A Comparison of Text and Shape Matching for Retrieval of Online 3D Models

with statistical significance testing

Patrick Min

institute of information and computing sciences, utrecht university

technical report UU-CS-2004-26

www.cs.uu.nl

Abstract

Because of recent advances in graphics hard- and software, both the production and use of 3D models are increasing at a rapid pace. As a result, a large number of 3D models have become available on the web, and new research is being done on 3D model retrieval methods. Query and retrieval can be done solely based on associated text, as in image retrieval, for example (e.g. Google Image Search [15] and [40, 51]). Other research focuses on shape-based retrieval, using methods that measure shape similarity between 3D models (e.g., [14]).

The goal of our work is to take current text- and shape-based matching methods, see which ones perform best, and compare those. We compared four text matching methods and four shape matching methods, by running classification tests using a large database of 3D models downloaded from the web [50]. We also investigated several simple methods to combine the scoring results of text and shape matching. Additionally, this report uses statistical tests for significance testing of differences in precision/recall results.

We found that shape matching outperforms text matching in all our experiments. The main reason is that publishers of online 3D models simply do not provide enough descriptive text of sufficient quality: 3D models generally appear in lists on web pages, annotated only with cryptic filenames or thumbnail images. Of the available text sources of a 3D model, we found that the text *inside* the model file was the most useful for classification. Also, adding synonyms and hypernyms (category names) of model filenames using WordNet [31] improved classification results.

Combining the results of text and shape matching further improved performance. The results of this work provide added incentive to continue research in shape-based retrieval methods for 3D models, as well as retrieval based on other attributes.

Chapter 1

Introduction

There has been a recent surge of interest in methods for retrieval of 3D models from large databases. Several 3D model search engines have become available within the last few years (e.g., [5, 30, 32, 55]), and they cumulatively index tens of thousands of 3D polygonal surface models. Yet, still there have been few research studies investigating which types of query and matching methods are most effective for 3D data. Some 3D model search engines support only text queries [30], while others provide "content-based" queries based on shape [14]. But how do shape-based and text-based retrieval methods compare?

To investigate this question, we measured classification performance of the currently best-performing text-based and shape-based matching methods. We also evaluated several functions that combine text and shape matching scores. For the text matching, a 3D model is represented by a text document, created from several sources of text associated with the model, as well as synonyms and hypernyms (category descriptors) of the 3D model filename (added using WordNet, a lexical database [31]). For the shape matching, a 3D model is represented by a *shape descriptor*, computed from the polygons describing the model's surface.

All classification tests were done using the *Princeton Shape Benchmark* [50] (PSB) 3D model test database. It contains 1814 3D models downloaded from the web, subdivided into a training set and a test set, containing 907 models each, manually classified into 90 and 92 comparable classes respectively. It is a subset of a larger database of about 33000 models downloaded from the web using an automatic crawler [33]. Retrieval results were evaluated using precision/recall curves [58].

We found that shape-based matching outperforms text-based matching in all our experiments. The main reason is that 3D models found on the Web are insufficiently annotated. They usually are presented in lists, annotated with at most a single name, which is often misspelled or a repeat of the filename. Of the available text sources, we found that the text inside the model file itself and the synonyms and hypernyms of the filename were the most discriminating. Additionally, we found that when combining the results of the shape and the text matching method, several combination functions produced a significant improvement over shape alone.

Overall, our results show that the relatively simple solution of using only associated text for retrieval of 3D models is not as effective as using their shape.

The rest of this report is organized as follows. The next chapter discusses all issues relevant to evaluating the matching methods, namely the test database used, performance metrics, and statistical significance testing. Text matching and our approach for maximizing text retrieval performance is described in Chapter 3. Chapter 4 discusses shape matching and shows the performance of several recent shape matching methods. Text and shape matching are compared in Chapter 5 and combined in Chapter 6. Conclusions and suggestions for future work are in Chapter 7.

Chapter 2

Evaluation Method

2.1 Introduction

In this chapter, we describe the test database and the performance metrics that were used for evaluating the various matching methods. We also discuss some options for statistical tests to evaluate the significance of differences in performance.

2.2 Test Database

Retrieval performance was measured with respect to a "ground truth" test database: a database of 3D models classified into categories. The goal in a retrieval experiment then becomes: given a query model from a certain category, retrieve all models from that same category.

The test database we used is the *Princeton Shape Benchmark* (PSB) 3D model test database [50]. It contains 1814 3D models downloaded from the web, subdivided into a training set and a test set, containing 907 models each, manually classified into 90 and 92 comparable classes respectively.

Next, we briefly describe how the PSB was constructed. Its initial source was a collection of about 44000 models acquired in three separate web crawls [33, 50]. After removal of duplicates, model files with errors, etc., about 33000 models remained, which had to be manually classified. To avoid having to manually identify models with a (nearly) identical shape, we clustered the models using a 3D shape similary metric (by comparing *Spherical Harmonics Descriptors*, see Section 4.2) as a distance measure [22]. For example, we found multiple copies of the same model at different URLs, multiple levels of detail for the same object, and different colours/textures for models with the same geometry (e.g., 483 spheres, 261 cubes, 33 cylinders, etc.). This resulted in 15990 clusters. The model at the centroid of each cluster was chosen as its representative. These representative models formed the database of models which had to be manually classified.

An initial classification of this 15990 model database was done by an undergraduate student (David Bengali), resulting in 384 classes. Classes such as abstract geometric shapes, data visualizations, and molecule models were left out, because they contained either many unspecific and/or abstract models, or were difficult to classify further without expert knowledge.

We then further refined the classification, resulting in a set of approximately 5000 models. From this set we selected a subset of 1000 models, subdivided into 81 classes [32]. The classes were chosen such that (1) they represented a wide variety of models, (2) no single class would become too large (e.g. larger than 10% of the database size), and (3) there was a wide range of class sizes. This dataset was later expanded into the 1814 model Princeton Shape Benchmark.

2.3 Performance Metrics

2.3.1 Precision/Recall

Retrieval results are evaluated using precision/recall curves [58]. Here we briefly review how these are computed.

Given a classified test database of 3D models, each model is submitted as a query and matched to all models (including itself) using a matching method. The models are then ranked according to their matching score.

The query model is a member of a certain class of size c. For each number of returned results k, and a number of relevant results rel (i.e. models that are members of the same class) within these returned results,

$$recall = rel/c$$

$$precision = rel/k$$

A perfect classification method would always return objects from the same class as the query object in the top c results. In this case the precision would be 1.0 for each recall value, and the precision/recall plot a horizontal line at $y = precision = 1.0$. In practice the goal is to achieve as high as possible precision values. For a certain number of recall intervals (20 in our implementation) in the range $[0, 1]$, precision values are averaged over all models. The average over all models is called the *micro*-average. A *macro*-average is computed by first computing the average of the models in each class, and then averaging these class averages.

To gain some insight into what it means if average precision differs by, for example, 0.1, we examine the following retrieval results. Consider a query from a class of size $c = 10$, and the first twenty retrieved results. In the table below three different ranked result lists are given, where an "R" denotes a relevant result (i.e. one from the same class as the query). The top row shows the result rank, the next three rows

rank → / test nr. ↓	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
I		R		R	R				R		R			R	R		R	R		R
II	R		R	R				R		R			R	R		R		R		R
III		R		R	R	R		R			R	R		R	R		R			

show the positions of the 10 class members from three different retrieval tests. Row II is the same as row I, but with the *top eight* results shifted up by *one* position. Note that changes at the top contribute relatively a lot to the average precision: for example, the number two result (precision = $1/2 = 0.5$) moves to the top spot (precision = $1/1 = 1.0$), resulting in an increase of $(1.0 - 0.5)/10 = 0.05$ in average precision. Row III is the same as row I, but with the *bottom seven* results shifted up by *three* positions. Both these changes result in an increase in average precision of about 0.1 when compared to the first row (the actual average precision values are 0.486, 0.588, and 0.578, so the percentage improvements are 21% and 19%).

2.3.2 Discounted Cumulative Gain

An alternative metric is the *Discounted Cumulative Gain* (DCG) [21]. The DCG weighs correct results near the top of the ranked list of results more than results lower in the list. The motivation behind this metric is that the user is most likely to examine the top of the results list (as opposed to examining all results which could be spread across 50 pages of 16 results each, for example).

A formal description (copied from [50]) is: the ranked results list R is converted to a list G, where element G_i has value 1 if element R_i is in the correct class and value 0 otherwise. Discounted cumulative gain DCG_k (with k the number of results) is then defined as [21]:

$$DCG_i = \left\{ \begin{array}{ll} G_1, & i = 1 \\ DCG_{i-1} + \frac{G_i}{\log_2(i)}, & otherwise \end{array} \right\}$$

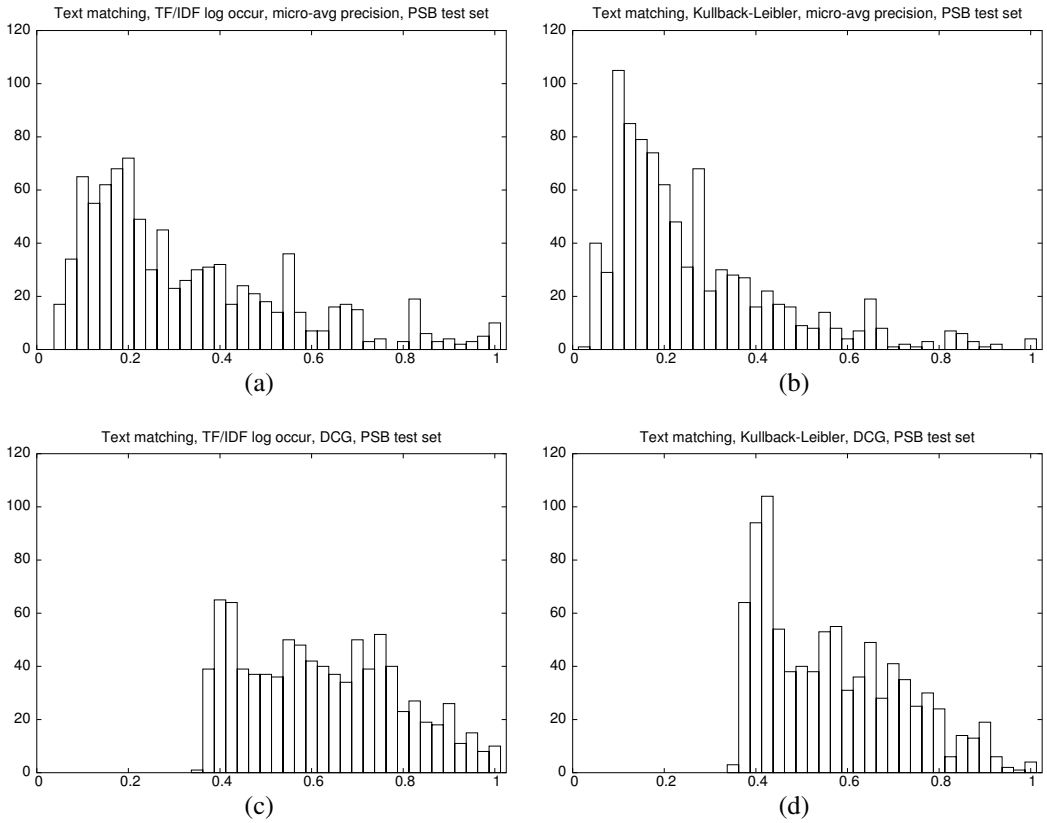


Figure 2.1: Example score distributions for two different metrics (micro average precision in (a) and (b), and DCG in (c) and (d)) of two different text matching methods (TF/IDF in (a) and (c), Kullback-Leibler in (b) and (d))

This result is then divided by the maximum possible DCG (i.e., that would be achieved if the first c results were in the correct class, where c is the size of the class) to give the final score:

$$DCG = \frac{DCG_k}{1 + \sum_{j=2}^{c} \frac{1}{\log_2(j)}}$$

DCG seems a more attractive measure, especially in cases when a lot of relevant results from a large class appear both near the top and far down the results list. Even though the user is happy (because many good results show up on the first page), the average precision is pulled down a lot by the lower-ranked results.

2.4 Statistical Significance of Differences in Precision/Recall Graphs

The distribution of performance scores (e.g., average precision, DCG, etc.) depends on which 3D models are in our test database, how they are classified, and (obviously) the used matching method. For example, Figure 2.1 shows sample distributions of two different metrics (micro average precision and DCG) of two different text matching methods ("TF/IDF log occur" and "Kullback-Leibler", see Chapter 3 for details), tested using the PSB test set.

If the distribution of the differences is approximately normal, then we can use the well-known Student's paired t-test for significance testing. To establish this, we first have to run a normality test on the distribution of differences. Figure 2.2 shows the distribution of differences of the score samples in Figure 2.1.

If the normality test says the distribution of differences is *not* "sufficiently normal", then we have to use a non-parametric significance test, for example Randomization (on means of matched pairs) or Wilcoxon's

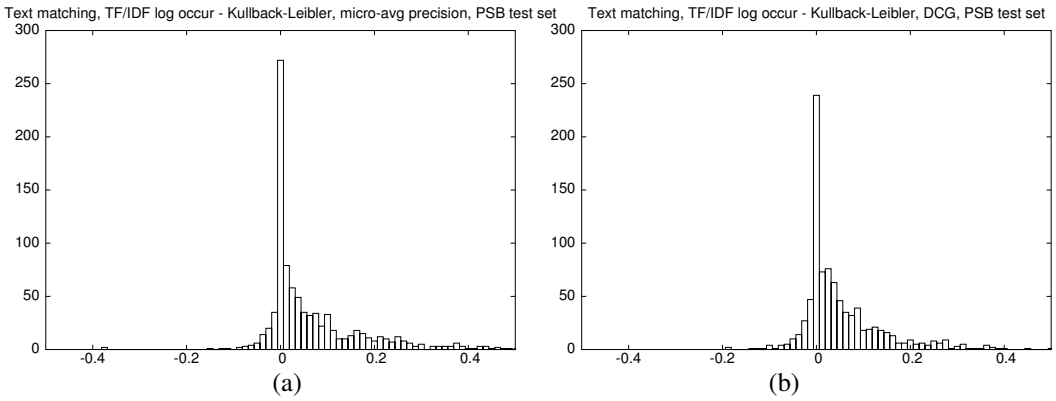


Figure 2.2: The distribution of differences of the micro average precision scores of (a) TF/IDF log occur minus Kullback-Leibler, and (b) the same distribution for the DCG scores

Sum of Signed Ranks test (more details about these tests appear below). This procedure was also suggested by Hull [19] (he uses Wilcoxon's test and a sign test as the non-parametric alternatives). Strictly speaking this whole procedure (i.e. testing for normality and then applying another test depending on the outcome) becomes a new statistical test. Therefore we can no longer claim that a result is significant at the 5% level, for example, even if in the last step the test we picked said so. Also, in practice the Student's t-test can be used even if the data being tested is not normally distributed (except when distributions are heavily skewed or contain many outliers). As will be seen below, this is supported by our results.

First, we will describe each step of the suggested procedure in more detail. Next, the three significance tests are compared in Section 2.4.5. Note that all tests were 2-tailed, in other words, no assumptions were made beforehand about which set of scores was higher than the other.

2.4.1 Test for Normality

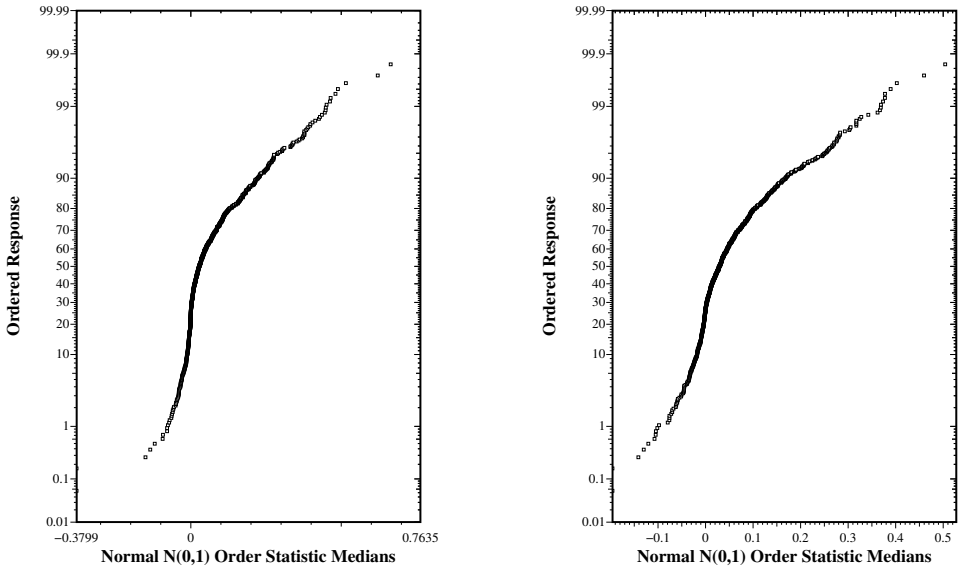
The normality test we use was first described in [11]. This test computes the correlation coefficient of the *normal probability plot* of our data [4]. In a normal probability plot, the data is plotted against a theoretical normal distribution such that if it is approximately normal, then the plotted points should form an approximate straight line. The "straightness" of this line can be quantified by its correlation coefficient (ranging between 0.0 and 1.0, 1.0 corresponding to a perfectly straight line). If the correlation coefficient is less than a certain cutoff value (determined by the chosen significance limit (e.g. 5%) and the number of degrees of freedom ($= N - 1$, with N the number of data points) [11, 25]), then we reject the null hypothesis that the data is normally distributed.

To compute the normal probability plot, we use a program called `plotmtv` [39]. This program was slightly modified to store the (x, y) values of the plot in a file, from which the correlation coefficient could be computed. Figure 2.3 shows the normal probability plots produced by `plotmtv` for the difference distributions shown in Figure 2.2.

The correlation coefficients of these two plots are 0.9013 and 0.9278 respectively. From the cutoff values in [25] we find that both are significant at the 0.01 level, so both distributions are not normally distributed.

2.4.2 Student's paired t-test

If the result of the normality test is that the data is "sufficiently" normally distributed, then we can use the well-known Student's paired t-test to test the significance of the differences. From MathWorld [60]: "Given two paired sets of n measured values, the paired t-test determines whether they differ from each other in a significant way under the assumptions that the paired differences are independent and identically normally distributed". We refer the interested reader to [60] and the references cited there.



(a)

(b)

Figure 2.3: The normal probability plots of the distributions of differences shown in Figure 2.2, produced using the `plotmtv` program [39]

2.4.3 Randomization Tests

A non-parametric test (i.e. a test with no assumptions about the underlying distribution of the data) we can use is the Randomization test, first introduced by Fisher [13]. In our case we use the Randomization test on means of matched pairs, which works as follows: first we compute a statistic $t = t_{obj}$ of the differences of all paired scores. Then, using the null hypothesis that the differences are there by accident, implying that for each paired result the difference could equally likely have been positive as negative, we flip the sign of a random subset of all differences and recompute the statistic $t = t_{new}$. We then check if $t_{new} > t_{obj}$. This process is repeated a large number of times (say 10^5 times) and we tally the number of times that $t_{new} > t_{obj}$. If the null hypothesis were true, this should happen a significant number of times (in more than 5% of all cases), implying no significant difference between the input distribution pair.

In our implementation, t is the normalized mean difference (the mean difference divided by the standard error, where the standard error equals the standard deviation divided by the square root of the number of scores). Because we are doing a two-tailed test, we take the absolute value of the normalized mean difference.

For the Randomization test, N times a random number of differences is flipped, and the t statistic is recomputed. Some C code for this step is given below.

```
float compute_mean_difference()
{
  float sum_diff = 0;
  for(int i=0; i < nr_scores; i++) {
    float diff = data[0][i] - data[1][i];
    float abs_diff = fabs(diff);

    float random = (float) rand() / RAND_MAX;
    if (random < 0.5) diff = -abs_diff;
```

```
    else diff = abs_diff;
    sum_diff += diff;
  }
  float mean_diff = sum_diff / nr_participating;
  return mean_diff;
}
```

The main loop then is (with `NR_COMBINATIONS` equal to, say, 10^5, and `std_err` (the standard error) and `t_obj` already computed):

```
  int nr_larger = 0;
  for(int i=0; i < NR_COMBINATIONS; i++) {
    float mean_diff = compute_mean_difference();
    mean_diff /= std_err;
    mean_diff = fabs(mean_diff);
    if (mean_diff >= t_obj) nr_larger++;
  }
  float P = (float) nr_larger / NR_COMBINATIONS;
  if (P > 0.05) printf("No significant difference (at 0.05)\n");
```

2.4.4 Wilcoxon's Sum of Signed Ranks Test

Another non-parametric significance test is the Wilcoxon Sum of Signed Ranks Test [28, 61]. Its significance value is computed as follows. Given two sets of scores for N 3D models, then:

1. for each, compute the absolute difference in score, and remember the sign of the difference
2. discard differences that are less than some ϵ value
3. sort the resulting differences
4. assign an average rank value to models with an equal difference (e.g., 2 models both have an absolute difference of 0.31 at rank 4 and 5: their new rank will be $(4 + 5)/2 = 4.5$
5. change the sign of each rank according to the original sign of the difference
6. sum the resulting ranks $= W$

For high N ($>= 10$) we can approximate the distribution of the resulting sum W with a normal distribution. It can be shown that the standard deviation is

$$\sigma = \sqrt{\frac{N(N+1)(2N+1)}{6}}$$

From σ and W we can compute the probability that the difference in ranks was caused by chance, assuming a null hypothesis of equal underlying processes.

Step 2 in the above algorithm is important: it determines how many of our sample values are discarded. If, for example, we consider precision values that are $\epsilon = 0.1$ apart to be essentially equal, it could be the case that 40% of all score pairs are discarded. Even if then the statistical significance test says the overall difference is significant, we can only conclude that this is the case for the 60% that participated in the test.

Clearly the probability value (P value) depends a lot on the chosen value of ϵ. To examine this dependency more closely, we can plot the value of P depending on ϵ. Figure 2.4 shows an example of such a plot. The solid line shows P, the histogram shows the number of participating scores depending on ϵ (from this we can see that the P values for larger ϵ (e.g. > 0.2) do not carry much information about the whole population anymore). This plot shows that for all subsets of scores (ranging from the pairs with small differences to those with large differences) their difference is insignificant (because the P graph is above 0.05 everywhere).

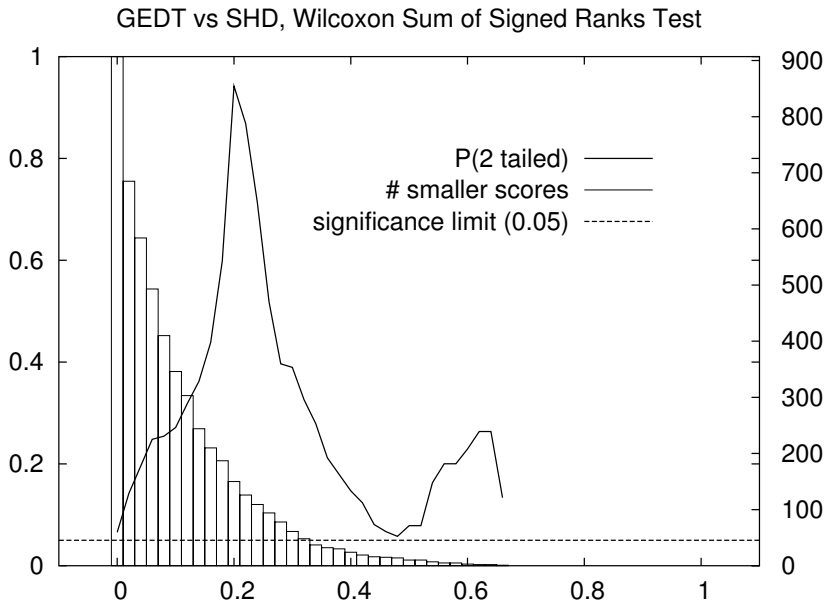


Figure 2.4: This graph shows both the P value and the number of scores participating in Wilcoxon's test depending on ϵ. The horizontal line marks the significance limit of $P = 0.05$ (so in this graph all comparisons show an insignificant difference)

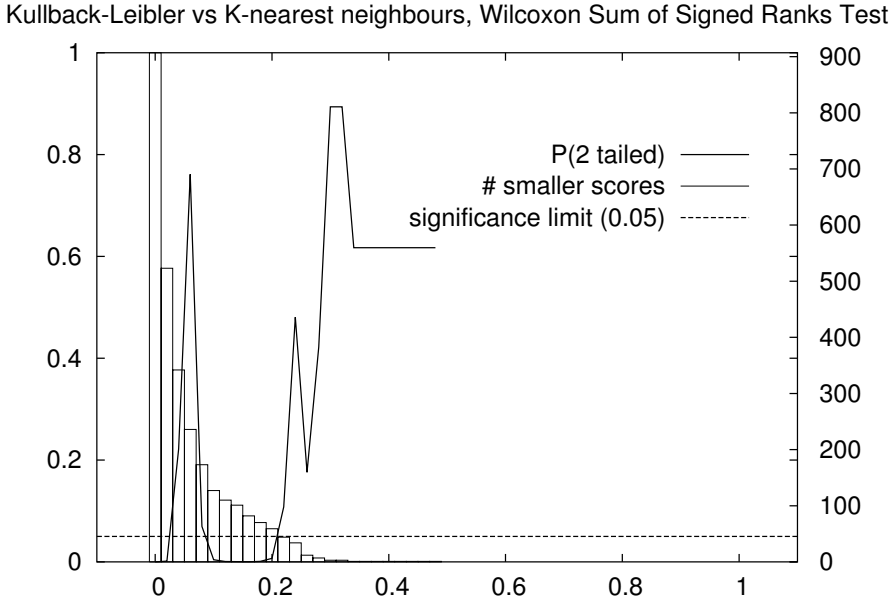


Figure 2.5: The same graphs as in Figure 2.4, comparing two text matching methods. Note that the choice of ϵ greatly influences the significance result: e.g. 0.05 will show an insignificant difference, 0.1 will show a significant one

Figure 2.5 shows the same plot, comparing the scores of two text matching methods. Note that the choice of ϵ greatly influences the significance result: e.g. 0.05 will show an insignificant difference, 0.1 will show a significant one. This means that we cannot just pick an ϵ value and trust the result of Wilcoxon's test.

For these reasons (the inability to compare significance tests when discarding scores, and the arbitrariness of picking an ϵ value) we recommend to leave out step 2 altogether (in other words, set ϵ to 0).

Weighted Cumulative Probability P_{wc}

To avoid having to look at such a graph for every significance test, we propose to first compute the following metric: the *weighted cumulative probability* P_{wc}, computed as follows (with k the number of ϵ samples we are taking, n_ϵ the number of scores participating given ϵ, $P(\epsilon)$ the Wilcoxon P value for the subset of scores determined by ϵ, and N the total number of scores):

$$P_{wc} = \sum_{i=1}^{k} \frac{n_\epsilon}{N} \cdot P(\epsilon) \ \wedge \ \epsilon = i/k$$

In words, this is an approximation of the area under a graph defined as the P graph weighted by the percentage of participating scores at each ϵ. It is an indicator of the "strength" of the significance: a low P_{wc} probably indicates a signifcant difference (i.e. $P < 0.05$) across the board. If P_{wc} is high (e.g. > 0.2) then we may decide to look at the actual graph.

For example, the graph in Figure 2.6 has a P_{wc} value of 0.23. From an ϵ value of 0.1 we would have found the scores to be significantly different (the Randomization and Student's t-test both say they are significantly different). The graph shows that this is indeed the case, *except* for the about 15% highest scores.

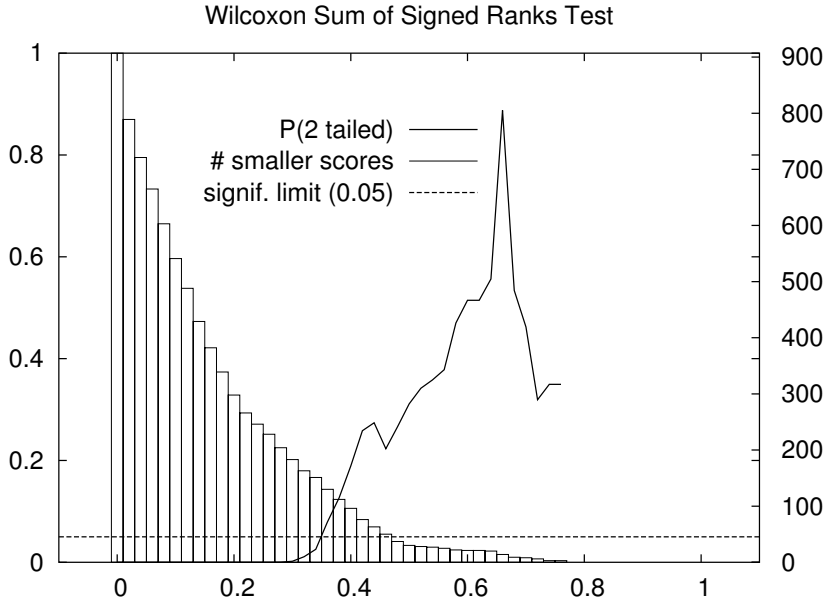


Figure 2.6: The P_{wc} value for this graph is 0.23 leading us to decide to examine the actual graph, which shows that the scores are significantly different, except for the about 15% best scores

To investigate the correlation between the P_{wc} value and the outcome of the Randomization test, we produced a histogram of the P_{wc} values of pairs of score sets for which the Randomization test said their difference was significant, and compared it to a histogram of P_{wc} values of insignificant difference: see Figure 2.7. The histogram suggests that P_{wc} values below about 0.3 imply a significant difference, and

values above 2.0 an insignificant difference. For P_{wc} values inside this interval the actual graph may need to be examined.

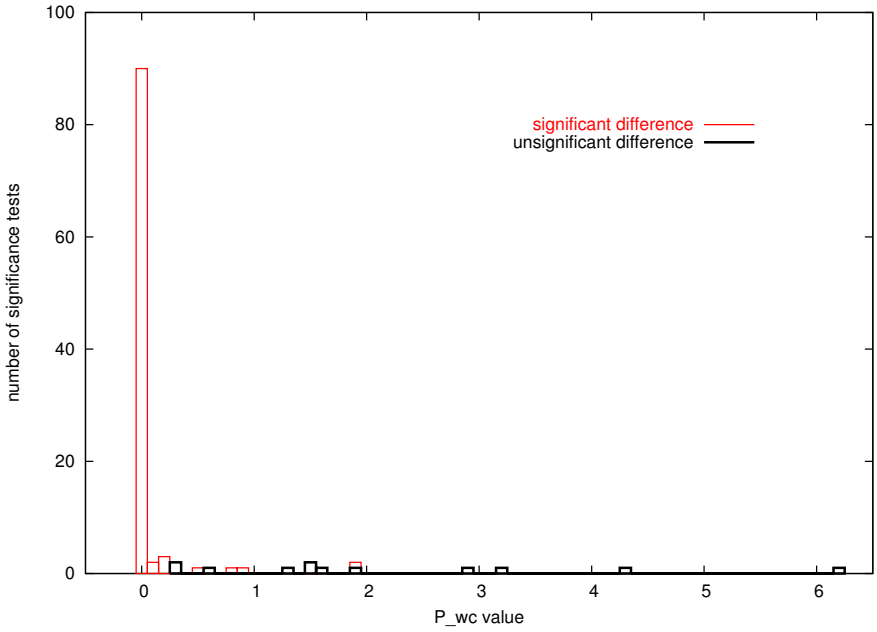


Figure 2.7: P_{wc} values corresponding to significant and insignificant differences (using the Randomization test), for a total of 112 tests (100 significant, 12 insignificant differences)

QQ plot

Of course the graphs as produced in Figure 2.6 are of our own design, and not used in the statistics literature. For completeness, we briefly decsribe a more common way to compare two distributions: the *quantile-quantile plot*, or *QQ plot*. In our case, comparing two equally sized distributions, the QQ plot compares two ordered sets of values. From the way the plot deviates from the line $x = y$ one can make some qualitative statements about the difference between the distributions. For example, Figure 2.8 shows the QQ plot comparing the same two distributions as in Figure 2.6. This shows that the x values (category name queries) are smaller then the y values (representative document queries) until about 0.12, after which the opposite is true. Note that here actual average precision values per query are being plotted, and that it is not possible to derive the distribution of differences from this plot, because the query to query correspondence is lost.

2.4.5 Comparing the Wilcoxon Signed-Rank Sum test, the Randomization test, and the Paired t-test

Each time we had to apply a statistical test, we used all three tests, to see if they produce consistent (qualitative) results. Out of the 112 pairs of sets of scores tested for this report, each time all three tests agreed, even though normality tests showed we could not use the Student's t-test in almost all cases. This suggests that for our type of data the Student's t-test works just as well as the non-parametric tests.

Also note that with any significance testing method a distribution which is consistently higher than another by a small margin will result in a very small P value, showing it is significantly higher. So the significance does not lie in the *size* of the margin, but in its consistence. Attaching an importance to a difference of a certain size is another matter (e.g. see Section 2.3.1).

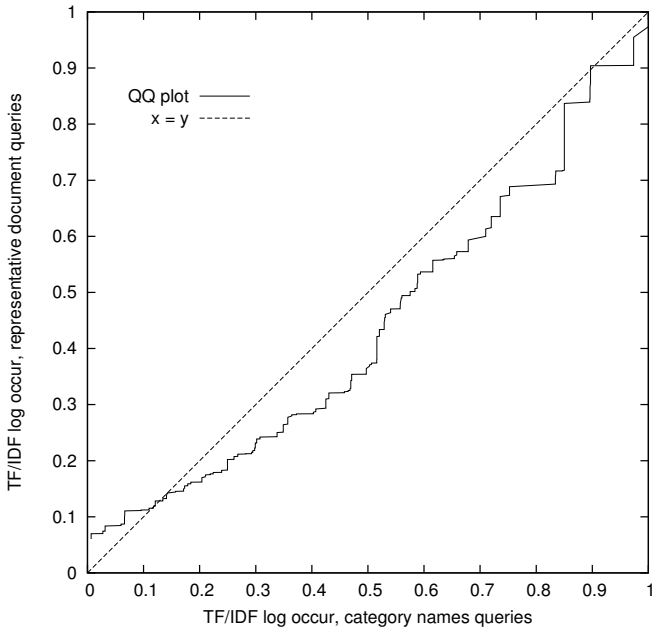


Figure 2.8: A QQ plot comparing the same two distributions as in Figure 2.6

2.4.6 Further Reading

For an extensive discussion of potential problems with precision/recall graps and some alternatives, see Huijsmans and Sebe [18]. The papers by Knapp [23] and Salzberg [45] contain examples and discussion of things to watch out for when selecting and using statistical tests. Statistical tests with cross-validation are evaluated in a paper by Dietterich [10]. The procedure we follow is equivalent to the one suggested by Hull [19]. For a gentle introduction to the field of inferential statistics, see the online book by Lowry [28].

Chapter 3

Text Matching

In this chapter, we first review related work on retrieval of non-textual data using associated text. Note that we do not discuss text retrieval itself. We refer the interested reader to [2, 44, 49]. We then describe the sources of text found with 3D models crawled from the web and investigate how this text can be used to compute a similarity measure for the associated 3D models.

3.1 Related Work

There has been relatively little previous research on the problem of retrieving non-textual data using associated text. The web is an example of a large database for which such methods can be useful: (1) it contains many non-textual objects (e.g., images, sound files, applets) and (2) these objects are likely to be described on web pages using text. Examples of web search engines that take advantage of associated text are Google Image Search [15] (for images), FindSounds [12] (for sound files), and MeshNose [30] (for 3D models).

Probably the largest site for searching images using text keywords is Google's image search. Unfortunately there are no publications available about the method they use. A related FAQ page suggests that heuristics are used to determine potentially relevant text related to an image, for example, the image filename, link text, and web page title. Each source is probably assigned a different weight, depending on its importance, similar to how the main Google search site assigns weights to text terms depending on whether they are in the title, headers, and so on.

Sable and Hatzivassiloglou investigated the effectiveness of using associated text for classifying images from online news articles as indoor or outdoor [42]. They found that limiting the associated text to just the first sentence of the image caption produced the best results. In other work, Sable *et al.* use Natural Language Processing (e.g., identifying subjects and verbs) to improve classification performance of captioned images into four classes [41]. Our problem is slightly harder since our source text is less well-defined (i.e., there is not an obvious "caption"), and the number of classes is much higher.

Even though promising results have been achieved in text-based image retrieval, it is not obvious if text matching will outperform shape matching for 3D model retrieval: 3D shape matching is in some respects easier than image matching: 3D models are often already segmented from their background and do not contain shadows, occlusions, projections, etc. To our knowledge, there has never been a study investigating the effectiveness of text indexing for 3D models.

3.2 Text Sources

In our study, we focus on the common "bag of words" approach for text matching: all text that is deemed relevant to a particular 3D model is collected in a "representative document," which is then processed and indexed for later matching. This document is created using several potentially relevant text sources. Because we are indexing 3D model files linked from a web page, we are able to extract text from both the model file itself as well as the web page (note that because we convert all models to the VRML 2.0 format,

we only refer to text sources of this format). The following list describes the text sources we can use:

From the model file:

1. **model filename:** The filename usually is the name of the object type. The extension determines the filetype. For example, `alsation.wrl` could be the filename of a VRML file of an Alsation dog

2. **model filename without digits:** From the filename we create a second text source by replacing all digits with spaces. Very often filenames contain sequence numbers (for example, `chair2.wrl`) that are useless for text keyword matching (though not always, as in, for example, `f16.wrl`)

3. **model file contents:** A 3D model file often contains labels, metadata, filenames of included files, and comments. In VRML, it is possible to assign a label to a scenegraph node (a part of the model) and then re-use that node elsewhere in the file. For example, in a model of a chair, a leg can be defined once, assigned the identifier `LEG`, and then re-used three times to create the remaining legs. As such, these identifiers typically describe names of parts of the model. To describe metadata, a VRML 2.0 file may contain a `WorldInfo` node, which is used to store additional information about the model, such as a detailed description, the author name, etc. Filenames of included files can be names of other model files, textures, or user-defined nodes. Included model files are also parsed for relevant text. Finally, a model file may contain descriptive comments. The model file comments were left out from our experiments because we found that many files contain commented-out geometry, which, when included, would add many irrelevant keywords

From the web page:

4. **link text:** This is the descriptive text of the hyperlink to the model file, i.e., the text between the `<a>` and `</a>` HTML tags. For example: `<a href="b747.wrl">a VRML model of a Boeing 747</a>`

5. **URL path:** These are the directory names of the full URL to the model file. If multiple models are organized in a directory structure, the directory names could be category names helpful for classification. For example, as in the URL `http://3d.com/objects/chairs/chair4.wrl` (which would yield the words `objects` and `chairs`)

6. **web page context (text near the link):** We define the context to be all plain text after the `</a>` tag until the next `<a href>` tag (or until the next HTML tag if there is none). This text could for example read "1992 Boeing 747-400 passenger plane, 210K, created by John Doe". Context found *before* the link text was found to be mostly irrelevant

7. **web page title:** The title of the web page containing the link to the 3D model. It often describes the category of models found on the page, for example, "Models of Airplanes"

Additional text source:

8. **Wordnet synonyms and hypernyms:** We create an additional eighth text source by adding synonyms and hypernyms (category descriptors) of the filename using WordNet, a lexical database [31] (if no synonyms or hypernyms can be found for the filename, the link text is tried instead). In related work, Rodriguez *et al.* use WordNet synonyms [9], and Scott and Matwin use synonyms and hypernyms [48] to improve classification performance. Recently, Benitez and Chang showed how WordNet can be used to disambiguate text in captions for content-based image retrieval [3]. Adding synonyms and hypernyms enables queries like "vehicle" to return objects like trucks and cars, or "television" to return a TV. WordNet returns synonyms and hypernyms in usage frequency order, so we can limit the synonyms and hypernyms used to only the most common ones. Currently we use the first three synonyms and the first hypernym (all combinations of 0-5 synonyms and 0-5 hypernyms were tested)

3.2.1 Text Processing

Following common practices from text retrieval, all collected text goes through a few processing steps. Here we illustrate these steps in detail by describing for an example model what text is used and how this text is processed. The model is of the Parthenon, and was found on a website called "The VRML Shrine" (`http://www.geocities.com/BourbonStreet/1855/shrine.html`): see Figure 3.1 for a screenshot of this website.

For some of the processing steps (stop word removal and stemming) we use a tool called `rainbow`, a program of the Bow toolkit, a freely available C library for statistical text analysis [29]. Before passing text to `rainbow`, each word is converted to lowercase, and words of length 1 are removed.

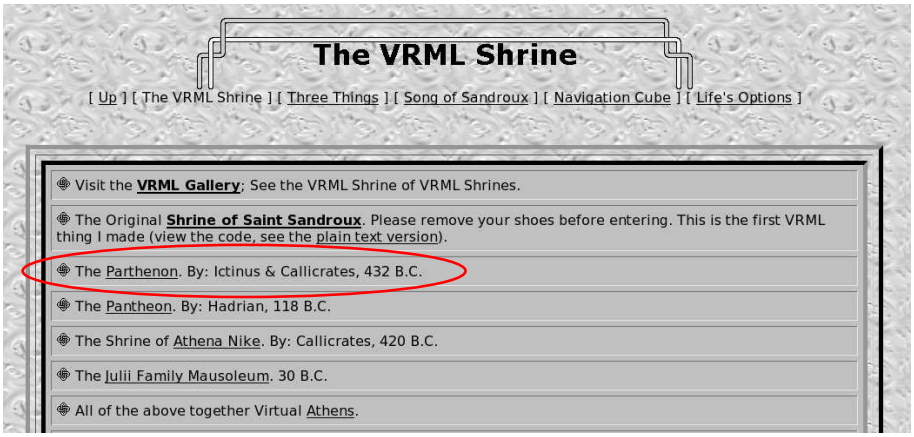


Figure 3.1: An example website containing 3D models. The text processing steps are illustrated using the Parthenon model on this site

Retrieved Text

From the model file:

1. **model filename:** `parthenon.wrl`
2. **model filename without digits:** `parthenon.wrl`
3. **model file contents:** `WorldInfo` node, with the following text in the `info` field: `Author: Mr. Phillip, Company: phi, LastSaveBy: Mr. Phillip, EditingTime: 28837.` Additionally, many nodes have an identifier assigned to them (by using `DEF`): `angel band bint (23 times, with sequence numbers) bmid (6) ceiling_wall chsl (7) cube (2) doric (46) dress floor (5) hair halo int (4) label (28) mid (6) _object0 roofa roofa2 roofb root skin tie torch wings`

From the web page:

The part of the web page source relevant to our example model is:

```
<td VALIGN="MIDDLE"><!--mstheme--><font face="verdana, arial, helvetica">
<font face="Verdana"><img SRC="knot.gif" WIDTH="16" HEIGHT="16">
The <a href="Parthenon.wrl">Parthenon</a>.
By: Ictinus &amp; Callicrates, 432 B.C.
</font><!--mstheme--></font></td>
```

4. **link text:** `parthenon`
5. **URL path:** `bourbonstreet 1855`

6. **web page context (text near the link):** `by ictinus callicrates 432 b c`
7. **web page title:** `the vrml shrine`

Additional text source:

8. **Wordnet synonyms and hypernyms:** `parthenon temple`
   These words are derived from the output of `wn parthenon -synsn`:

```
 Synonyms/Hypernyms (Ordered by Estimated Frequency) of noun parthenon

1 sense of parthenon

Sense 1
Parthenon
       => temple
```

The representative document now contains the following text:
filename: `parthenon wrl`
filename without digits: `parthenon wrl`
extension: `wrl`
path: `bourbonstreet 1855`
title: `the vrml shrine`
link text: `parthenon`
context: `by ictinus callicrates xfourxxthreexxtwox the`
modelfile: `author mr phillip company phi lastsaveby editingtime 28837 doric object chsl label bint int bmid roofb roofa ceiling wall cube mid floor angel root band halo torch tie wings hair skin dress`
synonyms: `parthenon temple`

Stop Word Removal

Next, *stop words* are removed. These are common words that do not carry much discriminating information, such as "and," "which," and "my". We use the SMART system's stop list of 524 stop words [43], as well as stop words specific to our domain. The domain-specific stopwords we use are:
`img src border jpg gif www com http edu png rgb rgba bw tga kb mb gb file html copyright shape gz gzip gzipped texcoord coord faces entity noname bmp object continue download example static surface spline faceset splin axis geometry defaultcamera gravity vp netscape java browser mat byte bytes url defmat ctrl alt`

Additionally, all VRML node and field names are removed: they frequently occur in user-defined node names, as in `DEF Transform12 Transform { ...`, for example. This includes node and field names from VRML 1.0, because user-defined identifiers are preserved by the VRML 1.0 to 2.0 converter (we used `vrml1ToVrml2`, running under Irix). A few node names are *not* removed, for example `Sphere, Camera, Switch, Spotlight`, etc., because these could also be object identifiers.

As the list of common stop words we initially used the "DVL/Verity stop word list" [26] (containing 457 stop words). The `rainbow` program by default removes the 524 stop words of the SMART retrieval system [43]. Currently both lists are applied, because the DVL/Verity list contains a few words that are not in the SMART list. For our example, this results in the removal of the words `the by the author mr` (the last two are present in the DVL/Verity list, but not in the SMART list).

Stemming

Finally, the resulting text is *stemmed* (normalized by removing inflectional changes, for example "wheels" is changed to "wheel"), using the Porter stemming algorithm [38]. This step is also done automatically by

the `rainbow` program.

For our example, the following words are changed: `ictinus` → `ictinu`, `callicrates` → `callicr`, `company` → `compani`, `lastsaveby` → `lastsavebi`, `editingtime` → `editingtim`, `ceiling` → `ceil`, `wings` → `wing`, `temple` → `templ`.

Erratum: from a recently (June 30, 2004) discovered bug in the `rainbow` source we found that stemming never takes place, even when the commandline parameter `--use-stemming` is specified. After fixing this bug we found that *with* stemming average precision (of the best performing matching method, see Section 3.3) is lower by about 6%. As a consequence, all results in this technical report, and in [32] and [34], are for *unstemmed* text (even though in these references it says the text is stemmed). Note that the conclusions from the results in [32, 34] remain unchanged because the best performing text matching method did not change.

3.3 Text Matching Methods Performance Evaluation

Given a representative text document for each 3D model, we can define a textual similarity score for every pair of 3D models as the similarity of their representative text documents. To compute this score, we use a variety of text matching methods provided by `rainbow`, a program of the Bow toolkit, a freely available C library for statistical text analysis [29]. The tested methods were: three variations of TF/IDF [44], Kullback Leibler [24], K-nearest neighbours, and Naive Bayes [35].

In our tests, each 3D model's text document is assigned its own class (as opposed to putting, for example, all text documents of the "helicopter" class in a single class) because our target application is retrieval, not classification. In other words, the results returned by `rainbow` should be text documents of individual models, not class names.

Figure 3.2 shows the precision/recall results obtained when the representative text document for each 3D model in the test set of the Princeton Shape Benchmark was matched against the representative text documents of all the other 3D models. The matches were ranked according to their text similarity scores, and precision-recall values were computed with respect to the base classification provided with the benchmark. From this graph, we see that the "TF/IDF log occur" method produces the highest scores for our dataset. Before we select this method as our "reference" text matching method, we have to test the significance of the score differences.

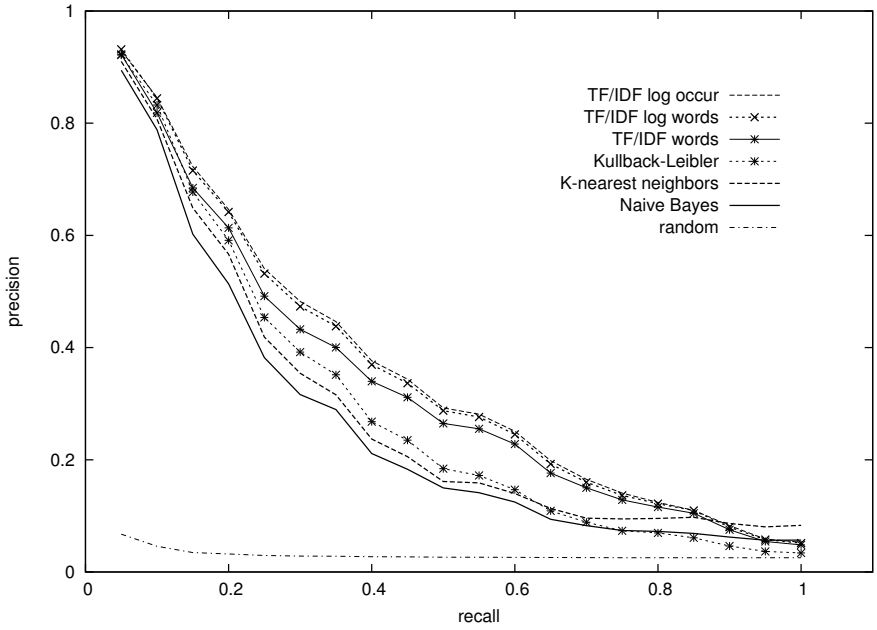


Figure 3.2: Average precision/recall for seven different text matching methods, and random retrieval

3.3.1 Significance Testing of Micro Average Precision

Following the procedure described in Section 2.4, we first test the distribution of all sets of differences of all pairs of scores for normality. From these tests we found that all sets of 907 differences are not normally distributed, so we cannot use the Student's t-test for these.

Randomization tests on the results of all $\binom{7}{2}$ combinations of two matching methods showed that all differences are significant, *except* Kullback-Leibler vs. K-nearest neighbours.

Next, we computed P_{wc} for all combinations (see Section 2.4.4). All P_{wc} values were below 0.09, except for Kullback-Leibler vs. K-nearest neighbours it was 0.34. The graph with the Wilcoxon P value depending on ϵ for these two methods was shown in Figure 2.5. From it we get a more detailed picture, and see that large subsets of the sets of paired scores do not differ significantly.

3.3.2 Other Performance Metrics: Macro Average Precision, DCG

Macro Average Precision

Other performance metrics show the same qualitative behavior: for example, Figure 3.3 shows the macro-averaged precision/recall plots for the same text matching methods, which are in the same order as the micro-averaged plots (in Figure 3.2).

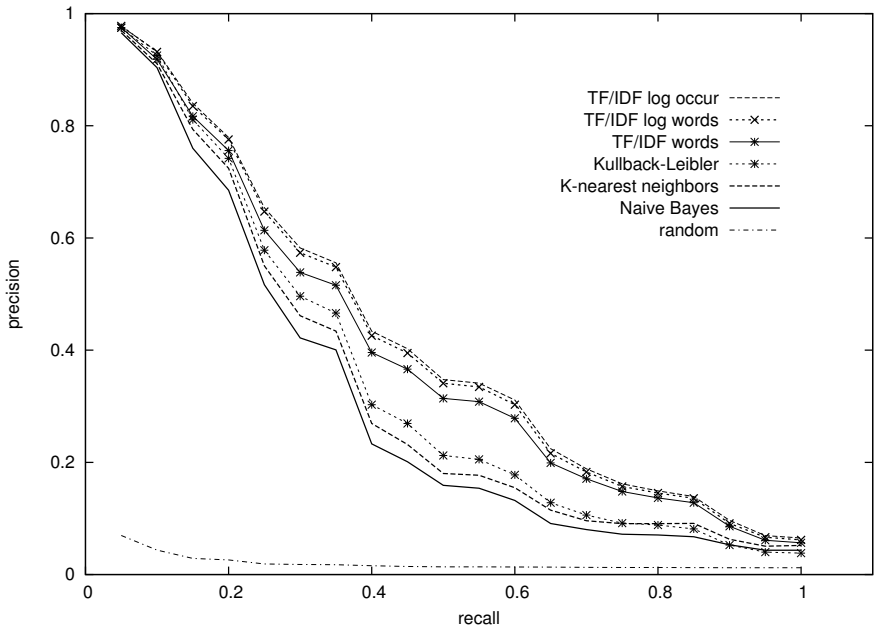


Figure 3.3: Macro-averaged precision/recall for the same methods as in Figure 3.2

Significance testing (both the Randomization and Student's t-test, normality testing showed only the distribution of differences between "Naive Bayes" and "TF/IDF log occur" to be approximately normally distributed though) showed that all differences in class averages were significant. P_{wc} values were all below 0.03.

Discounted Cumulative Gain

Table 3.1 shows the Discounted Cumulative Gain (DCG [21], see Section 2.3.2) for the same set of methods. Again following our significance testing procedure (outlined in Section 2.4), we found that (1) all distributions of differences of DCG scores are not normal, (2) both the Student's t-test and the Randomization test showed *all* differences to be significant, and (3) the P_{wc} value for all combinations were below 0.03. So, in short, all differences are statistically significant.

method	DCG
TF/IDF log occur	0.635
TF/IDF log words	0.631
TF/IDF words	0.613
Kullback-Leibler	0.580
K-nearest neighbours	0.565
Naive Bayes	0.539
random	0.272

Table 3.1: Average Discounted Cumulative Gain (DCG) [21] for the same methods as in Figure 3.2

3.3.3 The Best Method: TF/IDF

We can conclude that the TF/IDF-based methods perform the best on our data. We now briefly describe this method, for more details see [44].

The TF/IDF method assigns a vector of term weights to each document. A term's weight is based on its frequency in the document (Term Frequency (TF), higher is better), and its frequency over all documents (Inverse Document Frequency (IDF), lower is better, in other words, terms that do not appear in many *other* documents are more discriminating). More precisely, the weight is set to $tf \log(N/df)$, where tf is the term frequency in a document, N is the number of documents, and df is the term frequency over all documents[1]. For the "TF/IDF log occur" method, the document frequency is the number of documents in which a term occurs at least once. For the "TF/IDF log words" method, it is the total number of times a term occurs in all documents. Classes are represented by the sum of the vectors of their individual documents. The similarity score between two vectors is simply the cosine of the angle between them.

3.3.4 Other Methods

Other methods supported by the `rainbow` program were also tested but most failed to run to a finish. In particular, the methods "active" (Active Learning), "em" (Expectation Maximization), and "maxent" (Maximum Entropy) generated an assertion failure when creating the text index. The method "prind" (Probabilistic Indexing) generated an assertion failure for each query. We *were* able to run the method "SVM" (Support Vector Machines). However, it took on average about 180 times longer to process a query (when compared to "TF/IDF log occur", 3 seconds instead of 1/60th of a second on a 2.6 GHz Pentium IV) and the resulting average precision was 36% lower (but note that we did not experiment with the various parameter settings of the SVM method).

3.3.5 Parameter Settings

For most methods implemented in the `rainbow` program it is possible to adjust several parameters. We experimented with different parameters settings for the methods TF/IDF log occur, Naive Bayes, and K-nearest Neighbours, trying to maximize average precision for the training set. The settings were then evaluated using the test set.

Three different parameters are available for pruning words from the representative documents:
`--prune-vocab-by-infogain`
only keep the top N words with the highest information gain (also called *average mutual information* [7]),
`--prune-vocab-by-doc-count`
remove words that occur in $<= N$ documents, and
`--prune-vocab-by-occur-count`
remove words that occur $<= N$ times. For the last two parameters the documentation states $< N$ as the condition, but the `rainbow` implementation uses $<= N$.

[1] In the rainbow implementation, the first term is $\log(tf + 1)$ instead of tf (we ran tests to confirm that this extra log does not affect the results).

TF/IDF log occur

All three pruning strategies only worsen results for the TF/IDF log occur method. This may be explained by the fact that the presence of certain words, no matter how few other words they match, can only improve results, given that our database is relatively small: the chance that words generate incorrect "inter-class" matches is small.

Naive Bayes

The results for Naive Bayes were similar, except for `--prune-vocab-by-occur-count=2`, which produced an increase in average precision of about 5%.

Other parameters are specific to the Naive Bayes method:

`-naivebayes-normalize-log`
improves average precision by about 11% and the average DCG by about 8%

`--smoothing-method`
sets the method for smoothing word probabilities to avoid zeroes. The default method ("laplace") produced the best results

`--uniform-class-priors`
this makes no difference in the results, since in our case the class priors are all equal (1/907)

`-event-model=[word|document|document-then-word]`
the default model (`word`) produced the best results

Combining the two parameter settings that resulted in a performance increase did not improve performance any further (`--naivebayes-normalize-log` and `--prune-vocab-by-occur-count=2`).

K-nearest Neighbours

The K-nearest neighbours method also has a few method-specific parameters that can be set:

`-knn-k=`k: sets the number of nearest neighbours to use. The default value is 30. Settings in the range $[20, 100]$ yield similar results. Note that these are just the k best-scoring results that are returned. Depending on the class size distribution (of our classification), at some point k becomes large enough to include all relevant results. Making k even larger does not help anymore after that. By making k smaller we lose valid results

`--knn-weighting=<xxx>.<xxx>`: sets the term weighting options for the model and the query respectively. The first character sets the TF (term frequency) weighting, to one of the set {n, b, m, a, l}, meaning {none, binary, max-norm, aug-norm, log}, and corresponding to the value {f, 1, f / (max f in doc), 0.5 + 0.5 · (f / (max f in doc)), 1.0 + $\ln f$} (with f the actual term frequency). The second character sets the IDF (inverse document frequency), and is one of {n, t}, i.e. {none, tfidf = ln(total docs / docs containing term)}. The third character specifies if normalization is used, is one of {n, c}, equalling {none, 1 / $\sqrt{\sum (tf \cdot idf)^2}$} (summing over all terms)

After testing all possible combinations on the training set, we found the following:

- the TF weighting can be one of `lamn`, but not `b`, which worsens results
- the IDF weighting should be enabled
- the score normalization does not affect results

Evaluating using the test set with `--knn-weighting=ntn.ntn` results in a 1.5% improvement in average precision over the TF/IDF log occur method. Significance testing showed the following: the set of differences with TF/IDF log occur was not normally distributed, so the Student's t-test could not be used (it showed a significant difference at the 0.1 level, so not at the 0.05 level). A randomization test said it was insignificant, but barely (the P value was 5.6%). The P_{wc} value is 0.68, and the graph shows the difference is significant only for the *middle 30%* of scores. Randomization tests on the *macro* averages showed the difference between this method and the TF/IDF log methods to be highly insignificant (P values were 73% and 50%). Overall, we conclude that this improvement is statistically insignificant.

source	percentage in top 50
model file	100
synonyms and hypernyms	100
link	62
filename without digits	58
filename	58
path	56
page title	54
page context	50

Table 3.2: Percentage of all occurrences of each text source appearing in the best 50 combinations

3.3.6 Text Source Combinations

To determine the most useful combinations of text sources, we ran a classification test using each combination of n out of the eight text sources for the representative text document, with $n \in \{1, ..., 8\}$ (so the total number of combinations tested was $\sum_{n=1}^{8} \binom{8}{n} = 255$). The text matching method used was "TF/IDF log occur". The performance of each combination was measured as the average precision over twenty recall values.

From these tests, we found that adding as many text sources as possible improves overall performance, in general. This may be explained by our observation that the addition of keywords helps classification performance if the keywords are relevant, but does not hurt performance if they are irrelevant, since they do not match many other models. We expect that as the database size increases, this property will no longer hold because irrelevant keywords would generate cross-class matches.

Looking more closely at how often each source occurs in the best combinations, we counted the number of times each source appears in the top 50 combinations (i.e., the 50 combinations out of 255 with the highest average precision). The results are shown as percentages in table 3.2. We see that the identifiers found inside the 3D model files themselves provided the most information for classification. The WordNet synonyms and hypernyms also turned out to be very useful, despite the fact that for 279 models (31%) no synonym or hypernym was found (model names for which WordNet did not return a synonym or hypernym included names (e.g., "justin"), abbreviated words ("satellt"), misspelled words ("porche"), and words in a different language ("oiseau")).

3.3.7 Text Source Weights

Finally, we investigated if we could improve classification performance by adjusting the weights of each text source. Because the TF/IDF method computes a term's weight (i.e., importance) from its frequency, we can increase the weight of a text source by simply including it multiple times in the representative text document. In the previous experiments, all text sources were included once. We experimented with many different weight settings but found no significant improvement in classification performance. This may be explained by the effect that a single occurrence of a text source is sufficient for this type of matching method: a word is either present in another representative text document or it is not. Changing the number of times that a source appears in the document will change the matching score, but not the relative ordering of all documents.

3.4 Conclusions

The overall conclusions of this chapter are:

- text matching methods based on TF/IDF (term frequency, inverse domain frequency) scoring perform best on our data
- statistical significance tests show the performance difference to be significant (when compared to non TF/IDF-based methods)
- the most useful text sources were the text found inside the 3D model file itself, and the WordNet synonyms and hypernyms of the filename or link text
- simply using all text sources worked best, but this may be due to the relatively small size of our dataset: adding text sources is not likely to add false cross-class matches

Chapter 4

Shape Matching

In this section, we briefly review previous work on shape-based retrieval of 3D models. Then, we present results comparing several state-of-the-art shape matching methods to determine which works best on the Princeton Shape Benchmark 3D model database.

4.1 Related work

Retrieval of data based on shape has been studied in several fields, including computer vision, computational geometry, mechanical CAD, and molecular biology. For surveys of recent methods, see [27, 56]. For our purpose, we will only consider matching and retrieval of isolated 3D objects (so we do not consider recognition of objects in scenes, or partial matching, for example).

3D shape retrieval methods can be roughly subdivided into three categories: (1) methods that first attempt to derive a high-level description (e.g., a skeleton) and then match those, (2) methods that compute a feature vector based on local or global statistics, and (3) miscellaneous methods.

Examples of the first type are skeletons created by voxel thinning [54], and Reeb graphs from meshes [17]. However, the voxel-based methods are usually sensitive to noise and small features. The mesh-based methods typically require the input model to be 2-manifold. Unfortunately, many 3D models are created for visualization purposes only, and often contain only unorganized sets of polygons ("polygon soups"), possibly with missing, wrongly-oriented, intersecting, disjoint, and/or overlapping polygons. Fixing such degenerate models is a difficult open problem [16, 36].

Methods based on computing statistics of the 3D model are more suitable for our purpose, since they usually impose no strict requirements on the validity of the input model. Examples are shape histograms [1], feature vectors composed of global geometric properties such as circularity or eccentricity [57], and feature vectors (or *shape descriptors*) created using frequency decompositions of spherical functions [22]. The resulting histograms or feature vectors are then usually compared by computing their L_2 distance.

Some alternative approaches use 2D views (2D projections of a 3D model), justified by the heuristic that if two 3D shapes are similar, they should look similar from many different directions. Examples are the "prototypical views" of Cyr and Kimia [8], and the "Light Field Descriptor" of Chen *et al.* [6].

4.2 Shape Matching Methods

In our experiments, we considered four shape matching methods: (1) the *Light Field Descriptor* (LFD) [6], (2) the *Radialized Spherical Extent Function* (REXT) [59], (3) the *Gaussian Euclidian Distance Transform* (GEDT) [22], and (4) the *Spherical Harmonics Descriptor* (SHD) [22]. These four methods have been shown to be state-of-the-art in a recent paper [50]. Each of these methods first normalizes a 3D model for translation (by placing the origin at the center of mass) and scale (using the average distance of surface samples to the center of mass, for example). To review, we now give a short description of each shape descriptor.

LFD

a set of 10 2D projections as seen from half the vertices of a dodecahedron. To compare two LFDs, a similarity score is computed for each correspondence generated by 60 symmetries of a dodecahedron (i.e. all symmetries excluding reflections). This similarity score is the sum of the scores of 10 pairwise image comparisons (using an image matching method). The similarity score of two LFDs is then the minimum score of the 60 correspondences. Each 3D model is represented by 10 LFDs, evenly distributed across the viewing sphere [6]

REXT

a collection of spherical functions that first decomposes 3D space into concentric shells and then computes the Spherical Extent Function (a function returning the distance of a polygonal surface from the center of mass depending on spherical angle and radius) of the intersection of the model with each shell independently. The spherical harmonic decomposition of each spherical function is computed and the norms of the complex coefficients are stored. Shapes are compared by computing the L_2 distance between two sets of norms [59]

GEDT

a 3D grid function, whose value at each point is given by the composition of a Gaussian with the Euclidian Distance Transform (EDT) of the (rasterized) surface. Spherical representations are computed by intersecting the voxel grid with concentric spheres about the center and scaling each spherical function by the square root of the corresponding area. The spherical harmonic coefficients, up to a certain order (e.g., 16), are stored for each one, yielding a 3D descriptor. Shapes are compared by computing the L_2 distance between two descriptors. Note that this is the only method out of the four tested that is not rotation invariant [22]

SHD

a rotation invariant representation of the GEDT obtained by computing the restriction of the function to concentric spheres and storing the norm of each (harmonic) frequency, resulting in a 2D rotation invariant descriptor. Shapes are compared by computing the L_2 distance between two descriptors [22]

We ran an experiment in which these four methods were used to compute a similarity score for every pair of 3D models in the test set of the Princeton Shape Benchmark. The similarity scores were used to rank the matches for each model and compute an average precision-recall curve for each matching method with respect to the benchmark's base classification. Results are shown in Figure 4.1 (see [50] for details). From these curves, we find that the Light Field Descriptor provides the best retrieval performance in this test, and thus we use it in all subsequent experiments.

Normality tests on all pairs of differences showed that they are all not normally distributed. Each significance test showed that all differences are significant, *except* between GEDT and SHD: the randomization P value is 33% and P_{wc} is 1.62.

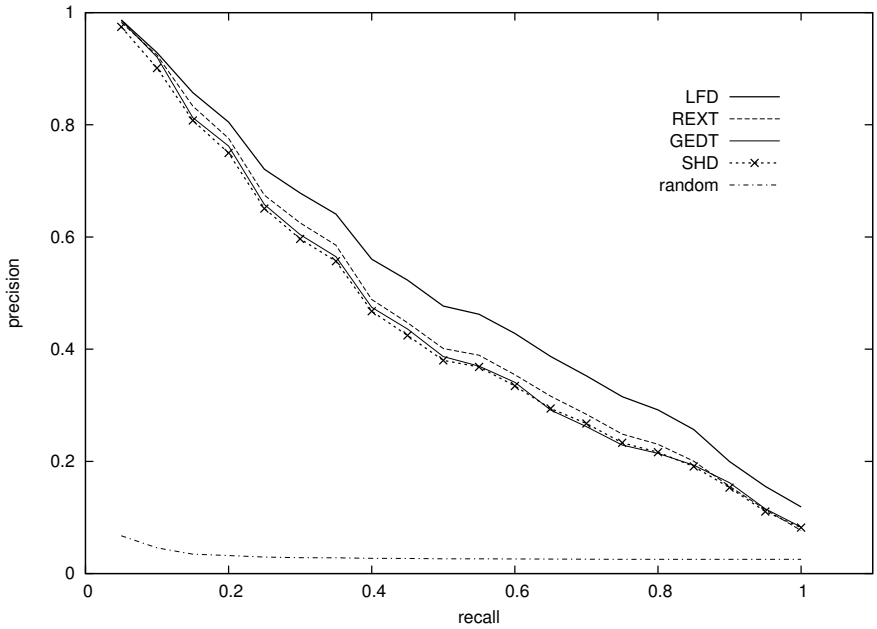


Figure 4.1: Average precision/recall for four shape matching methods, and random retrieval. This figure is a partial reproduction of one in [50]

Chapter 5

Comparing Text Matching to Shape Matching

Next, we compare the classification performance of the best text matching method to the best shape matching method. As simulated text queries we used (1) the model's representative document (as described in Section 3.2) and (2) the category names of the Princeton Shape Benchmark. Additionally, we examined retrieval performance when the query is left out of the results.

5.1 Using the Representative Text Documents as Queries

Figure 5.1 shows the resulting average precision/recall plot. The shape matching method significantly outperforms text matching: average precision is 44% higher. All significance tests show this difference to be significant.

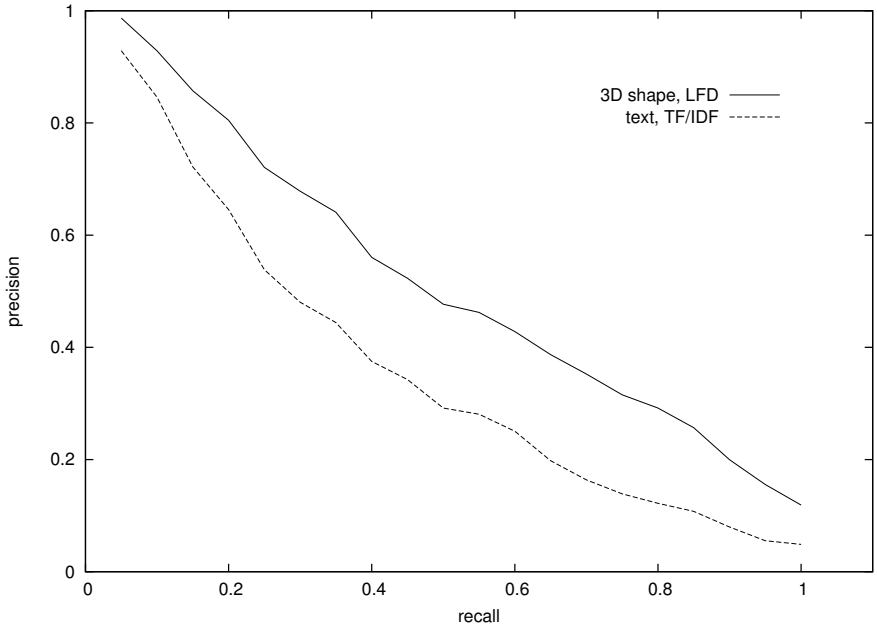


Figure 5.1: Average precision/recall for text and 3D shape matching, using a model's representative text document as a simulated user query

source	percentage
filename	100
path	99.6
model file	83
page title	79
synonyms and hypernyms	69
link	49
page context	24

Table 5.1: For each source, percentage of training set models *with* text

The main cause of the relatively poor performance of the text matching method is the low quality of text annotation of online 3D models: it is either missing, or of poor quality. Names of models were, for example:

- meaningless (e.g., "avstest" for a face)
- misspelled (e.g., "ferrar" for a ferrari)
- too specific (e.g., "camaro" for a car)
- not specific enough (e.g., "model" for a car)
- in a different language (e.g., "oiseau" for a bird)

By running a spell checker (`aspell, http://aspell.sourceforge.net`) on the filenames with the digits removed, we found that 36% of all model filenames were not English words.

Furthermore, often for several potential text sources there simply was no useful text available. For example, many link texts were either a repeat of the filename, or contained no text at all: for 446 models in the training set (51%) no link text could be found (usually a thumbnail image is used instead). Table 5.1 shows for each source for what percentage of the models in the training set text could be found.

Even commercial 3D model databases are not necessarily well annotated. Of three commercial databases available to us (provided by CacheForce, De Espona, and Viewpoint, containing approximately 2000, 1000, and 1000 models respectively), only one was consistently well annotated.

5.2 Using Category Names as Queries

In all text matching experiments, the representative document created for each 3D model was used as a query. However, because the size and quality of text annotation varies a lot from model to model, one may argue that this text is not representative of actual user queries. Users of a retrieval system are more likely to enter a few descriptive keywords or class names. To investigate classification performance given this kind of user queries, we ran an additional classification experiment in which the full category names of the Princeton Shape Benchmark were used as simulated user queries (e.g. "plane stealth bomber f117" for the category with models of F117 planes). Some obvious keywords were added manually (e.g., "blimp" and "zeppelin" for the "dirigible hot air balloon" category, or "house" for the "home" category, see [50] for a complete list of the category names).

Figure 5.2 shows the resulting precision/recall plot. The average precision achieved when using these query keywords was 11% higher than when using the representative documents, but still 30% lower than the best shape matching method. Statistical significance tests show the results when using category names as queries to be significantly higher than when using the representative text documents as queries, and significantly lower than the shape matching scores (randomization P values were 0% in both cases, P_{wc} was 0.24 and 0.01 respectively).

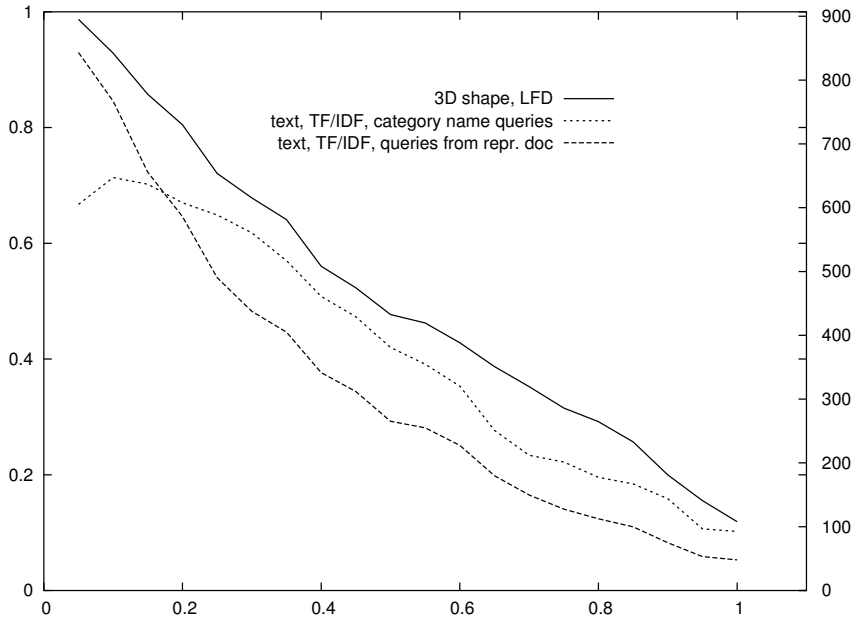


Figure 5.2: Comparing shape matching (micro averaged precision) scores to text matching scores when using category names as queries and when using the models' representative documents as queries

5.3 Leaving the Query out of the Results

In all retrieval tests, the query itself is part of the ranked results and usually appears at or near the top position. If we leave the query itself out of the results, any reasonable matching method will show a drop in average precision. This drop is relatively small when category names are used as queries (as described in Section 5.2) because in this case the only change is in the number of relevant results. Figure 5.3 shows the resulting average precision/recall graph. Overall average precision is 0.389 for the shape matching method, and 0.350 and 0.233 for the text matching methods.

Statistical significance testing (using all three tests) show that now the difference between shape matching and text matching using representative documents as queries is still significant (Randomization $P = 0\%$, $P_{wc} = 0.003$), but *no longer* significant when using category names as queries (Randomization $P = 59\%$, $P_{wc} = 6.29$).

Note that for a fair comparison we should also run an experiment in which we first identify 3D models as "category representatives" for each category, and then use these as simulated shape queries (i.e. the equivalent of using the text category name queries).

5.4 Conclusions

The main conclusion of this chapter is that for our type of data (3D models downloaded from the web), shape matching performs better in classification tasks than text matching. The reason is the poor quality of text annotation of such 3D models.

Note that the values of our performance measures depend a lot on the way our test database is classified. For example, certain shape matching results (e.g. a fighter jet query returning both fighter jets and passenger planes) can be considered "good" results, yet are punished because the results are members of multiple classes. The effect of the classification granularity on performance metrics, as well as new metrics for hierarchical classifications, are the subject of future research.

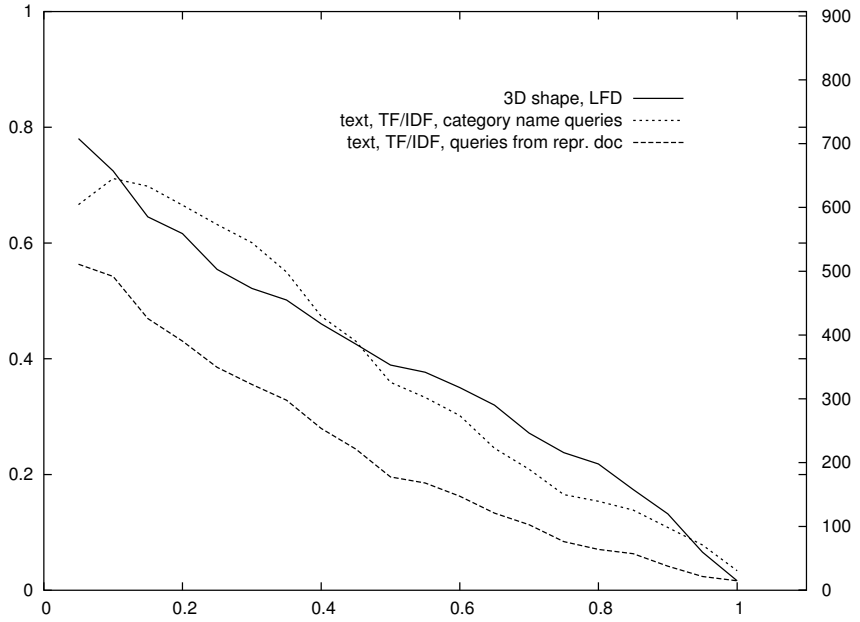


Figure 5.3: Comparing shape matching (micro averaged precision) scores to text matching scores when using category names as queries and when using the models' representative documents as queries, but this time leaving the query out of the results

Chapter 6

Combining Text and Shape Matching

In our final tests, we investigate how to combine the best text and shape matching methods to provide better retrieval results than can be achieved with either method alone.

6.1 Related Work

A considerable amount of research has been presented on the problem of how to best combine multiple classifiers [20]. Most work in this area has been done in content-based image retrieval. For example, Srihari presents a system ("Piction") that identifies faces in annotated news photographs using a face detection algorithm and Natural Language Processing of the captions [53]. Smith and Chang describe a system for the retrieval of arbitrary online images [51]. Relevant text for each image is extracted from the URL and the `alt` parameter of the HTML `<img>` tag, for example. However, searches based on low-level image features or on text can not be combined. This combination *has* been investigated in later retrieval systems. La Cascia *et al.* combine text and image features into a single feature vector, to improve search performance [47]. Text is extracted from the referring web page, with different weights assigned depending on the HTML tag that enclosed it (e.g., text in a `<title>` is more important than text in an `<h4>` (small header) tag). Paek *et al.* present a method that combines a text- and image-based classifier for the classification of captioned images into two classes ("indoor" and "outdoor") [37], which improved classification accuracy to 86.2%, from 83.3% when using the text alone. In recent work, Sable presents results for the same data set (but using different training and test subsets), and shows an improvement in classification accuracy from 84% to 84.9% (averaged over seven text classifiers) [40].

In a review on multimodal classification in video indexing, Snoek and Worring write that this classification method is an "emerging trend in video indexing research" [52]. They present a framework for multimodal retrieval of video, using video, audio, and text classifiers.

Jain *et al.*, in their survey paper on statistical pattern recognition, distinguish methods by the execution order of the classifiers: parallel, serial, or hierarchical [20]. Sebastiani calls a set of multiple classifiers a *classifier committee*, which is defined by (1) a choice of k classifiers and (2) a choice of combination function [49]. Both papers suggest a number of alternatives for the combination function. Many choices for the combination function exist. The simplest are static combiners, for example, voting, (weighted) averaging, or minimum. Others are trainable combiners, for which classifiers are trained such that their independence is maximized. Examples include *bagging* and *boosting* [46]. Typically many classifiers are needed for these methods, however.

6.2 Multiclassifiers

In previous work we suggested that the results of text and shape matching can be combined to improve classification performance [14], and proposed a combination function that simply averaged mean-normalized matching scores. However, no evaluation was done to see which combination function works best. Here

we consider the simple case of combining the scores of two classifiers, using a static combination function. We experimented with four types of functions: (1) linear weighted average, (2) minimum, (3) weighted average/minimum rank, and (4) using confidence limits. Each combiner (except the ones based on rank) was also tested on mean-normalized scores.

1. **linear weighted average:** If s_{text} and s_{shape} are the matching scores of a pair of models, then the combined score is $w \cdot s_{text} + (1 - w) \cdot s_{shape}$, with w the weight setting. We computed average precision/recall for $w \in \{0, 0.05, 0.1, ..., 1.0\}$, and picked the value of w which resulted in the highest overall precision. The optimal weight setting for the training set was $(0.1 \cdot s_{text} + 0.9 \cdot s_{shape})$

2. **minimum:** the lowest matching score (signifying highest similarity) is returned

3. **rank:** The matching scores are ordered, and the resulting rank of each query becomes its new matching score. We can then apply one of the first two functions (linear weighted average or minimum)

4. **confidence limits:** The "confidence limits" method is based on the idea that if a similarity score of a single classifier is sufficiently close to zero, then that classifier can be trusted completely. The output of other classifiers is then ignored. Sable uses a variant of this method when combining a text- and image-based classifier [40]: feature vectors from both are classified using a Support Vector Machine, and a confidence level is assigned to the classification, depending on the distance of the vector from the dividing hyperplane (the decision boundary). If the confidence level of the image-based classifier is high enough, then the text-based classifier is ignored. If not, then the text-based classifier is used and the image-based classifier is ignored.

   We used the training set to determine optimial limit settings of 0.09 and 0.22 for shape and text matching respectively (and -2.45 and -1.5 for mean-normalized scores). If both scores were above their limit, we reverted to the linear weighted average (other alternatives yielded worse results)

Figure 6.1 shows (in colour) the average precision/recall obtained using eight different combination methods and shape matching alone (on the PSB test set). Table 6.1 shows the resulting average precision values achieved for each combiner, and the percentage improvement over using shape matching alone (computed using the test set). The top three methods achieve an additional 7-11% improvement in overall average precision. The third column of Table 6.1 shows P values of the Randomization significance test, which shows that the top three methods produce a significant improvement. Further tests showed these three methods to be significantly different from one another as well[1] To conclude, this means that the combination method "weighted average of normalized scores" produces the largest improvement over shape matching alone.

method	average precision	% improvement over shape alone	randomization P value
shape (LFD)	0.496	-	-
weighted average, normalized scores	0.550	11	0
weighted average	0.537	8.3	0
confidence limits	0.533	7.5	0
minimum, normalized scores	0.497	0.2	93%
minimum	0.497	0.2	45%
minimum rank	0.487	-1.8	8.6%
confidence limits, normalized scores	0.481	-3.0	4.0%
weighted average rank	0.480	-3.4	0.1%

Table 6.1: Average precision achieved when combining the matching scores of the text and shape matching methods using various static combiners, and the percentage improvement over shape matching alone

[1] Even though none of all possible pairs of difference are normally distributed, the Student's t-test showed the same significance results as the randomization test. Also, P_{wc} values were low (¡ 0.02) for the top three methods, high ($[0.35 - 3.3]$) for the bottom five.

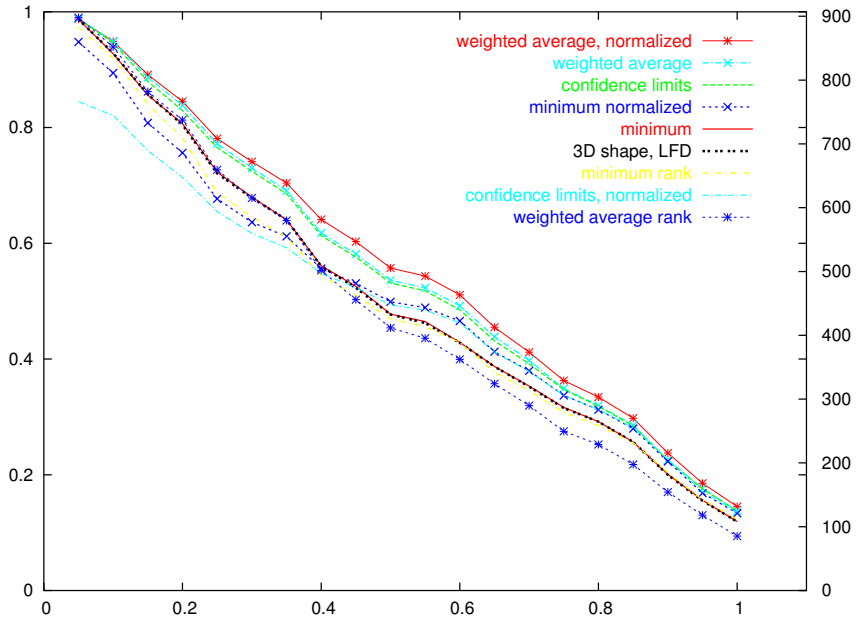


Figure 6.1: Average precision/recall achieved when combining the matching scores of the text and shape matching methods using various static combiners (colour graph)

These results confirm that the text and shape representations of a 3D model are sufficiently independent, such that when they are combined, they become more discriminating. There may well be other representations (e.g. appearance-based) that capture a very different aspect of a 3D model, and as such could increase performance even further.

6.3 Conclusions

The main conclusion of this chapter is that by using a simple static combination function on the shape and text matching scores, a significant retrieval performance improvement can be achieved.

Chapter 7

Conclusions and Future Work

This report evaluates text and shape matching methods for retrieval of online 3D models, as well as their combination, and is an extended version of the paper [34]. Classification tests were done using the Princeton Shape Benchmark, a large benchmark database of 3D models downloaded from the web.

For text matching, we found that a variant of TF/IDF showed the best classification performance. Text found inside a 3D model file itself and synonyms and hypernyms (category descriptors) of the model filename were most useful for classification.

The currently best shape matching method (which uses *Light Field Descriptors* [6]) significantly outperformed the best text matching method, yielding 44% higher average precision. The main reason is that the quality of text annotation of online 3D models is relatively poor, limiting the maximum achievable classification performance with a text-based method.

We investigated several simple multiclassifiers, and found that a function returning the weighted average of normalized matching scores produced a statistically significant improvement in average precision of about 11%.

The main contribution of this work is that it demonstrates the advantage of using shape-based matching methods over text-based methods for retrieval of 3D models. This should encourage designers of future 3D model retrieval systems to incorporate query methods based on shape, and other attributes that do not depend on annotation provided by humans, as they hold much potential for improving retrieval results.

A separate contribution is a chapter on evaluation metrics and statistical significance testing of the results of classification experiments. Unlike in our earlier paper [34], all results have now been tested for statistical significance. We found that in all cases the Student's t-test and the Randomization test (on means of matched pairs) produced identical significance results, even though normality tests showed that we should not use the Student's t-test.

For statistical significance testing we propose a new metric P_{wc}, the *weighted cumulative probability*, computed using Wilcoxon's Sum of Signed Ranks statistical test. It can be used as an indication of the statistical significance of differences across a distribution. If P_{wc} is relatively high (e.g. > 0.25), then the graph from which it is computed should be examined to gain insight into which subset of the differences is significant (e.g. the middle 30%, the top 25%, etc.).

Some topics for future work are:

- In this work we found that the text *inside* the 3D model file is relatively useful for classification. This text is the combination of several different text sources found inside the file (e.g. metadata, texture filenames, part names, etc.). Next, we would like to investigate what the relatively value is of each of these sources
- Applying Natural Language Processing (NLP) as an extra preprocessing step to the source text
- Incorporating other attributes of 3D models, such as colour, texture, and structural information, perhaps using more sophisticated multiclassifiers
- Developing new performance metrics for hierarchical classifications
- Investigating if shape matching and clustering methods can be used to improve the text annotation of 3D models

Acknowledgements

We thank Professor Richard Gill of the Department of Mathematics of Utrecht University for his many useful comments on the statistical significance testing of our results. Patrick Min was supported in part by the AIM@SHAPE Network of Excellence grant 506766 by the European Commission. The National Science Foundation provided partial funding for this project under grants CCR-0093343 and IIS-0121446.

Appendix: Detailed Results of Statistical Significance Tests

This appendix has the results of all statistical significance tests we ran. Each table cell has, from top to bottom: (1) the significance value of the normality test, (2) the significance value of the Student's t-test, (3) the P value of the Randomization test, and (4) the P_{wc} value.

If for the normality test or the t-test the absolute t value is too high (i.e. beyond the highest value in our lookup table), then the cell contains **NOT SIG**. If for the t-test the significance value is higher than 0.05 then the value is printed in bold. The same is true for the Randomization P value. In this way it is easy to see that both tests always produce the same qualitative answer.

			1	2	3	4	5	6
0	**TF/IDF log occur**	norm	0.01	0.01	0.01	0.01	0.01	0.01
		t-test	0.0005	0.0001	0.0001	0.0001	0.0001	0.0001
		rand	0.0001	0	0	0	0	0
		P_{wc}	0.0482	0.0757	0.0216	0.012	0.0388	0.0005
1	**TF/IDF log words**	norm		0.01	0.01	0.01	0.01	0.01
		t-test		0.0001	0.0001	0.0001	0.0001	0.0001
		rand		0	0	0	0	0
		P_{wc}		0.0249	0.0169	0.0139	0.0401	0.0007
2	**TF/IDF words**	norm			0.01	0.01	0.01	0.01
		t-test			0.0001	0.0001	0.0001	0.0001
		rand			0	0	0	0
		P_{wc}			0.0204	0.0143	0.0394	0.0037
3	**Kullback-Leibler**	norm				0.01	0.01	0.01
		t-test				**0.2**	0.0001	0.0001
		rand				**0.1767**	0	0
		P_{wc}				0.337	0.087	0.0085
4	**K-nearest neighbours**	norm					0.01	0.01
		t-test					0.0001	0.0001
		rand					0	0
		P_{wc}					0.0536	0.0102
5	**Naive Bayes**	norm						0.01
		t-test						0.0001
		rand						0
		P_{wc}						0.0126
6	**Random**	norm						
		t-test						
		rand						
		P_{wc}						

Table 7.1: Text matching methods, micro-average precision (Section 3.3.1)

			1	2	3	4	5	6
0	**TF/IDF log occur**	norm	0.01	0.01	0.01	0.05	**NOT SIG**	0.01
		t-test	0.05	0.0005	0.0001	0.0001	0.0001	0.0001
		rand	0.012	0	0	0	0	0
		P_{wc}	0.0084	0.0128	0.0072	0.0075	0.0053	0.0269
1	**TF/IDF log words**	norm		0.01	0.01	0.05	0.05	0.01
		t-test		0.0001	0.0001	0.0001	0.0001	0.0001
		rand		0	0	0	0	0
		P_{wc}		0.0024	0.0078	0.0082	0.0035	0.029
2	**TF/IDF words**	norm			0.01	0.01	0.05	0.01
		t-test			0.0001	0.0001	0.0001	0.0001
		rand			0	0	0	0
		P_{wc}			0.0101	0.011	0.0074	0.0272
3	**Kullback-Leibler**	norm				0.01	0.01	0.01
		t-test				0.01	0.0001	0.0001
		rand				0.0025	0	0
		P_{wc}				0.0183	0.0053	0.0145
4	**K-nearest neighbours**	norm					0.01	0.01
		t-test					0.001	0.0001
		rand					0	0
		P_{wc}					0.009	0.0218
5	**Naive Bayes**	norm						0.01
		t-test						0.0001
		rand						0
		P_{wc}						0.0245
6	**Random**	norm						
		t-test						
		rand						
		P_{wc}						

Table 7.2: Text matching methods, macro-average precision (Section 3.3.2)

			1	2	3	4	5	6
0	**TF/IDF log occur**	norm	0.01	0.01	0.01	0.01	0.01	0.01
		t-test	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
		rand	0	0	0	0	0	0
		P_{wc}	0.0349	0.0243	0.0136	0.0072	0.0068	0.0026
1	**TF/IDF log words**	norm		0.01	0.01	0.01	0.01	0.01
		t-test		0.0001	0.0001	0.0001	0.0001	0.0001
		rand		0	0	0	0	0
		P_{wc}		0.0134	0.0113	0.0073	0.0055	0.0029
2	**TF/IDF words**	norm			0.01	0.01	0.01	0.01
		t-test			0.0001	0.0001	0.0001	0.0001
		rand			0	0	0	0
		P_{wc}			0.008	0.0071	0.0072	0.0049
3	**Kullback-Leibler**	norm				0.01	0.01	0.01
		t-test				0.0001	0.0001	0.0001
		rand				0	0	0
		P_{wc}				0.0274	0.0254	0.0067
4	**K-nearest neighbours**	norm					0.01	0.01
		t-test					0.0001	0.0001
		rand					0	0
		P_{wc}					0.0138	0.0084
5	**Naive Bayes**	norm						0.01
		t-test						0.0001
		rand						0
		P_{wc}						0.0091
6	**Random**	norm						
		t-test						
		rand						
		P_{wc}						

Table 7.3: Text matching methods, DCG (Section 3.3.2)

			1	2	3
0	**LFD**	norm	0.01	0.01	0.01
		t-test	0.0001	0.0001	0.0001
		rand	0	0	0
		P_{wc}	0.0134	0.0285	0.0077
1	**REXT**	norm		0.01	0.01
		t-test		0.01	0.0005
		rand		0.0081	0.0001
		P_{wc}		0.118	0.135
2	**GEDT**	norm			0.01
		t-test			**NOT SIG**
		rand			**0.3251**
		P_{wc}			1.616
3	**SHD**	norm			
		t-test			
		rand			
		P_{wc}			

Table 7.4: Shape matching methods (Section 4.2

			1	2
0	**3D shape, LFD**	norm	0.01	0.01
		t-test	0.0001	0.0001
		rand	0	0
		P_{wc}	0.0054	0.0116
1	**text, using repr. docs**	norm		0.01
		t-test		0.0001
		rand		0
		P_{wc}		0.2341
2	**text, using category names**	norm		
		t-test		
		rand		
		P_{wc}		

Table 7.5: Comparing shape matching, text matching using representative documents as queries, and text matching using category names as queries (Section 5.2)

			1	2
0	**3D shape, LFD**	norm	0.01	0.01
		t-test	0.0001	**NOT SIG**
		rand	0	**0.5941**
		P_{wc}	0.0029	6.2924
1	**text, using repr. docs**	norm		0.01
		t-test		0.0001
		rand		0
		P_{wc}		0.009
2	**text, using category names**	norm		
		t-test		
		rand		
		P_{wc}		

Table 7.6: Comparing shape matching, text matching using representative documents as queries, and text matching using category names as queries, and leaving the query out of the results (Section 5.3)

			1	2	3	4	5	6	7	8
0	**3D shape, LFD**	norm	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
		t-test	0.0001	0.0001	0.001	0.05	0.0001	**NOT SIG**	**NOT SIG**	**0.1**
		rand	0	0	0.0005	0.0395	0	**0.4485**	**0.9298**	**0.0859**
		P_{wc}	0.0078	0.0134	0.8544	1.9831	0.0124	0.3671	3.2962	1.5035
1	**weighted avg.**	norm		0.01	0.01	0.01	0.01	0.01	0.01	0.01
		t-test		0.0001	0.0001	0.0001	0.0005	0.0001	0.0001	0.0001
		rand		0	0	0	0.0002	0	0	0
		P_{wc}		0.0057	0.015	0.0667	0.0572	0.0066	0.2932	0.0437
2	**weighted avg., norm.**	norm			0.01	0.01	0.01	0.01	0.01	0.01
		t-test			0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
		rand			0	0	0	0	0	0
		P_{wc}			0.0088	0.0213	0.0234	0.0118	0.0724	0.0137
3	**weighted avg. rank**	norm				0.01	0.01	0.01	0.01	0.01
		t-test				**NOT SIG**	0.0001	0.0005	0.005	**0.2**
		rand				**0.8025**	0	0.0003	0.0031	**0.1803**
		P_{wc}				4.3711	0.0137	0.9577	0.0862	1.5829
4	**conf. limits, norm.**	norm					0.01	0.01	0.01	0.01
		t-test					0.0001	0.05	0.0001	**0.2**
		rand					0	0.0274	0	**0.1766**
		P_{wc}					0.0428	1.9067	0.0052	1.9649
5	**conf. limits**	norm						0.01	0.01	0.01
		t-test						0.0001	0.0001	0.0001
		rand						0	0	0
		P_{wc}						0.0066	0.2133	0.0392
6	**minimum**	norm							0.01	0.01
		t-test							**NOT SIG**	**0.1**
		rand							**0.9834**	**0.0596**
		P_{wc}							2.9484	1.3982
7	**minimum, norm.**	norm								0.01
		t-test								0.005
		rand								0.0024
		P_{wc}								0.51
8	**minimum rank**	norm								
		t-test								
		rand								
		P_{wc}								

Table 7.7: Comparing shape matching and text+shape matching using various static combination functions (Section 6.2)

Bibliography

[1] A.P.Ashbrook, N.A.Thacker, P.I.Rockett, and C.I.Brown. Robust recognition of scaled shapes using pairwise geometric histograms. In *Proc. BMVC*, pages 503–512, Birmingham, UK, July 1995.

[2] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.

[3] A. B. Benitez and S.-F. Chang. Semantic knowledge construction from annotated image collections. In *Proc. Int. Conf. on Multimedia and Expo ICME*, Lausanne, Switzerland, 2002.

[4] J. Chambers, W. Cleveland, B. Kleiner, and P. Tukey. *Graphical Methods for Data Analysis*. Chapman & Hall, 1983.

[5] D.-Y. Chen and M. Ouhyoung. A 3D object retrieval system based on multi-resolution Reeb graph. In *Proc. Computer Graphics Workshop*, pages 16–20, Taiwan, June 2002.

[6] D.-Y. Chen, M. Ouhyoung, X.-P. Tian, Y.-T. Shen, and M. Ouhyoung. On visual similarity based 3D model retrieval. In *Proc. Eurographics*, Granada, Spain, September 2003.

[7] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley-Interscience, 1991.

[8] C. M. Cyr and B. B. Kimia. 3D object recognition using shape similarity-based aspect graph. In *Proc. ICCV*. IEEE, 2001.

[9] M. de Buenaga Rodríguez, J. M. Gómez-Hidalgo, and B. Díaz-Agudo. Using WordNet to complement training information in text categorization. In *Proc. RANLP*, Stanford, March 1997.

[10] T. G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1923, 1998.

[11] J. Filliben. The probability plot correlation coefficient test for normality. *Technometrics*, 17(1):111–117, 1975.

[12] FindSounds. Sound file search engine. `http://www.findsounds.com`.

[13] R. A. Fisher. *Design of Experiments*. Macmillan, New York, 9th edition edition, 1971.

[14] T. Funkhouser, P. Min, M. Kazhdan, J. Chen, A. Halderman, D. Dobkin, and D. Jacobs. A search engine for 3D models. *ACM Transactions on Graphics*, 22(1), January 2003.

[15] Google. Image search. `http://www.google.com/images`.

[16] A. Gueziec, G. Taubin, F. Lazarus, and W. Horn. Converting sets of polygons to manifold surfaces by cutting and stitching. *IEEE Visualization*, pages 383–390, 1998.

[17] M. Hilaga, Y. Shinagawa, T. Kohmura, and T. L. Kunii. Topology matching for fully automatic similarity estimation of 3D shapes. In *Proc. SIGGRAPH*, pages 203–212, 2001.

[18] D. Huijsmans and N. Sebe. How to complete performance graphs in content-based image retrieval: Add generality and normalize scope. *IEEE PAMI*, 27(2), February 2005.

[19] D. Hull. Using statistical testing in the evaluation of retrieval experiments. In *Proc. ACM Conference on Research and Development in Information Retrieval*, pages 329–338, Pittsburgh, USA, 1993.

[20] A. K. Jain, R. P. Duin, and J. Mao. Statistical pattern recognition: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):4–37, 2000.

[21] K. Järvelin and J. Kekäläinen. IR evaluation methods for retrieving highly relevant documents. In *Proc. ACM SIGIR*, pages 41–48, Athens, Greece, 2000.

[22] M. Kazhdan, T. Funkhouser, and S. Rusinkiewicz. Rotation invariant spherical harmonic representation of 3D shape descriptors. In *Proc. SGP*, pages 156–165. ACM, June 2003.

[23] T. R. Knapp. Comments on the statistical significance testing articles. *Research in the Schools*, 5(2):39–41, 1998.

[24] S. Kullback and R. Leibler. On information and sufficiency. *Ann. Math. Stat.*, 22:79–86, 1951.

[25] N. I. T. Laboratory. Critical values of the normal ppcc distribution. `http://www.itl.nist.gov/div898/handbook/eda/section3/eda3676.htm`.

[26] D. V. Library. DVL/Verity stop words list. `http://dvl.dtic.mil/stop_list.html`.

[27] S. Loncaric. A survey of shape analysis techniques. *Pattern Recognition*, 31(8), 1998.

[28] R. Lowry. Concepts and applications of inferential statistics. `http://faculty.vassar.edu/lowry/webtext.html`.

[29] A. McCallum. Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. `http://www.cs.cmu.edu/~mccallum/bow`, 1996.

[30] MeshNose. 3D objects search engine. `http://www.deepfx.com/meshnose`.

[31] G. A. Miller. WordNet: A lexical database for English. *CACM*, 38(11):39–41, 1995.

[32] P. Min. *A 3D Model Search Engine*. PhD thesis, Princeton University, January 2004.

[33] P. Min, A. Halderman, M. Kazhdan, and T. Funkhouser. Early experiences with a 3D model search engine. In *Proc. Web3D Symposium*, pages 7–18, St. Malo, France, March 2003. ACM.

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

[35] T. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[36] T. Murali and T. Funkhouser. Consistent solid and boundary representations from arbitrary polygonal data. In *SIGGRAPH Symposium on Interactive 3D Graphics*, pages 155–162, March 1997.

[37] S. Paek, C. L. Sable, V. Hatzivassiloglou, A. Jaimes, B. H. Schiffman, S.-F. Chang, and K. R. McKeown. Integration of visual and text-based approaches for the content labeling and classification of photographs. In *ACM Workshop on Multimedia Indexing and Retrieval*, 1999.

[38] M. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[39] C. S. E. Project. `plotmtv`, 1995.

[40] C. Sable. *Robust Statistical Techniques for the Categorization of Images Using Associated Text*. PhD thesis, Columbia University, 2003.

[41] C. Sable, K. McKeown, and K. W. Church. NLP found helpful (at least for one text categorization task). In *Proc. EMNLP*, Philadelphia, PA, 2002.

[42] C. L. Sable and V. Hatzivassiloglou. Text-based approaches for the categorization of images. In *Proc. Research and Advanced Technologies for Digital Libraries*, Paris, 1999.

[43] G. Salton. *The SMART retrieval system*. Prentice-Hall, Englewood Cliffs, NJ, 1971.

[44] G. Salton. *Automatic text processing: the transformation, analysis, and retrieval of information by computer*. Addison-Wesley, Reading, Massachusetts, 1988.

[45] S. L. Salzberg. On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Mining and Knowledge Discovery*, 1(3):317–327, 1997.

[46] R. Schapire. The strength of weak learnability. *Machine Learning*, 5:197–227, 1990.

[47] S. Sclaroff, M. L. Cascia, and S. Sethi. Unifying textual and visual cues for content-based image retrieval on the world wide web. *CVIU*, 75(1-2):86–98, July/August 1999.

[48] S. Scott and S. Matwin. Text classification using WordNet hypernyms. In *Proc. Workshop Usage of WordNet in Natural Language Processing Systems*, pages 45–52, August 1998.

[49] F. Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, March 2002.

[50] P. Shilane, M. Kazhdan, P. Min, and T. Funkhouser. The Princeton Shape Benchmark. In *Proc. Shape Modeling International*, pages 167–178, Genoa, Italy, June 2004.

[51] J. R. Smith and S.-F. Chang. Searching for images and videos on the world-wide web. Technical Report 459-96-25, Columbia University, August 1996.

[52] C. G. Snoek and M. Worring. Multimodal video indexing: A review of the state-of-the-art. In *Proc. Multimedia Tools and Applications*, 2003.

[53] R. K. Srihari. Automatic indexing and content-based retrieval of captioned images. *IEEE Computer*, 28(9):49–56, September 1995.

[54] H. Sundar, D. Silver, N. Gagvani, and S. Dickinson. Skeleton based shape matching and retrieval. In *Proc. SMI*, Seoul, Korea, May 2003.

[55] M. T. Suzuki. A web-based retrieval system for 3D polygonal models. In *Proc. IFSA/NAFIPS*, pages 2271–2276, Vancouver, Canada, July 2001.

[56] J. W. Tangelder and R. C. Veltkamp. A survey of content based 3D shape retrieval methods. In *Proc. Shape Modeling International*, pages 145–156, Genoa, Italy, June 2004.

[57] G. Taubin and D. Cooper. *Geometric Invariance in Computer Vision*, chapter Object recognition based on moment (or algebraic) invariants. MIT Press, 1992.

[58] C. J. van Rijsbergen. *Information Retrieval*. Butterworth, 1979.

[59] D. V. Vranić. An improvement of rotation invariant 3D shape descriptor based on functions on concentric spheres. In *Proc. ICIP*, volume 3, pages 757–760, September 2003.

[60] E. W. Weisstein. Paired t-test. MathWorld, `http://mathworld.wolfram.com/Pairedt-Test.html`.

[61] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics*, 1(6):80–83, December 1945.