Characterizing Structural Relationships in Scenes Using Graph Kernels

Matthew Fisher* Manolis Savva† Pat Hanrahan‡

Abstract

Modeling virtual environments is a time consuming and expensive task that is becoming increasingly popular for both professional and casual artists. The model density and complexity of the scenes representing these virtual environments is rising rapidly. This trend suggests that data-mining a 3D scene corpus to facilitate collaborative content creation could be a very powerful tool enabling more efficient scene design. In this paper, we show how to represent scenes as graphs that encode models and their semantic relationships. We then define a kernel between these relationship graphs that compares common virtual substructures in two graphs and captures the similarity between their corresponding scenes. We apply this framework to several scene modeling problems, such as finding similar scenes, relevance feedback, and context-based model search. We show that incorporating structural relationships allows our method to provide a more relevant set of results when compared against previous approaches to model context search.

Keywords: 3D model search, scene modeling, graph kernel, structural relationships

1 Introduction

A growing demand for massive virtual environments combined with increasingly powerful tools for modeling and visualizing shapes has made a large number of 3D models available. These models have been aggregated into online databases that other artists can use to build up scenes composed of many models. Numerous methods for querying a model database based on properties such as shape and keywords have been proposed, the majority of which are focused on searching for isolated objects.

When a scene modeler searches for a new model, an implicit part of that search is a need to find objects that fit well within their scene. Understanding which models best fit into a scene requires developing a way to compare the relevant parts of the supporting scene against scenes already in the database. The focus of this work is on representing scenes in a way that captures structural relationships between objects, such as coplanar contact or enclosure, and can enable this type of comparison.

Scene comparison is a very challenging problem because scenes contain important structure at many different resolutions. The challenge of comparing highly structured data occurs in a wide variety of fields, such as web search, protein function prediction, and image classification. In all of these problems, attempting to directly compare the finest-level data is rarely successful. Instead, the data is often transformed into a new representation that enables the comparison of important features. In this work, we will show how to transform scenes into a relationship graph whose nodes represent semantically meaningful objects or collections of objects, and whose edges represent different types of relationships between nodes. This graph representation greatly facilitates comparing scenes and parts of scenes.

One of the simplest approaches to scene comparison is to directly compare the tags artists have provided for a scene or the name attached to the scene. Unfortunately, while a scene name can provide useful information about the scene's category, it cannot easily express the stylistic variation within these categories. Likewise, it is challenging for the scene tags to encompass all the interesting substructures within the scene. In Figure 1, we show nine scenes retrieved from Google 3D Warehouse using a keyword search for "living room". A user issuing this query because they were looking for models to add to an entertainment center would only be pleased with three of these scenes — and doing a keyword search for "entertainment center" will miss all of these scenes entirely. These problems all demonstrate the need for a more effective way to characterize and compare the substructure of scenes.

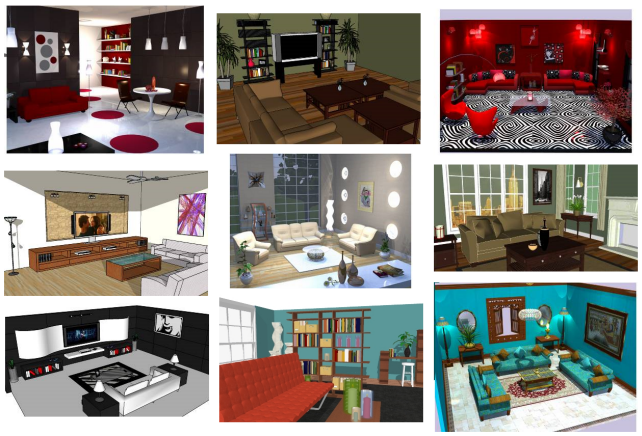

Figure 1: *A set of scenes in the Google 3D Warehouse with "living room" in their scene name. Many properties of a scene are not reflected well in the scene name. Understanding the relationships between scenes requires a method to compare different aspects of the scene's internal structure. All images in this paper are used with permission from Google 3D Warehouse.*

In this work we will describe how we can take a 3D scene and extract a set of spatial relationships between objects in the scene. We show how we can use this set of spatial relationships to define a positive-definite kernel between any two 3D scenes. We use this kernel to execute several different types of queries for complete scenes that incorporate the semantic relationships between objects. We show how our scene kernel can also be used to search for models that belong in a particular context and have a specified spatial relationship to other objects. For example, a user could issue a search for models that can be hung on a wall in the bedroom they are modeling.

2 Background

2.1 3D Model Search

Many techniques have been developed for querying 3D model databases [Funkhouser et al. 2003; Chen et al. 2003]. Two common approaches are to use a keyword search, or for the user to ask for models that are similar to a 3D model they have already found [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

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

models [Tangelder and Veltkamp 2008]. Although these methods are not designed for comparing entire scenes, determining the similarity between two isolated models is an important subroutine for a scene comparison algorithm.

Another approach to 3D model search uses a partially modeled scene as context [Fisher and Hanrahan 2010]. In this work, the user places a query box in the scene and the algorithm searches for models that belong at this location. Models in the database are ranked by the similarity of their spatial offsets to objects in the supporting scene. Only pairwise relationships between models are considered. Because this method uses only geometric relationships between objects, it is very sensitive to the relative position of objects in scenes. One goal of our work is to develop context-based search queries that incorporate the semantic relationships between objects and can reflect complex scene properties not well captured by pairwise geometric comparisons between objects.

2.2 Scene Comparison

A small number of attempts have been made at comparing 3D scenes. One approach is to partition the mesh of each object into a set of regions, connect adjacent regions, and then use the spectral decomposition of the resulting graph Laplacian matrix to compare scenes [Paraboschi et al. 2007]. This method is sensitive to the mesh segmentation and was not tested on scenes with a large number of objects. Also, because their focus was on manifold meshes it does not map well to the datasets we explore. Nevertheless, it is similar to our approach in that it first reduces scenes to a graph and then compares two scenes using properties of their graphs.

A problem that has many parallels to scene comparison is image comparison, where the goal is to relate two images based on their semantic content. One approach to image comparison is to first segment the image into regions and then construct a graph by connecting regions that are touching [Harchaoui and Bach 2007]. Two regions are compared by using their color histograms. The images can then be compared by looking at their respective segmentation graphs. Our approach can be seen as the natural extension of this idea to 3D scenes: we first segment our scene into meaningful objects, then insert edges that represent relationships between pairs of objects.

2.3 Graph Kernel

Kernel-based methods for machine learning have proven highly effective because of their wide generality. Once a kernel is defined between the entities under consideration, a wide range of learning algorithms such as support vector machines can be applied [Cristianini and Shawe-Taylor 2000]. In addition, techniques such as multiple kernel learning can be used to intelligently combine results from a large number of kernels [Bach et al. 2004].

There is considerable work on defining kernels between data types that are highly structured [Shawe-Taylor and Cristianini 2004]. In particular, several different kernels between two graphs have been proposed [Kashima et al. 2004]. These have been successfully applied to a variety of problems such as molecule classification [Mahé et al. 2004] and image classification [Harchaoui and Bach 2007].

In its most general form, a graph kernel takes as input two graphs with labeled nodes and edges, a kernel $k_{\textbf{node}}(n_a, n_b)$ defined between node labels and a kernel $k_{\textbf{edge}}(e_a, e_b)$ defined between edge labels, and returns a non-negative real number reflecting the similarity between the two graphs. The node and edge kernels used depend on the types of labeling used and are application specific. In molecular comparisons, they are often simple kernels that use exact matching between atoms and bond types. In image classification, one node kernel that has been used is to compare the color histograms of two image segments [Harchaoui and Bach 2007]. In our work, nodes represent individual models or collections of models, and we can use any of the model comparison techniques used in 3D model database search. Our edges represent different types of relationships between models, and the kernel used depends on the types of relationships.

containment	encircled	circlement
contact	surface contact	support
attachment	adhesion	hanging
piercing	impaled	proximity
above	below	vertical equality
horizontal support	front	behind
viewport equality		

Table 1: *A list of spatial primitives used to study how humans reason about spatial relationships.*

Given node and edge kernels, constructing an efficient kernel over graphs is a challenging problem with a diverse set of existing approaches [Gartner et al. 2003]. One method that has proven very successful is to first reduce the graph to a finite set of strings, and then use kernels defined over these strings. In particular, a graph walk kernel can be defined by considering all walks of a fixed length as the set of strings. As we will see, this admits a simple and efficient dynamic programming solution. Another mapping from graphs to sets of strings is to consider all possible α-ary tree walks in the graph of a fixed depth [Harchaoui and Bach 2007]. Unfortunately, we found tree walks to be intractable for our problem because there is not a natural way of ordering the edges around a node; successful applications for this type of graph kernel have relied on properties like graph planarity to obtain such an ordering. It has been shown that as long as the underlying node and edge kernels are positive semi-definite, the resulting walk and tree walk graph kernels will also be positive semi-definite [Shawe-Taylor and Cristianini 2004].

2.4 Spatial Relationships

In order to represent a scene as a graph, we need a way to take the geometric representation of the scene and produce a set of relationships between pairs of objects. These relationships might be largely geometric in nature ("object A is horizontally displaced by two meters relative to object B") or largely semantic ("object A is in front of object B"). Capturing semantic relationships is desirable because they are are more stable in the presence of model and scene variation.

Computer vision has used the spatial relationships between two objects in a photograph to assist with problems such as scene content understanding and object categorization. For example, many objects and materials are difficult to tell apart (sky vs. water) but can be disambiguated using spatial relationships (sky is rarely found below grass). One approach uses a conditional random field to maximize the affinity between object labels using semantic relationships [Galleguillos et al. 2008]. The relationships they consider are {inside, below, around, above}. Although these are useful examples of relationships between objects or materials in 2D images, they are not representative of semantic relationships between 3D shapes.

Psychologists have tried to understand what set of spatial primitives humans use to reason about spatial concepts [Xu and Kemp 2010]. Although the nuances of human spatial understanding are too complicated to construct a comprehensive list of all possible

primitives, in Table 1 we show one list of spatial primitives that has been used with some success [Feist 2000]. Our goal is to test for these relationships given only the geometry of two objects. Many of these relationships are highly geometric in nature (circlement, above, containment), but some are very difficult to infer from geometry alone (attachment vs. adhesion).

3 Representing Scenes As Graphs

Our algorithm takes as input a set of scenes represented as scene graphs. We start by constructing a corresponding relationship graph for each scene. The nodes of a relationship graph represent all models in the scene and the edges represent the relationships between these models. For now, we will assume that our scene graphs are good segmentations of the scene and that all nodes in the input scene graphs correspond to either transform nodes or meaningful objects. This means that, for example, we assume nodes are not over-segmented down to the level of individual triangles. This property is not true in general and real scene graphs will need additional processing to obtain a set of corresponding meaningful objects — see Section 6. Each non-transform node in the processed scene graph corresponds directly to a node in our relationship graph. Given the nodes of the relationship graph, we then determine the set of relationships between the nodes, thus creating an edge set for the relationship graph.

Good relationships should capture features of the scene that correspond with spatial primitives used by humans. We have used a subset of the relationships in Table 1 as our set of possible relationships. We chose relationships that were highly discriminative and could also be determined using only geometric tests.

We define a polygon in mesh A to be a contact polygon with respect to mesh B if there is a polygon in mesh B such that the distance between the two polygons is less than a small epsilon, and the angle between the unoriented face normals of the polygons is less than one degree. For the databases we use, we also have a well defined gravity vector that describes the global orientation of the scene.

Below is a list of the relationships we chose and the process used to test for them:

- **Enclosure:** Mesh A is enclosed inside mesh B if 95% of the volume of mesh A's bounding box is inside the bounding box of mesh B.
- **Horizontal Support:** Mesh A is horizontally supporting mesh B if there exists a contact polygon in mesh A whose face normal is within one degree of the gravity vector.
- **Vertical Contact:** Mesh A is in vertical contact with mesh B if there exists a contact polygon in mesh A whose face normal is within one degree of being perpendicular to the gravity vector.
- **Oblique Contact:** Mesh A is in oblique contact with mesh B if there exists a contact polygon that does not satisfy any other test.

The tests are performed in the order given and an edge of the corresponding type is created between the models with meshes A and B for the first test that is satisfied. In addition to the above relationships we also retain the original parent-child edges from the scene graph as a separate type of relationship. Figure 2 is a simple example illustrating the resulting relationship graph. Model nodes can be connected by both contact and scene graph inheritance relationships. Note how the monitor and keyboard nodes are both scene graph children of the computer node.

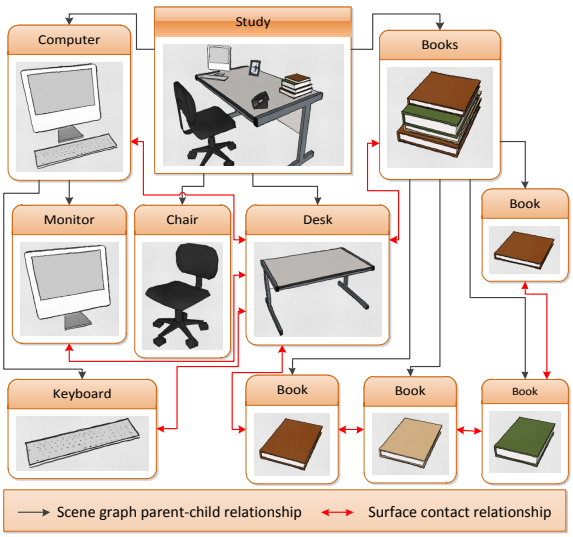

Figure 2: *A scene and its representation as a relationship graph. Two types of relationships are indicated by arrows between the model nodes.*

4 Graph Comparison

By constructing a node and edge set for each input scene we obtain its representation as a relationship graph. Our goal is to compare two relationship graphs or subparts of relationship graphs. To accomplish this comparison we first need a way to compare individual nodes and edges in these graphs. These will be key subcomponents in our graph kernel.

4.1 Node Kernel

The nodes of a relationship graph represent models within the scene. Each node contains a number of features that relate to the identity and semantic classification of a particular object. These properties include the size of the object and the geometry, tags, and texture of the underlying model. In this paper we adapt an existing approach to model comparison which uses a combination of tag and geometry descriptors to construct a kernel between two models [Fisher and Hanrahan 2010]. Each function is defined such that it indicates similarity of the model pair with respect to the chosen metric and is itself a positive semi-definite kernel. We have constructed all our kernels to be bounded between 0 (no similarity) and 1 (identical).

Model Identity Kernel: A simple Kronecker delta kernel which returns whether the two models being compared are identical. We take two models to be identical if they have nearly identical geometry and texture — see Section 6 for specifics. We represent this identity kernel as δ_{rs} for two arbitrary models r and s.

Model Tag Kernel: Each model may be tagged with a primary and secondary tag. A simple example is a desk lamp which would have a primary tag of "lamp" and a secondary tag of "desk". In a given model comparison, if both primary and secondary tags match we return a value of 1. For matches only in primary tags and only in secondary tags we return values of 0.5 and 0.1 respectively. If the models do not share any tags, we return a value of 0. We let $k_{\text{tag}}(r, s)$ represent this kernel.

Model Geometry Kernel: We use 3D Zernike descriptors to compare the geometry of two models [Novotni and Klein 2003]. In order to compute a Zernike descriptor for a model we closely follow

previous work using Google 3D Warehouse model data [Goldfeder and Allen 2008; Fisher and Hanrahan 2010]. The geometry is normalized to a unit cube, voxelized onto a 128^3 grid and thickened by 4 voxels. This grid is used to compute a 121-dimensional Zernike descriptor. The distance between two models is then normalized by estimating the local model density. Let d_{rs} denote the Zernike descriptor distance between models r and s, and $g_i(n)$ the distance to the n^{th} closest model from model i. The model geometry kernel is given as:

$$k_{\textbf{geo}}(r,s) = e^{-\left(\frac{2d_{rs}}{\min(g_r(n),g_s(n))}\right)^2} \quad (1)$$

The minimum value of $g_i(n)$ is chosen as the normalization term and we set $n = 100$ for all of our results.

Final Model Kernel: We combine the above kernels into a final node kernel, using the same weights as in Fisher and Hanrahan [2010]:

$$k_{\textbf{node}}(r,s) = \sigma(r)\sigma(s)\left(0.1\delta_{rs} + 0.6k_{\textbf{tag}}(r,s) + 0.3k_{\textbf{geo}}(r,s)\right) \quad (2)$$

The $\sigma(r)$ and $\sigma(s)$ terms are node frequency normalization scalars whose computation is described in Section 4.4. The result of this kernel evaluation is clamped to 0 if it is less than a small epsilon ($\epsilon = 10^{-6}$). This model kernel can be precomputed between all possible models in the database.

4.2 Edge Kernel

We now define an edge kernel to provide a similarity metric between edges representing relationships. In our implementation we choose to represent each relationship as a different edge with a simple string label indicating the type. The kernel between two edges e and f with types indicated by labels t_e and t_f respectively, is then simply $k_{\textbf{edge}}(e,f) = \delta_{t_e t_f}$.

4.3 Graph Kernel

Given node and edge kernels we now define a graph kernel to perform the comparison between two scenes. Our approach is heavily based on a graph kernel algorithm used for image classification [Harchaoui and Bach 2007].

A walk of length p on a graph is an ordered set of p nodes on the graph along with a set of $p-1$ edges that connect this node set together. Unlike a path, nodes in a walk may be repeated.

Let $W_G^p(r)$ be the set of all walks of length p starting at node r in a graph G. As defined earlier, $k_{\textbf{node}}(r,s)$ and $k_{\textbf{edge}}(e,f)$ represent the node and edge kernels. Considering nodes r and s in relationship graphs G_a and G_b respectively we now define the *p-th order rooted-walk graph kernel* k_R^p:

$$k_R^p(G_a,G_b,r,s) = \sum_{\substack{(r_1,e_1,\ldots,e_{p-1},r_p)\in W_{G_a}^p(r)\\(s_1,f_1,\ldots,f_{p-1},s_p)\in W_{G_b}^p(s)}} k_{\textbf{node}}(r_p,s_p)\prod_{i=1}^{p-1} k_{\textbf{node}}(r_i,s_i)k_{\textbf{edge}}(e_i,f_i)$$

This kernel is comparing nodes r and s by comparing all walks of length p whose first node is r against all walks of length p whose first node is s. The similarity between two walks is evaluated by directly comparing the nodes and edges that compose each walk using the provided kernels for these object types. In Figure 3, we visualize one step of the computation of k_R^2 for two nightstand scenes.

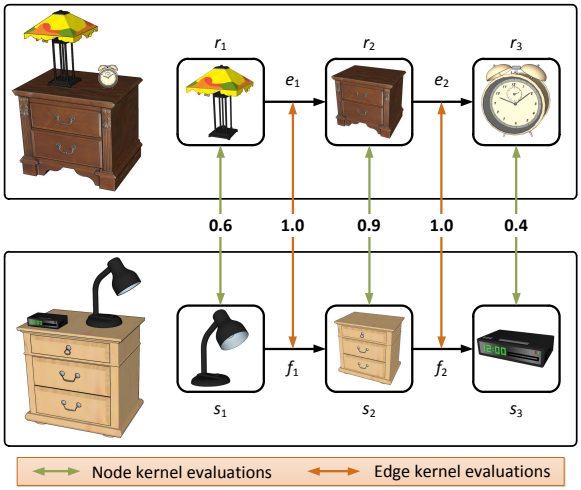

Figure 3: *Comparison of two walks. Left: The two scenes being compared. Right: Two walks in each scene, both rooted at the lamp node. The two walks are compared by taking the product of kernel evaluations for their constituent nodes and edges. The similarity between these two walks is* $0.6 * 1 * 0.9 * 1 * 0.4 = 0.22$.

If we also define $N_G(x)$ to be the set of all neighboring nodes of x in the graph G we can formulate a recursive computation for $k_R^p(G_a,G_b,r,s)$:

$$k_R^p(G_a,G_b,r,s) = k_{\textbf{node}}(r,s)\times \sum_{\substack{r'\in N_{G_a}(r)\\s'\in N_{G_b}(s)}} k_{\textbf{edge}}(e,f)k_R^{p-1}(G_a,G_b,r,s) \quad (3)$$

where $e = (r,r')$ and $f = (s,s')$ are the edges to neighboring nodes of r and s. The above computation can be initialized with the base case $k_R^0(G_a,G_b,r,s) = k_{\textbf{node}}(r,s)$. We can use this recursive expression to construct a dynamic programming table for each pair of relationship graphs. We store values for all node pairs between the two graphs and for all walk lengths up to p. The kernel we have thus defined can be used to compare the local structure of two relationships graphs rooted at particular nodes within those graphs.

We can use k_R^p to define a *p-th order walk graph kernel* k_G^p which compares the global structure of two relationship graphs. Here we use V_G to mean the set of all nodes of graph G. This kernel is computed by summing k_R^p over all node pairs across the two graphs:

$$k_G^p(G_a,G_b) = \sum_{\substack{r\in V_{G_a}\\s\in V_{G_b}}} k_R^p(G_a,G_b,r,s) \quad (4)$$

The running time complexity for computing k_G^p between two graphs is $O(pd_Gd_Hn_Gn_H)$ where d_G is the maximum node degree and n_G is the total number of nodes in the graphs [Harchaoui and Bach 2007]. As we show in Section 7.4, in practice these evaluations are very fast.

The graph kernel we have presented can be interpreted as embedding the graphs in a very high dimensional feature space and computing an inner product $\langle G_a, G_b \rangle$. While inner products are widely useful for many applications, some applications such as the Gaussian kernel $K(G_a, G_b) = e^{-\|G_a - G_b\|^2/\sigma}$ operate on a distance between the objects instead of an inner product. Given a positive-definite kernel, there are many possible distance functions that can be defined over the feature space spanned by the kernel [Ramon and Gärtner 2003]. For a p-th order walk graph kernel k_G^p, a simple corresponding distance function is:

$$d(G_a, G_b) = \sqrt{k_G^p(G_a, G_a) - 2k_G^p(G_a, G_b) + k_G^p(G_b, G_b)}$$

4.4 Algorithm Details

Graph Kernel Normalization. Normalization of the walk graph kernel is used to account for the fact that scenes containing many objects will tend to match better against all other scenes by virtue of their broader coverage of the dataset of models and relationships. For example, the relationship graph formed by a union of all the scenes in the database would match well to every scene. To combat this problem, we implement a normalization term by dividing the result of a graph kernel by the maximum of the evaluation between each graph and itself. For each graph kernel k_G we have a normalized graph kernel $\hat{k_G}$:

$$\hat{k_G}(G_a, G_b) = \frac{k_G(G_a, G_b)}{\max\left(k_G(G_a, G_a), k_G(G_b, G_b)\right)}$$

This normalization ensures that a graph will always match itself with the highest value of 1 and other graphs with values between 0 and 1.

Node Frequency Normalization. The importance of a model within a scene is intrinsically affected by the number of occurrences of that object within the scene. For instance, the existence of a book model in a scene is an important cue from which we can infer the type of room the scene is likely representing. The existence of hundreds more book models will naturally influence our understanding of the scene. However, the relative importance of each additional book diminishes as it is in essence an instance of an agglomeration. In order to represent this we introduce an occurrence frequency normalization factor for the node kernel evaluation following a term weighting approach used for document retrieval normalization [Salton and Buckley 1988]. Concretely, for a node n_a in the set of nodes V_G of a graph G:

$$\sigma(n_a) = \frac{1}{\sum_{n_b \in V_G} k_{\mathbf{node}}(n_a, n_b)}$$

This normalization factor scales the node kernel evaluation defined in Equation 2. The computed value for n_a is equal to 1 if the node is unique and decreases to 0 with an increasing number of similar or identical nodes. Using this approach we avoid the problem of large agglomerations of object instances, such as books in a library, drowning out other interesting structure in a scene.

Architecture Comparisons. Architectural structure is hard to compare to other architecture because it will not in general be related through geometry or tags. However, we would still like to be able to take walks through the architectural support of a scene. To see more clearly why this comparison is challenging, consider two scenes that consist of a desk and a chair on top of a floor. Both scenes contain the walk {"chair", "floor", "desk"} and to a human these walks should be relatively similar. However, using our definition of k_R^p these walks will not contribute to the final graph kernel term unless the model kernel between the two floor models in the scenes is non-zero. Although it is possible that these nodes may have been explicitly tagged as architecture or the two rooms may have an extremely similar design (and thus will have similar Zernike descriptors), in many cases the model kernel we have proposed will evaluate to zero for two different room architecture models.

We use the Google SketchUp database (see Section 6), where almost all of the the scene's architectural geometry is directly attached to the root node. Our approach is to let our node kernel always evaluate to a minimum value for root nodes so that paths going through the root node are not eliminated from consideration. We therefore enforce $k_{\mathbf{node}}(root(G_a), root(G_b)) \geq c$ where $root(G)$ is the root node of a graph G and c is a constant that we set to 0.1 for this work. For other databases, explicit or automatic tagging of architecture could be used instead of this approach.

Parameter Selection. Our graph kernel, k_G^p, is parameterized by p, the length of the walks taken. Different choices for p will capture scene features at different resolutions, and it is unlikely that a single value of p will be the best kernel for any given task. This is a very common problem in machine learning and several multiple kernel learning techniques have been developed to allow learning tasks to benefit from information provided by multiple kernels [Bach et al. 2004]. Although it is possible for machine learning classifiers to directly make use of multiple kernels, it is very convenient to define a single kernel that is a linear combination of k_G^p for different values of p. We use the term "basis kernels" to refer to the individual kernels that are summed to form the final kernel. Given any machine learning task, we can use cross-validation to decide on good weights for each of the basis kernels. Here we formulate a machine learning task that we will use to automatically learn the parameters.

Relevance feedback is a technique used by search engines to improve results. The user selects relevant scenes from a candidate list of results, and the search engine attempts to transform the feature space to favor results similar to ones the user marks as relevant [Papadakis et al. 2008]. The input to our relevance feedback implementation is a set of scenes selected by the user, each marked as either a good or bad response to the query. Given a specific scene kernel, we train a soft margin support vector machine classifier using the selected scenes as the training examples. We use the sequential minimal optimization algorithm to train our SVM [Platt 1999]. Because the SVM is only trained on the selected scenes this training is extremely fast; even if the user selects 50 scenes the SVM optimization always took us less than 50ms. We then use this SVM to rank all scenes in the database according to their signed distance to the separating hyperplane.

To use relevance feedback to learn our kernel parameters we first need to develop a training set. We suppose that we have N different user-designed queries (an example query might be "scenes that contain interesting sconces"). For each task, we assume users have gone through every scene in the corpus and evaluated whether it is a good or bad response to the query.

We can use this dataset to evaluate the quality of our relevance feedback results for a given set of parameters. For each modeling task we randomly pick samples from the database, half from the positive class and half from the negative class. Given a fixed set of kernel weights, we can compute the expected classification error of our SVM and average it over the N modeling tasks. We compute this

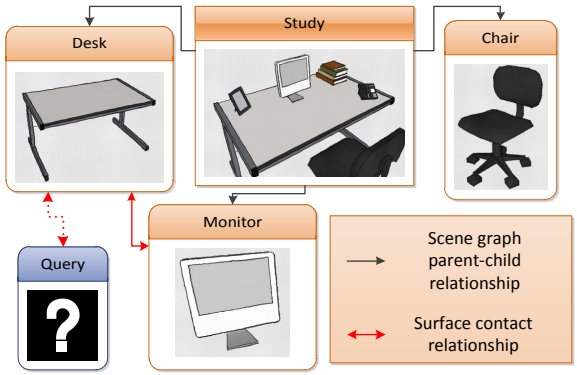

Figure 4: *A model context search expressed by introducing a query node to a relationship graph. The dotted edge represents the contact relationship that connects the query node to the scene and defines its context. Multiple edges of different relationship types can easily be introduced for a single query node.*

average classification error for a large set of possible parameters, and choose the parameters with the lowest average classification error. In addition to varying this over a large set of possible weights for each basis kernel, we also search over different possible values of the soft margin parameter C. Although we found this cross-validation approach to multiple kernel learning to be sufficient for our purposes, it is also possible to directly optimize the weights of the basis kernels as part of a modified SMO algorithm [Bach et al. 2004].

5 Model Context Search

Using our framework, there is an intuitive mapping between a context-based search for 3D models and our rooted-walk graph kernel (k_R^p). We implement such a search by placing a virtual query node in the graph. The relationships that the desired model should have with other models in the scene are defined through a labeled set of connecting query edges. Geometry, tags, and other node properties can optionally be provided by the user to refine the query beyond just considering relationships to other models in the scene. Figure 4 illustrates a query for an object that is in contact with a desk.

Once we have placed the virtual node within the query scene's relationship graph, we then evaluate k_R^p between this virtual node and all other relationship graphs in our scene corpus. For each scene in the database, we have to choose which nodes should be used as candidate models. Unless the user provides additional information to constrain the set of possible models, we use the simple approach of considering all nodes in all relationship graphs as candidates. The k_R^p evaluation for each node indicates how similar the environment around that model is to the environment around the query node. We use this evaluation to rank all models in the database.

In general we cannot directly apply Equation 2 because we do not know the geometry or tags of the query node. However, we would still like to be able to take advantage of size, geometry, or tag information if provided by the user. If any category of information is not provided, we take this to mean that, with respect to the corresponding metric, all nodes should be considered equally desirable.

Given a query node q and another node n against which the query node is being compared, our modified node kernel for query nodes will use tag and geometry kernels $k^Q_{\textbf{tag}}(q,n)$ and $k^Q_{\textbf{geo}}(q,n)$. These kernels return a constant value of 1 if q has no tag or geometry information respectively. Otherwise, the regular versions of the tag and geometry kernels $k_{\textbf{tag}}$ and $k_{\textbf{geo}}$ are evaluated for the pair (q,n). Using these component kernels we define a context query aware node kernel that will take into account all the provided properties for context search query nodes:

$$k^Q_{\textbf{node}}(q,n) = \sigma(n)\left(0.1 + 0.6k^Q_{\textbf{tag}}(q,n) + 0.3k^Q_{\textbf{geo}}(q,n)\right) \quad (5)$$

The walk length parameter p in k_R^p controls the size of the contextual neighborhood that is compared by the model context search. When $p = 0$, all models are ranked by considering only the geometry, tags, or size provided by the user for the query node. When $p = 1$, the algorithm additionally considers the geometry and tags of the models connected to the query node by query edges (in Figure 4, this would just be the desk model). Increasing to $p = 2$ and beyond provides an intuitive way of visualizing the region of support considered by the context search.

6 Dataset

To test the effectiveness of our algorithm, we need a set of 3D scenes. There are several potential large scene databases. For example, virtual worlds such as World of Warcraft and Second Life are gigantic environments that contain large numbers of models. Online databases such as Google 3D Warehouse also contain many scenes that have been uploaded by artists by combining individual models found in the database. We focus on using Google 3D Warehouse because it is publicly accessible and contains work submitted by a diverse set of artists who have modeled many different types of environments. Previous work has explored the use of Google 3D Warehouse as a scene corpus [Fisher and Hanrahan 2010].

Each scene in Google 3D Warehouse is represented using a scene graph and almost all are modeled using Google SketchUp. As an artist models a scene, they can perform operations such as adding tags to scene graph nodes or grouping geometry together forming a new node. For the majority of scenes we collected, most semantically meaningful objects are contained in their own scene graph node. This separation is often done for the artists' own convenience – they structure their scene objects so that related components can be easily manipulated as a single unit. However, as has been noted in previous research, there are still many scene graph nodes that do not correspond to widely useful objects. For example, many models are uploaded by manufacturing companies showcasing their products and are very finely segmented into their constituent components. Many of these components are not useful because they are not easily interchangeable with components in other models.

We use a very simple process to standardize the tagging and segmentation in our scenes. Our approach mimics the methods used in computer vision to construct 2D image datasets such as PASCAL, MSRC, and LabelMe [Russell et al. 2008]. We first accumulate a candidate list of all models in the database by considering all nodes in all scene graphs to be models. We skip scene graph nodes that do not introduce new geometry, such as transform nodes with only one child. We define two scene graph nodes to correspond to the same model if their geometry and texture are equivalent. We say two nodes have the same geometry if the length of the vector difference between their Zernike descriptors is less than a small epsilon (Zernike descriptors are invariant under scaling, rotation, and translation). Likewise, we say two nodes have the same texture if, after scaling both textures to be the same size, the difference between their texture bitmaps is less than a small epsilon. We then present a picture of each model to a human. They provide a main category

and optionally a subcategory for each model. Alternatively, they may choose to classify the model as "not meaningful". All root nodes are considered meaningful, as they correspond to complete scenes that have been uploaded to the database. While previous research has used the tags attached to the scene graph nodes, these tags are sparsely distributed throughout the database and are often unreliable and in many different languages [Fisher and Hanrahan 2010]. Our goal was to acquire a set of scenes with approximately uniform segmentation and tagging quality.

Using this per-model information we can convert our scenes into relationship graphs. All nodes corresponding to models marked as meaningful by users become nodes in the relationship graph. Scene graph parent-child relationship edges are added between a node and its parent. If a node's parent node does not correspond to a meaningful model, we recursively move to the next parent until a node corresponding to a meaningful model is found. Geometry-based relationship edges are then added as described in section 3, and the resulting relationship graphs are used as the input to our algorithm.

We focus on a subset of Warehouse scenes that are relevant to a specific category of queries. We consider all scenes that contain more than one object and have any of the following tags: "kitchen", "study", "office", "desk", or "computer". In total we have chosen to use 310 such scenes. Indoor room scenes are an interesting area to study because their interior design contains structural patterns for our algorithm to capture.

Although in general we have observed most Google 3D Warehouse scenes to be over-segmented, the architecture is usually not well segmented. For scenes such as a house with multiple rooms, the architecture itself contains interesting substructure that is often not captured in our scene graphs. While one might imagine several ways to automatically perform this segmentation, the subset of Google 3D Warehouse scenes we are considering are at most as complicated as a single room and usually do not have complex architectural substructures. Nevertheless, our algorithm can easily take advantage of more detailed information about the architecture that may be provided by some databases, such as computer-generated building layouts [Merrell et al. 2010].

7 Applications

There are a large number of applications of our graph-based scene kernel. We have chosen to focus specifically on applications of the scene kernel that improve scene modeling.

7.1 Relevance Feedback

Recall that k_G^p is parameterized by p and different choices for p capture scene features at different resolutions. As proposed in Section 4.4, we can use relevance feedback to perform parameter selection and determine a good aggregate kernel that captures features at different scales.

To build a training set we presented four different users with our scene database and asked them to think of a scene modeling task of their choice. For example, they might want to find scenes with interesting computer peripherals such as webcams or fax machines, or find scenes with wall sconces that fit well in a study they are modeling. They then classified each scene in the database as either being a good or bad response to their task.

We use this training set for parameter selection. Our basis kernels are walk graph kernels of length 0 to 4. We consider all possible linear combinations of these five kernels with weights ranging from 0 to 1 at 0.1 increments. We also consider the following values for the soft margin parameter C: $\{0.001, 0.01, 0.1, 1, 10, 100\}$. For this test we used 6 positive and 6 negative examples, and averaged the classification error over 10,000 randomly chosen training sets and over each of the four modeling problems. The best set of parameters is shown in Table 2 and had an average classification error of 20.9%. The varied nature of these coefficients suggests that different settings of our graph kernel are capturing different features of the scenes.

C	k_G^0	k_G^1	k_G^2	k_G^3	k_G^4
10	0.5	0	0.1	0.1	0.3

Table 2: *Weighting used for combining graph kernels with different path lengths.*

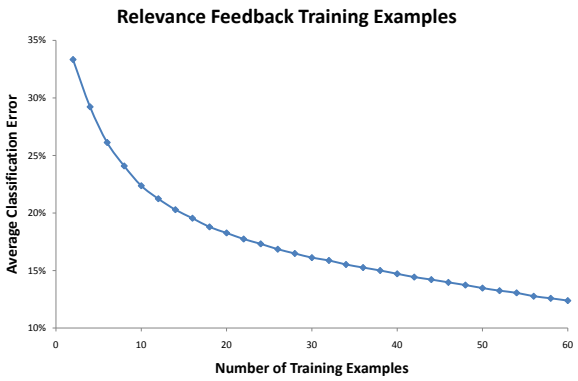

Figure 5: *Classification error using a support vector machine to distinguish between relevant and irrelevant scenes as a function of the number of user-selected training examples.*

In Figure 5 we compare the classification error using the weighted kernel from Table 2 as a function of the number of training examples used. Although the classification error decreases steadily, it remains close to 12% even when 30 positive and 30 negative training examples are used. This suggests that many of the queries designed by our users contain challenging subtleties that are not easily captured by our graph kernel.

In practice we have found that our relevance feedback implementation performs better than our classification error alone would suggest. This is because the SVM ranks the scenes based on confidence, so the user is first presented with suggestions that the algorithm determines are very likely to be relevant. In Figure 6 we show relevance feedback results using 4 training scenes and the coefficients given in Table 2. Even with a small number of selected scenes the algorithm is able to infer the user's preferences. All of the top 18 results are relevant to the query.

At first glance relevance feedback may seem cumbersome — users must first issue a query, classify multiple results, and finally ask for relevance feedback based on their selections. However it is possible for search engines to passively use relevance feedback methods by aggregating search history information from many previous users. For example, if a user issues a common query that the engine has seen many times before, click-through rates or model insertion rates could be used as a proxy to predict the relevance or irrelevance of each result.

7.2 Find Similar Scenes

One application of our scene kernel is to enable the user to search for similar scenes. This kind of query is supported in several 3D model search engines including Google 3D Warehouse [SketchUp 2009]. The user selects an existing scene (either one in the database

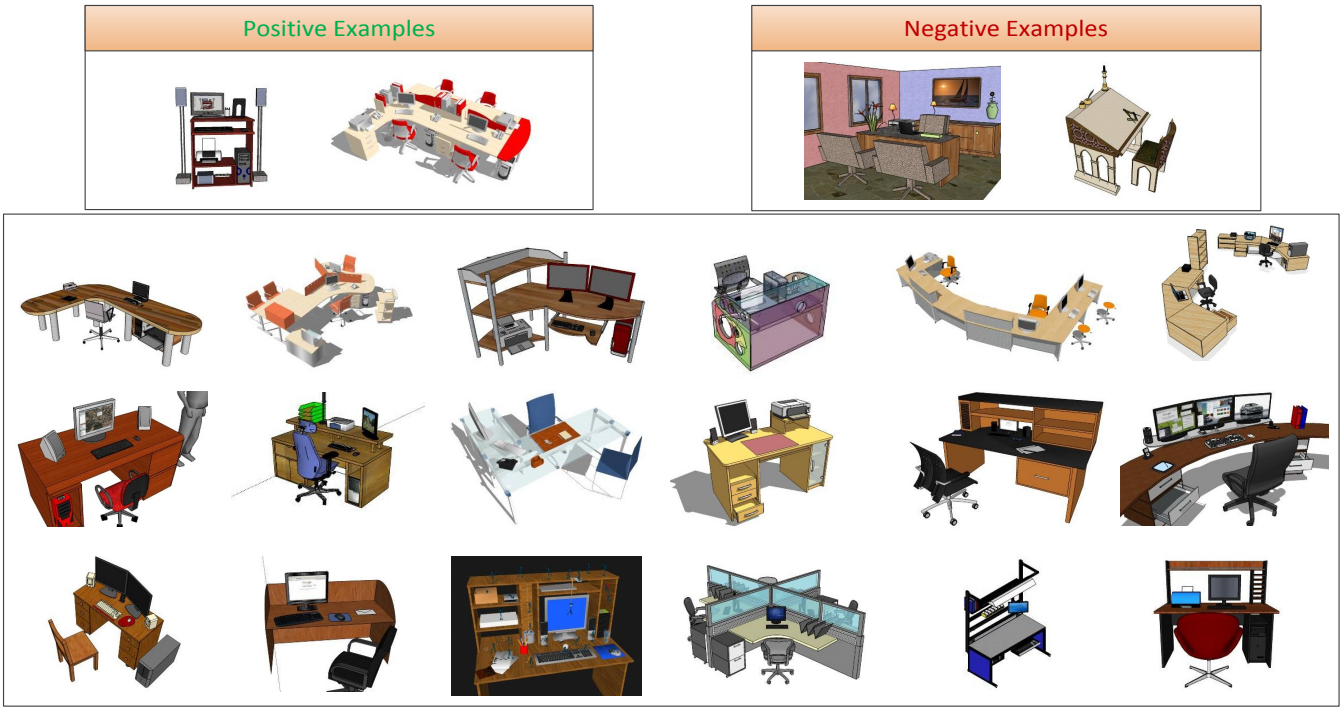

Figure 6: *Search results using relevance feedback. Top: A user looking for scenes with various computer peripherals selects two scenes they like and two scenes they do not like from the database. Bottom: The top 18 results using a scene search guided by the user's selections.*

or the scene they are currently modeling), and asks for similar scenes. We use the aggregate graph kernel described in Table 2 to compute the similarity between the chosen scene and all the scenes in the database. Scenes are ranked using this similarity value in decreasing order.

In Figure 7 we show the top-ranked results for five different queries. Our algorithm returns scenes that share many structural similarities with the query scene. For example, in the first scene many results contain a similar style of shelving. Likewise, in the second scene the top results are all simple desk scenes with laptop computers. These results also demonstrate the large amount of structure sharing used by artists in Google 3D Warehouse. For example, in the fourth scene query, the top ranked result uses the exact same models on the top of the desk, while changing other aspects of the furniture and layout.

7.3 Context-based Model Search

As described in Section 5, we can use the rooted-walk graph kernel k_R^p to search for models that fit well in a provided context. This is implemented by inserting a virtual query node into the scene's relationship graph. This query node is connected to nodes in the scene using edges which represent the desired relationships. We then evaluate k_R^p for all other nodes in relationship graphs of other scenes, and use this to rank all models in the database.

In Figure 8 we show the results of two context-based queries. In both cases the user is searching for models that would belong on top of the desk in their scene. A query node was inserted into these scenes and connected to the desk node through a horizontal support relationship edge. We then evaluate k_R^p for walk length $p = 3$ between the query node and all other relationship graph nodes in the database to determine model suggestions.

The results indicate how the existence of a computer and related models on the desk in the top scene produces computer peripheral and accessory suggestions. In contrast, the bottom scene's context induces results that are more appropriate for a study desk. Also observe that in both cases all of the highly ranked results are at least potentially relevant — only models that have been observed to be horizontally supported by desks (or models geometrically similar to desks) can be highly ranked.

The approach of Fisher and Hanrahan [2010] assumes all pairs of objects are random independent events. In contrast, our method considers the structural relationships between all objects in the scene. To illustrate the difference between these two approaches, we modeled a desk scene with a bowl and two cups, shown on the left side of Figure 9. Consider a user who wants to search for other objects to place on this desk. We want to execute this query using both methods.

There are two main differences between the formulation of context search queries in these algorithms that need to be resolved. First, our approach expresses the desired location as a relationship to existing objects, while their approach expresses the location as a 3D point in the scene. We have chosen a point 10cm above the center of the desk to correspond to our algorithm's "horizontal contact" relationship to the desk. Second, their algorithm makes use of a suggested object size. Though either method can easily incorporate a size kernel, we have chosen to make our comparisons in the absence of size information. Although size information can be very useful, there are a large number of queries where an approximate size is either unavailable or highly unreliable. For example, not all scenes in the database may use the same scale — it is fair to compare relative object size within the same scene but it can be dangerous to assume that absolute unit values are comparable across scenes.

In Figure 9, we show the results using both approaches on our database. Because our algorithm considers the structural relationships in the scene it returns many plausible objects that have been observed on desks in the database, such as lamps and speakers. On

Figure 7: *"Find Similar Scene" search results. Left: The query scene selected by the user. Right: The top six scenes in the database that match the query. The best match is shown on the left.*

Figure 8: *Context-based model search results. Left: A user modeling a desk scene issues a query for a model to be placed on the desk. Right: Suggested models for each query. Note how the context provided by the models in the query scene influences the categories of the suggestions.*

Figure 9: *Left: The user asks for an object on the desk. Right: Results comparing our method against previous work.*

the other hand, their algorithm considers all object pairs independently. It returns objects such as sinks and mixers because these objects are often found in the vicinity of bowls, cups and drawers. By not considering the semantic relationships between models, their algorithm is not able to determine that objects such as sinks are not commonly found on top of desks — our approach can never make such suggestions unless there is a scene in the database with a mixer or sink on top of a desk.

7.4 Performance

Real-time performance is critical for practical uses of our graph kernel. To evaluate the performance of our algorithm we computed the walk graph kernel k_G^4 between all pairs of scenes in our database. Note that the dynamic programming table used for this computation also stores intermediate results for all walk lengths up to $p = 4$. As described in Section 4.1, we precompute the model kernel evaluations between all possible models. It was very easy for us to distribute all the computations of k_G^p over multiple processors because all scene comparisons can be performed independently. The average graph to graph k_G^p evaluation took 0.150ms. Using this average value it would take approximately 1.5s to exhaustively compare a query scene against a 10,000 scene database. This experiment was run on a quad-core 2.53GHz Intel Xeon CPU. Note that this is also the approximate cost of executing a 3D model context query — a subcomponent of evaluating k_G^p between two scenes is to compute k_R^p between all possible node pairs. Overall, we feel our algorithm is fast enough for interactive use on most databases.

8 Discussion and Future Work

We have presented a novel framework for characterizing the structural relationships in scenes using a relationship graph representation. Our basis kernels compare the local information contained in individual models and relationships, and the walk graph kernel aggregates this information to compare the topological similarity between two scenes. We have shown that our algorithm returns a relevant set of results when applied to many scene modeling tasks, including finding similar scenes and context-based model search.

The modular nature of our framework makes it very extensible. For example, it is easy to incorporate new relationship kernels, such as comparing the relationship between two objects in horizontal contact by looking at the size and shape of the contact region. Likewise, we can take advantage of many advancements in graph kernels. For example, the walk graph kernel is subject to "tottering", where the walks are allowed to move along a direction and then instantly return to the original position. This can result in many redundant paths, but it is possible to transform the input graphs so as to disallow these walks [Mahé et al. 2004].

Using a kernel-based approach to scene comparison makes it possible to leverage the substantial literature on kernel methods from machine learning and opens many related applications that can be explored. For example, classification algorithms can be used to automatically classify new scenes that are uploaded to a scene database. Semi-supervised learning can be used to propagate a sparse set of scene tags across all scenes in the database [Goldfeder and Allen 2008]. Kernel-based clustering algorithms could be applied to the set of models or scenes, producing clusters like "silverware" and "table centerpieces".

While our goal was to compare scenes in a way that coincides with the human notion of scene similarity, there are many cases that our approach handles poorly. First, many of our relationships rely on two models being in planar contact; however, for our dataset we observed that our algorithm would sometimes fail to capture the correct spatial relationship between objects because they were represented by either inter-penetrating or floating geometry. Furthermore, many spatial primitives cannot be disambiguated by geometry alone but are often used by human observers to classify relationships.

One challenge we encountered was that scene modeling programs such as Google SketchUp do not encourage artists to maintain a reasonable segmentation of their scene. The scene graph is maintained mostly for the purpose of rendering, and certain types of editing operations can severely fragment meaningful objects into many nodes. As scene modeling tasks become more common, scene modeling programs will likely make more of an effort to help artists maintain a meaningful semantic segmentation of their scene and better track the relationships between these objects. This will make the limited manual segmentation we performed unnecessary and is extremely useful for enabling user interaction in applications such as virtual worlds. We feel that software that is aware of the relationships expressed in 3D scenes has significant potential to augment the scene design process.

Acknowledgments

Support for this research was provided by the Fannie and John Hertz Foundation. We would also like to thank Google for allowing the images of 3D Warehouse models to be published.

References

BACH, F., LANCKRIET, G., AND JORDAN, M. 2004. Multiple kernel learning, conic duality, and the SMO algorithm. In *Proceedings of the 21st international conference on Machine learning*, ACM.

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.

CRISTIANINI, N., AND SHAWE-TAYLOR, J. 2000. *An introduction to support Vector Machines: and other kernel-based learning methods*. Cambridge University Press.

FEIST, M. 2000. *On in and on: An investigation into the linguistic encoding of spatial scenes*. UMI, Ann Arbor, Michigan.

FISHER, M., AND HANRAHAN, P. 2010. Context-based search for 3D models. *ACM Trans. Graph. 29* (December), 182:1–182:10.

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.

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.

GARTNER, T., FLACH, P., AND WROBEL, S. 2003. On graph kernels: Hardness results and efficient alternatives. In *Proceedings of the 16th Annual Conference on Learning Theory*, 129.

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.

HARCHAOUI, Z., AND BACH, F. 2007. Image classification with segmentation graph kernels. In *Computer Vision and Pattern Recognition*, IEEE.

KASHIMA, H., TSUDA, K., AND INOKUCHI, A. 2004. Kernels for graphs. *Kernel methods in computational biology*, 155–170.

MAHÉ, P., UEDA, N., AKUTSU, T., PERRET, J., AND VERT, J. 2004. Extensions of marginalized graph kernels. In *Proceedings of the twenty-first international conference on Machine learning*, ACM, 70.

MERRELL, P., SCHKUFZA, E., AND KOLTUN, V. 2010. Computer-generated residential building layouts. *ACM Transactions on Graphics (TOG) 29*, 6, 181.

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.

PARABOSCHI, L., BIASOTTI, S., AND FALCIDIENO, B. 2007. 3D scene comparison using topological graphs. *Eurographics Italian Chapter, Trento (Italy)*, 87–93.

PLATT, J. 1999. Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods*, MIT press, 185–208.

RAMON, J., AND GÄRTNER, T. 2003. Expressivity versus efficiency of graph kernels. In *First International Workshop on Mining Graphs, Trees and Sequences*, 65–74.

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., AND BUCKLEY, C. 1988. Term-weighting approaches in automatic text retrieval. In *Information Processing and Management*, 513–523.

SHAWE-TAYLOR, J., AND CRISTIANINI, N. 2004. *Kernel methods for pattern analysis*. Cambridge University Press.

SKETCHUP, 2009. Google sketchup blog. `sketchupdate.blogspot.com/2009/05/3d-warehouse-now-in-better-shape.html`.

TANGELDER, J., AND VELTKAMP, R. 2008. A survey of content based 3D shape retrieval methods. *Multimedia Tools and Applications 39*, 3, 441–471.

XU, Y., AND KEMP, C. 2010. Constructing spatial concepts from universal primitives. *Proceedings of the 32nd Annual Conference of the Cognitive Science Society*.