1 Introduction

One method for helping designers create content is to make it easy to incorporate content produced by others into their designs. It is simple for anyone to search for images and clip-art on the Internet, and then insert the results into her or his project. Although it is easier to find 2D content than 3D content, systems such as Google 3D Warehouse are changing that by creating large, searchable collections of 3D models. Users can upload models into Google 3D Warehouse, tag them, and then others can search for models using keyword queries.

At present, the tasks of 3D model search and scene modeling are decoupled. The user first performs the keyword search, and then inserts a model into the scene. The goal of this research is to develop a context-based 3D search engine. The workflow is shown in Figure 1. The user first selects a 3D box where they want to insert an object into the scene, and then searches for relevant objects at this location. The user can further refine the search by adding keywords to the query. In a sense, we have changed the order of operations from first searching and then placing, to first locating and then searching. We will show that by using the spatial context of a target location, a better set of models will be returned.

Our context search uses a data-driven approach, attempting to learn spatial relationships from existing scenes. First, we extract a large number of complete scenes from Google 3D Warehouse and segment these scenes into their constituent components. We then preprocess this dataset to determine for each model a set of similar models based on properties such as the model geometry and tags. We make the assumption that similar objects occur in similar contexts. Given a user-provided context, our algorithm finds clusters of related models in the database that have appeared in a similar context.

2 Background

2.1 Geometric Search Engines

Several 3D model search engines have been proposed [Funkhouser et al. 2003; Chen et al. 2003]. The most common approach used by these engines is to use a keyword search, where the user enters a text query and models are ranked based on how close the query matches a set of tags associated with the object. These per-model tags are often either already provided, mined from filenames or textual context, or automatically propagated from similar models [Min

*mdfisher@stanford.edu
†hanrahan@cs.stanford.edu

et al. 2004; Goldfeder and Allen 2008]. Another query type is for the user to provide either a 2D sketch or a 3D model (typically found via another search query) and ask the system to return similar models [Chen et al. 2003; Funkhouser and Shilane 2006]. These similarity queries usually work by reducing the models to a feature space that can be readily compared to produce a distance metric between any two models; we will make use of these distance metrics when designing our context-based query.

Some 3D search engines use relevance feedback to improve results. The user selects relevant models from a candidate list of results, and the search engine attempts to transform the feature space to favor objects similar to ones the user marks as relevant [Papadakis et al. 2008]. Although we have not yet explored this approach in our work, we believe relevance feedback can be very effective in scene composition, since for a given query different users may have wildly varying preferences for which models are desireable.

Work by Funkhouser et al. [2004] also presents a data-driven object modeling system. In their work, a user starts with a base model and then issues queries for related models that have desireable parts. To determine whether a candidate model is a good match to the query, they approximate the sum of the distances from every point on one surface to the closest point on the other, and vice-versa, weighting selected regions on the surface higher. The main difference between this work and our approach is that our focus is on scene composition containing a large number of disjoint models and not on finding similar parts for a single model. When comparing two scenes, we will not use a surface deformation approach and instead leverage the semantic segmentation and tagging of the scenes. Our algorithm will also attempt to learn general relationships between models from the database.

2.2 Spatial Context in Computer Vision

Computer vision research has made significant progress on using spatial context in photographs. One use of context information is to disambiguate two visually similar but semantically different objects. Some early work learned object relationships by explicitly enumerating rules about spatial context [Strat and Fischler 1991]. More recent work learns context from an existing set of images, each of which has been manually segmented into semantically meaningful objects that are then explicitly tagged; three popular image sets are the PASCAL, MSRC, and LabelMe datasets [Russell et al. 2008]. Another possible source of contextual relationships is Google Sets, which uses the Internet as a knowledge base to learn object co-occurrence but does not contain spatial relationships. The majority of work using these datasets learns a probability model over the relationships between commonly-occurring categories in the images. This model is then used to resolve context ambiguities in a new image by selecting the most probable label assignment [He et al. 2004; Rabinovich et al. 2007; Galleguillos et al. 2008].

One way of evaluating the success of this work is to measure how accurately it labels a set of test images. Labeling objects in images has a clear analog to the case of 3D scenes, where we are given a 3D scene and are asked to decompose it into a set of semantically meaningful 3D objects that we then label. Although this is one of the most commonly used methods for evaluating contextual understanding in 2D scenes, we do not focus on this problem in this paper.

For a model search engine, a more relevant evaluation method is the Context Challenge [Torralba 2010]. In this problem, the goal is to determine the identity of a hidden object given only its surrounding context. Recent work has looked at this problem in both category-based and category-free frameworks [Malisiewicz and Efros 2009]. In the category-based framework, the goal is to directly identify the unknown object by returning a weighted set of possible categories the object belongs to. In a category-free framework, the goal is to provide a set of 2D objects (represented as bitmaps seen in other images) that belong in the unknown region. A category-free framework is sometimes advantageous, since many problems can arise when attempting to categorize the set of meaningful objects. A related problem to the Context Challenge is scene completion, which attempts to fill or replace undesired regions in an input image [Hays and Efros 2007].

The category-free Context Challenge problem, extended to 3D, is precisely the context-based query we develop in this paper: given a query box in a 3D scene, return an ordered set of models that belong at this location. Although there are differences that arise in the 3D version of the problem, many of the techniques used to solve the problem in 2D are highly applicable and can be extended to the 3D case. In 2D, solving problems like the Context Challenge is often seen as a stepping stone towards 2D scene understanding. In the 3D case solutions to the Context Challenge find a direct application in geometric search engines.

3 Dataset

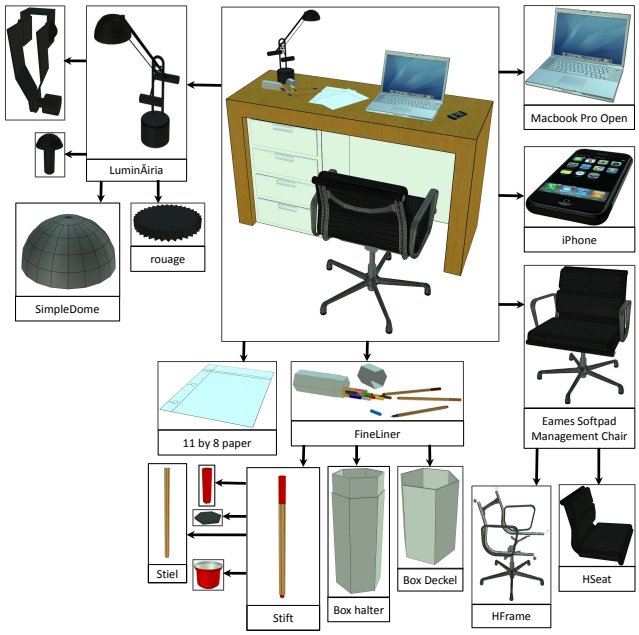
A 3D model database that contains only isolated objects, such as the Princeton Shape Benchmark, provides no information about the relationships between objects in real scenes [Shilane et al. 2004]. To learn these relationships, we need a dataset that contains many scenes, each composed of multiple objects.

Scene databases can largely be grouped into two categories: virtual worlds and "scene design" databases. Naturally, the choice of database will bias what kind of relationships are learned and may vary greatly between databases; for example, "axe" and "pillow" might be more closely related in a fantastical virtual world than in a scene design database. Using a virtual world as a learning source has several advantages: the world is often carefully designed by artists with a specific set of styles in mind that the context query could attempt to capture, the world is typically very large providing lots of training examples, and because users can interact with objects some semantic segmentation or tagging has often already been performed. Scene design databases, on the other hand, contain scenes that users have uploaded, often by combining other objects found in the database. These may contain models ranging from table decorations to dorm rooms to spaceship interiors, each with a wide range of artistic styles and object densities. Ultimately, the dataset to use for learning spatial context should be drawn from the target application whenever possible.

3.1 Segmentation

For this paper we use Google 3D Warehouse as our dataset, which contains both a large number of individual objects and scenes containing multiple objects. First, a set of models was acquired by searching with keywords that suggested scenes with multiple objects ("room", "office", "garage", "habitación", etc.). In total we acquired 4876 scenes this way. After downloading the scenes, we converted them to a COLLADA scene graph. The scene graph of a typical scene is seen in Figure 2.

The next step is to segment the scene graph into semantically meaningful objects. In these scene graphs, each node may point to any number of child graph nodes and to any number of child geometry objects. As can be seen in Figure 2, some nodes such as the Eames chair represent complete objects. Other nodes, such as the children of the Eames chair, represent parts of objects. We make little attempt to distinguish between whole objects and parts of objects. All nodes are considered objects, except for a few special cases. We



Name	Processed Tags
LuminAiria	None
rouage	Cog
SimpleDome	Simple, Dome
11 by 8 paper	Paper
FineLiner	Fine, Liner
Box Deckel	Box, Cover, Lid
Box halter	Box, Halter
Stift	Pin, Pencil
Stiel	Handle, Stem
Eames Softpad Mgmt. Chair	Soft, Pad, Management, Chair
HSeat	Seat
HFrame	None (frame is a stop word)
iPhone	iPhone
Macbook Pro Open	Macbook, Pro, Open

Figure 2: *Top: Typical scene and its scene graph decomposition (some nodes omitted for brevity.) Objects are labeled with their raw node names. Bottom: Scene graph node names and their final processed set of tags.*

ignore some common names that we found to be extremely suggestive of being an object part, such as "plant stem" or "flower petal".

After segmenting the scene into objects, we find objects that appear in multiple scenes. We use the term "model" to refer to the underlying texture and geometry. Each instance of the model is an object. Models are considered to match, if the following conditions are met:

- The shape (mesh) must be the same both topologically and geometrically, up to a rigid transform.
- The material properties must be the same. For models with textures, the textures must be the same to within a small epsilon when scaled to the same resolution.

This means that the same geometry may be found in multiple models each with different material styles.

One of the major predictors used in image scene understanding is size. Fortunately, Google 3D Warehouse recommends that the size of all objects be in inches. We have found that the vast majority of scenes respect the size metric, so it is easy to extract the absolute size of each object.

Scenes:	4,876
Scene Graph Nodes:	426,763
Objects:	371,924
Unique Models:	69,860
Tagged Models:	22,645
Shared Models:	10,509

Table 1: *Current snapshot of our database. A tagged model is a model with at least one tag. A shared model occurs in at least two scenes.*

3.2 Tags

Additional information about the object can be inferred from the names associated with the objects in the scene. In our dataset we gather naming information from three sources:

1. Scene graph nodes are sometimes named by the artist.
2. The root node of each scene (which corresponds to the entire file) is named by the artist as it is uploaded.
3. A model can be used in multiple scenes. We union the names from all instances of the model.

The first step in our tag processing pipeline is to clean up the names and translate them to English. First, we use Google auto-suggest to perform spelling correction and word separation (e.g., "deskcalender" becomes "desk calendar"). Second, we use Google Translate (which can auto-detect the source language) to convert Unicode sequences to English words. Finally, non-English words and stop words are removed.

The second step in the pipeline is to add related words to each word. For the top 60 most frequently occurring words, WordNet is used to find "hypernyms" of the word [Fellbaum et al. 1998]. Recall that hypernyms are enclosing categories of a word; for example, color is a hypernym of red which is a hypernym of crimson. We start with the word and follow "direct hypernym" links until a word in a set of stop categories is reached; we refer to the final set of words as the tags for that object.

Figure 2 shows a set of scene graph nodes, their raw names, and their processed tags.

Table 1 gives a summary of the dataset after processing. As we will show, this is a rich dataset and can be used to learn contextual relationships between a wide range of objects. We are currently in the process of obtaining permission to make this dataset publicly available.

4 Algorithm

We begin by defining some basic terminology for the context query. We are given a scene consisting of Q supporting objects, already placed by the user, and a query box with known coordinates where the user wants to insert a new object. Our goal is to rank each object in our database according to how well it fits into the query box (see Figure 1).

Although many of the algorithms used for learning 2D spatial context could be used as the basis for our context query, we chose to model our algorithm closely after The Visual Memex Model, because of its focus on the Context Challenge and category-free learning [Malisiewicz and Efros 2009]. Intuitively, whenever two objects f and g are observed in scene A, we take this as a suggestion that if we observe an object f' similar to f in scene A', an object g' that is similar to g is a good candidate model in scene A', provided the spatial relationship between f' and g' echoes that of f

and g. At a high level, our algorithm quantifies this concept of object similarity and the similarity between the spatial relationships of two objects, and then uses kernel density estimation over the set of observed object co-occurrences to determine the final ranking over all models.

4.1 Observations

We begin by considering all pairs of object co-occurrence across all scenes, each of which we will call an observation; we refer to the set of all such observations as O. Each observation has the following parameters: the 3D spatial relationship between the objects, and the size, geometry, texture and tags of each of the objects. Two observations are said to be similar if all of these properties are also similar. We use f_{st} as an abstract representation of the spatial relationship between two arbitrary objects s and t.

In the next two sections we will define the following similarity functions:

- $K_{\textbf{spatial}}(f_{st}, f_{uv})$: determines the similarity between two different spatial relationships. Evaluates to 0 if the spatial relationships are unrelated, and 1 if they are the same relationship.
- $S_{st}(\sigma_{\textbf{size}})$: determines the similarity between objects s and t by comparing their size, geometry, texture, and tags; $\sigma_{\textbf{size}}$ is the bandwidth of the size kernel which we will vary based on the objects being compared. Evaluates to 0 if the models are unrelated, and 1 if they are the same model.

Following the discussion of these two functions, we will describe our model ranking algorithm.

4.2 Spatial Relationships

We use a simple model to capture the similarity of the spatial relationship between arbitrary objects s and t. The model depends on two distances: the absolute height displacement z_{st} (along the Z-axis), and the absolute radial separation r_{st} (in the XY-plane). Both are measured between the centers of the bounding boxes of the objects. These distances have units of length, and we benefit from the fact that the input scenes use the same units.

We ran a simple experiment, and found that r_{st} and z_{st} are largely uncorrelated. We model the similarity of each distance with a Gaussian kernel $G(x, y, \sigma) = e^{-\|x-y\|^2/\sigma^2}$, and assume the two kernels are separable. Our final metric is:

$$K_{\textbf{spatial}}(f_{st}, f_{uv}) = G(z_{st}, z_{uv}, \sigma_z)G(r_{st}, r_{uv}, \sigma_r) \qquad (1)$$

The two bandwidths σ_z and σ_r have units of length. We choose both to be proportional to the length of the longest dimension of the objects being compared (l). Thus, the rate of fall-off in similarity is proportional to the size of the objects. In this study we used $\sigma_z = 0.05l$ and $\sigma_r = 0.5l$, which encourages objects to be aligned more precisely in z than r.

It would be nice to capture richer spatial relationships between objects. For example, the chair is in front of the desk, the couch faces the TV, or the plate is supported by the table. Measuring such relationships requires more sophisticated geometric analysis, such as the ability to define a consistent reference frame for each object. We leave such additional analysis for further work.

4.3 Object Similarity

Two objects are similar if both their sizes and models are similar. We assume that the size and model comparisons are separable kernels, and define the similarity between arbitrary objects s and t as:

$$S_{st}(\sigma_{\textbf{size}}) = G(\text{Size}(s), \text{Size}(t), \sigma_{\textbf{size}})K_{\textbf{model}}(s, t) \qquad (2)$$

To compare the size of two objects, we first sort the x-y dimensions of the bounding box of each object based on length. We consider the dimensions of the bounding box of each object to be a vector in $\mathbb{R}^3$. We use the Euclidean distance between these two vectors to compare the size of the objects.

$K_{\textbf{model}}(s, t)$ measures the similarity between two size-normalized models. For this paper we use a fixed linear combination of three functions:

Identical model: The simplest way to compare two models is to treat them as similar only if they are exactly the same. We denote exact matches using Kronecker Delta notation, δ_{st}. This metric ensures that we will always favor models that have been explicitly observed in proximity to the supporting objects. A high weight for this component also can help disambiguate objects that are both categorically and geometrically similar, such as pots for cooking vs. pots for planting.

Tags: If two models have a similar set of tags, they may be similar. One way of comparing two sets of tags is the Vector Space Model [Salton et al. 1975]; we use a simpler approach that looks at the relative overlap of tags. Let T_s and T_t be the set of tags for models s and t:

$$K_{\textbf{tag}}(T_s, T_t) = \frac{|T_s \cap T_t|}{\min(|T_s|, |T_t|)} \qquad (3)$$

Comparing models based on their tags has the same problems as using object categories in computer vision [Malisiewicz and Efros 2009]. Two very different types of objects (such as "skirt" vs. "skirt steak") may have similar tags. Nevertheless, tags offer a way to relate two semantically similar objects with very different shapes; for example, two very different types of "chairs."

Geometry: There is extensive research on shape similarity, and many of these methods could be used here. We use the 3D Zernike descriptors [Novotni and Klein 2003]. These descriptors have the desirable property that they are invariant under scaling, rotation, and translation.

Our Zernike descriptor computation closely follows work on auto-tagging models from Google 3D Warehouse [Goldfeder and Allen 2008]. We first scale the geometry into a unit cube and then voxelize it on a binary grid V that is 128 voxels on each side. We then thicken this grid by 4 voxels; a voxel in V' is set if there is a voxel set in V inside a sphere with a radius of 4 voxels. V' is used to compute a 121 dimensional Zernike descriptor using 20 levels of moments.

We use the Euclidean metric between two Zernike descriptor vectors as the distance between two shapes. We will use the distance to the n^{th} closest model as an estimate of the local density of models in the descriptor space. This distance is used to normalize the distance between two models. Let d_{st} be the Zernike descriptor distance between models s and t, and let $g_i(n)$ be the distance to the n^{th} closest model to model i. Our symmetric kernel between two models is given as:

$$K_{\textbf{geo}}(a, b) = e^{-\left(\frac{2d_{st}}{\min(g_s(n), g_t(n))}\right)^2} \qquad (4)$$

We chose the minimum value of $g_i(n)$ (corresponding to the region of greatest density) as our normalization term. This prevents

an object that lies outside any cluster from forming a geometric association with an object inside a good cluster. The results in this paper use $n = 100$.

Our final model comparison kernel is taken as:

$$K_{\textbf{model}}(s,t) = 0.1\delta_{st} + 0.6K_{\textbf{tag}}(T_s, T_t) + 0.3K_{\textbf{geo}}(s,t) \quad (5)$$

To make inference in our database easier, we zero out this model term if it is less than a small epsilon ($\epsilon = 10^{-6}$).

For our database, modifying these weights does have a noticeable effect on the results. While the geometry term was found to be informative (see Section 5.5), we weight tags higher than geometry for two reasons. First, for semantic categories with a high degree of structual variability, such as desks, geometry will be less useful than tags for forming good associations within that category. Second, in our database many different models are geometrically similar. For example, a large number of models are textured boxes or planes, and weighting the geometry term highly will make these models appear similar when they might not be. Although tags can also suffer from the homonym problem, we found this to be much less common than for geometry.

4.4 Model Ranking

We can use these two similarity metrics to estimate the probability that an object would appear in the context of other objects. We first define this probability for the case where the support scene contains only a single object b. Specifically, we need to compute the probability of placing an object a in the query box. Objects are ranked by their probabilities of being in the box.

We model this probability using the following conditional distribution:

$$p(a|b, f_{ab}) \propto \sum_{c \in M} S_{cb}(\sigma_0)\Psi(a, c, f_{ab}) \quad (6)$$

$$\Psi(a, c, f_{ab}) = \frac{\sum_{(u,v)\in O} S_{au}(\sigma_1)S_{cv}(\sigma_2)K(f_{ab}, f_{uv})}{\sum_{(u,v)\in O} S_{au}(\sigma_1)S_{cv}(\sigma_2)} \quad (7)$$

Here, $\Psi(a, c, f_{ab})$ is the pairwise compatibility between objects a and c. M is the set of objects in the database and O the set of observed object pairs.

This inference algorithm closely follows that used in the Visual Memex. The summation in Equation 7 can be seen as a weighted sum of kernels, one for each observation. The weight of each kernel, $S_{au}S_{cv}$, measures the similarity between the object pair under consideration (a, c) and the arbitrary object pair (u, v).

The parameters σ_0, σ_1, and σ_2 control how contextual information propagates between objects of different sizes. These have units of length and are chosen proportional to the length of the longest dimension of the objects being compared (l). We use $\sigma_0 = 2l$, $\sigma_1 = 0.2l$, and $\sigma_2 = 2l$. The small value of σ_1 places increased emphasis on finding objects that are close to the size of the query box, while the larger values of σ_0 and σ_2 permits objects of large size variation to contribute context neighborhood information. Increasing the value of σ_1 denotes increased uncertainty in the size of the user-specified query box; as this value approaches infinity the size of the query box becomes irrelevant.

Models are more highly ranked if they have the same size as the query box. We evaluate all models in sorted order, starting with the model whose size is closest to the query box size. At some point, the size kernel term will be essentially zero and it is likely not worthwhile to evaluate these models. The number of models that need to be evaluated depends on the value of σ_1, the density of objects around the query box size, and how many results are desired. For all the queries in this paper, we ranked all models in the database and found that the top 24 results were always in the first 2000 models, so we choose to restrict our search to this set. More models may need to be evaluated in databases with extremely dense size regions, if less emphasis is placed on the size of the query box, or if more than 24 results are desired.

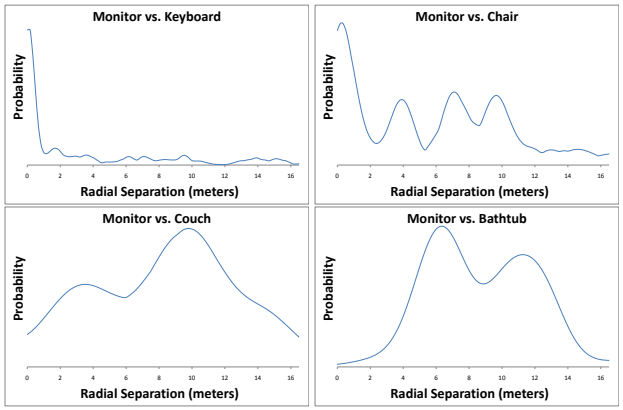


Figure 3: *Density estimation as a function of radial separation between objects. A representative object of each class was chosen, and Equation 6 was computed as a function of the radial separation between the objects. The shape of these curves approximates the expected relationship between these objects.*

Figure 3 provides a visualization of Equation 6. In this example, we plot the probability that a keyboard, chair, couch and bathtub are near a monitor as a function of the radial separation. Note that a keyboard has a high probability of being near a monitor, but is not likely to be far from the monitor. Examining the plot for a chair, we see there is interesting structure. First, chairs are further from monitors than keyboards, and can also be found at larger radii. Couches and bathtubs are much more common at large radii than close to monitors. This shows that our algorithm is able to learn some of the structure in the scenes.

We presented Equation 6 for the case of only one object in the scene. One natural assumption to make in the case of multiple support objects is to assume $p(a|b, f_{ab})$ is independent between all support objects b, in which case we would simply compute the product of this term for all supports (this is the assumption made by the Visual Memex.) We have found that this tends to unfairly penalizes objects that are mostly found in partial scenes, which may have many support objects where this probability score is effectively zero. Instead we compute $p(a|b, f_{ab})$ for all Q support objects in the scene, and store these in a list P for each object a, sorted from most to least probable. Our final model score is then taken as a product over the q strongest supporting objects in the scene (we use $q = 5$:)

$$p(a|b_1, ...b_K, f_{ab_1}, ...f_{ab_K}) \propto \prod_{i=1}^{q} P_i \quad (8)$$

To make queries in scenes with many small objects tractable, we use a simple heuristic that the most contextually informative objects are often either large (ex. furniture) or close to the query box. If d is the distance from a support object to the query box and r is the radius of the bounding sphere of the support object, we skip the computation if $r < 0.05d$.

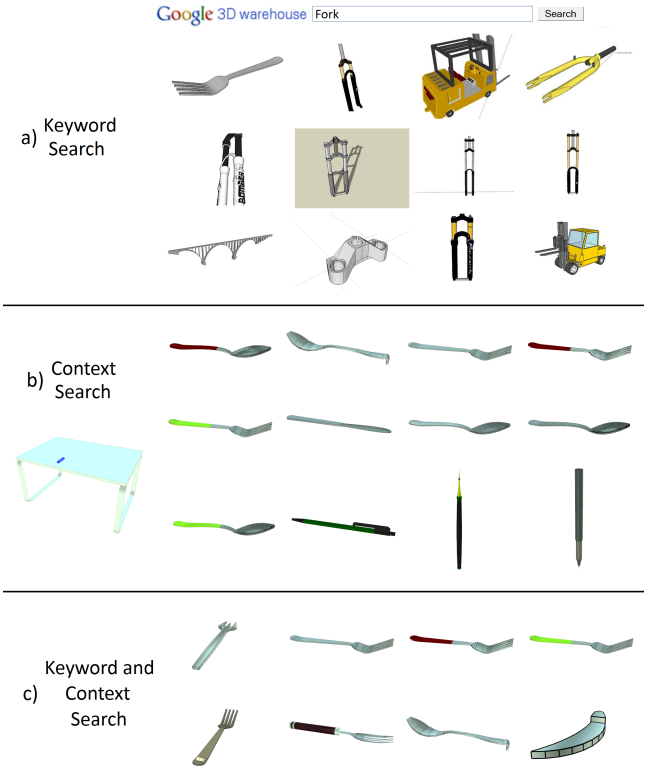


Figure 4: *Comparing keyword and context search. Top: Front page search results for "fork" in Google 3D Warehouse. Middle: Results of a context search in our database for a query box placed on a table. Bottom: Filtering the results of the context search for models with the "fork" tag. A search engine which can look at the target context and size can better discern the intent of the user's query, especially for keywords with multiple meanings.*

5 Results

5.1 Context Search vs. Keyword Search

One application of our algorithm is to suggest models to a user who wishes to add another object to a scene. For example, imagine a user who is modeling a dining room table and wishes to add a fork to it. One current approach is to use keyword search. To find a model, the user searches Google 3D warehouse using "fork" as a query term. The results returned are shown in Figure 4a. Using our system, the user searches for models using a context query. The query is generated by selecting a 3D bounding box on the dining room table. The context query returns the results in Figure 4b.

Figure 4a shows that a keyword search can have trouble returning a useful list of models. The keyword search was unable to determine what sense of the keyword "fork" was appropriate. In this case, the result set included only one table fork; the other eleven objects represent other senses of the term "fork" including "fork lift" and "bicycle fork." Although tuning forks are plausible, fork lifts cannot be placed on dining room tables. In contrast, the context search returned four table forks (Figure 4b). Since no keyword was given, small objects like pens and brushes were also returned. All the objects could be sensibly placed on a dining room table. Finally, Figure 4c show the results of doing a context plus keyword query search. In this example, the result set was all table forks, except for one extraneous model.

5.2 Adapting to Context

Figure 5 shows the results of a context query when the scene consists of a single object, in this case a desk. The top row shows the results for a query box in front of the desk. For this query, 21 of the top 24 results are chairs. The middle row shows the results for a tall box on the side of the desk. Here the context query returned 17 floor lamps. These two queries demonstrate the algorithm's ability to find different types of models using different contexts. The bottom row shows a query to find an object resting on the desk. This query can be satisfied by many different types of objects. Quantifying the relevance of these results is more challenging; nevertheless, there at least four distinct categories of results that are relevant. These include lamps, plants, goblets, and vases.

It is interesting to note that in all three example queries, the highest rated result (the one in the upper-left corner) would be a very plausible result for the given context. If the user requested a single result be immediately returned (by pressing "I'm feeling lucky"), they would not be disappointed.

5.3 Multiple Supporting Objects

We ran an experiment to test the effect of adding contextual objects to the scene. In this case, the user wanted to model a kitchen counter. The top row of Figure 6 shows the results of the query with only a sink as context. The query returns five sinks, and several other relevant models including a toaster oven, a mixer and a vase, and a few undesirable models such as a printer and at least five objects that are not meaningful. We now add a second object, a blender, to the scene, and reissue the query. The additional context significantly raises the rank of the three microwave models, and slightly increases the rank of the blender, toaster oven, and vase models. It also removes the printers and three meaningless objects. Overall the combination of the blender and sink noticeably improved the quality of the results.

5.4 User Evaluation

It is difficult to rigorously evaluate a context search algorithm because so much of it depends on both human perception and user intent. We made five test scenes, each with a single associated context query, and ran our algorithm to get 500 search results for each scene. The number of supporting objects in the scenes ranges from one to fifty. We created a candidate set of models for each scene by taking the union of our 500 search results with the 1000 objects in the database whose size is closest to that of the query box. Users were then presented with each model in the candidate set in a random order and asked to decide whether the model was relevant to the context query. Our user base consisted of five males and four females, none of whom had any formal design experience.

The results of this study were used to estimate the set of relevant models for each query: we assume that the models marked as belonging in the query box by at least half of the users are the only relevant models in the database. Note that we have not evaluated the complete set of relevant models because we have only presented users with a subset of models in the database. This may miss some relevant models and biases our results in favor of our algorithm.

Given our search results and a relevance set for each scene, we plot a precision-recall curve to evaluate the quality of our results. The solid curves in Figure 8 show this curve for each scene. These curves demonstrate that even without keywords, context-based search can successfully favor relevant models. In all five scenes the algorithm returned 50% of the relevant models with a

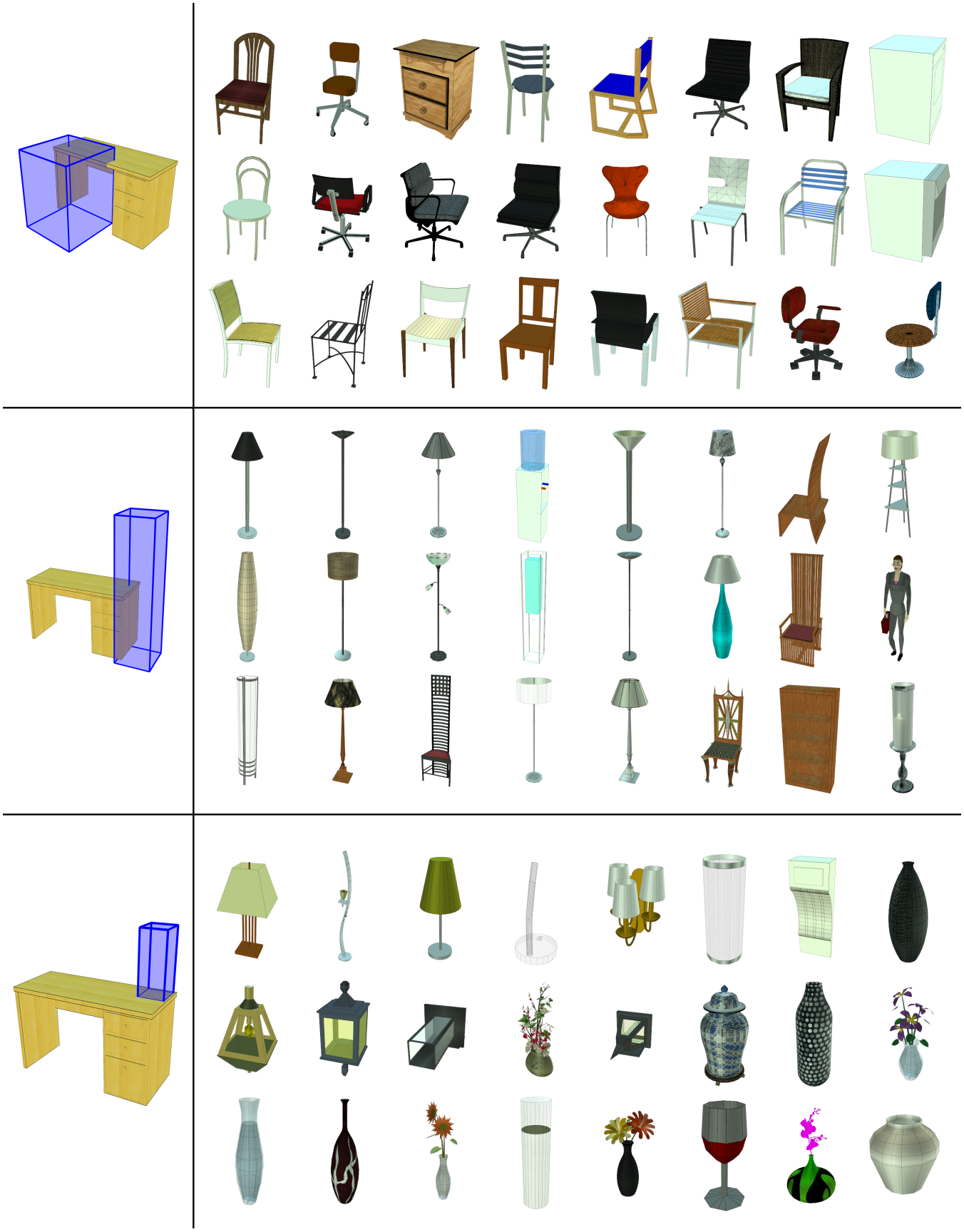


Figure 5: *Context query results with a desk as the only supporting object. Left: The user places a query box in the vicinity of the desk. Right: The top 24 search results for each query. When models with identical geometry but different textures occur, only the first result is shown.*

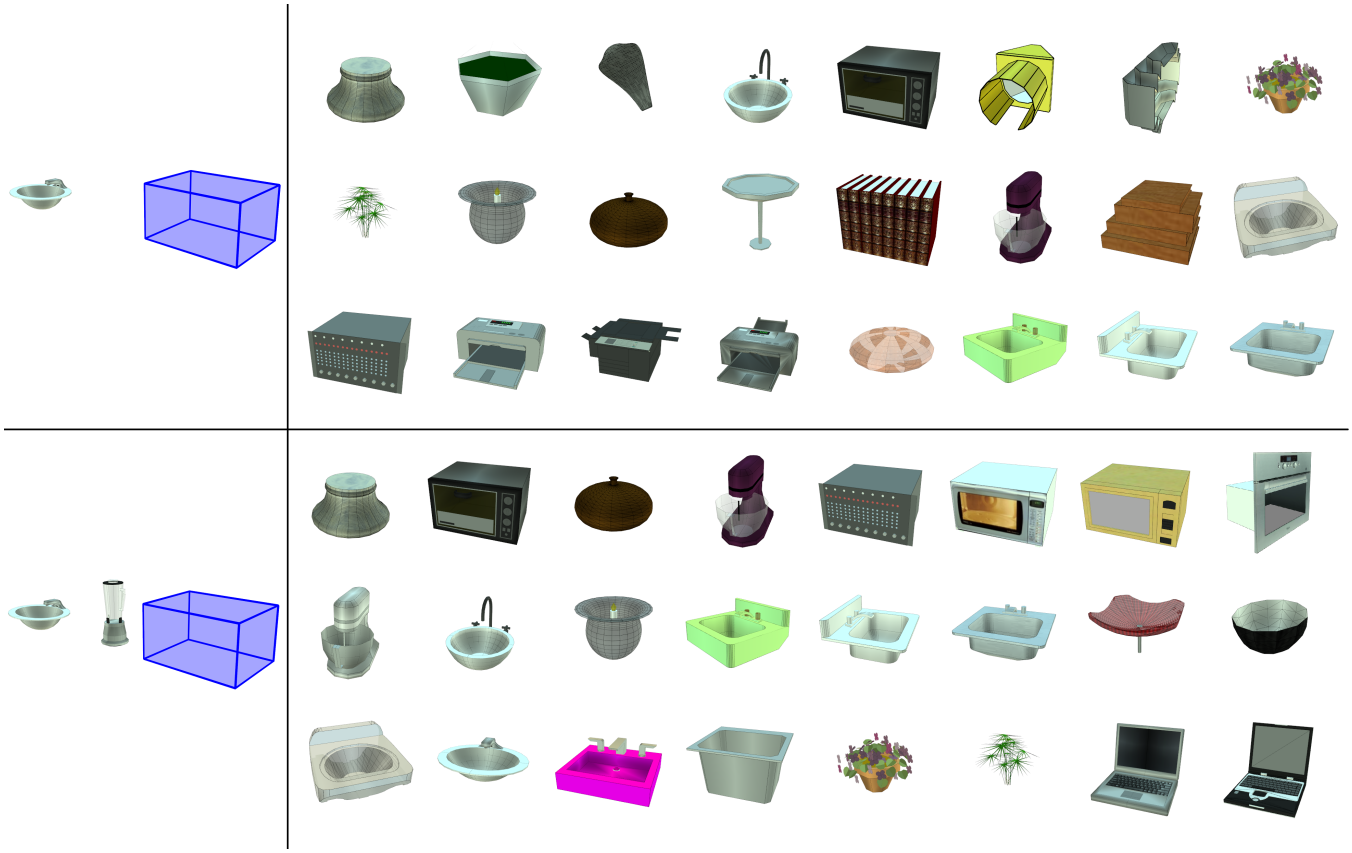


Figure 6: *Benefit of additional supporting objects. Top: The user places a sink into an empty scene and asks for an object four feet away. Bottom: The user places a blender in the scene between the sink and the target object, and repeats the same query.*

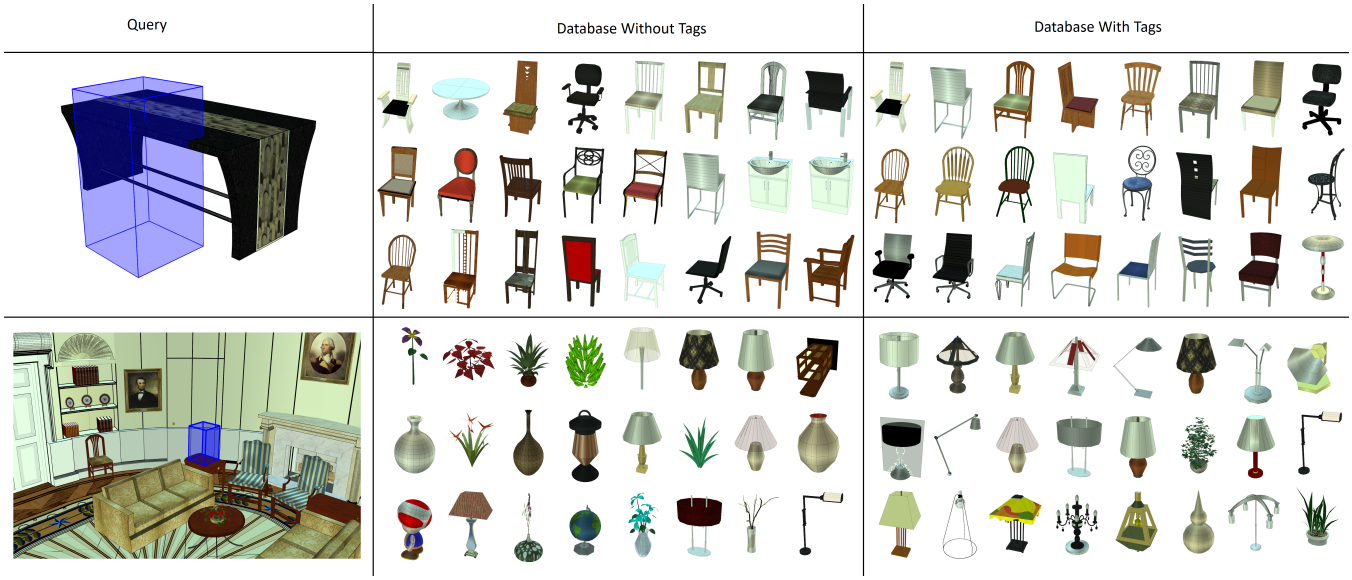


Figure 7: *Comparing results with and without database tags. Left: User performs two context queries; one for a chair in front of a dining room table and one for an object to go on a side table in a crowded scene. Middle: Results after discarding all tags in the database. Right: Results using the database with tags. The algorithm can still perform reasonable inference using only geometric relationships to determine model similarity.*

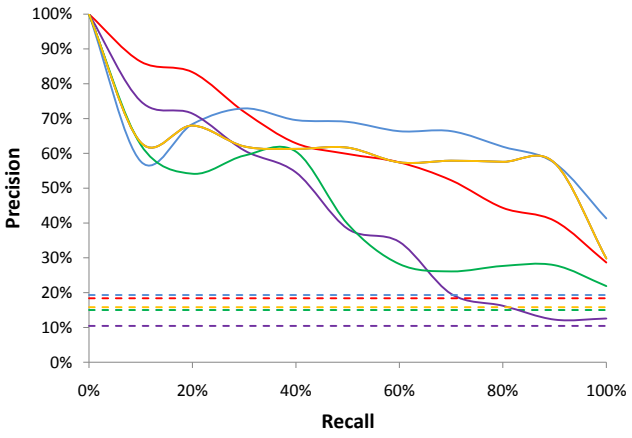


Figure 8: *The solid curves show the precision-recall curve for our algorithm on five test scenes. The relevance of each model was determined by surveying human subjects. The dashed lines show the expected precision-recall curve if the candidate model set had been presented to the user in a random order.*

precision of at least 35%, which implies that at least one in every three models was desirable.

To provide a comparison point for our search results, we computed the expected precision-recall curves after randomizing the models in each candidate set and using this as our final ranking. These are the dashed curves in Figure 8. All of these curves have a precision between 10% and 20%. Because the candidate sets select highly for the size of the objects, these results can be taken as an approximation of the performance of a size-only search engine.

The fact that our algorithm provides a noticeable improvement over a random ordering of the candidate set indicates that context provides some useful information. The actual information gained from context information compared against just the bounding box size will depend heavily on the scene and the density of objects in the database around the size of the query box. As a concrete example of the advantage of context information, a direct size-based ranking for the top query in Figure 5 would contain 8 toilets in the top 24 results.

5.5 Tags

A major feature of our algorithm is that we can take advantage of the semantic information contained in tags. To test the power of this feature, we created two databases, one with tags and one without tags. The one without tags was created from the same input database by removing all tagging information, and dropping terms involving tags from the ranking function. Figure 7 compares the results returned after searching the two databases. In the query for the dining room chair, both the tagged and untagged databases found chairs to be a dominant category. However, the results from the untagged database returned three clearly irrelevant results: a coffee table and two bathroom sinks. For the multi-object scene, the untagged database returned models such as the top half of a floor lamp, which is geometrically similar to a relevant category (desk lamps) but cannot be placed on the side table. We conclude that tagging can improve the search results, but that geometric information alone works quite well. This leads us to believe that our algorithm would work even on databases without tags.

The additional predictive ability of tags depends on the relative importance of geometric (proximity and size) vs. semantic information. Size and proximity are strong contextual cues. If the semantic information is not reliable, geometric information may work better alone. However, semantic information, if reliable, can add additional contextual cues. Tagging also facilitates the combination of spatial and keyword search.

5.6 Failure Cases

Although our algorithm does a reasonable job of returning relevant models, there are several failure modes that will cause an irrelevant model to be ranked highly. First, an object may be geometrically very similar to a relevant object but semantically very different. Likewise, an irrelevant object may have the same tag as a relevant object. For example, the model in Figure 6, 1st column, 3rd row, is tagged as "channel mixer". It is not a very relevant model to the query, but is both geometrically similar to a microwave model and shares a tag with the "mixer" model, both of which are relevant and appear in the top 24 results. Another failure mode can occur because our spatial relationships are overly simplistic. For example, in the top query in Figure 5, the three box-shaped objects are not very relevant at the selected location, but were ranked highly because they were found on the side of desks and our spatial relationships do not consider orientation.

5.7 Performance

We perform some simple tests to evaluate the performance of our implementation of the algorithm. The total time it takes to respond to a context query is roughly linear in both the number of supporting objects and the number of candidate objects that are considered. In all cases we use 2000 objects as our set of candidate objects. The task is very easy to distribute across multiple processors because each computation of Equation 6 is independent. These tests were all run using a multithreaded implementation of our algorithm on a quad-core 2.67GHz Intel Core i7 with hyperthreading enabled. The average query time for three tests scenes, each with a single support object was 0.094s. Using three test scenes each with thirty support objects, the average query time was 1.73s. We feel these results are sufficiently fast for interactive use, although different parameter choices (such as placing less emphasis on the size of the query box) will increase the computation time because more objects will need to be evaluated.

6 Discussion and Future Work

We have shown that context search can improve scene modeling tools. First, context search returns more relevant objects to the user than traditional keyword search. Although we do not have experimental data, we conjecture that it is just as easy to use. Placing a query box is quite easy, and would need to be done anyways to place the object in the scene. The user is also not required to think of good keywords, and only needs to provide them to refine the results. Second, we have observed that users already tend to construct scenes in a contextually relevant order; as Figure 5 demonstrates, a desk says quite a lot about the objects around it, and in practice, a user is likely to place the desk very early in the modeling process. To summarize, we believe forming context queries is a natural process and leads to more relevant search results.

The quality of the results could be improved by enhancing our methods for processing the scene database. One of the biggest improvements would be a better algorithm to segment the scene graph into meaningful objects. Likewise, improved tagging, either from crowdsourcing or autotagging methods, would facilitate learning semantic relationships.

We could improve the system further by extracting more meaning-

ful spatial relationships between objects. We currently consider only the radial separation between objects, and not their angle of separation, which would require defining a reference orientation for each object. Extracting relationships such as "resting," "hanging," "inside," or "in adjacent rooms" remains to be explored.

This work is quite similar to recent attempts to use spatial context in computer vision. If computer vision systems could reliably extract precise spatial relationships between objects, we could use that information to help place objects in synthetic scenes. An advantage of using photographs is that there are many more photographs than 3D scenes. Our methods for finding relationships between objects may also help computer vision systems by providing good priors about the spatial relationships of objects in the 3D world. Thus, methods from graphics and vision could be combined synergistically. We plan on making our dataset publicly available to help facilitate exploring the relationships between the 2D and 3D scene understanding problems.

In this paper we have used Google 3D Warehouse, which is a general model repository containing many different scenes with different styles. Virtual worlds, in contrast, usually have a strong artistic style from a particular place and time period. Our tools for preprocessing the virtual world dataset would presumably learn the particular style of that virtual world. This would make it easier for the artist to extend the world, while maintaining the existing style. We have done some preliminary work in extracting datasets from Second Life and World of Warcraft, and the results are very promising.

There are many other applications of context information beyond what we have shown in this paper. As a simple example, being able to choose a default orientation for a newly inserted object can save an artist time in placing the object. A more challenging problem is for the system to suggest objects to add without placing a query box. For example. after the user places a bed, they are very likely to want to place a pillow on the bed. A long range goal of scene modeling would be to intelligently perform complex actions such as "decorate this partially modeled dining room using the keywords turkey and thanksgiving" or "model this room after a photograph I took last thanksgiving." In this paper we have made a small step towards this goal.

Acknowledgments

Support for this research was provided by the Fannie and John Hertz Foundation. We would like to thank Ed Luong for writing the webcrawler that was used to acquire scenes from Google 3D Warehouse and the web interface used in the user study. We would also like to thank Google for allowing the images of 3D Warehouse models to be published.

References

CHEN, D., TIAN, X., SHEN, Y., AND OUHYOUNG, M. 2003. On visual similarity based 3D model retrieval. In *Computer graphics forum*, vol. 22, Amsterdam: North Holland, 1982-, 223–232.

FELLBAUM, C., ET AL. 1998. *WordNet: An electronic lexical database*. MIT press Cambridge, MA.

FUNKHOUSER, T., AND SHILANE, P. 2006. Partial matching of 3D shapes with priority-driven search. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, Eurographics Association, 142.

FUNKHOUSER, T., MIN, P., KAZHDAN, M., CHEN, J., HALDERMAN, A., DOBKIN, D., AND JACOBS, D. 2003. A search engine for 3D models. *ACM Transactions on Graphics 22*, 1, 83–105.

FUNKHOUSER, T., KAZHDAN, M., SHILANE, P., MIN, P., KIEFER, W., TAL, A., RUSINKIEWICZ, S., AND DOBKIN, D. 2004. Modeling by example. In *ACM SIGGRAPH*, ACM, 663.

GALLEGUILLOS, C., RABINOVICH, A., AND BELONGIE, S. 2008. Object categorization using co-occurrence, location and appearance. In *IEEE Conference on Computer Vision and Pattern Recognition, 2008. CVPR 2008*, 1–8.

GOLDFEDER, C., AND ALLEN, P. 2008. Autotagging to improve text search for 3d models. In *JCDL '08: Proceedings of the 8th ACM/IEEE-CS joint conference on Digital libraries*, ACM, New York, NY, USA, 355–358.

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

HE, X., ZEMEL, R., AND CARREIRA-PERPINÁN, M. 2004. Multiscale conditional random fields for image labeling. *CVPR*, 695–702.

MALISIEWICZ, T., AND EFROS, A. A. 2009. Beyond categories: The visual memex model for reasoning about object relationships. In *NIPS*.

MIN, P., KAZHDAN, M., AND FUNKHOUSER, T. 2004. A comparison of text and shape matching for retrieval of online 3D models. *Research and Advanced Technology for Digital Libraries*, 209–220.

NOVOTNI, M., AND KLEIN, R. 2003. 3D Zernike descriptors for content based shape retrieval. In *Solid modeling and applications*, ACM, 225.

PAPADAKIS, P., PRATIKAKIS, I., TRAFALIS, T., THEOHARIS, T., AND PERANTONIS, S. 2008. Relevance Feedback in Content-based 3D Object Retrieval: A Comparative Study. *Computer-Aided Design and Applications Journal 5*, 5.

RABINOVICH, A., VEDALDI, A., GALLEGUILLOS, C., WIEWIORA, E., AND BELONGIE, S. 2007. Objects in context. *IEEE 11th International Conference on Computer Vision*, 1–8.

RUSSELL, B., TORRALBA, A., MURPHY, K., AND FREEMAN, W. 2008. LabelMe: a database and web-based tool for image annotation. *International Journal of Computer Vision 77*, 1, 157–173.

SALTON, G., WONG, A., AND YANG, C. S. 1975. A vector space model for automatic indexing. *Commun. ACM 18*, 11, 613–620.

SHILANE, P., MIN, P., KAZHDAN, M., AND FUNKHOUSER, T. 2004. The princeton shape benchmark. In *Shape Modeling International*.

STRAT, T. M., AND FISCHLER, M. A. 1991. Context-based vision: Recognizing objects using information from both 2d and 3d imagery. *IEEE PAMI 13*, 1050–1065.

TORRALBA, A., 2010. The context challenge. `http://web.mit.edu/torralba/www/carsAndFacesInContext.html`.