PolySquare: A Search Engine for 3D Models with Tag Propagation

Minji Kim
mjkim@hcil.snu.ac.kr
Seoul National University

Junhoe Kim
jhkim@hcil.snu.ac.kr
Seoul National University

Gwanmo Park
gmpark@hcil.snu.ac.kr
Seoul National University

Jinwook Seo
jseo@hcil.snu.ac.kr
Seoul National University

ABSTRACT

Searching for desired 3D models is not easy because many of them are not well labeled; annotations often contain inconsistent information (e.g., uploaders' personal way of naming) and lack important details (e.g., detailed ornaments and pattern) of each model. We introduce PolySquare, a search engine for 3D models based on tag propagation—the process of assigning existing tags to other similar but unlabeled models considering important local properties. For instance, a tag 'wheel' of a wheelchair can be spread out to other objects with wheels. Furthermore, PolySquare allows people to interactively refine the search results by iteratively including desired shapes and excluding unwanted ones. We evaluate the performance of tag propagation by measuring the precision-recall of propagation results with various similarity thresholds and demonstrate the effectiveness of the use of local features. We also showcase how PolySquare handles the unrefined tags through a case study using real 3D model data from Google Poly.

CCS CONCEPTS

• **Human-centered computing** → **Interactive systems and tools**; • **Information systems** → *Search interfaces*; • **Computing methodologies** → Visual content-based indexing and retrieval; Machine learning approaches;

KEYWORDS

3D tagging, 3D text annotation, 3D model search, 3D data retrieval, Neural networks

ACM Reference format:
Minji Kim, Junhoe Kim, Gwanmo Park, and Jinwook Seo. 2020. PolySquare: A Search Engine for 3D Models with Tag Propagation. In *Proceedings of 25th International Conference on Intelligent User Interfaces, Cagliari, Italy, March 17–20, 2020 (IUI '20),* 11 pages.
https://doi.org/10.1145/3377325.3377484

IUI '20, March 17–20, 2020, Cagliari, Italy

ACM ISBN 978-1-4503-7118-6/20/03…$15.00
https://doi.org/10.1145/3377325.3377484

1 INTRODUCTION

The need for searching for 3D models has increased in recent years due to the fast growth of the AR/VR field and increase of people's interest [2], spawning various free or paid 3D model gallery websites. While many search sites adhere to keyword search, the tags on the models often contain unnecessary or insufficient information [30]. For instance, tags are often used for personal purposes such as indicating the purpose or inspiration of the uploader which does not describe the model itself. Furthermore, tags can be too general; for example, a tag 'guitar' does not provide enough details about the shape of a guitar model compared to other tags such as 'electric/acoustic.' Such lack of meaningful tags makes people hesitate to use keyword search and resort to manually browsing through the 3D model gallery until they find the models they want.

Automatically assigning appropriate tags from labelled models to unlabelled ones can mitigate this problem by enriching the text annotation. This process is called *tag propagation* [9, 19]. Goldfeder et al. and a few studies [16, 28, 31] adopted the concept of tag propagation for 3D data for the first time; these studies annotated a 3D model with the tags of its 15 most geometrically similar neighbors. However, since such a method only considers the overall geometric distance between models, tags denoting a local structure can be propagated incorrectly; for example, a chair with wheels and a chair without wheels look similar overall, but a tag 'wheel' from the first one should not be propagated to the second one. In contrast, there is an opposite case where tags should be propagated even though two models are not similar. For example, a chair and a locker are dissimilar overall but a tag 'wheel,' from the chair should be propagated to the locker if both have the similar local structure, i.e., wheels.

We propose PolySquare, a search engine for 3D models using tag propagation: sharing tags among nearest neighbors with respect to both global and local features. Figure 1 shows the pipeline of tag propagation in PolySquare. PolySquare propagates tags between similar models using global features, i.e., *full-body propagation*. Then, *per-part propagation* attaches tags to the models whose local properties are similar to each other. We also adopt a language model to detect and remove anomalous or irrelevant tags, reducing false positives during the propagation process. To diminish the impact of mispropagated tags, the search interface of PolySquare allows users to refine search results iteratively through sorting and filtering. Tag propagation can be used by the search site managers who want to enhance the search quality, or the researchers who need 3D models with rich text-annotations. Consumers of the 3D models can find

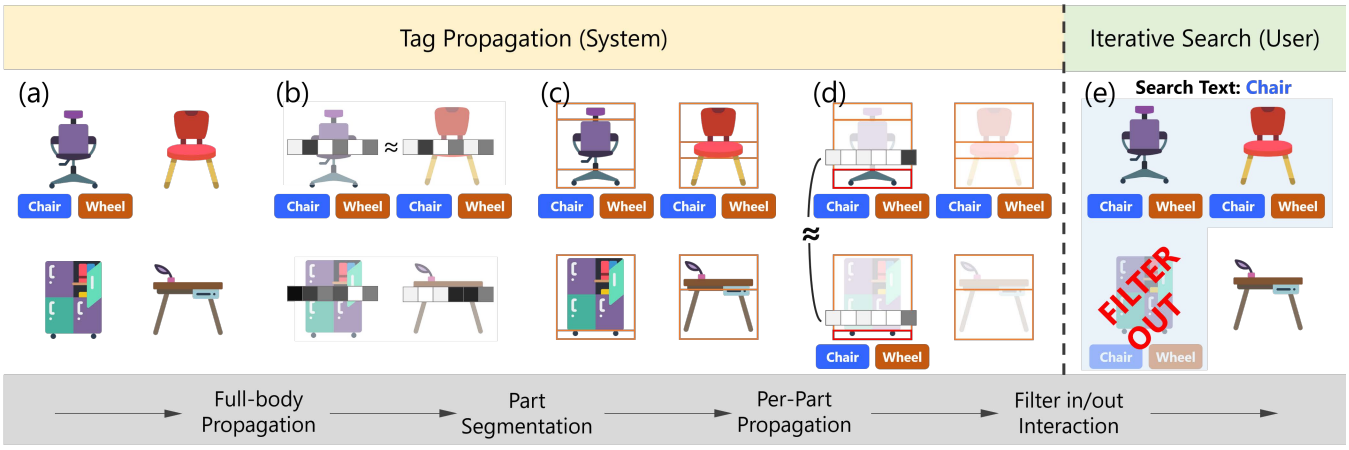

Figure 1: The pipeline of PolySquare. The tag propagation process contains 4 steps: full-body propagation, part-segmentation, per-part propagation, and anomaly detection. (a) Before propagating tags, only the purple chair has the tags 'chair' and 'wheel'. (b) In full-body propagation, the red chair copies tags from the purple chair, since the global features of two models are similar. (c) Part segmentation and feature extraction are conducted beforehand. PolySquare loads the features of each model. (d) In per-part propagation, the locker gets tags of the purple chair, since they have wheels in common. (e) When searching for a chair, the locker also appears in the search results. Users can filter out the locker with a simple click interaction. — Furniture icons designed by Smashicons from Flaticon.

their desired models faster using the system, and the authors of the models get recommended new tags to increase the search rate of their models.

We summarize our contributions as follows:

1. We design and develop PolySquare, the search engine that can perform a 3D model keyword search using the propagated tags.
2. We introduce tag propagation using a 3D similarity measure of local features extracted by a deep learning model. We also utilize a language model to prevent attaching irrelevant tags.
3. We designed an intuitive interaction to enable users to improve the quality of search results by iteratively adding the wanted objects and deleting the unwanted ones.
4. Through a precision-recall experiment, we show the usefulness of our tag propagation and analyze its performance in regards to various similarity thresholds.
5. Through a case study using real 3D model data from Google Poly, we show that PolySquare can be adapted to the existing wild dataset with various tags.

2 RELATED WORK

Our work is relevant to both tagging 3D models and 3D search methods. In this section, we summarize previous research on these two categories.

2.1 3D Text Annotation

Most of the studies on automatic 3D text annotation were about the effective and automatic class labeling of 3D data. Research on 3D text annotation can be classified under two large groups: interactive tools and automatic labeling. The first group includes research on the interactive tools to annotate 3D data efficiently, and the second one covers studies to label the 3D data automatically.

Ponchio et al. [35] suggested an interactive labeling through a two-pass rendering solution. Julien Valentin et al. [42] developed an interactive tool that can correct errors during scene segmentation by reflecting user labeling in real time. Papaleo et al. [32] proposed an interactive tool for annotating 3D data that allows the hierarchical semantic-driven tagging. These interactive tools that require users to annotate the model themselves are not suitable for tagging existing massive 3D data since they require a lot of time and effort.

Yi et al. [47] used both automatic labeling and human verification to enable annotation with minimal manual manipulation. When users annotated the model, the rest of the similar shapes were annotated automatically, and the labeling model was trained again when users verify the results. Many other tools have been developed to automatically perform semantic labeling, such as Koppula et al.'s semantic labeling for 3D point clouds of indoor scenes [25], Kalogerakis et al.'s 3D mesh segmentation and labeling method, and more [22, 39]. These tools attach predetermined labels to the separated parts. Therefore, most of the studies supporting automatic labeling do not use customizable tags, and only assign a name of the part to the model. In other words, it is difficult to borrow and apply diverse tagging methods used in general 3D gallery sites. Our work intends to enhance the text annotation of 3D data by accommodating user-defined tags, as well as simple labels.

Goldfeder et al. derived a tag propagation technique from the research area of image search for 3D models for the first time [9, 16, 17]. They utilized the existing geometric shape descriptor to propagate tags to similar objects. This attempt, however, could miss the tags that reflect the local properties of the object. Ohbuchi et al. [31] and Li et al. [28] also attempted to use similar methods

to enhance the text annotation of the data. Ohbuchi et al. used the existing class labels as tags, which do not reflect the detailed shape properties of the objects. These two studies attempted to use mixed features (global + local) through statistical feature extraction. However, they did not used local features to compare parts of each model. Furthermore, there was no elaborate anomaly treatment for irrelevant keywords other than assigning more weights to a title and tags than a description of the model. Therefore, the propagation could still assign wrong tags. Our work adopts the same concept of tag propagation, but enhances the robustness of the process by extracting both global and local features using a deep learning model. We also detect anomalies in a tag propagation process to reduce the number of irrelevant tags.

2.2 3D Retrieval and Searching

A lot of studies presented various search methods to increase the retrieval rate and the efficiency of the 3D model search. Min et al. [30], Bustos and Li [4, 27] compared the effectiveness of 3D searching methods. Their research focused on the content-based search rather than the keyword search, citing the lack of text annotation in 3D data as a main reason for a low performance of keyword search. Because of the lack of tag in 3D data, researchers presented and analyzed various 3D retrieval methods that do not depend on text annotations. Sketch, image, and shape search are representative content-based search [1, 18, 44, 46]. In addition, researchers tried to discover 3D features that can be utilized to improve the search efficiency. Recently, many studies performed feature extractions using deep learning. Using deep learning models, Su et al. [41] extracted 3D features using multi-view rendered images, and many researchers including Qi et al. used point clouds to generate feature vectors [36, 37, 45]. Hanocka et al. and Feng et al. introduced networks for 3D mesh representation [13, 20]. These studies improve the efficiency of the 3D model search, but require more interactions than the simple keyword search, such as preparing a 3D shape or making a separate 2D sketch. PolySquare uses keyword search instead of shape or sketch search, and utilizes a deep learning model to extract the features, which can be used to measure similarities between 3D models.

Some studies have attempted to deal with 3D search efficiency from the users' point of view rather than a low-level analysis such as retrieval rate. Funkhouser et al. [15] introduced a 3D search engine that supports both keyword and sketch-based retrieval. Users can sketch the wanted object or type a keyword to search for a model and use the results to find similarly shaped models. There are also studies that allow 3D retrieval to be performed during modeling process, so that the intermediate results can be used for retrieval. Fan et al. [12] proposed a shadow guide and sketch based modeling. According to the drawing of the user, the search engine finds similar models and projects them on the screen as a shadow guide. Chaudhuri et al. [6] linked local parts of 3D models to semantic words (e.g., dangerous, scary, and strong) reflecting the intent people may have when modeling 3D objects. Users can select the component that is appropriate to the object they are making, considering the strength of semantics through the keywords. These studies are meaningful in that they drive 3D local parts to reflect the user's modeling intention. We take a user-centered approach that allows users to directly refine search results and interact with 3D models by simple interactions (click and drag).

3 TAG PROPAGATION

We use the tag propagation method to enhance the existing text annotations for 3D models. We choose which tags to share based on how similar the shapes of the models are using both global and local features. When considering local features, PolySquare segments the models and selects which part will be used for the propagation (Figure 1). The propagation method is implemented and tested with two different thresholds: (1) *count-based threshold*, which reflects n most similar models. (2) *score-based threshold*, which propagates tags with a distance smaller than k. Also, to prevent false positives, we developed a language model using GloVe [33] word vectors to remove anomalies. We used about 400,000 words encoded using GloVe, trained on the Wikipedia corpus. The language model treats words that are not included in the list of GloVe vectors as out-of-vocabulary (OOV), and it considers the tags not related to others attached to the same object as anomalies and excludes them from the final tag group.

Algorithm 1 Lazy Tag Propagation

```
1:  procedure Lazy Tag Propagation(threshold)
2:      initialize feature dictionary D of the models
3:      for model in database do feat = D[model]
4:          for id in D do
5:              distance = MSE(feat − D[id])
6:          end for
7:          sort the distances
8:          if Thresholding by nearest neighbors then
9:              ids = n most similar models (n = threshold)
10:             update lazylist ← ids
11:         else if Thresholding by distance then
12:             ids = models if distance < threshold
13:             update lazylist ← ids
14:         end if
15:     end for
16:     for id in lazylist do
17:         for _id in lazylist[id] do
18:             lazylist[id] = lazylist[id] + lazylist[_id]
19:         end for
20:         exclude anomalies from tags
21:         update tags of the ids
22:     end for
23: end procedure
```

3.1 Lazy Tag Propagation

Lazy propagation, also known as call-by-need, is a strategy to delay the execution until the values are needed [43]. If we naively traverse the models to propagate tags, the result differs based on the order of traversal. This is because the distribution of the tags continues to change during the traversal. Therefore, we introduce a lazy tag propagation algorithm to update tags only after each iteration ends.

Full-Body Propagation Lazy tag propagation is executed as in Algorithm 1. We first initialize the feature dictionary of 3D models.

Then, for all the models in the dataset, we find n most similar objects (or the objects with higher similarity score than the threshold) and store their id's in the lazy list. Here, the number of similar objects (or the similarity score) is a hyper parameter. After updating the lazy list, we iterate over it, and for each nearest neighbor's id, we add $lazylist[id]$ to the existing id list. That is, all the nearest neighbors of the nearest neighbors are also included in the results. Finally, we store the propagated tags to the database. The time complexity of the lazy tag propagation is $O(n^2)$.

Per-Part Propagation PolySquare segments each object based on its class (i.e., chair, airplane, laptop, knife, and guitar) and performs per-part propagation only when a notable part with distinctive characteristics exists in the object. In order to select an important part without prior knowledge about the relationship between the object and its tags (e.g., whether a tag 'wheel' refers to the leg or the armrest of a chair), we compute average normalized distance from each segmented part to the corresponding parts of other objects of the same class. If the smallest distance of a part of the target object is less than the distance to the nearest neighbor selected from the full-body propagation and is below the threshold ($\approx$0.5), we consider that part as significant. The tag propagation process for each significant part follows the same process as in Algorithm 1.

3.2 Feature Extraction and Segmentation

We explain a detailed method for extracting the features of 3D objects used throughout PolySquare, and then present how we segment local parts of 3D objects. We use Multi-View Convolutional Neural Network (MVCNN) [41] for extracting features of 3D objects. The view-based descriptor is easy to train using the pre-trained model of existing 2D image corpus, and the computation is efficient because even a complex model can be represented as a constant size image. MVCNNN inherited the previous view-based descriptors [7], and used projected views of 3D models in different directions. Unlike when a person decided important features by hand, MVCNN used deep learning to extract the features and showed better performance [24]. In recent years, various methods for 3D feature extraction have been tried [14]. PolySquare uses a light-weight MVCNN model with relatively stable performance.

The model consists of two stages of CNNs. The first stage is the image-based CNN, and we use ResNet-18 [21] as a backbone. This network is pre-trained on ImageNet [38] images. The outputs of each of the 12 multi-view images are aggregated in a view-pooling layer using element-wise maximum operation and sent to the second network. The second step is a directed acyclic graph, and we also use ResNet-18 from [21]. We use the penultimate layer after ReLU non-linearity as a shape descriptor. For generating input multi-view images, we render the models from 12 view angles: rotated by 30 degrees around the y axis, using Blender [8] rendering tool. The rendering camera is pointing towards the center of the object and about 30 degrees above the ground. The model is trained on the Modelnet40 [48] dataset using an Adam optimizer [23], with a learning rate of 5e-5 inspired by the original paper [41]. We assume 3D models to be upright, and normalize the size and the position of these models. Feature vectors of the models are pre-computed as 512 dimensions and stored in the database.

In order to use the features of the local parts of the data in tag propagation and a filtering method, we employ PointNet [36] to segment the 3D models into local parts. We use PointNet for its robustness to input perturbation and its simple but efficient network. The part segmentation concatenates the local and global features of the model, generated from the fully connected layers. (The global feature here is the output of the max-pooling layer) Then the network predicts the scores for each point, and these scores are used to determine the class of the points. Since we use 3D mesh data for the tag propagation input, we sample the input points from the face centers of the model. We sample 2,500 unique random points from the model, and segment class for each sampled point. The rest of the points follow the class of the nearest sampled point. The model is trained on ShapeNetCore [5] using the Adam optimizer [23] with a learning rate of 1e-3 as denoted in the original paper [36]. All the segmented parts are generated beforehand, and stored in the file system.

3.3 Language Analysis

We use pre-trained GloVe [33] word vectors to exclude unrelated tags from a propagation process. The GloVe vector is a powerful word embedding method, where a dot product of two vectors equals the logarithm of the words' probability of co-occurrence. Other popular models Bert [10] and Elmo [34] are not suitable for embedding tags because they do dynamic embedding.

Using the word vectors, we exclude tags that are not in the dictionary (i.e., meaningless indices, username of the author) and find anomalies from the tag group. We use two methods to find the anomalies from the words: one is K-Means [29] clustering, and the other is classifying the tag to be safe if the distances from all other tags are below a certain level. K-Means clustering performed better than other clustering algorithms such as DBScan [11]. This is because tags corresponding to a single 3D model are often grouped in a similar sense, with a bundle of three to five such topics. In K-Means algorithm, we fix the value of k to 2 and consider the larger cluster to be the main one. Adjusting the k value from 2 to 8, the smaller k value shows better performance because there are not many tags in the experimental data. The tags are classified as anomalies if each feature has a larger distance from the cluster center than the threshold. The threshold value is set to 6 by an empirical experiment.

The latter method is more flexible in judging anomaly because it classifies tags to be safe when they are close to just one other tag. The threshold value for distance is set to 6 by experiment, same as for K-Means. We compared the performance of K-Means clustering with the latter classifying method and found that the the performance degrades when the number of tags is small. This is because the cluster center does not reflect the value of safe tags if the number of words is small. We chose the second method, heuristic that filters the anomaly in a more conservative way to increase recall of search results. The remaining false positives are expected to be excluded through user interaction. The language model phase thus filters out extraneous words not specified in the dictionary.

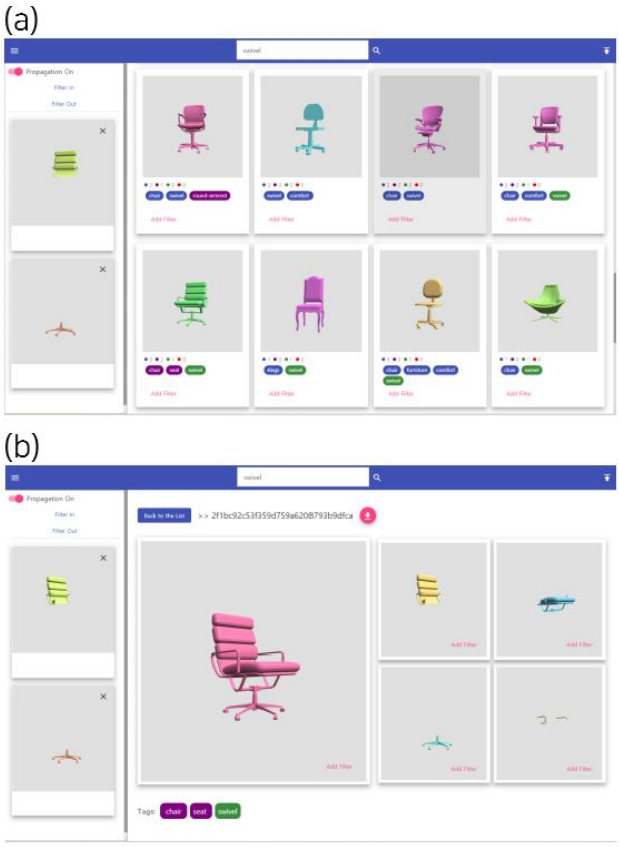

Figure 2: Searching interface (a) Overview—The search results appear as a list. The left of the screen is a filter panel. Users can either filter-in or out by clicking the button. Each model has the tags below the model. (Blue: original tag, Purple: propagated tag, Green: propagated tag from the part) (b) Detail view—The model and its parts can be rotated using the left mouse button and drag interaction. The model can be downloaded as obj format, and each part can be added to the filtering panel using the 'Add Filter' button.

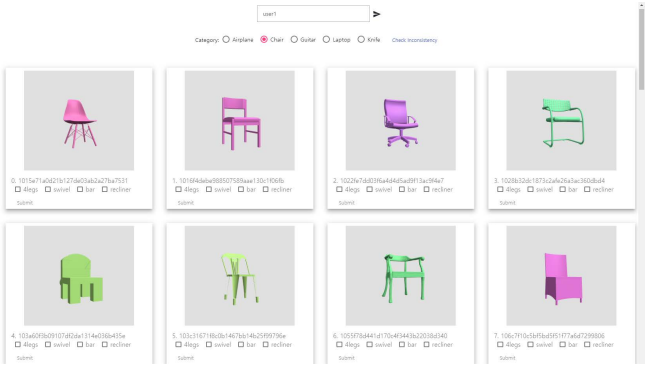

Figure 3: Tagging interface for building a ground truth dataset. For the consistency of the dataset, we took the following tagging process: First, we attach tags to 20% of the models respectively. Next, we compared the differences and made a consensus of criteria for associating them. Finally, we encode the rest of the models according to the rules.

4 SEARCHING INTERFACE

Based on the method described in this paper, we designed a search interface for users to interactively improve search results (Figure 2). When users query a keyword, PolySquare sends results that match one of their attributes (e.g., title, description, tags, augmented tags) against the keyword. Through the filter in/out interaction, the users can refine the results to what they want. In this section, we describe the components of the interface and interactions.

4.1 Overview and Detail

In the overview, the interface displays search results in a grid view with four models in each row (Figure 2(a)). Users can scroll down to explore the results and find a desired model. Also, users can turn on and off the propagation mode that determines whether to use the propagated tags in a searching process. The detail-view (Figure 2(b)) shows the model and its local parts with tags. Users can rotate models freely by dragging with the right mouse button and download them in a .obj format. All tags appear in both the overview and the detail view. The tags that the model originally had appear in indigo color by default. While the propagation mode is on, two types of tags are added without duplication. The tags propagated from similar models through full-body propagation are added in purple, and the ones propagated by per-part propagation are added in green. Lastly, the tags identified as anomalies by PolySquare's language model are shown in red.

4.2 Iterative Search

To complement failure cases where the incorrect tags are attached to the models, the interface supports the iterative refinement of the search result using filter-in/out operations. When users encounter unsatisfactory results from keyword search, they can register desired parts (or full-body) of the object to filter-in, and the unwanted ones to filter-out as *filters*. The **filter-in** operation adds models that are similar to the filters, although they were not included in the previous result. Then it sorts the results according to the shapes of the filters. The more similar the shape is to the filters, the earlier it appears on the results. On the other side, the **filter-out** operation excludes the models that are similar to the selected filters. In a filter-in process, PolySquare finds 30 nearest models for each filter based on the feature representation. They are included into the result without duplication before sorting them. Likewise, PolySquare finds five nearest models to exclude when users apply the filter-out operation. For the filtering operation, users can add the whole model and its parts to the filter panel by clicking on the 'Add filter' button. They can also add a specific part separately to the list in the detail view. In the filtering panel on the left side, users can adjust the filtering priority by dragging the model cards. Once users have determined the filter models and their order, they click the 'filter-in/out' button.

Table 1: The number of models and the parts for each category, and a list of the tags used in evaluations.

	# models	# parts	tested tags
Airplane	500	4	propeller, high_tail
Chair	500	4	4legs, wheels, recliner, bar
Guitar	500	3	v-head
Knife	392	2	curved, dart
Laptop	445	2	-

Table 2: Cohen's κ coefficient of the tagging result. Four tags are marked as 'Very Good', and the rest of the tags are marked as 'Good'. The tags with the coefficient value below 0.75 are not used in evaluations, except the first precision-recall test.

tag	category	Cohens's κ coefficient	strength of agreement
high_tail	Airplane	0.953	Very Good
propeller	Airplane	0.78	Good
4legs	Chair	0.931	Very Good
wheels	Chair	0.962	Very Good
bar	Chair	0.839	Very Good
recliner	Chair	0.701	Good
v-head	Guitar	0.931	Very Good
curved	Knife	0.615	Good
dart	Knife	0.872	Very Good

5 EVALUATION

We evaluated how tags were propagated using ShapeNetCore [5], a large repository of 3D data (55k objects) that is actively used in 3D retrieval research. From the entire ShapeNetCore dataset, we chose five categories (airplane, chair, guitar, knife, and laptop) to use for the evaluation since the segmentation algorithm, PointNet [36], that we used to segment the local parts of each object, was known to perform the best among those categories. The distribution of the data is shown in Table 1. We initially tested ShapeNetSem [40] as it contains richly annotated data. However, compared to ShapeNetCore, ShapeNetSem is smaller and densely annotated. When we extracted the models belonging to the five categories, the number of models in each category was below 100. We decided to annotate the ShapeNetCore data with the labels given in the ShapeNetSem dataset, and as Ohbuchi et al. [31] used class labels to annotate the dataset, we used the names of the class and their synonyms, along with other related words used in Google Poly.

We selected the tags that represent a local feature of the object as test tags. For instance, in the Chair category, '4legs, wheel, bar' are the tags that show the legs' property of the chair, and 'recliner' reflects the angle of the backrest. The tested tags are given in Table 1. We generated the ground truth data for the test tags by encoding all the models of ShapeNetCore according to a specific encoding rule, and then used it to measure the precision-recall of the tag propagation. We implemented a tagging web page for encoding the models (Figure 3). We did not include the test tags for the Laptop category because the shapes of the models are relatively not

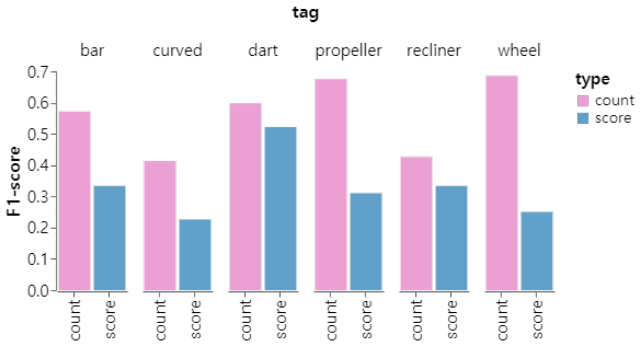

Figure 4: Comparison between the F1 scores of the count-based threshold and the score-based threshold for each tag. In all cases, count-based thresholds showed a higher performance than score-based ones.

distinctive. Two authors participated in the tagging process. After finishing encoding 20% of the whole data independently, we compared the results and confirmed the tagging criteria. Cohen's κ coefficients [26] for each tag are reported in Table 2. Using the consensus criteria, we tagged the rest of the data. Since the percentage of tagged objects vary from dataset to dataset, we arbitrary attached the test tags to 30% of the ground truth data so that propagation can be made.

In this section, we look into the performance differences based on the tag propagation method, the features used, and the use of the language model. Then, we report how precision-recall changes after using our iterative search.

5.1 Tag Propagation Thresholds

We tested and compared both methods using (1) *the number of similar models* (count-based) and (2) *the similarity score* (score-based) to propagate tags. The former method is determining which tags to propagate based on a certain number of similar models. The latter method is to propagate tags by referencing the models with a similarity score higher than a certain threshold.

We measured precision-recall of the propagation while changing the thresholds. The count-based thresholds ranged from 1 to 30, and the score-based thresholds ranged from 0 to 20. The F1 score is calculated based on the point where two graphs of precision and recall meet. This is to set a proper threshold value considering both precision and recall. Tables of detailed thresholds and corresponding precision-recall values are attached as the supplementary material.

Figure 4 presents the difference in F1 score for each tag with a similarity number and score. Propagating tags using the count-based threshold performs much better than using the score-based thresholds. In the case of score-based thresholds, the precision-recall graph showed a steep slope because the distribution of similarity scores of objects corresponding to the tag is uneven. Since the performance of the count-based threshold was obviously better than the score-base, we used the count-based threshold by default for later experiments . The best performing tags in count-based

threshold experiments were 'wheel' and 'propeller', and the worst performing tags were 'curved' and 'recliner'. The 'recliner' had especially low recall and accuracy values, which means that similarity analysis was not done properly. The low match rate from the encoding stage suggests that the features of the 'recliner' and the 'curved' themselves were not strong enough to represent important characteristics of the models, and the ground truth data became inconsistent. Also, the precision-recall charts of these two tags were unstable, and showed the abnormal patterns, different from the charts of the rest of the tags. Therefore, we excluded them from the rest of the experiment after the first precision-recall test.

5.2 Per-Part Propagation Performance

We used the feature of each segmented part of the model as a local feature. To make sure that local features actually contribute to performance improvements in propagation, we examined the precision-recall of tag propagation using local features. We compared the result of each of the local features, along with the global ones. As in Figure 5, 'bar', denoting the chairs with c-shaped legs, and '4legs' had the best performance when using the legs of the chair. 'V-head' showed the best F1 score when using the head of the guitar, and the performance of 'dart' (denoting the knifes with symmetrical blades) was the best when using the part 'blade'. The best performing local features of these tags matched the part most closely related to the tag. (e.g., '4legs'-'legs', 'v-head'-'head') As a result, we can infer that if the tag-related local part is used in tag propagation, the performance of the propagation could increase.

However, 'wheel' performed well with the global feature, even if the local features of the part 'legs' had the highest F1 score among the other local features. At the time of encoding, all chairs with wheels were chosen as correct answers. After encoding, however, we found that many of the chairs with wheels also had armrests and this resulted in false propagation of the tag 'wheel' to chairs with armrests. 'Propeller' and 'high_tail' (high-positioned plane's tail wings) also had lower F1 scores when using local features than using global features. Airplane models contain propellers both in the body and the wings of the model. That is to say, the feature of the tag 'propeller' has been split into both local parts. In these cases, we could improve the performance of per-part propagation by selecting multiple local parts that are equally dominant in a model. (e.g., selecting both 'legs' and 'arms' as significant local parts)

Unlike the legs of the chair with the tag 'bar' or '4legs', wheels take up only a part of the legs. Propellers are on both the body and the wing of an airplane, but they take up much larger portion on the wing than on the body. Therefore, the local feature corresponding to the wing have a greater impact on propagation performance than the body. The tag 'high_tail' denotes the tail wing of an airplane, and it refers to a portion of the part 'tail'. In conclusion, the percentage of the local parts related to the tag has a significant impact on tag propagation, as well as what part of an object the tag belongs to.

We also considered the mixed feature (local + global feature) mentioned in Ohbuchi et al. [31] by performing tag propagation using max-pooling results of local and global features. Using mixed features did not make a significant difference from using global features. The result of the experiment using the mixed feature is attached to the supplementary material.

We selected a dominant local part from the object and used it in per-part propagation. We tested if the propagation using selected local features improves the propagation performance, using `count-based threshold=4`, and clipping the distance above 0.3 for each propagation. When we compare the result considering local features to the case only using global features, the precision of the 'dart' increased from 0.5699 to 0.5789, and the precision of the 'wheel' increased from 0.4737 to 0.5, while the precision values of the rest of the objects remained the same. As tags of the similar objects are added to, and not deleted from the existing list of tags, the recall values also increased with precision.

5.3 Language Model and Iterative search

We tested how the language model contributes to the precision performance through anomaly detection by comparing F1 scores of tag propagation with and without the language model. The tags that are a composition of two words, such as 'high_tail', or that do not exist in the dictionaries like 'v-head', were tested without applying the language model, since these tags are classified as out-of-vocabulary (OOV). In most cases, the performance was better when the language model was applied. 'Bar', 'wheel', and 'dart' showed approximately 0.01 increase in F1 score, while 'propeller' had the same F1 score before and after using the language model. In this experiment, the improvement in performance before and after applying the language model was minor because the text annotation of the tested dataset was all deeply related to the shape of the 3D models. Therefore, few tags were deleted as anomalies and even the deleted ones were most likely related to the model. In the case study section, we tested the language model in a more anomaly-prone, real-world dataset.

In order to ensure that irrelevant models do not appear in the search results due to incorrect tag propagation, we tested the iterative search method by specifying a desired shape from the list of models. The iterative search method deletes models that are not similar to the selected filter shape (filter-out), and add similar ones (filter-in). That is, filter-out is a method to increase precision, and filter-in is to increase recall. We experimented whether there is an increase in both precision and recall. After searching for a given tag, filter-in and filter-out were performed together using a local part of the randomly selected ground truth model. As a result, filtering increases the precision-recall values in all test cases (Figure 6). Since the fixed number of models are added (up to 30) or removed (10 models) per filter operation, the fixed threshold of the filter becomes the upper bound of the performance improvement. However, it is meaningful in that users can improve the quality of the search results by taking part in a search process through simple interactions.

6 CASE STUDY

We conducted a case study to test how tag propagation responds to the real-world Google Poly data. Poly data was collected in five categories: chairs, airplanes, guitars, laptops, and knives through the API supported by Google. Among the 100 models searched in each category, we used those that are downloaded in .obj format. The

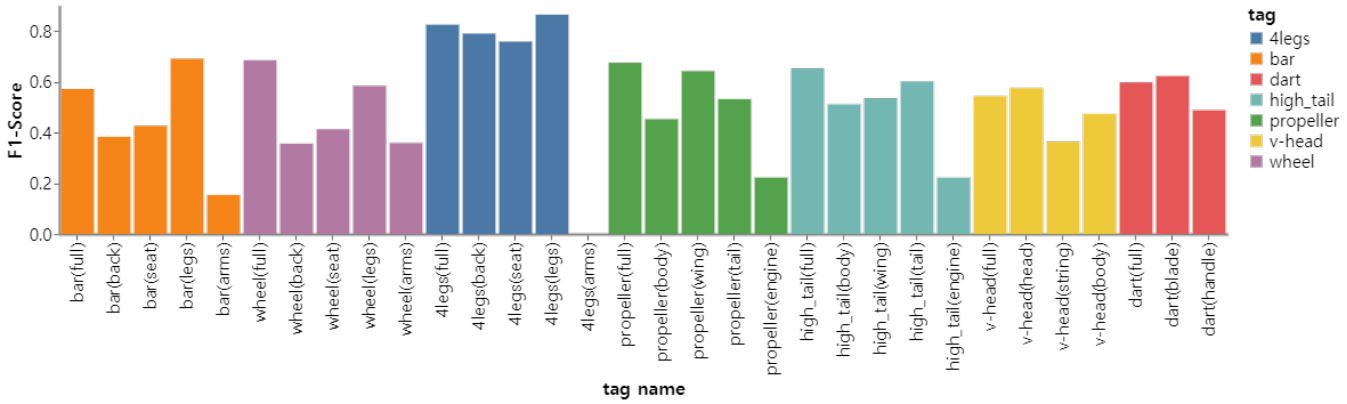

Figure 5: F1 scores of tag propagation using global and local features. Each color denotes the individual tag. 'bar', '4legs', 'v-head', and 'dart' performed best when using the local features, while 'wheel', 'propeller', 'high_tail' had the highest F1 score when using the global features.

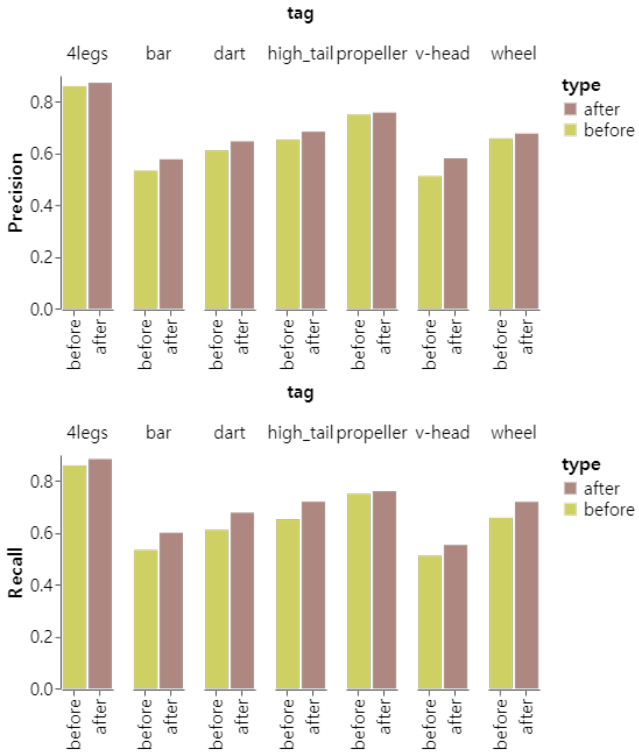

Figure 6: Changes of precision-recall values after filtering operation: for all the tags, both precision and recall increased after the iterative search.

Figure 7: Example cases in which per-part propagation works well. 'Wheel' and 'round-leg' tags are propagated to the chair models whose overall shapes are different but legs are similar. Similarly, 'air-wheel' tag is sent to the airplane models because of the wheels in their bodies. The failure cases occur when PolySquare selects a wrong part for propagation or extracted features of the part do not show the expected characteristics.

search results by keyword in Google Poly included many models unrelated to the search term. Since the segmentation model that we used have to know the category of an object, it was difficult to use local features of general objects in Poly data. Therefore, we conducted a case study on shape retrieval, iterative search, and language analysis with Google Poly data, and discovered cases about per-part propagation using ShapeNet data. We normalized the positions and the sizes of the Poly data and extracted features. The tags on the models contained words in the titles, descriptions, and hash tags. The collected Poly data can be explored on the implemented search interface.

Language model found anomalies in Poly data. Since Poly's text data contains a lot of information that is not related to the shape of the 3D model, we measured the changes before and after language analysis. On average, about 21.4% of all propagated

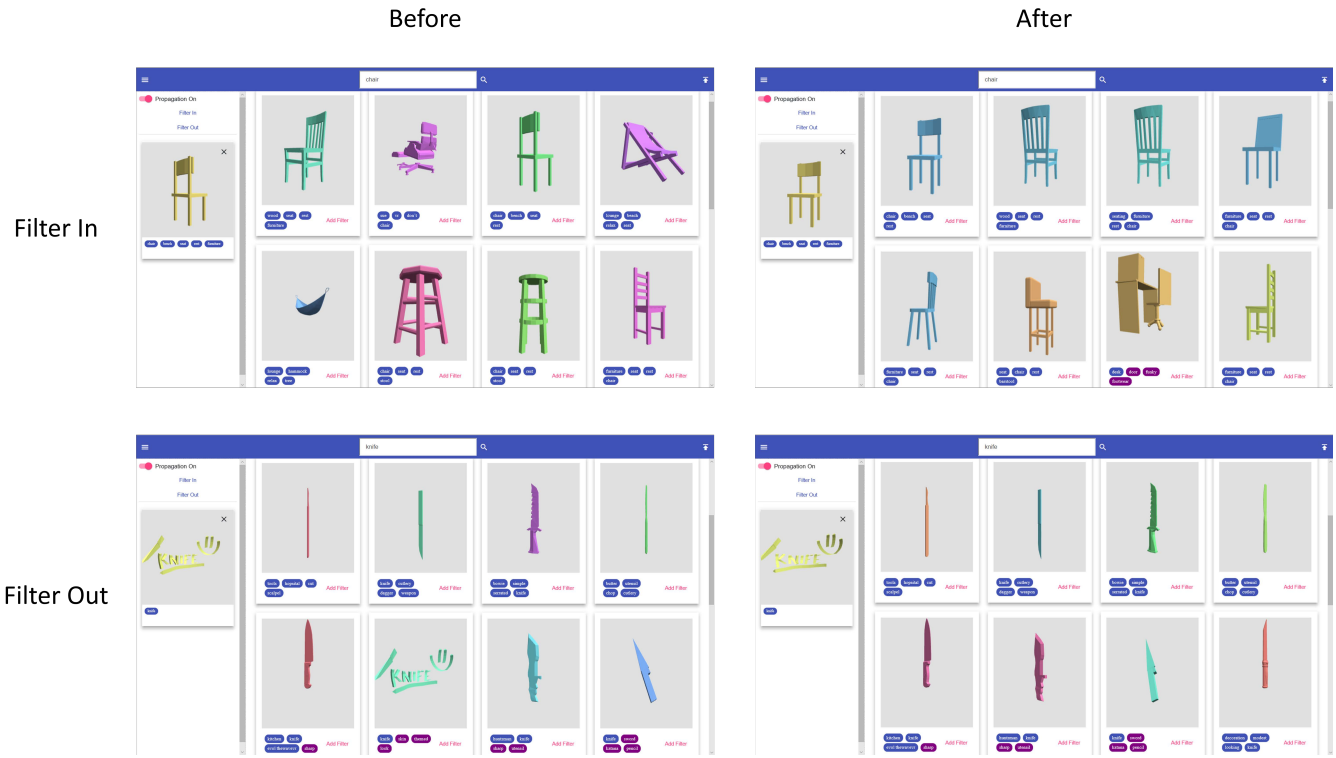

Figure 8: Example cases for filtering operation. The above picture is the screen capture of before and after the filter-in operation for a keyword 'chair'. The picture below is before and after the filter-out operation for a keyword 'knife'. The filter object is in the filter panel on the left of the page. As a result of the filter-in, chairs similar to the filter are placed on top of the page, and as a result of the filter-out, the 3D text 'knife' is removed.

tags were classified as anomaly, and 7% were classified as out-of-vocabulary (OOV) words. The rest of the tags were classified as safe and were used for the actual search. Grammar terms, such as prepositions and pronouns, that are not related to the properties of an object, were also treated as OOV. Also, for the use of the language model, this study did not deal with languages other than English. Examples of tags in Poly data classified as OOV are: OBJ1, Ataboy, FormFonts, Guiatr-Guiar, SPEECH2019, TheWaveVR, angelblade, magicavoxel, Busget, Speeach, etc. These words include the names of the 3D model studios, typing errors, and made-up words. Composite words without a space in-between were also excluded during the case study since splitting them takes additional effort and is out of scope of this work. In the future, we could develop into Fasttext [3] so that we can better respond to OOVs and rare words.

Poly data revealed various anomaly patterns than in ShapeNet data. Words classified as anomaly have a lexical meaning but are separated from the other tags of the object. They are likely to contain local features that are specific to the originated object and not likely to be included in other objects. In Poly data, the model 'Stool' originally had tags 'stool, seat, rest, chair'. After the propagation, the model additionally got safe tags 'furniture' and 'surface', while the tags 'pawn', 'chess' were classified as anomaly. However, one of the failure cases would be when the words 'table, dining' are added to the safe tag list. These tags are likely to be classified as safe, since they are related to the other tags ('chair, furniture') even if they are not directly related to the object (stool). Another failure case is when the group of tags is divided into two categories. If the number of tags in one category is greater than the other, the category with a small number of tags is assumed to be anomalous. For example, the model 'pixel sunglasses' had 'space', 'station', and 'weapon' as propagated tags, which are not related to the sunglasses. The tag 'sunglasses' and other related tags were treated as anomaly, because the number of unrelated tags was greater than the related ones. We believe that this problem could be improved once we know a direct relationship between the tags and the parts of the object.

Iterative search improved the quality of search results. Before propagating tags in Poly data, some of the chair models did not match the result when searching for a 'seat', since only 16 objects contained the tag 'seat'. After propagating tags, 38 chairs and 12 other objects got the tag 'seat'. We filtered out the unrelated ping-pong ball, and filtered in the relevant shape of chair (chairs with bars in the back) among the objects in the result. (Figure 8—Filter-In). As a result, chairs with similar shapes appear at the beginning of the search result, and ball-like shapes disappeared from the list. We found additional cases of finding the desired object through the use of filtering operation. In Figure 8—Filter-Out, searching for a

keyword 'knife' with the propagated tags contained the unwanted 3D text labeled 'knife'. After filtering the result using the knife model, the frying pan and the 3D text disappeared from the list, and only the knives appeared at the top of the search result.

Per-part propagation spread tags related to local features better. We expected that the per-part propagation process would be useful when there were tags that needed to be propagated between models whose overall shapes were different but shared local features. Since our dataset had only few tags that represented local features, we defined additional five tags: '6-in-line' for guitar tuners, 'double-head' for guitars, 'round-leg' and 'armrest' for chairs, and 'air-wheel' for airplanes. For each of these tags, we picked a model among all the models that matched with the tag, and attached the tag to it to act as the originator. The models that looked the most distinctive were selected so that the full-body propagation would not propagate the tags successfully. Because our language model did not work well with composite words such as double-head, we deactivated it for this study. The clipping threshold was set more leniently than the preceding evaluation to observe more propagation. After running the tag propagation pipeline, we found that three tags ('wheel', 'round-leg', and 'air-wheel') were correctly propagated to the models whose overall shapes were largely different from the originators. Of the 10 models that got the tag 'wheel', 9 of them did have legs with wheels. Unfortunately, the tags '6-in-line', 'double-head', and 'armrest' did not get propagated properly. PolySquare skipped them during per-part propagation because the parts failed to pass our threshold. There also were failure cases in which PolySquare chose wrong parts for per-part propagation. The failure example on the first row of Figure 7 got the tag 'wheel' because the backrest was selected for per-part propagation instead of the legs. Another type of failure case can happen when the feature extraction model incorrectly determines that the parts are similar, as seen on the second row of Figure 7.

7 LIMITATIONS AND FUTURE WORK

The object segmentation model in PolySquare can only be trained to segment a certain category of object and cannot be used on general objects. Since local features are extracted from the segmentation results, the quality of features and consequently the performance of tag propagation can be improved with a better deep learning segmentation model.

Also in many cases, per-part propagation found similar objects that were already found by full-body propagation. If tested on a dataset in which the tags about local features were more prevalent, the role of per-part propagation is expected to increase significantly.

When determining which local part represents prominent characteristics of the object during per-part propagation, we considered the distance to the nearest neighbors. However, there could be better, more intelligent ways. For example, a classifier could be trained to decide the correspondence between tags and parts, or tags could be assigned to each part independently, not just to the whole object.

Finally, there might be properties other than shape (e.g., the texture of the object or the positional relationship between the parts) that could be considered for tag propagation.

In future research, PolySquare could be expanded to encompass all the various factors mentioned above. Furthermore, as more filtering results from users pile up, PolySquare could learn from these information and improve the propagation result, becoming a smarter interface. If PolySquare develops into a multi-modal system, it will be possible to analyze and retrieve the relationship between 2D sketches, 3D shapes, text tags, and many other information. Also, the way of users directly participating in improving search results and actively reflecting preferences can be used in many other intelligent interfaces.

8 CONCLUSION

We presented PolySquare, a search engine utilizing the tag propagation method with local features. PolySquare aims to propagate tags even when the models share shape properties only in part, while trying to prevent incorrect propagation. To achieve this purpose, PolySquare introduced the four steps of the pipeline: full-body propagation, part segmentation, per-part propagation, and iterative search (filtering method). Using the similarities of local features along with the global ones, PolySquare also attached tags to the models that are not entirely, but partially, similar. Since not all parts of the object have notable characteristics, PolySquare selected a dominant local feature of the model and used the selected features in per-part propagation. To reduce the irrelevant search results caused by invalid propagation, PolySquare used a language model at the algorithm level, and allowed users to filter the results in a search interface. By measuring the precision and recall in evaluation, we presented that tag propagation works well using global and local features. In per-part propagation, PolySquare attached tags to similar objects that full-body propagation had never found before. As a final step, the iterative search and the language model could increase robustness of the search results. By testing PolySquare with 3D objects downloaded from Google Poly, we showed that it could also be applied to actual wild dataset.

ACKNOWLEDGMENTS

This work was supported by Samsung Electronics Co., Ltd.

REFERENCES

[1] Armen Avetisyan, Angela Dai, and Matthias Nießner. 2019. End-to-End CAD Model Retrieval and 9DoF Alignment in 3D Scans. *ArXiv* abs/1906.04201 (2019).
[2] Ronald T. Azuma. 1997. A Survey of Augmented Reality. *Presence: Teleoper. Virtual Environ.* 6, 4 (Aug. 1997), 355–385. https://doi.org/10.1162/pres.1997.6.4.355
[3] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching Word Vectors with Subword Information. *arXiv preprint arXiv:1607.04606* (2016).
[4] Benjamin Bustos, Daniel Keim, Dietmar Saupe, Tobias Schreck, and Dejan Vranić. 2006. An experimental effectiveness comparison of methods for 3D similarity search. *International Journal on Digital Libraries* 6, 1 (01 Feb 2006), 39–54. https://doi.org/10.1007/s00799-005-0122-3
[5] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. 2015. *ShapeNet: An Information-Rich 3D Model Repository.* Technical Report arXiv:1512.03012 [cs.GR]. Stanford University — Princeton University — Toyota Technological Institute at Chicago.
[6] Siddhartha Chaudhuri, Evangelos Kalogerakis, Stephen Giguere, and Thomas Funkhouser. 2013. Attribit: Content Creation with Semantic Attributes. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology (UIST '13)*. ACM, New York, NY, USA, 193–202. https://doi.org/10.1145/2501988.2502008
[7] Ding-Yun Chen, Xiao-Pei Tian, Yu-Te Shen, and Ming Ouhyoung. 2003. On Visual Similarity Based 3D Model Retrieval. *Comput. Graph. Forum* 22 (09 2003), 223–232. https://doi.org/10.1111/1467-8659.00669
[8] Blender Online Community. 2018. *Blender - a 3D modelling and rendering package.* Blender Foundation, Stichting Blender Foundation, Amsterdam. http://www.

blender.org
[9] Ritendra Datta, Weina Ge, Jia Li, and James Z Wang. 2007. Toward bridging the annotation-retrieval gap in image search. *IEEE MultiMedia* 14, 3 (2007), 24–35.
[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR* abs/1810.04805 (2018). arXiv:1810.04805 http://arxiv.org/abs/1810.04805
[11] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *KDD*.
[12] Lubin Fan, Ruimin Wang, Linlin Xu, Jiansong Deng, and Ligang Liu. 2013. Modeling by Drawing with Shadow Guidance. *Comput. Graph. Forum* 32 (2013), 157–166.
[13] Yutong Feng, Yifan Feng, Haoxuan You, Xibin Zhao, and Yue Gao. 2018. MeshNet: Mesh Neural Network for 3D Shape Representation. In *AAAI*.
[14] Yifan Feng, Zhang Zizhao, Xibin ZâĂĘhao, Rongrong Ji, and Yue Gao. 2018. GVCNN: Group-View Convolutional Neural Networks for 3D Shape Recognition. 264–272. https://doi.org/10.1109/CVPR.2018.00035
[15] Thomas Funkhouser, Patrick Min, Michael Kazhdan, Joyce Chen, Alex Halderman, David Dobkin, and David Jacobs. 2003. A Search Engine for 3D Models. *ACM Trans. Graph.* 22, 1 (Jan. 2003), 83–105. https://doi.org/10.1145/588272.588279
[16] Corey Goldfeder and Peter Allen. 2008. Autotagging to Improve Text Search for 3D Models. In *Proceedings of the 8th ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL '08)*. ACM, New York, NY, USA, 355–358. https://doi.org/10.1145/1378889.1378950
[17] C. Goldfeder, Haoyun Feng, and P. Allen. 2008. SHRECâĂŹ08 entry: Training set expansion via autotags. In *2008 IEEE International Conference on Shape Modeling and Applications*. 233–234. https://doi.org/10.1109/SMI.2008.4547983
[18] Alexander Grabner, Peter M. Roth, and Vincent Lepetit. 2018. 3D Pose Estimation and 3D Model Retrieval for Objects in the Wild. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018), 3022–3031.
[19] M. Guillaumin, T. Mensink, J. Verbeek, and C. Schmid. 2009. TagProp: Discriminative metric learning in nearest neighbor models for image auto-annotation. In *2009 IEEE 12th International Conference on Computer Vision*. 309–316. https://doi.org/10.1109/ICCV.2009.5459266
[20] Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. 2019. MeshCNN: a network with an edge. *ACM Trans. Graph.* 38 (2019), 90:1–90:12.
[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), 770–778.
[22] S. Hu, J. Cai, and Y. Lai. 2018. Semantic Labeling and Instance Segmentation of 3D Point Clouds using Patch Context Analysis and Multiscale Processing. *IEEE Transactions on Visualization and Computer Graphics* (2018), 1–1. https://doi.org/10.1109/TVCG.2018.2889944
[23] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014).
[24] Leif Kobbelt, P. Schrder, Michael Kazhdan, Thomas Funkhouser, and Szymon Rusinkiewicz. 2003. Rotation Invariant Spherical Harmonic Representation of 3D Shape Descriptors. *Proc 2003 Eurographics* vol. 43 (07 2003).
[25] Hema Swetha Koppula, Abhishek Anand, Thorsten Joachims, and Ashutosh Saxena. 2011. Semantic Labeling of 3D Point Clouds for Indoor Scenes. In *Proceedings of the 24th International Conference on Neural Information Processing Systems (NIPS'11)*. Curran Associates Inc., USA, 244–252. http://dl.acm.org/citation.cfm?id=2986459.2986487
[26] J. Richard Landis and Gary G. Koch. 1977. The Measurement of Observer Agreement for Categorical Data. *Biometrics* 33, 1 (1977), 159–174. http://www.jstor.org/stable/2529310
[27] Bo et al. Li. [n. d.]. A Comparison of 3D Shape Retrieval Methods Based on a Large-scale Benchmark Supporting Multimodal Queries. *Comput. Vis. Image Underst.* 131, C ([n. d.]), 1–27.
[28] Lihaixiong Li, S.-S Zhang, X.-L Bai, and R. Huang. 2013. Auto-tagging algorithm of 3D CAD models. *Jisuanji Jicheng Zhizao Xitong/Computer Integrated Manufacturing Systems, CIMS* 19 (07 2013), 1484–1489.
[29] J. MacQueen. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*. University of California Press, Berkeley, Calif., 281–297. https://projecteuclid.org/euclid.bsmsp/1200512992
[30] Patrick Min, Michael Kazhdan, and Thomas Funkhouser. 2004. A Comparison of Text and Shape Matching for Retrieval of Online 3D Models. In *Research and Advanced Technology for Digital Libraries*, Rachel Heery and Liz Lyon (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 209–220.
[31] Ryutarou Ohbuchi and Shun Kawamura. 2009. Shape-Based Autotagging of 3D Models for Retrieval. In *Proceedings of the 4th International Conference on Semantic and Digital Media Technologies: Semantic Multimedia (SAMT '09)*. Springer-Verlag, Berlin, Heidelberg, 137–148. https://doi.org/10.1007/978-3-642-10543-2_14
[32] Laura Papaleo and Leila De Floriani. 2010. Manual Segmentation and Semantic-based Hierarchical Tagging of 3D models. *Eurographics Italian Chapter Conference 2010*, 25–32. https://doi.org/10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2010/025-032
[33] Jeffrey Pennington, Richard Socher, and Christoper Manning. 2014. Glove: Global Vectors for Word Representation. *EMNLP* 14, 1532–1543. https://doi.org/10.3115/v1/D14-1162
[34] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proc. of NAACL*.
[35] Federico Ponchio, Marco Callieri, Matteo Dellepiane, and Roberto Scopigno. 2019. Effective Annotations Over 3D Models. *Computer Graphics Forum* (05 2019). https://doi.org/10.1111/cgf.13664
[36] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. 2016. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), 77–85.
[37] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J. Guibas. 2017. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *NIPS*.
[38] Olga Russakovsky, Jun Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Li Fei-Fei. 2014. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision* 115 (2014), 211–252.
[39] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, Andreas Holzbach, and Michael Beetz. 2009. Model-based and Learned Semantic Object Labeling in 3D Point Cloud Maps of Kitchen Environments. In *Proceedings of the 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'09)*. IEEE Press, Piscataway, NJ, USA, 3601–3608. http://dl.acm.org/citation.cfm?id=1733023.1733323
[40] Manolis Savva, Angel X. Chang, and Pat Hanrahan. 2015. Semantically-Enriched 3D Models for Common-sense Knowledge. *CVPR 2015 Workshop on Functionality, Physics, Intentionality and Causality* (2015).
[41] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik G. Learned-Miller. 2015. Multi-view convolutional neural networks for 3d shape recognition. In *Proc. ICCV*.
[42] Julien Valentin, Vibhav Vineet, Ming-Ming Cheng, David Kim, Jamie Shotton, Pushmeet Kohli, Matthias NieÃ§ner, Antonio Criminisi, Shahram Izadi, and Philip Torr. 2015. SemanticPaint: Interactive 3D Labeling and Learning at your Fingertips. *ACM Transactions on Graphics (TOG)* 34 (November 2015).
[43] Christopher P. Wadsworth. 1971. Semantics and pragmatics of lambda-calculus.
[44] Fang Wang, Le Kang, and Yi Li. 2015. Sketch-based 3D shape retrieval using Convolutional Neural Networks. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), 1875–1883.
[45] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. 2018. Dynamic Graph CNN for Learning on Point Clouds. *ArXiv* abs/1801.07829 (2018).
[46] Kun Xu, Kang Chen, Hongbo Fu, Wei-Lun Sun, and Shi-Min Hu. 2013. Sketch2Scene: Sketch-based Co-retrieval and Co-placement of 3D Models. *ACM Transactions on Graphics* 32, 4 (2013), to appear.
[47] Li Yi, Vladimir G. Kim, Duygu Ceylan, I-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas Guibas. 2016. A Scalable Active Framework for Region Annotation in 3D Shape Collections. *ACM Trans. Graph.* 35, 6, Article 210 (Nov. 2016), 12 pages. https://doi.org/10.1145/2980179.2980238
[48] Zhirong Wu, S. Song, A. Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and J. Xiao. 2015. 3D ShapeNets: A deep representation for volumetric shapes. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1912–1920. https://doi.org/10.1109/CVPR.2015.7298801