A 3D Model Search Engine

Patrick Min

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
By the Department of
Computer Science

January 2004

Abstract

This thesis describes an online search engine for 3D models, focusing on query interfaces and their corresponding model/query representations and matching methods.

A large number of 3D models has already been created, many of which are freely available on the web. Because of the time and effort involved in creating a high-quality 3D model, considerable resources could be saved if these models could be re-used. However, finding the model you need is not easy, since most online models are scattered across the web, on repository sites, project sites, and personal homepages.

To make these models more accessible, we have developed a prototype *3D model search engine*. This project serves as a test bed for new methods in web crawling, query interfaces, and matching of 3D models. This thesis focuses on query interfaces and their accompanying matching methods. We investigated query interfaces based on text keywords, 3D shape, 2D shape, and some combinations.

By testing matching methods that use text, 3D shape, and 2D shape, we found that the 3D shape matching method outperforms our text matching method for our application domain, due to the insufficient text annotation of 3D models on the web. Furthermore, classification performance was improved by combining the 3D shape- and text-based matching methods. The results of a user study also suggest that text can combine with shape to make queries more effective.

We compared shape matching methods based on matching multiple 2D projections of a 3D model and found that from a set of side, corner and edge projections, the combination of side and corner projections produced the best results. However, these results were still worse than those of the 3D shape matching method.

We present a 2D structural interface and accompanying matching method that produces better classifications than image matching for certain types of objects.

Our prototype search engine has been used extensively across the world (in 18 months, almost 300,000 queries have been processed from more than 100 countries) and has proven to be useful (models have been downloaded over 50,000 times, and almost 30% of all visitors per day are returning users).

Acknowledgments

First and foremost I would like to thank my advisor, Tom Funkhouser, and my advisor in my first two years, David Dobkin: Tom for his patience and his persistence in trying to turn me into a scientist, and David for his guidance and for always having time. Many thanks also to my other committee members, Szymon Rusinckiewicz, Andrea LaPaugh and David Jacobs, and former committee member Adam Finkelstein.

Many students played an important part in this project. I would like to thank Alex Halderman, who wrote the crawler, Joyce Chen, who ran two user studies, and David Bengali and Phil Shilane for their work in creating 3D model test databases. And of course thanks to Misha Kazhdan, for his excellent work, and with whom it was great to collaborate.

I wish to acknowledge the help of the supporting technical and administrative staff at the Princeton Computer Science department: Joe Crouthamel, Steve Elgersma, Ginny Hogan, Chris Kranz, Tina McCoy, Melissa Lawson, Jim Roberts, Chris Tengi, and Trish Zabielski.

Thanks to Viewpoint Corporation, Jose Maria De Espona, and Cacheforce for donating large collections of 3D models for use in our experiments. I would also like to acknowledge the National Science Foundation, who provided partial funding for this project under grants CCR-0093343, 11S-0121446, CCR-99-88173, and DGE-9972930, and the Army Research Organization, who provided partial funding under grant DAAD19-99-1-0205. My advisor, Tom Funkhouser, is partially supported by an Alfred P. Sloan Fellowship.

A large number of friends made my time in Princeton special. In no particular order, and with some sadness because opportunities to see each other are fewer and further between: Rudro Samanta, Sanjeev Kumar and Sushma Bhope, Steve and Judy Kleinstein, Jason Lawrence, Iannis Tourlakis, Amal Ahmed, Tom Weaver, Susan Uhm, Mariesha Blazik, Patricia Sung, Chris Dunworth, Bob and Heather Thomas, Alejo Hausner, Wenjia Fang, Emil Praun, Allison Klein, Misha Kazhdan, Tony Wirth, Lena Petrović, Petra Sijpestein, Riky Schmidt, Wagner Corrêa, and George Karakostas. To my friends in the Netherlands, who fortunately still kept in touch: Rob Klein, Nico van Paridon, Chris Nyqvist, Arjen van der Ziel, Daniëlle Hemels, Lizza Westerhof, Jacqueline Neumann, Reinoud van Leeuwen, Hanno Liem, Florine Asselbergs, and Karolien van der Ven.

Thanks to my parents, for always supporting me, no matter what I chose to do. And finally, to my dear Katerina, for being part of my life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Search Engines

The world-wide web is changing the way we find and use information. It provides access to a vast amount of text and image data on every conceivable topic. Unfortunately, the sheer quantity of information can make it difficult to quickly find what you are looking for. To aid in this search, many *search engines* exist that index large portions of the web. They typically provide a search interface based on text keywords: a user enters some descriptive keywords (e.g. "boeing 747"), after which web pages containing these keywords are returned. Examples of popular text-based search engines are Google [17] and AltaVista [3].

However, the web does not just contain text pages, but a lot of non-textual data as well, such as images, sound files, or CAD models. Many so-called *specialized* search engines targeting these specific kinds of data have been developed. Perhaps the biggest such search engine is Google Image Search [37], which indexes hundreds of millions of images. Other examples are FindSounds [34], a search engine for sound files, and MeshNose [67], a search engine for 3D models. Many of these specialized search engines take advantage of the fact that even though the indexed objects are of a non-textual type, often they are annotated with descriptive text. The search engine then simply tries to match the user-entered keywords to this descriptive text.

3D Models

One such non-textual data type is the 3D model, the basic building block of many operations in 3D computer graphics applications. 3D models are used in, for example, Computer Aided Design (CAD): a designer uses a 3D modeling tool to create a 3D representation (i.e. a 3D model) of a new product. This model can then for example be visualized in different colors and lighting conditions. It can also serve as a blueprint for guiding the subsequent manufacturing process. Because creating such a high-quality 3D model takes a lot of time and effort, it would be beneficial to have the option of re-using and possibly adapting existing 3D models. Perhaps someone else created a similar product before, and the designer can start by adapting an existing model instead of having to create one from scratch. Existing 3D models could also be re-used, for example, in the creation of a virtual environment (e.g. for an architectural walkthrough), as characters in a game, or for movie special effects.

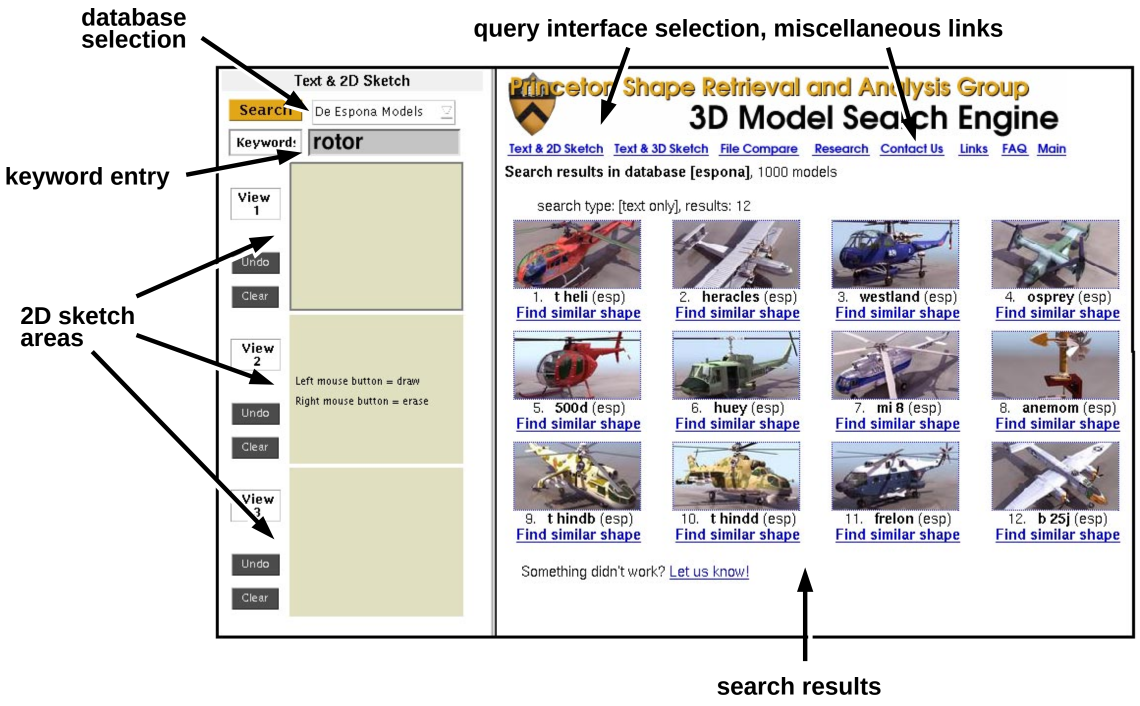
The number of existing 3D models is already large, and we expect this number to increase at a growing rate for three reasons. First, high-performance PC graphics hardware has become very affordable, creating an increased demand for 3D models to be used in applications such as games, online stores, scientific visualizations, and so on. Second, it is becoming easier to create 3D models, using new and improved methods to acquire models from the real world (e.g. using 3D scanners [24], computer vision [35]), and new modeling tools (e.g. Maya [2], which is now free for non-commercial use, Blender [13], a free modeling system, Teddy [45], a Java applet for 3D sketching). The improvement in PC hardware performance also enables many more people to use these 3D modeling tools to create 3D models themselves. Third, as is true for all digital data, a lot of these 3D models are available on the web for free download.

A 3D Model Search Engine

To improve the accessibility of all these online 3D models, we have created a prototype specialized search engine for 3D models. It supports a text-based query interface and several different types of shape-based query interfaces. Figure 1.1 shows a screenshot of the actual search engine web site, with its main components annotated. On the left side of the page the user can enter queries and specify which database is to be searched (both free and commercial databases have been indexed). On the right side search results are shown, as well as links that allow the selection of different query interfaces and miscellaneous pages with a feedback form, information about our research, and links to similar projects elsewhere.

Challenges

The challenges in creating such a search engine are threefold. First, the 3D models available on the web have to be found. Unfortunately, most of the online 3D models are scattered across the web, on repository sites, project sites, and personal homepages. Second, it is not obvious what the best query interface is for searching a database of 3D models. Additionally, an important goal for our search engine is that it is useable by the average user. By "average user" we mean someone who knows what "3D" means, and is able to manipulate a 3D model using a viewing interface, but not necessarily a computer graphics professional. This means that the query interfaces should be intuitive and easy to use. And finally, user-entered queries have to be matched to 3D models. For each type of query a suitable matching method needs to be developed. Also, because the search engine is an interactive system, a matching method has only limited time to run.



Left mouse button = draw
Right mouse button = erase

Search results in database [espona], 1000 models

search type: [text only], results: 12

Something didn't work? Let us know!

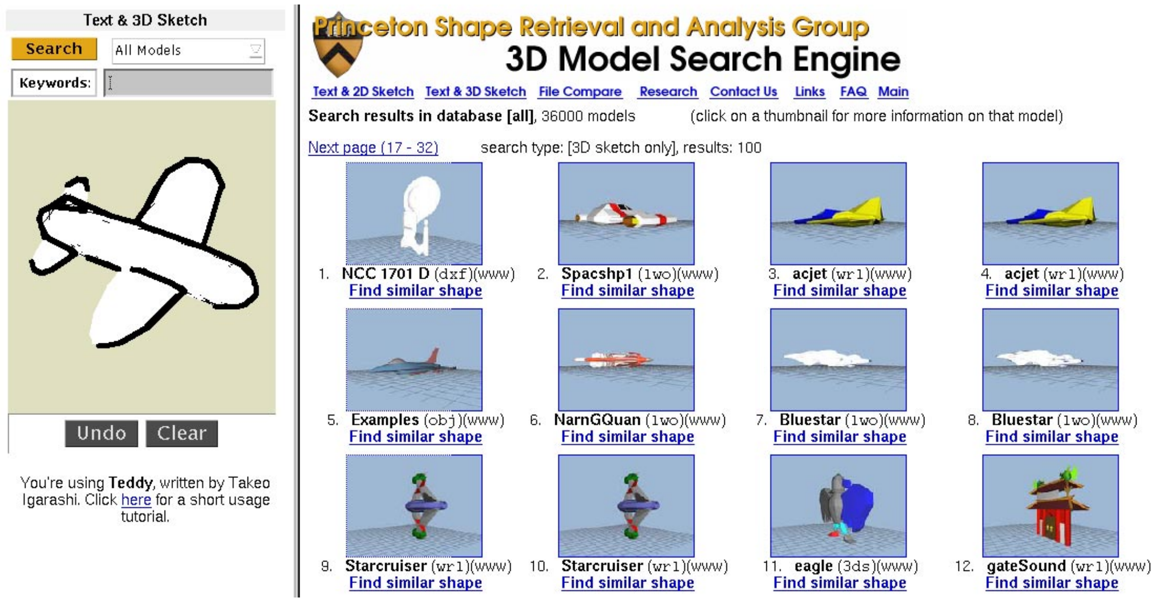
Figure 1.1: The main components of the search engine web site, with an example text query "rotor" and its results from a commercial model database provided by De Espona [27]

Query Interfaces

This thesis focuses on query interfaces for our 3D model search engine, and their corresponding model/query representations and matching methods. 3D models typically contain textual, appearance (color, texture) and shape information, which can all be queried. We investigated query interfaces based on text keywords, 3D shape, 2D freeform sketches, and 2D structural input (i.e. sets of parts), and their associated matching methods. We now describe each query interface in more detail.

Text Queries

Because text query interfaces are very common and potentially very effective, we developed a text query interface and accompanying matching method for our search engine (see Figure 1.1 for an example text query "rotor" and the results from a commercial database provided by De Espona [27]). The interface works as follows. User-entered keywords are matched to a representative text document, one of which has been created for each 3D model. This text document contains text from the model file itself (e.g. its filename, part names) and from the web page it was found on (e.g. the link text, web page title). A lexical database called WordNet [68] was used to add synonyms and hypernyms (category descriptors) of the filename or link text



You're using **Teddy**, written by Takeo Igarashi. Click here for a short usage tutorial.

Search results in database [all], 36000 models (click on a thumbnail for more information on that model)

search type: [3D sketch only], results: 100

1. **NCC 1701 D** (dxf)(www) Find similar shape
2. **Spacshp1** (lwo)(www) Find similar shape
3. **acjet** (wrl)(www) Find similar shape
4. **acjet** (wrl)(www) Find similar shape
5. **Examples** (obj)(www) Find similar shape
6. **NarnGQuan** (lwo)(www) Find similar shape
7. **Bluestar** (lwo)(www) Find similar shape
8. **Bluestar** (lwo)(www) Find similar shape
9. **Starcruiser** (wrl)(www) Find similar shape
10. **Starcruiser** (wrl)(www) Find similar shape
11. **eagle** (3ds)(www) Find similar shape
12. **gateSound** (wrl)(www) Find similar shape

Figure 1.2: An example 3D sketch query using Teddy, and the first 12 results

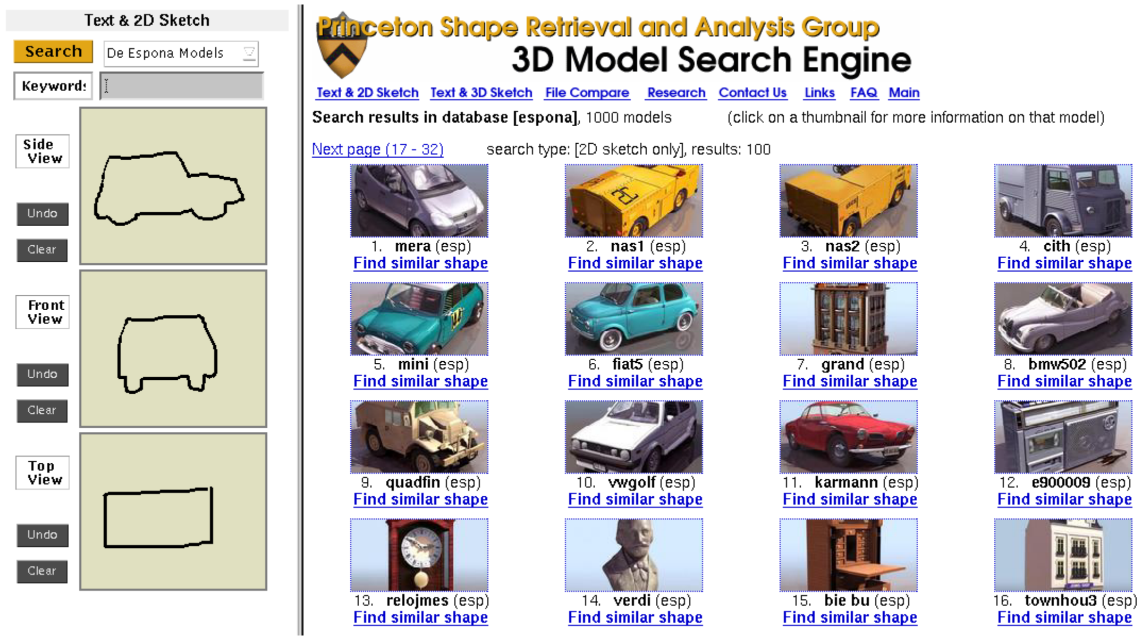
to the representative text document. We investigated the classification performance of several different text matching methods, operating on all possible combinations of eight different text sources of a 3D model. We found that the TF/IDF text matching method [89] produced the best results, and that the text found inside a model file and the WordNet synonyms and hypernyms were the most useful for classification purposes.

3D Shape Queries

A defining attribute of a 3D model is its *shape*. It therefore makes sense to allow the user to query using shape, for example, by providing a rough sketch of what the desired object looks like. We investigated several shape-based query interfaces, initially supporting only 3D query shapes: the user may submit an existing 3D model, either by (1) selecting a 3D model that was the result of a previous search (implemented as a "Find Similar Shape" link below a thumbnail image of a result, see Figure 1.1), or by (2) uploading a local 3D model file. A 3D model can also be (3) created from scratch, using a simple online 3D modeling tool called Teddy [45]. Figure 1.2 shows an example 3D query shape created using Teddy, and some search results. From the usage results of our search engine we found that the "Find Similar Shape" method is the most popular of these three methods. It also generates the most user interest in the search results, measured by the percentage of searches that result in at least one model download. The simple 3D modeling tool has so far proven to be too complex for most users of our search engine.

2D Sketch Queries

Because of the apparent complexity of the 3D sketch interface, we decided to develop a simpler 2D sketch based query interface. Here the user can submit freeform sketches of a target object as seen from up to three directions. It is similar to the pen-drawing interface in Paintbrush-style programs: pixels can be drawn by dragging the mouse. The sketches are matched to several 2D projections of each 3D model using an image matching method. An example 2D sketch query and some results are shown in Figure 1.3.



Search results in database [espona], 1000 models (click on a thumbnail for more information on that model)

search type: [2D sketch only], results: 100

1. mera (esp) 2. nas1 (esp) 3. nas2 (esp) 4. cith (esp)
5. mini (esp) 6. fiat5 (esp) 7. grand (esp) 8. bmw502 (esp)
9. quadfin (esp) 10. vwgolf (esp) 11. karmann (esp) 12. e900009 (esp)
13. relojmes (esp) 14. verdi (esp) 15. bie bu (esp) 16. townhou3 (esp)

Figure 1.3: An example 2D sketch query and its results

Based on the results of a user study in which subjects were asked to draw several sketches of various objects, we decided to match these sketches to exterior outlines of 2D projections of the 3D models in our database. All possible pairings of user sketches to these projections are compared, with individual pairs of images being matched using an existing image matching method. The question then arises from which viewpoints 2D projections of a 3D model should be stored. We created thirteen 2D projections for each 3D model, subdivided into side views (three, looking from the center of the side of a bounding cube towards its center), corner views (four, from the corners), and edge views (six, from the edges). All possible combinations of using side, corner, and edge views were evaluated by running classification tests using a test database of 1,093 models donated by Viewpoint [107], classified into 84 categories. We found that using the side and corner views, and matching models by finding the best-matching three out of those seven views, resulted in the best classification performance, which was still worse than that of the 3D shape matching method, however. Furthermore, the matching performance of the online 2D sketch interface is negatively affected by the poor quality of the sketches that are submitted: they are often inaccurate and contain interior detail. The image matching method we use is very sensitive to interior detail, and as a consequence, searches using the 2D freeform sketch interface result in very few model downloads.

2D Structural Queries

We attempted to circumvent the drawback of having to draw exactly the right image by developing a 2D *structural* query interface. There are many objects that belong to the same class, but look slightly different because of variations in the size, shape, position, and orientation of their parts (e.g. humans, tables, helicopters). Furthermore, evidence has been presented in the perception literature that humans think of shape as composed of a set of simpler, basic shapes. Biederman showed in his seminal "Recognition by Components" experiments that humans tend to partition shape into convex parts, whose boundaries are recognized near regions of deep concavity [11]. To capture this notion in a shape query interface, we developed a parts-based query interface and accompanying matching method, in which the user provides a 2D *structure*. Initially we experimented with an interface based on drawing sets of ellipses (see Figure 1.4 for an example query).

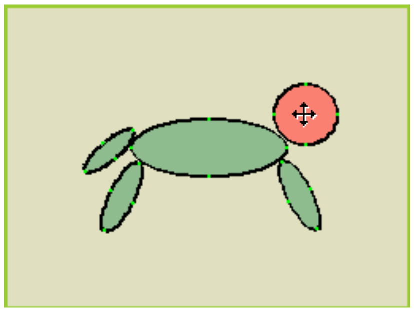


Figure 1.4: An example parts-based query of a 2D structural query interface

The ellipses of a query are placed into a tree structure using heuristics based on size and distance. Each ellipse (except the root) is then parameterized in terms of its parent. A set of ellipses is matched to a 2D projection of a 3D model by running a non-linear optimization that minimizes an error-of-fit function. This function rewards (1) ellipses overlapping the 2D image and (2) ellipses aligning with the local maxima of the Euclidian Distance Transform of the image, i.e. with the Medial Axis of the image [14]. This second term takes advantage of the structural significance of the Medial Axis, without suffering from its sensitivity to noise (a common problem of matching methods using the Medial Axis). The error-of-fit function also penalizes ellipses overlapping each other, and excessive part deformation. The method parameters were optimized for classification using a training set of 75 3D models classified into 15 categories of 5 models each. The method was evaluated using a test set of the same size. We found that for certain classes of objects, this matching method can improve classification performance over an image matching method. Objects of these classes usually have a clear, similar structure, and a relatively large variation in their parts (e.g. tables). However, the non-linear optimization is computationally expensive, and as a result this method is not yet suitable for use in an interactive system.

Matching Methods Comparison

We compared the classification performance of the three matching methods we use for our online 3D model search engine (i.e. text-, 3D shape-, and 2D shape-based). For this comparison we used a test database of 1,000 3D models downloaded from the web and classified into 81 categories. This test database was created using our full database of 31,000 downloaded models as a source, and as such is representative of the kind of data we have to process.

In a classification test, a similarity score is computed for each model by comparing its representation (e.g. representative text document, 3D shape descriptor) to that of all other models. The models are then ranked by their similarity score, and *precision/recall* values for this model are computed. Precision values are averaged over all models. For an explanation of the precision/recall performance metric see Appendix A.

We found that the 3D shape-based matching method significantly outperform text-based matching. This is mainly due to the insufficient text annotation of models found on the web. They are often contained in lists, annotated with their filename or a thumbnail image only. The text-based matching still outperforms the 2D shape-based method. Also, we found that by combining the matching scores of the text- and 3D shape-based methods, classification performance improved even further.

The Online Search Engine

Each query interface, except the one based on 2D structure, has been available online on our search engine site (at `http://shape.cs.princeton.edu`) for about a year and a half, yielding extensive usage results. We use the percentage of searches (of a particular query type) that result in at least one downloaded model as an indication of the user interest in the search results, and hence the quality of the results. We found that the text-based search and the "Find Similar Shape" search (in which a result from a previous search is submitted as a 3D shape query) generated the highest percentage of model downloads. These were also the most popular query interfaces. The sketch-based interfaces (both 2D and 3D) resulted in the smallest percentage of downloads. Currently (May 2003), the search engine has a steady number of visitors, processing about 4,000 searches per week from about 1,200 different hosts, with a total of almost 300,000 processed queries and over 53,000 model downloads. About 25% of the visitors each day are returning users.

Summary and Outline

Now we summarize the main conclusions of this thesis. First of all, 3D shape matching outperforms text matching in classification tests of models downloaded from the web, mainly due to the insufficient text annotation of web models. The 2D shape-based matching method performed worse than the text matching method. Combining matching scores of the 3D shape- and the text-based matching methods further improves classification performance. For certain classes of objects, a method based on matching 2D structural representations produces better results than one based on matching images. Finally, the practical result of this work, an online 3D model search engine, has proven to be useful to the general public, evidenced by the large number of users and model downloads, and the large percentage of returning users.

This thesis is organized as follows. The following chapter discusses related work in content-based retrieval, including other 3D model search engines. The next four chapters examine four types of query interfaces: text, 3D shape, 2D sketch, and 2D structure, respectively. The matching methods we use and some combinations are compared in Chapter 7. Usage results of our search engine are in Chapter 8, followed by conclusions and future work in Chapter 9. A discussion on the precision/recall metric used throughout this thesis may be found in Appendix A. Finally, Appendix B has implementation details.

Chapter 2

Related work

2.1 Introduction

To help cope with the unprecedented increase in the amount of available online information, many different *search engines* have been created. In its most general form, a search engine is a program that facilitates access to a database, consisting of a front-end where a user can enter a query, and a back-end that performs the actual search. The front-end is called the *query interface*. This query interface can be made available through a web page, making it possible to access a database from anywhere on the internet. For example, such a page could allow a user to search in a national phonebook or a product catalog, from any internet-connected PC.

If the database being searched contains data collected from other sites on the web, we call the search engine a *web search engine*. From here on, we will take *search engine* to mean *web search engine*. Search engines can be classified as *general* or *specialized*. Specialized search engines index data from a specific domain (e.g. paintings, molecules, research papers), as opposed to general search engines, which attempt to index a broad range of information. An important advantage of specialized search engines, of which our 3D model search engine is an example, is that they can have a domain specific query interface. A query for a painting, for example, could be a simple 2D sketch.

Search engines may be classified by the type of information they index, and the types of queries they support. Table 2.1 lists a few examples of search engine sites, for three types of information (text, 2D image, and 3D model) and three query data types (text, 2D shape, 3D shape).

The remainder of this chapter is organized by these information types: the following three sections discuss related work in search engines and matching methods for searching text, 2D images, and 3D models, respectively.

information type → ↓ query type	Text	2D Image	3D Model
Text	Google, CiteSeer	Google Image	MeshNose [67], CADLib [19]
2D Shape	×	QBIC [33]	Taiwan Univ. [20], Princeton [36]
3D Shape	×	×	ShapeSifter [23], Ogden IV [101]

Table 2.1: Some examples of search engines for combinations of the type of information indexed, and the type of query supported

2.2 Text Search Engines

Most of the online information exists in the form of text, which is why the largest and the most popular search engines are text-based. Forerunners in this area are large text-based general search engines such as Google [17] and Altavista [3]. There also exist specialized search engines for textual information. Examples are CiteSeer, a search engine for scientific papers [16], and HomePageSearch, for homepages of computer scientists [40].

Because these sites index text, their query interfaces are usually text-based as well. This means that user-entered text keywords have to be matched to an index of the text database. In this thesis, we will not investigate text query interfaces and matching methods in detail, but instead rely on existing work to select an appropriate matching method for our text query interface. We refer the interested reader to the large body of work in the indexing and matching of text documents. Seminal works include the books by Gerard Salton [88, 89], and many papers from the proceedings of the Text REtrieval Conference (TREC [104]) and the conference of the ACM Special Interest Group on Information Retrieval (SIGIR [94]).

Subject indices such as Yahoo [110] are no search engines because no search is performed: links to web pages are organized into a hierarchy (either manually or automatically), which the user can browse. This kind of interface is often used for accessing 3D model collections, however (also see Section 2.4.1).

2.3 2D Image Search Engines

Many search engines index non-textual data, such as images and sounds. In this section, we discuss related work in image search engines, subdivided into searching using associated text and using image content.

2.3.1 Text-based

Probably the largest site for searching images using text keywords is Google's image search. Unfortunately, no publications are available about the method they use. One of Google's FAQ pages states [38]:

> "How does image search work? Google analyzes the text on the page adjacent to the image, the image caption and dozens of other factors to determine the image content. Google also uses sophisticated algorithms to remove duplicates and ensure that the highest quality images are presented first in your results." (April 4, 2003)

This 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.

In related work, Sable and Hatzivassiloglou investigated the effectiveness of using associated text for classifying images from online news articles as indoor or outdoor [86], and found that image captions can be used to achieve an accuracy close to that of a human. An image-based classifier was less accurate.

2.3.2 Content-based

A lot of previous work has been done in content-based image retrieval (CBIR), where typically the user sketches a rough approximation of the desired image, which is then matched to images in a database. For surveys of CBIR methods and systems, see [7, 65, 96, 100, 106]. The methods vary by what the user input should be (e.g. arbitrary images, basic shapes, strokes), and how the sketches are matched (using low-level (statistical) or high-level (structural) matching). Because we cannot make assumptions about the position, scale, or orientation of the user input nor the database images, most methods attempt to be invariant under similarity transformations (i.e. translation, scale, rotation). For example, invariance to translation can be achieved by aligning the center of mass of the images to be matched, or by computing features that do not depend on image position (e.g. depending on the image boundary alone). A few examples of CBIR methods and systems follow.

Image Input

Matching pixels

In the Query by Visual Example (QVE) interface by Kato *et al.*, the user draws a simple binary sketch, which is then matched to edges detected in database images using pixel-by-pixel comparisons [53]. The matching score is computed by subdividing both images into 64 grid cells of 8×8 pixels each, and then adding for each cell a weighted combination of the number of matching image pixels, the number of matching background pixels, and the number of mismatches. To allow for inaccuracies, a query cell may be translated by up to δ pixels in the vertical, and ϵ pixels in the horizontal direction. The (δ, ϵ) combination with the best score is picked. As a result, this matching method is not invariant under similarity transformations. Also, the compensation for small differences in both images is limited to small translations in arbitrary rectangular parts of one image.

Matching wavelet signatures

A CBIR system using image matching is presented by Jacobs *et al.* [48]. Using their query interface the user can draw simple images using colored brush strokes. An image matching method based on comparing wavelet signatures is used to find the most similar images in a database. The matching method is quite sensitive to similarity transformations.

Matching edges using Fourier descriptor of boundary function

The Netra system by Ma and Manjunath also supports shape matching [63]. Edges are detected in a query image, which are then combined into closed contours. Next, shape descriptors are computed using the amplitudes of the Fourier transform of three types of boundary functions (with normalized arc length as a parameter): curvature, centroid distance and complex coordinate functions. Two descriptors are compared by computing their Euclidian distance.

Matching edges using histograms

For example, Huet and Hancock compute histograms of relative angles between pairs of line segments extracted from an image [42]. This still requires a step in which edges are extracted from the image, however.

Shape Input

Matching feature vectors of polygons

An example of a system which uses high-level matching is the shape matching component of the Query By Image Content (QBIC) project by Faloutsos *et al.* [33]. The user inputs a 2D polygon, which is converted to a feature vector (with features such as area, circularity, and major axis orientation). This vector is then matched to feature vectors of the outline shapes in a database. Some results are presented for performing six queries on a database of 259 airplane silhouettes.

Matching position and size of colored regions

In the VisualSEEk system the user can draw multiple regions of a certain color, and specify their spatial relationships [98]. Matching these regions to database images is done by comparing the position and size of the region's minimum bounding rectangles, and the ordering of the regions when they are projected onto the x and y axis (making the method orientation dependent).

Matching binary shapes using feature vectors, deformable templates, strokes

A recent application area of CBIR is trademark retrieval. Proposed new trademarks have to be sufficiently dissimilar from existing ones in order to be accepted for use. Jain and Vailaya use a two-stage approach, in which first feature-based matching is used to select a small number of potential matches from a larger database. Next, the boundary contour of a query is matched to this smaller set using deformable template matching [49]. Recently, Leung and Chen proposed converting trademark images into "strokes" (primitives such as circles, polygons, and lines), each with an associated confidence measure [60]. Segments of a trademark image are stored either as contours or skeletal edges (found by thinning the segment). A dissimilarity score is computed for two sketches based on similarity of corresponding strokes, and the distance between their centers.

Matching binary shapes using statistical matching

Other methods compute statistical invariants (i.e. features invariant under similarity transformations) of the image. Examples are moment invariants such as regular moments [79], Zernike moments [56], or invariants based on a decomposition in Angular Radial Transform (ART) basis functions (as used in the MPEG-7 region based descriptor) [15]. However, low order moments (e.g. principal axes) tend to be sensitive to the location of subparts in an image, and higher order moments are sensitive to noise.

A method developed by Michael Kazhdan is less sensitive to noise and the location of subparts [36]. Briefly, after normalizing for scale and translation, the Euclidian Distance Transform (EDT) of a 2D binary image is intersected with concentric circles, yielding a function on each circle. The amplitudes of the Fourier decomposition of each function are used as a 2D shape "signature". By just using the amplitudes the method is less sensitive to the location of subparts, and using the EDT makes it less sensitive to noise. This method is described in more detail in Section 5.3 and in [36].

2.4 3D Model Search Engines

Many web sites allow users to find 3D models. Examples are online repository sites, such as 3D Café [1] and Avalon [9], or 3D modeling company sites, such as Cacheforce [18] and Viewpoint [107]. Other sites, such as CadLib [19] and MeshNose [67], index multiple 3D model collections.

These sites can be classified according to the ways available to the user for searching the database.

2.4.1 Browsing a Hierarchy

Most 3D model repositories provide a browsing interface, where the user finds the desired model by traversing a hierarchy, selecting the right keyword at each level.

This interface has several drawbacks, however: (1) it requires that the models be manually organized in a hierarchy, which may be impractical for large databases, (2) an object may be a member of more than one class, (3) the user has to know beforehand which class a model is in, and (4) the possible results are restricted to members of semantic classes. If, for example, the user would like a list of stick-like objects, this will require browsing many classes that could contain such objects (lamps, swords, rifles, etc.). While browsing of a hierarchy may be useful for some applications, we have not investigated its use in our 3D model search engine.

2.4.2 Searching by Text Only

Other sites index 3D models using only text. Examples are CADlib [19], MeshNose [67], and the National Design Repository [31, 81], which index multiple 3D model collections. CADlib indexes a description, filename, id number, etc., of each CAD model. MeshNose simply indexes text that was found on the web pages of several 3D model repository sites. The National Design Repository allows searches by text keyword (querying the part name or filename), and by file type and size, or by browsing through directories.

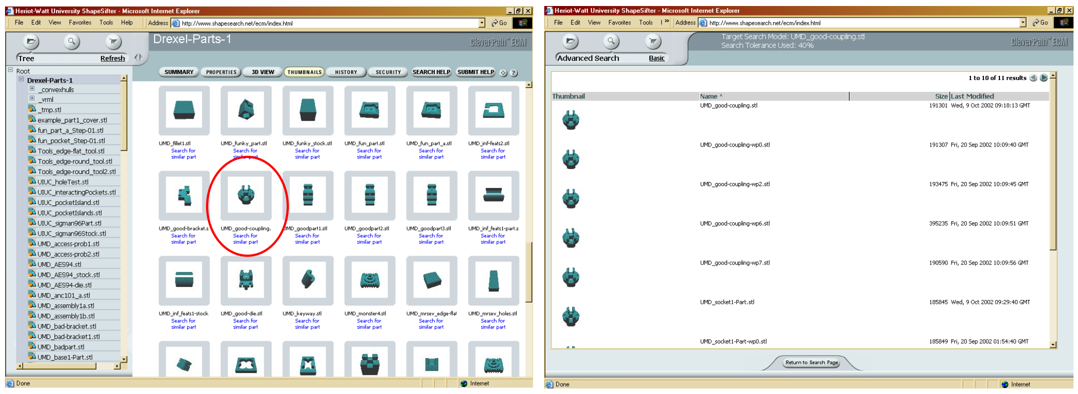
2.4.3 Searching by Shape

Recent 3D search engine sites allow searching based on shape and/or shape features. For example, at the "ShapeSifter" site of Heriot-Watt University [23], the user can select from a long list of shape features, such as surface area, bounding box diagonal length, and convex hull volume, and perform a search with conditions on these features. The search is in a CAD test database with 102 L-shaped blocks and several transformed versions of about 20 other models. Figure 2.1 shows an example in which a CAD part is submitted as a shape query.

In the online demo of the commercial system "Alexandria," the user can set weights of individual model attributes (for example "geometry," "angular geometry," "distributions," and "colour") to be used in matching, and search in a database of 4,500 commercial models (the database does seem to contain many identical models in different orientations, however) [74]. See Figure 2.2 for an example search using "Ephesus" (an online lightweight version of Alexandria), starting from a random model.

In the experimental system "Ogden IV," [101] the user can choose between matching grid-based and rotation invariant feature descriptors at several different grid resolutions, and search a database of 1,500 VRML models, which are not available for download. Figure 2.3 shows an example shape search starting from a random model.

At the experimental site "3D Shape Retrieval Engine" of Utrecht University [102, 105], the user can pick a query model by number, and one of three matching methods (Gaussian curvature, Normal variations, Midpoints) and one of three test databases:



Thumbnail	Name	Size	Last Modified
	UMD_good-coupling.stl	191301	Wed, 9 Oct 2002 09:18:13 GMT
	UMD_good-coupling-wp0.stl	191307	Fri, 20 Sep 2002 10:09:40 GMT
	UMD_good-coupling-wp2.stl	193475	Fri, 20 Sep 2002 10:09:45 GMT
	UMD_good-coupling-wp6.stl	395235	Fri, 20 Sep 2002 10:09:51 GMT
	UMD_good-coupling-wp7.stl	190590	Fri, 20 Sep 2002 10:09:56 GMT
	UMD_socket1-Part.stl	185845	Wed, 9 Oct 2002 09:29:40 GMT
	UMD_socket1-Part-wp0.stl	185849	Fri, 20 Sep 2002 01:54:40 GMT

(a) (b)

Figure 2.1: An example query using the "ShapeSifter" site of Heriot-Watt University [23]. The object marked in (a) is selected as a query. The closest matches are shown in (b)

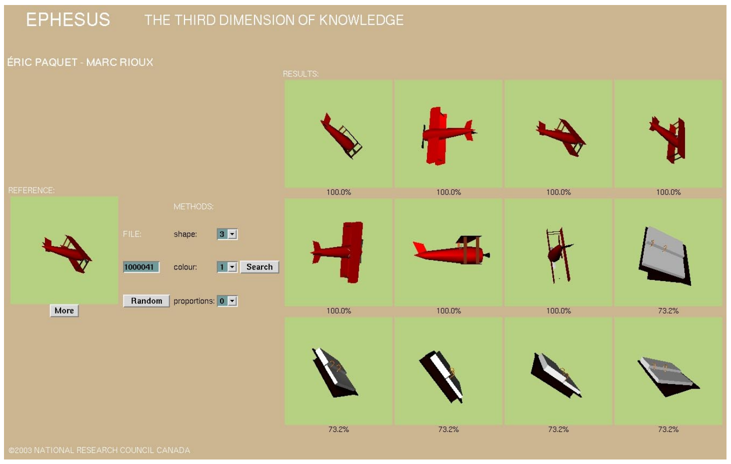


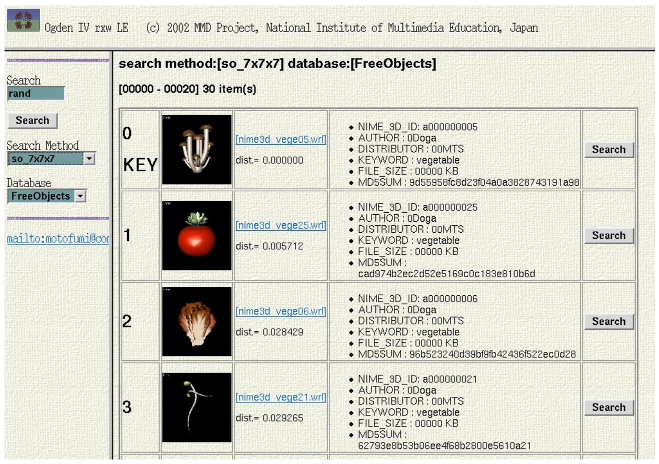
Figure 2.2: An example search using the "Ephesus" system [74]

(1) a database of 133 web models collected from the web by Osada *et al.* [72], (2) a database of 684 models (containing 366 airplanes) the authors collected from the web, and (3) the same database of 102 L-shaped blocks used in the ShapeSifter site. Figure 2.4 shows an example search using this site.

Another experimental site of the Informatics and Telematics Institute (ITI) in Thessaloniki, Greece, indexes 37 VRML models [47, 57]. The user can select one of these models, which is then matched against the other 36 using shape descriptors based on a set of geometric features. Figure 2.5 shows an example query and some search results.

The "Visual Query Interface" system at Arizona State University provides an elaborate 2D sketching interface for drawing the outline curve of a ceramic vessel [80, 84]. A user-drawn curve is matched to a database of outline curves of scanned 3D models of vessels. Figure 2.6 shows an example query being constructed with this system.

A “3D Object Retrieval System” from National Taiwan University [20] supports a few query methods similar to ours: the user can enter text keywords, and draw one or two 2D sketches (not outline sketches, but filled-in shapes), or select a result model from a previous search as a query. The 2D sketches are matched to 2D projections of a 3D model as seen from the vertices of a dodecahedron, with an image matching method using the MPEG-7 region based descriptor [95]. 3D models are matched using the same 2D projections. The model database has 10,000 models, manually downloaded from the web. See Figure 2.7 for an example 2D sketch query using this system.

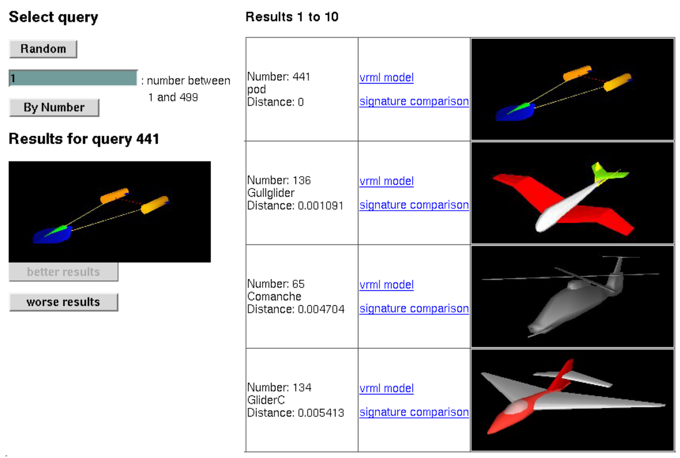

Ogden IV rxw LE (c) 2002 MMD Project, National Institute of Multimedia Education, Japan
Search
rand
Search
Search Method
so_7x7x7
Database
FreeObjects
mailto:motofumi@co

search method:[so_7x7x7] database:[FreeObjects]

[00000 - 00020] 30 item(s)

0 KEY		[nime3d_vege05.wrl] dist.= 0.000000	• NIME_3D_ID: a000000005 • AUTHOR : 0Doga • DISTRIBUTOR : 00MTS • KEYWORD : vegetable • FILE_SIZE : 00000 KB • MD5SUM : 9d55958fc8d23f04a0a3828743191a98	Search
1		[nime3d_vege25.wrl] dist.= 0.005712	• NIME_3D_ID: a000000025 • AUTHOR : 0Doga • DISTRIBUTOR : 00MTS • KEYWORD : vegetable • FILE_SIZE : 00000 KB • MD5SUM : cad974b2ec2d52e5169c0c183e810b6d	Search
2		[nime3d_vege06.wrl] dist.= 0.028429	• NIME_3D_ID: a000000006 • AUTHOR : 0Doga • DISTRIBUTOR : 00MTS • KEYWORD : vegetable • FILE_SIZE : 00000 KB • MD5SUM : 96b523240d39bf9fb42436f522ec0d28	Search
3		[nime3d_vege21.wrl] dist.= 0.029265	• NIME_3D_ID: a000000021 • AUTHOR : 0Doga • DISTRIBUTOR : 00MTS • KEYWORD : vegetable • FILE_SIZE : 00000 KB • MD5SUM : 62793e8b53b06ee4f68b2800e5610a21	Search

Figure 2.3: An example search using the online “Ogden IV” system [101]

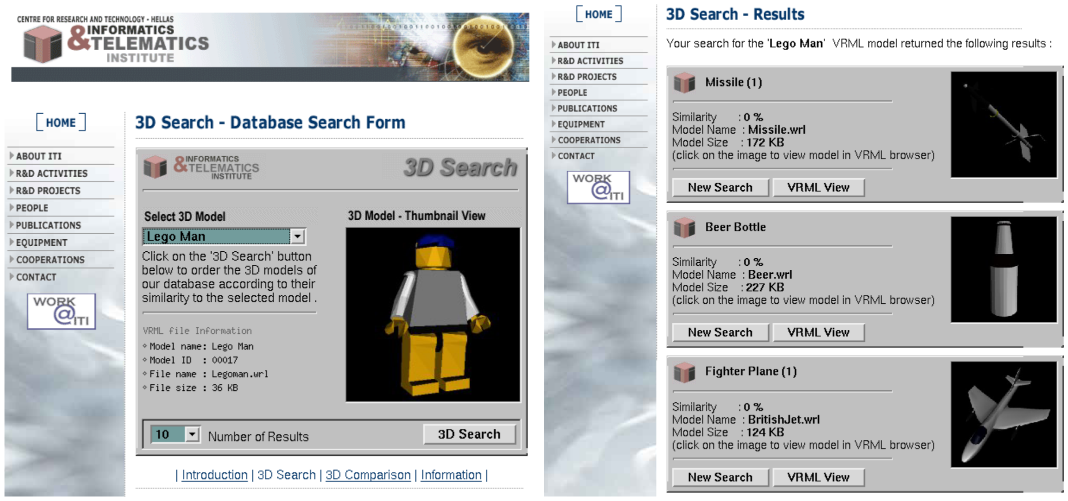


Number: 441 pod Distance: 0	vrml model signature comparison	
Number: 136 Gullglider Distance: 0.001091	vrml model signature comparison	
Number: 65 Comanche Distance: 0.004704	vrml model signature comparison	
Number: 134 GliderC Distance: 0.005413	vrml model signature comparison	

Figure 2.4: An example search and some results from Utrecht University’s 3D Shape Retrieval Engine [102, 105]

2.4.4 Overview of Current 3D Search Engines

Table 2.2 shows for each search engine site the number of models it indexes, whether these are freely available, and the types of query interfaces it supports. As can be seen from the table, our search engine provides one of the most comprehensive studies of alternative query interfaces and public access to the largest database of free models on the web.


CENTRE FOR RESEARCH AND TECHNOLOGY - HELLAS
INFORMATICS & TELEMATICS INSTITUTE
HOME
ABOUT ITI
R&D ACTIVITIES
R&D PROJECTS
PEOPLE
PUBLICATIONS
EQUIPMENT
COOPERATIONS
CONTACT
WORK @ ITI
3D Search
New Search
VRML View
3D Search
| Introduction | 3D Search | 3D Comparison | Information |

3D Search - Database Search Form

Select 3D Model: Lego Man

3D Model - Thumbnail View

Click on the '3D Search' button below to order the 3D models of our database according to their similarity to the selected model.

VRML file Information
- Model name: Lego Man
- Model ID : 00017
- File name : Legoman.wrl
- File size : 36 KB

10 Number of Results

3D Search - Results

Your search for the 'Lego Man' VRML model returned the following results :

Missile (1)
Similarity : 0 %
Model Name : Missile.wrl
Model Size : 172 KB
(click on the image to view model in VRML browser)

Beer Bottle
Similarity : 0 %
Model Name : Beer.wrl
Model Size : 227 KB
(click on the image to view model in VRML browser)

Fighter Plane (1)
Similarity : 0 %
Model Name : BritishJet.wrl
Model Size : 124 KB
(click on the image to view model in VRML browser)

Figure 2.5: Example results from ITI's 3D Search site when searching for the Lego man model on the left [47, 57]

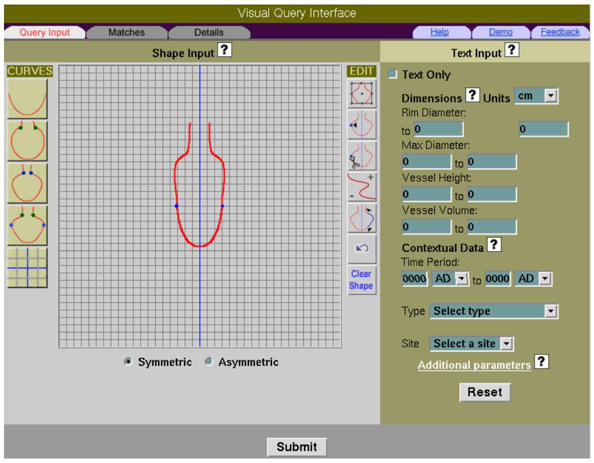


Figure 2.6: Example outline sketch query for a 3D ceramic vessel using Arizona State University's "Visual Query Interface" [80, 84]

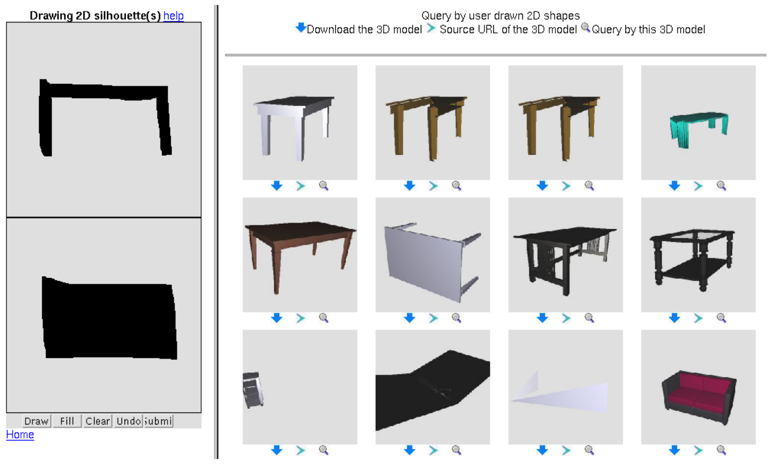

Drawing 2D silhouette(s) help
Draw Fill Clear Undo Submit
Home
Query by user drawn 2D shapes
Download the 3D model
Source URL of the 3D model
Query by this 3D model

Figure 2.7: Example results from a 2D sketch query at National Taiwan University's 3D Object Retrieval System [20]

	Alexandria	Arizona	ITI	Ogden	ShapeSifter	Taiwan	Utrecht	Princeton
Nr of models	∼ 4,500	94	37	∼ 1,500	∼ 122	∼ 10,000	919	∼ 36,000
Free models	0	0	All	0	All	All	0	∼ 31,000
Query Types:								
Text	No	No	No	No	No	Yes	No	Yes
2D Sketch	No	Yes	No	No	No	Yes	No	Yes
Result Model	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
Upload Model	No	No	No	No	No	No	No	Yes
3D Sketch	No	No	No	No	No	No	No	Yes
Other	visual & geometric features	geometric data, metadata	No	No	geometric features	No	No	No

Table 2.2: Current 3D model search engines, their model database size, the number of freely downloadable models, and supported query interfaces

Chapter 3

Text Query Interfaces

3.1 Introduction

Searching based on text keywords is by far the most common type of query interface available on the web (which is no surprise since most information is stored as text). The most popular search sites on the web are text based (for example, Google [17] and AltaVista [3]), as well as many sites that provide a search function for a single site. As a result, people are used to text-based queries and relatively skilled at creating queries that produce the desired result. Another consequence is that much research in information retrieval has been aimed at developing effective text retrieval methods.

For these reasons, we decided to implement a text query interface and accompanying matching method for our 3D model search engine. User-entered text keywords are matched to a representative text document created for each 3D model. This document can be created from many potential sources, both from the model file itself as well as the web page it is linked from. A 3D model file has a filename and typically contains part names, material names, and metadata, such as a short description of the model and the name of its author. From the web page, more text is available in the link text, the context near the link, and the web page title. We added a text source by including synonyms and hypernyms (category descriptors) of the model filename or link text to the representative text document. This was done using WordNet, a lexical database [68].

It is not obvious which text sources of a 3D model are the most discriminating. For example, we expect both the filename of a model and the link text on the web page to be useful for retrieval purposes. But, we do not know which source generally provides the most information (e.g. filenames may be abbreviated because of a filename length limit). Using a test database of 1,000 models downloaded from the web (described in detail in Section 7.2), classified into 81 categories, we investigated which text sources of a 3D model were most effective for classification. All combinations of eight available text sources of each 3D model were evaluated. We found that the text from the model file itself (e.g. part names, metadata) and the WordNet synonyms and hypernyms provided the most information for classification purposes.

Instead of implementing a text matching method ourselves, we use the methods provided by the *Bow toolkit* [66], a freeware software package for statistical text analysis. Seven different matching methods supported by this package were evaluated. From these experiments, we found that for our database the "Term Frequency/Inverse Domain Frequency" method (TF/IDF, described in Section 3.3) produced the best classification performance.

In the next section we describe the available text sources of a 3D model in more detail. In Section 3.3 we select the best performing text matching method for our application, using our test database of 1,000 models. Next, we evaluate the influence of the individual text sources on classification performance. The last section has a summary and some conclusions.

3.2 Representative Text Document

A representative text document is created for each 3D model in our database, 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 use:

From the model file:

1. **model filename:** The filename usually is the name of the model. 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, because very often filenames contain sequence numbers (for example, `chair2.wrl`) that are useless for text keyword matching

3. **modelfile contents:** This includes 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 (the `Info` node in VRML 1.0), 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. 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="747.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`

6. **web page context (text near the link):** This text, which we define 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), may also refer to the model. 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:** For example, "VRML 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 [68] (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 [26], and Scott and Matwin use synonyms and hypernyms [90] to improve classification performance. Recently, Benitez and Chang showed how WordNet can be used to disambiguate text in captions for content-based image retrieval [10]. 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. Adding these words may also increase the number of bad matches, however. For example, WordNet returns "extremely low frequency" as a synonym for "elf", and as a result the text query "frequency" returns some models of elves (as well as more relevant, audio related models)

Following common practices from text retrieval, all collected text goes through a few more processing steps. First, so called *stop words* are removed. These are common words that do not carry much discriminating information, such as "and," "or," and "my". We use the SMART system's stop list of 524 stop words [88], as well as stop words specific to our domain (e.g. "jpg," "www," "transform"). Next, the resulting text is *stemmed* (normalized by removing inflectional changes, for example "wheels" is changed to "wheel."), using the Porter stemming algorithm [77].

3.3 Text Matching Method

To be able to select the best text matching method for our purpose, we first evaluate the classification performance of several common text matching methods, using the test database of 1,000 models described in Section 7.2. The implementation we used is a program called *rainbow*. It is part of the Bow toolkit, a freely available C library for statistical text analysis [66]. The toolkit supports many different classification methods, including TF/IDF [50, 83, 89], Naive Bayes [70], K-nearest neighbors, the Kullback-Leibler divergence metric [59], etc.

When creating the text index using rainbow, each representative document is assigned its own class. In this way, when we ask the program later to classify a query, single models are returned as search results (thus mimicking the search engine behavior), after which we can use our classification to evaluate the retrieval performance. The representative text documents contained every text source described in the previous section, and were used as simulated user queries. The representative text document of each model was submitted as a query and compared to all other documents. All models were then ranked according to their text similarity score.

From these rankings we compute precision/recall values for each model. Appendix A describes this metric in detail. To briefly review, given a query model from a certain class of size c, and a number of returned results k, and a number of relevant results (i.e. models that are members of the same class) within these returned results rel, then

$$recall = rel/c$$

$$precision = rel/k$$

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 20 recall intervals 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 each class, which are then averaged. For each test, macro-averages were also computed to verify that they showed the same qualitative behavior.

Figure 3.1 shows the resulting precision/recall graphs for seven different text matching methods (two variations of Naive Bayes, three of TF/IDF, K-nearest neighbors, and Kullback-Leibler), and a method which returns random results for comparison. The other methods provided by the rainbow program were also tested but failed to run to a finish.

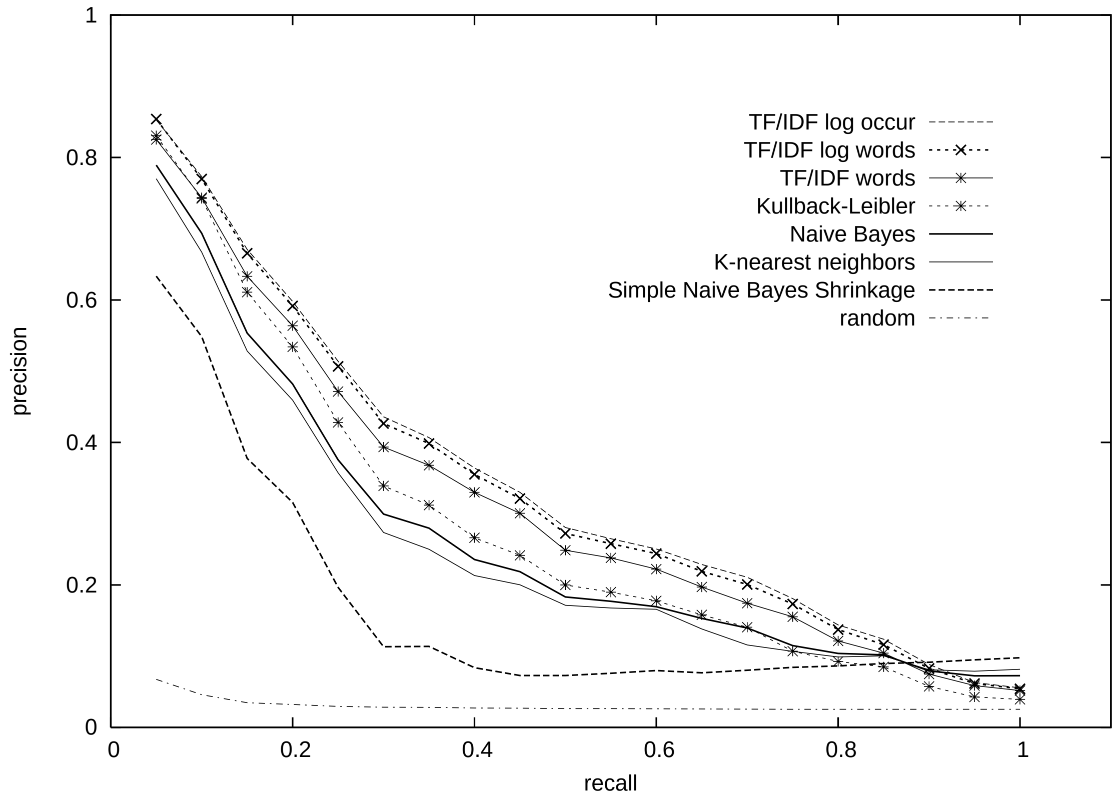


Figure 3.1: Average precision/recall for seven different text matching methods, and random retrieval

From this graph we see that “TF/IDF log occur” and “TF/IDF log words” show the best performance for our test database. We currently use the latter method for our search engine. 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 [89]. In the rainbow implementation, the first term is $\log(tf + 1)$ instead of tf. 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.

3.4 Selection of Text Sources

Now that we have identified the available text sources and the matching method to use, we would like to determine the relative importance of each source. In related work, Sable and Hatzivassiloglou investigated the effectiveness of using associated text for classifying images from online news articles as indoor or outdoor [86]. They found that limiting the associated text to just the first sentence of the image caption produced the best results. They further improved performance by using just open-class words (adjectives, nouns, verbs and adverbs) and prepositions from the source text, normalizing category vectors, and by using a probability density function to estimate the likelihood that a query belonged to a certain class. In other work, Sable *et al.* use Natural Language Processing (NLP, e.g. identifying subjects and verbs) to improve classification performance of captioned images into four classes [85]. Our problem is more difficult since our source text is less well-defined, and our number of classes is much higher. Modifications such as just using open-class words could still be helpful, however, and are the subject of future work.

To determine the most useful combinations of text sources, we ran a classification test for each combination of n out of the eight text sources, with $n \in \{1, ..., 8\}$. The total number of combinations tested was

$$\sum_{n=1}^{8} \binom{8}{n} = 255$$

Table 3.1 shows the average precision achieved for the top 10 combinations (the numbers shown in the combinations refer to the numbers in Section 3.2). From this data, we see 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 best 50 and the best 100 combinations. The results are shown as percentages in table 3.2. Clearly, and perhaps surprisingly, the identifiers found inside a 3D model file provide the most information for classification. The WordNet synonyms and hypernyms also turn out to be very useful, despite the fact that for 313 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"), words in a different language ("oiseau"), and so on). The number of synonyms and hypernyms used does not greatly affect classification performance. We tested all combinations of 1-5 synonym senses and 1-5 hypernyms of these senses, and found

rank	combination	average precision
1	1234678	0.354
2	123478	0.353
3	12345678	0.351
4	234678	0.350
5	1234578	0.349
6	123678	0.349
7	23478	0.348
8	134678	0.348
9	1235678	0.347
10	12378	0.346

Table 3.1: The average precision values achieved for the ten best combinations of text sources. The numbers in a combination refer to the numbers in Section 3.2

source	percentage in top 50	percentage in top 100
synonyms and hypernyms	100	84
modelfile	100	79
filename without digits	64	65
filename	62	58
page title	54	61
link	54	57
page context	52	52
path	50	51

Table 3.2: Percentage of all occurrences of each text source appearing in the best 50 and the best 100 combinations

no significant performance improvements. Because Wordnet returns word senses in usage frequency order, it does make sense to limit the number of senses to avoid including synonyms that are relatively obscure. Currently we store the synonyms of the first three senses, and the hypernyms of only the first sense.

Next, 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.

However, we still assign different weights to each text source based on assumptions about which text source is likely to have the most relevant information. For example, if the user enters a keyword "plane", we prefer it if the search engine returns models with the filename "plane", instead of models which have this word in the model file itself. For the same reason, the extension of a filename is added separately and assigned the highest weight, to favor the restriction to a subset of the model database containing models of the requested type.

3.5 Summary and Conclusions

Both because text query interfaces are popular and easy to use and because a lot of textual information may be available for a 3D model downloaded from the web, we added a text query interface to our 3D model search engine.

User entered text keywords are matched to representative text documents, one for each 3D model in the database. These documents are created from text sources both from the model file itself (e.g. its filename) as well as the web page it is linked from (e.g. the web page title). For the implementation, we use a program called *rainbow*, which is part of the Bow toolkit, a freely available C library for statistical text analysis [66]. We evaluated the classification performance of 7 different text matching methods supported by this toolkit, and found that the "TF/IDF log occur" and "TF/IDF log words" methods produced the best results.

Another experiment showed that the model file contents and Wordnet synonyms and hypernyms of the filename or link text provide the most information for classification purposes. We found that by weighing different text sources differently, classification performance could not be improved. However, again to improve the search engine results, text sources were assigned different weights (such that, for example, finding a user keyword in the model filename carries more weight than finding it in the model file contents).

In many cases, a search based on a few text keywords is very effective. Objects that have a well-defined, unique name (e.g. "dna," "fork") are usually easy to find. However, text-based search is not without problems. We found that 3D models on the web are usually poorly annotated, which limits retrieval performance. Of the model names we encountered, many were meaningless ("el2out" for an elephant), in a different langauge ("carra" for a train car), misspelled ("ferrair" for a ferrari), too specific ("camaro" for a car), or not specific enough ("test" for a room model). Furthermore, the user may not know the correct name of an object, or its name may be ambiguous (e.g. "plane"). Also, it may be difficult to describe certain properties of a 3D model using text (for example, "a chair with a straight back and slats in the back").

For all these reasons, we think that there are many cases where queries based on text alone are insufficient, and other attributes of a 3D model (i.e. its shape and appearance) should be queried as well. In the following chapters, we will investigate several shape-based query interfaces.

Chapter 4

3D Shape Query Interfaces

4.1 Introduction

In this chapter, we examine *3D shape* query interfaces for our 3D model search engine. Because our database contains 3D shapes, it makes sense to support other 3D shapes as queries. This means that the user provides a 3D model that has a shape similar to the desired model. This 3D query model is then compared to the 3D models in the database using a shape matching method, after which the most similar shapes are returned. When designing this type of interface, we need to answer two important questions: (1) how should query shapes be created?, and (2) what method should be used to compute 3D shape similarity?

Regarding the creation of queries, it should be simple to quickly specify a 3D shape similar to the shape of the desired 3D model (i.e. simple enough for the average user). To this end, we provide a way to submit results from an earlier search as queries, simply by clicking a link underneath a thumbnail image of a search result (the "Find Similar Shape" link). Also, the user can upload a local 3D model file, if available. However, a suitable search result or local model file may not be available. In this case a query must be created from scratch, using, for example, a 3D modeling tool. For our application such a tool should be easy to learn and use, and it should be possible to quickly specify overall shape, yet detailed enough to retrieve the desired 3D model. For this purpose we provide a simple 3D sketching tool ("Teddy," a 3D sketching Java applet created by Takeo Igarashi [45]).

The second issue is how to compare 3D shape queries to the 3D models in our database. We selected a matching method that compares feature vectors computed using spherical harmonics, developed by Kazhdan [54]. Its properties suit our application: the feature vectors (or "shape descriptors") are efficient to compute and compare, are invariant under similarity transformations (after normalization for translation and scale), and can be computed for arbitrary polygonal models. This method was shown to outperform many other shape matching methods in classification experiments on a large test database [36].

We evaluated the effectiveness of our 3D sketching query interface by classifying 1,000 3D sketches submitted to our online search engine during a period of six months. We found that about 90% of these query shapes were unidentifiable blobs or sticks. The search results of these queries generated little user interest: for example, only 3% of the queries was followed by at least one model download. This shows there is still much room for improvement for our 3D sketching interface.

In the next two sections we describe the three 3D shape query interfaces of our search engine. Section 4.4 briefly reviews the 3D matching method we use. Section 4.5.1 reproduces a comparison of 3D matching methods from Funkhouser *et al.* [36]. The test database used for these experiments is described in detail, as it is also used for experiments presented in Section 5.5 in the next chapter. An evaluation of 3D sketches submitted to our search engine appears in Section 4.5.2, followed by a summary and conclusions in Section 4.6. For usage results of these interfaces, see Chapter 8.

4.2 Submitting Existing 3D Models

Perhaps the simplest method to provide a 3D shape is to submit another, existing 3D model file. For the query to be effective, its shape should be close to that of the desired model. We support this type of query in two ways. First, the user can upload a local 3D model file. Second, the user can submit a database model by selecting a result of a previous search.

Upload Local 3D Model

The user can enter a filename, or browse for one, and upload a 3D model file in one of several 3D file formats (for example, VRML 2.0, PLY, Wavefront OBJ). The uploaded file is first converted to only geometry in the PLY format. From this file its shape signature is computed, which is then compared to the database signatures.

Select Result Model from Previous Search

This search option is implemented as a “Find Similar Shape” link below each result model’s thumbnail image on a results page. Because the model’s signature is already present in the signature database, the query interface only has to send a unique model identifier to the matching process. To improve the matching results, the best matches for each model may be precomputed in an off-line preprocessing step, using a more expensive matching method.

Elad *et al.* present an interesting extension of this query method. On a results page, the user can mark specific models as “good” or “bad,” after which weights in the matching function are adjusted appropriately and the matching is repeated [32]. We have not investigated this approach in our system.

4.3 Sketching a 3D Model

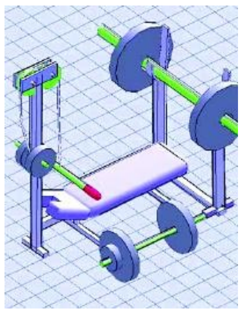
If a 3D model with a shape similar to the desired model is not available, then a 3D shape query must be created from scratch. We support this by providing a simple online modeling tool on the search engine site. This tool should be easy to learn (also for the average user) and enable the quick specification of overall shape. The design of such a tool is a difficult open problem. Instead of designing it ourselves, we use an existing simple modeling package in our investigation.

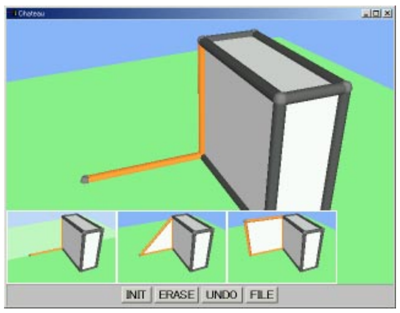
One candidate system was SKETCH, developed by Zeleznik *et al.* [113]. The user can draw lines, select and group objects, manipulate the camera, etc., using a 3-button mouse and the shift key. Primitives are created using *gestures*. For example, three non-collinear line segments meeting at a corner generate a box. Other primitives are cones, cylinders, extrusions, and so on. Most primitives are forced to be axis-aligned. A set of heuristic rules controls placement of new objects. Objects can be moved, resized, or rotated, possibly along or around a single constraint axis. An example object created using SKETCH is shown in Figure 4.1 (a). Igarashi and Hughes later described a "3D suggestive interface," in which a user gives hints to the system by highlighting related parts of a scene, and the system responds by suggesting several different modeling operations (see Figure 4.1 (b)) [45]. Both these systems are too complex for our purpose, however. Their user interfaces have many features, and are likely to be too hard to master for the average user.

A simpler system is Teddy, by Igarashi *et al.* [46]. The user draws a 2D outline, which is then extruded into a 3D triangular mesh, with its "thickness" at any point determined by the local width of the outline. Next, other parts may be attached (called "extrusions"), or cut out ("extrusions" *into* the existing shape), or pieces can be sliced off. Because of its simplicity, the type of shape that can be created is restricted: simple, blobby objects of genus 0. See Figure 4.1 (c) for an example. A similar, more flexible modeling interface, based on variational implicit surfaces instead of a polygonal mesh, was described by Karpenko *et al.* [52]. Here a model can consist of multiple parts, which may be individually manipulated, and eventually blended (Figure 4.1 (d)).

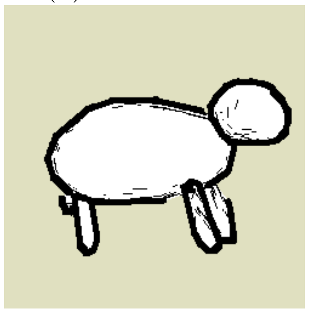
All these modeling systems struggle with the trade-off between ease of use and the type and complexity of the models that can be created. The goal is to create a minimal set of intuitive user actions, but large enough to allow the creation of interesting shapes.

For our 3D sketch query interface, we chose the "Teddy" 3D sketching program for the following reasons [46]: (1) it is the simplest (it has the smallest number of features and user interface operations), and (2) hence easiest to learn (recall that our target user is the "average user"). We simplified the interface even more by removing the "bend" function, which we thought would be hard to understand for the average user. We also removed the "style" option, which switches between the default non-photorealistic rendering style and wireframe, because the wireframe rendering may


(a)



(b)


(c)

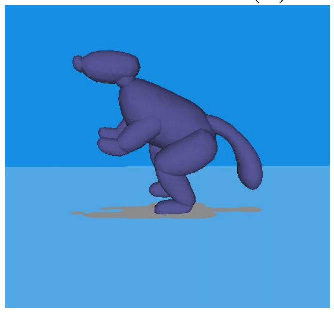

(d)

Figure 4.1: Example objects created using (a) SKETCH [113], (b) a "suggestive interface" [45], (c) Teddy [46], (d) variational implicit surfaces [52]

be confusing for the user. Finally, (3) a practical advantage of Teddy is that it has been implemented in Java and the source code is freely available.

4.4 Matching Method

After the query model has been submitted, we have to match it to the 3D models in our database.

Since these models have been downloaded from a variety of sources on the web, we cannot assume that they are closed 2-manifold meshes. Most of the models are unorganized sets of polygons ("polygon soups"), possibly with missing, wrongly-oriented, intersecting, disjoint, and/or overlapping polygons. We also cannot make any assumptions about their scale and orientation. Consequently, the matching method should be (1) robust under model degeneracies, and (2) invariant under similarity transformations.

Rather than processing the models themselves for every match, they usually are converted to a more compact representation (a *shape signature* or *feature vector*), which can be matched more efficiently. Because ours is a real-time application, these signatures should (3) be fast to compute (because a single one must be computed for a 3D sketch or an uploaded model file), and (4) be fast to match. Also, the (5) signature should be small enough such that it is practical to store tens of thousands of them. Finally, (6) the matching method should be effective, in other words, the matching results should correspond to our human notion of shape similarity (this also means that the shape signature should represent significant shape features).

For our search engine we use a 3D matching method developed by Michael Kazhdan [54], which satisfies all the above requirements. The method computes a 2D shape signature from a 3D model. A shape comparison is then done simply by computing the Euclidian distance between two shape signatures. The method is (1) robust under model degeneracies, and (2) invariant under similarity transformations (with normalization for scale and translation). (3) A signature can be computed quickly (2 seconds on average on a 2.2 GHz Xeon CPU), and (4) matched efficiently (matching a single descriptor to a database of about 36,000 descriptors takes less than 0.4 seconds). Also, a signature is (5) compact (about 2 KB). Finally, (6) the method outperforms other current 3D shape matching methods in precision/recall tests on a large test database (1,890 models classified into 85 classes, also see the next section)(see Appendix A for an explanation of the precision/recall performance metric). For full details, see [36] and [54].

4.5 Results and Discussion

This section presents matching performance results of the 3D shape matching method we use, and some results on the usage of the 3D sketching interface of our search engine.

4.5.1 3D Shape Matching

In this section we compare the classification performance of our 3D shape matching method with that of several other methods. These results were presented in more detail in Funkhouser *et al.* [36].

Test Database

For this experiment, we used a test database with 1,890 models of "household" and "miscellaneous" objects provided by Viewpoint [107]. Objects were clustered into 84 classes based on functional similarities, largely following the groupings provided by Viewpoint. Examples from ten representative classes are shown in Figure 4.2.

Out of the 1,890 models, 653 were put into a "miscellaneous" class because they did not fit into any meaningful class. An additional 144 models in 51 classes were added to the "miscellaneous" class because these classes were smaller than our size limit of 5 models. This left 1,093 models in 84 classes. Table 4.1 lists all classes and their sizes, and Figure 4.3 shows a histogram of the class sizes (except the "diningchair" class).

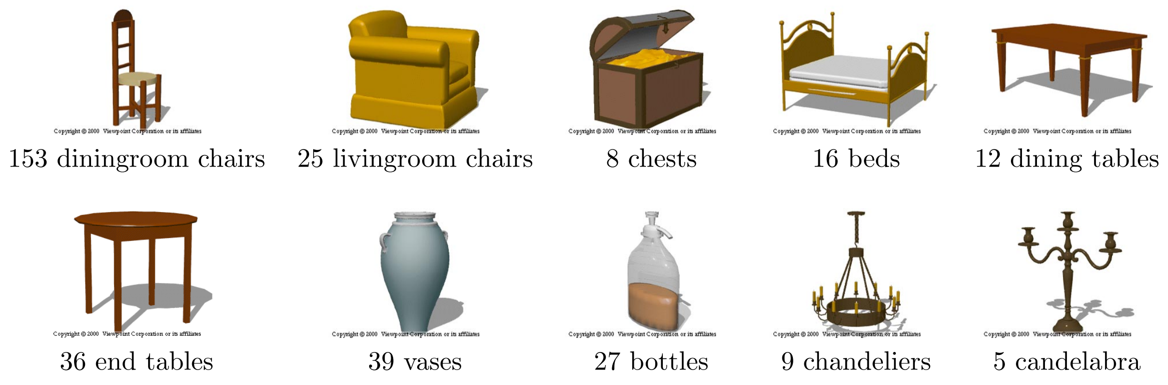


Figure 4.2: Samples from ten representative classes from the Viewpoint "household" and "miscellaneous" database (images courtesy of Viewpoint)

name	size	name	size	name	size	name	size
armoire	9	chest	8	fork	8	plate	8
axe	12	clock	14	fruit	16	plateoffood	17
bakingpan	7	coffeemug	10	goblet	19	pot	7
barchair	6	coffeetable	19	grandfatherclock	5	pumpkin	8
barrel	7	coffin	6	hat	5	refrigerator	7
bartap	6	conferencetable	19	heart	6	rifle	10
bed	16	curtains	7	humanbody	5	shelves	15
beermug	6	deskchair	10	iron	5	shield	9
bench	11	desk	7	key	15	sink	5
bookcase	19	diningchair	153	kitchentable	18	sofa	29
bottle	27	diningtable	12	knife	24	spoon	11
bowl	6	directorchair	5	livingroomchair	25	star	8
boxcontainer	6	dishcabinet	5	loungechair	7	stool	21
buffet	12	displaycase	9	mace	6	suitcase	7
cabinet	41	doorhandle	6	mirror	11	sword	24
candelabra	5	doorknob	5	mopandbroom	7	toilet	5
candleholder	6	dresser	21	ottoman	12	tombstone	7
candle	6	drinkingfountain	6	oven	11	trashcan	20
cannon	5	endtable	36	pictureframe	8	tub	5
cart	6	faucet	5	pistol	5	utensils	7
chandelier	9	fireplace	7	pitcher	8	vase	39

Table 4.1: The 84 classes in our test database of 1,093 models, donated by Viewpoint [107], and their sizes

We chose this Viewpoint database because it provides a representative repository of models with uniform quality and because it is difficult for shape-based classification. In particular, several distinct classes contain objects with very similar shapes. For example, there are five separate classes of chairs (153 dining room chairs, 10 desk chairs, 5 director's chairs, 25 living room chairs, and 6 lounge chairs, respectively). Meanwhile, there are objects spanning a wide variety of shapes (e.g. 8 forks, 5 cannons, 6 hearts, 17 plates of food, etc.). Thus, the database stresses the discrimination power of our shape matching algorithm while testing it under a variety of conditions.

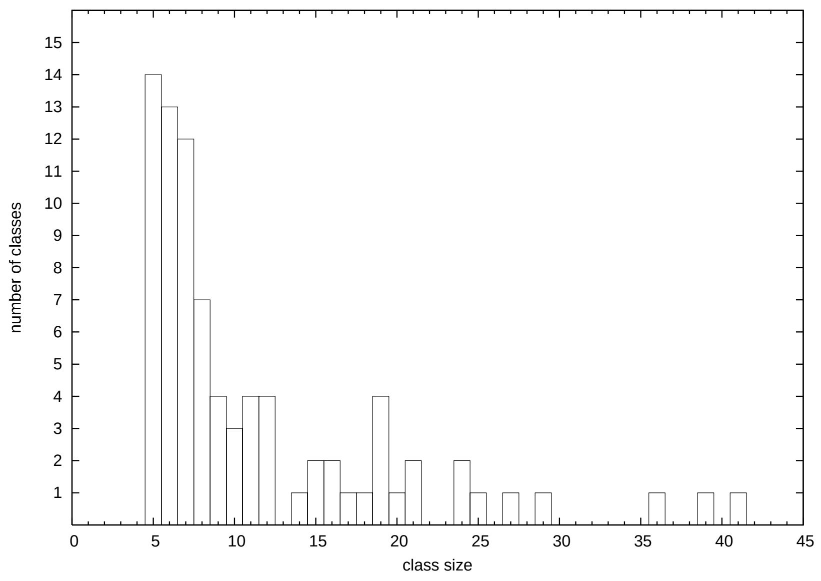


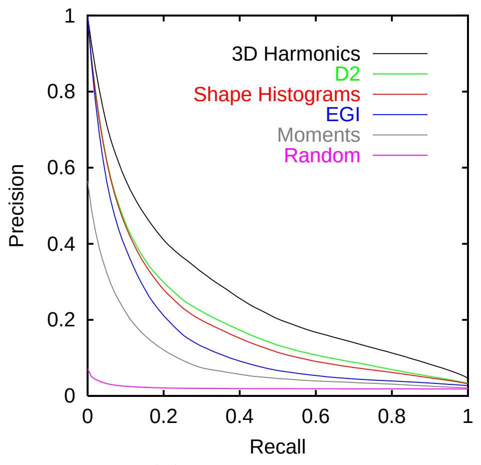
Figure 4.3: A histogram of the class sizes of the 84 classes in the Viewpoint database. Note that the "diningchair" class (containing 153 models) is not shown

Each model (except those in the "miscellaneous" class) was used as a query, comparing it to each model in the entire database (i.e. all 1,890 models). The resulting rank of each model was compared to the manual classification.

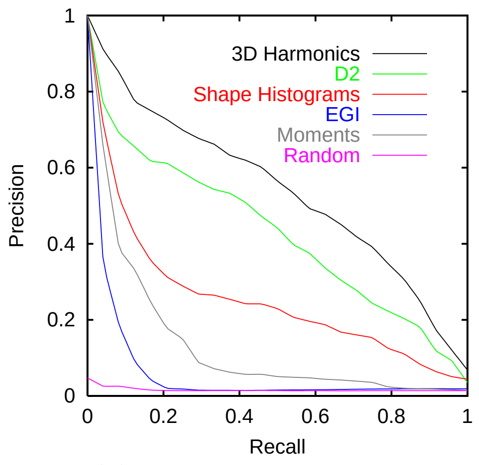
While the purpose of the experiment is mainly to evaluate our matching method, the results are indicative of how well our search engine works when a user provides his own 3D model and asks our system to find similar ones, or when a user clicks on the "Find Similar Shape" link under the image of an object returned by a previous query.

For comparison purposes, five other shape matching algorithms were implemented by Michael Kazhdan: (1) random, (2) moments [32], (3) Extended Gaussian Images (EGI) [41], (4) shape histograms [5], and (5) D2 shape distributions (D2) [72].

Figure 4.4 (a) shows the average precision/recall obtained with our matching algorithm as compared to the other methods (see Appendix A for an explanation of the precision/recall performance metric). Results for a single example class (containing 153 models) are in Figure 4.4 (b). Note that for every recall value, our method (spherical harmonics, black curve) gives better precision than the competing methods. On average, the precision values are 46% higher than D2, 60% higher than Shape Histograms, 126% higher than EGIs, and 245% higher than moments. For more details on each matching method and these results see [36].



(a) All classes



(b) Living room chairs

Figure 4.4: Precision/recall plots of our 3D shape matching method versus other methods

4.5.2 3D Sketch Quality

When analyzing logs of our online search engine, we find that the 3D sketching interface is used very little, less than 1.5% of all queries. Furthermore, most of the queries that are submitted are of very low quality. To get a sense of the quality of the queries, we classified 1,000 3D sketches submitted in the first half of 2002 into eight categories. Figure 4.5 shows a representative example of each category, and Table 4.2 shows their relative sizes. From these numbers it is apparent that unidentifiable blob-shaped models form by far the majority of the queries. As a result, there was minimal user interest in the search results. For example, only 3% of the queries was followed by a model download (compared to 13-15% for some other query types).

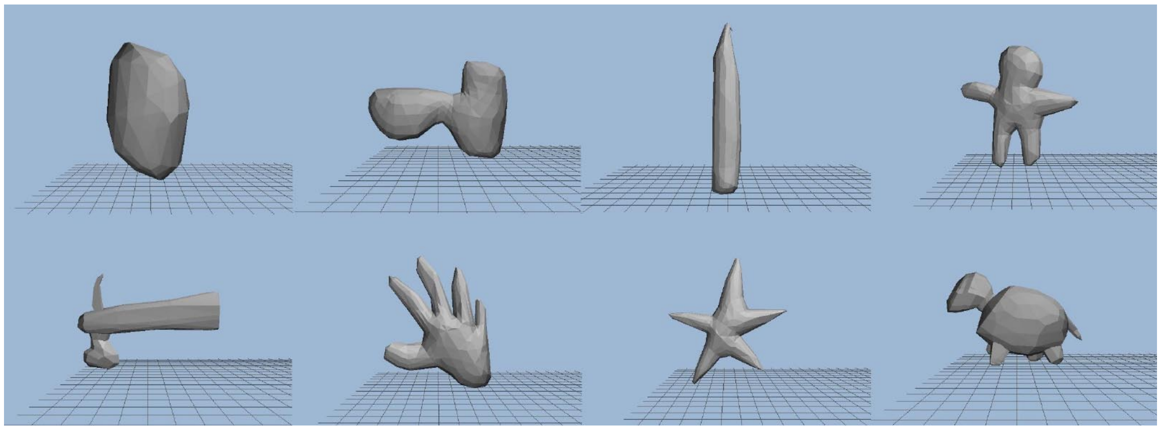


Figure 4.5: A representative example of each of eight categories for 1,000 submitted 3D sketches

Category	Percentage
convex blob	52
blob with few concavities	30
stick-like	8
human	3
miscellaneous objects	3
hand	1
star-shaped	1
quadrupeds	1

Table 4.2: Relative sizes of the 3D sketch categories

4.6 Summary and Conclusions

In this chapter, we described 3D shape query interfaces supported in our 3D model search engine. The simplest ways for a user to submit a 3D shape are (1) uploading an existing 3D model file, and (2) selecting a result model from a previous search.

Because the average user may not have local 3D model files available, and because it may be difficult to find a 3D model with a shape close to the desired model using other search methods, we also provide a query interface with which the user can create a 3D shape query from scratch. In our initial approach, we used an existing simple 3D modeling tool called "Teddy" [46].

However, we found that Teddy is very limited in the kinds of shapes that can be created with it: coarse, blobby shapes of genus 0. An evaluation of 1,000 3D sketches submitted during a period of six months showed that over 80% of all queries were exactly that: coarse blobs, not resembling any useful object. As a consequence, the user interest in the search results was very low. The percentage of 3D sketch queries resulting in at least one model download is about 5 times smaller than for the most popular query interfaces.

Designing a simple 3D modeling tool with a minimal set of intuitive operations enabling the creation of interesting models is a difficult problem. Previous work in this area mostly targets graphics professionals, or at least people used to working with 3D graphics. Designing effective 3D modeling tools for the average user is an important area of future work.

We think that even if such tools were available, there still are many users who do not have sufficient 3D skills to be able to use them effectively. Therefore we should also put our efforts into finding a *simpler* shape query interface that is still effective. One approach is to drop one dimension and create a 2D query interface. Interfaces of this type are the subject of the next two chapters.

Chapter 5

2D Shape Query Interfaces: 2D Free-form Sketch

5.1 Introduction

This chapter investigates a 2D free-form sketching interface for our search engine.

In the previous chapter, we concluded that it is difficult for the average user to create a 3D shape query from scratch. Perhaps we can make the query interface simpler by dropping one dimension and asking the user to provide a 2D shape (e.g. a 2D sketch). From the user's perspective a 2D interface is simpler because people (1) have more experience with drawing in 2D, (2) are used to seeing 2D projections of 3D objects, and (3) are able to recognize 3D objects from a single 2D image. Also, both the input device (mouse or tablet) and output device (screen) are 2-dimensional. So, 2D input and output are most natural.

For these reasons, we decided to develop a simple 2D shape query interface and accompanying matching method. Our initial goal was to create an interface that is easy to use and allows the user to quickly specify an overall shape.

There are many design options for a 2D drawing interface, such as the choice of which primitives to use (pixels, line segments, disks, ...) and which extra features to include (cut, copy & paste, alignment tools, ...). Because we want the interface to be usable for any user who knows how to use a mouse and perhaps has used programs like Paintbrush or Photoshop before, it should be simple and use only well-known interaction techniques. We chose a sketching interface similar to the pen-drawing interface as used in Paintbrush-style programs: pixels can be drawn by dragging the mouse (see Figure 5.1 for an example single stroke drawn with our interface). The "Clear" button clears the image, the "Undo" button removes the last stroke or undoes a "Clear". The right mouse button clears a small circular area of pixels at the mouse cursor. The user can submit up to three sketches in three separate drawing areas of a desired model as seen from different directions. The system then returns 3D models whose 2D projections match those sketches.

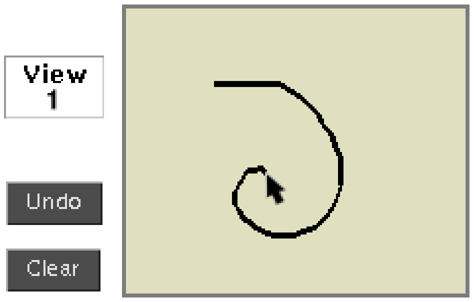


Figure 5.1: An example single stroke drawn using our 2D sketching interface

This approach brings up several new questions: What kind of sketches will people create, and which viewpoints do they select? How should we match a user sketch to a precomputed 2D projection? How can we use sketches from multiple viewpoints? Unfortunately, the vast literature on how trained artist draw [82], how people use characteristic views [73], and how computers recognize photographic images [39] is not directly applicable in our case. Rather, we are interested in how untrained artists make quick sketches and how a computer can match them to 3D objects.

To investigate what people draw when asked to draw a certain object, we ran a pilot study with 32 students who were asked to quickly sketch various objects. Based on the results of this study we decided to store the *exterior outlines* of 2D projections of the 3D models ("2D views") in our database as the images to match user sketches to.

To match a single user sketch to a single 2D view we use a 2D image matching method that matches 2D feature vectors, computed in a similar way as the feature vectors of our 3D shape matching method. The matching method has some tolerance for small user inaccuracies.

When multiple user sketches have to be matched to multiple stored 2D views of a 3D model, we compare all permutations of n sketches to all combinations of n out of m 2D views, with the restriction that no two user sketches can match a single view. The matching score of multiple user sketches is then the smallest sum of the individual scores, for all combinations. For each 3D model, we precomputed thirteen 2D views, grouped into three side views (looking from the center of a side of the bounding cube to its centroid), four corner views (looking from the corner), and six edge views (looking from the center of an edge).

Using a test database of 1,093 classified models (described in detail in Section 4.5.1), we ran several classification experiments, varying the number of stored views and the number of query views (the stored outlines were used as queries). From these experiments we concluded that (1) using more query views makes a query more effective, (2) using side and corner views for both queries and as the model representation has the best performance, (3) storing side and corner views while submitting only side or corner views does not significantly impact performance, and

(4) this matching method is outperformed by our 3D shape matching method.

To investigate the effectiveness of queries in which text and 2D sketches are combined, we ran a user study with 18 students, who were asked to write down descriptive keywords and draw up to three 2D sketches for five target objects. From this study, we found that text keywords are effective for picking out small, well-described classes, and that sketches can be useful for finding specific objects in a larger class (for example, a chair with a straight back from a large number of chairs).

We evaluated the effectiveness of our online 2D sketch interface by classifying 589 sketches submitted over a period of 3.5 months (selected from the total set of 11,000 submitted sketches by restricting it to sketches from hosts that submitted at least 20 sketches). The sketches were classified according to the following criteria: (1) identifiable or not, (2) interior lines or not, (3) accurate or not. Note that (1) and (3) are subjective criteria. We found that many sketches were inaccurately drawn, and as a consequence, there was little user interest in the search results: this interface has the lowest percentage of model downloads of all available query interfaces of our search engine.

In the next section, we describe the pilot study in which subjects made sketches of various objects. Section 5.3 briefly reviews the 2D image matching method we use, followed by an explanation of how multiple sketches are matched to multiple 2D views in Section 5.4. Section 5.5 has results of classification experiments, the text & 2D sketch user study, and the evaluation of the sketches submitted online. Section 5.6 presents a summary and conclusion.

5.2 What Do People Draw?

First, we have to decide what kind of images the 2D projections should be (e.g. filled, having silhouette edges only, etc.). For this, we need more information about what the user sketches may be like. To investigate what people draw when asked to draw a certain object in a short time, we ran a pilot study with 32 students from an introductory computer science class. They were asked to "draw the shape of an <object>" (using pen and paper), for eight different objects, with a time limit of 2 minutes for each object. Representative results appear in Figure 5.2.

What we found is that people tend to sketch objects with fragmented boundary contours and few other lines, they are not very geometrically accurate, and they use a remarkably consistent set of view directions. Based on the results of this study, we decided to include only the exterior outlines in 2D projections of objects in the database.

Terminology: ***2D View***
In the remainder of this chapter we will refer to such an image (i.e. the exterior outline of a 2D projection of a 3D model in our database) as *2D view*, or simply *view*.

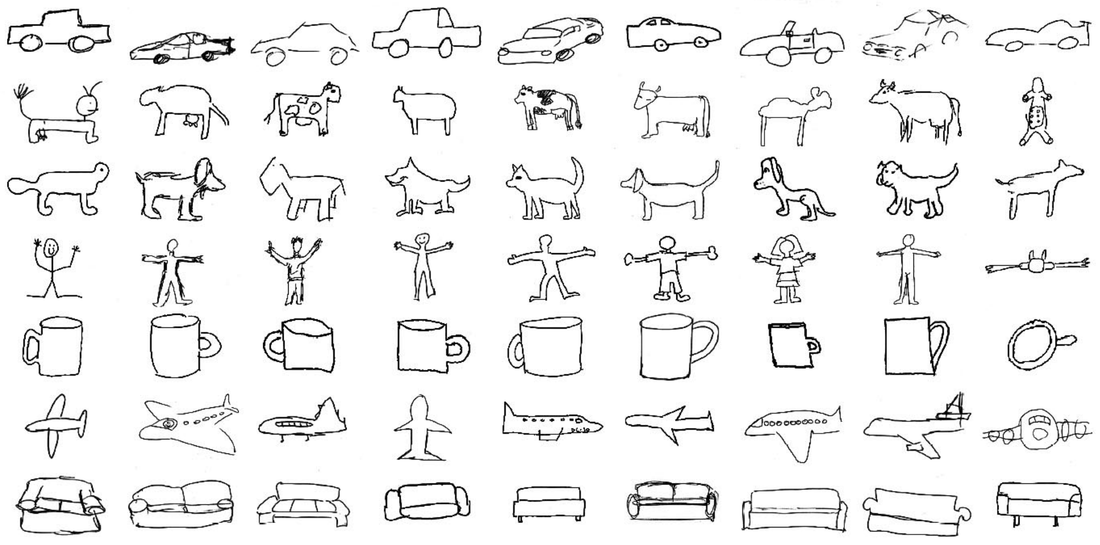


Figure 5.2: Pen-drawn sketches by people asked to "draw the shape of a" camaro car, cow, dog, human with outstretched arms, mug, DC10 airplane, and sofa

5.3 Matching a User Sketch to a 2D View

Once the user has sketched an image, it must be matched to 2D views of objects in the database using a suitable matching method (see Figure 5.3 for an illustration of this matching problem).

Unfortunately, because the user sketches are free-form, methods that match (possibly closed) planar curves are not directly applicable. Methods that define a function on a curve, parameterized by arc length (e.g. the turning function for polygonal shapes [6], the arch height function [62], Fourier descriptors based on such functions [63], or the Curvature Scale Space image [71]), cannot easily be used because the user input is usually fragmented into many short strokes rather than one continuous contour.

For these reasons, we chose to use an image matching method that has some tolerance for user inaccuracies. It was developed by Michael Kazhdan, and is described in detail in Funkhouser *et al.* [36]. We now briefly describe this method. The method computes a similarity value for a pair of images by comparing 2D shape descriptors computed for both images. Figure 5.4 demonstrates the steps taken to compute the descriptor: (1) Compute the Euclidian Distance Transform (EDT) of the boundary contour. (2) Obtain a collection of circular functions by restricting to different radii. (3) Expand each circular function as a sum of trigonometric functions. (4) Using the fact that rotations do not change the amplitude within a frequency, define the signature of each circular function as a list of the amplitudes of its constituent trigono-

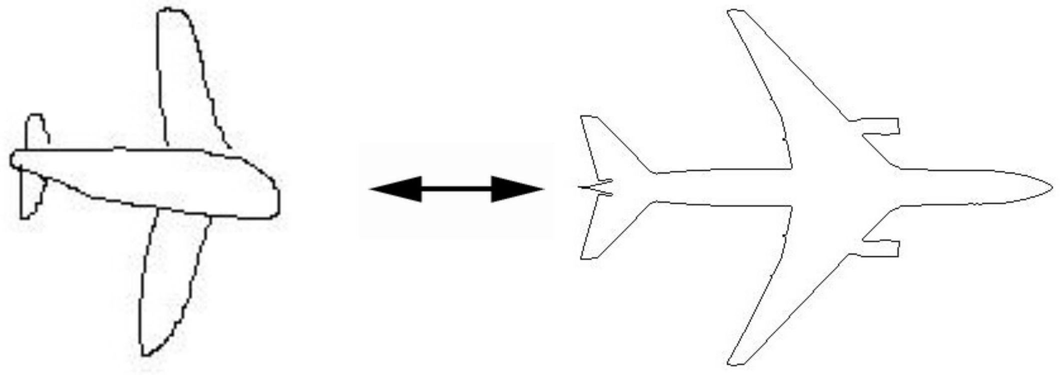


Figure 5.3: A 2D user-drawn sketch has to be matched to an outline of a 2D projection of a 3D model

metrics. (5) Finally, combine these different signatures to obtain a 2D descriptor for the boundary contour. This method is inspired by Zahn and Roskies' work on Fourier Descriptors [112], which provides a rotation invariant signature for boundary curves, obtained by computing the Fourier series and storing only the amplitude of each frequency component. Two descriptors are compared by computing the Euclidian distance between them.

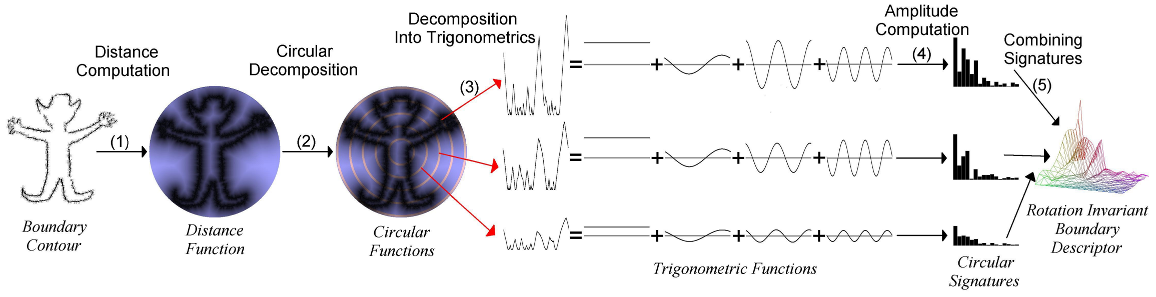


Figure 5.4: Computing Kazhdan's 2D shape descriptor [36]

Several other properties of this method are useful for our application. The descriptors are concise and efficient to compute, invariant to similarity transformations (with normalization for scale and translation) and reflections, and robust under small user inaccuracies. The advantage of using the EDT, instead of the binary image itself, is that increasing user inaccuracy causes a gradual decrease in the similarity score (i.e. a gradual increase in the distance between two descriptors).

5.4 Matching Multiple Sketches to Multiple 2D Views

Because we cannot predict from which viewing direction the user will draw a sketch, we have to store several views of each 3D model (which views these should be is investigated in Section 5.5.1). The best matching view of a 3D model is then found by simply comparing the user sketch to each stored view. Additionally, the user can make a query more specific by submitting multiple sketches, representing the desired model as seen from multiple different directions (see Figure 5.5 for an example query with two sketches). In this case, our matching problem becomes one where multiple user sketches have to be matched to multiple stored views.

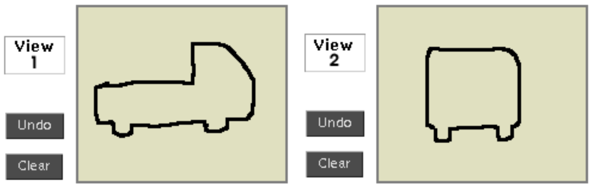


Figure 5.5: An example 2D sketch query containing more than one sketch: a side and front view of a pickup truck

Given our method to compare two images (i.e. by comparing their shape descriptors), we can use it to match n user sketches to m 2D views. This matching is done by comparing all permutations of n sketches to all combinations of n out of m views, with the restriction that no two user sketches can match a single view. The matching score of multiple user sketches is then the smallest sum of the individual scores, for all combinations. Figure 5.6 shows an example, matching two user sketches to seven 2D views.

Note that by matching this way we do not enforce consistency of direction: it is not a requirement that all pairwise angles between query views are identical to all pairwise angles between corresponding database views (for example, three mutually orthogonal query side views do not have to match three mutually orthogonal database views). We chose this approach because in our application we cannot make any assumptions about the view directions chosen by the user.

However, when matching 3D models with other 3D models using 2D projections, we *can* ensure direction consistency. For example, in recent work by Chen *et al.*, a 2D view based matching method for matching 3D models is presented [21]. The method uses a representation called the *LightField Descriptor* (LFD), a set of 10 projections of a 3D model 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. Using a test database of 1,833 models classified into 47 classes and one "miscellaneous" class of 1,284 models, they show that their method has better classification performance than other current 3D shape matching methods (including the 3D shape matching method we use [54]).

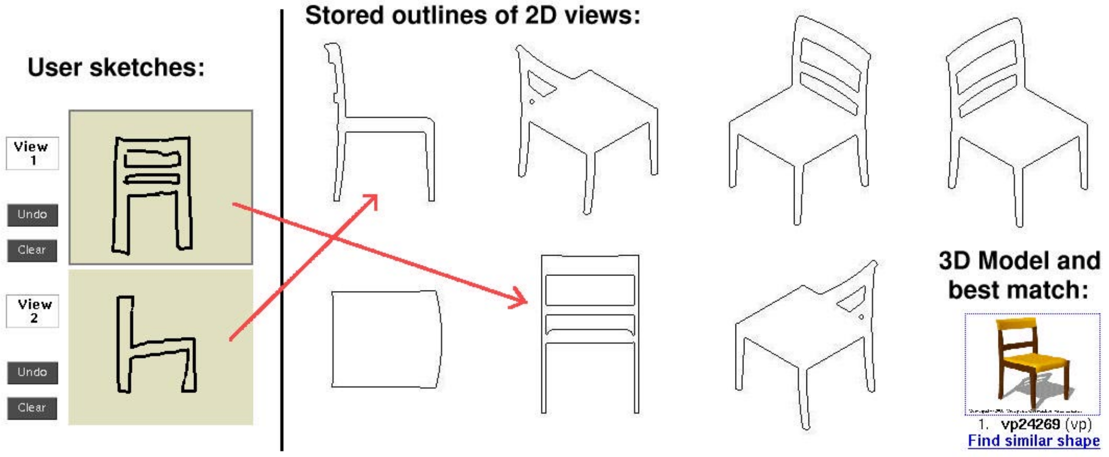


Figure 5.6: Matching $n = 2$ user sketches to $m = 7$ 2D views. Both permutations of the user sketches are compared to all $\binom{7}{2}$ combinations of the 2D views. The two best matches and the corresponding 3D model are shown

5.5 Results and Discussion

In this section, we describe a series of classification experiments designed to evaluate our 2D freeform sketching query interface for 3D model retrieval. In addition to testing whether it is effective, we also would like to investigate how many 2D views should be used for each 3D model and from which directions they should be taken. We also describe a user study in which the goal is to evaluate whether 2D sketch queries can augment the performance of text-based retrieval of 3D models. Finally, in Section 5.5.3, we evaluate the quality of the 2D sketches submitted by actual users of our search engine.

5.5.1 Outline View Selection

To investigate how the choice of the number and type of the stored 2D views of a 3D model affects the matching performance, we ran a series of classification experiments with different sets of 2D views per model.

Test Database

The classification experiments were run using several different test databases containing 2D shape descriptors of multiple 2D projections of 3D models from a 3D model database. The 3D model database was donated by Viewpoint [107] and contains 1,093 models. It is described in detail in Section 4.5.1. Note that in these experiments, we only use the 84 classes with 1,093 models, and not the "miscellaneous" class mentioned in Section 4.5.1.

View Sets

For practical reasons we have to pick a limited number of 2D views for each model. In related work, Cyr and Kimia construct prototypical views of a 3D model from views clustered by their shape similarity measured by a shock graph matching method [25]. These prototypical views could be used as the set of 2D images to represent a 3D model, and thus be matched to the user sketches. However, the method produces many views for each model (on average about 7 for the models in a test database of 18 models), Also, these views are sampled from a circle around the object (not the full viewing sphere), which is inadequate if the models are not consistently oriented. Finally, it is not clear if the set of 2D views that is most discriminating according to their shape similarity metric will contain views that typically are sketched by a human.

For our application, we take the following approach. Since most 3D models are oriented such that either the XOZ, YOZ, or XOY plane is the "ground plane" (i.e. the plane on which the model "rests"), it is reasonable to assume that the front, side, and top views of a model usually produce these plan views (but not necessarily in that order). We will call the front, side, and top views the *side views*, and define them as the orthographic projection of the model as seen from the center of three of the six sides of a cube (with no two sides opposite each other), looking in the direction of its center. To these three views we add four *corner views*, whose viewing directions are from four of the cube's eight corners to its center, and six *edge views*, looking from the center of six edges to the cube's center. This makes the total number of directions to choose from $3 + 4 + 6 = 13$. Recall that our matching method (Section 5.3) is invariant to reflection, which is why we only store views from half the sides, corners and edges. Figure 5.7 shows two directions of each view type, and Figure 5.8 shows a 3D model and a few example projections.

We constructed seven "view sets", containing 2D shape descriptors of different subsets of 13 possible views for each model. Table 5.1 shows for each view set which views were chosen.

Database name	3 Side	4 Corner	6 Edge	Total
S	×			3
C		×		4
E			×	6
SC	×	×		7
SE	×		×	9
CE		×	×	10
SCE	×	×	×	13

Table 5.1: Seven "view sets" produced from different subsets of the total set of 13 views

Classification Experiments

The goal of these experiments was to investigate how many 2D views should be used for each 3D model and from which directions they should be taken.

We ran $7 \times 7 = 49$ tests, trying all combinations of using query views and database views from the set {S, C, E, SC, SE, CE, SCE}.

In each test we submitted $n \in \{1, 2, 3\}$ queries (we found that for values of $n > 3$ the classification performance increased only marginally, or in some cases even decreased. Additionally, it may become too cumbersome for the user to draw four or more sketches. Thus, we do not consider tests with $n \geq 4$ further in this thesis). If the view set has m views stored for each 3D model (i.e. m shape descriptors for the outline images from those views), then we can choose from $\binom{m}{n}$ different views. For each 3D model, we try each of these view combinations and pick the best one. The best query combination is the one with the smallest sum of dissimilarity scores

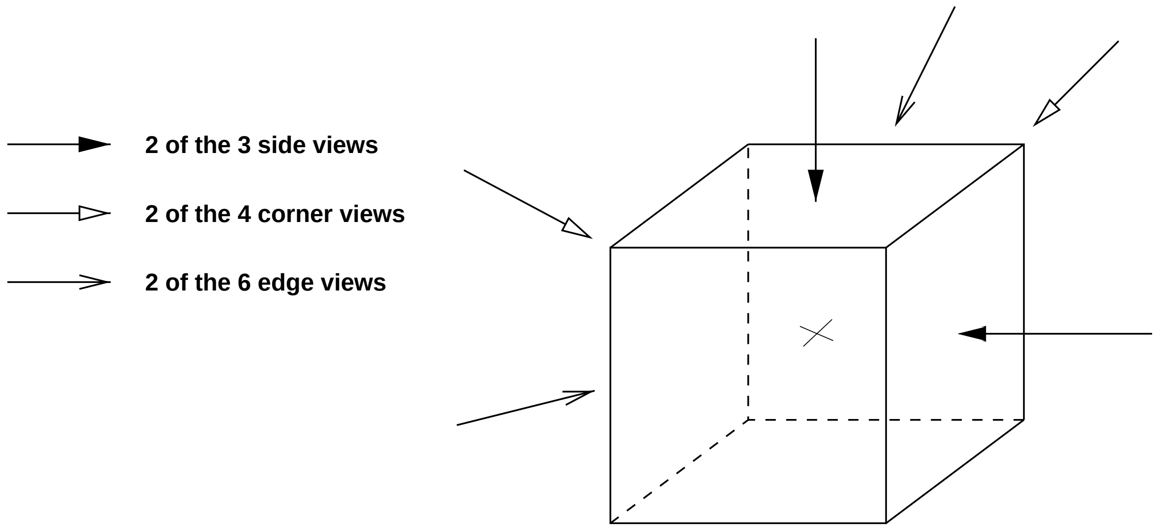


Figure 5.7: Two examples of each direction type (side, corner, and edge views)

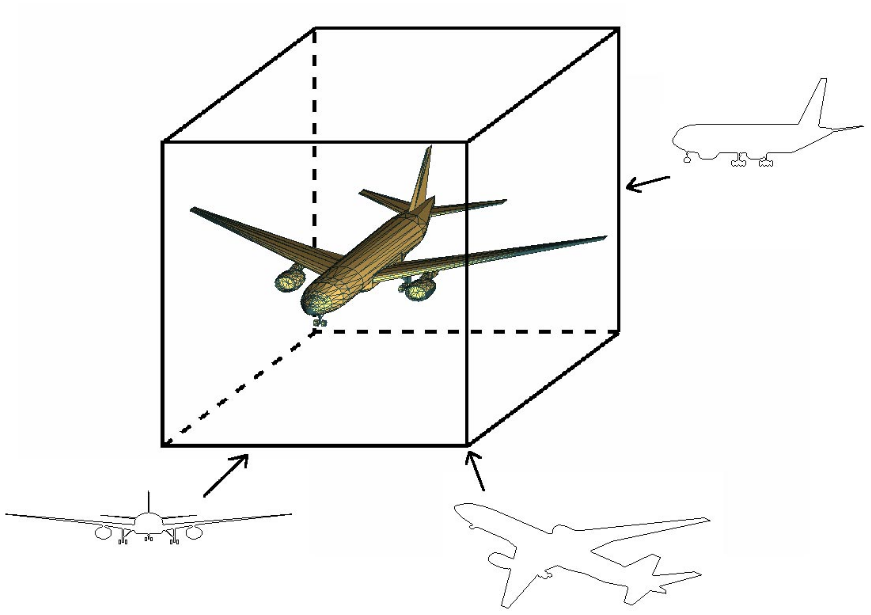


Figure 5.8: Outlines of 2D projections (*2D views*) are created for each 3D model

between the query images and the database images, over all permutations of the database images. In other words, given the available m 2D views of a 3D model, and the number of query images n, we find the n out of those m images with the smallest possible sum of dissimilarity scores. It may not be reasonable to expect the user to consistently pick the most discriminating views for every type of object. We do think, however, that humans are skilled at picking discriminating views (when asked for the most discriminating view of a table, for example, few people would pick the top view), and thus the results are loosely indicative of human performance.

The results of the tests are given as precision/recall graphs, with average precision values at recall values 0.2, 0.4, 0.6, 0.8 and 1.0 (see Appendix A for an explanation of the precision/recall performance metric). The average precision values are micro-averages, i.e. averaged over all models. For all tests, the macro-averages (average over all classes of averages for each class) were computed as well, and we found that they showed the same behavior. Note that in the following graphs, the range along the vertical axis is $[0, 0.65]$.

How Many Query Views

First we examine the effect of the number of query views. Figure 5.9 shows the average precision/recall when using 1, 2, and 3 query views chosen from the SCE view set, averaged over using each of the 7 possible view sets as the database to query into. Note that using more query views improves performance, with the biggest improvement when going from one to two views. All 49 (query view set, database view set) combinations showed the same behavior, except for some cases when the query view type was not contained in the database view set (for example, for the combination (S, CE)).

Which Query Views

Next, we look at the effect of the number and type of the query views that are used. Figure 5.10 shows the average precision/recall when using 3 query views for each of the 7 query view sets, querying into the corresponding view set (i.e., S into S, SC into SC, and so on). From these results we see that (1) of the query view sets with a single type, the corner views are most discriminating, followed by the edge views and side views, (2) adding more views does not necessarily mean that performance goes up: for example, the (E, E) combination is better than the (SE, SE) one, and (3) the (SC, SC) combination has the best performance. Also note that the combination of the largest set (SCE, SCE) is significantly worse than (SC, SC). We think that in this case there are too many matching possibilities between three images of the query view set to three images of the database view set.

Note that in our matching method we do not enforce consistency of direction, e.g. three mutually orthogonal side views do not have to match three mutually orthogonal views. We intend to add this condition in a future version of this matching method.

Number of Stored Views

From the previous graph, we could conclude to use side and corner views to represent each 3D model. However, we still should verify if this view set is not too large, in other words if performance will suffer if we query using side or corner views only. Figure 5.11 compares the average precision/recall for three queries from a few (query set, database set) combination pairs: in this graph, compare (S, S) to (S, SC), and (C, C) to (C, SC). Both comparisons show that there is no significant difference when using the larger SC view set. This means we can use this view set without negatively impacting performance should the user submit, for example, just side views of an object.

Comparison to 3D Matching

Finally, we compare the 2D view-based matching method to our 3D matching method described in the previous chapter. Figure 5.12 shows average precision/recall for a few (query set, database set) combinations and number of queries: a single query from (S, S), two queries from (C, C), and three queries from (SC, SC), as well as for our 3D matching method. We see that the 3D matching method is better than our currently best 2D view set based matching method (on average the precision is 12% higher).

As was noted in Section 5.4, when using 2D views for matching 3D models, we should enforce direction consistency to improve matching performance (i.e. all pairwise angles between query views should be identical to all pairwise angles between corresponding database views). Our current 2D view based matching method does

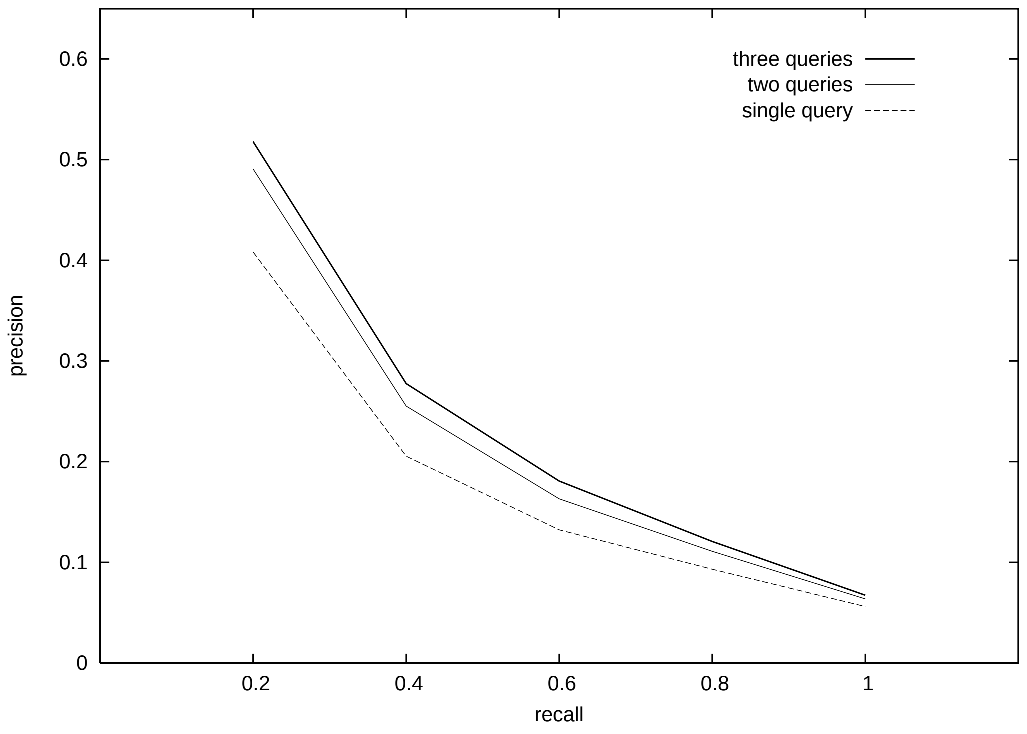


Figure 5.9: Average precision/recall using 1, 2, and 3 query views chosen from the SCE view set, averaged over the 7 possible database view sets

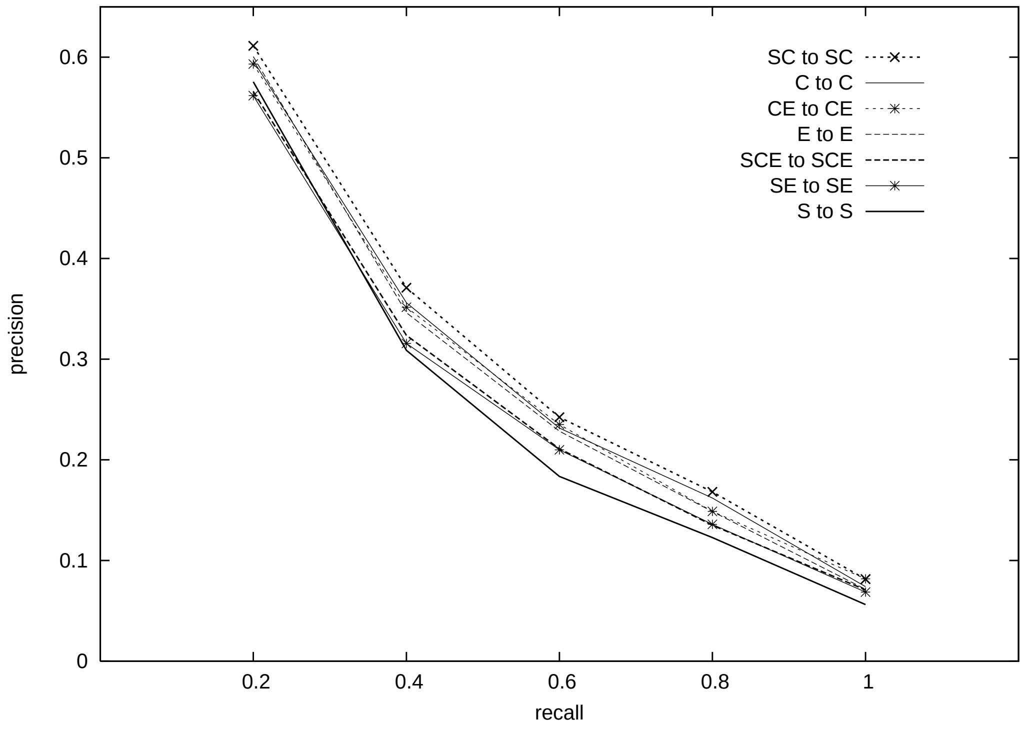


Figure 5.10: Three queries from each view set, into the same set

not do this, because in our application we cannot make any assumptions about the view directions chosen by the user.

For our online user interface we chose to store the seven side and corner views (SC). Edge views were not used because they hurt matching performance. Even if in a future version of our matching method (e.g. one in which direction consistency is enforced) the edge views are helpful, we still think we should not include them in the representation of models of our online database: we believe that this type of view is more difficult to imagine for a human, and therefore unlikely to be submitted as a query. To illustrate this, Figure 5.13 shows a corner view, side view, and edge view of four example models (a car, chair, dog, and man). Thus, including edge views will in this case only increase the number of false positive matches.

5.5.2 Text and Sketch User Study

In a user study, we investigated whether 2D shape queries can augment the performance of text-based retrieval of 3D models. Prior studies of Content-Based Retrieval for images show that shape, color and texture provide little benefit in a search engine with text matching - text similarity of captions dominates these matching modalities [97]. Our goal is to investigate whether this same behavior occurs when shape is added to text in a 3D model search engine. Our hypothesis is that 2D sketches are useful in conjunction with text for finding specific objects. To test this hypothesis, we ran an experiment where we compared the retrieval performance of text and 2D sketches provided by untrained users.

The subjects in this experiment were 43 students in an introductory computer science class (not for computer science majors, a different class from the one used in Section 5.2). Each subject was given a pen and sheet of paper and told that their task was to write text and draw sketches that could be used by a search engine to retrieve "target objects" from a database of household objects. After the process was demonstrated once by the professor, the subjects performed the test for five target objects from the Viewpoint database (described in Section 4.5.1). For each test, the target object was shown rotating around on a projection screen at the front of a classroom. After fifteen seconds (three rotations) it disappeared, and the students were asked to write up to five text keywords and to draw three 2D sketches from front, side, and top views that distinguish it from other household objects. They were given two minutes for each target object, and no feedback was given after each object.

Later, we scanned their sketches and logged their keywords so that we could enter them as input to our search engine (example sketches for a chair and an elf are shown in Figure 5.14). Table 5.2 lists results achieved with queries using: 1) only their text keywords, 2) only their 2D sketches, and 3) both text keywords and 2D sketches combined in a multimodal query. For each query type, the table lists the median

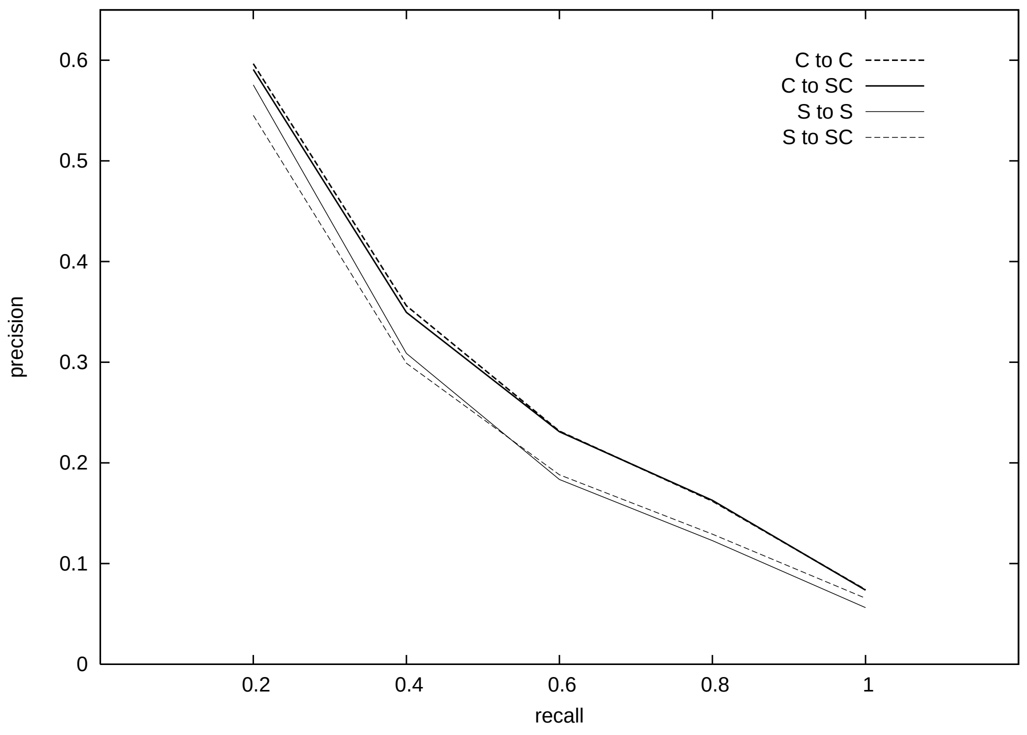


Figure 5.11: Three queries from S, C into the same database, and into SC

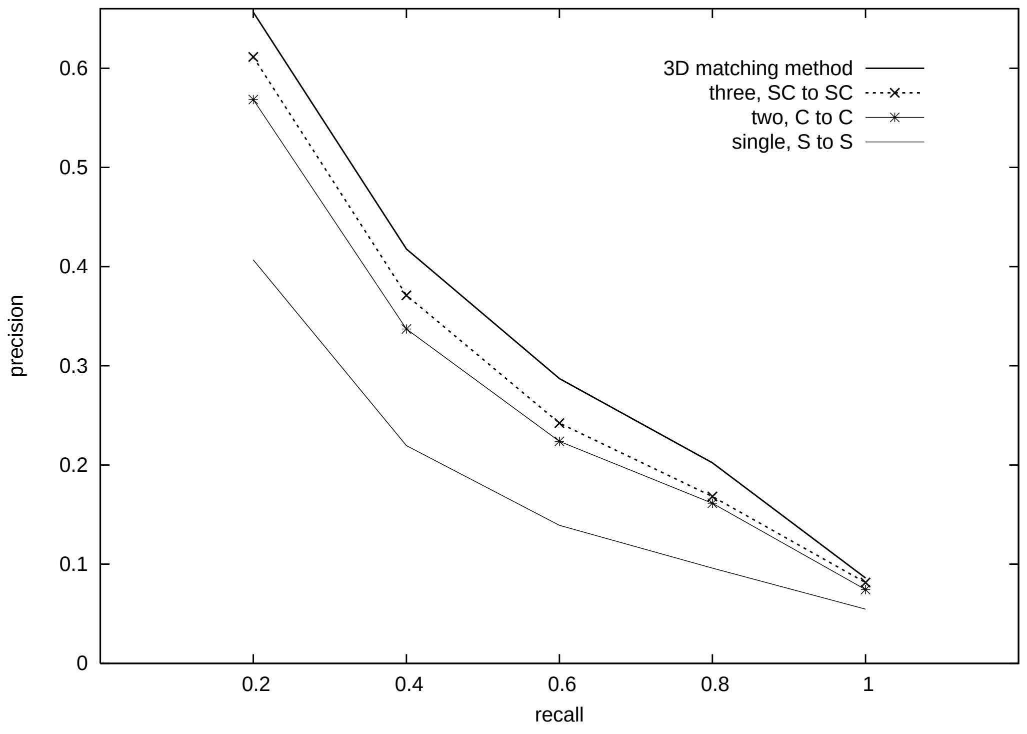


Figure 5.12: Average precision/recall for our 3D shape matching method, three queries from SC to SC, two queries from C to C, and a single query from S to S

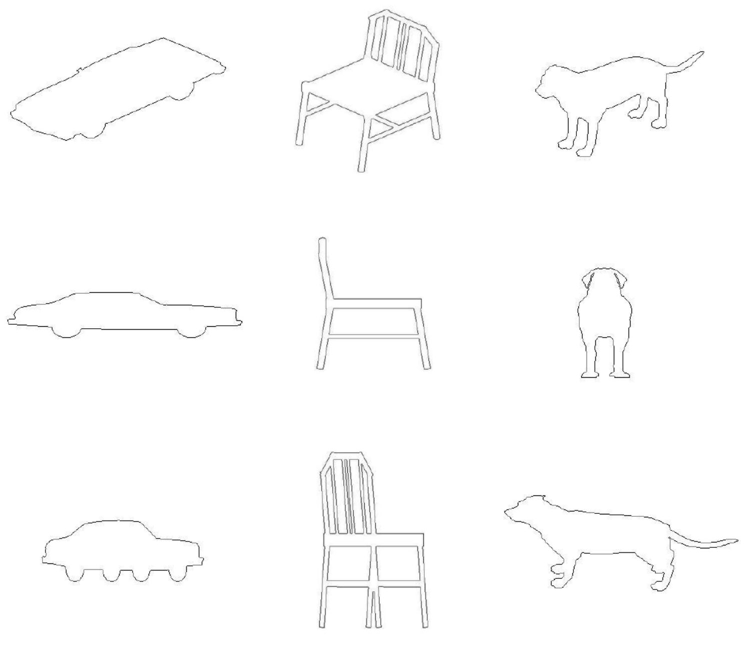


Figure 5.13: A corner view (top row), side view (middle row), and edge view (bottom row) of four example models (from left to a right, a car, chair, dog, and man), to illustrate the point that often an edge view is not the view we expect a user to sketch

ranks of the target object and the percentage of the queries where the target object appeared among the top 16 matches. The latter statistic reflects how often the target would appear on the first results page in our search engine.

The results in Table 5.2 suggest that text and shape can be complementary in the information they provide a search engine. For example, for the chair, text keywords were not discriminating enough to differentiate it from the hundreds of other chairs and related furniture in the database, and yet very simple sketches were able to describe it fairly precisely. On the other hand, simple text keywords picked out the five cannons and four bunk beds, while the 2D sketches were not as discriminating. Generally speaking, the text was effective at identifying classes of objects, while the sketches were helpful at selecting the best matches from within a class. The net result was that the combination of sketches and text usually produced a better match than either one alone.

5.5.3 Sketch Quality

To evaluate the quality of the 2D sketches created by web users, we examined a subset of all 2D sketches submitted in a period of about 3.5 months (November 9, 2002 until February 27, 2003). From the about 11,000 queries involving 2D sketches

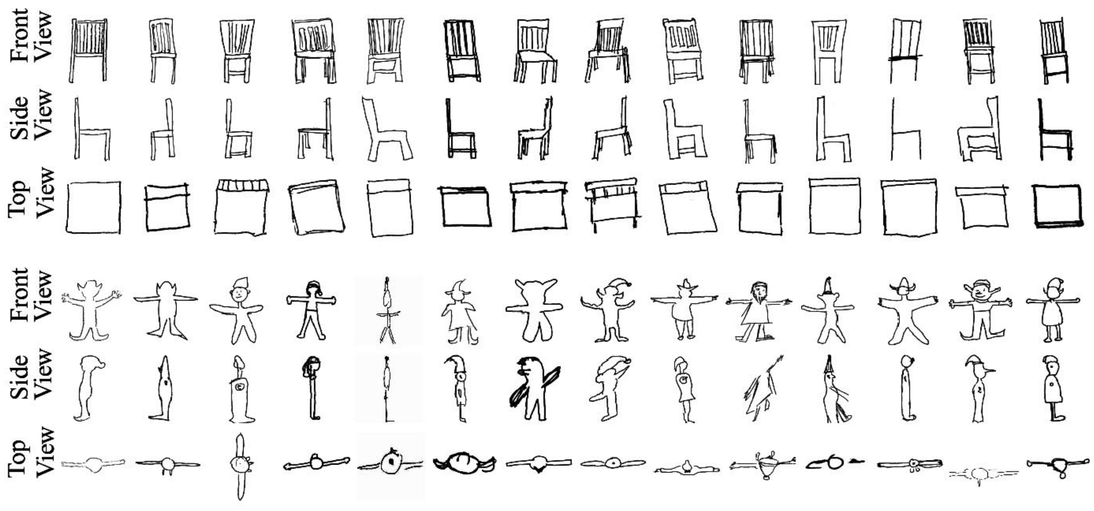


Figure 5.14: Sketches drawn by students to retrieve a specific chair (top three rows) and an elf (bottom three rows) during an in-class experiment

submitted in that period, we selected only those sketches from users (strictly speaking, from hosts with distinct IP addresses) who performed at least 20 such queries. This selection consists of 589 queries from 29 hosts, totalling 1,093 sketches. 46 queries were combined text+sketch, the remaining 543 were sketch only.

The sketches were classified according to the following criteria: (1) identifiable or not, (2) interior lines or not, (3) accurate or not. Note that (1) and (3) are subjective criteria. Figure 5.15 shows example sketches from this set, and how they were classified.

Of the 589 queries, 65% were for identifiable objects such as cars, trees, and humans. 49% of the queries had sketches with interior detail, which affects the shape

Target object name	Median rank of target (out of 1890)			% of queries with target in top 16 results		
	Text only	Sketch only	Both combined	Text only	Sketch only	Both combined
Chair	216	17	28	0.0%	46.2%	25.6%
Elf	10	12	2	89.7%	53.8%	97.4%
Table	100	571	252	5.1%	5.1%	10.3%
Cannon	7	40	2	82.1%	33.3%	89.7%
Bunkbed	3	64	2	89.7%	20.5%	89.7%

Table 5.2: Comparison of retrieval results with queries comprising only text, only 2D sketches, and both combined

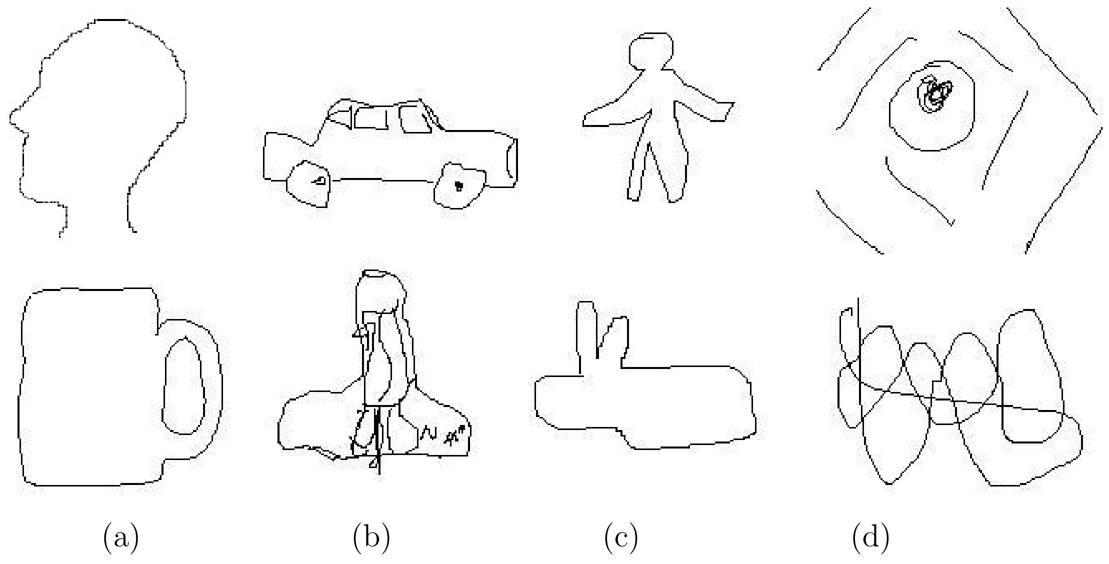


Figure 5.15: Examples of sketches submitted by web users, which were classified as follows: (a) identifiable, no interior lines, accurate, (b) identifiable, interior lines, not accurate, (c) identifiable, no interior lines, not accurate, (d) not identifiable, interior lines, not accurate

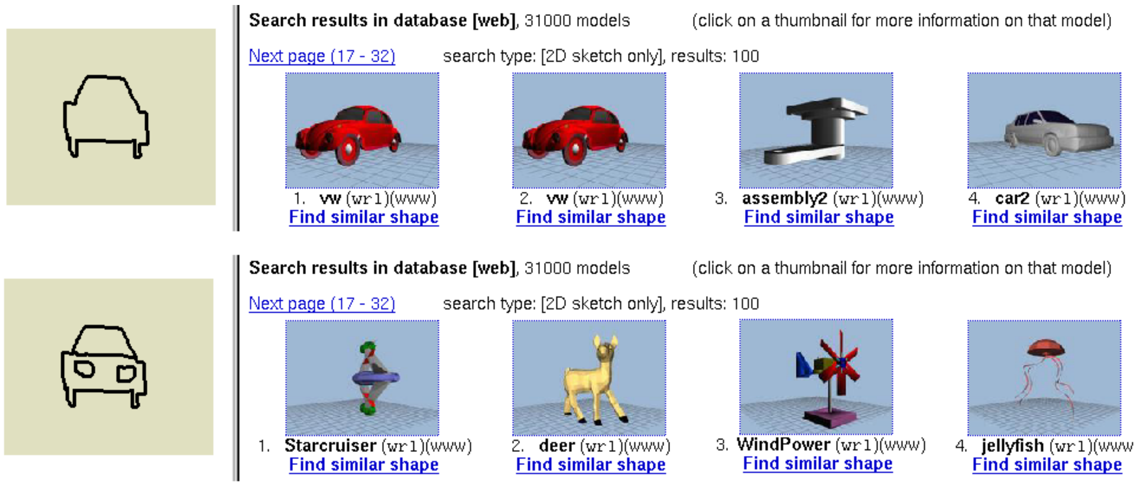
descriptor considerably. 71% of the queries were sloppily drawn (these were sketches with random looking lines, large errors in proportions, etc., as determined by us, also see Figure 5.15 (b) and (d)). The overall conclusion from examining these sketches is that the query sketches created by web users are mostly inaccurate, badly drawn (not everyone can draw well using pencil and paper, let alone using a mouse), and are not the boundary contours which would be best for matching.

The main problem for the matching method is the influence of interior detail, which greatly changes the EDT of the sketch, and thus the shape descriptor. To illustrate this effect, Figure 5.16 shows the first four matching results from our web database using a simple outline sketch of a front view of a car, and the same sketch with some interior detail added. This suggests that the user interface could be improved by making it easier to draw accurate sketches (for example using a line segment primitive), and by preventing or removing interior detail, or by augmenting our 2D image matching method to be less sensitive to the presence of interior detail.

5.6 Summary and Conclusions

In this chapter, we have investigated a simple 2D sketch-based query interface for our 3D model search engine. In our initial approach, we created a simple pen-drawing interface as used in Paintbrush-style programs. The benefits of this interface are its ease of use and the possibility to quickly sketch an overall shape.

To get a sense of the kind of sketches people draw in limited time, we conducted a


Search results in database [web], 31000 models (click on a thumbnail for more information on that model)

Next page (17 - 32) search type: [2D sketch only], results: 100

1. **vw** (wrl)(www) Find similar shape
2. **vw** (wrl)(www) Find similar shape
3. **assembly2** (wrl)(www) Find similar shape
4. **car2** (wrl)(www) Find similar shape

Search results in database [web], 31000 models (click on a thumbnail for more information on that model)

Next page (17 - 32) search type: [2D sketch only], results: 100

1. **Starcruiser** (wrl)(www) Find similar shape
2. **deer** (wrl)(www) Find similar shape
3. **WindPower** (wrl)(www) Find similar shape
4. **jellyfish** (wrl)(www Find similar shape

Figure 5.16: The first four matching results from our web database using a simple outline sketch, and the same sketch with some interior detail added

user study in which the subjects were asked to "draw the shape of an <object>", for eight different objects, in a short amount of time. Based on the results, we decided to precompute 2D views with *exterior outlines* of objects in the database, as seen from different directions. The user sketched outlines are matched to these 2D views of a 3D model using a fast image matching method developed by Michael Kazhdan [36]. The method compares 2D shape descriptors, which are invariant to similary transformations (with normalization for scale and translation) and reflections, and robust to small inaccuracies.

To evaluate the 2D matching method's performance and to investigate which 2D views are the most effective for classification, we ran a series of experiments using side, corner, and edge views of 3D models and their combinations, both as query views and as the views to compare against (i.e. the views stored with each 3D model). The classification tests were done using a large test database of 3D models, containing 1,093 manually classified models provided by Viewpoint [107]. Our main conclusions from these tests are: (1) using more query views improves the results, (2) using side and corner views for both queries and as the model representation has the best performance, (3) storing side and corner views while submitting only side or corner views does not significantly impact performance, and (4) this matching method is outperformed by our 3D shape matching method.

In another user study, we investigated whether 2D shape queries can help text-based queries for retrieval of 3D models. This time, subjects wrote descriptive text and drew several sketches of five target objects. We then submitted the text and sketch queries separately, and combined. We found that text keywords were effective at identifying classes of objects, and that sketches were helpful at selecting specific object in a class (for example, a chair with a straight back).

In future work, we plan to further investigate the question of which limited set of 2D views is best to represent a 3D object for matching purposes. Other tests will investigate the type of 2D image to use (e.g. outlines, filled images, dilated images) and the effect of enforcing consistency of projection directions when matching. We also intend to experiment with different ways to guide the user while drawing, for example by letting the user oversketch an image, modify an actual image outline, or by using other primitives (e.g. line segments).

Unfortunately, the usage results of our online 2D sketch interface show that most of the submitted 2D sketches are inaccurate and contain interior detail, greatly affecting the quality of the matching results. To improve this, an approach could be to provide more guidance to the user in the query interface, for example by letting the user draw (and manipulate) parts instead of freeform lines. Such a "structural" (parts-based) interface is the subject of the next chapter.

Chapter 6

2D Shape Query Interfaces: 2D Structure

6.1 Introduction

This chapter investigates a 2D structural query interface for our search engine.

An important drawback of using an image matching method to match user sketches to database images is that both images have to be similar in a pixel-by-pixel comparison. Both may be images of an object in the same class of objects, and have a similar *structure* (for example, both represent a table as seen from the side). But if they look slightly different (for example, the legs are in a different position), an image matching method will consider them not very similar at all. There are many types of objects that belong to the same class and have the same overall structure (e.g. humans, tables, motorcycles, helicopters, ...), with some variation in size, shape, position, and orientation of their parts (e.g. the orientation of a human's arms, or the height of a table). Our goal is to develop a query interface and matching method that can work for these types of object classes.

Describing objects as made up out of parts is attractive when considering how humans perceive objects. Evidence has been presented in the perception literature that humans think of shape as composed of a set of simpler, basic shapes. Biederman showed in his seminal "Recognition by Components" experiments that humans tend to partition shape into convex parts, whose boundaries are recognized near regions of deep concavity [11]. Because of this assumed correspondence to the human notion of shape organization, parts-based representations are popular in computer vision applications (for example, hierarchically organized cylinders [64], or superquadrics [76]).

To capture this notion in a shape query interface, we developed a parts-based query interface and accompanying matching method, in which the user provides a 2D *structure*. An example of a query for an animal created using our interface is shown in Figure 6.1. Parts can be drawn by dragging the mouse (similar to drawing ellipses or rectangles in Powerpoint, for example). Once created, parts may be deleted, moved, scaled, or rotated, using the mouse and/or keys.

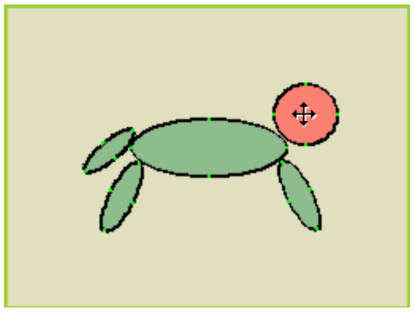


Figure 6.1: An example parts-based query for an animal using ellipses as primitive parts. Note that the head is being moved

The new challenge then is to match the user-provided parts to 2D projections of a 3D model. One approach could be to derive structure from the submitted parts and from the 2D projections, and match structure to structure. However, there does not exist a robust image segmentation method for arbitrary 2D projections of 3D models. Therefore, we decided to match the user input directly to the 2D projections.

For this approach, we need to address several issues: (1) which part primitive to use, (2) how to parameterize the user input, and (3) how to match a set of parts to a 2D image. For the part primitive we chose the ellipse, because it provides a good trade-off between expressiveness and the number of parameters. A tree is derived from a set of user-provided parts by letting the user select the root, and assigning parent-child relationships using heuristic rules based on part size and proximity. To match a set of ellipses to a 2D image we minimize an objective function consisting of four terms: it rewards (1) parts overlapping the image, and (2) parts aligned with the image's Medial Axis, and penalizes (3) parts overlapping each other, and (4) part deformation.

The various weights and parameters of the methods were optimized for classification using a training set of 75 models, classified into 15 classes of 5 models. We then compared the classification performance of this method to the 2D image matching method using a test set of the same size. We found that on average our structural matching method showed better classification performance, especially for classes in which the models are structurally very similar, but can have many different designs (for example, tables). Unfortunately the method is too slow (taking about 1 second for a single match on average) for interactive applications.

The remainder of this chapter is organized as follows. After a discussion of related work on computing and matching 2D structure, we describe our approach in Section 6.3. Results on the matching performance of our method are in Section 6.4, followed by a summary and conclusions.

6.2 Related Work

A wide variety of methods exist that compute 2D structure from 2D shape. For our matching problem, 2D structures could be precomputed for the 2D views of the 3D database models, and then matched to the user-provided structure using a structural matching method, such as attributed graph matching.

Shape Decomposition

Following Biederman's result that humans tend to partition shape into convex parts, whose boundaries are near regions of deep concavity, we can try to automatically decompose a shape into convex parts [11]. Marr and Nishihara show an example in which concavities are identified in a shape contour, which are then connected using heuristics [64]. However, it is unclear which heuristics have been used, and whether they would work in general. Optimal convex decomposition algorithms do not necessarily decompose a shape into parts the way a human would. Figure 6.2 shows two examples of simple polygonal shapes and a decomposition produced by the method of Keil and Snoeyink [55].

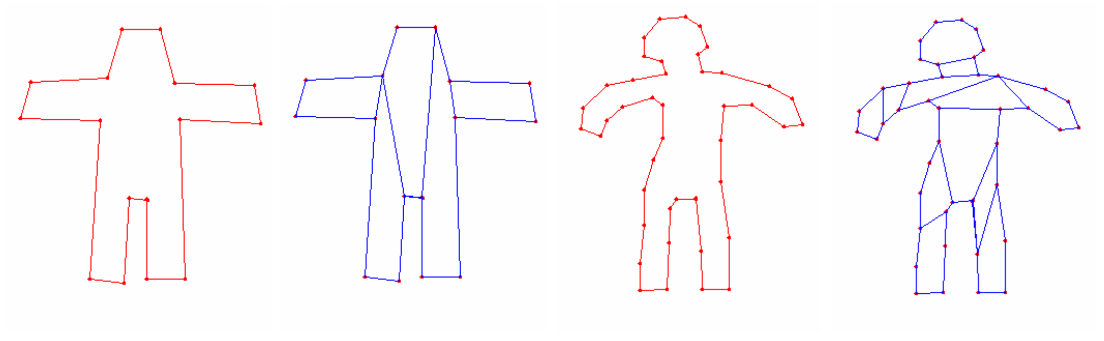


Figure 6.2: Optimal convex decomposition algorithms do not necessarily produce a good segmentation

Medial Axis Transform

The *Medial Axis Transform*, originally proposed by Blum in 1967 [14], has attracted a lot of attention because of its potential as a structural shape descriptor. Blum illustrated the transformation with a "grassfire" analogy: a fire is started at the shape contour, which propagates with constant velocity. The locus of points where the firefront meets (where the fire is quenched) is called the Medial Axis. It follows that these points are also the centers of the contour's maximal inscribed circles. Figure 6.3 shows an example 2D boundary, its Medial Axis, and two maximal inscribed circles.

There exist many methods to compute the Medial Axis (or approximations thereof)

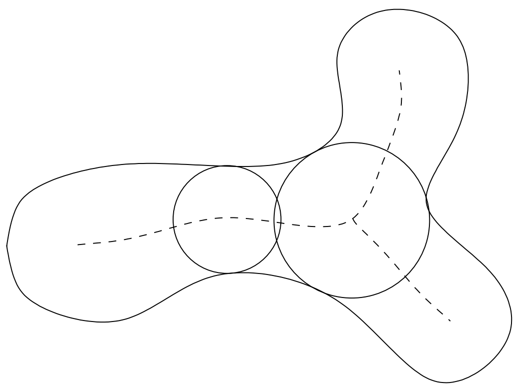


Figure 6.3: An example 2D contour, its Medial Axis (the dashed line), and two maximal inscribed circles

for 2D shapes (e.g. using topological thinning [58], active contours [61], Voronoi diagrams [8], subpixel tracing of flux field discontinuities [30], etc.).

However, the Medial Axis suffers from the problem that small changes of the boundary can cause large changes in the axis, both by creating extra branches as well as changing graph topology. Several methods attempt to alleviate this problem by incorporating a user-set detail threshold, which determines for example the size of the features (branches) that are included. Other methods prune the MA in a post-processing step (also using a user-set threshold that determines which branches are relevant).

As a result, corresponding matching methods have to compensate for these spurious branches and differing topologies (e.g. [8, 75, 91, 93]). For all these methods, however, it is not clear how well they will perform in the presence of noise: examples usually are given only for fairly simple shapes (much simpler than typical 2D views of a 3D model) with little boundary noise.

Deformable Templates

Methods based on deformable templates are either not parts-based [49], or target specific types of images such as ones containing cars [51] or faces [111]. In order to match arbitrary images, many different templates have to be pre-designed, each representing a certain class of objects. This is the approach followed by Anderson *et al.* [4] for recognizing scanned 3D clay models. They constructed representative 3D templates for 13 classes of objects, and added 9 variations of each representative template to account for part size variations within a single class. Using a similar approach for our matching problem would require a large amount of work, since we would have to create templates for every object class in our database. This method could still be useful if representative templates for object classes could be computed automatically, given a suitable training set (similar to the Active Shape Models by Cootes *et al.* [22]). This is a subject of future research.

We may conclude that currently there exists no method to decompose an arbitrary image of our type (i.e. a 2D projection of a 3D model) into a canonical set of simple parts. Even if we had such a method, we still cannot be sure that the user will provide the same set of parts for a certain object. Pentland, in his paper on representing 3D shapes using superquadrics, notes that domain experts may form different shape descriptions than those created by naive users [76]. In general, it is not clear what the canonical decomposition is of many shapes. For example, in Figure 6.4 two possible decompositions of a car are shown. This is an important challenge for methods which derive and match structure. It means that multiple nodes in one graph may correspond to multiple nodes in the other graph, greatly increasing the size of the search space.

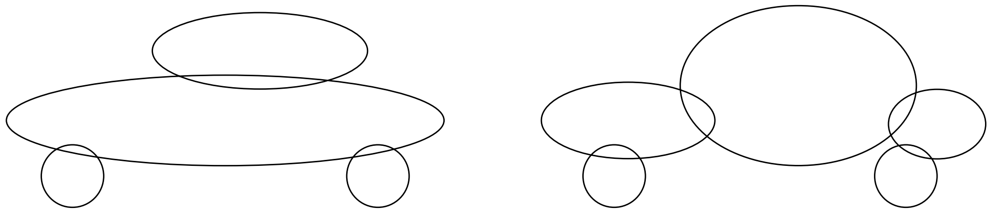


Figure 6.4: Two different part decompositions of a car

6.3 Our Approach: Matching Directly to the 2D Views

Because of this lack of a suitable decomposition method, we decided to avoid the shape decomposition problem altogether, and attempt to match the user input *directly* to the 2D views.

This amounts to matching a set of primitive parts to a 2D image, while allowing some structural deformations (see Figure 6.5 for an example). A similarity score is computed for the pair (user-drawn parts, image) by fitting the parts to the image. This is done by minimizing an objective function which computes an error of fit. This function rewards parts overlapping the image and parts aligning with the local maxima of the image's Euclidian distance transform (i.e. the Medial Axis), and penalizes deformation of parts and parts overlapping other parts. It is minimized using a common non-linear optimization method for complex functions for which a derivative cannot easily be computed analytically.

The advantages of this approach are threefold: (1) it is not necessary to precompute a robust 2D structural representation of the database images, (2) the user does not have to provide input consistent with such a representation, and (3) by allowing for variation in the relative position, sizes, etc., of parts when matching, it will be

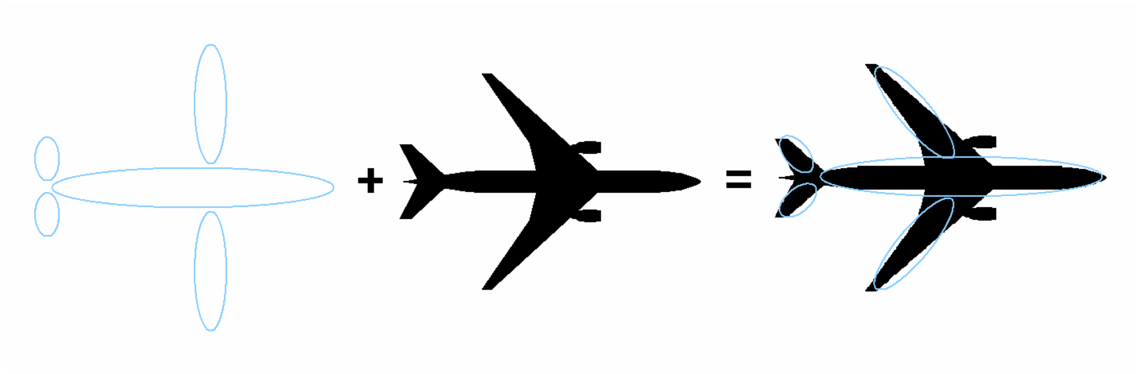


Figure 6.5: A 2D user-drawn set of primitive parts has to be matched to a 2D projection of a 3D model

possible to do matching of structure. A possible disadvantage is that this approach for matching structure may be slower than matching (simple) graphs.

For this method, we need to address the following issues: (1) which type of parts to use, (2) how to derive structure from a set of user-drawn parts, and (3) how to match this set of parts to a 3D model. These issues are the topic of the following sections.

6.3.1 Choice of Part Primitive

First, we have to decide what kind of primitive to use as the basic part of the interface. The interface should be easy to use, so the chosen primitive should be easy to create and manipulate by the user. For this reason, a simple primitive with few parameters (e.g. a circle) is preferred. On the other hand, the primitive should be sufficiently expressive to represent a part of an image of an object. This requirement favors a primitive with more parameters (e.g. a superquadric). As a result, there is a trade-off between ease of use and the "expressiveness" of a primitive. Additionally, the complexity of the primitive determines the number of parameters that may be varied in the matching step, and thus the complexity of the search space and the time until convergence. Since we have limited time to do the matching, having fewer parameters is better.

We now discuss several 2D parameterized primitives, in order of increasing complexity (i.e. the number of parameters), and their appropriateness for our application. They are shown in Figure 6.6: points, circles, squares, line segments, ellipses, rectangles, superquadrics [99], geons [29, 109], and generalized "ribbons" (the 2D equivalent of generalized cylinders [12]).

We dismissed the point primitive because it does not specify any extent. Line segments may be appropriate to represent articulated objects, but less so for other types of objects (for example, a mug, desk, or car). For non-articulated objects, and to be able to specify local width, we need to use a primitive that has area. Circles

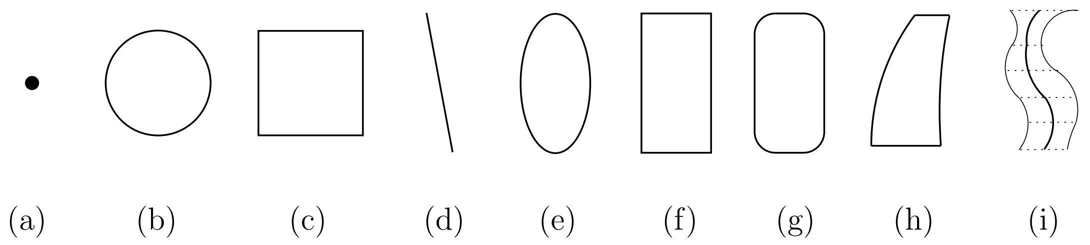


Figure 6.6: Possible choices for a part: (a) point (b) circle (c) square (d) line segment (e) ellipse (f) rectangle (g) superquadric (h) geon (i) generalized “ribbon”

or squares are the simplest choices, but unfortunately they cannot represent oblong parts. This leads us to rectangles, ellipses and superquadrics, which have a length and a width parameter. Additionally, a superquadric has a parameter that determines the “roundness” of its corners.

Increasing the number of parameters per primitive results in greater flexibility. The 2D equivalent of the 3D generalized cylinder, a generalized “ribbon”, is parameterized by an axis curve, and a width function along the axis. In practice, the curve is approximated by a piecewise linear curve, so it requires $3n + 4$ parameters ((x, y) and local width for each sample point, assuming the part is locally symmetric about the axis, and 4 for the starting position, orientation and scale), giving a minimum of 10 parameters per part.

By limiting the number of choices for the axis shape and size (i.e. local width) function, we can create a set of 2D *geons* (similar to 3D geons: for example, the size function can be constant, monotonically increasing, or monotonically increasing then decreasing [11]). This is a small subset of all possible 2D generalized ribbons. Even though both geons and generalized ribbons are much more expressive than the other primitives, we think these types of parts are too complicated for the average user. Also, the large number of parameters per part makes the matching step much more expensive.

Given this trade-off between (number of parameters, ease of use) and expressiveness, we chose the ellipse as the part primitive in the initial design of our structural interface. Sets of ellipses are easy to draw interactively, and they are expressive enough to describe many naturally occuring objects. The rectangle is almost equivalent, but is more awkward to manipulate when not oriented at a right angle. The superquadric adds one more parameter, which makes the interface more complicated (now the user has to specify the “roundness” of each part), and the optimization step more expensive.

6.3.2 Ellipse Parameterization

In this section we describe how we parameterize a set of ellipses as a single object. If we parameterize each ellipse independently (i.e. each ellipse has its own position, size,

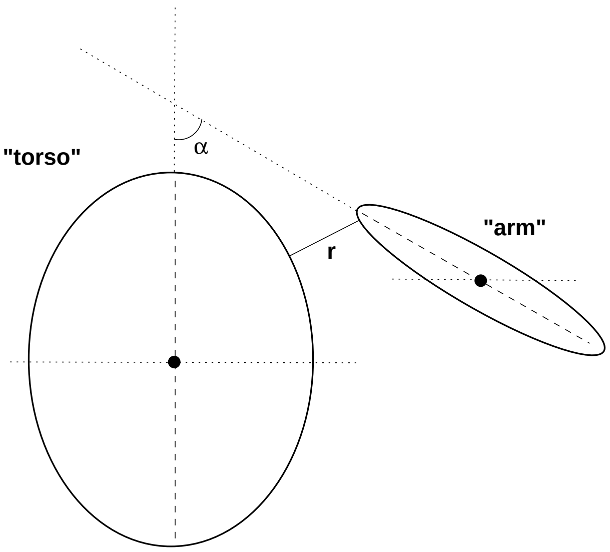


Figure 6.7: The relative angle α between two ellipses, and their shortest distance r

and orientation), it becomes difficult to express the kind of variation that we want to allow. For example, Figure 6.7 shows two ellipses representing a torso and an arm. If the parameters α, their relative angle, and r, the shortest distance between them, are part of the optimization, then we do not have to recompute them (i.e. derive them from position, size, and orientation of individual ellipses) at every optimization step.

Instead, we first apply a tree structure to a set of ellipses. The "root ellipse" is picked by the user, and every other ellipse is reparameterized in terms of its parent. In this way, changes to a parent ellipse propagate to all its children. For example, if the orientation of an "arm" part is changed, the "hand" part stays attached, which would not happen if all ellipses were parameterized independently. It is now also easier to specify allowed deformations in terms of the parameters of the representation.

To create this tree, first a complete graph is constructed, with each ellipse as a node. The weights of the edges of this graph are set to $wd^2 + 1/(\sqrt{a_1 + a_2})$, where d is the shortest distance between the two ellipses, and a_1 and a_2 are their areas. The distance term favors edges between nearby parts, and the area term between large parts, with w a weight controlling the terms' relative importance (currently set to 1/8000, through experimentation with a few common sets of parts). Next, the minimum spanning tree (MST) of this graph is computed. Figure 6.8 shows the edges that are created with these settings for a set of ellipses representing a human, for different positions of the leg. Note that even though the leg marked "5" is very close to the other leg in Figure 6.8 (a), it is still connected to the torso. Moving it further away from the torso creates a link between both legs. In Figure 6.8 (c) it is far enough away from the other leg to be again connected to the torso.

For the root ellipse the parameters (position (x, y), size (w, h) and orientation α) are kept. All other ellipses are parameterized in terms of their parent (see Figure

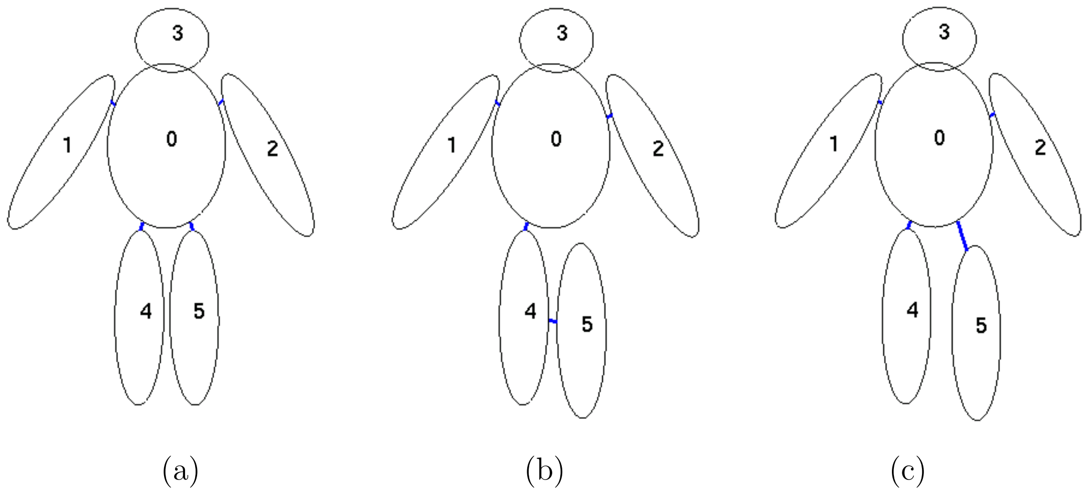


Figure 6.8: Links created for three different "leg" positions

6.9). The "child ellipse" parameters are:

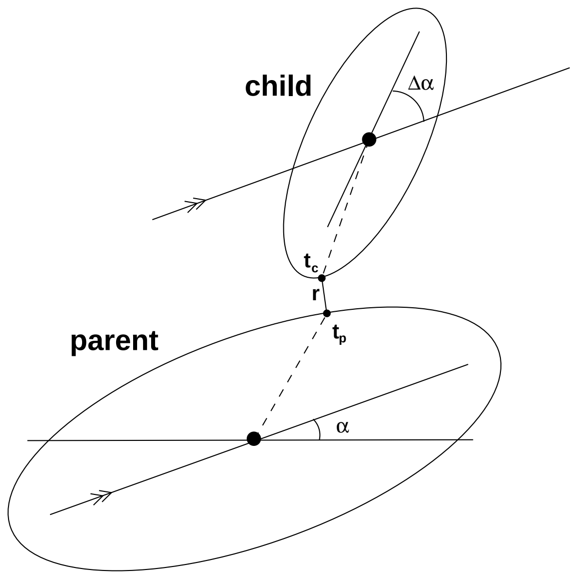


Figure 6.9: Reparameterizing a child ellipse in terms of its parent

- t_p, the attachment point on the parent. t_p is the t value in the parameteric equations for the ellipse, $x = a\cos t, y = b\cos t$ (with a and b the major and minor axis length)
- t_c, the attachment point on the child
- r, the distance between both attachment points

- $\Delta\alpha$, the relative angle of the child
- relative scale: the child's area divided by the parent's area
- ρ, the aspect ratio (height divided by width)

Note that now there is one more parameter for each ellipse because the child's position is represented by three instead of two parameters. This allows us to specify the penalty for changes in the attachment point on the parent, attachment point on the child, and the distance between those points.

6.3.3 Matching Method

The next issue is matching user provided structure to 2D projections of 3D models (the 2D views). The user-drawn parts are matched to a 2D view by minimizing an objective function which computes an error of fit. The design of this objective function, and the choice of minimization method are the subject of the following two sections.

Objective Function

Next, we need an expression for how well a set of ellipses fits an image and how much it has been deformed to do so. Intuitively, if a set of ellipses matches an image perfectly, then (1) the image is completely covered, (2) the "parts" of the image each correspond to an ellipse, (3) the ellipses do not overlap each other, and (4) the ellipses have not been deformed, at least not in a way that is not allowed. The objective function has four terms, each term corresponding to one of these criteria. As a formula, with weights added for each term:

$$E = w_i \cdot image\_overlap + w_a \cdot alignment + w_d \cdot deformation + w_p \cdot parts\_overlap$$

All terms are scaled such that they fall between 0 and 1, with 0 being the best matching score, and 1 the worst.

Image Overlap

This term gives the relative coverage of an image by a set of ellipses, and is computed as $1 - n_c/n_i$, with n_c the number of covered pixels, and n_i the total number of pixels in the image. To reduce the amount of compute time required, we store k random samples of each image, and compute the coverage value as $1 - s_c/k$, with s_c the number of covered sample pixels.

Parts Alignment

The second term is the only term that rewards "structural correspondence" between the set of ellipses and the image. This is done by comparing a sampling of the longest axis and boundaries of the ellipses to the Euclidian Distance Transform

(EDT) of the image. The computed value measures how well the longest axis of each ellipse aligns with the local maxima of the EDT, i.e. with the Medial Axis. As was described in Section 6.2, the Medial Axis (MA) is a popular structural shape descriptor. The major drawbacks of this representation are its sensitivity to boundary noise, and the instability of its graph topology.

We circumvent these problems by computing a continuous value for the alignment of an ellipse with the local maxima of the EDT. This alignment value is computed as follows. In preprocessing, the EDT of each image is computed, using the method developed by Saito and Toriwaki [87]. See Figure 6.10 for an example image and its EDT. For an ellipse to align with the local maxima of the EDT, along its major axis

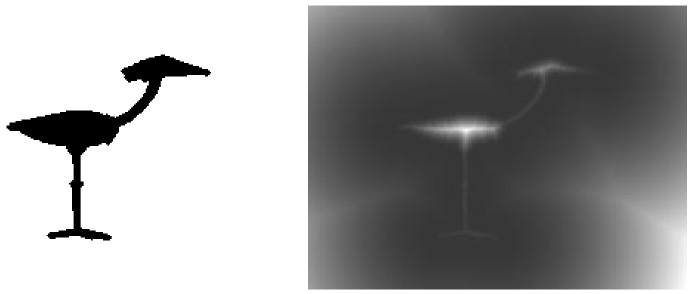
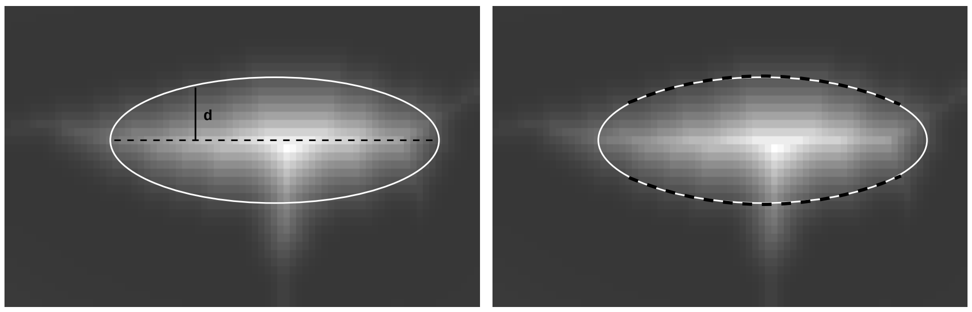


Figure 6.10: An example image and its Euclidian Distance Transform (EDT) (lighter pixels correspond to larger distances from the boundary)

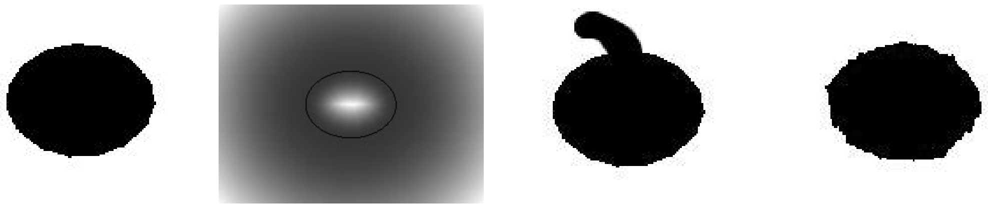
the local width should be identical to the EDT value, and along its boundaries the EDT should be zero. The alignment value is computed by sampling both the major axis and these “major boundaries”. Figure 6.11 (a) shows a closeup of the EDT of the bird in the previous figure, with an ellipse and some samples of the major axis. At each sample point the absolute difference between the ellipse’s local width d and the EDT value at that point is computed. This difference is normalized by dividing it by the minor axis length. Figure 6.11 (b) shows the same image with a number of samples on the major boundaries. For these sample points the average EDT value at each point, normalized by the minor axis length, is computed. The alignment value for a single ellipse is a weighted sum of the axis and boundary averages. The alignment value for a set of ellipses is simply the average alignment value over the entire set.

This alignment value is only slightly affected by boundary noise, and the location of MA branch points. Figure 6.12 (b) shows an example of an ellipse aligned with the EDT of the ellipse image in Figure 6.12 (a). Figures 6.12 (c) and (d) show the effect of adding (c) a protrusion, and (d) significant boundary noise. The alignment values for the ellipse are 0.04, 0.06, and 0.07 respectively (i.e. all close to zero). In contrast, the Medial Axis varies greatly for each image. Figure 6.13 shows the pruned “shock graph” (this is a Medial Axis based representation [92]) for the same three images (images in Figure 6.13 courtesy of F. Leymarie). The full shock graphs were pruned according to the method described by Tek and Kimia [103].



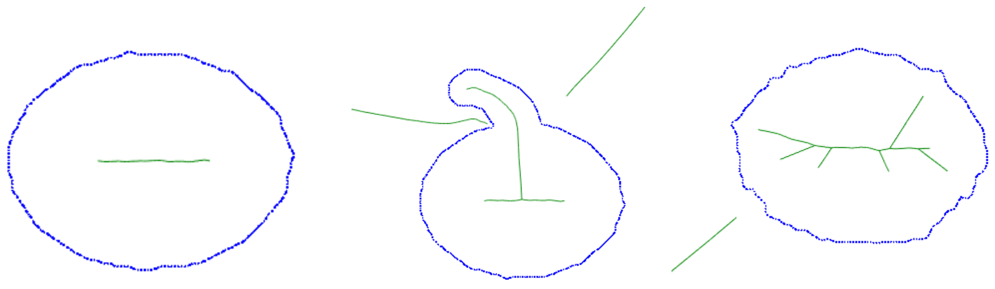
(a) (b)

Figure 6.11: Computing the alignment of an ellipse with the EDT



(a) (b) (c) (d)

Figure 6.12: (a) an image of an filled ellipse, (b) an ellipse aligned with the EDT of the image (alignment value 0.04), (c) the same ellipse image with a protrusion added (alignment value of the same ellipse 0.06), and (d) with more boundary noise added (0.07)



(a) (b) (c)

Figure 6.13: The simplified shock graphs for the images in Figure 6.12 (a), (c), and (d)

Parts Deformation

Besides terms that reward correspondence between the ellipses and the image, terms that penalize unwanted behavior are also required. Therefore we add a deformation term that penalizes changes in parameters such as aspect ratio, relative orientation, and attachment point. By weighing the deformation term and its individual parameters appropriately, we can control to what extent part deformation affects shape similarity. For example, if the user draws a set of ellipses representing an airplane with the wings at right angles, and this set should match an image of an airplane with the wings at 45 degrees, then the penalty for changing the relative angle of parts with respect to their parents should be relatively small.

Next we describe in detail how the deformation term is computed. In the following formulas in this section, the subscript $_1$ denotes values of the original (starting) set of ellipses, and the subscript $_2$ values of the deformed set.

The deformation error for the root ellipse is different from the error for the other ellipses. Because its error should be translation, scale, and orientation independent, it depends only on a change in aspect ratio, and equals $w(1 - \rho_1/\rho_2)^2$, with w a weight factor, and assuming $\rho_1 < \rho_2$ and both aspect ratios < 1. This error function measures relative change, is symmetric across the line $\rho_1 = \rho_2$, and is C1 continuous.

The error for the remaining ellipses, i.e. the ones that have a parent and are parameterized in terms of their parent, depends on the change in (1) the attachment point on the parent, (2) relative orientation, (3) relative scale, (4) aspect ratio, and (5) distance to parent. In the equations below, $\ominus$ is an operator which returns the smallest difference of two values in a circular domain (which is $[0, 2\pi]$ both for t and α, the resulting range is $[0, \pi]$). For example, if $t_1 = \pi/6$, and $t_2 = 11\pi/6$, then $t_1 \ominus t_2 = \pi/3$, see Figure 6.14.

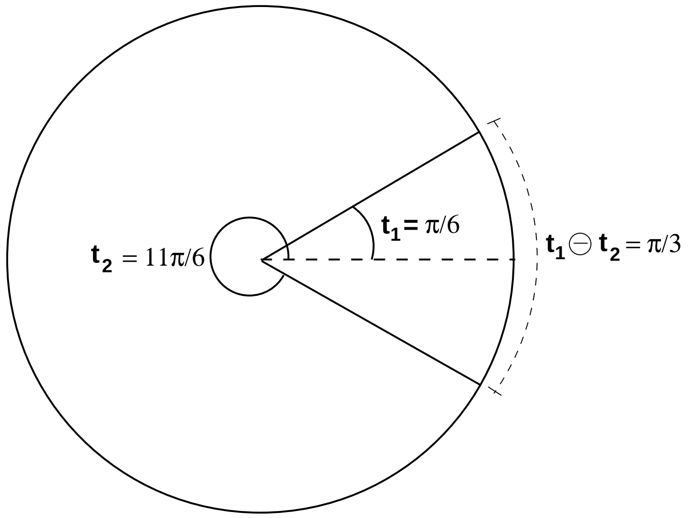


Figure 6.14: An example of a result of the $\ominus$ operator

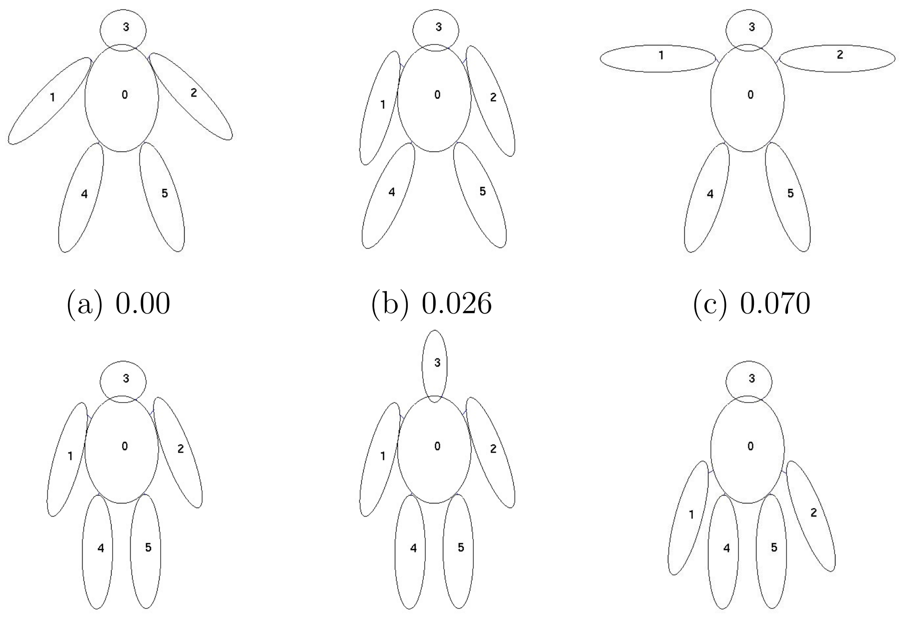
1. $err_t = (t_1 \ominus t_2)^2$
2. $err_\alpha = (\alpha_1 \ominus \alpha_2)^2$
3. $err_s = (1 - s_1/s_2)^2$, assuming $s_1 < s_2$
4. $err_a = (1 - \rho_1/\rho_2)^2$, assuming $\rho_1 < \rho_2$
5. $err_r = (|r_1 - r_2|/r_{img})^2$, with r_{img} the average radius of the database image

The total error is a weighted sum of the individual terms:

$$(w_s err_s + w_\alpha err_\alpha + w_t err_t + w_a err_a + w_r err_r)/5$$

Each of the weights has to be set to an appropriate value, such that its range corresponds to our intuitive notion of which deformations are acceptable and which are not. To determine these values, we explored the configurations that were allowed for error values less than 1.0, for various sets of ellipses and combinations of the individual terms. Figure 6.15 shows an example of a set of ellipses representing a human figure, and some pose variations with their corresponding deformation error values.

From these tests we determined the weight settings. Currently we use the following settings: $w_t = 2, w_\alpha = 1/(\pi/4), w_s = 4, w_a = 8, w_r = 5$. The deformation error for a set of ellipses is simply the average error over all ellipses.



(a) 0.00 (b) 0.026 (c) 0.070

(d) 0.035 (e) 0.14 (f) 0.39

Figure 6.15: (a) A set of ellipses representing a human figure. (b)-(f) Pose variations and their corresponding deformation error values

Parts Overlap

In certain cases, the deformation term does not penalize undesirable configurations. For example, two ellipses may initially be very near each other and end up covering the same part in the image, at a low deformation cost. For this reason, we added a fourth term that penalizes parts overlapping each other.

Instead of computing the exact intersection between each pair of ellipses, we use an appromixation. The boundary of each ellipse is sampled at k points, and each point is checked to see if it is inside any of the other ellipses. The overlap value then is n_i/k, with n_i the number of boundary points that falls inside another ellipse. The overlap values are averaged over all $2 \cdot \binom{e}{2}$ combinations (with e the number of ellipses).

Because ellipses may overlap in the initial query set, we compute all pairwise overlaps before the optimization starts, and use these as the "zero point" for each (i.e. these values are subtracted from the computed overlaps during the optimization).

Optimization Method

Because our matching method is a component of an interactive application (a shape query interface), the optimization stage in which a set of ellipses is matched to an image has only limited time to run. Therefore, the optimization method we pick should quickly converge to a local minimum (given the limited time we cannot hope to find a global minimum). We can only compute approximations of the derivatives of the first two terms of our objective function, because they depend on the image we are matching to. These are expensive to compute, and not likely to be of use to methods that use the derivative to speed up convergence, because of the irregular nature of the image.

We tested three different optimization methods that do not need derivatives, using the implementations presented in Press *et al.* [78]. In early experiments we used simulated annealing to find a global minimum, but this method turned out to be too slow for real-time applications (4 minutes for a single match on a 1.5 GHz Pentium III). Another tested method was Powell's direction set method, which runs much faster. However, because we cannot run the method long enough to improve the direction set, the obtained result becomes dependent on the order in which the parameters are optimized. A third method, the Multidimensional downhill simplex method, was able to produce reasonable minima in limited time. It optimizes by changing the position of the vertices of a high-dimensional simplex, usually picking the vertex with the highest objective function value at each step. Possible vertex position changes include reflection, reflection and expansion, and contraction. The optimization ends when the simplex is small enough.

Initial Guess

If a set of ellipses has roughly the same size, position and orientation as the image it is being fit to, then the optimization step will be more likely to find a good local (or global) minimum. Therefore, before we start the optimization, we try to find a good initial guess. We normalize for translation (position) by aligning the center of mass of the ellipses and the image, and normalize for scale using the average radius (the average distance of each pixel to the center of mass) of both. Both these steps are less sensitive to outliers than when normalizing based on a bounding rectangle.

For finding a good initial orientation, we tried two methods. Both methods try a limited number of orientations and pick the one with the smallest objective function value. The first method tries all eight possible alignments of the two pairs of principal component vectors of the set of ellipses and the image. The second method simply tries a relatively large number of orientations (the set of ellipses rotated in steps of $\pi/25$ radians, also reflected through a line through the origin, for a total of 100 orientations).

We evaluated both methods by looking at the alignments it produced for a test database of 75 sets of ellipses and 75 corresponding images. The images were classified into 15 classes of 5 images each. Each set of ellipses should align correctly with the images in its class, yielding $15 \times 5 \times 5 = 375$ correct alignments. We found that the principal component based aligment produced a correct alignment in 77% of all cases, and the method that tries 100 orientations in 85% of all cases. The images in six of the 15 classes were relatively easy to align, because of their strong similarity (these classes were fish, floorlamp, gun, fighterjet, helicopter and sword). Leaving out these "easy classes," the correctness percentages went down to 65% and 78% respectively.

Based on these results, in all subsequent tests we used the second alignment method. The actual number of initial orientations to try was determined by evaluating the classification performance depending on this number.

6.4 Results and Discussion

In this section we present results of experiments designed to evaluate the classification performance of our structural matching method. Two test databases were created. The first (the "training set") was used to optimize the weight settings of the objective function, the second (the "test set") to evaluate classification performance.

6.4.1 Test Databases

The matching method was evaluated using two databases of 75 models each, both classified into 15 classes of 5 models. The models were selected from our model databases of about 31,000 web models and 5,000 commercial models. Types of models that were expected to benefit from a structural matching method were preferred, since we are evaluating a method whose goal it is to match structure. So, for example, there are classes "human" and "helicopter," and no classes "couch" or "telephone". Figure 6.16 shows all 75 models of the training set, Figure 6.17 shows all 75 models of the test set.

1.	bird	
2.	car	
3.	chair	
4.	dinosaur	
5.	fish	
6.	gun	
7.	helicopter	
8.	human	
9.	jet	
10.	lamp	
11.	plane	
12.	quadruped	
13.	sword	
14.	table	
15.	tree	

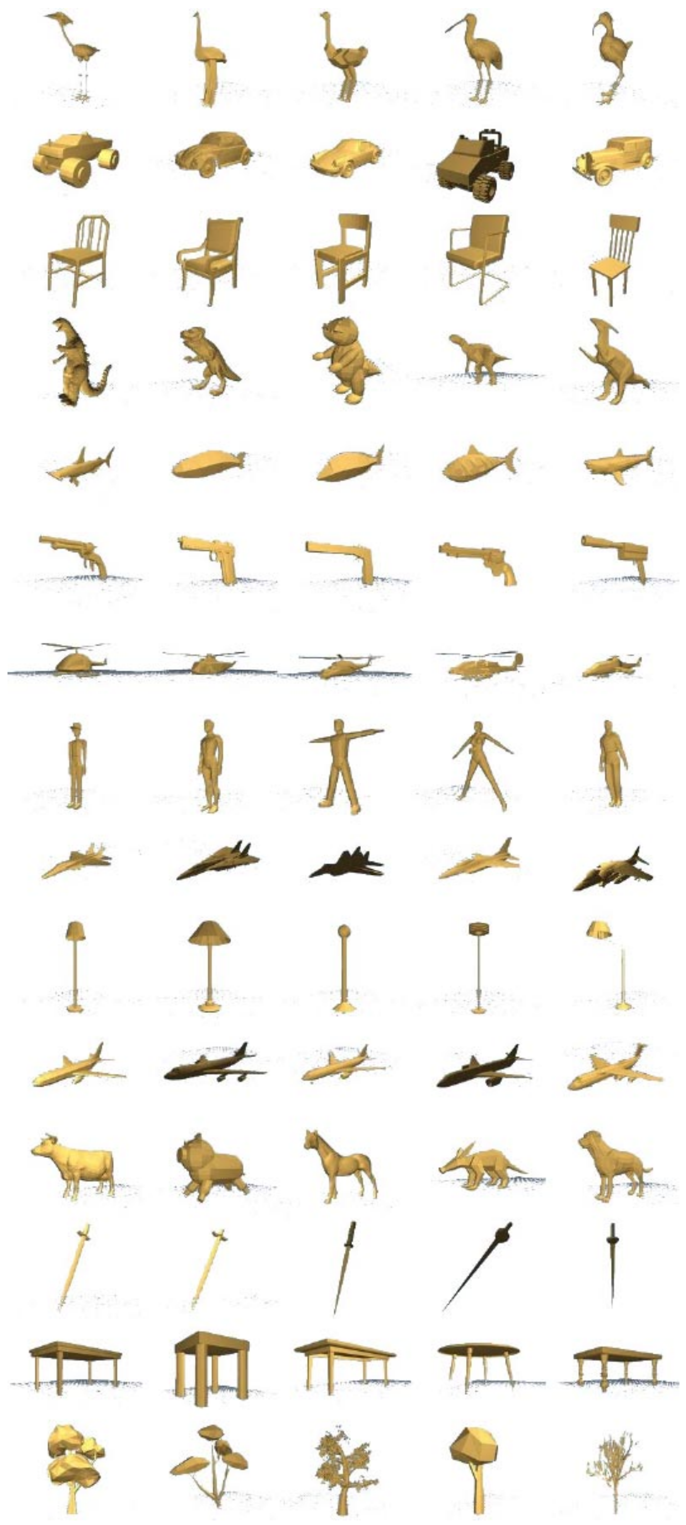


Figure 6.16: Screenshots of all 75 models in the training set

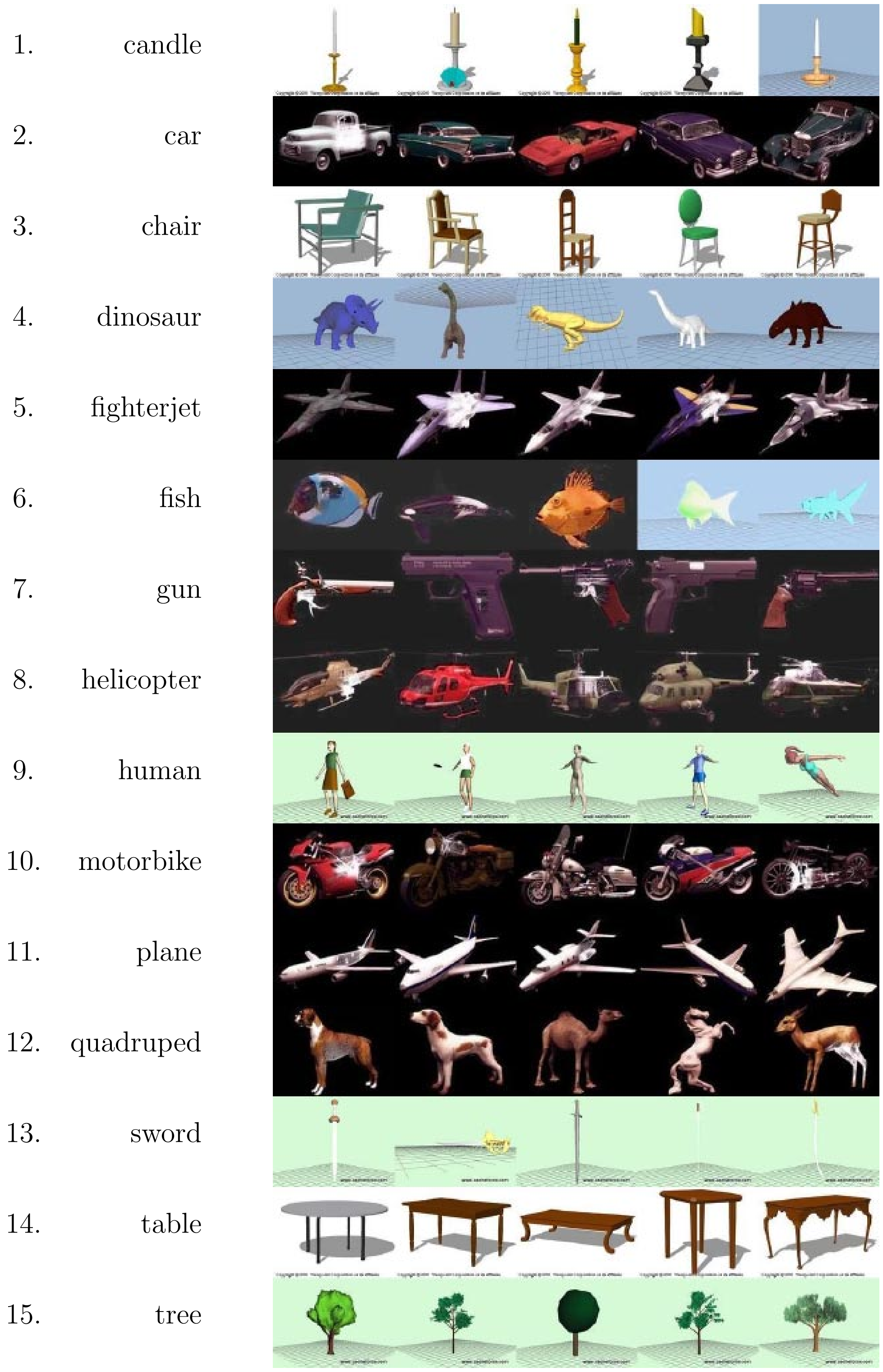


Figure 6.17: Screenshots of all 75 models in the test set

One of the databases was used as a training set, to optimize parameter settings. The parameters that were optimized are the four weights of the individual terms of the objective function, as well as several weights within each term (for example, the number of random image samples to use for the image overlap term).

In our matching method, we are matching sets of ellipses (the user input) to 2D views (2D images, projections of the 3D models). To evaluate the method, we have to create both sample user input, as well as sets of 2D views of the 3D models. The 2D views we used were the three orthographic "plan" views (side, front and top views, which were labeled "S" in Section 5.5.1). One of the 75 models was not aligned with any of the XOY, XOZ, or YOZ planes, and had to be manually aligned. To ensure that thin features (such as the blades of a helicopter) would still appear in the relatively small (256×192) images, we initially created all views at a 512×384 resolution, dilated those images three times, and then scaled them down to 256×192. We created sample user input for each model by oversketching *one* of the three views with a set of ellipses (we found that matching a single set of ellipses to an image already took a relatively long time, so using two or three sets would increase the turnaround time by a factor of two or three). Out of the three views, one was selected as the most "characteristic" view and oversketched using a simple ellipse drawing program, using up to 7 ellipses (to limit the number of parameters of the optimization). This set of ellipses was used as the user input targeting this particular model. Figure 6.18 shows some example 3D models, their three views, and the view that was chosen to oversketch. Figure 6.19 shows the ellipses drawn for the selected views.

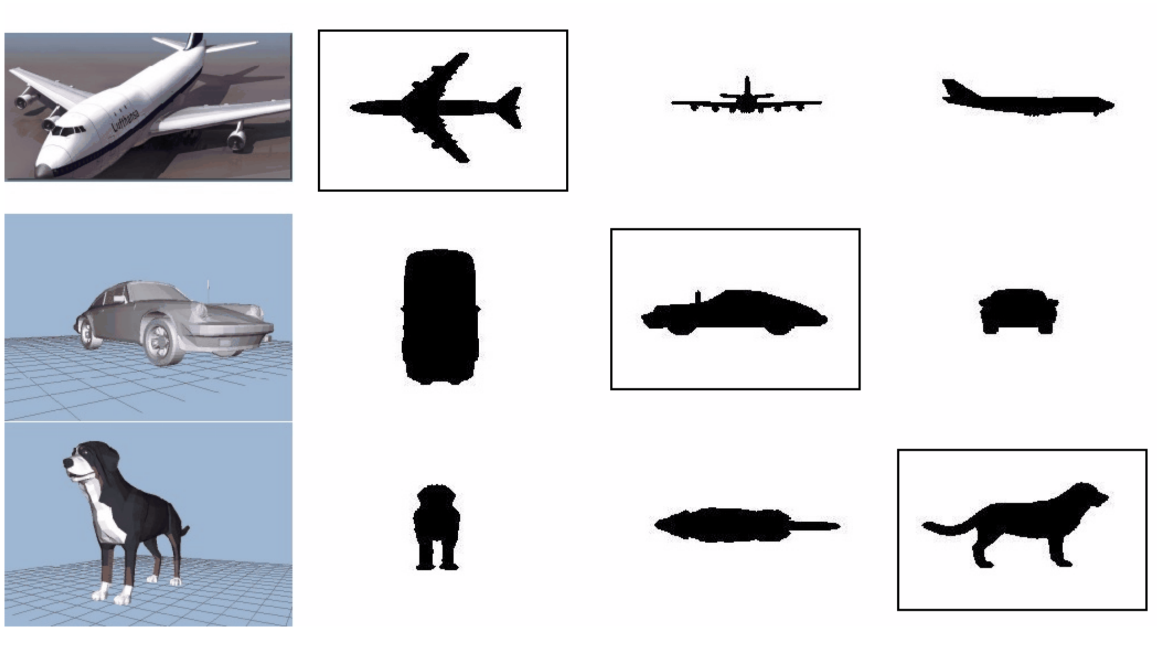


Figure 6.18: Example models from two small test databases, their three plan views, and the selected "characteristic" view

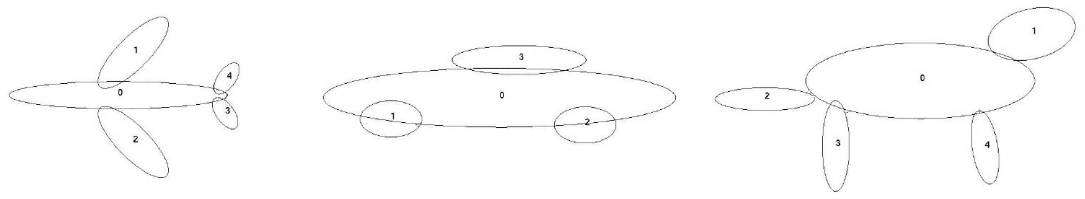


Figure 6.19: Example ellipses drawn for the selected views in Figure 6.18

6.4.2 Parameter Optimization

We used the training set to optimize the weights w_i, w_a, w_d, and w_p of the four terms of our objective function:

$$E = w_i \cdot image\_overlap + w_a \cdot alignment + w_d \cdot deformation + w_p \cdot parts\_overlap$$

To evaluate a certain weights setting, we measured the average precision (i.e. the precision for recall values $0.2, 0.4, 0.6, 0.8, 1.0$ divided by 5) when submitting each of the 75 sets of ellipses as a query, and computing similarity values based on matching them to the $75 \times 3 = 225$ database images (see Appendix A for an explanation of the precision/recall performance metric).

Initially we set all weights to 1.0. Then each weight was varied individually. Figure 6.20 shows how the average precision changes when the weight of each of the objective function's terms is varied individually. For each weight, the average precision measure reaches a maximum for the value 1.0. As a result, we used the original weight setting to evaluate classification performance on our test database.

Decreasing or increasing a term weight has a different effect on the results depending on the term. We will now look at some specific examples to illustrate these effects. Each example shows what happens when a weight is set too low, "just right", or too high (there are of course specific cases where a low or high setting of a weight resulted in better matches, but on average the middle setting worked best).

Figure 6.21 shows the first five matches for a set of ellipses representing a bird, for three settings of the **image overlap** term weight (0.25, 1.0, and 3.0). A result with a yellow background indicates a match with the query itself, and results with a green background matches with objects from the same category. The first setting (0.25, Figure 6.21 (a)) is not high enough, resulting in matches with images that have good EDT alignment, but too many uncovered pixels. Setting the weight to 3.0 (Figure 6.21 (c)) yields matches with bad EDT alignment in favor of covered pixels. The best setting (Figure 6.21 (b)) results in matches with three birds in the top five results.

Figure 6.22 shows a similar sequence for a set of ellipses representing a floorlamp, this time setting the **EDT alignment** weight to the same three values (0.25, 1.0,

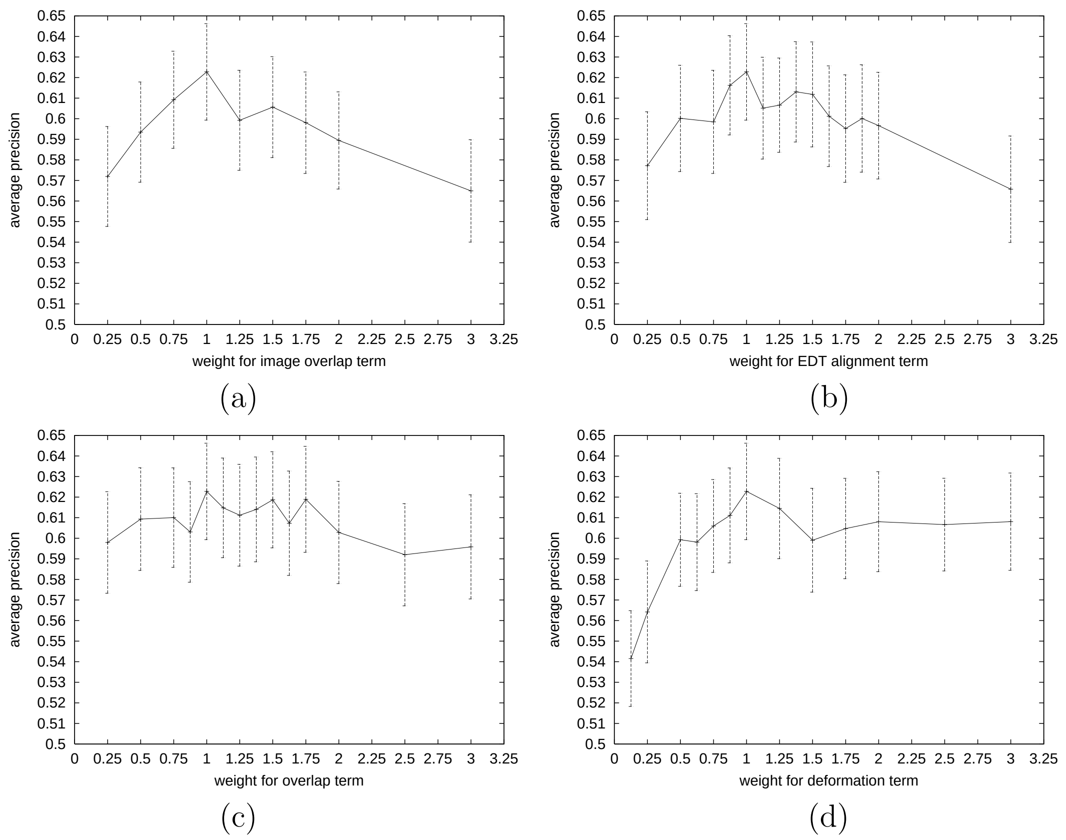


Figure 6.20: Average precision for different values of the weight of the (a) image overlap term, and (b) EDT alignment term, (c) ellipse overlap term, (d) deformation term. Note that in each graph the y-axis starts at 0.5 and ends at 0.65

and 3.0). In Figure 6.22 (a) the table and passenger plane are top matches because the ellipses cover the image well, without a lot of deformation or mutual overlap, and the misalignment with the EDT is underweighed. Conversely, in Figure 6.22 (c) the ellipses are deformed too much to achieve a good EDT alignment.

The weight of the **part overlap** term had very little effect on classification performance (as could already be seen in Figure 6.20 (c) from the small variations in average precision depending on this weight). Figure 6.23 does show an instance of a query for which 1.0 was the optimal weight setting, but there were very few cases like this.

Finally, Figure 6.24 shows the effect of setting the **deformation** weight to 0.25, 1.0, and 5.0 respectively, for a set of ellipses representing a table (5.0 because 3.0 showed no decrease in performance). Note that in this case, because of the large variability in the class of tables, and their structural similarity, the lowest weight setting actually produces the best results.

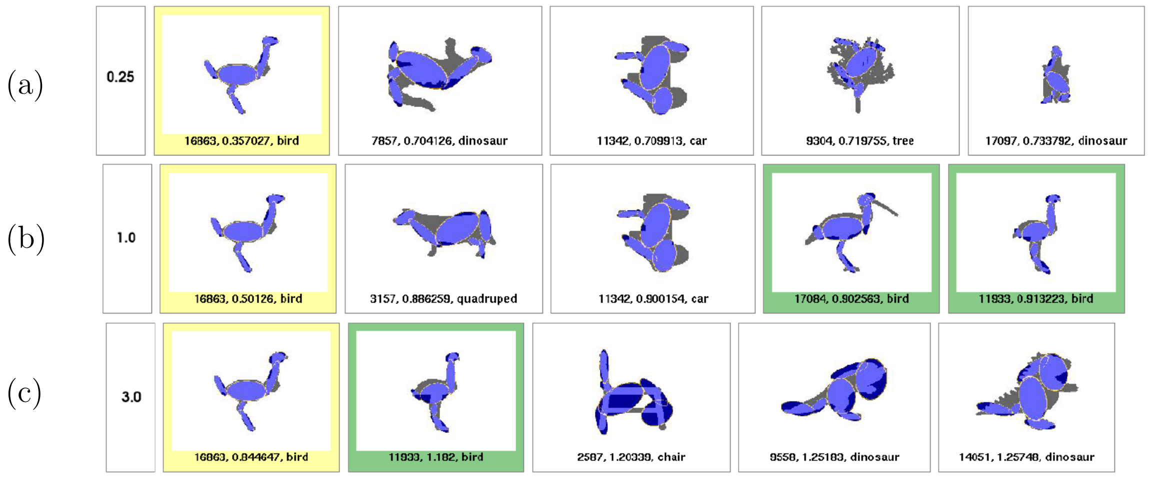


Figure 6.21: The top five results for a set of ellipses representing a bird, with the image overlap weight set to (a) 0.25, (b) 1.0, (c) 3.0. Below each image an id number, similarity score, and class name are shown. The id number of the query model was 16863

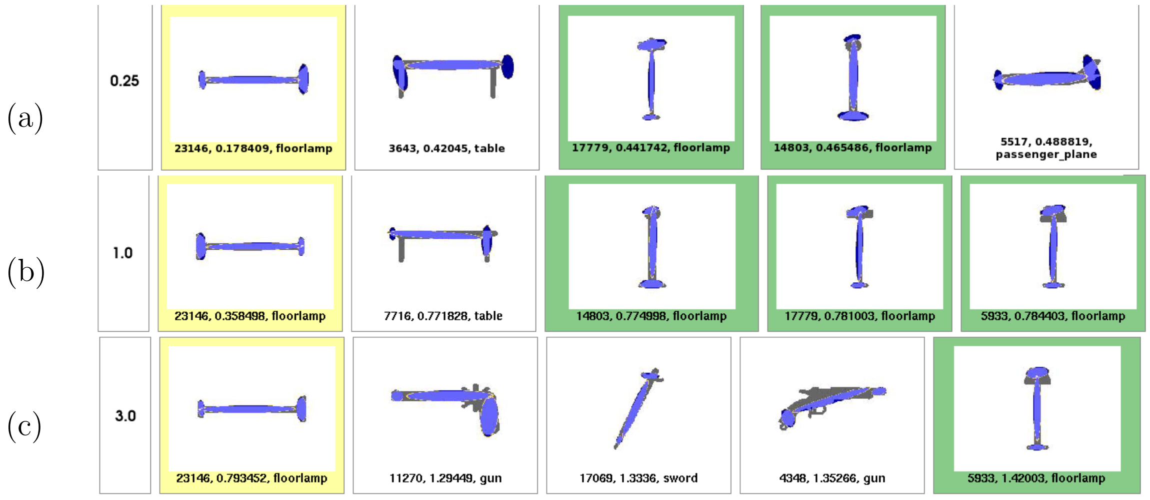


Figure 6.22: The top five results for a set of ellipses representing a floorlamp, with the EDT alignment weight set to (a) 0.25, (b) 1.0, (c) 3.0. Below each image an id number, similarity score, and class name are shown. The id number of the query model was 23146

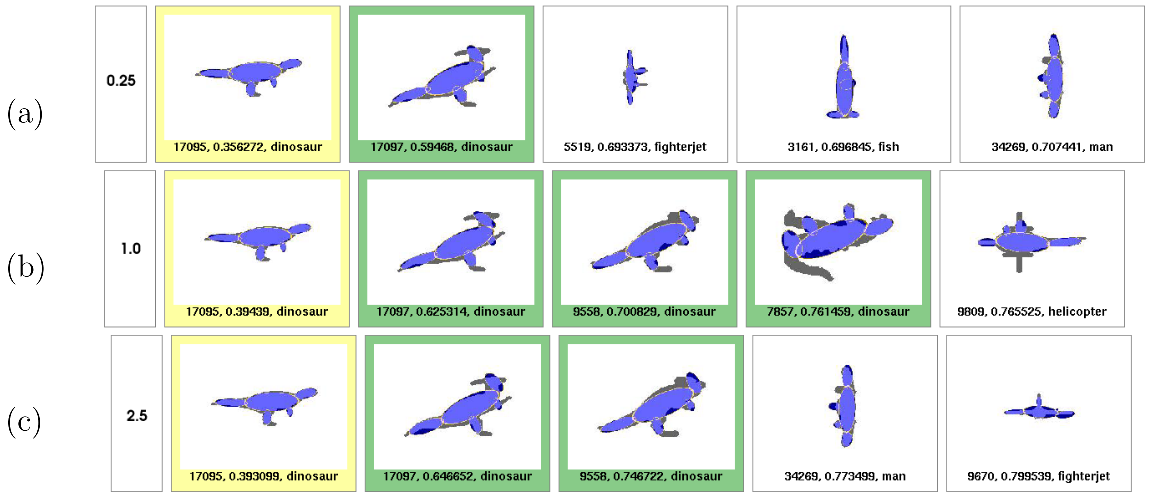


Figure 6.23: The top five results for a set of ellipses representing a dinosaur, with the part overlap weight set to (a) 0.25, (b) 1.0, (c) 2.5. Below each image an id number, similarity score, and class name are shown. The id number of the query model was 17095

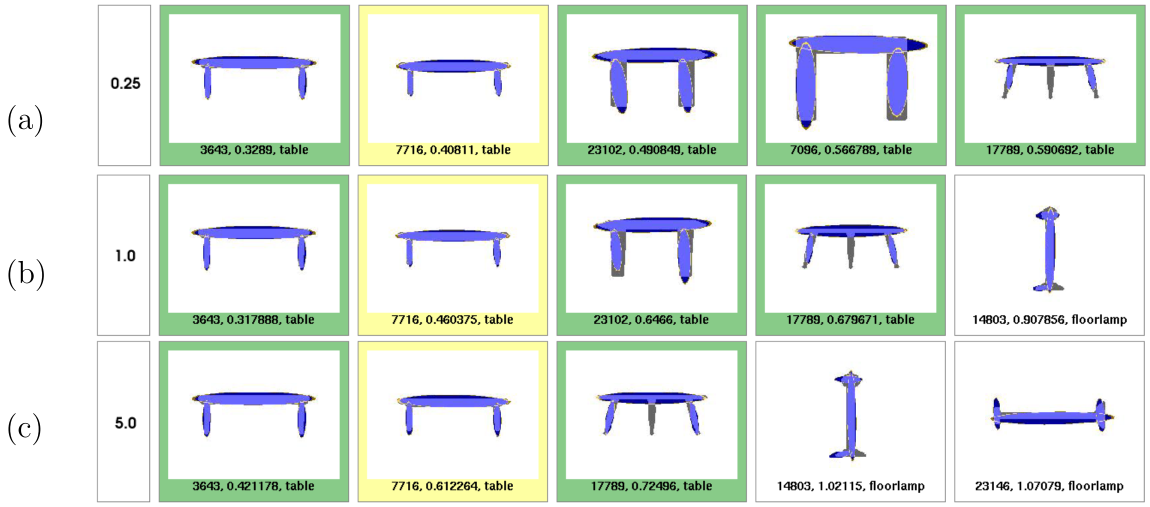


Figure 6.24: The top five results for a set of ellipses representing a table, with the deformation weight set to (a) 0.25, (b) 1.0, (c) 5.0. Below each image an id number, similarity score, and class name are shown. The id number of the query model was 7716

Other Parameters

Next, we fixed the weight settings to 1.0 and measured average precision when varying several "secondary" parameters of the optimization.

The first is the number of initial orientations to try to find a good initial alignment guess. Figure 6.25 (a) shows the average precision (of the five average precision values at recall values 0.2, 0.4, 0.6, 0.8 and 1.0) for different settings of this parameter. Based on these results, we set this number to 100. Figure 6.25 (b) shows the average precision for different numbers of random image samples to use for the image overlap term. The horizontal line shows the average precision achieved when the exact number of overlapped image pixels is counted for this term. Given these results, we picked the value 200. Figure 6.25 (c) shows the same results for the number of boundary samples that are used to compute an approximation of the ellipse overlap term. The value we used throughout is 32, yielding a performance not significantly lower than the apparent maximum at 24. Figure 6.25 (d) shows the results when varying the number of samples used along the ellipse's major axis when computing the EDT alignment term. Originally we used 37 samples (from -0.9 to +0.9 along the axis, with 0.0 the center and -1.0 and 1.0 the extremes, in steps of 0.05), but this graph shows that 16 would have sufficed.

Finally, using the initial pairwise overlap (i.e. the overlap specified by the user) between ellipses as the "zero point" (as described in Section 6.3.3) resulted in an increase of the average precision value by 8%, when compared to penalizing any overlap.

6.4.3 Comparison to our 2D Sketch Matching Method

To see if our structural matching method can improve classification performance over a 2D image matching method, we compared it to our 2D sketch matching method (as described in Section 5.3) using our test database of 75 models. The outlines of the sets of user drawn ellipses were matched to the outlines of the three images stored for each database model. Figure 6.26 shows a precision/recall graph for both methods, using the test database. For comparison, the precision/recall of image-to-image matching (picking the single best image of the three available), and the curve of a method that returns random results are included.

The graph shows an improvement in average precision of 18% of the structural matching method over the image matching method. Interestingly, using the ellipse outlines as input for our image matching method produces better classification results than using the more detailed images themselves as queries. This may be explained by the fact that the sets of ellipses better represent the "average shape" of the objects in a class.

Figure 6.28 shows a *similarity matrix* for both methods. Darker squares indicate better similarity scores (lower dissimilarity scores). The brightness value of each pixel in each matrix has been normalized by the average similarity score of the whole

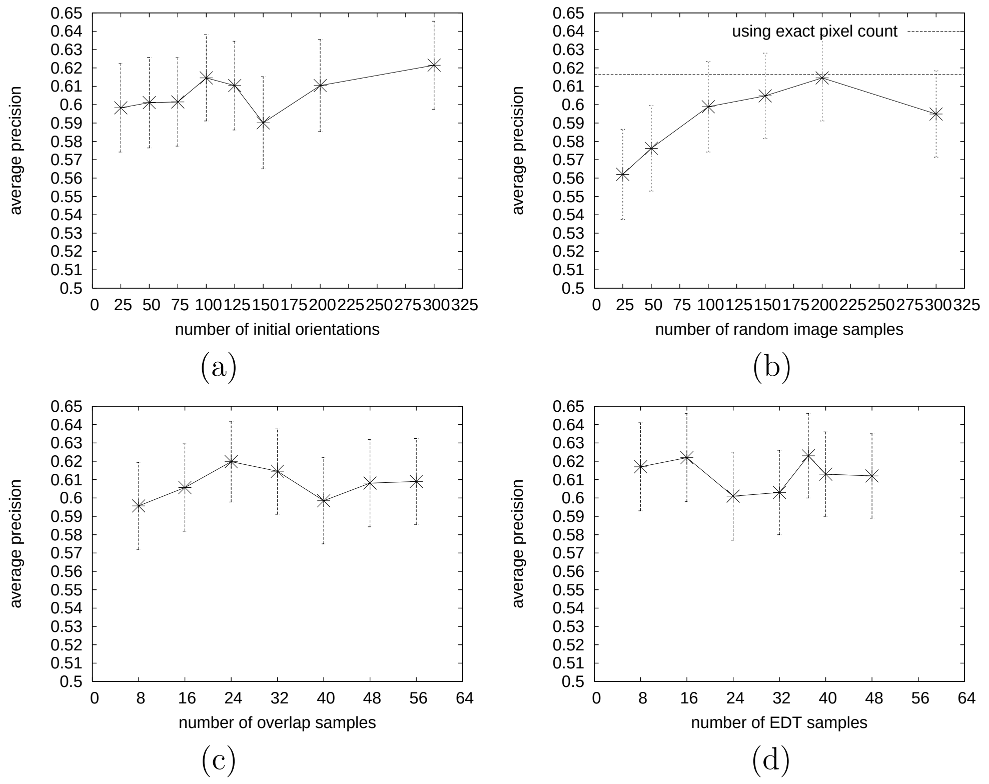


Figure 6.25: Average precision for different values of (a) the number of orientations to try for the initial alignment, (b) the number of random image samples to use for the image overlap term (the horizontal line shows the optimal value, achieved when computing exact image overlap in pixels), (c) the number of boundary samples used to compute the approximate ellipse overlap term, (d) the number of samples along the ellipse's major axis to compute the EDT alignment term

matrix. From these similarity matrices we see that the matrix of the image matching method has higher contrast, reflecting the property of the method that pairs of images usually are rated either very similar or very dissimilar.

Examining the matching results in more detail, we can identify the classes for which the structural matching yields an improvement, and for which it does not. Figure 6.29 shows the same similarity matrices, but now with only three possible markers in each matrix cell: $\boxed{\text{n}}$ corresponds to the "nearest neighbor" match (i.e. the image with the lowest dissimilarity score), ◯ to the next four best matches, and • to the next four. A perfect matching result for a class would result in five $\boxed{\text{n}}$ markers on its diagonal, and the remaining 20 squares having a ◯.

In these matrices we observe the following effects: (1) objects of the same class that are structurally similar but have varying geometric properties are matched better using our structural matching method (compare for example the table, chair,

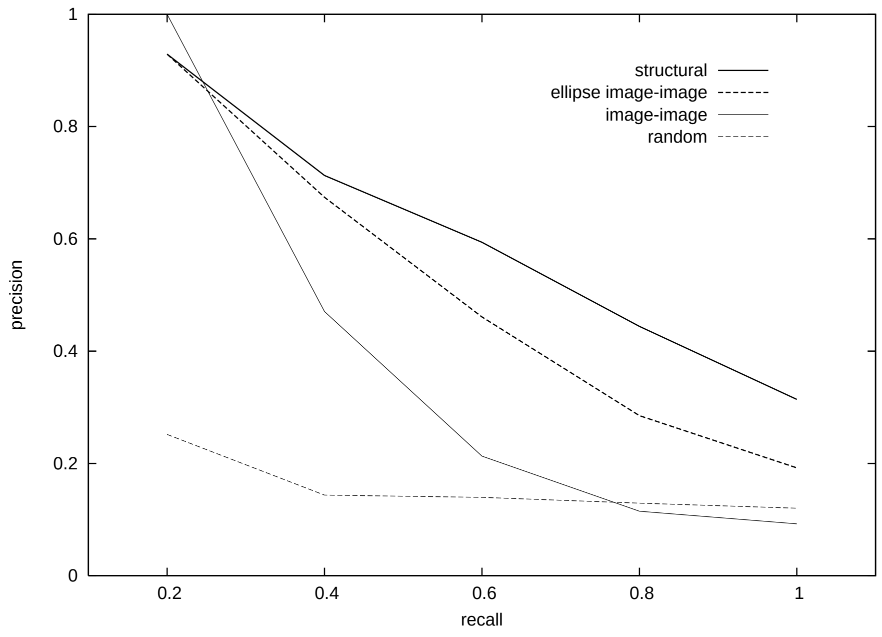


Figure 6.26: Precision/recall for 2D structural matching, 2D outline image matching (of the ellipse outlines to the stored 2D view outlines), 2D image-to-image matching, and random retrieval, for a test database of 15 classes of 5 models each

and helicopter classes), and (2) objects of the same class that look very similar are matched well using our image matching method (see for example the gun and passenger plane classes). Furthermore, the structural matching method returns some reasonable matches across classes, for example men to birds, and fighterjets to passenger planes (see Figure 6.27, these are examples from the training set). We conclude that our structural matching method performs well for the classes of objects for which it was intended, i.e. classes with structurally similar objects and relatively large geometric variations.

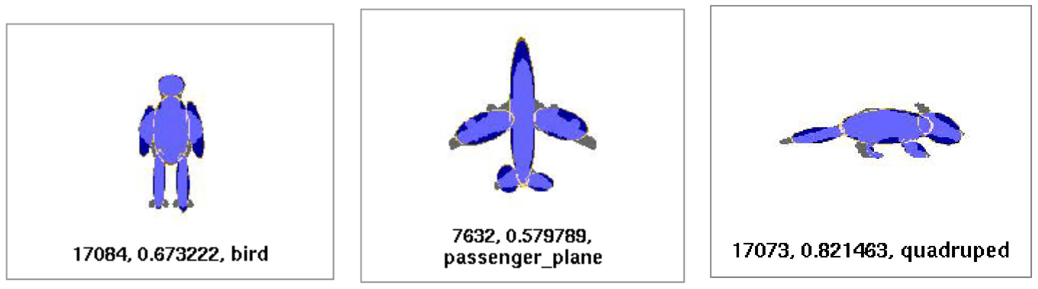


Figure 6.27: Examples of structural matching across classes: (a) a man to a bird, (b) a fighterjet to a passenger plane, (c) a dinosaur to a quadruped

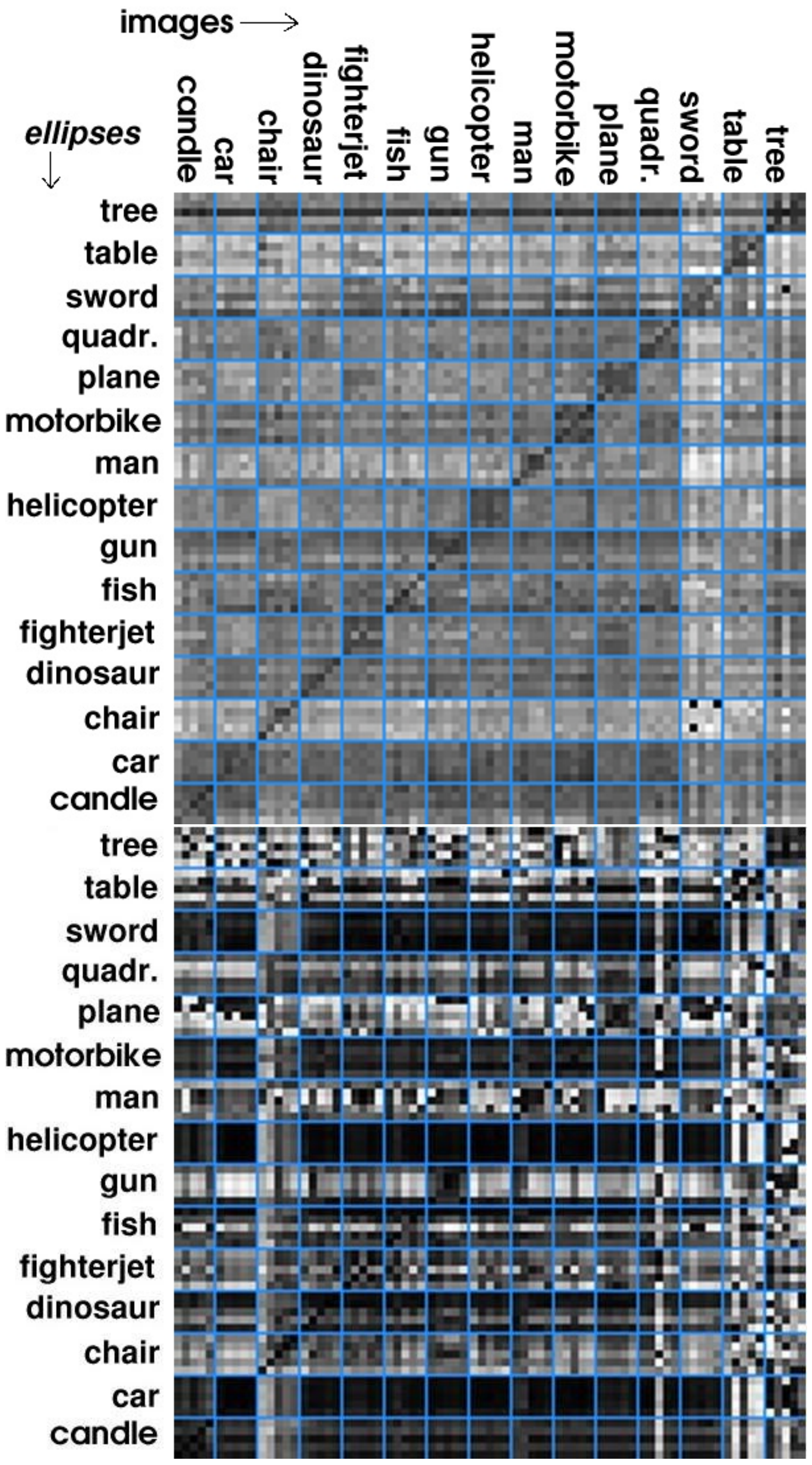


Figure 6.28: Similarity matrix of (top) 2D structural matching and (bottom) 2D image matching. The brightness value of each pixel in each matrix has been normalized by the average similarity score of the whole matrix

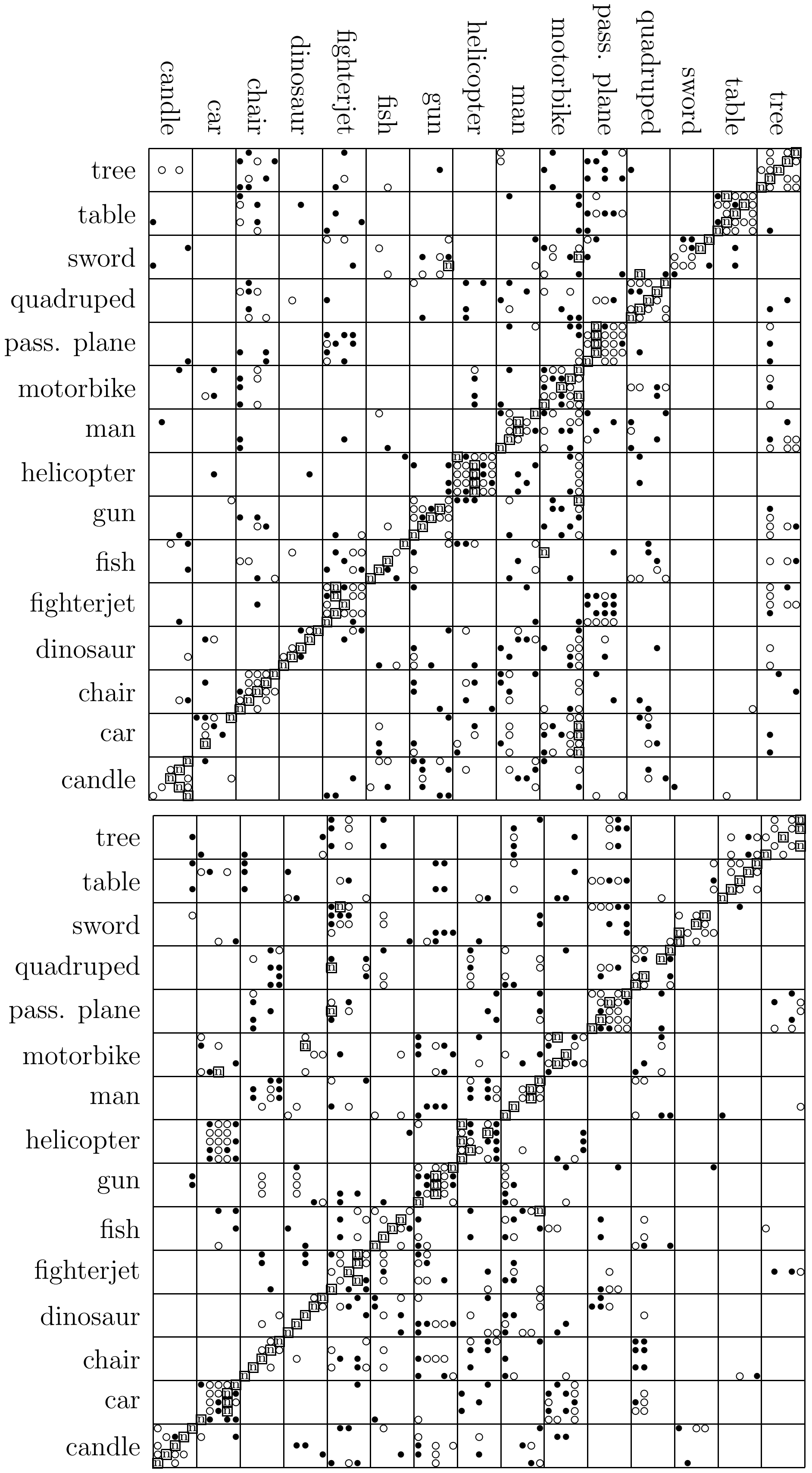


Figure 6.29: Similarity matrix of (top) 2D structural matching and (bottom) 2D image matching. $\boxed{\text{n}}$: nearest match, ○: next four best matches (2-5), •: the next four (6-9)

6.4.4 Timing Results

Currently our structural matching method is not fast enough for interactive applications. It takes about 1 second on average for a single match between a set of ellipses and an image (0.34s on average for 3 ellipses, 0.68s for 4, 1.11s for 5, and 1.8s for 6). The image matching method, on the other hand, takes on average 0.02 seconds per match (all timings were taken on a 2.2 GHz Pentium III with 1 GB memory). To make this approach feasible for our application, we intend to investigate how to reduce the running time without impacting classification performance. Possible approaches are using a coarser representation derived from the 2D view images that contains most essential information for matching structure, and/or combining our approach with a coarse prefiltering step that selects promising candidates quickly.

6.5 Summary and Conclusions

In this chapter we investigated a 2D structural query interface and accompanying matching method for our 3D model search engine. Its design was motivated by the fact that our freeform sketching interface and image matching method require queries to look similar to the 2D projections of a desired 3D model. By allowing the user to specify a structure instead of a sketch, we can attempt to return objects with a similar structure. Also, evidence has been presented in the perception literature that humans think of shape of composed of a set of simpler, basic parts.

In our initial approach, we created a simple drawing interface that supports the drawing and manipulation of ellipses. The accompanying matching method has to match sets of user-drawn ellipses to 2D images. A possible approach is to precompute similar structural descriptors for these 2D images, and then use structural matching. However, it is hard to robustly segment arbitrary images. We can also not be sure that every user will decompose a certain object the same way. In our approach, we match a set of ellipses directly to a 2D image. It is therefore not necessary to precompute structural descriptors and users do not have to provide consistent input. Furthermore, by allowing some part variation (e.g. in size, orientation) it is still possible to match structure.

The actual matching is done by running a non-linear optimization to minimize an error-of-fit objective function that rewards image overlap and alignment of the ellipses with the Medial Axis, and penalizes ellipses overlapping other ellipses and shape deformation. The classification performance of this approach was evaluated using two test databases containing 15 classes of 5 objects each. The first was used as a training set to optimize the weights of the objective function. The second was used to compare the classification performance with that of the image matching method of the previous chapter. We found that the classification performance improves significantly for classes of objects that have a similar structure but varying geometric properties, for example, tables, chairs, and motorbikes. Other classes of objects show no improvement, either because the objects look sufficiently similar for an image matching method to be effective (e.g. passenger planes), or because the structural query was too “generic”, and ended up matching many different objects (e.g. guns, fish).

A drawback of the structural matching method is its longer running time, currently about 50 times higher than our image matching method for a single pairwise match. In future work, we intend to investigate how to reduce this running time without impacting classification performance, for example by combining it with a coarse prefiltering step that selects promising candidates quickly.

In other future work, we intend to design a 3D version of this interface, in which the user draws ellipsoids. Because of the much higher cost of matching sets of ellipsoids to 3D models, we plan to investigate if our fitting approach may be used as a segmentation method, so each database model can be represented as a set of parts. A user study is required that tests to what extent human shape decompositions are consistent. Other user studies need to be run to evaluate the ease of use and effectiveness of this structural matching method in practice.

Chapter 7

Comparison of Matching Methods

7.1 Introduction

In this chapter, we evaluate the classification performance of three matching methods described in this thesis: text, 3D shape, and 2D outline view matching, as well as their combinations. We used a test database of 1,000 models, classified into 81 categories, which is described in more detail in the next section. This database is a subset of our model database of 31,000 models downloaded from the web, and as such is representative of the kind of data we have to deal with.

From testing the individual methods, we found that both the 3D shape matching method outperforms the text matching method. This is mostly due to the insufficient annotation of 3D models on the web. The 2D matching method showed the worst performance. Combining the text- and 3D shape-based matching methods by simply computing a weighted linear combination of their matching scores further improved classification performance.

The next section describes the creation and contents of the test database used for these experiments. Section 7.3 has the results of the individual method comparison. Results for combinations are in Section 7.4, followed by a summary and conclusions.

7.2 Test Database

It is important to be able to compare the performance of different classification methods. For this purpose, a good test database is essential. This database should be large, and have a sufficient number of classes containing many different types of objects (e.g. differing in complexity, genus, man-made vs. organic, and so on). Both small classes (e.g. four objects) and large classes (e.g. fifty objects) should be featured. No class should dominate the database because of its size.

Unfortunately, such a test database is not yet available for 3D shape matching. Databases that have been used in previous work are different in every paper. Some of these databases are dominated by certain types of shapes, creating a strong bias in the results. The 3D model database created by Osada *et al.* has been used by other researchers, but is far too small (only 133 models) [72]. For these reasons we decided to create a new test database, using our database of 31,000 3D models retrieved from the web as the source. The eventual goal is to publish this database and other, larger ones, enabling more accurate performance comparisons between different 3D model matching methods.

7.2.1 Creating the Database

Here we briefly describe the procedure followed to create our test database. To avoid having to manually identify near-duplicates, we clustered the models using our 3D shape similary metric as a distance measure. Both the similarity metric and clustering method were developed by Michael Kazhdan [54]. This resulted in 15,990 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.

Next, an undergraduate student, David Bengali, created an initial classification of these models, 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. With the help of a graduate student, Phil Shilane, we further refined the classification, resulting in a set of approximately 5,000 models. From this set we selected a subset of 1,000 models, subdivided into 81 classes. 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. Finally, the classes were placed in a hierarchy, using as much as possible the superclass names provided by Wordnet (a lexical database) [68]. The hierarchy information will be used in future work for evaluating matching methods that work at a higher level than the "single-class" level.

7.2.2 Result

Table 7.1 lists all 81 classes, and their place in the hierarchy. The actual classes are printed in boldface. The righthand column shows the number of models in each class.

Figure 7.1 shows a thumbnail image of a representative member of each of the 81 classes.

Figure 7.2 shows a histogram of all the class sizes. The seven smallest classes have 4 models, the largest class has 73 (humans). The median class size is 8, the average is 12.3. 39 classes (48%) have 4-7 models, 24 (30%) classes have 8-13 models, and 18 classes (22%) have more than 13 models.

1	animal	arthropod	insect	**ant**		5
2				**butterfly**		6
3			**spider**			9
4		**bird**				7
5			**duck**			5
6		**fish**				7
7			**dolphin**			4
8			**shark**			7
9		mammal	carnivore	canine	**dog**	4
10			hoofed	**bovine**		5
11				swine	**pig**	4
12			primate	**human**		73
13					**arms_out**	35
14					**walking**	8
15		reptile	dinosaur	**trex**		5
16			**sea_turtle**			5
17	body_part	**brain**				6
18		**face**				32
19		**hand**				14
20		**head**				29
21		**skull**				7
22	building	**castle**				8
23		**church**				4
24	container	**bottle**				11
25		**glass_with_stem**				6
26		**mailbox**				7
27		**tank**				5
28		**vase**				18
29	device	display	**computer_monitor**			12
30			**tv**			12
31		electrical_circuit	**microchip**			7
32		instrument	**eyeglasses**			7
33			**microscope**			5
34			timepiece	**hourglass**		6
35				**watch**		4
36		lamp	**desk_lamp**			10
37			**street_light**			7
38		musical_instrument	**electric_guitar**			10
39			**piano**			5
40		weapon	firearm	**handgun**		15
41			**sword**			18
42		wheel	**gear**			12
43			**wheel**			8
44	food	dessert	**ice_cream**			6
45	furniture	**school_desk**				5
46		seat	**bench**			7
47			chair	**desk_chair**		15
48				**dining_chair**		22
49				**patio_chair**		5
50			**couch**			13
51		**shelves**				25
52		table	**rectangular**			53
53			**round**			12
54				**single_leg**		10
55	plant	**bush**				9
56		**flower_with_stem**				13
57		**potted_plant**				46
58	tool	**shovel**				5
59	vehicle	aircraft	airplane	**biplane**		13
60				**commercial**		12
61				**fighter_jet**		50
62				**glider**		11
63				**multi_fuselage**		5
64				**stealth_bomber**		9
65			**helicopter**			32
66			**hot_air_balloon**			8
67			spaceship	**enterprise_like**		21
68				**satellite**		5
69				**space_shuttle**		5
70				**tie_fighter**		6
71				**x_wing**		4
72		**military_tank**				10
73		vessel	**ship**			10
74			**submarine**			8
75		wheeled	cycle	**bicycle**		7
76				**motorcycle**		5
77			**jeep**			4
78			**pickup_truck**			7
79			**race_car**			13
80			**sedan**			16
81			**sports_car**			19

Table 7.1: The 81 classes in our test database of 1,000 3D web models, and the hierarchy in which they are organized. The actual classes are printed in boldface. The right column shows the number of models in each class

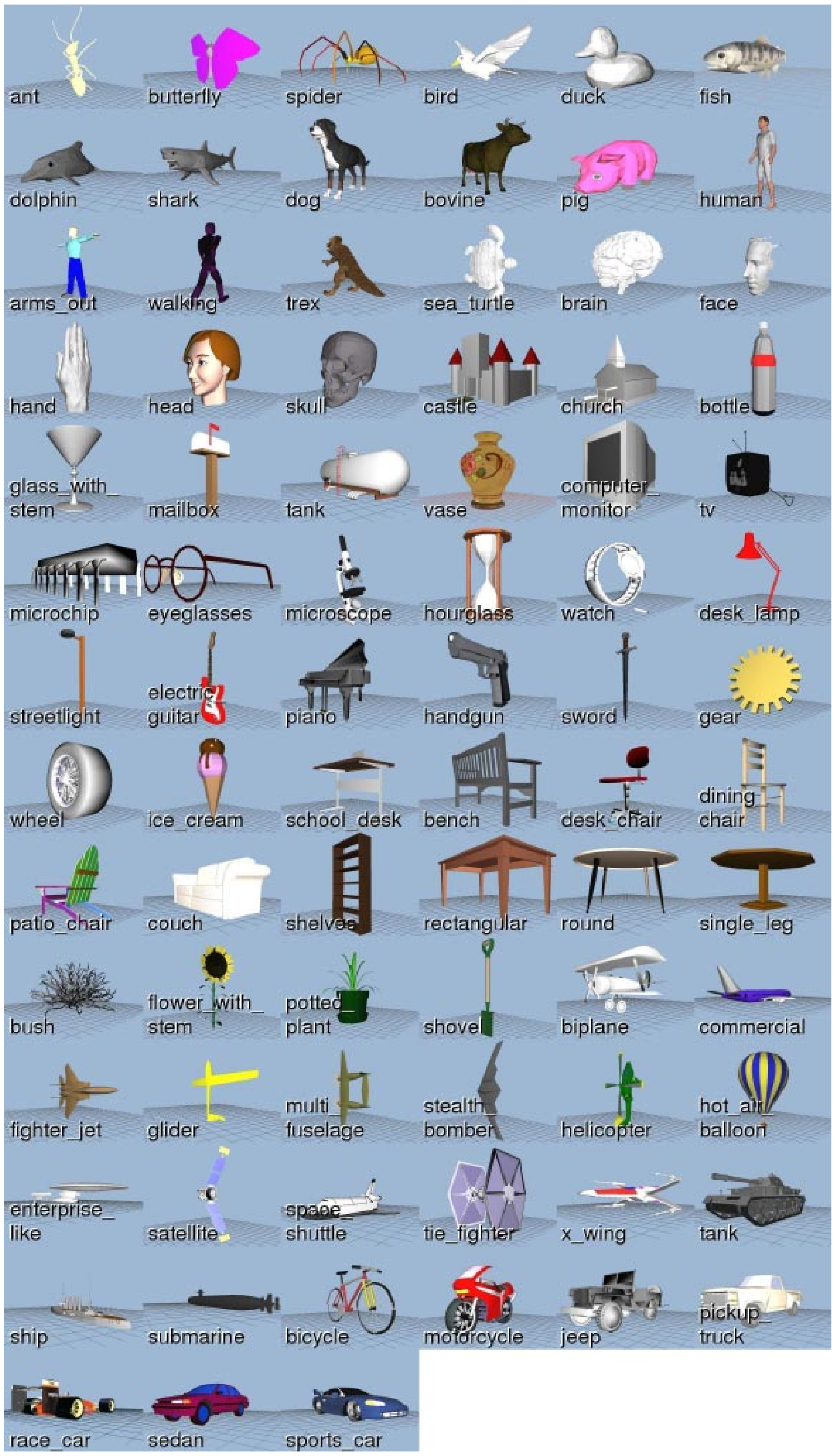


Figure 7.1: A thumbnail image of a representative member of each of the 81 classes

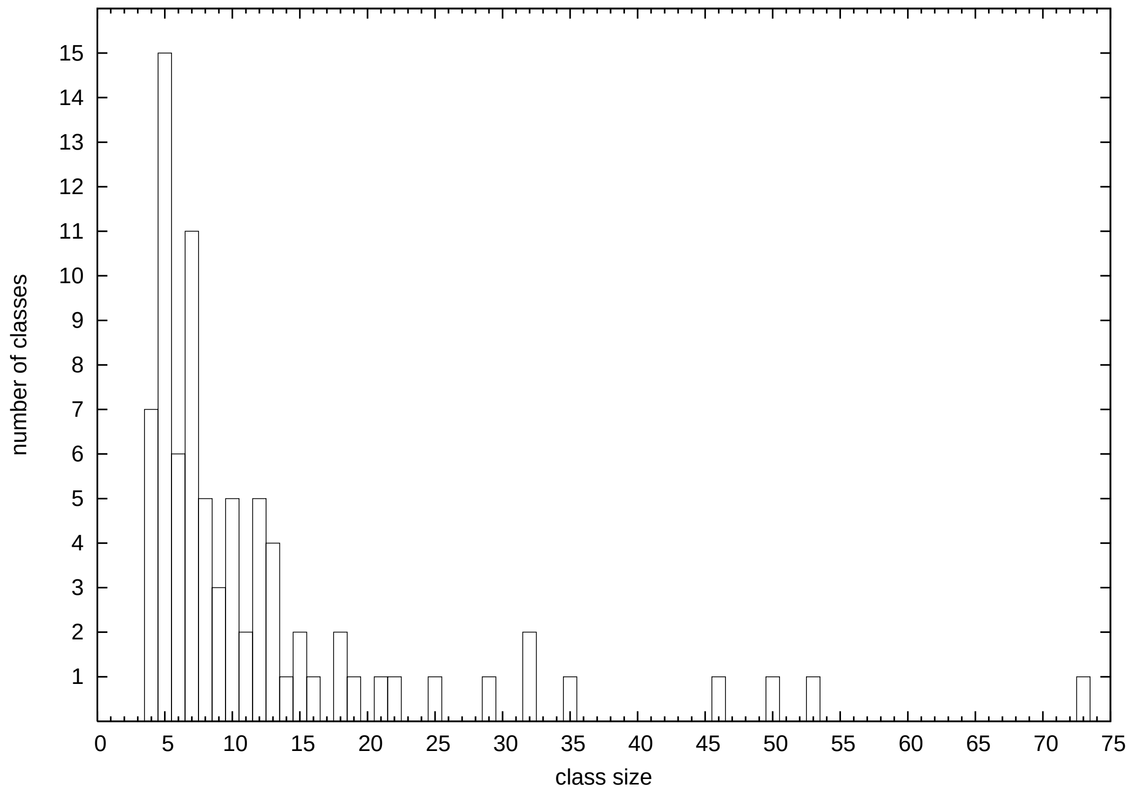


Figure 7.2: A histogram of the class sizes of the 81 classes in our test database

7.3 Comparison of Individual Methods

First we examine the matching performance of each matching method individually. Figure 7.3 shows the average precision/recall achieved by the text, 3D shape, and 2D shape matching methods (see Appendix A for an explanation of the precision/recall performance metric). For the 2D matching, each 3D model was represented by seven outline views (the side and corner views, also see Section 5.5.1). The results of two 2D matching variations are shown: one for which the best single side view query was picked for each model (simulating a user who draws the most discriminating side view for each model), and one for which the best three views out of the full set of seven available views was used. Using more than three views did not improve the results. Note how the 3D shape matching methods outperforms the text matching method, signifying the poor quality of text annotation of 3D models on the web. The 2D sketch matching method performs even worse, showing that even if the user would be able to supply three accurate discriminating sketches of a model, it still likely would not be worth the effort.

7.4 Comparison of Combined Methods

Next, we investigated if these matching methods can be combined to improve performance. To combine the matching scores of two methods, we first mean-normalized the scores of each individual model, and then computed a weighted average of the resulting values: i.e. if s_1 and s_2 are the mean-normalized matching scores of a pair of models, then the combined score is $w \cdot s_1 + (1-w) \cdot s_2$, 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. We found that the combination text+3D resulted in higher average precision than either method alone. This did not happen for the combinations text+2D and 2D+3D. Figure 7.4 shows the resulting precision/recall curve of the combination text+3D, compared to the curves of the individual methods. The optimal weight setting was $(0.15 \cdot text + 0.85 \cdot 3D)$.

These results suggest that to some extent the text and 3D shape model representations each contain information unique to its domain, such that when they are combined, they become more discriminating. Also, there may well be other representations (e.g. appearance-based) that capture a very different aspect of a 3D model, and as such can improve classification performance even further when combined with our current classifiers. This is an interesting area for future research.

7.5 Summary and Conclusions

In this chapter we compared the performance of the matching methods we use for our search engine.

From classification experiments (using a test database of 1,000 models, representative of our entire database of 31,000 models downloaded from the web), we found that 3D shape-based matching outperform text-based matching. The main reason is the poor quality of text annotation of web models (also see the summary of the chapter on text queries, in Section 3.5). Text matching still outperformed our 2D sketch matching method.

Combining the text- and 3D shape-based matching methods further increased the average precision. This suggests that it is a good strategy to combine different types of matching methods. In future work, we intend to experiment with different classifier combination functions, and investigate if more information can be derived from 3D models, to further improve the discriminating power of combined classifers.

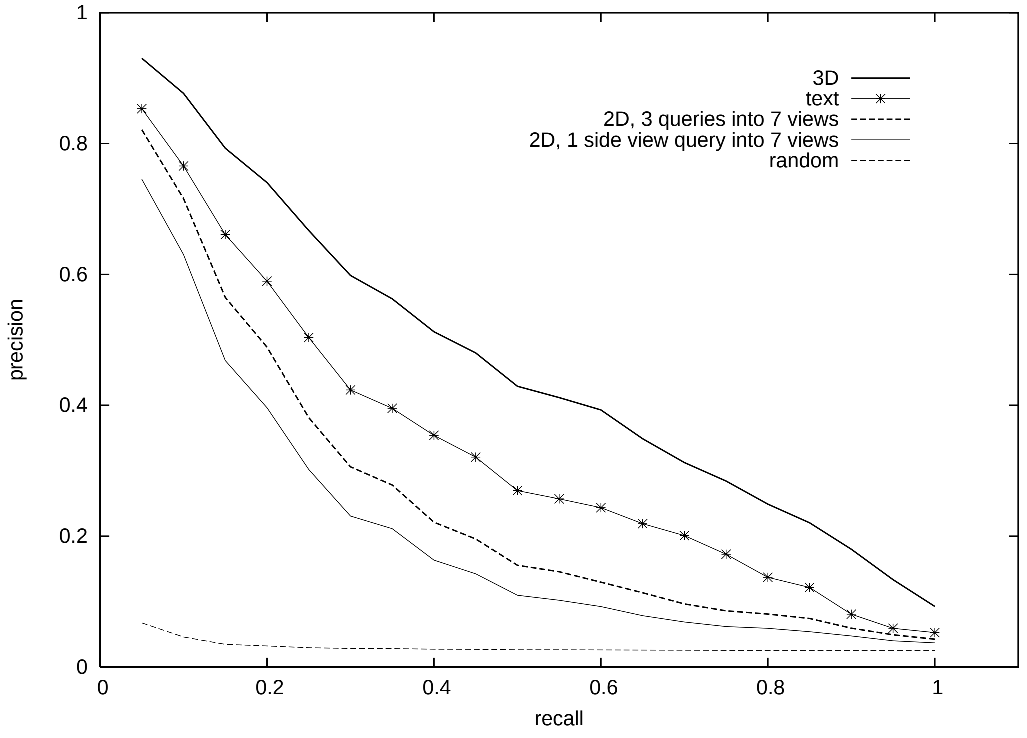


Figure 7.3: Average precision/recall for text, 3D shape, and 2D shape matching

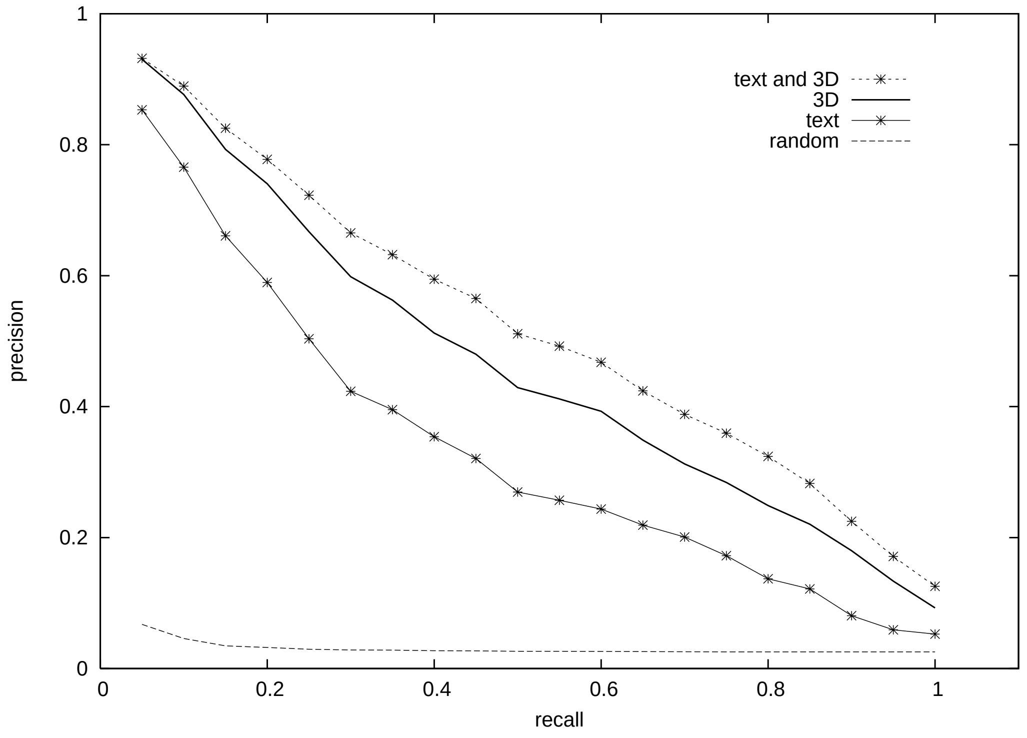


Figure 7.4: Average precision/recall for the combination text+3D, and either method by itself

Chapter 8

Search Engine Usage Results

8.1 Introduction

In this chapter, we present several statistics about the usage of our search engine in practice. To enable evaluation, every query submitted to our site is logged. The logged data includes the query contents, client IP address, query processing time, search results, and so on. We also log whether a user opens an information window, a referring page is visited, and/or a model is downloaded. Note that local queries (from the `princeton.edu` domain) were not included in any results. Usage results over a shorter period of about 11 months were presented in Min *et al.* [69].

Of the 294,523 queries processed in an 18 month period, about 30% involved a shape-based query interface, the remaining 70% were text queries. The most popular shape-based query was the "Find Similar Shape" link. This query type and the text query type resulted in the most user interest in the search results, measured as the percentage of searches that resulted in at least one model download (about 15%). The 3D and 2D sketch query interfaces resulted in far fewer model downloads, except when they were used in combination with text keywords.

Measurements of the processing time of each query shows that most queries are processed in less than a second (using a 1.5 GHz Pentium III processor). Queries that involve the computation of a 3D shape signature (3D sketch, 3D file upload) take up to 5 seconds.

Our search engine currently processes queries from about 1,000 different hosts each week. About 25% of the visitors on each day are returning users.

The remainder of this chapter is organized as follows. The next section has overall usage results, followed by sections with more detailed results on the text, 3D shape, 2D shape, and combined query interfaces respectively. Section 8.7 presents the results of our query processing performance tests. Section 8.8 has details about the number of visitors over time, and where most of them are from. A short summary and conclusions can be found in Section 8.9.

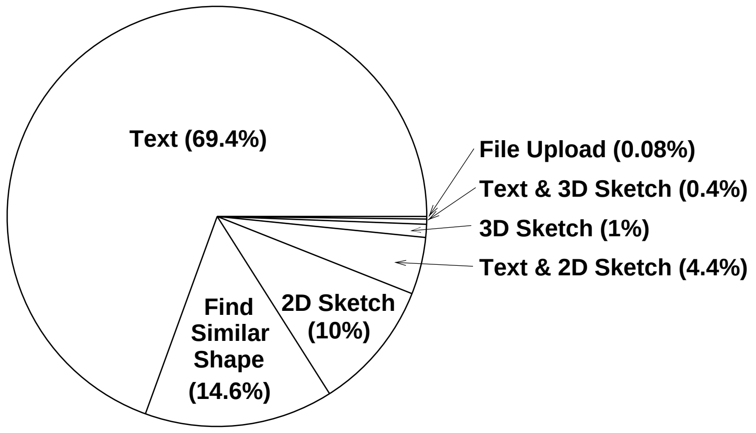


Figure 8.1: A pie chart of the relative use of each query interface

8.2 Overall Usage

During an 18 month period starting on November 18, 2001, 311,909 queries have been processed. 294,523 of these resulted in an actual search being performed (15,556 queries were empty, i.e. searches with no text or shape input, and 1,830 file upload queries failed). Figure 8.1 shows the relative use of each individual query interface as a pie chart. 69.4% of all queries were text-only searches, while the remaining 30.5% were shape-based, possibly combined with text. Table 8.1 shows the number of searches performed for each query interface type, and what percentage resulted in, at least once, (1) the opening of an information window, (2) the visiting of a referring page, and (3) the downloading of an actual 3D model. We hypothesize that the last three numbers are an indication of user interest in the search results. For example, if one query resulted in more model downloads than another, we think that the search results were likely to have been more relevant to the user.

	# searches (%)	% info window	% ref. page	% model download
text	204,512 (69.4)	35.1	9.3	13.5
find similar shape	43,032 (14.6)	40.0	11.7	15.3
2D sketch	29,535 (10.0)	19.8	4.1	2.4
text & 2D sketch	12,980 (4.4)	28.0	6.9	8.0
3D sketch	2,883 (1.0)	17.8	2.6	3.0
text & 3D sketch	1,152 (0.4)	31.5	9.0	10.5
file upload	234 (0.08)	32.1	8.5	11.5

Table 8.1: Relative use of each query interface, and user interest in the search results

keywords	number
car, truck, vehicle	8,829
woman, female, girl	4,765
house, building, architecture	3,876
table, chair, sofa, furniture	3,812
human, people, person	3,722
animal, dog, horse, cat	3,363
tree, plant	3,277
ship, boat	2,569
plane, airplane, aircraft	2,267

Table 8.2: Some of the most popular query terms, grouped into categories

8.3 Text Query

The text-only query interface was the most popular, accounting for almost 70% of all queries. This may be explained by the following reasons: (1) it is the most familiar kind of interface on the web, (2) it is easy to supply a text query, (3) the total time from specifying the query to seeing the results is very short, and (4) users are unfamiliar with shape queries.

The 204,512 text queries contained 33,427 unique queries. Table 8.2 shows the most popular query words grouped into categories, and the number of times any of these words was submitted.

15,289 text queries (or 7.5%) resulted in zero matches: these included queries that were misspelled or in a foreign language, but also queries for objects not in our database, e.g. "audi," "cellphone," and "spiderman." These also included queries for objects that *are* present in our database but are annotated incorrectly (for example, the database contains buddha models but their filename is misspelled). In these cases, a shape-based query may have been more effective.

8.4 3D Shape Queries

The most popular shape query type was the "Find Similar Shape" search, in which a result from a previous search is submitted as a new query. It accounted for about half of all shape-based queries.

In 86% of all cases, the "Find Similar Shape" search was preceded by a text search. Combined with the fact that this query type generated the most user interest (e.g. model downloads) in the results, this suggests that many users start with a text search, and are able to use the "Find Similar Shape" query to home in on a desired model.

Few people used the other 3D shape-based interfaces. About 1.4% of all the queries involved the 3D sketching interface, and the sketches that were submitted are of very low quality. To use the 3D sketching tool effectively, users should take a small tutorial, and have some 3D manipulation skills. We think this was too much of a barrier for the average user. An evaluation of 1,000 3D sketches submitted in the first half of 2002 was presented in Section 4.5.2. It showed that most of the models created with Teddy are unidentifiable blobs. As a result, very few users subsequently show interest in the search results (for example, only 3% of the queries was followed by a model download, compared to 13-15% for other query types). Also, of the 1,653 different hosts (i.e. unique IP addresses) that used the 3D sketch interface in a period of 1.5 years, only 78 (about 5%) used it more than once.

Even fewer people used the "upload 3D model file" feature (it is the least used query type, with 234 queries in 18 months (0.08%)). Most of the time it is misunderstood, and used to enter text keywords, or upload image files.

8.5 2D Shape Queries

10% of all searches included one or more 2D sketches, with most users preferring three sketches (50%) over a single sketch (41%) and two sketches (9%). As was discussed in Section 5.5.3, because of the low quality of the submitted sketches, and the matching methods's sensitivity to extra interior detail, the search results were less useful than for the other query types.

We also investigated what number of sketches (1, 2, or 3) resulted in the most user interest in the matching results. The number of sketches was directly related to user interest: of all single sketch queries, 17% resulted in the opening of an information window. For two sketch queries this was 20%, for three sketches 24%. This suggests that, in spite of the general low quality of the user-drawn sketches, users are able to improve their queries by using multiple sketches. Alternatively, the increased user interest may be explained because the user spent more time and effort creating the query.

8.6 Combined Queries

Both the 2D and 3D sketch queries can be combined with text. These combined queries were not used very often (4.4% and 0.4% of all queries, respectively), but they did generate more user interest in the search results when compared to their sketch-only counterparts (for example, 8% and 11% searches followed by at least one model download, compared to 2.4% and 3%). Given the poor performance of the sketch interfaces by themselves, we think that the improved quality of the results is mostly due to the added text keywords.

8.7 Query Processing Performance

Here we report the response times of our site for the various query types and show that most queries are typically satisfied in less than a second (not counting network transmission times, and local processing times).

The response time a user experiences is the sum of the time it takes for the following operations: (1) connecting to our web server and sending query data, (2) executing a CGI script on the web server (which connects to and has to wait for the matching server), (3) processing and matching of the query on the matching server, (4) returning the results to the user, and (5) rendering the results web page on the user's machine. The time taken in steps (1) and (4) depends on the available bandwidth between the user's machine and our web server, step (5) depends on the performance of the user's machine. Step (2) adds an estimated overhead of about 1–2 seconds. We can report accurate timings for step (3): processing and matching of the query on the matching server.

Table 8.3 shows for each query type the average time used for processing and matching on the matching server (a dual 1.5 GHz Pentium III running Red Hat Linux 7.2), averaged over about 16,000 searches.

These numbers show that the response time is mostly determined by the amount of query data that needs to be processed. Text and "Find Similar Shape" queries take the least time, followed by queries that involve 2D sketch(es) (for which 2D image(s) are converted to image descriptors), and queries that require the conversion of 3D models (3D sketch and file upload). In the latter case, the 3D model file size dominates the total processing time. The conversion times for a few example model sizes are: 60 KB, 2,000 triangles: 4 seconds, 800 KB, 6,500 triangles: 6 seconds, 2 MB, 61,000 triangles: 13 seconds (2 MB is currently the file size limit on the "file upload" feature).

The time between when a query arrives at the web server and when its results are ready is about 0.4 seconds on average. To investigate how this average time increases under increasing load, we ran two experiments in which we sent queries to our web server at a higher rate. These queries were taken from a set of 4,000 representative

query type	processing and matching time (in sec)
text	0.22
2D sketch	0.61
text & 2D sketch	0.59
3D sketch	3.2
text & 3D sketch	3.2
Find Similar Shape	0.36
file upload	5.0

Table 8.3: Average time for processing and matching for each query type

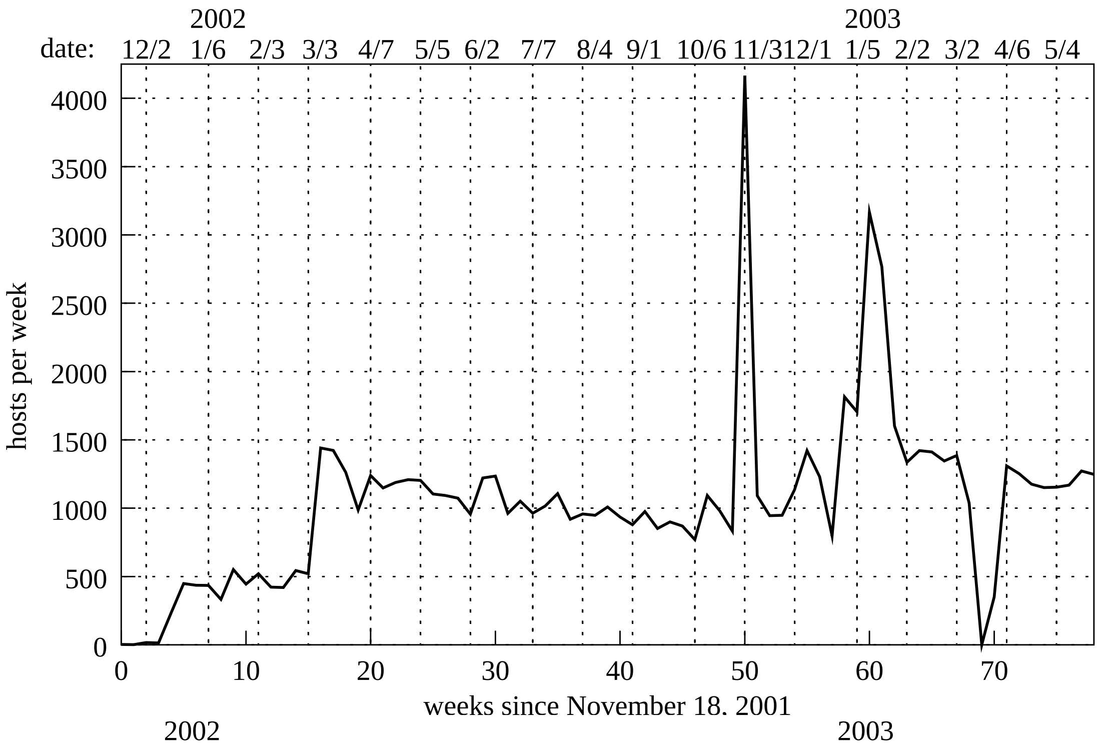
queries submitted to our search engine, logged during a one week period in July 2002. In the first test we sent on average one query every 4 seconds for a period of two hours from a single host. In the second test we sent twice the number of queries from 4 different hosts (which is a more than 80 times higher rate than the currently typical rate of about 600 queries per day). The average processing time per query increased to 0.88 and 1.46 seconds per query, respectively (the maximum times were 5 and 7 seconds), which shows that our current system will be able to accomodate much higher loads without incurring a significant performance penalty (i.e. one where the turnaround time becomes unacceptable for the user).

8.8 Visitors

To investigate whether people find the search engine useful, we measured the number of unique hosts using our site per week, and what percentage of them return after a first visit. Figure 8.2 (top) shows the number of unique hosts per week using our site since the first hit arrived on November 18, 2001. Note that hits from `*.princeton.edu` were not counted. During each visit on average 3.2 searches were performed. The sharp increase in the week starting on March 10th was due to the improved ranking of the site at Yahoo and Google. The peak in the week of November 3 occurred because our site was mentioned on Slashdot, a popular online discussion board for technical news. As a result, during the first six hours after the post over 3,800 queries were processed. The second peak was caused by a mention on Memepool, another online discussion board. The recent drop to zero was caused by a departmental network outage.

Figure 8.2 (bottom) shows the percentage of returning hosts per week. A visitor is counted as returning if a subsequent visit occurred on a later date than a previous visit. It shows a clear upward trend, and currently approaches 30%, suggesting that an increasing number of people are using the site as a useful resource.

We were also interested in how widespread the usage of our search engine is. For this purpose we store the hostname of each visitor's PC and count the number of unique hostnames from each top-level domain. We received queries from 129 different top-level domains, 112 of which were from specific countries. Table 8.4 shows the number of unique hosts visiting from the 20 most frequent top-level domains. Note that these numbers only give a rough indication of the relative use of our site in each country: the `.net` and `.com` domains are international, for 25% of the searches a hostname lookup failed, and dynamic IP addresses may cause the same host to be counted more than once. The numbers do indicate that the usage of our site is widespread, and highest among industrialized nations.



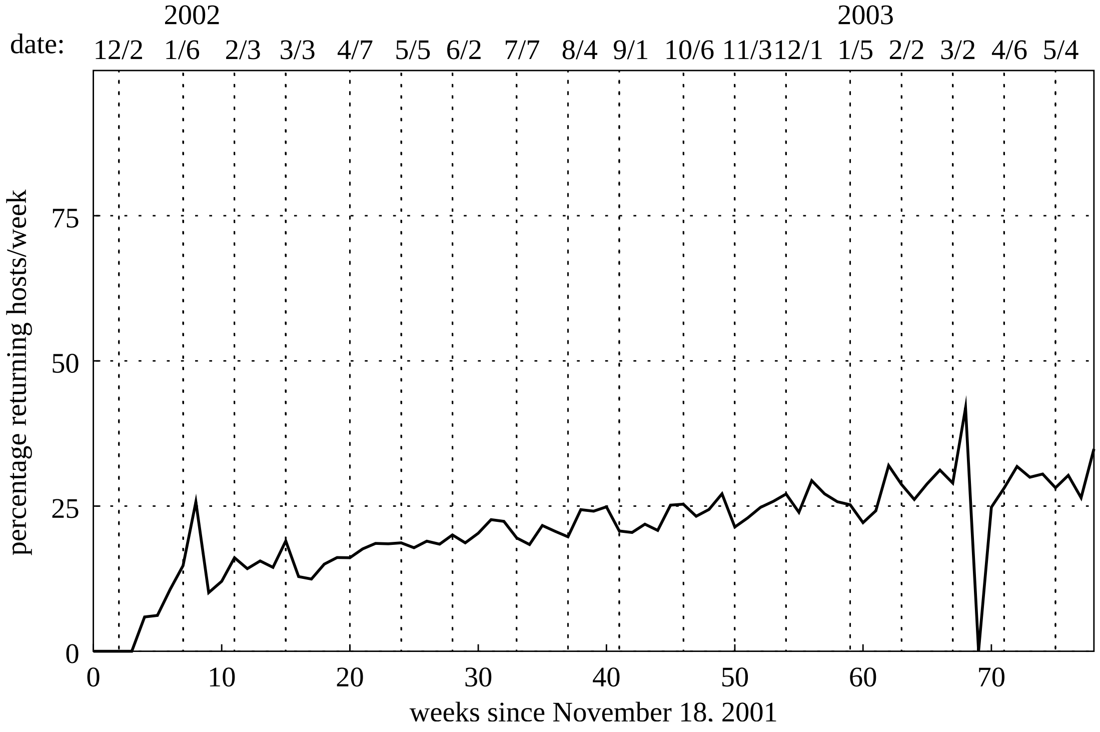


Figure 8.2: The number of unique hosts using the search engine per week (top) and the percentage of returning hosts per week (bottom)

	domain	hosts		domain	hosts
1	net	15,666	11	au (Australia)	987
2	com	11,214	12	jp (Japan)	824
3	fr (France)	3,093	13	es (Spain)	790
4	edu	2,145	14	be (Belgium)	489
5	ca (Canada)	1,402	15	mx (Mexico)	486
6	de (Germany)	1,349	16	pl (Poland)	437
7	nl (Netherlands)	1,335	17	ch (Switzerland)	420
8	it (Italy)	1,242	18	at (Austria)	411
9	br (Brazil)	1,105	19	ar (Argentina)	359
10	uk (UK)	1,064	20	hu (Hungary)	322

Table 8.4: Number of different hosts from the 20 most frequent top-level domains

8.9 Summary and Conclusions

In this chapter we presented results on the actual usage of the search engine site.

From analyzing the usage logs of the site, we see that the search engine has a steady number of visitors, processing about 4,000 searches per week from about 1,200 different hosts, with a current (May 2003) total of almost 300,000 processed queries. Examining the relative usage of each query type, we think that ease of use and familiarity of a query interface determine its popularity: text is most popular (familiar and easy to use), followed by "Find Similar Shape" (less familiar, but intuitive and easy to use), 2D sketch (familiar, but harder to use), and finally the 3D shape based interfaces (unfamiliar, and relatively hard to use).

A problem which affects the usefulness of the site is the slow increase of the number of "dead links" (about 15% of our database when measured in February 2003), i.e. links to referring pages or models which have become invalid. In the future, we plan to provide cached copies of the models which are no longer available through their original link (similar to the Google cache of web pages).

Chapter 9

Conclusions and Future Work

This thesis examines the issues that arise when creating an online search engine for 3D models, focusing on those related to query interfaces. The main research contributions are:

1. new query interfaces for searching 3D models based on text keywords, 2D and 3D shape, and the combinations text+2D and text+3D

2. a 2D structural matching method based on fitting a tree of ellipses to a 2D image, by optimizing an error-of-fit objective function. One term of the objective function computes an error-of-fit with the image's Medial Axis, benefiting from the Medial Axis's structural significance, while circumventing its noise sensitivity problem

3. evaluations of matching methods, showing that (1) 3D shape matching outperforms text matching, mainly due to the poor quality of annotation of web models, (2) combining 3D shape and text descriptors improves matching performance, (3) both 3D shape matching and text matching outperform a shape matching method based on matching multiple 2D projections of a 3D model, and (4) certain classes of objects are better classified using our 2D structural matching method, however the method is still too slow for interactive applications

4. results of a user study suggesting that shape combined with text can improve the effectiveness of a query

5. a prototype online 3D model search engine which incorporates these query interfaces, and which has been used extensively across the world. For example, in 18 months we processed over 294,000 queries from 63,574 different hosts in 112 countries, resulting in 53,192 model downloads. Furthermore, currently 20–25% of the about 1,000 visitors per week are returning users

6. usage results from our search engine, showing that (1) most people prefer the simplest query interfaces (text keywords, find similar shape), and (2) both 2D and 3D sketches are mostly of low quality, and as such produce inferior matching results

7. a to-be published test database for evaluating 3D model matching, containing 1,000 models classified into 81 categories

There are many directions for future work. Some of the options are described below, organized by whether they apply to query interfaces, matching methods, or new applications altogether.

Query Interfaces

First, the existing query interfaces can be improved. As was noted in Section 5.5.3, interior detail in the user sketches greatly affects the 2D shape signature, and as a consequence the matching results. Improvement can either be sought in increased guiding of the user input, post-processing of user sketches, and/or incorporating the interior detail when matching (by, for example, storing 2D views of 3D models *with* interior lines, e.g. by marking edges of high concavity).

The 3D sketching interface could also be improved with increased guidance. For our application, perhaps a parts-based interface (e.g. drawing ellipsoids) is more useful than a freeform interface like Teddy. User studies are required that test to what extent human shape decompositions are consistent, and that test the ease of use and effectiveness of various query interfaces.

It will be interesting to consider different methods for specifying shape-based queries. For instance, the following constraint-based description might be used to retrieve 3D models of a chair: "give me objects consisting of a box-shaped seat with four equal length and nearly cylindrical legs attached to the bottom side of each corner and a box-shaped back above the seat with width matching that of the seat, etc." This approach captures parameterized classes of objects with a compact description [28, 108].

Matching Methods

There are still other attributes of 3D models that may be queried, for example color, texture, or structure, and for which new matching methods and query interfaces will have to be developed. Furthermore, as we saw in Section 7.4, combining the results of text and 3D shape matching methods can improve matching results. There may be better ways of combining matching methods than simply mean-normalizing and averaging individual matching scores.

We believe there is still room for improvement in the derivation of structure (segmentation) of a 3D model, and its 2D projections (views). Matching user input 2D structure directly to 2D views currently is too slow for interactive applications (see Section 6.4.4), but matching (simple) structures can be done very quickly. Maybe a number of suitable templates can be designed by hand (as in [4]), which are then matched to database models in preprocessing, to determine their class membership and individual parameter settings (i.e. deviation from the average). Furthermore, instead of designing the tempates by hand, these could maybe be computed from a suitable training set [22].

Other types of shape matching problems should also be considered, for example partial shape matching, which can be useful when searching objects in a scene (e.g. a car in a city, a chair in a room). For example, a simple adjustment to the objective function of our 2D structural matching method may allow sets of parts to match part of a 2D image.

It will also be interesting to see to what extent the existing query interfaces and matching methods can be used, or how they should be modified, for application areas with more limited shape domains, such as molecular biology (e.g. searching for protein antibodies), mechanical CAD, and medicine.

New Applications

Future 3D modeling systems should consider integrating shape-based matching and retrieval methods into interactive sketching tools. For example, imagine a user interface in which a user draws a rough sketch of a desired model, after which the system cleans up the sketch, suggests alternatives for parts, etc., all using a large underlying database of models and/or parts, and a fast shape matching method. In this example shape matching and retrieval methods are integrated into a modeling system, instead of being a separate tool like our 3D model search engine. The key benefit in both systems is the re-use of (parts of) existing models.

To conclude, our prototype 3D model search engine has proven to be a flexible test bed for several diverse matching and querying methods. We believe that we have merely scratched the surface in this area of research, and that many interesting and exciting directions still lie ahead.

Appendix A

Precision/Recall

A.1 Introduction

This appendix discusses the precision/recall performance metric, used throughout this thesis.

A common metric for evaluating classification experiments is the so-called *precision/recall* (or *p/r*) plot, which is produced as follows. A database of objects is classified beforehand. The precision/recall of a single object is computed by first matching it to all objects in the database (possibly including itself), which are then ranked according to their similarity score. This object is a member of a certain class with c members. We can now distinguish *recall* values $1/c, 2/c, ..., c/c$, corresponding to $1, 2, ..., c$ retrieved relevant results from the same class. Given a certain recall value, for example $3/c$, we can find the rank k of the 3rd object of this class in the results. The *precision* is then equal to $3/k$, in other words the number of relevant results divided by the number of results it took to get this number of relevant results.

In formula form, given a query model from a certain class of size c, and a number of returned results k, and a number of relevant results within these returned results rel, then

$$recall = rel/c$$

$$precision = rel/k$$

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.

A.2 Things to Consider

In order to be able to properly compare classification methods, several factors have to be taken into account when producing p/r plots of information retrieval results. For a more elaborate discussion on p/r plots, their problems, and suggested improvements, see Huijsmans and Sebe [43, 44].

A.2.1 Generality

The most important parameter that is often ignored when p/r plots are presented is the *generality*, which is the size of a class relative to the database size (i.e. c/d, with d the total number of objects in the database). If precision values are simply averaged for each model (*micro-averaging*), then classes with high generality can bias the results. If, for example, one class in a test database of 1,000 models consists of 300 models, and the objects in this class happen to be easily matched using the tested method, the results will be biased. To alleviate this problem, sometimes *macro-averages* are also shown: here the class averages are computed first, which are then averaged. However, this could cause small classes to bias the results.

A solution is to explicitly show the dependency on generality by adding a third axis to the precision/recall graph. In practice, the generality values can be binned to integer values $-\log_2(g)$ (which for our 1,000 model test database would be five bins, $[-\log_2(0.004), -\log_2(0.073)] \approx [4, 8]$). The results can then be presented as five 2D precision/recall graphs, one for each generality bin.

Additionally, the distribution of class sizes should be published (as for example in Figure 7.2). When plots for different generality bins are given, the number of classes in each bin should also be published.

Showing these results can be especially helpful when different methods are compared using different databases. In our case, when comparing methods using the same database, it may still be useful to see if different methods behave differently depending on the class size. In this thesis we show graphs with micro-averages only, with the added note that all corresponding macro-averaged graphs were also produced, to verify that they showed the same qualitative results.

A.2.2 Miscellaneous class

A next, smaller issue is how to deal with the "miscellaneous" class present in some test databases, i.e. a class of objects that does not fit into any meaningful class, or whose class consists of only one object (Huijsmans and Sebe call this the "embedding", the number of additional irrelevant items in the database [43]). It is important to state if the class was discarded altogether, or if its members could be part of the results (because this makes it more difficult to achieve high precision). It does not make sense to use miscellaneous objects as queries.

The only test database used in this thesis that has a "miscellaneous" class is the database described in Section 4.5.1. In the experiments described in Section 4.5.1 and 5.5.2, the miscellaneous objects could be part of the results.

A.2.3 Possible Query Results

Another fact which should be clearly stated in the results is whether the query itself can be part of the results. This greatly affects the average precision values, since it is expected that if the query can be part of the results, it will show up very high in the ranked results.

For most matching methods we expect the “nearest neighbor” (i.e. the closest match) of each query to be the query itself. Thus, for example, in the similarity matrices in Figure 6.29 we expect the nearest neighbor (denoted by the $\boxed{\text{n}}$ symbol) to appear along the diagonal.

A.2.4 Random Results

Each precision/recall graph should include a curve which shows the performance of a random retrieval method. Besides giving an indication of the improvement of a certain method over random retrieval, it also gives us a clue about the “average generality” of a database: if there are only a few large classes, the chance of success of a random query would be too high, which would be visible in the height of the random retrieval precision/recall curve.

A.2.5 Averaging Precision Values in Recall Bins

In this last section we describe in detail the method we use for averaging precision values. If we want to present an average precision/recall graph in which the precision values are averaged in recall bins, and the test database has varying class sizes, then we have to be careful about how to average values of small classes.

Typically the domain of recall values $[0, 1]$ is subdivided into intervals (bins) of equal width. Given a certain bin, the closest actual recall value for a specific class is computed, and the corresponding precision value is added to this bin. However, for small classes it does not make sense to compute the precision for small recall values. For example, for a class of size 4 the relevant recall values are 0.25, 0.5, 0.75, and 1.0 (corresponding to 1, 2, 3, and 4 returned relevant results). If, for example, we need the precision value for a bin with its center at 0.35, for this class we can return the precision at 0.25 (since it is the closest recall value). However, if the bin has its center at a value < 0.125, we can no longer use the value at 0.25. The precision value effectively is 0.0/0.0 and thus undefined, and should not participate in the averaging for this bin.

To illustrate this procedure more precisely, we now give a function which computes a precision value given a recall value, query class size, and matching results. If the number of bins is 20, for example, this function is called with recall values 0.05, 0.1, 0.15, ..., 1.0. The bins then are the intervals [0.025, 0.075], [0.075, 0.125], etc. Note that this function is not the most efficient possible.

```
//
// given:
//
// query_id       model id of query model
// class_ids[]    class ids of models, indexed by model id
// class_sizes[]  class sizes, indexed by class id
// results[]      sorted array of matching results (model ids)
//

int compute_precision(float recall, float *precision_p)
{
  int query_class_id = class_ids[query_id];
  int target_relevant_retrieved =
    (int)(floor(recall * class_sizes[query_class_id] + 0.5));

  // if we're below the minimum, signal that this value should
  // not participate in the average
  if (target_relevant_retrieved == 0) return 0;

  int nr_relevant_retrieved = 0;
  int nr_retrieved = 0;
  int i = 0;

  while(nr_relevant_retrieved < target_relevant_retrieved) {
    int model_id = results[i];
    i++;
    nr_retrieved++;

    int other_class_id = class_ids[model_id];
    if (query_class_id == other_class_id)
      nr_relevant_retrieved++;

  }  // while

  *precision_p = (float) nr_relevant_retrieved / nr_retrieved;

  // signal success
  return 1;

}  // compute_precision_recall
```

Appendix B

Implementation Details

B.1 Introduction

This appendix describes implementation details of our search engine. Many sections refer to actual files and directories on the network of the Princeton computer science department.

Our 3D model search engine system executes in three main stages, as shown in Figure B.1. In the first stage, the 3D model data we want to make available is acquired. Next, this data is indexed such that later queries can be answered quickly. The third stage is the online component of the system, which matches user queries to the database in real-time, and returns results.

Figure B.2 shows an overview of the dataflow in the search engine. The individual steps are described in more detail in the following sections.

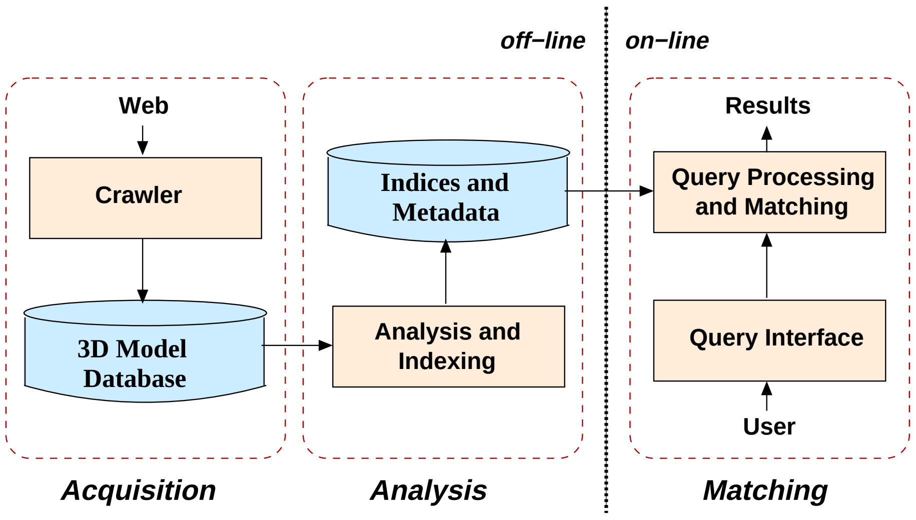


Figure B.1: High-level schematic of the three main components of our search engine

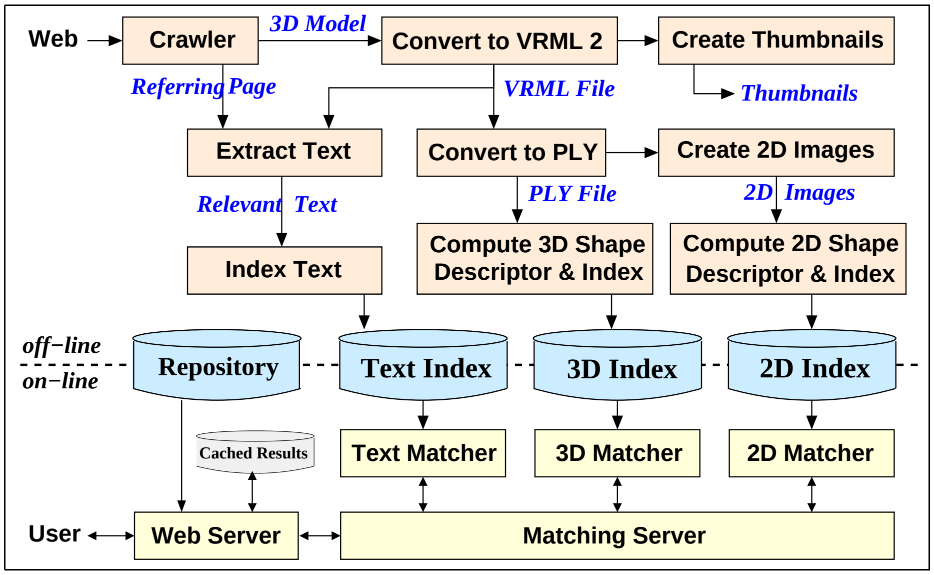


Figure B.2: Dataflow overview of the search engine

B.2 Acquistion

B.2.1 The Crawler

The crawler consists of a single server and multiple clients. The server maintains hash tables of visited and pending URLs, manages site priorities, and coordinates operations on the clients. It is written in C (because it performs all the computationally intensive tasks) and runs on a 733 MHz Pentium III machine with 384 MB memory (`square`). The server memory size determines the maximum size of the hashtable, and thus the maximum number of URLs that can be crawled. Note that the server makes sure that only one client at a time visits a site, and that the `robots.txt` protocol is being honored.

During a crawl, we run 80-100 simultaneous client processes on a single 400 MHz Pentium II machine with 384 MB memory, and 400 GB of RAID-5 disk space (`triangle`). The client software is written in Perl.

The following instructions were written by Alex Halderman on August 11, 2002:

Starting the crawler

Start server on square:
```
cd /data/c3/server
./c3s
```
Start clients on triangle:
```
cd /data/c3/client
./c3c
```

Terminating the crawler

You can kill all the processes by removing the lockfile:
```
rm -f /data/c3/lockfile
```
This happens automatically when either client or server is shut down.

Monitoring progress

See models as they are located:
```
tail -f /data/c3/client/data/models.list
```
Client logs are located in `/data/c3/client/data/log`.
Remotely observe server status:
```
watch cat /data/c3/server/data/stats.latest
```

Administration

Start a new crawl from scratch:

1. Stop the client and server
2. Remove existing data files:
   ```
   rm /data/c3/server/data/*
   rm /data/c3/client/data/log/*
   rm /data/c3/client/data/tmp/*
   rm /data/c3/client/models.list
   ```
3. Generate new server mmap:
   ```
   cd /data/c3/server/src
   make init
   ```
4. Restart server, add seeds, restart clients

Add initial seeds: This is only necessary when starting a fresh crawl or to add new seeds to a running crawl. With the server running, type:
```
cd /data/c3/utils/seed
./seed
```

Place more urls and queries in `data/forced.1` and `data/queries.1` to assign them to priority level 1. Run `./seed -q` to re-execute Google queries instead of using cached results.

B.3 Preprocessing

B.3.1 File Organization

All model databases and related files are at `/n/fs/shapedata`, a 200 GB network disk partition. The support staff does not make backups of this disk. One recent (March/April 2003) backup of all four model databases is on `triangle:/data`. Program files are stored on a 2 GB partition at `/n/fs/shape`, which is backed up once every night. These partitions are mounted as `/project/shapedata` and `/project/shape` on `chef`, which is an SGI running Irix 6.5. `triangle:/data` is mounted as `/hosts/triangle/data` on the public Linux servers.

The web model database is in a subdirectory of `shapedata` called `db`. The commercial databases are in `vp_db` (Viewpoint), `esp_db` (De Espona), and `cf_db` (CacheForce). In future work, the commercial databases should be combined into a single database.

Each model has been assigned a unique number, or "model id" (or `mid`). For the Viewpoint database, these are identical to the numbers that Viewpoint uses. The files pertaining to a single model are then located in a directory `/n/fs/shapedata/<db>/<subdir>/m<mid>`, where `<subdir>` = $\lfloor$`<mid>` / 1000$\rfloor$.

In the web database, models with ids < 23000 are models from the first crawl. Models with ids >= 25000 are models from the second crawl. Models with ids >= 23000 and < 24000 were the models originally collected for Rob Osada's Shape Distributions paper [72]. These have been removed since they were copies of web models anyway.

The following list describes the subdirectories in each model directory, and the files they contain.

- `model`
  For the models from the first crawl: the main model file, originally named `m<mid>u<url_id>.root.wrl`. Symlinks `m<mid>.wrl` and `root.wrl`, pointing to this file, have been added. This directory also contains dependent files, i.e. included VRML files and texture files. These files have been renamed, both in the directory as well as in the VRML file(s) itself. File references to VRML EXTERNPROTO's currently are not parsed correctly by the crawler. If the files were originally in the VRML 1.0 format, the original file is also present with the extension `.wrl1`.
  For the models from the second crawl: because these files may be from compressed archive files, and a single archive file potentially contains many models, here the `model` directory is a symbolic link to a directory below `triangle:/data/c3/pipeline/data` which contains all relevant files. In the future, these files should also be moved to appropriate directories on `shapedata`

- `info`
  Informational files about the model. The file `info.dat` has counts of the number

of triangles, quads, other polygons, and triangles after triangulation. It also has bounding box dimensions (after normalization such that the model fits in a unit cube), and three suggested cameras (position, lookat, up vectors), which are no longer used.

The files `link.m` and `link.r` have information on the link status of the model file and the referring page, respectively. These files are produced by the script `/n/fs/shape/jhalderm/linkcheck/lc.pl`.

The file `error.dat` is generated during the conversion from VRML 2.0 to PLY, and contains error messages that were generated during conversion (see also Section B.3.5)

- `outlines`
  If present, this directory contains the seven outline images used for the 2D sketch matching, as JPEG files. The `.sig` files there are the accompanying 2D signature files (these may be outdated). Note that not all models from the web database have these images (see Section B.3.7 for details)

- `ply`
  This directory contains a single file `m<mid>.ply`, a PLY file with only the converted geometry of the model, after triangulation

- `segments`
  Segmented versions of the model, by connected components (`m<mid>_cn.ply`), VRML IndexedFaceSet nodes (`m<mid>_fs.ply`), or VRML leaf grouping nodes (`m<mid>_gr.ply`). The corresponding `.dat` files contain the number of segments in each of these model files. These files were used for a different project

- `text`
  The text information for the model. `referrer.txt` has a copy of the referring HTML page (as well as some other data fields at the top, for example, the original model URL). `terms.txt` contains the words that were extracted from the various text sources for this model. `terms_proc.txt` is the processed version of `terms.txt` (stop words removed, synonyms added ...). This last file is used to build the text index. Also see Section B.3.4

- `thumbnails`
  Various thumbnail images of the model. `[small|large][0|1|2].[gif|jpg]` are the original thumbnail images (generated using Cortona, Internet Explorer, and a Visual Basic script running under Windows 2000). The files `new_[small|large][0-7].jpg` are the new thumbnails, generated using `mc` (my "all in one" program). The Viewpoint and De Espona thumbnails were automatically retrieved from the web using a dedicated script. Their filenames are slightly different (also see Section B.3.3 on thumbnail generation, and Section B.4.6 on building a results page)

Format	Converter
VRML 1.0	`vrml1ToVrml2`
Inventor	`ivToVRML`
AutoCAD DXF Wavefront OBJ Lightwave LW 3D Studio 3DS	`polytrans`
3D Studio MAX	3D Studio MAX

Table B.1: Supported 3D model formats, and converters used to convert to VRML 2.0

Several directories (including `model`) have a zero-length file `index.html`. This is a simple way to prevent people from browsing directories should they enter specific paths to model file directories in the URL field of their browser.

In principle each file and directory should be writeable by members of the Unix `shape` group (but this may not be the case, many files may still be owned by `min` or `jhalderm`). Note that all directories with models from commercial databases are accessible by the file owner only.

Most processing scripts that take ranges take three parameters: database name, starting directory, and ending directory (these scripts can be recognized by their inclusion of `../text/set_start_end.pl`).

B.3.2 Conversion to VRML 2.0

Because we generate thumbnail images, parts information, and text information from VRML 2.0 files, all files that are not in this format first have to be converted. Table B.1 lists the formats that are currently supported, and for each format the converter we use to convert it to VRML 2.0. All converters run under Irix 6.5, except 3D Studio MAX, which runs on a PC. 3D Studio MAX was used to convert the models from the De Espona database (using a separate batch conversion script).

B.3.3 Thumbnail Creation

The initial thumbnail generation setup (the Cortona VRML client by ParallelGraphics, running in Internet Explorer under Windows 2000, driven by a Visual Basic script) proved to be very unstable. After adding material and texture support to our VRML parser, we could use our own program to generate thumbnails.

The script `create_models_list.pl` in `/n/fs/shape/min/thumb` creates a list of model files for which thumbnails have to be generated. Only files with at least one triangle are included. For example: `create_models_list.pl db 0 1` will create a file `list_db_0.dat` in `/u/min/mc/mc`, containing entries for the first two directories of the web database. This list is input to `mc` (which is in `/u/min/mc/mc`). This program has a macro facility which allows the recording and playback of button clicks. The file `thumb.macro` in `/u/min/mc/mc/macros` contains the appropriate button sequence for generating the current set of 8 thumbnail images. The first line of the list of models has the name of the macro to be executed for that list. Invoke `mc` as follows: `mc -dp -ml <model list filename>`. Click the button "Process list" to start the process. The images are captured to a subdirectory `capture` in `/u/min/mc/mc`. A checkbox in `mc` controls whether the thumbnails are written to their appropriate subdirectories.

Make sure that the disk which holds the `capture` directory has plenty of space available. Do capture from a local display, on a PC with a fast graphics card (i.e. do not run the capture process on one of the public Linux servers and display locally).

B.3.4 Text Extraction and Indexing

The aforementioned file `referrer.txt` and the model's VRML file(s) are used as sources for the text extraction.

The Perl scripts dealing with text are in `/n/fs/shape/min/text`. The scripts to run, and their functions, are:

- `extract_words.pl`
  This script processes the `referrer.txt` file and the VRML model file. It produces a file `terms.txt` that contains the filename, the filename without digits, extension, web page title, anchor text, web page context, and model file identifiers. Digits are encoded as words, with an 'x' before and after it (for example, a 3 becomes "xthreex"). This to enable text queries with numbers (like "747"). Recently this script has been updated to also parse included model files (included using the VRML `Inline` node)

- `rem_stop.pl`
  This script removes stop words from the `terms.txt` file, and adds Wordnet synonyms for the filename. If no synonym could be found for the filename, the anchor text is tried instead. Note that the following environment variable has to be set: `setenv WNHOME /n/fs/shape/min/wordnet1.7.1`. It produces a file called `terms_proc.txt`. Constants near the top of the script control the maximum total number of words, and the maximum number of Wordnet senses and hypernyms to use

- `make_rainbow_source.pl`
  This script combines the `terms_proc.txt` files into one large text file, with one line per model. The first and second word on each line have to be the document name and class name. The class name is just the model id

- `create_indices`
  This shell script runs `rainbow` to create indices for the total database, and for each of the four databases individually (web, viewpoint, de espona, cacheforce).

Each index is written to a subdirectory in
`/n/fs/shape/min/server/sigfiles/text`

B.3.5 Conversion from VRML to PLY

The same program that is used to generate the thumbnail images, `mc`, is used to convert the VRML files to PLY. A derived program called `mconv` (which essentially is `mc` without the user interface) can also be used to do the conversion. Run `mconv` without parameters for a usage summary.

A Perl script called `cply.pl` (in `/n/fs/shape/min/process`) runs the conversion for a sequence of models. Note that this script also creates the `info.dat` and `error.dat` files in the model's `info` subdirectory.

For those interested, the source code to support VRML is in `/u/min/cc/vrml`. The classes for the internal mesh data structure are in `/u/min/cc/geom`. The class `MeshFile` in that directory has an extensive (excessive) function `traverse_vrml_node` which does the right thing for most of the VRML geometry nodes (in particular, material and texture associations, triangulation, and texture coordinate computation).

B.3.6 3D Shape Descriptors

Use the program `L2ModelDatabase`, written by Misha Kazhdan, to generate shape signature databases. The file is in `/n/fs/shapedata/misha/InProgress/L2ModelDatabase`. A copy is in `/n/fs/shape/min/server/bin`. It takes two files as input: one with a list of PLY files, one with a list of model ids. The commandline to use:

```
L2ModelDatabase new HarmonicFourierCharged3D 1 <PLY file list> <id list>
```

Run `L2ModelDatabase` without parameters to get a usage summary.

B.3.7 2D Views

The 2D orthographic outline images of the models are created in a way similar to the thumbnail images (see Section B.3.3). The same model list file has to be created. After the first crawl, about 2,000 models were designated to participate in the 2D sketch matching. The list of ids was converted to a list suitable as input to `mc` by a script `vis_to_filelist.pl` in `/n/fs/shape/min/process`.

A macro `outlines` is used to generate the 7 outline images. Note that several options have to be set in `mc` before clicking the "Process list" button: disable thumbnail, enable "black and white", disable wireframe, disable (toggle) lighting, enable orthographic, disable grid, disable axes, enable pgm, enable outline, enable capture in subdirs.

In future work, the source images could be made more suitable for matching by first filling in interior holes, and then dilating the resulting image a few times (to smooth the boundary), before detecting the boundary.

B.3.8 2D Shape Descriptors

To generate a 2D shape descriptor database you should also use `L2ModelDatabase` (see above, Section B.3.6). The commandline changes slightly:
`L2ModelDatabase new <db name> Fourier2D 7 <PGM file list> <id list>`
In this case the PGM file list will have 7 times more lines than the id list.
Run `L2ModelDatabase` without parameters to get a usage summary.

B.4 The Search Engine Site

B.4.1 Overview

The search engine site is accessible at the URL `http://shape.cs.princeton.edu`. This is a virtual web server run by the Princeton CS department. A request for the main site URL will load the file `index.cgi` from `/n/fs/shape/www`. This script logs the date, time, remote hostname, referring web page and browser used (to `reflog.dat` in `/n/fs/shapedata/min/reflog`) and then redirects the browser to the file `search.html`.

`search.html` simply creates two HTML frames. The "query frame" on the left is filled by `query.cgi`, the "results frame" on the right by `results.cgi`. Both scripts are in `./search`, a symbolic link to `/n/fs/shape/min/search`. This directory holds all relevant scripts, pages, applets, etc., for the online site, as well as other data such as a copy of the matching server's log files, and, for example, the CS111 user study site.

For the text & sketch query types, `query.cgi` constructs a small web page with a Java applet that combines the text and the 2D/3D sketch interfaces. The Javascript code is there only to disable and enable the "Search" button at the appropriate times (it is important that it is not possible to start multiple searches from the same page, and to give feedback to the user that a search is underway).

The Java applet calls `results.cgi` for the results frame, and passes it the necessary parameters. `results.cgi` runs on the web server, and calls `get_results.cgi`, which is a relatively small script that communicates with the matching server. Once the results are in, `results.cgi` constructs a results page. The results page has a small Javascript function that re-enables the Search button, and another function that disables the search again if a user clicks a "find similar shape" link.

B.4.2 Text and/or 2D Sketch Queries

The query interface for text and/or 2D sketch queries is implemented completely in a Java applet. The source code for this applet is in `/n/fs/shape/min/search/sketch`.

The main class is `Sketch`, and is derived from `SearchApplet`. `SearchApplet` is an applet that implements a `do_search` function (from the interface `Search`)(this interface can probably be dropped). The `Sketch` class has an array of three `SketchCanvas` classes, in which the actual drawing takes place. Note that the drawing is rendered using two-pixel thick lines, but is stored as single-pixel lines. `SketchCanvas` also keeps track of the drawing history (commands and timestamps, mouse coordinates) and stores them in an instance of the `History` class.

The main applet also has an instance of `QueryPanel`, which holds the "Search" button, the model database selection list, and the text keyword entry field. This class, as well as the `SearchApplet` class is also used by the text and/or 3D Sketch interface (described in the next section).

For a query, the entered text, 2D sketches, and drawing history are sent to the web server. The URL with this information is constructed in the `Submit` class. This class also sends the data. The images are sent as indices of pixels that are set. The `Submit` class requests the script `sketch.cgi`. This script saves the the sketches as PGM files to `image_upload_<random>.pgm` in `/n/fs/shapedata/tmp/<year>/<month>/<day>`, where `<random>` is 6 random characters. The history files are saved to the same filename with the extension `.hist`. The PGM filenames are returned to the `Submit` class, which uses those to construct a query (by calling `results.cgi` from `Sketch`).

The `Globals` class holds some global constants (fonts, colors, etc.).

B.4.3 Text and/or 3D Sketch Queries

The query interface for text and/or 3D sketch is very similar to the text and/or 2D sketch interface, and also implemented in Java. The source code is in the directory `/n/fs/shape/min/search/teddy_comb`.

The main class is `Teddy`, also derived from `SearchApplet`. The original source code (written by Takeo Igarashi) was modified slightly: some buttons (such as the "Save" and "Bend" buttons) were removed. Also, the `QueryPanel` class mentioned in the previous section was added, and a `do_search` function was added to `Teddy`.

`QueryPanel.java`, `SearchApplet.java`, `Search.java` and `Globals.java` are symbolic links to the corresponding files in the 2D sketch source directory.

`Submit.java` is different for this interface, because different data is sent (in this case, the 3D model in Wavefront `.obj` format). The sequence is the same: first a script `teddy.cgi` is called, which saves the model file to a file `teddy_upload_<random>.obj` and returns this filename to the applet. After this, `results.cgi` is called (from `Teddy`).

B.4.4 3D File Upload

This interface is implemented as an HTML form in `query.cgi`. It has five fields: a Search button, a database selection list called "dataset", one file browser field called "modelfile", and two hidden fields "input" (value = "file") and "method" (value =

"harmonic"). These variables and their values are passed to `results.cgi` after the user submits the form.

B.4.5 Find Similar Shape

The "Find similar shape" links on the results page directly call `results.cgi` again with the appropriate parameters, the most important being `mid`, which holds the model id of that particular result model.

B.4.6 The Results Page

The results page is constructed by `results.cgi`. Many different pages can be generated by this script. For now, we'll just look at the function `print_results`. This function parses the result string returned by `get_results.cgi` (the script that did the actual communicating with the matching server). It fills variables as they come along, and as soon as the word `end_model` is encountered, a function `print_model_info` is called, which generates the right HTML for that particular result model. This includes generating the right links for the information window, a "find similar shape" request, and the thumbnail image. `thumbnail.pl` has a function which returns the right thumbnail image URL depending on the database and model id.

Caching Results

The matching server writes the model ids and scores of each search to an individual file in `/n/fs/shapedata/tmp/<year>/<month>/<day>/results_<ip>_<search id>.dat`.

If `results.cgi` determines that it was called with a "browse" request (the `si` (search_id) parameter was not empty), it will read the results from this file, and use these to construct the results page.

Note that the "page view" itself is logged, appended to a file `pageview_<ip>.dat` in the same directory.

B.4.7 The Information Window

If a user clicks on a thumbnail image on a results page, an information window about that particular model is shown. The information in this window is created by the script `info.cgi`.

The script itself should be self-explanatory. Note that the action is logged in a file `/n/fs/shapedata/min/info_log/<year>/<month>/<day>/info_<ip>_<search id>.dat`.

Logging redirects

The links in the information window to the referring page and the model file itself are indirect, and go through a script `redirect.cgi`.

The fact that such a link was clicked is logged in
`/n/fs/shapedata/min/redirects/redirect_<ip>_<search id>_<type>.dat`
where type is either `model` or `ref`.

B.4.8 Miscellaneous

Feedback

The "Contact Us" link in the header brings up a simple HTML form where people can type in their comments, and optionally their e-mail address. This page is also constructed by `results.cgi`.

Upon submitting the form, `feedback.cgi` is called, which saves it to
`/n/fs/shapedata/min/feedback/feedback_<ip>_<random>.txt`
where `<random>` is a random sequence of six characters) and forwards a copy to my e-mail address.

Error log of `http://shape.cs.princeton.edu`

The support staff has set up an error log viewer at:
`https://csguide.cs.princeton.edu/log_view/?server=shape.cs.princeton.edu`
The most frequently occuring errors are references to `.au` files (sound files) from Teddy, which should be removed from the source. There are also many references to non-existing "BeanInfo" classes, which are caused by a quirk in Internet Explorer. A solution could be to create empty stub classes for each of these.

B.5 The Matching Server

The files related to the matching server are in `/n/fs/shape/min/server`. There, the subdirectory `bin` holds required binaries, and `sigfiles` holds all necessary indices (2D and 3D signature databases, and text indices for the various model sets). The current machine we use as a matching server is `hexagon`, a Dell Precision 530 with two Intel Xeon processors at 1.5 GHz, 512 MB memory, running Red Hat Linux 7.2. It has one 36 GB SCSI disk.

The main server control script is `server.cgi` (should be called `server.pl`). It listens on port 7073 for matching requests from `get_results.cgi`
(in `/n/fs/shape/min/search`). For each client, it forks a separate process which reads commands and executes those. Most commands just set the values of parameters. The `query` command causes an actual search to be executed.

All matchers run as separate server processes on `hexagon`. Shell scripts to start these servers are `run_harm` (for the 3D server), `run_2d` (for the 2D server), and `run_text` (for the text servers) in `./bin`. The 2D and 3D servers load all databases

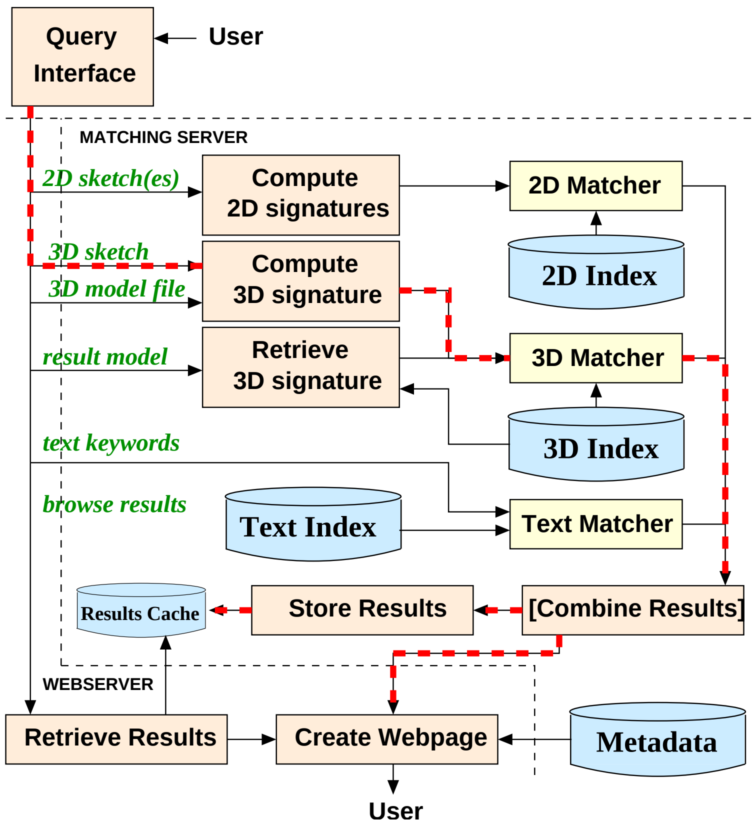


Figure B.3: Query processing and matching stage. The path of a 3D sketch query has been highlighted

and are capable of answering queries on subsets. For text queries, five servers are run (one for each database, and one for all of them combined).

The script `cs.pl` has functions for processing each query type. It calls a function in `matching_server.pl` to connect to the appropriate server, then sends the query, and stores the results. For combined queries (for example, text and 2D sketch) both queries are performed separately and the results are combined (the scores are mean normalized and averaged, note the special cases for models that have a score from one query but not from the other).

The supporting script `gs.pl` has functions to generate 2D and 3D signatures, convert models, etc. These are called for a 3D file upload query, for example.

Also note the careful logging and message printing code. Log files should always be locked before writing to them. The log messages related to one client are first collected in a string, and then written in one go after a client disconnects (because there can be many clients connected at the same time).

B.5.1 Running the server

The server is run *locally* on `hexagon`. The files are in `/usr/min/server`.

First, run `/home/min/tst` on `hexagon` to see if the server is running. If it is, it should show one `server.cgi` process, up to four `L2ServerNov3` processes, two with port 9100, two with port 9200, five `rainbow` processes (one for each database, and one for all), and one `check_server.pl` process. This last process is a small script that checks periodically if the `L2ServerNov3` processes are running, and if not, restarts them (they rarely crash, but still do).

To start a server from scratch, do the following:

- in `/usr/min/server/bin`, run `run_text`, `run_2d &` and `run_harm &`
- from the same directory, run `check_server.pl &`
- in `/usr/min/server`, run `server.cgi &`

B.5.2 Log Analysis

First, kill the running `server.cgi` and start a new one. The script will create a new log file in `/usr/min/server/log`. Copy the latest log file to `/n/fs/shape/min/search/log`. **Careful:** note the differing sequence numbers in both directories. The script `next.pl` in `/usr/min/server` performs these steps automatically. This script requires `/n/fs/shape` to be mounted on `hexagon` at `/shape`. To mount this directory, run:

```
smbmount //fs/shape /shape -o username=<username>,workgroup=cs,password=<password>
```

Next, on one of the public Linux servers, change to `/n/fs/shape/min/weblog` and run `analyze.pl`. It starts by default with log file `log00085.dat` (there were no outside hits before then). This script creates web pages in `/u/min/public_html/mc/log`. Some of the data files it creates are input to PHP scripts in that directory. These scripts use the JpGraph package to generate a few graphs on the log pages.

Bibliography

[1] 3D Café. 3D Café free 3D models meshes. `http://www.3dcafe.com`.

[2] Alias Wavefront. Maya. `http://www.aliaswavefront.com`.

[3] AltaVista. Altavista search engine. `http://www.altavista.com`.

[4] D. Anderson, J. L. Frankel, J. Marks, A. Agarwala, P. Beardsley, J. Hodgins, D. Leigh, K. Ryall, E. Sullivan, and J. S. Yedidia. Tangible interaction + graphical interpretation: A new approach to 3D modeling. In *Proc. SIGGRAPH*, pages 393–402, New Orleans, LA, July 2000. ACM.

[5] M. Ankerst, G. Kastenmüller, H.-P. Kriegel, and T. Seidl. 3D shape histograms for similarity search and classification in spatial databases. In *Proc. SSD*, 1999.

[6] E. M. Arkin, L. P. Chew, D. P. Huttenlocher, K. Kedem, and J. S. Mitchell. An efficiently computable metric for comparing polygonal shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(3):209–216, March 1991.

[7] Y. Aslandogan and C. Yu. Techniques and systems for image and video retrieval. *IEEE Trans. on Knowledge and Data Engineering*, 11(1):56–63, 1999.

[8] D. Attali and A. Montanvert. Computing and simplifying 2D and 3D continuous skeletons. *Computer Vision and Image Understanding*, 67(3):261–273, September 1997.

[9] Avalon. Avalon 3D archive. `http://avalon.viewpoint.com`.

[10] 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.

[11] I. Biederman. Recognition-by-components: A theory of human image understanding. *Psychological Review*, 94(2):115–147, 1987.

[12] T. Binford. Visual perception by computer. In *IEEE Conference on Systems Science and Cybernetics*, 1971.

[13] Blender. Blender 3D modeler. `http://www.blender3d.org`.

[14] H. Blum. A transformation for extracting new descriptors of shape. In W. Wathen-Dunn, editor, *Proc. Models for the Perception of Speech and Visual Form*, pages 362–380, Cambridge, MA, November 1967. MIT Press.

[15] M. Bober. MPEG-7 visual shape descriptors. *IEEE Tr. on Circuits and Systems for Video Technology*, 11(6):716–719, June 2001.

[16] K. Bollacker, S. Lawrence, and C. L. Giles. CiteSeer: An autonomous web agent for automatic retrieval and identification of interesting publications. In K. P. Sycara and M. Wooldridge, editors, *Proceedings of the Second International Conference on Autonomous Agents*, pages 116–123, New York, 1998. ACM Press.

[17] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[18] Cacheforce. Cacheforce 3D model libraries. `http://www.cacheforce.com`.

[19] CADlib. CADlib web based CAD parts library. `http://www.cadlib.co.uk`.

[20] 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.

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

[22] T. Cootes, C. Taylor, and A. Lanitis. Active shape models: Evaluation of a multi-resolution method for improving image search. In *Proc. British Machine Vision Conf.*, pages 327–336, 1994.

[23] J. Corney, H. Rea, D. Clark, J. Pritchard, M. Breaks, and R. MacLeod. Coarse filters for shape matching. *IEEE Computer Graphics & Applications*, 22(3):65–74, May/June 2002.

[24] Cyberware. 3D scanning systems and software. `http://www.cyberware.com`.

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

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

[27] De Espona. De Espona Infografica 3D model collection. `http://www.deespona.com`.

[28] P. E. Debevec. *Modeling and Rendering Architecture from Photographs*. PhD thesis, University of California at Berkeley, 1996.

[29] S. J. Dickinson, R. Bergevin, I. Biederman, J.-O. Eklundh, R. Munck-Fairwood, A. K. Jain, and A. Pentland. Panel report: the potential of geons for generic 3-D object recognition. *Image and Vision Computing*, 15(4):277–292, April 1997.

[30] P. Dimitrov, C. Phillips, and K. Siddiqi. Robust and efficient skeletal graphs. In *Proc. CVPR*, Hilton Head, South Carolina, June 2000.

[31] Drexel University. National design repository. `http://www.designrepository.org`.

[32] M. Elad, A. Tal, and S. Ar. Content based retrieval of VRML objects - an iterative and interactive approach. In *Proc. EG Multimedia*, pages 97–108, September 2001.

[33] C. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic, and W. Equitz. Efficient and effective querying by image content. *Journal of Intelligent Information Systems*, 3(3/4):231–262, 1994.

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

[35] D. A. Forsyth and J. Ponce. *Computer Vision, A Modern Approach*. Prentice Hall, 2002.

[36] 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.

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

[38] Google. Image search faq. `http://images.google.com/help/faq_images.html`, April 2003.

[39] W. Grimson. *Object recognition by computer: the role of geometric constraints*. MIT Press, Cambridge, MA, 1990.

[40] G. Hoff and M. Mundhenk. Creating a virtual library with HomePageSearch and Mops. In *Proc. 19th SIGDOC Conference*, Santa Fe, NM, October 2001. ACM.

[41] B. Horn. Extended Gaussian images. *Proc. of the IEEE*, 72(12):1671–1686, December 1984.

[42] B. Huet and E. R. Hancock. Structural indexing of infra-red images using statistical histogram comparison. In *Proc. IWISP*, Manchester, UK, November 1996.

[43] D. Huijsmans and N. Sebe. Extended performance graphs for cluster retrieval. In *Proc. CVPR*, volume 1, pages 26–31, Hawaii, December 2001. IEEE.

[44] D. Huijsmans and N. Sebe. How to complete performance graphs in content-based image retrieval: Add generality and relevant scope. *IEEE PAMI*, 2004. *to appear.*

[45] T. Igarashi and J. F. Hughes. A suggestive interface for 3D drawing. In *Proc. UIST*, pages 173–181, Orlando, November 2001. ACM.

[46] T. Igarashi, S. Matsuoka, and H. Tanaka. Teddy: A sketching interface for 3D freeform design. In *Proc. SIGGRAPH 1999*, pages 409–416, Los Angeles, CA, August 1999. ACM.

[47] Informatics & Telematics Institute. 3D search. `http://3d-search.iti.gr`.

[48] C. E. Jacobs, A. Finkelstein, and D. H. Salesin. Fast multiresolution image querying. *Proc. SIGGRAPH*, pages 277–286, August 1995.

[49] A. K. Jain, Y. Zhong, and S. Lakshmanan. Object matching using deformable templates. *IEEE PAMI*, 18(3):267–278, March 1996.

[50] T. Joachims. A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization. In *Proc. Int. Conf. on Machine Learning*, pages 143–151, 1997.

[51] M.-P. D. Jolly, S. Lakshmanan, and A. K. Jain. Vehicle segmentation and classification using deformable templates. *IEEE PAMI*, 18(3):293–308, March 1996.

[52] O. Karpenko, J. F. Hughes, and R. Raskar. Free-form sketching with variational implicit surfaces. In *Proc. EUROGRAPHICS*, volume 21, 2002.

[53] T. Kato, T. Kurita, N. Otsu, and K. Hirata. A sketch retrieval method for full color image database. In *Proc. ICPR*, pages 530–533. IEEE, 1992.

[54] M. Kazhdan, T. Funkhouser, and S. Rusinkiewicz. Rotation invariant spherical harmonic representation of 3d shape descriptors. In *Proc. Symposium on Geometry Processing*, pages 156–165. ACM, June 2003.

[55] M. Keil and J. Snoeyink. On the time bound for convex decomposition of simple polygons. *Int. J. of Comp. Geometry & Appl.*, 12(3):181–192, 2002.

[56] A. Khotanzad and Y. H. Hong. Invariant image recognition by Zernike moments. *IEEE PAMI*, 12(5):489–497, May 1990.

[57] I. Kolonias, D. Tzovaras, S. Malassiotis, and M. G. Strinzis. Fast content-based search of VRML models based on shape descriptors. In *Proc. Int. Conf. on Image Processing*, Thessaloniki, Greece, October 2001. IEEE.

[58] T. Kong and A. Rosenfeld. Digital topology: Introduction and survey. *Computer Vision, Graphics, and Image Processing*, 48(3):357–393, December 1989.

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

[60] W. H. Leung and T. Chen. Trademark retrieval using contour-skeleton stroke classification. In *Proc. Int. Conf. on Multimedia and Expo*, volume 2, pages 517–520. IEEE, 2002.

[61] F. Leymarie and M. D. Levine. Simulating the grassfire transform using an active contour model. *IEEE PAMI*, 14(1):56–75, January 1992.

[62] Y. Lin, J. Dou, and H. Wang. Contour shape description based on an arch height function. *Pattern Recognition*, 25(1):17–23, 1992.

[63] W.-Y. Ma and B. Manjunath. NeTra: A toolbox for navigating large image databases. *ACM Multimedia Systems*, 7(3):184–198, 1999.

[64] D. Marr and H. Nishihara. Representation and recognition of the spatial organization of three-dimensional shapes. In *Proc. of the Royal Society of London, series B*, volume 200, pages 269–294, February 1978.

[65] M. D. Marsicoi, L. Cinque, and S. Levialdi. Indexing pictorial documents by their content: a survey of current techniques. *Image and Vision Computing*, 15(2):119–141, February 1997.

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

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

[68] G. A. Miller. WordNet: A lexical database for English. *Communications of the ACM*, 38(11):39–41, 1995.

[69] 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.

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

[71] F. Mokhtarian, S. Abbasi, and J. Kittler. Efficient and robust retrieval by shape content through curvature scale space. In *Proc. Int. Workshop on Image Database and Multimedia Search*, pages 35–42, Amsterdam, the Netherlands, August 1996.

[72] R. Osada, T. Funkhouser, B. Chazelle, and D. Dobkin. Matching 3D models with shape distributions. In *Proc. Shape Modeling International*, pages 154–166, Genova, Italy, May 2001. IEEE Press.

[73] S. Palmer, E. Rosch, and P. Chase. Canonical perspective and the perception of objects. *Attention and Performance IX*, pages 135–151, 1981.

[74] E. Paquet and M. Rioux. Nefertiti: A tool for 3-D shape databases management. *SAE Transactions: Journal of Aerospace*, 108:387–393, 2000.

[75] M. Pelillo, K. Siddiqi, and S. W. Zucker. Attributed tree matching and maximum weight cliques. In *Proc. Image Analysis and Processing*, pages 1154–1159. IEEE, 1999.

[76] A. Pentland. Perceptual organization and the representation of natural form. *Artificial Intelligence*, 28(2):293–331, 1986.

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

[78] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C*, chapter 10, Minimization or Maximization of Functions, pages 463–469. Cambridge University Press, 2nd edition, 1992.

[79] R. J. Prokop and A. P. Reeves. A survey of moment-based techniques for unoccluded object representation and recognition. *CVGIP: Graphical Models and Image Processsing*, 54(5):438–460, 1992.

[80] A. Razdan, D. Liu, M. Bae, M. Zhu, G. Farin, A. Simon, and M. Henderson. Using geometric modeling for archiving and searching 3D archaeological vessels. In *Proc. Conf. on Imaging Science, Systems and Technology*, Las Vegas, June 2001.

[81] W. C. Regli and V. Cicirello. Managing digital libraries for computer-aided design. *Computer-Aided Design*, 32(2):119–132, February 2000.

[82] R. D. Reyna. *How to Draw What You See.* Watson-Guptill Pubns, September 1996.

[83] J. Rocchio. *The SMART Retrieval System: Experiments in Automatic Document Processing*, chapter Relevance Feedback in Information Retrieval, pages 313–323. Prentice-Hall, 1971.

[84] J. Rowe, A. Razdan, D. Collins, and S. Panchanathan. A 3D digital library system: Capture, analysis, query, and display. In *Proc. 4th Int. Conf. on Asian Digital Libraries (ICADL)*, Bangalore, India, December 2001.

[85] C. Sable, K. McKeown, and K. W. Church. Nlp found helpful (at least for one text categorization task). In *Proc. Empirical Models in Natural Language Processing*, Philadelphia, 2002.

[86] 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.

[87] T. Saito and J.-I. Toriwaki. New algorithms for Euclidian distance transformation of an n-dimensional digitized picture with applications. *Pattern Recognition*, 27(11):1551–1565, 1994.

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

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

[90] 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.

[91] T. B. Sebastian, P. N. Klein, and B. B. Kimia. Recognition of shapes by editing shock graphs. In *Proc. ICCV*, pages 755–762, 2001.

[92] K. Siddiqi and B. B. Kimia. A shock grammar for recognition. In *Proc. CVPR*, pages 507–513. IEEE, 1996.

[93] K. Siddiqi, A. Shokoufandeh, S. J. Dickinson, and S. W. Zucker. Shock graphs and shape matching. *Int. Journal of Computer Vision*, 35(1):13–31, 1999.

[94] SIGIR. ACM special interest group on information retrieval. `http://www.acm.org/sigir/`.

[95] T. Sikora. The MPEG-7 visual standard for content description - an overview. *IEEE Tr. on Circuits and Systems for Video Technology*, 11(6):696–702, June 2001.

[96] A. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. Content-based image retrieval at the end of the early years. *IEEE PAMI*, 22(12):1349–1380, December 2000.

[97] J. Smith and S.-F. Chang. Multi-stage classification of images from features and related text. In *Proc. EDLOS Workshop*, San Miniato, Italy, August 1997.

[98] J. R. Smith and S.-F. Chang. Visualseek: a fully automated content-based image query system. In *Proc. ACM Multimedia*, pages 87–98, Boston, MA, November 1996.

[99] F. Solina and R. Bajcsy. Recovery of parametric models from range images: The case for superquadrics with global deformations. *IEEE PAMI*, 12(2):131– 147, February 1990.

[100] P. Stanchev. Content-based image retrieval systems. In *Proc. Bulgarian Comp. Sc. Conf.*, Bulgaria, June 2001.

[101] M. T. Suzuki. A web-based retrieval system for 3D polygonal models. *Joint 9th IFSA World Congress and 20th NAFIPS International Conference (IFSA/NAFIPS2001)*, pages 2271–2276, July 2001.

[102] J. W. Tangelder and R. C. Veltkamp. Polyhedral model retrieval using weighted point sets. *International Journal of Image and Graphics*, 3(1):209–229, 2003.

[103] H. Tek and B. B. Kimia. Boundary smoothing via symmetry transforms. *J. of Mathematical Imaging and Vision*, 14(3):211–223, 2001.

[104] TREC. Text REtrieval conference. `http://trec.nist.gov`.

[105] Utrecht University. 3D shape retrieval engine. `http://www.cs.uu.nl/centers/give/imaging/3Drecog/3Dmatching.html`.

[106] R. C. Veltkamp and M. Tanase. Content-based image retrieval systems: A survey. Technical Report UU-CS-2000-34, Utrecht University, October 2000.

[107] Viewpoint. Viewpoint corporation. `http://www.viewpoint.com`.

[108] A. Witkin, K. Fleischer, and A. Barr. Energy constraints on parameterized models. In *Proceedings of SIGGRAPH 1987*, Computer Graphics Proceedings, Annual Conference Series, pages 225–232. ACM, 1987.

[109] K. Wu and M. Levine. Recovering parametric geons from multiview range data. In *Proc. CVPR*, pages 159–166, June 1994.

[110] Yahoo. Yahoo web directory. `http://www.yahoo.com`.

[111] A. Yuille, P. Hallinan, and D. Cohen. Feature extraction from faces using deformable templates. *Int. Journal of Computer Vision*, 8(2):133–144, 1992.

[112] C. Zahn and R. Roskies. Fourier descriptors for plane closed curves. *IEEE Trans. Computers*, 21:269–281, 1972.

[113] R. Zeleznik, K. Herndon, and J. Hughes. SKETCH: An interface for sketching 3D scenes. In *Proc. SIGGRAPH*, pages 163–170. ACM, 1996.