Using Text N-Grams for Model Suggestions in 3D Scenes*

Laureen Lam† Stanford University
Sharon Lin† Stanford University
Pat Hanrahan† Stanford University

Abstract

Creating 3D scenes requires artistic skill and is time-consuming. A key challenge is finding novel models to place in a partial scene. We present a new algorithm to propose relevant models by leveraging text data. Our algorithm takes a partially completed 3D scene as input and a user-specified region of interest. It then suggests additional models according to the point-wise mutual information between the labels of nearby models in the scene and the labels of models in the database. We show that our text-based system suggests more models that result in model arrangements not observed in the training corpus, compared to a Graph Kernel system that trains on 3D scene data. Furthermore, combining the Graph Kernel system with our new system increases the number of unobserved model arrangements for the Graph Kernel, with higher precision according to human evaluators.

CR Categories: I.3.5 [Computing Methodologies]: Computer Graphics—Computational Geometry and Object Modeling I.2.7 [Artificial Intelligence]: NLP—Text Analysis;

Keywords: 3D model search, scene modeling

1 Introduction

Virtual game worlds need many realistic 3D models to create rich and immersive environments. However, modeling 3D scenes is difficult and time-consuming, partly because they contain many objects. Our goal is to make 3D scene modeling faster, easier, and more enjoyable. In particular, this paper tackles the problem of finding relevant and novel models to complete a partial scene.

Previous approaches to this problem often involve learning model relationships from examples of 3D scenes [Fisher et al. 2011; Fisher and Hanrahan 2010]. While these approaches are effective when a training corpus of 3D scenes is available, in practice such corpora are often small, domain-specific, and time-consuming to produce. Because of this, it is important for a suggestion system to be able to provide suggestions that result in model arrangements not previously seen in the training corpus.

In this paper, we are primarily interested in suggestions that result in previously unobserved model arrangements. A model arrangement is defined as an observed arrangement if any of its model pairs were seen within the same neighborhood in the training examples. For instance, a suggested monitor model for a desk scene is considered an observed arrangement if the same monitor model appeared with the same desk model in a training scene.

We present a system that leverages co-occurrence statistics in text to create suggestions. Some advantages of text are that it is plentiful, has broad coverage, and is widely available. Given enough text, the fact that two objects are mentioned within a few words of each other can indicate that those objects occur together spatially. We use point-wise mutual information, which is explained in Section 4. Specifically, we look at co-occurrences of model labels in an N-gram, an ordered sequence of N words or other tokens. We take N to be five, as this is the limit of our text corpus. This is not to be confused with the N-grams used in [Torralba et al. 2009], which come from annotated objects in scene data.

To evaluate our system, we compared it against a Graph Kernel system which learns from the labels, spatial relationships, and geometry of models in a training corpus of 3D scenes [Fisher et al. 2011]. We found that the text-based system presented a much higher percentage of models that result in previously unobserved model arrangements. Furthermore, combining our text-based system with the Graph Kernel system resulted in one that suggested more unobserved model arrangements with higher precision than the Graph Kernel system alone (69% vs. 62%), where precision is the percentage of suggested models that user study participants marked as sensibly belonging in a given partial scene.

2 Related Work

This work is influenced by [Fisher and Hanrahan 2010] and [Fisher et al. 2011], which suggest models to place in a new scene by learning scene and model relationships from a 3D scene corpus. In [Fisher and Hanrahan 2010], users specify a 3D bounding box in a partially constructed scene, and the algorithm performs context-based queries for relevant models. The algorithm learns spatial offset relationships between pairs of models from a corpus of 3D scene data to predict the strengths of relationships between candidate models and existing objects in the partial scene. It augments keyword search with this information, to return relevant models.

[Fisher et al. 2011] converts scenes into graphs (rather than pairwise relationships) that encode models and their spatial relationships semantically. A graph kernel then compares common structures in two scene graphs to determine the similarity between the scenes. This can be used in context-based model search and other tasks. Using spatial context to help determine objects in a scene has also been applied successfully in the area of computer vision [Strat and Fischler 1991; Galleguillos et al. 2008]. The text-based algorithm presented in this paper likewise uses a context-based approach, but is trained on text data rather than 3D scene data.

The WordsEye project demonstrated that text N-grams (in the form of phrases) can be automatically converted into scenes [Coyne and Sproat 2001]. However, in that project, the objects being rendered in a scene are explicitly specified in the input phrase, and thus the work does not suggest models.

3 Datasets

For our experiments, we use two main datasets. For our 3D scene data, we use the Stanford Scene Database [Fisher and Savva], since

†e-mail: {laureen, sharonl, hanrahan}@cs.stanford.edu

it provides models that are well segmented. This database contains approximately 140 artist-made scenes and 1,736 object models. We take 120 of these scenes (encompassing 1,580 of the available models) to be in our 3D scene training set. Each model also has labels (both names and tags) taken from Google 3D Warehouse. Since any user can label models in Google 3D Warehouse, the labels are not guaranteed to be accurate, spelled correctly, or even in English.[1] However, our system was robust to this noise.

For our text corpus, we use the Google Web 1T N-grams dataset [Brants and Franz 2006]. This is a cleaned dataset based on a crawl of the Internet in 2006, and contains approximately 1.2 billion 5-grams and their associated counts. Approximately 52% of the corpus contains at least one label from our 3D model database.

4 Algorithm

We describe the three algorithms compared in this paper: the text-only N-gram Analyzer, a modified Graph Kernel system informed by 3D scene data [Fisher et al. 2011], and a merged system.

4.1 N-gram Analyzer

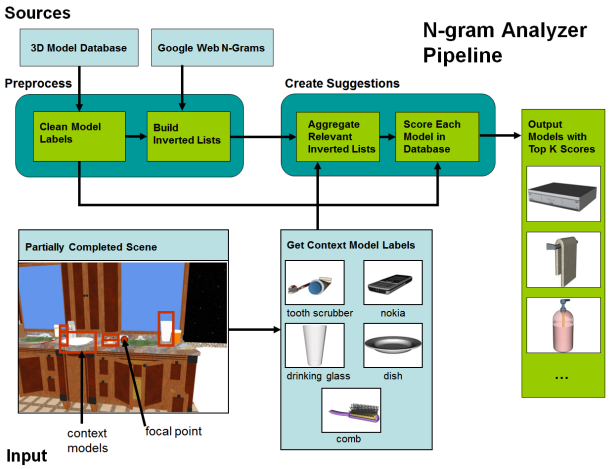

Figure 1: *The N-gram Analyzer pipeline.*

The N-gram Analyzer has two main stages. In the preprocessing stage, we clean the model labels from the 3D scene database: we convert all labels to lowercase to enforce consistency, and filter out any tag labels that are stopwords.[2] We do not filter out name labels that are stopwords, since some models may only contain stopwords as labels. There are a total of 1,620 name labels and 5,266 tag labels in the Stanford Scene Database. Filtering the tag labels removed 65 tags, leaving a total of 5,201 cleaned tag labels.

Next, we take all labels present in the Stanford Scene Database, and use the Web N-grams corpus to calculate the score between each pair of labels. We take the point-wise mutual information (PMI) between label pairs as the score, as this metric is well-established in NLP (most notably in collocation extraction [Bouma and Kuhn 2009]). PMI is defined to be:

$$\mathrm{PMI(A,B)} = \log\left(\frac{\Pr(\mathrm{AB})}{\Pr(\mathrm{A})\Pr(\mathrm{B})}\right). \quad (1)$$

[1]For instance, some name labels are: *m*, *p*, and *zapatos*, while some tag labels are: *, *refridgerator*, and *waschbecken*.

[2]Stopwords include those in the Lextek stoplist [Lextek] and custom stopwords: **and** $* - + . ( ) | ,$

For our purposes, Pr(AB) is the probability that word A and word B both occur within the same 5-gram.

After computing the scores between all label pairs, we store them in an inverted list, which allows the lookup of labels that have the highest score relative to the query label.

We can now move to the suggestion stage. Our first step is to convert the given 3D scene into text. We take the five closest models to the focal point where an object is to be placed, and store their labels to use as context for the user query. We then aggregate the inverted lists for all query labels by summing them up and normalizing the contributions from each context model. Figure 2 shows a visual example of creating an aggregate inverted list given two context models labeled *fork* and *spoon*.

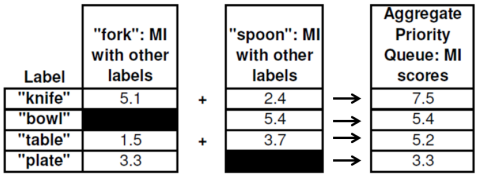

Figure 2: *A simple example showing how two inverted lists may be aggregated by using score summation.*

In this example, the *knife* label has the highest aggregate score of 7.5. Mathematically, the aggregate score of a label o is:

$$score(o) = \sum_{m \in M} score(o, m) \quad (2)$$

$$score(o,m) = \frac{w_n}{|N(m)|} \sum_{n \in N(m)} PMI(o,n) + \frac{w_t}{|T(m)|} \sum_{t \in T(m)} PMI(o,t) \quad (3)$$

where M is the set of context models, $N(m)$ is the set of name labels and $T(m)$ is the set of tag labels associated with context model m. The terms $|N(m)|$ and $|T(m)|$ in the denominators prevent models with a large number of similar labels from getting an unfair advantage over models that have only a few labels.

Once we have an aggregate inverted list of highest-scoring labels, we go through each of the 1,736 models in our database and calculate a model score for each. The candidate model score is defined as:

$$model\_score = \frac{w_n}{|N|} \sum_{n \in N} score(n) + \frac{w_t}{|T|} \sum_{t \in T} score(t) \quad (4)$$

where $score(l)$ is the score of label l in the aggregate list, N is the set of name labels, and T is the set of tag labels, associated with the model. We tune the weights w_n and w_t according to a separate development dataset.

One issue with the above algorithm is that some labels in the Stanford Scene Database are longer than five words, and, thus, they would never co-occur with other labels in our 5-gram corpus. We use a form of smoothing to take into account partial label co-occurrences. This results in a few changes to the algorithm:

In the preprocessing step, we break all labels in the model database into their component subgrams (unigrams up to 4-grams), and calculate PMI scores between all pairs of these labels (including subgrams).

Equation (3) is redefined to take into account subgram labels:

$$score(o,m) = \frac{w_n}{|N(m)|} \sum_{n \in N(m)} PMI_s(o,n) + \frac{w_t}{|T(m)|} \sum_{t \in T(m)} PMI_s(o,t) \quad (5)$$

$$PMI_s(o,n) = PMI(o,n) + \delta \sum_{s \in S(n)} PMI(o,s) \quad (6)$$

$$PMI_s(o,t) = PMI(o,t) + \delta \sum_{s \in S(t)} PMI(o,s) \quad (7)$$

where $S(n)$ are the subgrams of the name label n, and $S(t)$ are the subgrams of the tag label t.

Finally, in Equation (4), $score(n)$ and $score(t)$ are replaced by:

$$label\_score(l) = score(l) + \sum_{s \in S(l)} score(s) * \alpha^{|l|-|s|} \quad (8)$$

where $|l|$ and $|s|$ are the total number of tokens in the label l and subgram s respectively. The value of α scales up contributions from higher-order subgrams. Both δ and α were tuned between 0.0 and 1.0 according to a separate development dataset (described in Section 4.3).

4.2 Graph Kernel System

We implement the Graph Kernel system described in [Fisher et al. 2011]. In this system, a model and its neighborhood are represented as a graph where nodes are models and edges are surface contacts. Model similarity is computed by comparing all walks in the graph rooted at model A with all walks in the graph rooted at model B and summing the scores. Two walks are compared to each other by using a node kernel that compares each node in the walk and an edge kernel that compares each edge in the walk. More details on this algorithm can be found in [Fisher et al. 2011].

The length of the walk along the relationship graph is set to three, which is sufficient for scenes in our dataset. Longer walks would only be needed for tall stacks of models. Given a query point in a scene, the system creates suggestions by going through each scene in the database and ranking models according to the similarity between the query point neighborhood and the model neighborhood.

4.3 Merged System

To combine the N-gram Analyzer system with the Graph Kernel system, we simply merge the model suggestions lists from each system of interest. The Graph Kernel system generally calculates scores between 0.0 and 1.0, normalized according to [Fisher et al. 2011], and the N-gram Analyzer returns scores between negative infinity and infinity. We normalize the Graph Kernel score to be between 0.0 and 1.0 if the suggestions list for a query contains scores over 1.0, and do the same for the N-gram Analyzer. We discard suggestions from either list that have a score of 0.0 or lower. Finally, we weight the scores of each suggestion according to which analyzer they came from, and add the suggestion and score to a merged list.

For our experiments, we tuned the weight value using a separate development dataset containing six diverse scenes from the Stanford Scene Database with two regions of interest per scene. For this dataset, we had two people mark all of the models in the database that were good suggestions for each region of interest. Their answers were unioned to get a final dataset of "good" model suggestions for each region of interest. Our goal with the tuning was to maximize the "goodness" of the overall suggestion set while trying to introduce more suggestions resulting in unobserved model arrangements.

5 Experiments & Results

For our experiments, we selected 9 regions of interest from a variety of scenes, including a bathroom scene, a bedroom scene, a living room scene, and an office scene. We consider the top 10 suggestions from each of the three analyzers for each of the scenes

Our first question is whether the N-gram Analyzer suggests more unobserved model arrangements, compared to the Graph Kernel system. We considered a model arrangment observed if any of its model pairs were seen within a walk length of 3 in the training examples. Because the Graph Kernel is more limited to model combinations seen in its training examples, we believed that combining it with the N-gram Analyzer would result in more unobserved model arrangements. We looked at the top 10 suggestions for each of our 9 scenes and counted the number of suggestions resulting in unobserved model arrangements.

With this number established, the next question is, are these suggested model arrangements reasonable? To evaluate the number of suggestions people considered reasonable, we conducted a user study (N=16) that compared the suggestions from the N-gram Analyzer, Graph Kernel, and the Merged system from these scenes. Users were shown a screenshot of the region of interest and one suggested model out of the pool at a time in a random order. Users were asked whether or not the suggested model *made sense* as either a replacement model for some other model in the visible part of the scene, or as a new addition to the scene.

5.1 Number of Unobserved Arrangements

The text-based N-gram Analyzer and the Graph Kernel systems often gave different suggestions. The overlap in the top 10 suggestions for all scenes was just 1.1%. Figure 3 shows the top 10 suggestions for a desk scene, with the unobserved arrangements highlighted. In this example, the N-gram Analyzer suggested many more models that result in unobserved arrangements than the Graph Kernel system, which suggested models already seen on the same desk in its training examples.

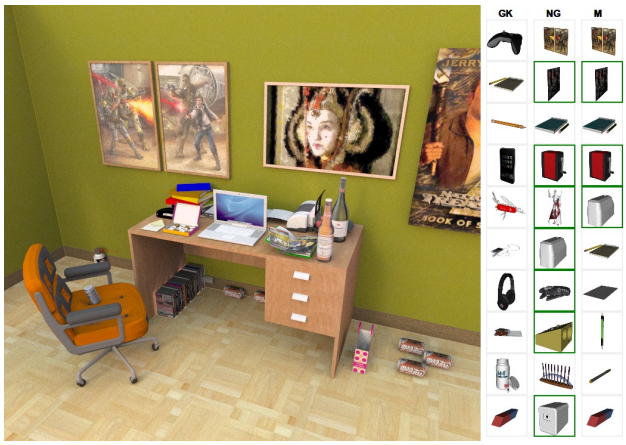

Figure 3: *Top 10 suggestions for a desk scene. Highlighted suggestions are ones that result in unobserved model arrangements.*

Table 1 shows the percentage of suggestions that resulted in unobserved model arrangements across all 9 scenes. Overall, the N-gram Analyzer and the Merged system suggested more unobserved model arrangements compared to the Graph Kernel (82.2% and 68.6% vs 45.6%, respectively).

Table 1: *Percentage of suggestions from each system that result in unobserved model arrangements.*

N-grams	Merged	GK
82.2%	68.6%	45.6%

5.2 User Study Results

We conducted a repeated-measures two-way ANOVA to study the effects of the system and whether the suggestion resulted in an unobserved arrangement on precision. The analysis indicated that the N-gram Analyzer may perform differently relative to the other systems ($p < 0.01$) depending on if the suggestion results in an unobserved or observed arrangement.

Table 2 shows the precision as percentage of suggestions "making sense" for each system. The N-gram Analyzer and the Graph Kernel had comparable precision when it came to suggesting models resulting in unobserved arrangements. The Merged system also had slightly better precision on these models than the Graph Kernel system (p = 0.02 for paired, one-tailed t-tests). Overall, the suggestions from the Merged system are rated comparably with respect to reasonableness to the suggestions from the Graph Kernel. Thus, not only does the Merged system suggest more unobserved model arrangements, but it also maintains the overall "goodness" of the suggestion set.

Table 2: *Precisions as "making sense" for each system for suggestions that result in unobserved and observed model arrangements. Bolded maximum precisions are significant at $p < 0.01$.*

System	Unobserved	Observed	All
NG	66.04%	70.70%	66.88%
GK	62.04%	**80.48%**	**72.08%**
M	**68.64%**	**83.87%**	**73.89%**

5.3 Discussion

The Graph Kernel can only suggest a model that is contained within a training scene. In the Stanford Scene Database, 90% of the models are contained within the 120 training scenes. However, there are often cases where there is a large database of models but only a few scenes, and so, the Graph Kernel only has a small number of models it can suggest. In these cases, the N-gram Analyzer can be particularly helpful in adding diverse suggestions.

Because the N-gram Analyzer uses PMI statistics from a large text corpus, it is not tied to a smaller corpus of 3D scenes. Most of the suggestions from the N-gram Analyzer were unobserved arrangements in the training corpus used by the Graph Kernel. In addition, these unobserved arrangements were rated as reasonable as the ones from the Graph Kernel in our user study. Combining the N-gram Analyzer with the Graph Kernel resulted in more unobserved arrangements with a slightly higher precision.

A possible reason for this comparable, but slightly higher, precision is that the Merged system has more reasonable suggested models to choose from. A suggestion that is scored highly by the Graph Kernel means that the system could find a similar model neighborhood in its training scenes, while a suggestion that is scored lowly means that the neighborhoods in the training scenes were not very similar. These low-scoring suggestions from the Graph Kernel are then replaced by suggestions from the N-gram Analyzer.

6 Conclusion & Future Work

We have shown a method to facilitate the modeling of rich 3D scenes by suggesting additional models that plausibly fit in the scene. Previous approaches learn good suggestions from a relatively limited corpus of 3D scenes. Our approach combines data from a 3D scene corpus with text data from the Internet, which is abundant and publicly available. We show that the N-gram Analyzer suggests a much higher percentage of models resulting in unobserved arrangements than a Graph Kernel system trained on 3D scene examples (82% vs. 46%). Additionally, we find that combining the N-gram Analyzer with a Graph Kernel system via a simple merge of the suggestions lists from each gives better results for novel models than the Graph Kernel system by itself (p = 0.002) and adds more unobserved model arrangments than the Graph Kernel alone.

For future work, we plan to experiment with more metrics to use with the N-gram Analyzer. There are also alternative methods of using N-grams to determine which models belong with other models in a scene, such as using distributional similarity or relation extraction. There is also the possibility of combining the N-gram Analyzer and Graph Kernel system in different ways, rather than just a simple weighted merge of output suggestion lists.

7 Acknowledgements

This work was supported by NSF grant CCF-1111943, and the Intel Science and Technology Center on Visual Computing. We would also like to thank Jay Ponte and Gabor Angeli for helpful discussions.

References

BOUMA, G., AND KUHN, J. 2009. Normalized (pointwise) mutual information in collocation extraction. GSCL 2009, 31–40.

BRANTS, T., AND FRANZ, A., 2006. Web 1t 5-gram version 1.

COYNE, B., AND SPROAT, R. 2001. Wordseye: an automatic text-to-scene conversion system. SIGGRAPH 2001, 487–496.

FISHER, M., AND HANRAHAN, P. 2010. Context-based search for 3d models. SIGGRAPH Asia 2010, 182:1–182:10.

FISHER, M., AND SAVVA, M. Stanford scene database. Available from http://code.google.com/p/stanford-scene-database/.

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

GALLEGUILLOS, C., RABINOVICH, A., AND BELONGIE, S. 2008. Object categorization using co-occurrence, location and appearance. In *CVPR 2008*, 1–8.

LEXTEK. Onix text retrieval toolkit: Stop word list 1. Available from http://www.lextek.com/manuals/onix/stopwords1.html.

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

TORRALBA, A., RUSSELL, B. C., AND YUEN, J. 2009. Labelme: Online image annotation and applications. Tech. rep.