Autotagging to Improve Text Search for 3D Models

Corey Goldfeder, Peter Allen

Abstract—**Text search on 3D models has traditionally worked poorly, as text annotations on 3D models are often unreliable or incomplete. In this paper we attempt to improve the recall of text search by automatically assigning appropriate tags to models. Our algorithm finds relevant tags by appealing to a large corpus of partially labeled example models, which does not have to be preclassified or otherwise prepared. For this purpose we use a copy of Google 3DWarehouse, a database of user contributed models which is publicly available on the Internet. Given a model to tag, we find geometrically similar models in the corpus, based on distances in a reduced dimensional space derived from Zernike descriptors. The labels of these neighbors are used as tag candidates for the model with probabilities proportional to the degree of geometric similarity. We show experimentally that text based search for 3D models using our computed tags can work as well as geometry based search. Finally, we demonstrate our 3D model search engine that uses this algorithm and discuss some implementation issues.**

I. Introduction

Historically, interest in 3D models has largely been confined to expert users such as video game programmers, computer animators, and engineers prototyping new products. Recently, however, a number of new applications have greatly expanded the pool of users who deal with 3D models. Free, user-friendly modeling tools such as Google Sketchup and new casual uses for 3D models such as modeling one's home for Google Earth or staking out a digital existence in Second Life have resulted in an increased popular interest in 3D models and to an explosive growth in the amount of 3D data available on the web. As these datasets continue to expand, existing methods of searching databases of 3D models are proving insufficient.

There are three dominant approaches for 3D model search. The simplest method is text search on descriptive tags associated with each model. While this form of search is the most natural from a user perspective, its power is limited by a dependence on the accuracy of the descriptions, and more fundamentally by the requirement that such annotations even exist. Min, Kazhdan, and Funkhouser experimentally confirmed that searching on text alone is a poor retrieval strategy for 3D models drawn from the web [1], although in specialized databases one might expect somewhat better results.

Manuscript received December 9, 2007. This work was supported in part by a National Defense Science and Engineering Graduate Fellowship.

Corey Goldfeder is a PhD student in Computer Science at Columbia University, NY; coreyg@ cs.columbia.edu

Peter Allen is a Professor of Computer Science at Columbia University; allen@ cs.columbia.edu

A second approach is to allow a user to submit a sketch as a query. In [2] users input 2D sketches as search queries, which are compared to 2D renderings of the object taken from viewpoints sampled on the enclosing sphere, while in [3], 2D queries are supported for only a small number of viewpoints. In both cases, only the projected contours of the 3D model are considered for matching, and as such any features which do not lie on the visual hull cannot be used in the retrieval. Using suggestive contours [4] might break the dependence on the visual hull, but it has not been attempted. The major disadvantages of the sketch based approach are that it demands a certain amount of artistic ability from users, and that composing a query is both slower than other methods and more difficult to repeat consistently.

Finally, many search algorithms [5] [6] can take an initial 3D model as a query and retrieve other similar models. Search is done in an iterative fashion – an initial query model is selected via some other search method, such as text search, and a number of similar models are retrieved. The user can then select any of the retrieved models as the source of a new query. While this paradigm is well suited for browsing a database, it has serious limitations for searching, as the only way to break out of the very local view of the database is to select an entirely new 3D model as a query.

It is well known that human notions of similarity are not purely geometric [7] and that there is a *semantic gap* between what users intend when they search and what geometric shape descriptors can deliver. Recent work in shape matching has focused on bringing human intelligence into the loop. The most common approach has been to use relevance feedback [8][9], which has been applied with great success in other fields, particularly 2D image search. Relevance feedback is a supervised learning technique, and as such inherits the limitation of supervised methods, which is that learning new classes will generally require new training.

We propose a different method of harnessing human intelligence, by making use of existing textual annotations and keywords associated with 3D models. We use geometric relationships between models to propagate information between similar models and improve the saliency of the text annotations. Our goal is to improve the precision and recall of text based 3D model search to the point that it is comparable to geometric shape descriptors. This is desirable because text search is a very natural search interface.

In the past, text annotations on 3D models have been largely dismissed by shape search researchers as being unreliable and incomplete, and therefore of limited use. In this paper, we consider how the availability of very large

databases changes this equation and hypothesize that given a large enough corpus the semantic information contained in the whole can be more than the sum of its parts. We use a corpus consisting of user contributed models freely available on the Internet. Compared to similar recent work in tagging 2D images [10], our algorithm for 3D autotagging has the advantage of being unsupervised. This means that the tags that our algorithm can produce are not limited to a small predefined set. Additionally, we do not require an explicit training stage.

The remainder of this paper is structured as follows. In Section II we explain our algorithm and describe our dataset. In Section III we describe our implementation of shape similarity. Section IV shows experimental validation of our algorithm. Section V discusses our shape search system, which uses this algorithm. Section VI presents conclusions and future work.

II. AUTOTAGGING

A. *The algorithm*

Given an unlabeled 3D model ω, we wish to assign to ω a set of text tags from the set of all possible tags

$$\Lambda = \{\lambda^1, \lambda^2 \ \ldots \ \lambda^n\} \tag{1}$$

Specifically, for each tag λ^i we wish to assign a confidence value $P(\lambda^i, \omega)$ which we interpret as the probability that λ^i is a relevant tag for ω. We informally define relevancy to mean that a conscientious annotator would apply tag λ^i to model ω. We do not place any constraints on the set of possible tags. Λ is merely a notational convenience borrowed from the vector space model of information retrieval, which is discussed further in Section IV.

To tag ω, we make use of a corpus of known models

$$\Omega = \{\omega_1, \omega_2 \ \ldots \ \omega_n\} \tag{2}$$

Each model ω_x in the corpus has associated tag probabilities $P(\lambda^i, \omega_x)$ for the tags in Λ, most of which will be zero. We make two simplifying assumptions.

First, we assume that two physical objects with the same geometry (excluding scale) are the same. 3D models are discrete approximations of physical objects, and so two models that represent the same physical geometry may not be identical. We use the $\approx$ operator to denote that two models are intended to represent the same physical geometry.

Our second assumption is that models that represent the same physical geometry should have the same tags. Formally,

$$\omega_x \approx \omega_y \Rightarrow P(\lambda^i, \omega_x) = P(\lambda^i, \omega_y) \tag{3}$$

Let $P(\omega \approx \omega_x)$ be the probability that our untagged model ω and some model ω_x represent the same physical geometry. In that case,

$$P(\lambda^i, \omega) = P(\omega \approx \omega_x) \wedge P(\lambda^i, \omega_x) \tag{4}$$

Intuitively this just means that λ^i is a good tag for ω if ω and ω_x represent the same physical object and λ^i is a good tag for ω_x. We can use (4) to propagate tags to an untagged model.

Our algorithm is based on the idea that the geometric distance between ω_x and ω_y can be interpreted as an approximation of $P(\omega_x \approx \omega_y)$. We assign a distance $D'(\omega, \omega_x)$ from model ω to every model ω_x in Ω, based on their geometric dissimilarity. We use the Zernike descriptors [5], described in Section III, as our geometric distance function, but in principle any reasonable shape descriptor should suffice. We choose an appropriate threshold $\tau(\omega)$ and define

$$\mathrm{D}(\omega, \omega_x) := \frac{D'(\omega, \omega_x)}{\tau(\omega)} \tag{5}$$

Note that the threshold is allowed to be a function of the model, which allows for adaptively defining the threshold based on the density of models in a given portion of the descriptor space. We define a set of neighbors $N(\omega)$ as

$$N(\omega) = \{\omega_x | D(\omega, \omega_x) \in [0, 1]\} \tag{6}$$

For our method to perform well, we will need $N(\omega)$ to consist mostly of models that represent physical objects very similar to ω, but we do not need the set to be very large. We therefore prefer small values of $\tau(\omega)$ in order to capture the higher precision most shape descriptors are capable of at small recall values.

As we have suggested above, we set

$$P(\omega \approx \omega_x) = 1 - D(\omega, \omega_x) \tag{7}$$

for the models in $N(\omega)$. It follows from (4) that given a single neighbor ω_x

$$P(\lambda^i, \omega) = (1 - D(\omega, \omega_x)) P(\lambda^i, \omega_x) \tag{8}$$

Considering the full set of neighbors, we can generalize (4) to

$$P(\lambda^i, \omega) = \bigcup_{n=1}^{|N|} P(\omega \approx \omega_n) \wedge P(\lambda^i, \omega_n) \tag{9}$$

in which case (8) becomes

$$P(\lambda^i,\omega)=\sum_{n=1}^{|N|}(-1)^{n-1}\sum_{\substack{S\subset\{1,\ldots|N|\}\\|S|=n}}\prod_{s\in S}(1-D(\omega,\omega_s))P(\lambda^i,\omega_s) \tag{10}$$

We note that if ω has no neighbors closer than the threshold distance, all of the tag probabilities will be zero. In practical terms, this means that our autotagging algorithm may not successfully tag every input model.

B. *The corpus*

For Ω we use a large, tagged corpus of 3D models downloaded from the Internet. We cannot assume that the models have useful geometric properties such as being watertight or even connected, and so we treat them as polygon soups. We also do not assume that the models are particularly well tagged, and so we assign an initial $P(\lambda^i,\omega)$ to every tag on every model that represents our subjective confidence in how well the dataset is tagged.

Our corpus consists of 192,343 models downloaded from Google 3DWarehouse. Each model has a title, a set of keywords, and a text description, although for many models one or more is blank. A good deal of the text is composed of nonsense words or blatantly incorrect labels that have nothing to do with the models they are applied to. We found that the title and keyword fields were usually more reliable than the description, and so we assumed $P(\lambda^i,\omega)=0.7$ for tags drawn from the title and keywords and $P(\lambda^i,\omega)=0.5$ for tags drawn from the words of the description. Tags were stemmed using WordNet [11] and words that appeared ona list of stop words were ignored.

Although we have implied a separation between the query model ω and the corpus, in practice we make no such distinction. As the corpus is not specially prepared in any way, we can choose any model from 3DWarehouse itself as ω and improve its tags based on its neighbors; computing this for every model can be seen as a smoothing operation on the tags. Additionally, any new query model can be immediately added to the corpus, which allows the system to learn better tags with time.

III. Geometric Similarity

In order to find geometrically similar models, we need to define similarity of 3D objects. Many descriptors have been proposed for 3D, including the Spherical Harmonic descriptors [6] [12], Lightfield descriptors [2], Zernike descriptors [5] and Spherical Wavelet descriptors [13]. Due to the nature of our dataset, not all of these descriptors are equally appropriate. Many of the models in our corpus are architectural, where internal structure is significant, which rules out the use of Lightfield descriptors as they only capture the visual hull. Additionally, many models include a backdrop or a supporting plane, which would be problematic for the Spherical Wavelet functions or the [12] version of Spherical Harmonics, both of which capture only the maximal spherical extent function. The remaining two functions, the [6] version of Spherical Harmonics and the Zernike descriptors, are very similar. We use the Zernikes because they are more compact.

A. *Zernike descriptors*

Ignoring rotation, we can compare two functions in $\mathbf{R}^3$ by taking the square root of the inner product of their difference, which can be thought of as a measure of their overlap. This can have undesirable results for shape matching, because it is easy to construct intuitively similar shapes that cannot be aligned to have significant overlap. To deal with this, we can project the two functions into an orthonormal moment space, ignore higher order moments, and then take the norm, which has the effect of smoothing the functions and increasing their overlap. This is the approach of [14], using Krawtchouk moments.

Novotni and Klein observed that normalizing rotation is a difficult problem, and proposed making the moments comparison rotationally invariant by comparing only the energy levels at each frequency rather than the actual moment coefficients [5]. Their basis of choice was the 3D Zernike polynomials, originally derived by Canterakis [15]:

$$\begin{gathered}Z_{nl}^m(\mathbf{x})=R_{nl}(r)\cdot Y_l^m(\upsilon,\phi)\\ l\le n \qquad (n-l)=2k \qquad -l\le m\le l\end{gathered} \tag{11}$$

where Y_l^m is a spherical harmonic. The R_{nl} terms are radial polynomials, and we refer the interested reader to [5] for their definition. These functions form an orthonormal basis for the unit sphere, and so for a function contained within the unit sphere, its Zernike moments are

$$\Omega_{nl}^m:=\frac{3}{4\pi}\int_{|\mathbf{x}|\le 1}f(\mathbf{x})\overline{Z_{nl}^m(\mathbf{x})}d\mathbf{x} \tag{12}$$

and its Zernike descriptor is

$$F_{nl}:=|\Omega_{nl}| \tag{13}$$

where all we have done is taken the norms of the moments over the m index and collected the results in a vector. These descriptors are then compared using the L^2 norm, although the original geometric meaning as a measure of smoothed overlap is lost. Novotni and Klein recommended using moments up to $n=20$, which results in a descriptor with 121 components.

B. *Implementation*

We use Zernike descriptors with a few minor variations from [5]. Our calculations are done on a voxel grid of 128 voxels per side and we thicken our surfaces with a kernel 4 voxels wide. For scaling, we use 7 point Gaussian numerical integration to find the center of mass of a uniform mass

distribution on the surface of the object. Another integration gives us the mean distance and standard deviation from surface points to the center of mass. We scale so that the mean distance and 3 standard deviations fit within the unit sphere and clip anything that lies outside. Scaling in this fashion is robust to moderate changes in shape and to outliers. We voxelize our models using a fast software voxelizer which we wrote, and compute the descriptors using a tuned version of Novotni and Klein's publicly available reference implementation.

C. *Dimensionality reduction*

In analyzing the descriptors of our 192,343 models, we observed strong correlation between components of the descriptor with *n* indices of the same parity, and also that the correlation increases as *l* increases. (The latter effect is due to the band limiting imposed by voxelization). These correlations implied that we could reduce the dimensionality of the descriptor, and indeed Principal Components Analysis produced a 57 dimensional vector that preserved 99.9% of the variance of the original. All Zernike descriptor distances referred to in this paper were computed in the reduced space. Aside from the storage savings, PCA moves most of the variance into a few dimensions which enabled us to do real time nearest neighbor search as described in Section VI.

IV. Experimental Validation

To validate the quality of our automatically produced tags, we use the Princeton Shape Benchmark (PSB) [16]. We computed Zernike descriptors for every model in the PSB, matched them against the models in our 3DWarehouse corpus, and produced tags for the models using our algorithm. In computing these tags we treated the PSB as if it consisted of completely untagged models, ignoring the model classifications provided with the benchmark and any other text associated with the models. For $\tau(\omega)$ we used an adaptive threshold, which we defined as the radius of the hypersphere containing the first 15 nearest neighbors.

A. *Discriminative power*

Our first set of experiments is designed to test how *discriminative* our tags are, in the sense that models of the same class get similar tags, and models in different classes get dissimilar tags. We use the Vector Space Model [17] to define a distance between the tags of any two models. In the Vector Space Model, every possible tag $\lambda_i \in \Lambda$ is assumed to be an independent dimension, and a model's tags are represented as a vector in this Λ-space. Each model ω_x has an associated tag vector $\mathbf{\Xi}_x$

$$\mathbf{\Xi}_x = [W(\lambda^1, \omega_x), W(\lambda^2, \omega_x) \ \ldots \ W(\lambda^n, \omega_x)] \quad (14)$$

The component $W(\lambda^i, \omega_x)$ along each dimension is given using the "tag frequency, inverse document frequency," [18] or tf-idf, method:

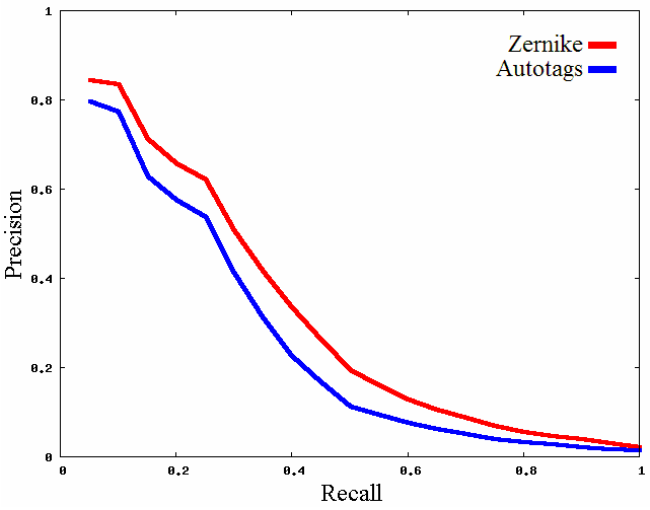


Fig. 1. The precision/recall graph for our automatically generated tags, as compared to the precision/recall of the geometric distance given by the Zernike descriptors.

$$W(\lambda^i, \omega_x) = \frac{freq(\lambda^i, \omega_x)}{\sum_k freq(\lambda^k, \omega_x)} \cdot \log \frac{|\Omega|}{|\{\omega_j : freq(\lambda^i, \omega_j) > 0)\}|} \quad (15)$$

where $freq(\lambda^i, \omega_x)$ is the number of times tag λ^i appears on model ω_x. The Λ-space distance between two models is taken to be 1 - the cosine of the angle between their tag vectors:

$$D_\Lambda(\omega_x, \omega_y) = 1 - \cos(\theta) = 1 - \frac{\mathbf{\Xi}_x \cdot \mathbf{\Xi} y}{|\mathbf{\Xi}_x||\mathbf{\Xi} y|} \quad (14)$$

If either model has no generated tags, we set the distance to be 1.0, which is the maximum possible distance using the cosine metric.

Using this cosine distance, we computed the distance matrix for the models of the PSB. Fig. 1 shows the precision/recall graph for our computed tag distances, as compared to the precision/recall for the Zernike descriptors. It is important to remember that the autotag values are for text search, while the Zernikes require an input 3D model. From Fig. 1 we can see that our algorithm captures *most* of the discriminative power of the underlying shape descriptor, and makes it accessible to text search without requiring the user to provide an initial 3D model as a query.

As a control, we compared the quality of our computed tags to the original tags that came with the PSB models, using the method of [1]. Like them, we used seven sources of text for each model, including the model's filename, original URL, text from the referring webpage, and synonyms from WordNet. We computed the cosine distance matrix for these tags in the same fashion as before. Fig. 2 shows how our computed tags compare to the original tags. The initial precision of our tags is significantly superior to that of the

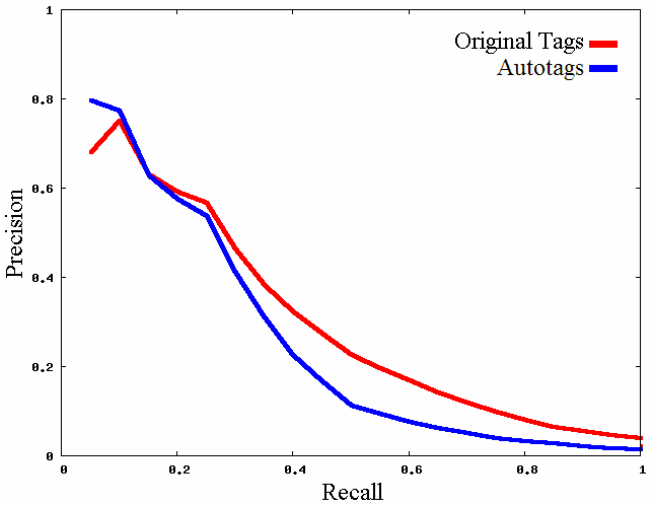


Fig. 2. The precision/recall graph for our automatically generated tags, as compared to the original PSB tags weighted by tf-idf.

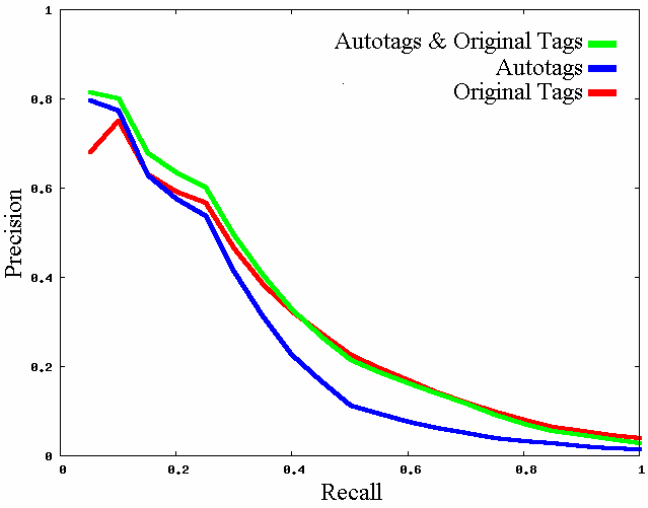


Fig. 3. The precision/recall for the original tags combined with our autotags is better than either set of tags individually.

original tags, although at greater recall the original tags still show more precision. It is clear that the quality of our automatically generated annotations is comparable to the quality of the original tags. However, the original tags and the computed tags are not identical, which suggested to us that it would be worthwhile to combine both sources of tags. Fig. 3 demonstrates how the combination of original and computed tags outperforms either tag source alone. In fact, the results are quite close to the precision/recall of the Zernike descriptors, as can be seen in Fig. 4. We feel that this result is strong evidence against the notion that text based search can never compete with other forms of 3D search such as 2D sketches and 3D query models.

B. *Search quality*

For our next experiment, we simulated some example searches on the original tags and on our autotags. The queries were chosen to map directly onto classes in the PSB classification, so that we could evaluate the precision and recall of the results.

Given a search query λ, we retrieved the models ω_x that were tagged with λ, ordered by descending $P(\lambda, \omega_x)$. For the original tags, we weighted all tags equally, since we have no probability information for them.

Fig. 5 shows the precision/recall for the queries "airplane," "swords" and "heads" where we have capped the recall at the point where there are no more models tagged with the query string, and so any further retrieval would be random. Note that the autotags have equal or greater precision to the original tags nearly always. More importantly, text search on the computed tags successfully recovered over 65% of the relevant results for each query. This is in strong contrast to the search on the original tags, for which 2 of the queries only retrieved 10% of the relevant models, and one retrieved 40%.

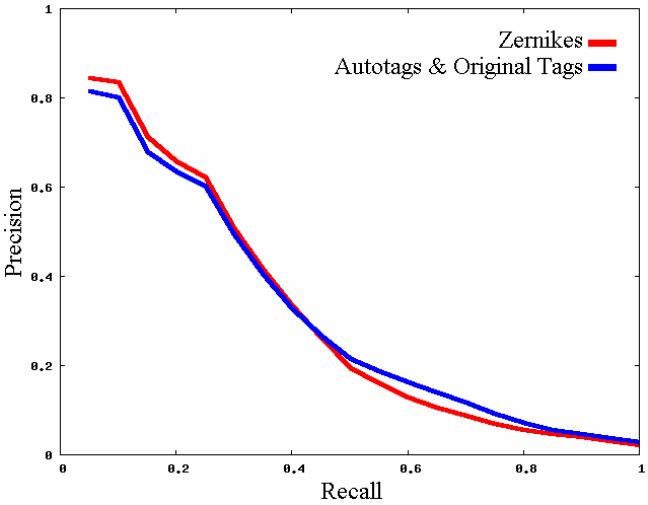


Fig. 4. The precision/recall for our autotags combined with the original tags is very similar to the precision/recall of the Zernike descriptors.

V. Search Engine

Based on the work described in this paper, we have designed a shape search engine that uses autotagging. Our search engine has access to copies of the 3DWarehouse and the PSB and can find models by their original tags or by their autotags. For any model in the database, our engine can also return the nearest neighbors in the Zernike sense.

In Section III.C we described how a PCA of the models in our database resulted in a 57 dimension linear subspace of the 121 dimensional Zernike descriptor space that preserved nearly all of the original variance. Aside from dimensionality reduction, PCA also packs as much variance as possible into the first few dimensions. We made use of this fact to build a very fast *k*-nearest neighbors implementation. This is the core of our search engine, since we need to find neighbors in Zernike descriptor space in order to do autotagging.

Our approach is based on [19]. To find all of the neighbors of *p* within radius *r* they first prune the space to a

hypercube with sides of $2r$, centered on p. To support this operation, they maintain n separate lists of the points, each sorted along one dimension. Pruning to a hypercube then reduces to rejecting any points with a distance greater than r in any one dimension, and then finding the intersection of n lists. The points which remained are then brute force searched, and those which lie outside of the radius r hypersphere are rejected. If k neighbors are not found, the algorithm can be run again with a larger value of r.

We mapped this algorithm to a PostgreSQL based database implementation. Instead of n lists, we maintain a table of n columns, where each row represents a single n-dimensional point. We also maintain an index on each column, which is algorithmically equivalent to maintaining a sorted list on each dimension.[1] To search for the nearest neighbors of a model described by point p in the reduced Zernike space, we can perform the entire [19] pruning algorithm as a single SELECT statement:

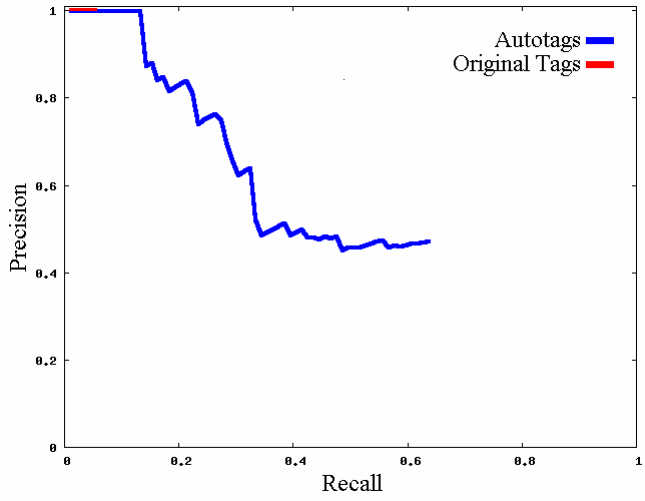
```
SELECT * FROM model
WHERE
(z1 BETWEEN p[1]  - r AND p[1] + r) AND
   ⋮
(z57 BETWEEN p[57]  - r AND p[57] + r)
AND distance(model, p) < r
ORDER BY distance(model, p);
```

where distance (model, p) is a stored procedure that gives the Euclidean distance between a row and p. Aside from the simplification of having the database do all the pruning and intersection work, this implementation makes use of the fact that, due to PCA, most of the variance is in the first few dimensions. PostgreSQL keeps statistics on the variance of each column and executes the BETWEEN clause on the columns with the most variance first. Columns with a higher variance are likely to have fewer neighbors within the search distance, and so most rows are pruned very early and do not need to be repeatedly considered for intersection. In practice, our implementation running on a 2.4 GHz Intel CPU can search 192,343 Zernike descriptors and return the 50 nearest neighbors of a query descriptor in approximately 3 seconds.

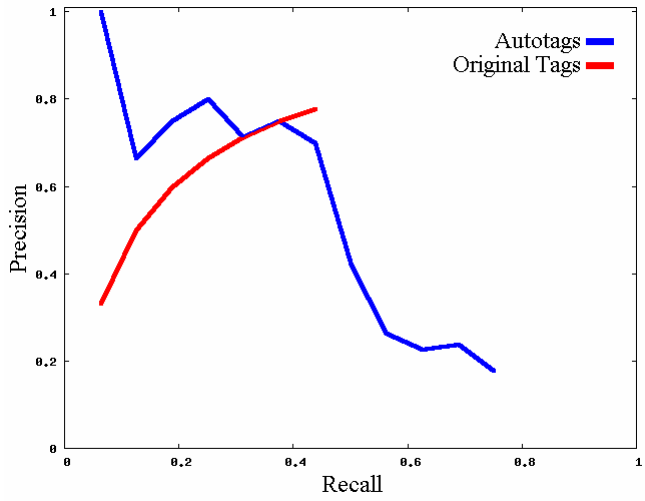
Fig. 6 shows the user interface of our search engine, which combines text and geometry search. Models are shown with their original and computed tags. Users can search by filename, original tags, and computed tags. Hovering over a thumbnail pops up a context menu that gives access to both the original 3D file and the voxelized version used in the computation of the Zernike descriptors. Clicking on a thumbnail retrieves the model's geometric neighbors. The user can separately select which collections of models are to be searched and which are to be considered part of the corpus for autotagging.

[1] We recommend PostgreSQL over MySQL for this algorithm, as the current version of MySQL limits the number of indices on a table to 64.

Search for "airplane"



Search for "sword"



Search for "head"

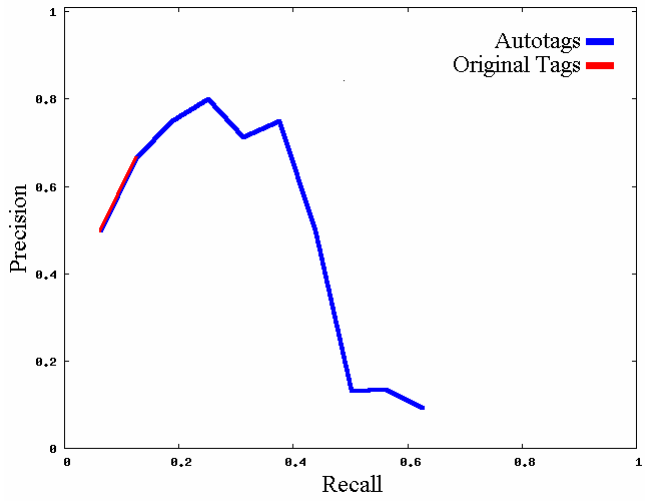


Fig. 5. Precision/recall for 3 searches. The recall is capped at the point where there are no more tagged models (and any further retrieval would be random). The precision of the autotags equals or exceeds that of the original tags in nearly all instances, and the autotags can retrieve many times more models than the original tags, since only a fraction of the models have relevant original tags. Note to readers: it may be helpful to zoom in to better see the areas where the red and blue lines overlap.

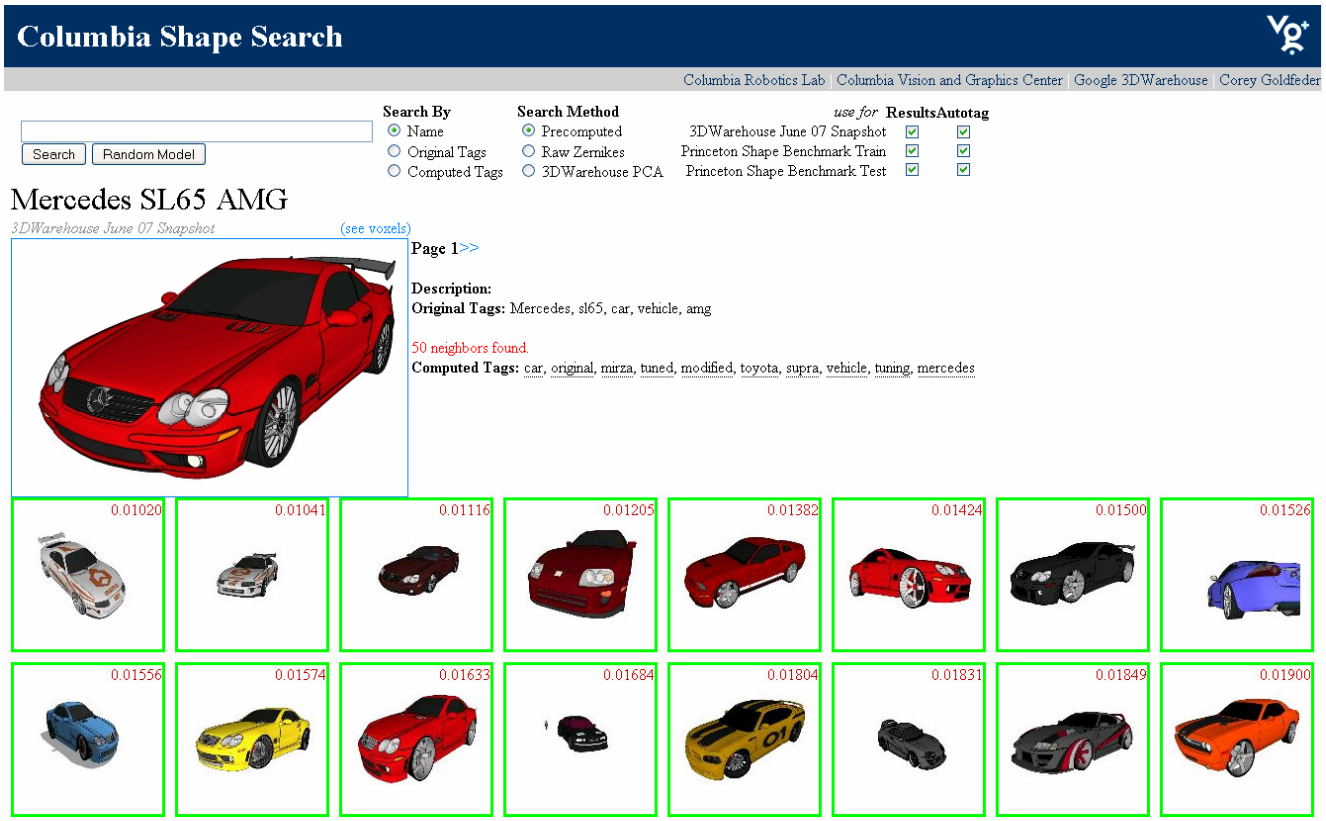


Fig. 6. Our web based autotagging and shape search interface.

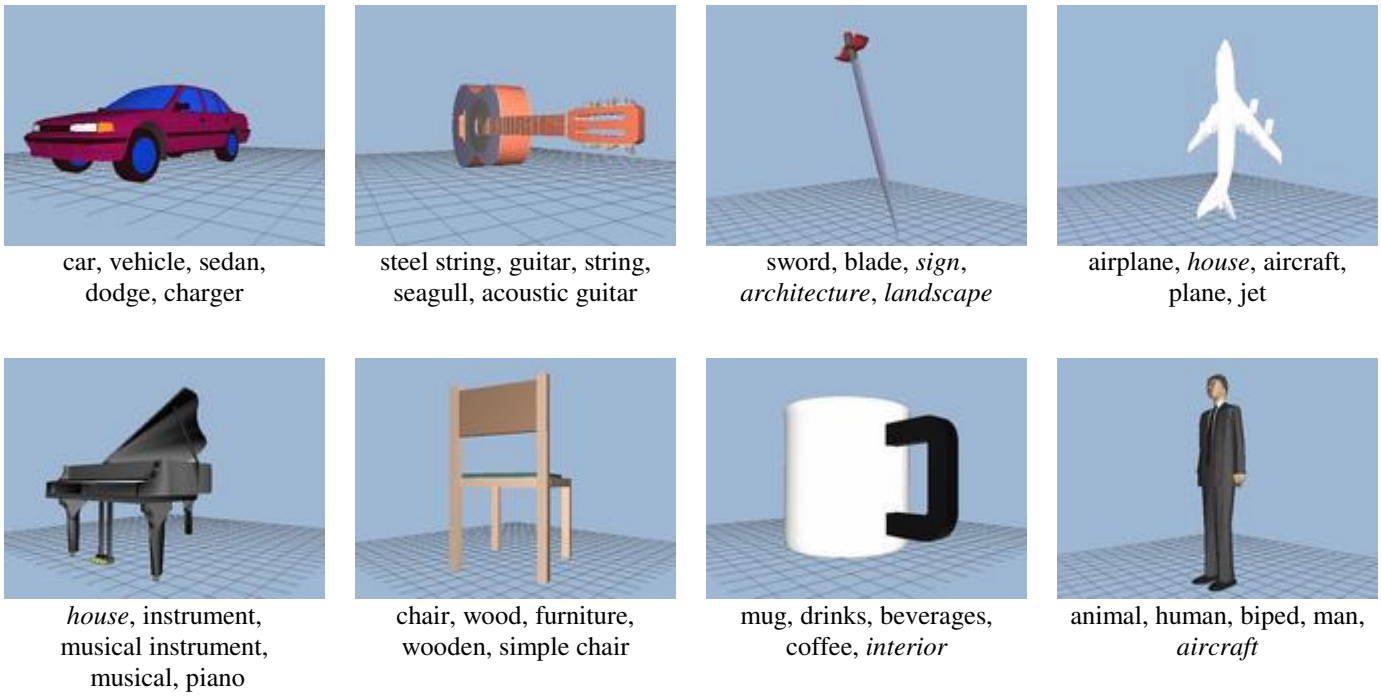


Fig. 7. Eight models from the PSB and their 5 best autotags. Tags we deemed to be irrelevant are shown in italics. ("Seagull" is considered relevant because it is a brand of guitar).

Fig. 7 shows some example models and their 5 best autotags. We examined the results and italicized tags that did not fit. Some of the autotagger's mistakes are due to geometry that is difficult to distinguish from other classes of models. For example, 3DWarehouse contains many long, thin street sign models that are geometrically rather close to the sword, which explains why the sword received the tag "sign." Other errors are more likely artifacts of our choice of corpus. In particular, 3DWarehouse contains a very large number of buildings, which tends to skew tagging towards words like "house" and "architecture."

VI. Conclusions and Future Work

We have demonstrated an automatic tagging system that learns new tags for an unknown 3D model by comparing it to a large set of tagged models and probabilistically propagating tags from neighbors. We have shown that the precision/recall of text search using our autotags approaches that of the underlying shape descriptor, and that searching for models based on our autotags can be much better than searching on the original tags. It is important to point out that the quality of our results is highly dependent on the quality of the corpus we use, in terms of both tag quality and coverage of the space of common 3D shapes. In choosing 3DWarehouse as our corpus, we have emphasized coverage over tag quality. The 192,343 models in our dataset are varied enough to have excellent (although not complete) coverage, but the quality of the annotations is very poor as compared to using a hand-classified database. We believe that our method would produce even better results with a more accurately annotated corpus, should one with comparably broad coverage become available.

Although we have focused in this paper on autotagging to improve shape retrieval, there are several other domains where automatically annotating 3D models can be helpful. For example, when users submit models to a collection such as 3DWarehouse, they often get to choose tags for the models. If we can autotag the model before this stage, we can suggest tags that already exist on other models, which could improve the consistency of annotations in the collection.

The system presented in this paper should be seen as a proof-of-concept for autotagging 3D models based on geometrically similarity. There are a number of ways the algorithm could be improved, such as using the original tags of a model to help find better autotags by boosting our confidence in neighbors with similar tags. In our future work we intend to examine this and other more sophisticated approaches to autotagging.

Acknowledgement

We wish to thank Google for providing us with a complete copy of 3DWarehouse converted to the .obj format, which made processing the corpus much easier.

References

[1] P. Min, M. Kazhdan, T. Funkhouser, "A comparison of text and shape matching for retrieval of online 3D models," *European Conference on Digital Libraries*, Bath, UK, 2004.

[2] D. Chen, X. Tian, Y. Shen, M. Ouhyoung, "On visual similarity based 3D model retrieval," *Eurographics*, Granada, Spain, 2003.

[3] T. Funkhouser, P. Min, M. Kazhdan, J. Chen, A. Halderman, D. Dobkin, D. Jacobs, "A search engine for 3D models," *Transactions on Graphics*, *22(1)*, pp. 83-105, 2003.

[4] D. DeCarlo, A. Finkelstein, S. Rusinkiewicz, A. Santella, "Suggestive contours for conveying shape," *SIGGRAPH*, San Diego, 2003.

[5] M. Novotni, R. Klein, "3D Zernike descriptors for content based shape retrieval," *Solid Modeling and Applications*, Seattle, 2003.

[6] M. Kazhdan, T. Funkhouser, S. Rusinkiewicz, "Rotation invariant spherical harmonic representation of 3D shape descriptors," *Symposium on Geometry Processing*, Aachen, Germany 2003.

[7] A. Tversky, "Features of similarity," *Psychological Review*, 1977.

[8] G. Leifman, R. Meir, A. Tal, "Semantic-oriented 3D shape retrieval using relevance feedback," *The Visual Computer, (Pacific Graphics)*, 2005.

[9] M. Novotni, G. Park, R. Wessel, R. Klein, "Evaluation of kernel based methods for relevance feedback in 3D shape retrieval, *International Workshop on Content-Based Multimedia Indexing*, Riga, Latvia, 2005.

[10] R. Datta, W. Ge, J. Li, J. Z. Wang, "Toward bridging the annotation-retrieval gap in image search," *MultiMedia 14(3)*, pp. 24-35, 2007.

[11] C. Fellbaum, *WordNet: An Electronic Lexical Database*, MIT Press, Cambridge, Ma, 1998.

[12] D. Saupe, D. V. Vranic, "3D model retrieval with spherical harmonics and moments," *Symposium on Pattern Recognition*, London, 2001.

[13] H. Laga, H. Takahashi, M. Nakajima, "Spherical wavelet descriptors for content-based 3D model retrieval," in *Shape Modeling and Applications*, Washington D.C., 2006.

[14] A. Mademlis, A. Axenopoulos, P. Daras, D. Tzovaras, M. G. Strintzis, "3D content-based search based on 3D Krawtchouk moments," in *International Symposium on 3D Data Processing, Visualization, and Transmission*, Washington D.C., 2006.

[15] N. Canterakis, "3D Zernike moments and Zernike affine invariants for 3D image analysis and recognition," *Scandinavian Conference on Image Analysis,* Kangerlussuaq, Greenland, 1999.

[16] P. Shilane, P. Min, M. Kahzdan, M. Funkhouser, , "The Princeton shape benchmark," *Shape Modeling International*, Genova, Italy, 2004.

[17] G. Salton, "Mathematics and information retrieval," *Journal of Documentation 25(1)*, pp. 1-29, 1979.

[18] G. Salton, "Term-weighting approaches in automatic text retrieval," *Information Processing and Management 24(5)*, pp. 513-523, 1988.

[19] S. A. Nene, S. K. Nayar, "A simple algorithm for nearest neighbour search in high dimensions," *Transactions on Pattern Analysis and Machine Intelligence, 19(9)*, 1997.