Object Representation and Recognition*

Sven J. Dickinson
Center for Cognitive Science and
Department of Computer Science
Rutgers University
New Brunswick, NJ 08903

1 Introduction

One of the primary functions of the human visual system is object recognition, an ability that allows us to relate the visual stimuli falling on our retinas to our knowledge of the world. For example, object recognition allows you to use knowledge of what an apple looks like to find it in the supermarket, to use knowledge of what a shark looks like to swim in the other direction, and to use knowledge of the landmarks in your neighborhood to find your way home. Recognition allows us to understand the content of images. Only by assigning a label to an image object can we ground the object in our own experience.

Human object recognition seems effortless. From prototypical or generic knowledge of an object, you can easily recognize novel instances of the object. For example, your internal model for how a dog appears is sufficient for you to recognize a new breed of dog (that you've never seen before) as a dog, regardless of whether it is standing, sitting, or running. And in cluttered scenes where an object is only partially visible (or occluded), recognition is still possible. Seeing only the front end of the car peeking out from behind the billboard is enough to allow you to recognize it as a police cruiser. These examples raise a number of important questions: How is visual knowledge of an object encoded? What information is recovered from an image in order to recognize an object? And how is this information compared to our stored knowledge in order to recognize the object?

Humans use many visual cues to recognize an object. For example, the pattern of black and white stripes on an animal may be a more powerful cue to identifying the animal as a zebra than the shape of its body. Or, back in the supermarket, although many of the fruits share the same spherical shape, the color orange draws you to the box of oranges. Clearly, the most powerful cue used to identify an object is its shape. Even while covering one eye, the human observer can quickly recognize the three-dimensional objects that appear as two-dimensional line drawings in a slide show. Coined "couch potato vision" by the psychologist, Irving Biederman, the experiment shows that the shape information recovered

*This paper appears as a chapter in: E. Lepore and Z. Pylyshyn (eds.) Rutgers University Lectures on Cognitive Science, Basil Blackwell publishers, 1999, pp 172–207.

from image contours is sufficient for object recognition; color, texture, shading, and stereo (the depth information gained by uncovering your other eye) are *not* essential to the task. Since shape plays such an important role in object recognition, this chapter will focus on the representation and recognition of objects based on their shape.

In this chapter, we will explore object recognition from the standpoint of computer vision. Building a computer vision system to perform a given visual recognition task requires careful attention to the entire process, including object representation, feature extraction, object database organization, and model indexing. Building computer vision systems allows us to prototype competing representations and algorithms, yielding powerful constraints and insight which can be used to postulate and evaluate theories of human object recognition. Conversely, to the extent that a computer vision system aspires to solve a recognition problem as well as a human, the insight acquired by the human visual recognition community should be exploited when designing the system, and the human visual system should become the ultimate measuring stick by which the system is evaluated.

In the remainder of this chapter, we begin by illustrating the recognition problem using some well-known examples drawn from the computer vision community. Next, we discuss the problem of choosing an object representation given the recognition task, outlining a number of representational properties and their trade-offs. We then discuss the problem of matching recovered image features to object models, beginning with an illustrative case study drawn from the optical character recognition (OCR) domain, and returning to our well-known examples from the computer vision community. One of the more powerful recognition paradigms to emerge, called recognition by parts, is explored through two different approaches, one drawn from the human vision community and one from the computer vision community. The limitations of these two approaches will lead to a discussion of where the field of object recognition is heading and what the outsanding problems are.

2 Some Example Problems

The input to an object recognition system is a digital image, a two-dimensional array of numbers called pixels. Each pixel represents a measurement recorded by the sensor responsible for acquiring the image. For example, a typical black and white video camera will produce an image whose pixels represent how "bright" the picture is at that point; dark areas in the image have low-valued pixels while light areas have high-valued pixels. Another popular sensor in the computer vision community is the *laser rangefinder*, which produces an image whose pixels represent the distance to objects in the world. For example, objects in the field of view which are near the rangefinder camera will have low-valued pixels while objects far away will have high-valued pixels. Such *range images* are effectively three-dimensional, avoiding the ambiguity inherent in two-dimensional images.[1]

To illustrate the kinds of object recognition problems facing computer vision systems, we show the actual images presented to a number of well-known object recognition systems. Figure 1(a) shows a typical input image for the recognition system of Murase and Nayar

[1]There is no way of telling whether a disk in a video image is a small disk positioned close to the camera or a large disk positioned further away. On the other hand, a range image containing the same disk will specify the actual distance from the sensor to the disk.

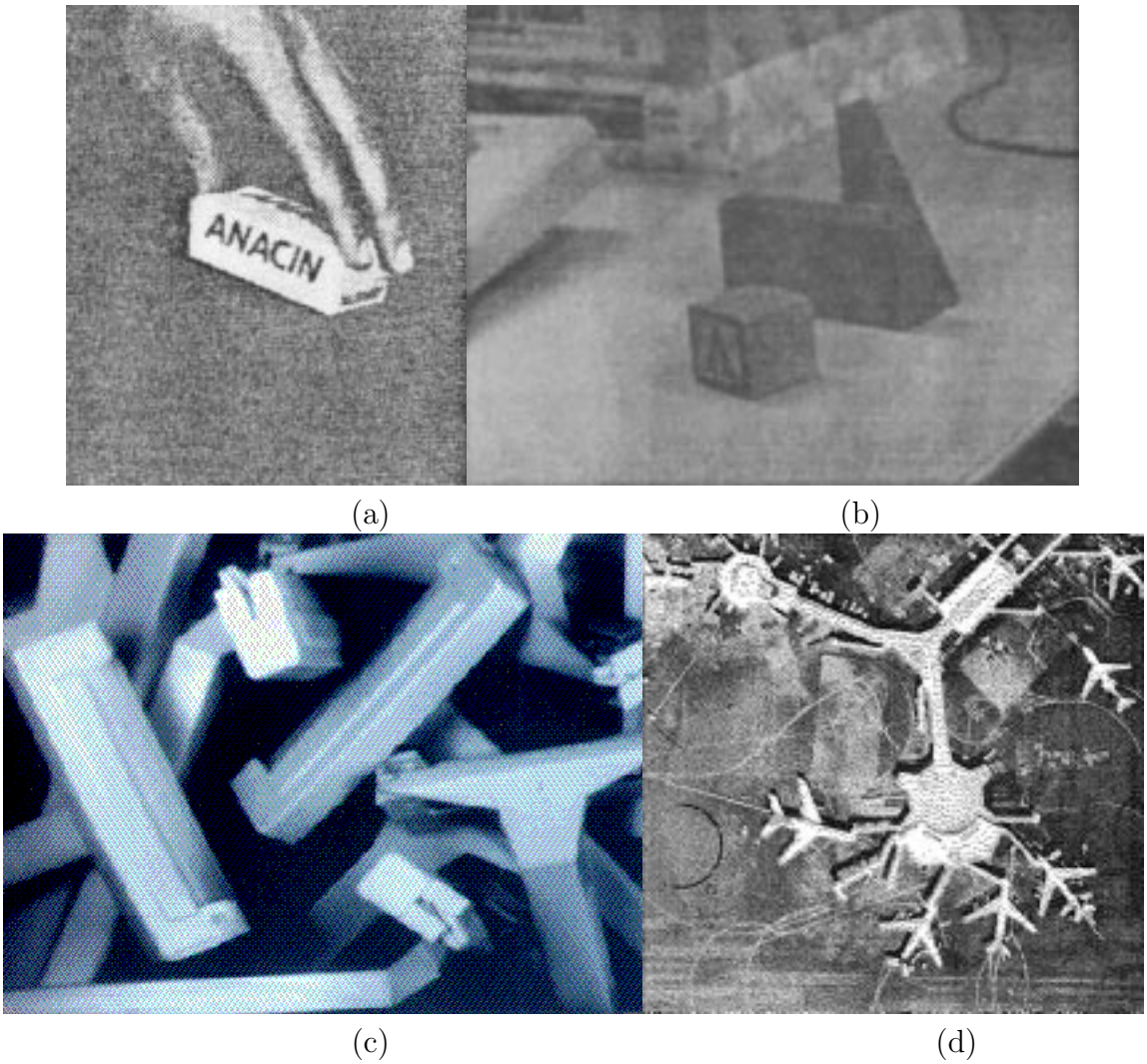


Figure 1: Example Object Recognition Tasks

[39]. The task is to identify the single, unoccluded object in the image from a database of 100 model objects. Figure 1(b) shows a typical input image for the recognition system of Huttenlocher and Ullman [29]. The task is to identify the polyhedral objects in the image from a small database of polyhedral object models. In this case, there may be multiple objects in the image and the system is expected to not only identify objects that are partially visible, but determine their exact position and orientation in the world. This latter task is called *pose estimation*.

Figure 1(c) shows a typical input image for the recognition system of Lowe [34]. As in the previous example, multiple, occluded objects may be present in the image. However, in this case, the system is told what object (razor) is present, and the task is to compute the pose of any instance of that object (razor) found in the image. The final example, Figure 1(e), shows a typical input image for the recognition system of Brooks [11]. From aerial images of an airport, the task is to not only locate any jet aircraft, but to determine what kind of aircraft they are, e.g., wide-bodied vs non-wide-bodied. The aircraft are assumed to be

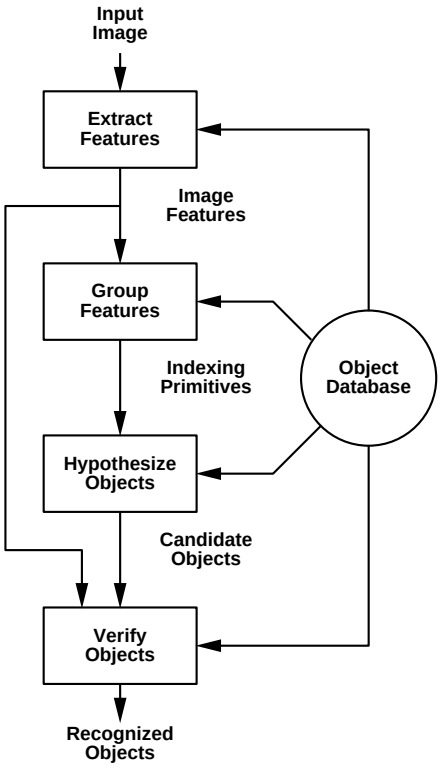


Figure 2: The Components of an Object Recognition System

unoccluded and the approximate viewpoint (both height and direction) is assumed to be known.

3 The Components of an Object Recognition System

The input to each of the recognition systems outlined above is a digital image, along with knowledge of one or more object models. The output is an object labeling and, in some cases, a specification of object pose. How, from a matrix of pixel values, is an object found? The typical framework for a recognition system is shown in Figure 2. From the input digital image, an appropriate set of features are extracted; common features include edges (brightness discontinuities), corners (edge intersections), and regions (homogeneous image patches). The goal of the feature extraction module is to take a large amount of image data and retain only that information necessary to identify or distinguish the object. Note that knowledge of what kinds of objects are present in the image may be used to govern the extraction of features. For example, if we know we're looking at aerial images of airports (e.g., Figure 1(d)), we may choose a feature extraction operator that looks for long lines in the image (corresponding to runway boundaries).

Object recognition can be simply thought of as a database search problem, in which search keys are used to retrieve records from the database. In our case, the database contains object models, not records, while the search keys are collections of extracted features. Returning to Figure 2, the extracted features must be grouped into meaningful collections, called indexing

primitives. Examples of indexing primitives include collections of extracted edges or lines, collections of corner features, or even collections of homogeneous image regions. An indexing primitive represents a query to the object database of the form, "Get me all objects in the database that have this primitive as a component." As in the case of feature extraction, knowledge of the image domain may be used to affect the feature grouping process.

Once such a query has been posed, a matching algorithm compares the indexing primitive to the object models in the database, returning a set of *candidate objects*, all of whom contain the indexing primitive. Unless the query returns only one candidate model, we're not yet finished, since we must decide which of the candidate objects we're looking at. Thus, the final component of the recognition algorithm evaluates, or *verifies*, each of the candidates in terms of how well it accounts for the image data. A score is typically assigned to each candidate and the best-scoring candidate, or hypothesis, is chosen as the interpretation (or label) of the object. If there are other objects in the image, the entire process can be repeated until all the image features are accounted for.

4 A Two-Dimensional Case Study

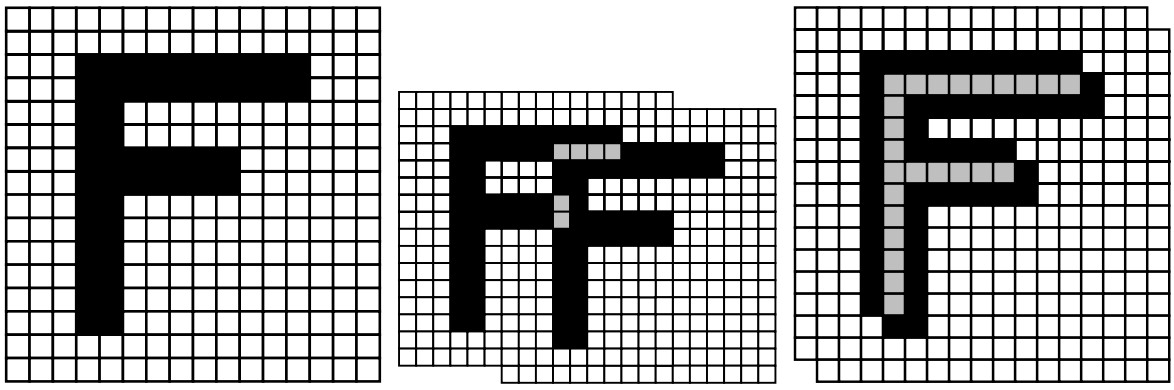
The backbone of any recognition system is its model object representation, for it governs what kinds of features are extracted, how features are grouped, and how features are matched to models. To illustrate the various components of an object recognition system and how the choice of object representation affects the design of each component, we will examine the design of a system whose goal is to take an image containing an alphanumeric character and recognize the character. This problem, known as *optical character recognition*, or OCR, is a classic problem in object recognition.

For a few hundred dollars, you can purchase software to run on your PC which will recognize all the characters on a scanned page and automatically insert them into a file. Why would you want such a tool? If you want to enter a page of text (e.g., magazine article, newspaper article, legal document, scientific article, etc.) into your computer, an OCR module frees you from manually typing the text. OCR systems have also been used in conjunction with speech synthesizers to provide reading machines for the visually impaired.

We will look at two solutions to the OCR problem. In the first case, we will assume that we know the font (including its point size) that will appear on the page, while in the second case, we assume no a priori knowledge of the font appearing on the page. We will call the first problem single-font recognition and the second problem omnifont recognition. The contrasting solutions to these two subproblems will help to illustrate a number of critical issues in choosing an object representation for a given recognition problem.

4.1 Single-Font Recognition

Given a digital image corresponding to an entire page of text, the first task is to *segment* the individual characters from the page image, resulting in a collection of character subimages, each containing a single character. In general, identifying the parts of an image that correspond to a single object, i.e., the segmentation problem, is a challenging problem. For our discussion, we will assume that some segmentation process has segmented the characters on



(a) (b) (c)

Figure 3: Template Matching for OCR: (a) model character template; (b) overlaying (with offset) the model template on the character subimage, with shaded pixels contributing to the score; (c) better alignment of model with image, reflecting a higher score (shaded pixels).

the page. This reduces our problem to identifying single characters, each centered in its own small subimage extracted from the entire page image.

Since we know the font and point-size of the characters on the page, we will choose a model character representation that reflects our specific knowledge of the image objects. The simplest such representation for a character would be an image of the character itself, called a *template*. The model template might be acquired by, for example, scanning in a page containing a single character repeated many times. The resulting subimages could be "averaged" to yield a representative subimage for that character. The ASCII label for that character would be stored with the model in the database. The process would then be repeated for each character in the font, resulting in a model database of character templates.

The matching of an image character with a model character is shown in Figure 3. The model character image is effectively overlaid on the input character image. For each pixel that is "on" in both images, the score of the match is incremented by one. To accommodate alignment error, the model character is overlaid in a number of different positions with the best-scoring position chosen as the best match. Finally, the best score is compared to a threshold which must be met in order for the character to be recognized. For example, if the score exceeds 90% of the "on" pixels in the character template, then at least 90% of the model character has been found in the input image.

The advantages of the template matching approach are clear. Looking back at our recognition framework in Figure 2, we have essentially ignored both the feature extraction step as well as the grouping step. Absolutely no abstraction or transformation of the input image is required in order to compare it with the model. Having such exact knowledge of the shape of the objects that can appear in the image means that the challenging problems of feature extraction and grouping can be avoided.

Until the mid 1980's, most commercial OCR systems worked on this principle, scanning and recognizing all the characters on a page in under 30 seconds. Template matching systems performed exceptionally well on both original documents, where error rates better than 1 in 100,000 characters were reported, and photocopied documents, where error rates better

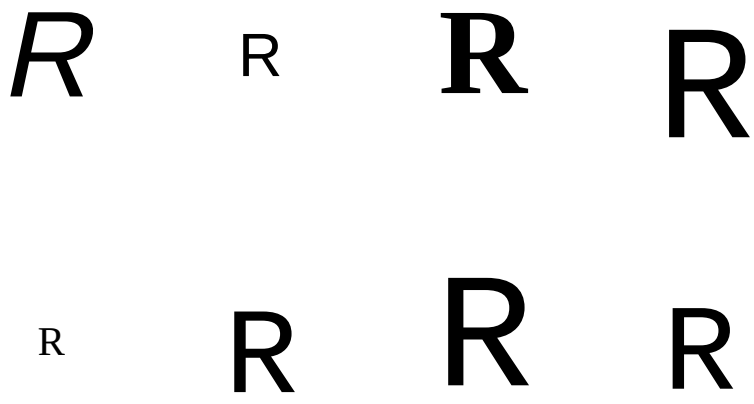


Figure 4: Different Instances of the Letter "R".

than 1 in 50,000 were reported. If, due to poor reproduction, a number of pixels in the input character image were inverted or lost (i.e., set to zero), the score was relatively unaffected, regardless of where the pixels were lost. Furthermore, the template matching algorithm could be easily implemented in hardware, resulting in recognition rates exceeding 50 characters per second.

The disadvantages of the template matching approach are also clear. The system works for a single font and will not recognize characters from other fonts or point sizes.[2] Thus, character templates are not invariant to scale, rotation, stroke thickness (due to boldface), slant (due to italics), or any shape deformation due to font change. Furthermore, the recognition complexity (or time required to recognize an input character) scales linearly with the number of fonts supported, resulting in slower recognition times as more fonts are added.

4.2 Omnifont Recognition

Each of the characters shown in Figure 4 are the same despite their differences in size, stroke thickness, slant, the presence of serifs (the small linear terminators on some lines), etc. What is it about each of these characters that gives them the same label? One possible description that is invariant to size, slant, stroke thickness, etc., is shown in Figure 5 (top) . The letter "R" can be represented as a closed section above and attached to a concave section facing south or down. These two strokes, and the relation defined between them, can be represented as a graph; nodes represent (possibly overlapping) portions of the character while arcs define relations between the nodes.

The simple representation for the letter "R" shown in Figure 5(top) is ambiguous in that both the letters "R" and "A" are described by the same graph; each has a closed contour above and attached to a concave contour facing downwards. In Figure 5(bottom), a more powerful representation is shown in which nodes in the graph represent sections of contour that have been cut at places where the direction of the contour suddenly changes or where

[2]Some template matching systems in the mid-1980's were able to accommodate as many as 12 different fonts by storing 12 sets of templates.

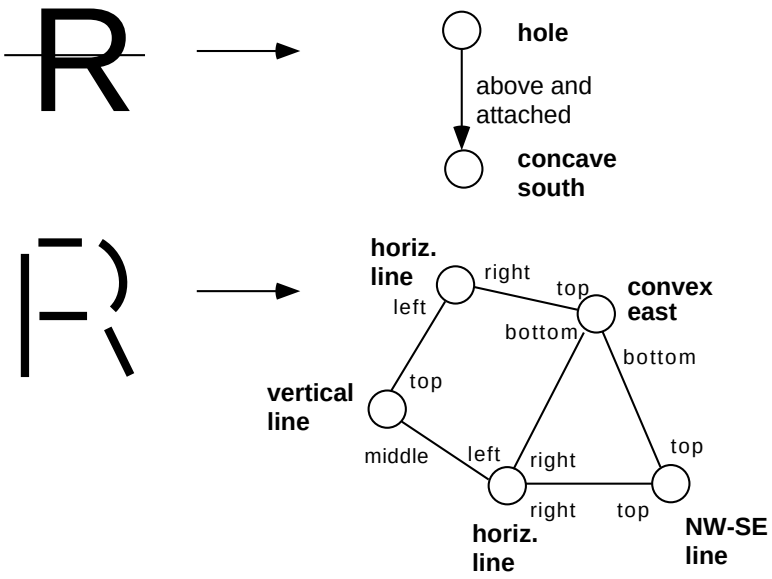


Figure 5: A possible description for the letter "R": (above) a simple, graph-based representation of the letter "R" which is ambiguous (letter "A" yields same graph); (below) more complex, graph-based representation of the letter "R" which is unambiguous.

the shape of the contour changes, e.g., straight to curved or vice versa.

Nodes in our new graph are labeled with a contour shape (e.g., line, convex curve, or concave curve) and a direction (e.g., horizontal, vertical, NW-SE, etc., for lines, and north, east, etc., for curves). The arcs, or relations, between nodes specify the attachments between the contour sections. For example, the horizontal line at the top of the "R" has a vertical line attached to its left end, while the vertical line has the horizontal line attached at its top. What we see here is that in order to make the representation of the "R" distinctive from that of the "A", we have had to add considerable complexity. The original "R" required two nodes and one arc, while the new "R" requires five nodes and six arcs. Storing the model for the enhanced "R" therefore requires more space or memory than the original "R".

Assuming that such a graph-based representation can be recovered from an image, it must be compared to the graph-based descriptions corresponding to the model characters. Just as we developed a method for comparing two template-based character descriptions, we now need a method for comparing two graph-based character descriptions. The method we will use to compare, or match, two graphs is known as interpretation tree (IT) search, as proposed by Grimson and Lozano-Pérez [24]. Although applied here to graphs representing characters, the method is quite general, supporting the matching of any description made up of features and constraints on the features. In fact, in some sense, the method can be seen as attempting to align two graphs just as we attempted to align two images in the template matching approach.

The method is illustrated in Figure 6. Assume that some preprocessing algorithm yields a graph of recovered image features, labeled I6 through I10. Shown below this graph is the graph describing the model character "R", labeled M1 through M5. Of course, the character

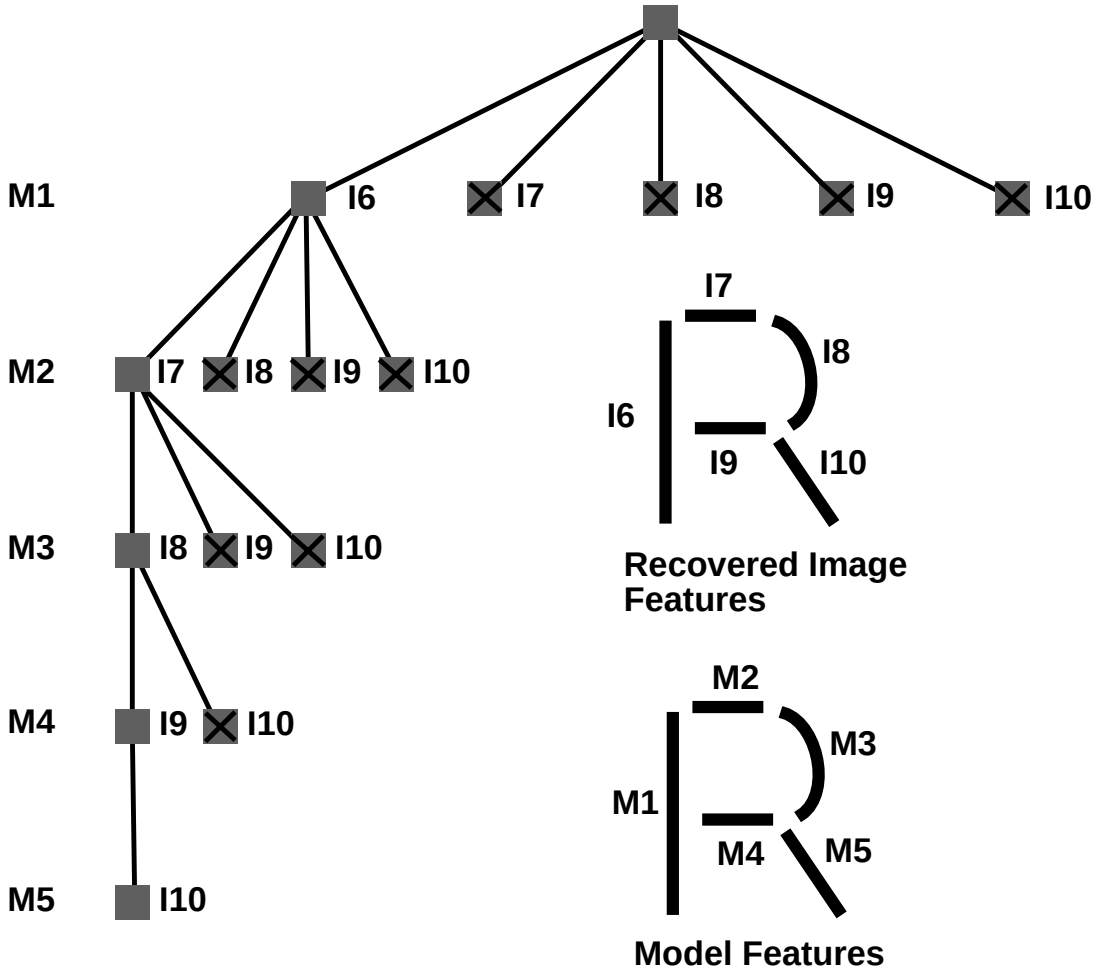


Figure 6: Using Interpretation Tree Search to Match two Character Descriptions.

defined by the input character graph is initially unknown and has to be compared to each of the model character graphs to find the best fit. For illustrative purposes, we show the process of comparing the input graph to the model graph corresponding to the letter "R". Since the input character is, in fact, an "R", we would expect the score of this comparison to be the maximum of all comparisons of the input character to a model character.

To begin the process of matching the two characters, we set up the interpretation tree whose number of levels (excluding the root of the tree) is equal to the number of nodes in the model character graph. The branching factor (number of children) of the tree's root is equal to the number of nodes in the input graph.[3] The first level of the tree represents the assignment of an input feature to the first model feature (M1). For example, the left-most branch defines the interpretation of input feature I6 as model feature M1.

M1 and I6 are features which have the same label, i.e., they are both vertical lines. I7 and M1, on the other hand, have different labels, i.e., one is a vertical line while the other is

[3]Conversely, we could have set the depth equal to the size of the input graph and the branching factor equal to the size of the model graph.

a horizontal line. Since these features are inconsistent, the interpretation of input feature I7 as model feature M1 is inconsistent, and the interpretation (node) is marked with an "x." The interpretation tree will therefore not be expanded below I7 at the first level since there is no interpretation of the input character as model character "R" with I7 corresponding to the vertical segment (M1) of the "R". Similarly, I8 cannot be interpreted as M1 because I8 is a convex curve and M1 is a vertical line. In fact, since there are no other vertical lines in the input graph, I6 is the only feature that can be interpreted as M1. The search below all the nodes in the tree other than I6 at the first level can therefore be terminated.

At the next level, we attempt to assign an input feature to M2. Since I6 is the only active tree node at the first level, we need only expand I6. Since I6 has already been interpreted as M1, I6 cannot also be interpreted as M2. Thus, we have one fewer children of I6, and therefore consider the interpretation of I7 through I10 as model feature M2. Examining the feature labels of I7 through I10, we see that both I7 and I9 are horizontal lines supporting their interpretation as M2. However, now that we have two nodes in our interpretation (I6 as M1 and, for example, I7 as M2), we need to make sure that the arcs between any assigned input nodes are consistent with the arcs defined over their matching model nodes. In this case, the arc between M1 and M2 is satisfied by the arc between I6 and I7, but not by the arc between I6 and I9 ("middle" connection instead of the correct "top" connection). In general, every time an input feature is added to the interpretation, any arcs between its corresponding model feature and other previously-defined model features must be consistent with the arcs defined by their corresponding image features.

We can proceed in this fashion, using both model node labels and arc labels to prune the search below inconsistent interpretations. The "better" an input graph matches a given model graph, the deeper the search will continue. In our example, since the input and model graphs exactly match, the interpretation tree extends to its maximum depth, correctly assigning an input feature to each model feature. One possible method of assigning a score would be to determine the percentage of model nodes and arcs that were found in the input graph – very similar to the scoring method proposed in our single-font OCR solution.

The advantages of the above omnifont recognition scheme are clear. A vertical line is vertical regardless of its length and thickness, while a convex curve facing east is convex as long as its radius is finite and the center of its circular approximation lies approximately to its left. In general, the node and arc labels are *invariant* to the size of the character (point size) and to minor deformations in the shape of the character. If we replaced absolute orientation with relative orientation in our model description, our description would be invariant to rotation of the character. In this manner, our description should be able to handle any input "R", provided that the graph belonging to the input "R" can be reliably recovered.

The reliable recovery of the input graph is, in fact, the major stumbling block of our omnifont method. It requires that we not only detect and extract the contours from our input character image, but that we correctly partition the contours into nodes and correctly label the relations (arcs) between the nodes. Errors in this challenging task can lead to graphs with more or less nodes (if too many or too few cuts are made), incorrect node labels (if orientations are miscalculated or borderline), or incorrect relations. These problems are compounded when there are gaps in the input character strokes. Although there are a number of clever extensions to the IT search method to handle partial matches of an input graph, the price of recovering invariant features is the difficulty and unreliability of their

recovery.

4.3 Selecting a Model for Matching

We have presented two contrasting recognition approaches to comparing an input character image to a model character. Given an input character description, we have not yet discussed how model characters are chosen for comparison. Do we simply compare each model, in turn, to the input character, selecting the best-scoring model in the end to represent the interpretation of the character? Or, can we avoid this linear search by somehow ranking the model characters for comparison in decreasing order of likelihood?

The problem of selecting a model for comparison is called *indexing* and is a critical component of a recognition system. Ideally, you would like to extract some clues from the input character description that allow you to suggest some likely candidate models to test. If no clues can be recovered, then a linear search of the model database is required – bad news indeed if the object model database is large. On the other hand, if you can recover some really distinguishing clues, you may have only one model to test, in which case you don't even need to test it!

The clues we're looking for are nothing but the grouped features that we saw in our recognition framework, shown in Figure 2. The purpose of grouping our features is to extract more discriminating meta-features that can serve as powerful indices into our database. We could, of course, simply use the features themselves as clues without grouping them. It may, in fact, be the case that the features themselves are powerful enough to suggest a small number of candidate models. The best approach would be a dynamic indexing scheme in which the complexity of the indexing feature would be a function of its indexing power.

Let's consider an example. In our template matching paradigm, our original feature is really nothing more than a pixel value, and our grouping operation is nothing more than selecting all pixels. If we choose one pixel as our index whose value is on, we would choose for comparison all model templates whose pixel at that location (or perhaps at a nearby location to account for positional error) was also on. Such an index is practically useless for two reasons. First, there are likely many characters which, when overlaid onto our input character, have many pixels in common. Second, due to noise, the input pixel may be incorrect (inverted or off), returning a set of candidate characters which don't occupy that position. In the end, you would likely be testing most if not all the model characters!

Before we give up on our one-pixel index, lets consider how we might generate a set of candidates from this index. If the input image was not a binary image (pixel values of 0 or 1) but rather a one-byte image whose values fall between 0 and 255, we might compile a table which, for each possible pixel value, contained a list of those images which contained a pixel of that value and where those pixels were located. This table could be indexed by the pixel value in a random-access fashion; for example, if the pixel value is 55, then go to row 55 of the table and see what character images contain a pixel whose value is 55. More complicated mappings from feature values to table locations are computed by *hash functions*.[4]

Given the weakness of our single-pixel index, we might decide to check a subimage of our

[4]For a discussion on how the topology of a 2-D silhouette can be compactly encoded for use as an index, see the work of Siddiqi et al. [48]

input character image. For example, we might choose a 10×10 subimage near the center of the character. This subimage, of size 100, is nothing more than a string of bits which could be interpreted as a single integer. This integer could be used to index to models which have that subimage embedded in them. The table would be precomputed off-line, so that it wouldn't cost any more to index with the subimage than with a single pixel. However, the size of the table grows rapidly with the size of the subimage.

Returning to our omnifont approach, we could build a table that contains all models having an eastward-facing convex curve or a vertical line. Or, we could use feature groups as more powerful indices. For example, our table could store all models which have a vertical line with a horizontal line extending to the right from the top of the vertical line (top-left of "R"). If only a few characters have an eastward-facing convex curve, then it doesn't make much sense to spend the time grouping additional contours with it to form a more powerful index. On the other hand, if many characters share a vertical line (which is, in fact, true), then it does make sense to add an additional line to the group to make the index more powerful.

What we have here is an important trade-off between the cost and reliability of grouping features into more powerful indices and the cost of testing or verifying more candidate models. As databases grow in size, the need for better indices (hence, better grouping) is clear. However, as we discussed before, more complex features (or feature groups) are more prone to error, making effective indexing more difficult. This optimization can be computed on the fly: once a feature is extracted, index into the database to see how many models contain it. If there are too many, then go back and increase the scope of the indexing feature through further grouping until a manageable number of candidate models is returned.

4.4 A Final Note on Our Two Approaches

There is a very important trade-off illustrated by our two character recognition schemes. In the template matching approach, the model is overconstrained, specifying the exact shape of the character down to where each pixel is. With such a strong model, little, if any, feature extraction is necessary. Our omnifont description, on the other hand, is a more abstract description of the character's shape, offering invariance to scaling, orientation, and even some deformation. However, matching an abstract description to image data means recovering a comparable abstraction from the image. Recovering an abstract description from an image is the most important problem facing today's object recognitions systems.

Historically, OCR systems relied on the template matching approach. However, as mentioned earlier, this approach gave way to the omnifont approach in the mid 1980's, when low-cost, reliable feature extraction approaches were developed. As computing power increased, more and more features could be extracted in the allowable time frame, leading to font-invariant software systems that can be run on a PC. As we will see in the next section, when we return to 3-D object recognition, the evolution from template-like approaches to more abstract, generic approaches is proceeding at a much slower pace. Since the mid-1980's, the computer vision community has been preoccupied with approaches that assume knowledge of the exact geometry or appearance of an object.

5 Adding the Third Dimension

Our case study in character recognition has revealed the issues in selecting an object representation for a recognition task. How many models are in the database? How much is known about the shapes of the input objects? And how reliable are the extracted features and feature groupings? When moving to three dimensions, the issues are identical. In addition, we must consider the problem of occlusion, in which a nearer object obstructs the view of an object further from the camera. Accommodating occlusion means being able to both index and match using only partial information of the occluded object. It also means making sure that your indexing features come from the same object which, in turn, means that your grouping module not group features from different objects.

The real problem in recognizing a three-dimensional object is that although the object is three dimensional, the image and its extracted features and feature groups are two-dimensional. As the viewpoint with respect to the object changes, so do the image features. For example, the front of a car looks different from the back, the top, or the side. In response to this phenomenon, two schools of 3-D recognition have emerged. In the *viewer-centered* approach, the 3-D object is modeled as a set of 2-D images, one for each different "appearance" of the object. In this way, 3-D recognition is reduced to 2-D recognition, except that a single 3-D object may consist of a large set of 2-D views. The complexity or size of the resulting object database is a serious concern and makes effective indexing even more important.[5]

In the *object-centered* approach, a single 3-D model is used to describe the object. Although providing a much more compact and efficient representation than the viewer-centered approach, we now have somehow to compare our 2-D extracted features to 3-D features making up the models. Since the model features look different in the image depending on where they are viewed from, we need to choose features whose appearance doesn't depend on viewpoint. If we could find such viewpoint-invariant features, they would give us a powerful means by which a 2-D image feature could be linked to a 3-D model feature.

What are these viewpoint-invariant features? For example, if two 3-D lines intersect in the world, then from almost all possible viewpoints, the two intersecting lines will appear as two intersecting lines in the image. Only when the camera lies in the plane defined by the two lines is the viewpoint-invariant feature (intersecting lines) lost; in this case, the two lines appear as a single line in the image. This allows us to hypothesize, with great confidence, that if we see two intersecting lines in the image, we're really looking at two 3-D lines intersecting in the world.

In fact, there are a suite of viewpoint-invariant features that have been used by computer vision researchers, most of which were proposed by the Gestalt psychologists, e.g., [55]. These viewpoint-invariant features include 3-D lines and curves (which project to 2-D lines and curves), parallel 3-D lines and curves (which project to parallel 2-D lines and curves), collinear 3-D lines (which project to 2-D collinear lines), and co-curvilinear 3-D lines (which project to co-curvilinear 2-D lines). An excellent discussion of these viewpoint-invariant features can be found in [34]. Other geometric invariants, such as the cross-ratio of any 4 points on an ellipse, have been used to index into databases of object models [38].

[5]Plantinga and Dyer have shown that a polyhedral object with n faces has an aspect graph (structured collection of views) whose number of views is $O(n^9)$, implying an explosive growth in the number of 2-D models in the database [42]. Articulated objects only compound the problem.

At this point, let's recap what we've discovered. We've seen that there is a trade-off between the power of complex indexing features to discriminate objects and the difficulty with which such features can be reliably recovered from an image. We've also seen that more abstract model representations can be used to describe more than one particular object exemplar, e.g., a single "R" chosen from a particular font. Finally, we've discussed two approaches to 3-D model representation offering yet another trade-off: the potentially large number of views needed to reduce a 3-D model to a 2-D model versus the problem of inferring a compact 3-D model's features from 2-D image features. Armed with this insight, let's go back and see how the recognition problems in Figure 1 were solved.

5.1 A Viewer-Centered Approach Using Pixels

In the approach of Murase and Nayar [39], each image is represented as a point in a high dimensional space. Imagine laying all the rows of the image side by side until the entire image is a $1 \times n$ vector, where n is the number of pixels in the image. Murase and Nayar take the viewer-centered approach to object modeling that models a 3-D object as a set of views. Each of these views is an image of the object taken from a different viewpoint, as shown in Figure 7, and each image becomes a vector, as described above. You may ask why we need a set of different views of the object, and why one view won't suffice? The reason is that as you move the camera around the object, its appearance may change dramatically. If we're going to store an object as a collection of views, we need to make sure that we have one view for every possible appearance of the object.

Imagine now that we have a database of objects, with each object modeled as a collection of vectors (images). If we're presented with an arbitrary view of an unknown object and asked to identify the object (and perhaps the viewpoint at which the image was acquired), our task is to find the "closest" image in our database. One simple approach would be to compute the vector distance between the input image vector and all the vectors in the database, choosing the closest database vector as the object's identity. However, this would be a very computationally expensive procedure, since each vector has n components, where n is the size of the image in pixels.

Murase and Nayar's approach is an extension of a very clever approach proposed by Turk and Pentland [53]. Consider an n-dimensional space, and let each image in the database be a point in that space. It turns out that for the resulting cloud of points, there is a more convenient coordinate system with which to represent the points. Although this coordinate system has the same number of dimensions as the original one, it has the property that the positions of the points can be sufficiently approximated by a small number of the coordinates (e.g., ≤ 20 instead of 16K, for the images in Figure 7). This new coordinate system is defined by the eigenvectors of the covariance matrix derived from the cloud of points. Furthermore, these eigenvectors can be prioritized according to their corresponding eigenvalues. The vectors that have high eigenvalues represent more definitive axes or coordinates in our new coordinate system.

Each view of a model object can now be represented in this new coordinate system using only a small number of coordinates. This is an inexpensive process called projection. As we move around the object, its different views will trace out a curve in this new coordinate system, Since each object looks different, each object will have its own characteristic curve in

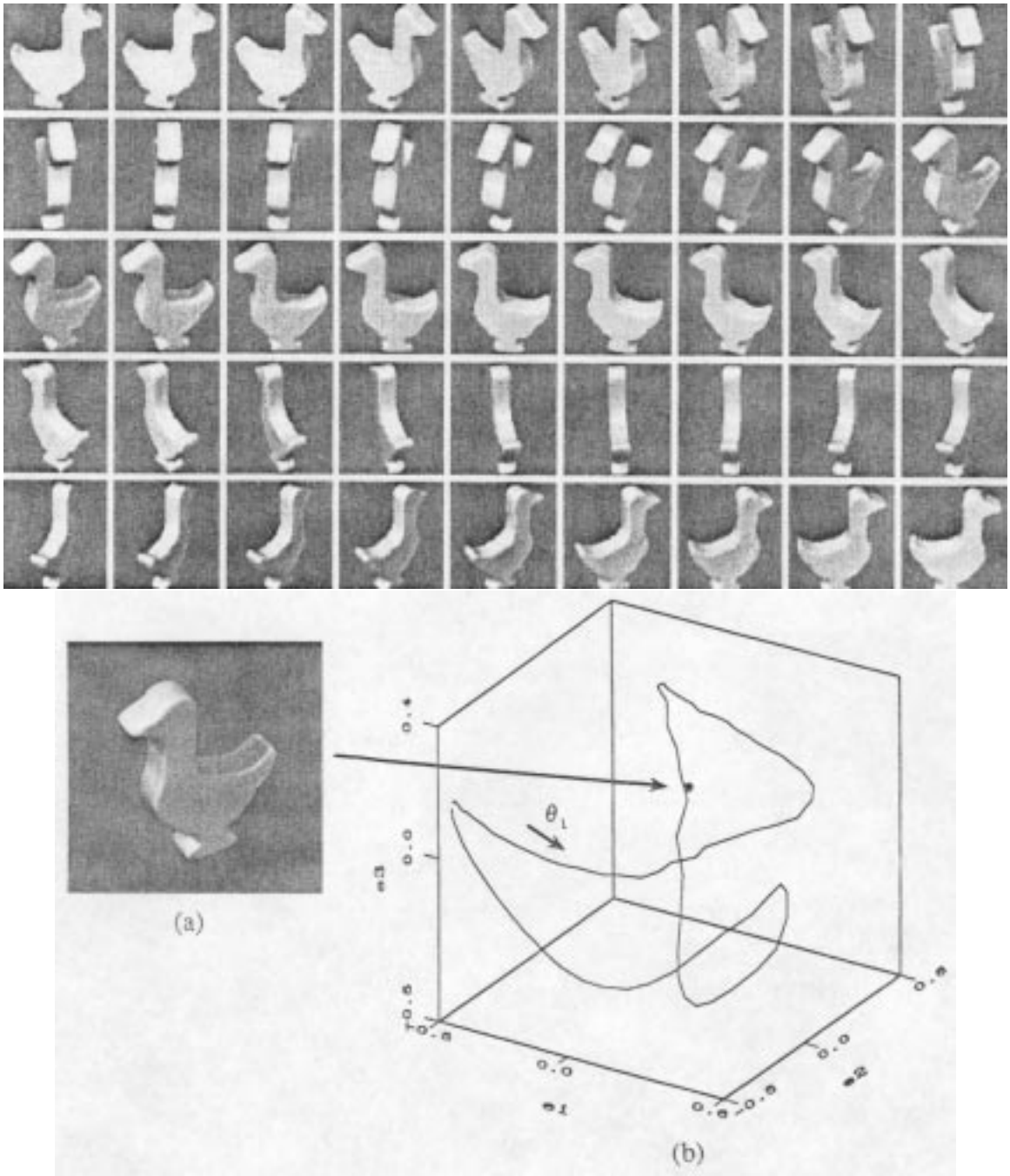


Figure 7: A View-Based Image Representation: (top) A dense set of views is acquired for each object; (bottom) The views trace out a curve in a low-dimensional space, with each view lying on the curve.

this coordinate system. The database now becomes this new space (coordinate system) and the objects become curves passing through this space. Murase and Nayar therefore propose a two-step procedure for recognizing a view of an object. First, project the view into the new coordinate system. Next, find the closest (traced-out) object curve in the coordinate system to determine the object's identity. Once the object's identity is known, we can move to a new coordinate system defined only by the views of that particular object. We again project the view on this object-specific coordinate system and find the nearest point on the curve to determine the object's pose (viewpoint), as shown in Figure 7.

Murase and Nayar's system is very impressive and can recognize and estimate the pose of unoccluded objects in real-time. Given a database of objects, some of which are shown in Figure 8 (top), the result of the recognition approach applied to the image shown in Figure 1(a) is shown in Figure 8(bottom). Some limitations of the approach should be noted, however. For example, if the object's shape changes slightly, or the texture or markings on the object change, or the lighting changes appreciably, or there are other objects in the scene, the approach is likely to fail. Although a number of researchers have extended the technique to provide limited invariance to these effects (e.g., [3, 32, 46, 47]), this technique draws its power from the fact that the object description is strictly local and therefore somewhat brittle. It is ideally suited to the recognition of object exemplars and not object categories or prototypes. The approach therefore resembles the template matching approach we saw in our single-font OCR example.

5.2 An Object-Centered Approach Using Corners

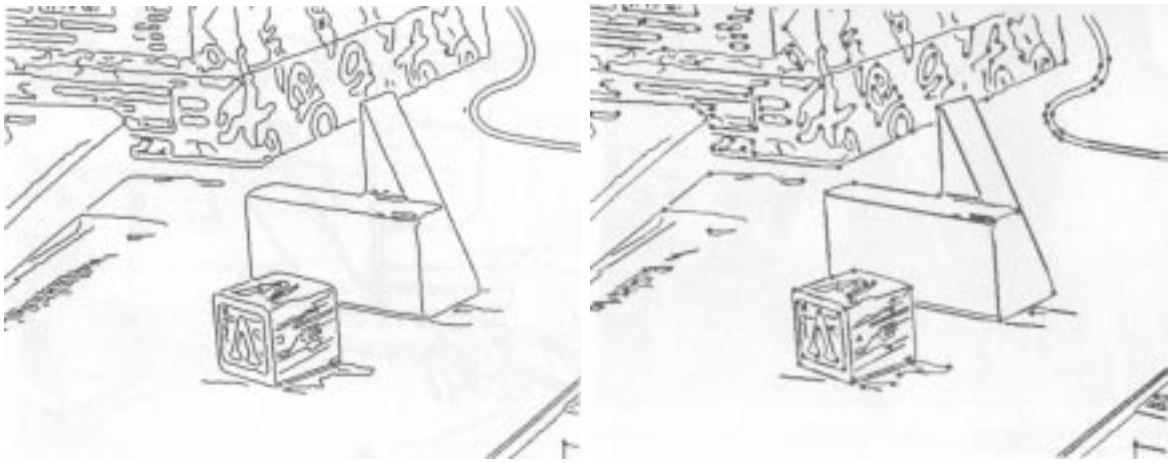
As seen in the previous subsection, Murase and Nayar took a viewer-centered approach to object modeling that reduced the recognition problem from 3-D to 2-D at the expense of requiring many views for each object. The alternative approach to object modeling (and recognition) attempts to model a 3-D object using a 3-D object-centered model that is invariant to viewpoint. Since the pioneering work of Roberts [45], many researchers have taken this approach; we will illustrate one such approach – that of Huttenlocher and Ullman [29].

In the object-centered approach to object recognition, we are faced with the problem that our input image is two-dimensional, while our object models are three dimensional. Somehow, we must establish correspondence between 2-D features in the image and 3-D features on the model. If enough corresponding features can be found, it might lead us to believe that we've selected the right model. What are these features that will allow us to link 2-D image information with 3-D model information?

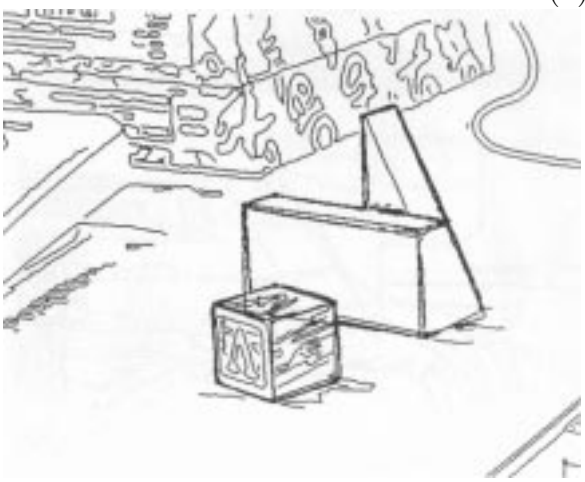
Let's back up a step and ask ourselves what desirable qualities we might look for in such features. First of all, a good feature on the object must not only be visible in the image, but easily and reliably recovered from the image. Second of all, a feature should be visible over a wide range of viewpoints. If you can see it only while looking from a particular viewpoint, the feature will rarely appear in the image and will therefore rarely serve to identify the object. The feature should also be sufficiently local, such that if you occlude part of the object, the feature will still be visible. If the feature is occluded, then we need enough features on the object to choose from so that the odds of not seeing any of the features are small.

The features that Huttenlocher and Ullman chose in their system are a popular choice

Figure 8: Final Recognition Results (bottom) from an Object Database (top).



(a) (b)



(c)

Figure 9: Recognizing a polyhedral Object Based on Hypothesizing Point-Based Correspondences: (a) Extracted Edges; (b) Extracted Corners (significant changes in curvature); (c) Detected Objects and Their Poses.

that balances viewpoint invariance, locality, and ease of recovery. Given the input image in Figure 1(b), they begin by extracting edges in the image, as shown in Figure 9(a). Edges are connected sets of points which correspond to positions in the image where brightness changes abruptly. The idea is that due to light reflecting in different directions off differently oriented surfaces of an object, the amount of light that each surface reflects towards the camera will be different. Depending on where the illumination source is, the different surfaces will have different brightness.

Along the seams of these surfaces, there will be a brightness discontinuity where the pixel values jump significantly. Detecting these discontinuities allows us to infer that on the object, we are looking at two different surfaces on either side of the line. This assumption, of course, is somewhat optimistic, for a painted line on the object will give rise to a line in the image which should not be interpreted as a surface discontinuity. Vision researchers will therefore often work with textureless objects or assume that enough of the lines in the image

correspond to surface (i.e., shape) boundaries on the object.

Once the edges are extracted, we could, for example, try and match edges in the image with surface boundaries on the model. Moreover, we might try to match long edges in the image with long boundaries on the model. The problem is that the length of the surface boundary, as it appears in the image, changes depending on viewpoint. If the surface boundary is heavily foreshortened, it corresponding image edge will be very small. If the boundary is facing the camera, the edge will be maximally long. In choosing edges, we've reduced our image data considerably, we've chosen features that are abundant on an object (particularly a polyhedral object), and our features are somewhat local (if you cover up part of the object, some edges should survive). The problem is that the edges (i.e., their lengths) are not viewpoint invariant.

Huttenlocher and Ullman rely on the fact that the places where two boundaries intersect *is* stable with changes in viewpoint. Consider two 3-D lines that meet at a corner. As discussed earlier, with the exception of viewing the two lines in the plane defined by the two lines, the intersection of those two lines will appear as two intersecting edges in the image. Thus, from the extracted edge maps, Huttenlocher and Ullman look for edge corners, i.e., places where the edge changes direction significantly. The underlying assumption is that these corners in the image correspond to vertices on the model (places where three or more surfaces meet). In Figure 9(b), the detected corners have been marked with very small dark circles.

Now we're faced with the problem of deciding which corners in the image correspond to which vertices on the model. Since all the corners and vertices look pretty much the same, let's simply choose three of the image corners and three of the model corners and assume that the correspondence is correct. Huttenlocher and Ullman exploit the fact that for their particular projection model (scaled orthographic), there is a unique transformation (translation, rotation, and scaling) of the model that will bring those three model vertices into alignment with the three image corners. Consider, for example, an image with a circle in it. Someone tells you that you're looking at a pencil. In order to verify that this statement is true, you mentally rotate a pencil in space until you're looking along the length of it. You then say to yourself, "Yes, if I was looking along the end of the pencil, I would, in fact, see a circle!". However, the circle in the image is somewhat large. To resolve this, you figure that either the pencil is very large, or it's very close to your eye.

The problem here is that practically any three (mutually visible) vertices on the model can be aligned with practically any three corners in the image. How do we know when we have the correct correspondence? The final step in the procedure is known as hypothesis verification, where the transformed model is "overlaid" into the image. We know that the three vertices and corners will line up. But if the other vertices and surface boundaries in the model don't line up with their corresponding corners and edges in the image, our hypothesized correspondence must have been wrong. What do we do in this case? We can keep the three image corners and pair them up with three new model vertices. Or, we can keep the three model corners and pair them up with three new image corners. In the end we'll eventually find a correct assignment of 3 image corners to 3 model corners, as confirmed by our verification step. In Figure 9(c), the correctly transformed model has been aligned with the image.

The Huttenlocher and Ullman approach offers some powerful advantages over the Murase

and Nayar approach, but suffers from some disadvantages as well. To begin with, the features are local, stable, and viewpoint-invariant. The approach can handle occlusion, changes in lighting, and changes in object scale (in the image). However, the features are not unique, forcing us to try all possible assignments of image features to model features. In fact, each polyhedral model in a database is made up of the same features, so we may have to repeat the process using every object model. Since the Murase and Nayar approach is image based, the objects can have rich markings on them, which would simply add extraneous corners to the current approach. The Huttenlocher and Ullman approach can be thought of as a slightly more abstracted recognition system than that of Murase and Nayar. Although the Huttenlocher and Ullman approach offers more transformation invariance than that of Murase and Nayar, it is still designed primarily for exemplar-based recognition. Relying on matching corners to vertices assumes that the object model is a geometrically exact duplicate to the object contained in the image. One could view the approach as a form of template matching where, rather than storing each particular template, the template is generated on the fly from a 3-D model, using some viewpoint-invariant clues.

5.3 An Object-Centered Approach Using Perceptual Groups

One way of dealing with the computational complexity of the Huttenlocher and Ullman approach (i.e., the high number of corner-vertex triple correspondences that need to be verified) is to somehow make the features themselves more discriminating. What other features are local, viewpoint-invariant, and easy to recover, but carry more information to reduce the number of correspondences that have to be tried? We turn to a very important approach proposed by Lowe [34], which took advantage of the powerful visual inferences identified by the Gestaltists.

Returning to the approach by Huttenlocher and Ullman, you recall that corner-vertex correspondences were chosen because when seeing a corner in the image, you can infer, with very high confidence, that you're seeing a corner in the world. In an earlier section, we discussed many such powerful inferences that are made by the human visual system. For example, parallel edges in the image suggest parallel lines in the world, collinear edges in the image suggest collinear lines in the world, and close proximity of two edges' endpoints in the image suggests that the two lines in the world have endpoints in close proximity. The grouping of images features according to parallelism, symmetry, collinearity, and proximity, is called *perceptual grouping*, and is an active area of research in both human and computer vision.

Lowe exploited the notion of perceptual grouping in order to reduce the size of the search space. We can follow his approach applied to the image in Figure 1(c). In Figure 10(a), Lowe begins by extracting edges from the image. Next, as shown in Figure 10(b), he looks for pairs of edges that are parallel in the image. The idea here is that the number of parallel edge groups is likely to be much smaller than the number of, for example, corners. A given extracted pair of parallel edges is then paired with a set of parallel lines that bound a single surface on the model. A transformation and verification step similar to that used by Huttenlocher and Ullman is needed to finally recognize the object, as shown in Figure 10(c).

By grouping lower-level features, such as edges, into more complex groups, such as parallel

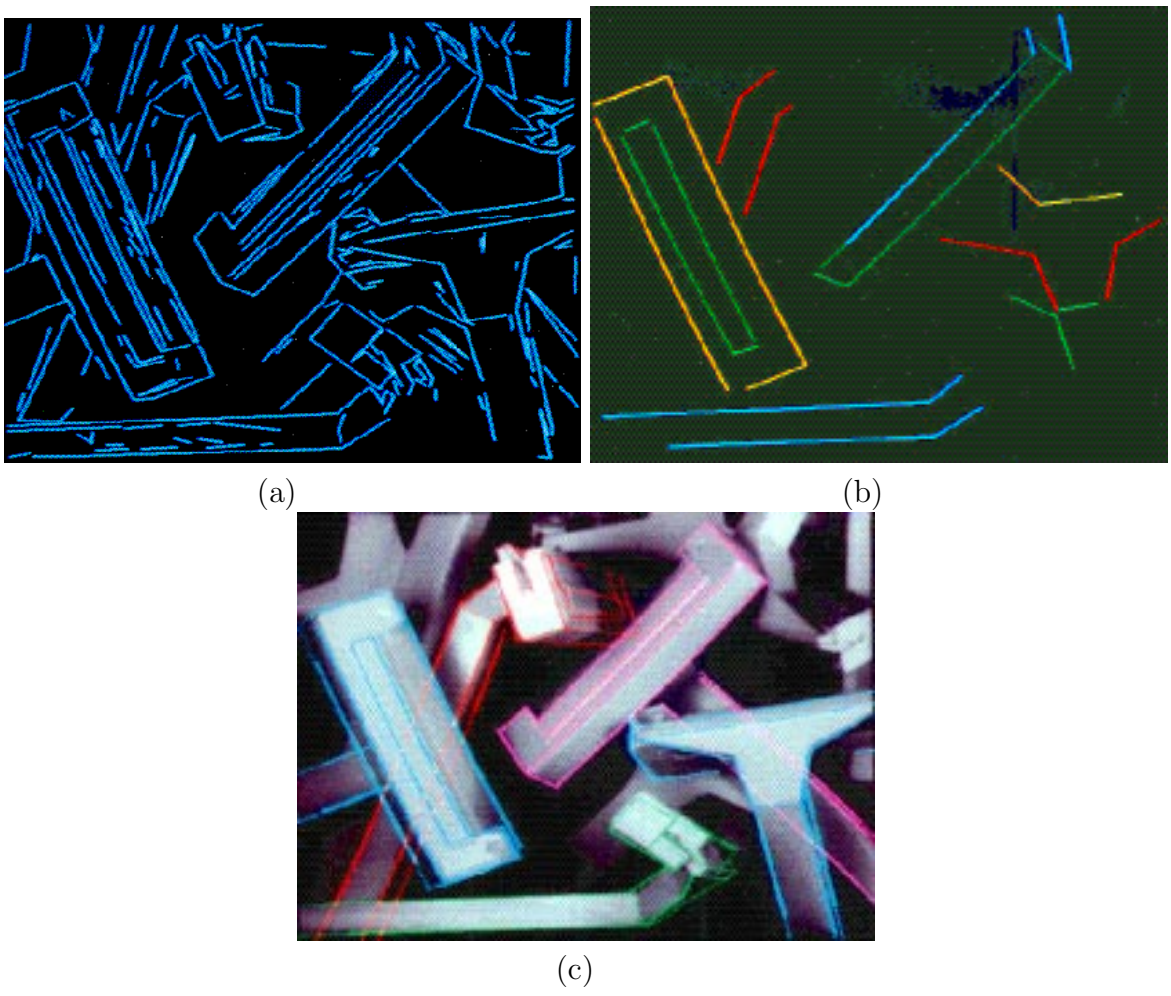


Figure 10: The Use of Perceptual Grouping to Prune Hypothesized Correspondences in 3-D Object Recognition: (a) Extracted Edges; (b) Extracted Perceptual Groups; and (c) Detected Objects and Their Poses.

edges, makes the resulting features more discriminating, leading to a more efficient search. It is also important to draw a powerful distinction between a triple of image corners and a perceptual group, e.g., a pair of parallel lines. If the scene is cluttered with many objects, then a given triple of corners will often contain corners arising from the vertices of different objects (although there are heuristics to reduce this effect). Such a triple is doomed, and the time expended to transform and verify the model is wasted. There is really no meaningful relationship among the edge corners, as they are purely local. The perceptual groups, on the other hand, are causally related in that they are very likely to belong to the same surface or boundary [56].

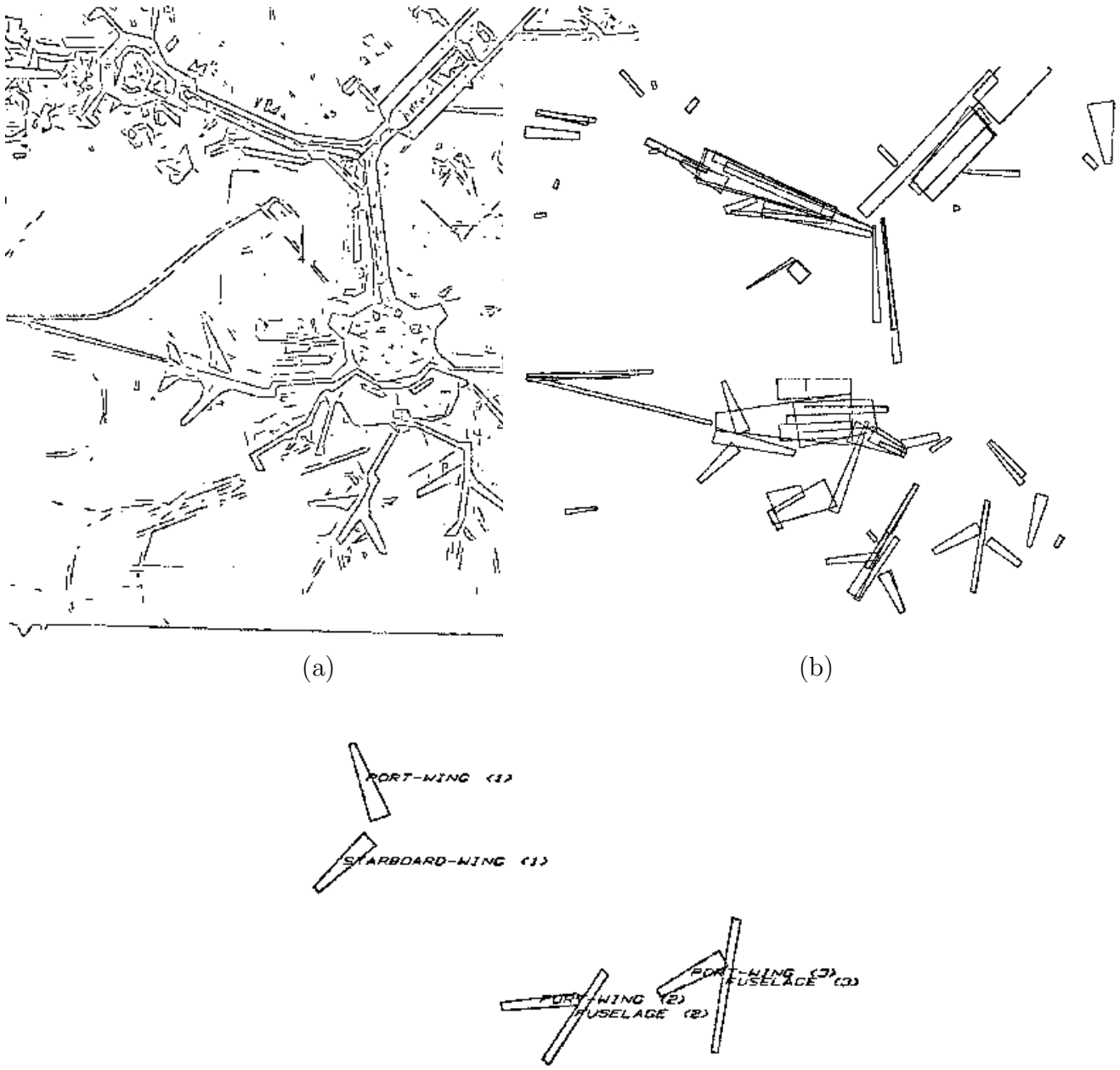
Despite the increased power of Lowe's approach, it was still applied only to polyhedral objects and it still relied on a one-to-one correspondence between image edge groups and model line groups. Like the other two approaches, it is suited to exemplar-based recognition, and cannot, in its present form, extend to more generic or prototypical object recognition. Like the Huttenlocher approach, it can be viewed as a form of template matching where the template is generated at run-time from a 3-D model. It differs from the approach of Huttenlocher and Ullman in that it will more efficiently arrive at the correct template, at the cost of having to detect additional complex relations among features. Despite these limitations, the use of perceptual grouping to form meaningful feature groups and to improve search efficiency was an important contribution to the computer vision community.

5.4 An Object-Centered Approach Using Volumes

The three approaches we've seen so far are very rigid in their insistence that the geometry (or appearance) of the model exactly mimic the geometry (or appearance) of the object being imaged. An example of a more flexible system is Brooks' ACRONYM system [11], which allowed some degree of object parameterization. Rather than modeling objects using polyhedra, Brooks chose to model his objects using constructions of volumetric parts called generalized cylinders [9, 1, 40, 36]. His modeling strategy allowed the user to parameterize the number of parts, the sizes and shapes of the parts, and the relative positions and orientations of the parts. This parameterization took the form of sets of constraints assigned to the parts of a model, allowing a single model to vary considerably within a class. Of the four systems we've seen so far, Brooks' system is the only one that can aspire to a single model that can describe all the coffee cups in your kitchen cupboard; the other systems would need separate appearance or geometric models for every differently-shaped cup in your cupboard.

Brooks applied his system to the recognition of wide-bodied aircraft in airport scenes, as shown in Figure 1(d). As shown in Figure 11(a), Brooks began by extracting edges from the image. Like Lowe's system, Brooks decided that since he was looking for elongated parts belonging to an object, these parts would appear as "ribbons" in the image, or pairs of lines that likely correspond to the occluding boundaries of a volumetric part. Figure 11(b) shows the extracted ribbons from the edge image.

The constraints on the model's volumetric parts and their inter-relations were mapped, using a complex constraint manipulation system, to corresponding constraints on the sizes, positions, and orientations of the ribbons extracted from the image. The object database consisted of hierarchically-defined models whereby coarse, prototypical models near the top of the hierarchy had weak constraints while models further down had stronger constraints,



(a) (b)

(c)

Figure 11: Recognizing 3-D Objects by Searching for their Volumetric Parts and their Part Relations: (a) Extracted Edges; (b) Extracted Ribbons; and (c) Recognized Objects.

specifying exemplars. In Brooks' case, aircraft were broken into wide-bodied and non-wide-bodied aircraft which, in turn, were broken into particular instances, e.g., Boeing 747. The results of labeling the airport scene are shown in Figure 11(c).

Brooks' ACRONYM system is admirable in its aspiration to model and recognize objects at different levels of abstraction. However, it has its limitations. For example, it was designed primarily as a target recognition system, e.g., "Find the planes in the image," rather than "What objects are in the image?". The power of the approach was never fully explored, although generalized cylinder modeling and recovery has survived (e.g., [54, 58]). The generalized cylinders used were heavily restricted as was the viewpoint. Although the constraint manipulation system was very powerful, it was somewhat cumbersome. Despite these shortcomings, Brooks' system was a very important contribution to the area of generic object recognition.

6 Indexing Primitive Trade-offs

In the previous section, we examined four approaches to object recognition based on successively more complex indexing features. We began with the Murase and Nayar approach which used an image's coordinates in a low-dimensional eigenspace to index into a database of model object views. These features were based on an image's pixel values and required no feature grouping or abstraction. Next, we saw how Huttenlocher and Ullman used triples of viewpoint-invariant curvature discontinuities (corners) to solve for a polyhedral model's model's pose with respect to the image. Lowe's approach went one step further than Huttenlocher and Ullman's by exploiting causal relationships among features using principals of perceptual grouping. Lowe's perceptual groups were fewer in number and richer in description than the corner triples used by Huttenlocher and Ullman. However, as they represent more complex features, they are harder to reliably recover from an image. Finally, we saw how Brooks attempted to generically model objects by parameterizing the geometries of their volumetric parts and the relations between the parts. The ribbons recovered from an image (corresponding to projections of volumetric parts) were the most complex features recovered of the four systems studied.

A comparison of object recognition systems according to their indexing primitives is given in Figure 12. In the left column are various indexing primitives ranging in complexity from low (e.g., 2-D points) to high (e.g., 3-D volumes), as depicted by the width of the leftmost bar (bar 1). Some of the indexing primitives are two-dimensional, while others are three-dimensional, often reflecting the type of input as intensity or range image data. Accompanying each indexing primitive is a reference to an example system that employs that primitive. Note that this list of indexing primitives is not complete; it is meant only to exemplify the range in complexity of possible indexing primitives.

Working from left to right in Figure 12, we see that as the complexity of indexing primitives increases, the number of features making up the object models decreases (bar 2), since an object can be described by a few complex parts or by many simple parts. This, in turn, implies that the search complexity, i.e., the number of hypothesized matches between image and model primitives, decreases with increasing primitive complexity (bar 3). The high search complexity involving simple indexing primitives is compounded by large object

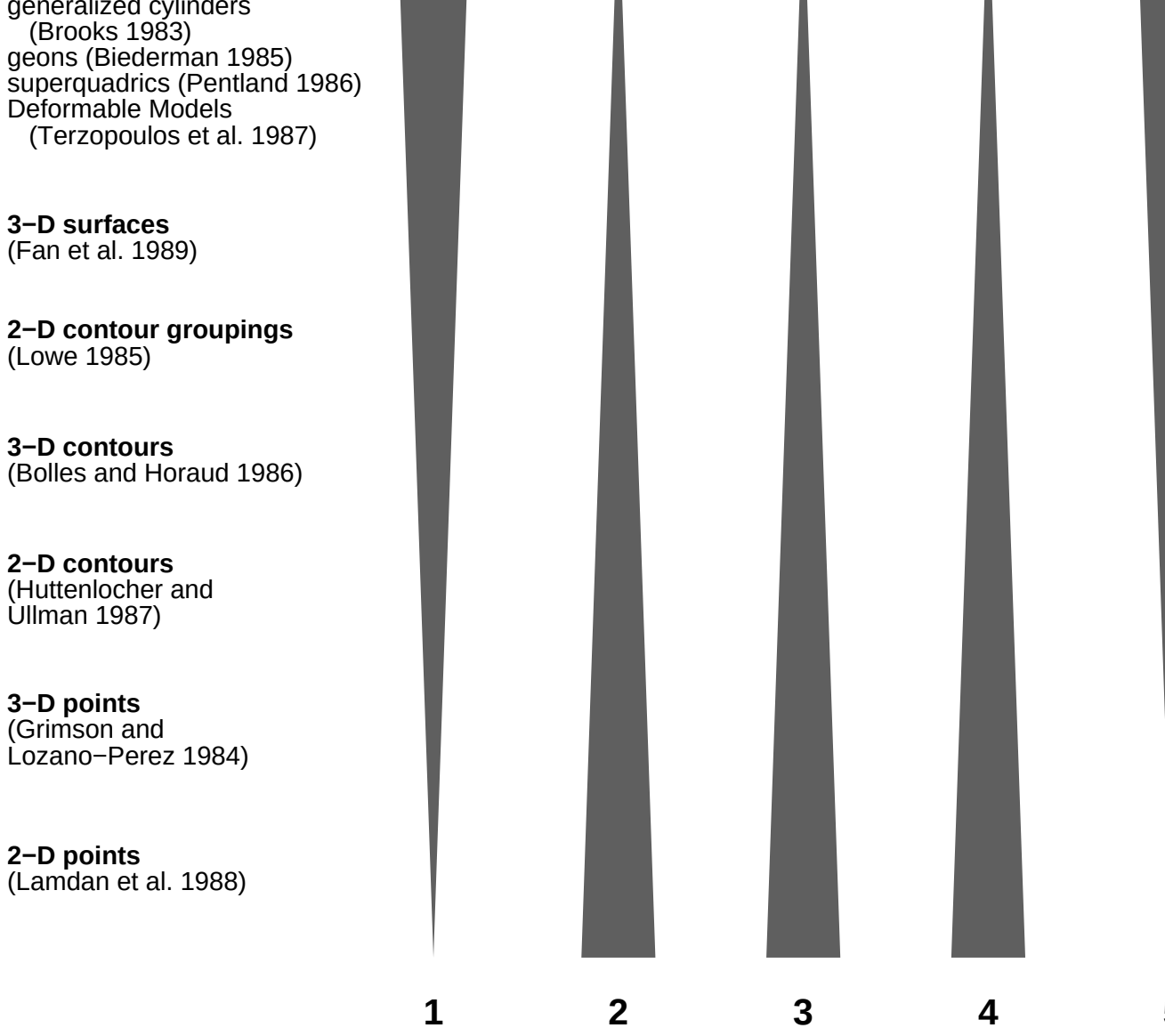


Figure 12: Trade-Offs in Choosing Indexing Primitives

databases. As a result, most systems using simple indexing primitives, e.g., Lowe [34], Huttenlocher and Ullman [29], Thompson and Mundy [52], and Lamdan et al. [31], are applied to small databases typically containing only a few objects.

Since the simple indexing primitives represent a more ambiguous interpretation of the image data (e.g., a few corners in the image may correspond to many corner triples on many objects), systems that employ simple primitives must rely heavily on a top-down verification step to disambiguate the data (bar 4). In this manner, the burden of recognition is shifted from the recovery of complex, discriminating indexing features to the model-based verification of simple indexing features. Since many different objects may be composed of the same simple features, these systems are faced with the difficult task of deciding which object to use in the verification step. However, there is a more fundamental problem with simple indexing features.

Relying on verification to group or interpret simple indexing primitives has two profound effects on the design of recognition systems. First, verifying the position or orientation of simple indexing features such as points or lines requires an accurate determination of the object's pose with respect to the image. If the pose is incorrect, searching a local vicinity of the image for some model feature may come up empty. Needless to say, accurately solving for the object's pose can be computationally complex, particularly when a perspective projection camera model is used.

Relying on verification also affects object modeling. Specifically, the resulting object models must specify the exact geometry of the object, and are not invariant to minor changes in the shape of the object (bar 5). Consider, for example, a polyhedral model of a chair. If we stretch the legs, broaden the seat, or raise the back, we would require a new model if our verification procedure were checking the position of points and lines in the image. Indexing with simple primitives restricts the object database to models whose exact geometry is known. Excellent work has been done to extend these techniques to certain types of parameterized models, e.g., Grimson [23], Huttenlocher [27], and Lowe [35]. However, by nature of the indexing primitives, these models do not explicitly represent the gross structure of the object, and therefore cannot easily accommodate certain types of shape changes.

Before leaving the issue of a model's sensitivity to shape changes, it is interesting to note that the same trade-offs arose in our OCR case study. In the template or image matching approach, no significant primitive extraction is performed. Instead, rigid models (image templates) are moved across the image until a match is found; if a model changes slightly, a new template is needed. In the feature matching approach, more complex features are extracted from the image, e.g., strokes and shape properties, and compared to structural character models. Since the object models (characters) capture gross structure, they can be used to recognize characters from many different fonts and point sizes.

So far, bars 1 through 5 in Figure 12 clearly indicate the advantages of using complex indexing features over simple ones. Why then are most 3-D from 2-D recognition systems using simple indexing primitives?[6] And why is the computer vision community moving away from more generic descriptions to more exact descriptions (in fact, the four 3-D systems presented in this chapter from less generic to more generic were, in fact, developed in decreasing

[6]Many of the more complex indexing primitives, e.g., 3-D surface patches, deformable models, and superquadrics, are typically recovered from range data images.

chronological order)? First of all, simple indexing primitives have proven to be quite successful in certain domains, e.g., typical CAD-based recognition, in which the object database is very small, object models are constructed from simple primitives, object shape is fixed, and exact pose determination is required. However, more importantly, the reliable recovery of more complex features, particularly from a single 2-D image, is a very difficult problem (bar 6), particularly in the presence of noise and occlusion. Clearly, the major obstacle in the path of any effort to build a recognition system based on complex indexing primitives will be the reliable recovery of those primitives.

7 Recognition By Parts: An Emerging Paradigm

In the previous section, we saw how increasing primitive complexity offered a number of recognition advantages at the expense of increased difficulty of recovery. At the upper end of the primitive complexity spectrum lies volumetric primitives, an example of which we saw in examining Brooks' ACRONYM system. Brooks chose generalized cylinders as his basic volumetric primitive – a representation specified by three arbitrary functions: cross-section, axis, and sweep-rule. Since recovering a volumetric part from an image means recovering its defining parameters, a simpler part with fewer parameters is therefore easier to recover. To cope with the unbounded complexity of generalized cylinders (the three functions can be arbitrarily complex), Brooks assumed that his generalized cylinders had straight axes and rotationally symmetric cross-sections.

In the mid-1980's, two vision researchers, one from the human vision community and the other from the computer vision community, introduced to their respective communities two competing volumetric part representations. In the human vision community, Biederman introduced a volumetric shape vocabulary known as geons [7], while in the computer vision community, Pentland introduced a shape representation known as superquadrics [41].[7] Both geons and superquadrics are restricted versions of Binford's generalized cylinders, attempting to capture a rich variety of volumetric shape with a much smaller number of parameters. In the following subsections, we will contrast these competing approaches, outlining both their strengths and their weaknesses.

7.1 Superquadrics

A superquadric can be thought of as a lump of clay subject to stretching, bending, twisting, and tapering deformations. The superquadric with length, width, and breadth a_1, a_2, and a_3 is described (adopting the notation $\cos\eta = \mathbf{C}_\eta$, $\sin\omega = \mathbf{S}_\omega$) by the following equation:

$$\mathbf{X}(\eta,\omega) = \begin{pmatrix} a_1\mathbf{C}_\eta^{\epsilon_1}\mathbf{C}_\omega^{\epsilon_2} \\ a_2\mathbf{C}_\eta^{\epsilon_1}\mathbf{S}_\omega^{\epsilon_2} \\ a_3\mathbf{S}_\eta^{\epsilon_1} \end{pmatrix} \tag{1}$$

[7] Superquadrics, or more formally, superquadric ellipsoids, were originally conceived by the Danish architect Hein [22], brought to the computer graphics community by Barr [2], and brought to the computer vision community by Pentland [41] and Solina and Bajcsy [49].

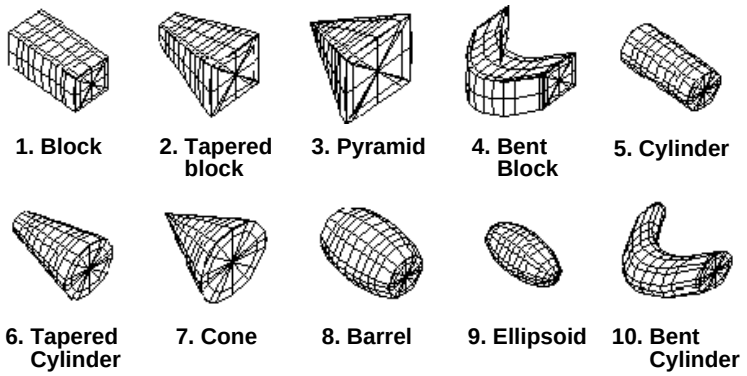


Figure 13: Some Examples of Deformable Superquadric Ellipsoids

where $\mathbf{X}(\eta, \omega)$ is a three-dimensional vector that sweeps out a surface parameterized in latitude η and longitude ω, with the surface's shape controlled by the parameters ϵ_1 and ϵ_2. Additional parameters can be added to provide tapering, bending, twisting, pinching, etc. Even with these additional deformation parameters, the resulting deformable superquadric can be specified in some 10-20 parameters. Some examples of deformable superquadrics are shown in Figure 13.

Superquadrics are appealing in that they not only capture a rich diversity of shape with a small number of parameters, but that they capture a human's intuitive notion of shape and deformation. As mentioned above, the parameters of a superquadric can be thought of as a set of intuitive deformations on a lump of clay. For example, the parameters define dimensional stretching of the clay, bending and tapering of the clay, and shaping of the clay to be smooth or faceted. Pentland argued that these parameters would be ideal for modeling the volumetric parts that make up many classes of objects in our environment. His introduction of superquadrics to the computer vision community spawned a great deal of subsequent work, particularly in the recovery of superquadrics from laser rangefinder (3-D) data, e.g., [25, 50, 37, 33, 43, 21, 57, 14, 13]).

7.2 Geons

Biederman took a different approach to reducing the complexity of generalized cylinders. Adhering to the three functions defining a generalized cylinder, he proposed to heavily restrict the three functions and add a fourth. The cross-section function would be specified as a binary-valued function that could take on the value of either straight-edged or curved-edged. Similarly, the axis function would be specified as a binary-valued function that could take on the value of either straight or curved. The cross-section sweep function would have three possible values: constant, expanding, or expanding followed by contracting. Biederman added a fourth function describing the symmetry of the cross-section. The cross-section function takes on one of three values: rotationally symmetric, reflectively symmetric, or asymmetric. By permuting the values of these properties, Biderman arrived at his vocabulary of 36 geons, as shown in Figure 14.

Biederman argued that the four definitive properties could be easily and quickly re-

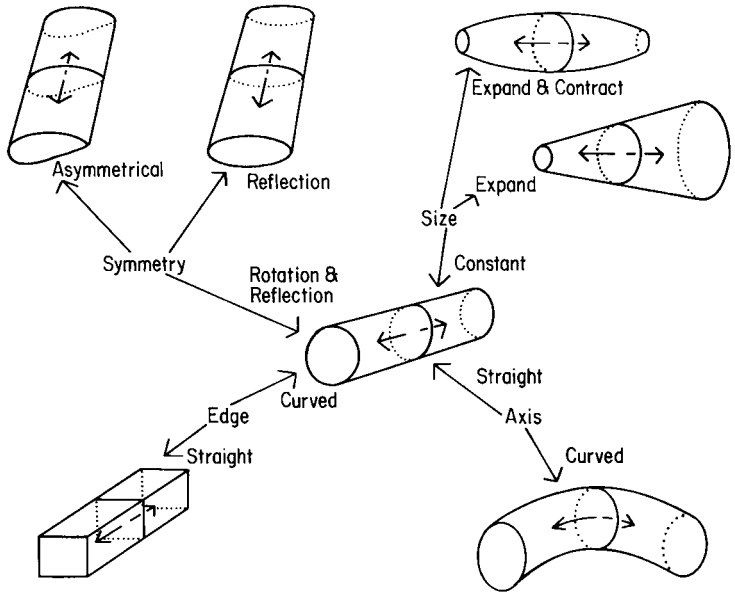


Figure 14: Biederman's Geons

covered from viewpoint-invariant properties of image contours. Geons offer a qualitative abstraction of shape whose 36 categories are invariant to minor deformations of shape. For example, the fact that a cylinder is bent is critical whereas the degree to which it is bent is not. Biederman's introduction of geons to the computer vision community motivated many researchers to build computer vision systems based on Biederman's geons, e.g., [57, 8, 26, 5, 4, 6, 18, 15, 17, 16, 12, 19, 43, 44, 30].

7.3 What's Missing?

Both superquadrics and geons represent a restricted form of generalized cylinders. Although not as powerful as generalized cylinders, they nevertheless offer a rich vocabulary with which to construct objects. The challenge has been in their recovery from image data. Superquadric recovery has been mainly restricted to laser rangefinder data, where the number of 3-D image data points greatly exceeds the number of parameters that must be recovered, overconstraining the shape fitting problem. For 2-D images, the data points to which the superquad can be fitted lie only along the gradient discontinuities or edges. These points are too poorly distributed along the surface of the superquadric to guarantee a unique solution for the recovered shape parameters. Hence, superquadric recovery from image data has met with very little success.

Even if superquadrics could be reliably recovered from image data, they still provide no shape categorization required for recognition. Somehow, the parameter space of the superquadric must be carved up into hypervolumes that correspond to some set of shape classes, e.g., [57]. Once these classes are known, a part label can be assigned to a recovered

superquadric, and this label, along with the labels of adjacent parts, can be used as an index into the object database. The part labels, or classes, can be used as a coarse-level index, while the sizes and relative orientations of the parts can be used to prune the candidates having the same coarse part structure.

Although geons provide a qualitative shape description, they capture no metric shape information. A description of two geons does not specify their absolute or relative orientation, their absolute or relative size, or for example, how curved or tapered they are (assuming that they are curved or tapered). Granted, Biederman's goal was a qualitative representation for distinguishing between different classes of objects; however, such information is essential for interacting with the object and for distinguishing between subclasses of an object.

Geons also represent a somewhat arbitrary choice of qualitative shape properties and their dichotomous and trichotomous properties. Why four properties and not three or five? Why tapering and not twisting? Why curved vs. straight cross-section shape, and not not some combination of both? Although Biederman's choice of shape properties and their values are well-motivated, the computer vision community has been reluctant to embrace them. Part of the problem has been the computer vision community's inability to recover geons from real images of real objects [12].

8 The Road Ahead: Generic Object Recognition

The key problem facing object recognition is shape abstraction. Although a particular image feature, e.g., line, curve, or region, may be salient in the *image*, it may not be salient in the *world*. For example, the edges separating the stripes on a coffee cup may have very high contrast and may yield the "best" edges in the image. However, in terms of a generic coffee cup model (for example, a handle attached to the side of a cylinder), such edges play absolutely no role. Any object recognition paradigm that assumes that such edges have corresponding edges on the model cannot recognize objects based on their prototypical shape. In such a paradigm, object models must mimic the exact structure of the objects appearing in the image. If three different coffee cups appear in an image, each with differently oriented stripes, a separate model will be required for each cup.

Deriving generic shape representations that capture the coarse, prototypical shape of an object is well within our grasp. Both superquadrics and geons, or example, are quite suitable for this task, with each capturing the definitive part structure of an object. However, in order to index into object models with such parts requires that such parts be recovered from the image. Complex objects with textured surfaces give rise to a plethora of edges or regions in the image. Somehow, these edges or regions must not only be grouped into larger structures, they must be abstracted to form "meta-regions" which correspond to the projected abstract surfaces on the object. This daunting task is seldom addressed in the computer vision community.

This chapter has sought to shed some light on the problem of representing and recognizing objects. To the designers of computer vision systems and to those modeling the human visual system, the issues are the same. What are the relevant image features? How are they grouped or abstracted? How are model objects represented, and how is indexing performed? Is recognition more bottom-up, more top-down, or a combination of both. We have explored

some of these issues through some illustrative examples; may references are provided for further study. Object recognition is a fascinating, multidisciplinary topic, offering many exciting avenues for further research.

References

[1] G. Agin and T. Binford. Computer description of curved objects. *IEEE Transactions on Computers*, C-25(4):439–449, 1976.

[2] A. Barr. Superquadrics and angle-preserving transformations. *IEEE Computer Graphics and Applications*, 1:11–23, 1981.

[3] P. Belhumeur and D. Kriegman. What is the set of images of an object under all possible lighting conditions. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 270–277, San Francisco, CA, June 1996.

[4] R. Bergevin and M. D. Levine. Extraction of line drawing features for object recognition. *Pattern Recognition*, 25(3):319–334, March 1992.

[5] R. Bergevin and M. D. Levine. Part decomposition of objects from single view line drawings. *CVGIP: Image Understanding*, 55(1):73–83, January 1992.

[6] R. Bergevin and M. D. Levine. Generic object recognition: Building and matching coarse 3d descriptions from line drawings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15:19–36, January 1993.

[7] I. Biederman. Human image understanding: Recent research and a theory. *Computer Vision, Graphics, and Image Processing*, 32:29–73, 1985.

[8] I. Biederman, J. Hummel, P. Gerhardstein, and E. Cooper. From images edges to geons to viewpoint invariant object models: A neural net implementation. In *Proceedings, SPIE Applications of Artificial Intelligence X: Machine Vision and Robotics*, pages 570–578, Orlando, FL, 1992.

[9] T. Binford. Visual perception by computer. In *Proceedings, IEEE Conference on Systems and Control*, Miami, FL, 1971.

[10] R. Bolles and P. Horaud. 3DPO: A three-dimensional part orientation system. *The International Journal of Robotics Research*, 5(3):3–26, 1986.

[11] R. Brooks. Model-based 3-D interpretations of 2-D images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(2):140–150, 1983.

[12] S. Dickinson, R. Bergevin, I. Biederman, J.-O. Eklundh, A. Jain, R. Munck-Fairwood, 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.

[13] S. Dickinson and D. Metaxas. Integrating qualitative and quantitative shape recovery. *International Journal of Computer Vision*, 13(3):1–20, 1994.

[14] S. Dickinson, D. Metaxas, and A. Pentland. The role of model-based segmentation in the recovery of volumetric parts from range data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(3):259–267, March 1997.

[15] S. Dickinson, A. Pentland, and A. Rosenfeld. A representation for qualitative 3-D object recognition integrating object-centered and viewer-centered models. In K. Leibovic, editor, *Vision: A Convergence of Disciplines*. Springer Verlag, New York, 1990.

[16] S. Dickinson, A. Pentland, and A. Rosenfeld. From volumes to views: An approach to 3-D object recognition. *CVGIP: Image Understanding*, 55(2):130–154, 1992.

[17] S. Dickinson, A. Pentland, and A. Rosenfeld. 3-D shape recovery using distributed aspect matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):174–198, 1992.

[18] L. Du and R. Munck-Fairwood. Geon recognition through robust feature grouping. In *Proceedings, 9th Scandinavian Conference on Image Analysis*, Uppsala, Sweden, June 1995.

[19] R. Fairwood. Recognition of generic components using logic-program relations of image contours. *Image and Vision Computing*, 9(2):113–122, 1991.

[20] T. Fan, G. Medioni, and R. Nevatia. Recognizing 3-D objects using surface descriptions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(11):1140–1157, November 1989.

[21] F. Ferrie, J. Lagarde, and P. Whaite. Darboux frames, snakes, and super-quadrics: Geometry from the bottom up. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 15(8):771–784, 1993.

[22] M. Gardiner. The superellipse: a curve that lies between the ellipse and the rectangle. *Scientific American*, 213:222–234, 1965.

[23] W. Grimson. Recognition of object families using parameterized models. In *Proceedings, First International Conference on Computer Vision*, pages 93–100, London, UK, 1987.

[24] W. Grimson and T. Lozano-Pérez. Model-based recognition and localization from sparse range or tactile data. *International Journal of Robotics Research*, 3(3):3–35, 1984.

[25] A. Gupta. Surface and volumetric segmentation of 3D objects using parametric shape models. Technical Report MS-CIS-91-45, GRASP LAB 128, University of Pennsylvania, Philadelphia, PA, 1991.

[26] J. Hummel, I. Biederman, P. Gerhardstein, and H. Hilton. From edges to geons: A connectionist approach. In *Proceedings, Connectionist Summer School*, pages 462–471, Carnegie Mellon University, June 1988.

[27] D. Huttenlocher. Three-dimensional recognition of solid objects from a two-dimensional image. Technical Report 1045, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, 1988.

[28] D. Huttenlocher and S. Ullman. Object recognition using alignment. In *Proceedings, First International Conference on Computer Vision*, pages 102–111, London, UK, 1987.

[29] D. Huttenlocher and S. Ullman. Recognizing solid objects by alignment with an image. *International Journal of Computer Vision*, 5(2):195–212, 1990.

[30] A. Jacot-Descombes and T. Pun. A probabilistic approach to 3-D inference of geons from a 2-D view. In *Proceedings, SPIE Applications of Artificial Intelligence X: Machine Vision and Robotics*, pages 579–588, Orlando, FL, 1992.

[31] Y. Lamdan, J. Schwartz, and H. Wolfson. On recognition of 3-D objects from 2-D images. In *Proceedings, IEEE International Conference on Robotics and Automation*, pages 1407–1413, Philadelphia, PA, 1988.

[32] A. Leonardis and H. Bischoff. Dealing with occlusions in the eigenspace approach. In *Proceedings, IEEE Conference on Computer Vision and Pattern Recognition*, pages 453–458, San Francisco, CA, June 1996.

[33] A. Leonardis, F. Solina, and A. Macerl. A direct recovery of superquadric models in range images using recover-and-select paradigm. In *Proceedings, Third European Conference on Computer Vision (Lecture Notes in Computer Science, Vol 800)*, pages 309–318, Stockholm, Sweden, May 1994. Springer-Verlag.

[34] D. Lowe. *Perceptual Organization and Visual Recognition*. Kluwer Academic Publishers, Norwell, MA, 1985.

[35] D. Lowe. Fitting parameterized three-dimensional models to images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(5):441–450, 1991.

[36] D. Marr and H. Nishihara. Representation and recognition of the spatial organization of three-dimensional shapes. *Royal Society of London*, B 200:269–294, 1978.

[37] D. Metaxas and D. Terzopoulos. Shape and nonrigid motion estimation through physics-based synthesis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(6):580–591, June 1993.

[38] J. Mundy and A. Zisserman, editors. *Geometric Invariance in Computer Vision*. MIT Press, Cambridge, MA, 1992.

[39] H. Murase and S. Nayar. Visual learning and recognition of 3-D objects from appearance. *International Journal of Computer Vision*, 14:5–24, 1995.

[40] R. Nevatia and T. Binford. Description and recognition of curved objects. *Artificial Intelligence*, 8:77–98, 1977.

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

[42] H. Plantinga and C. Dyer. Visibility, occlusion, and the aspect graph. *International Journal of Computer Vision*, 5(2):137–160, 1990.

[43] N. Raja and A. Jain. Recognizing geons from superquadrics fitted to range data. *Image and Vision Computing*, 10(3):179–190, April 1992.

[44] N. Raja and A. Jain. Obtaining generic parts from range images using a multi-view representation. *CVGIP:Image Understanding*, 60(1):44–64, July 1994.

[45] L. Roberts. Machine perception of three-dimensional solids. In J. Tippett et al., editors, *Optical and Electro-Optical Information Processing*, pages 159–197. MIT Press, Cambridge, MA, 1965.

[46] C. Schmid and R. Mohr. Combining greyvalue invariants with local constraints for object recognition. In *Proceedings, IEEE Conference on Computer Vision and Pattern Recognition*, pages 872–877, San Francisco, CA, June 1996.

[47] A. Shokoufandeh, I. Marsic, and S. Dickinson. View-based object matching. In *Proceedings, IEEE International Conference on Computer Vision*, page to appear, Bombay, January 1998.

[48] K. Siddiqi, A. Shokoufandeh, S. Dickinson, and S. Zucker. Shock graphs and shape matching. In *Proceedings, IEEE International Conference on Computer Vision*, page to appear, Bombay, January 1998.

[49] F. Solina and R. Bajcsy. Recovery of parametric models from range images: The case for superquadrics with global deformations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(2):131–146, 1990.

[50] D. Terzopoulos and D. Metaxas. Dynamic 3D models with local and global deformations: Deformable superquadrics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(7):703–714, 1991.

[51] D. Terzopoulos, A. Witkin, and M. Kass. Symmetry-seeking models and 3D object recovery. *International Journal of Computer Vision*, 1:211–221, 1987.

[52] D. Thompson and J. Mundy. Model-directed object recognition on the connection machine. In *Proceedings, DARPA Image Understanding Workshop*, pages 93–106, Los Angeles, CA, 1987.

[53] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.

[54] F. Ulupinar and R. Nevatia. Perception of 3-D surfa¡ces from 2-D contours. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15:3–18, 1993.

[55] M. Wertheimer. Laws of organization in perceptual forms. In W. Ellis, editor, *Source Book of Gestalt Psychology*. Harcourt, Brace, New York, NY, 1938.

[56] A. Witkin and J. Tenenbaum. On the role of structure in vision. In J. Beck, B. Hope, and A. Rosenfeld, editors, *Human and Machine Vision*. Academic Press, New York, 1983.

[57] K. Wu and M. Levine. Recovering parametric geons from multiview range data. In *Proceedings, IEEE Conference on Computer Vision and Pattern Recognition*, pages 159–166, Seattle, WA, June 1994.

[58] M. Zerroug and R. Nevatia. Volumetric descriptions from a single intensity image. *International Journal of Computer Vision*, 20(1/2):11–42, 1996.