# A Relational Pyramid Approach to View Class Determination

Haiyuan Lu, Linda G. Shapiro, and Octavia I. Camps

Department of Electrical Engineering
University of Washington
Seattle, WA 98195

*Index terms:* 3-D Representation and Recognition, Relational Matching, View Class, CAD Model

## ABSTRACT

Given a CAD model of an object, we would like to automatically generate a vision model and matching procedure that can be used in robot guidance and inspection tasks. We are building a system that can predict features that will appear in a 2D view of a 3D object, represent each such view with a hierarchical, relational structure, group together similar views into view classes, and match an unknown view to the appropriate view class to find its pose. In this paper, we describe the *relational pyramid* structure for describing the features in a particular view or view class of an object, the *summary structure* that is used to summarize the relational information in the relational pyramid, and an accumulator-based method for rapidly determining the view class(es) that best match an unknown view of an object.

## 1. Introduction

CAD-model-based vision is a growing research area in which vision models and matching procedures are automatically constructed from CAD models of objects and knowledge of the environment in which the vision-related task is to be performed. CAD-model-based systems are extremely useful for industrial vision tasks where a number of different manufactured parts must be automatically manipulated and/or inspected. Another area where vision systems based on CAD models is becoming important is in the United States space program. Since the space station and space vehicles are recent or even current designs, we can expect to have CAD models of these objects to work with. Vision tasks in space such as docking and tracking of vehicles, guided assembly tasks, and inspection of the space station itself for cracks and other problems can rely on model-directed vision techniques.

CAD-to-vision research is going on at a number of research institutions. Henderson and Bhanu (1986) are building a CAD-model-based-vision system that uses their local Alpha1 CAGD modelling system, which produces complex B-spline surface models, as its input. They convert the CAGD models to an intermediate representation consisting of a sampling of surface points and normals at these points. This representation is then used to match against 3D range data. Ikeuchi (1987) has constructed a system whose purpose is to enable a robot to find an object in a bin of parts and to grasp it. Starting from CAD data, he automatically generates an interpretation tree, a decision tree that can be used to determine the view class of an object (two views belong to the same class if the same surfaces are visible in each) and then to determine the exact pose within that class. The features to be used in the tree are selected automatically, but from a fixed set of features with a fixed ordering. The three-dimensional data that is input to the decision tree analysis comes from dual photometric stereo.

Ponce *et al.* (1987) are working with generalized cylinder models of objects. As part of their work, they have studied the contours of generalized cylinders under orthographic projection, proved properties of the contours that are invariant to viewing direction, and used these to implement recognition algorithms for finding the axes. Narasimhamurthi and Jain (1988) are working on a CAD-based system to recognize objects from range data. They have concentrated, in the early stages of the work, on the role of features in model-based recognition. Finally, Kak *et al.* (1988) have used the PADL-2 solid modeling system to derive boundary models of objects and matched these to 3D data derived from a structured lighting technique using relational matching techniques.

We are building a computer vision system that uses CAD models and other knowledge about the objects it will view to generate vision models and vision algorithms. The system has two major subsystems: one for offline processsing and one for online processing. The offline processing subsystem handles the task of converting each of the CAD models, which were meant to be used for design and display of objects, to a vision model and a procedure that can be used to recognize, inspect, or manipulate the object. The online processing uses the vision model and procedure to perform the inspection/guidance task. Our research has four main aspects: 1) prediction of images features from object models, 2) representation of the features predicted for a given view and their interrelationships, 3) generation of view classes by a clustering procedure, and 4) matching an unknown view to the representatives of the view classes derived from the clustering. This paper addresses aspects 2) and 4): representing view classes and rapidly identifying the view class of an unknown view of an object.

## 2. Features and Their Representation

A feature is an entity that can be extracted from a digital image of an object and that helps us to recognize the object or to determine its view class. A feature is useful if it can be economically and reliably extracted from the image (or predicted on the basis of other features that provide evidence of its existence) and if it helps differentiate one view class (or object) from another. Primitive features such as line segments and corners are commonly used when dealing with gray-tone image data. Since it is difficult or impossible to recognize individual line segments or corners from local operations, the spatial relationships among the primitive features form the basis for the global recognition of the entire object or view class. This global recognition usually involves relational matching procedures which can be computationally expensive.

Instead of using simple relationships among all the primitive features, we can select more complex features that are either unique or appear only a small number of times in a view of the object. For example, Bolles and Cain (1982) constructed a system which, in a training phase, selected

CH2813-4/89/0000/0177/$01.00 1989 IEEE

focal features and constructed higher-level features consisting of the focal features plus several nearby features and the relationships among them. We agree with the spirit of this endeavor; a higher-level feature, if it can be detected, is much more useful for rapid view class (or object) recognition than a primitive.

### 2.1 The Relational Pyramid

In our CAD-to-vision system, the prediction system will generate the features that appear in a given view, and those views that produce similar features will be grouped together as one view class. In order to decide if two views are similar enough to be grouped together, we need a representation for the features and their interrelationships. This representation should be simple enough that the primitive features can be easily accessed and complicated enough that powerful high-level relationships can be represented. This suggests a pyramid structure where simple primitives are represented at the bottom level, and the succeeding levels represent more and more complex relationships among the primitives. Thus the view depicted by this structure can be dealt with at any level of complexity desired. The structure is formally defined as follows.

Let $F$ be a set of detectable primitive features. Each feature $f \in F$ has an associated type $T_f$ and a vector of attributes $A_f$. A *relational pyramid* of height $h$ over feature set $F$ is a sequence of $h$ relational descriptions $(D_0, D_1, \ldots, D_{h-1})$. Description $D_0$ is a sequence of $n_0$ relations $< R_0^0, \ldots, R_{n_0}^0 >$, each relation representing one of the primitive types. A pair $(f, A_f)$ belongs to relation $R_i^0$ if $f \in F$ is a primitive feature of the type represented by $R_i^0$ and $A_f$ is its vector of attributes. Intuitively, at level 0 of the relational pyramid, each feature is associated with its attributes and is classified as one of several different legal types.

Description $D_1$ is a sequence of relations $< R_1^1, \ldots, R_{n_1}^1 >$ where each relation $R_i^1$ represents a relationship among two or more of the level-0 primitives. An attributed tuple of one of these level-1 relations $R_i^1$ has the form $((N_1, t_1), \ldots, (N_n, t_n), A)$ where each $N_j$ is the *name* of a relation $R_{N_j}^0$ at level 0, and the corresponding $t_j$ is a *tuple* of $R_{N_j}^0$. The semantics of $((N_1, t_1), \ldots, (N_n, t_n), A) \in R_i^1$ is that the level-0 attributed primitives $(t_1, \ldots, t_n)$ which are of types $(N_1, \ldots, N_n)$, are related according to the level-1 relationship $R_i^1$, and this level-1 relationship has attribute vector $A$. This idea can then be extended up the pyramid. At level $k$, description $D_k$ is a sequence of relations $< R_1^k, \ldots, R_{n_k}^k >$, where each relation $R_i^k$ represents a relationship among two or more of the entitities from level 0 to level $k-1$. (In the strictest kind of pyramid, they would all be from level $k-1$.) An attributed tuple of such a level-k relation $R_i^k$ has the form $((N_1, t_1), \ldots, (N_n, t_n), A)$ where each $N_j$ is the name of a relation $R_{N_j}^{k'}$ at a previous level $k'$ and $t_j \in R_{N_j}^{k'}$ for $j = 1, \ldots, n$. The semantics of $((N_1, t_1), \ldots, (N_n, t_n), A) \in R_i^k$ is that the attributed primitives $(t_1, \ldots, t_n)$ which come from levels 0 to $k-1$ and which are of relational types $(N_1, \ldots, N_n)$ are related according to the level-$k$ relationship $R_i^k$ and this level-$k$ relationship has attribute vector $A$.

Thus the relational pyramid structure allows us to define an object by its attributed primitives, relationships among those primitives, relationships among those relationships, and so on up to some predefined maximal level. It is a hierarchical, relational structure, but the hierarchy is defined on relationships instead of on larger and larger pieces of the object. Having formally defined the structure, we will now show how it can be used to describe a view or a view class of a three-dimensional object.

### 2.2 Current Object Representation

Our CAD models are generated by the PADL-2 solid modeling system, a constructive solid geometry system (CSG) whose primitive solids include rectangular polyhedra, spheres, cylinders, cones, wedges, and tori. PADL-2 was chosen because it had the flexibility of CSG input and the ability to convert CSG representation into a boundary representation consisting of the surfaces and edges of the object. For our first experimental system utilizing the relational pyramid concept, we selected a fixed set of two-dimensional primitives and relationships that form a four-level pyramid.

The level-0 primitives in our current system are junction points, where two or more line segments meet. The attribute vector for a junction point consists only of its coordinates. At level 1 are straight line segments, curved line segments, and vertical straight line segments. A straight line segment is represented by its starting point and its ending point, and a curved line segment is represented by its starting point, its ending point, and an interior point which is used in later calculations of relationships. Since we are working with stable views of the objects, the vertical straight line segments are also useful in recognition and are treated both as a subset of all straight line segments and as a separate level-1 relation.

The level-2 relations represent junctions where two or three segments meet. (This will later be extended to multi-segment junctions.) For junctions where only straight lines meet, we use the standard junction types FORK, ARROW, T, and L. For junctions including at least one curved line, we chose to define a new labeling scheme that helps us to build up relations at the next level of the pyramid. (For an alternate labeling scheme for junction with curves, where junction types represent 3D information rather than just 2D configurations, see Chakravarty (1979).) In our current scheme, a curved segment is considered concave (A) or convex (V) depending on the way it faces the segment previous to it in a clockwise ordering of the segments. (Since our curve segments come from spheres, cylinders, and cones, they will not have inflection points.) The first segment in a junction with a straight line segment and one or two curved line segments is defined to be the straight line segment. The first segment in a junction with two straight line segments and one curved line segment is the straight line segment counterclockwise from the curved segment. If there are no straight lines, the curved segment whose chord joining its start and end points is closest to vertical will be considered the first segment. While the labels of this last group of junctions are not rotation-invariant, they are sufficient for our current work that considers only a set of stable views of each object.

The label of a junction then depends on the labels of the two or three segments comprising it, in the ordering in which they are numbered. For example, LA is the label of a junction where a straight line segment connects to a concave curve segment, while LAV is the label of a junction where a straight line segment is followed (in clockwise order)

by a concave curve segment which is followed by a convex curve segment. The relations currently implemented at level 2 of the pyramid represent each of the junction types just described plus the LOOP relation, which consists of sets of segments that together form a minimal closed boundary. Other feasible level-2 relations for view classes would be parallel line segments, collinear line segments, and such spatial relations as above, below, left-of, and right-of (when they are invariant for all views in a view class).

The level-3 relations use level-2 tuples representing attributed junctions and loops and level-1 tuples representing line and curve segments as their primitives. The relations currently being implemented at level 3 are COLLINEAR, ADJACENT, and INSIDE. Because these relations are being defined on junctions rather than on line segments, they have special definitions.

The COLLINEAR relation consists of attributed sets of collinear junctions. Two junctions are considered collinear if there is one pair of corresponding edges which satisfy the same linear equation. The ADJACENT relation consists of pairs of junctions which are directly connected to each other by a common segment. If above, below, left-of, and right-of are invariant across a view class, these can be used as attributes to the adjacency of the junctions. Finally the INSIDE relation consists of a level-1 primitive and a level-2 loop, where the primitive lies inside the loop. Figure 1 shows a line drawing representing one view class of a three-dimensional object, and Figure 2 illustrates its relational pyramid representation.

## 3. Matching Unknown Views to View Class Descriptions

When an image is taken of a known 3D object from an unknown view, the first step before inspection or guidance is to determine the pose (position and orientation) of the object. To achieve this with a view class model, the vision system must first determine the correct view class, find the correspondences between the features extracted from the image and those in the view class representation, use the links between 3D features and view class features to find the correspondence between extracted features and 3D features, and use this correspondence to find the pose of the object.

It is desirable to determine the correct view class as rapidly as possible. Chakravarty and Freeman (1982) represented a view class by a vector containing the number of junctions of each junction type. They used the values in the vector to select the best view class, relying especially on the most frequent junction type. We feel that this approach, while simple and rapid, will not work very well when some of the segments do not show up in the image, causing missing or erroneous junction types, or when extra segments appear in the image. Ikeuchi (1987), using range data, created an interpretation tree during his offline processing phase. The interpretation tree was a decision tree used to select the best view class, depending on the values of various measurements. We will consider this approach in the future, but initially, we are trying a simple idea based on the accumulator method used in most implementations of the Hough transform, which we consider promising.

Suppose each view class is represented by a relational pyramid structure. For each relational pyramid, we can easily derive a *summary pyramid* structure. Where the relational pyramid has a relation R with $c$ tuples $\{((N_1, t_{1,j}), (N_2, t_{2,j}), \ldots, (N_n, t_{n,j}), A) \mid j = 1, \ldots, c\}$, the summary pyramid has a corresponding relation R with a single tuple $((N_1, N_2, \ldots, N_n), c)$ representing those $c$ tuples. For example, if the collinear relation has 4 tuples of the form $((FORK, f), (ARROW, a))$, then the collinear summary relation has one tuple $((FORK, ARROW), 4)$. This is done for each relation and at each level of the pyramid. At level 0, the summary is just the count $c$ of how many primitive features there are of each type. Figure 3 illustrates the summary structure for the relational pyramid of Figure 2.

Along with the original relational pyramid structures and the summary structures, the online system requires one more structure to be produced by the offline system: the *index structure*. The index allows direct access, given a summary tuple of the form $((N_1, N_2, \ldots, N_n), c)$, to a list of all view classes that have this tuple in their summary structures. It keeps an evidence accumulator for each view class, initialized to zero. For exact matching, the online system traverses the summary structure derived from the unknown view, and for each tuple in the summary structure, it adds one to the accumulators of all the view classes on the list attached to that tuple in the index. The view class or classes with maximal evidence are selected.

Exact matching will produce erroneous results, due to missing and extra segments. We have investigated two approaches toward the solution of this problem: 1) adding models of imperfect versions of each view to the database of view class models and 2) developing an inexact matching procedure that can work with either the original or extended database.

### 3.1 Extended Database Method

When our prediction system is completed, it will be possible to predict, for each view class, the variants within that view class and the probability of each variant. For this initial matching procedure, we assume that the only variants of the perfect line drawing of a view class are similar line drawings with at most two internal line segments missing, all boundary line segments present, and no extra line segments. We are well aware that this assumption is too restrictive for many real imaging situations, even with carefully controlled lighting. However, it does allow us to study the behavior of the matching routines.

For each view class of an object, a perfect line drawing of that view class is obtained from the PADL-2 system and converted to its relational pyramid and related summary representation. Each line segment of the perfect line drawing is assigned a probability of being detected in an image. Boundary line segments are assigned probability one and internal segments are assigned lesser probabilities, currently based only on observation and intuition. These probabilities will later be assigned by the prediction system. Next, a set of imperfect line drawings are constructed by removing one or two internal line segments from the perfect line drawing. Each perfect or imperfect example $V_n$ of a view class is assigned a probability $p(V_n)$ which is the product of the probabilities of its visible line segments multiplied by the product of one minus the probabilities of its missing line segments. Each perfect view class $V$ is assigned a probability $p(V)$ meant to indicate how often an unknown view is expected to belong to that view class. These probabilities are also assigned by the experimenter. The *a priori* probability of a perfect or imperfect view class $V_n$ which belongs to the perfect view class $V$ is defined to be $P(V_n) = p(V_n)p(V)$.

The database of relational pyramid summaries consists of summaries of all the perfect views of the object plus the summaries of all the imperfect views of each view class. The summary structure of an unknown view is matched against this entire database of summaries as described above. The accumulator method is used to determine a set of perfect and imperfect views that best match this summary. Now we can use the probability information to order the models in this set.

Let $\{V_1, V_2, \ldots, V_N\}$ be the set of view classes that best match unknown view $V$ via the accumulator method of matching. Let $P(V_n)$ be the *a priori* probability that any unknown view belongs to view class $V_n$ as defined above. Let $P(S \mid V_n)$ be the probability of a particular summary $S$ given that the summary came from an image whose view class is $V_n$. The quantity $P(S \mid V_n)$ is derived from the value in the accumulator for class $V_n$ after the matching is performed on summary $S$. Then the probability $P(V_n \mid S)$ of view class $V_n$ given summary $S$ is given by

$$P(V_n \mid S) = \frac{P(S \mid V_n)P(V_n)}{\sum_{i=1,\ldots,N} P(S \mid V_i)P(V_i)}.$$

The most probable view class $V_k$ is the one that satisfies

$$P(V_k \mid S) = \max_{n=1,\ldots,N} P(V_n \mid S).$$

Of course if several view classes have similar high probabilities, we must consider each of them further in the detailed matching (See Section 3.3.)

### 3.2 Inexact Matching Method

The inexact matching version of the procedure is as follows. Suppose the unknown view has a summary tuple $((N_1, N_2, \ldots, N_n), c)$. We add one to the accumulators of those view classes on the list attached to $((N_1, N_2 \ldots, N_n), c)$. We add a quantity less than one to the accumulators of those view classes on the lists attached to $((N_1, N_2 \ldots, N_n), c+1)$ and $((N_1, N_2 \ldots, N_n), c-1)$. We add an even smaller quantity to the accumulators of those view classes on the lists attached to $((N_1, N_2 \ldots, N_n), c+2)$ and $((N_1, N_2 \ldots, N_n), c-2)$. In general, we add $e^{\frac{-k^2}{2}}$ to the accumulators of those view classes on lists attached to $((N_1, N_2 \ldots, N_n), c+k)$ and $((N_1, N_2 \ldots, N_n), c-k)$, for $k = 0, \ldots, K$ where $K$ is the maximum amount of deviation allowed and can be controlled by the experimenter. Currently, we allow as big a deviation as the data presents. As in the exact matching case, this process should be repeated for each tuple in the summary structure of the unknown view. Finally, the view class or classes that have the highest evidence counts are selected and the probability analysis used to order the set as before. Because this method uses a Gaussian distribution in the accumulator updating process, we call it the *Gaussian distribution method.* The exact matching version, which merely adds one to the accumulators of models whose count is identical to that of the unknown and zero to the rest, is called the *spike distribution method.*

### 3.3 Detailed Matching

Once a view class has been selected, we must determine the correspondence between the primitives of its relational pyramid and those of the unknown view's relational pyramid. Since the relational pyramid structure is a highly constrained, relational representation of a view class or view, a variation of the A* algorithm can rapidly find the best mapping from view class structure to view structure. In particular, the higher-level relations aid in efficient pruning of the tree. Furthermore, it is not necessary to find a correspondence for every primitive feature of a view class. We need only find enough correspondences to reliably compute the pose of the object. A separate research effort is adressing this issue.

## 4. Experiments and Results

We have implemented the accumulator-based matching and run a sequence of experiments that test the four variants of the matching procedure: 1) spike distribution using erroneous (imperfect) views, 2) spike distribution using only perfect views, 3) Gaussian distribution using erroneous views, and 4) Gaussian distribution using only perfect views. The experiments were applied to both real and synthetic images of six test objects. The objects, which are metal, machined parts with low specularity, are shown in Figure 4. We will refer to them as cubehole, widget, cubecut, tarrow, cubecyl, and cube3cut in our discussion. The objects were modeled with the PADL-2 geometric modeling system, and the line drawings of various stable views produced by PADL were used to construct the database of perfect and imperfect view class pyramids and summaries, as described previously. Five test images were produced for each object, three from real images of the actual metal objects and two from a locally available and very flexible CAD system called Renaissance, which was developed by Prof. Anthony DeRose of our Computer Science Department. The names of the images indicate the object they represent and the method of formation. Those objects whose name ends in "mvi" followed by an integer were digitized on the MVI Genesis-2000 machine vision system. Those objects whose name ends in "tony" followed by an integer came from Renasissance.

All the test images were first processed on the Genesis-2000 machine to remove noise and to detect edges using a morphological edge operator (Lee *et al*, 1987). Next, the edge images were transferred to a Sun 3/280 computer where junctions were detected, and the pyramid and summary structures were constructed. Figure 5 shows the five original test images of the widget object, their edge images (after noise removal, thinning, and edge linking) and the detected junctions.

The accumulator method was implemented as described above, but with separate accumulators for each separate relation of each view class. The motivation for this approach was that some of the relations would turn out to be more important or more useful than others, and we wanted to be able to view the separate results before adding them all into one accumulator for the view class. As a result of our experience with the particular relations we chose, we were able to determine that some of the relations were more useful than others, some were not only not useful, but detrimental (parallel line segments under perspective projection), and some that we had not started with were added (junction points and vertical lines).

Based on our experience with the relations, we added a set of weights to the system. Each relation type was assigned a weight depending on its perceived usefulness. The final result in the accumulator for a view class V is a

weighted, normalized sum of the results in the accumulators for the separate relations of view class V. The numbers shown in our result tables are the final probabilities assigned to each view class after the probabilistic analysis followed in parentheses by the number of the specific perfect or imperfect view subclass that the unknown image matched best to in each of the six main view classes involved in the experiments.

The results for object widget are shown in Table 1. Table 2 indicates how many images matched best to the correct view and how many images matched best or second best to the correct view for all four methods.

## 5. Conclusions

As shown in Table 2, the test images matched best or second best to the correct view class in 90% of the experiments and 96% of the experiments using the Gaussian distribution with erroneous views. Most of the cases where the test image did not match best to the correct view class were due to erroneous versions of two different view classes being similar. Of the thirty images, three did not satisfy our assumption that at most two internal lines were missing. Of these, two which had three lines missing still matched best to the correct view class (one under the spike variant and one under both the spike and Gaussian variants) and the third which had six lines missing still matched second best to the correct view class using the Gaussian variant. For these images that did not meet our assumptions, the variants using the erroneous views did better than the variants that only knew about the perfect views.

## 6. Ongoing Work in Prediction

The assumptions used in this preliminary study are only valid for near–perfect images obtained by optimizing the lighting for each object. It is more realistic in industrial applications to assume an environment where the image acquisition procedure is carefully defined, but remains constant over all the objects in a class. Our current research involves the development of a prediction system that takes into account the object geometry, the lighting, and other environmental factors and predicts what will appear on an image. The system is called PREMIO (PREdiction in Matching Images to Objects).

PREMIO's goals are (1) to predict the features that will appear on an image of an object from a given viewpoint and under given lighting conditions, (2) to evaluate the features with respect to detectability, reliability, utility, and cost; and (3) to organize the data resulting from application of (1) and (2) at a sampled set of viewpoints into a structure that can be used effectively in matching. The major work so far has been the development of the software to convert a PADL-2 model into a generic hierarchical, relational structure and to use this structure to predict features that will appear on an image from a specified viewpoint. Because the purpose is vision, not graphics, we use analytic feature prediction instead of the slower ray casting approach. The result is a data structure of two–dimensional symbolic line segments that know what three–dimensional entity they came from. We are now working on adding a lighting model to the system which will ray trace along these potentially visible line segments to determine the effect of the lighting setup on the features. Thus for each view class of the object, we will be able to determine the likelihood of a line segment appearing as a whole, disappearing entirely, or breaking into disconnected pieces. The resultant probabilities will allow us to make realistic assumptions in our matching procedures for specific vision tasks. The design of this sytem is discussed in Camps et al. (1989).

## References


1. Bolles, R. C. and R. A. Cain, "Recognising and Locating Partially Visible Objects: the Local-Feature Focus Method", *International Journal of Robotics Research*, Vol. 1, No. 3, 1982, pp. 57-82.
2. Camps, O.I., L.G. Shapiro, and R.M. Haralick, "PREMIO: The Use of Prediction in a CAD–Model–Based Vision System," *Technical Report #EE-ISL-89-01,* Department of Electrical Engineering, University of Washington, July, 1989.
3. Chakravarty, I, "A Generalized Line and Junction Labelling Scheme with Application to Scene Analysis", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-1, No. 2, 1979, pp. 202-205.
4. Chakravarty, I. and H. Freeman, "Characteristic Views as a Basis for Three-Dimensional Object Recognition", *Proceedings of SPIE 336 (Robot Vision)*, 1982, pp. 37-45.
5. Henderson, T., C. Hansen, A. Samal, C.C. Ho, and B. Bhanu, "CAGD-Based 3-D Visual Recognition", *Proceedings of the Eighth International Conference on Pattern Recognition*, Paris, October 1986, pp. 230-232.
6. Ikeuchi, K. "Generating an Interpretation Tree From a CAD Model for 3D-Object Recognition in Bin-Picking Tasks", *International Journal of Computer Vision*, 1987, pp. 145-165.
7. Lee, S.J., R. M. Haralick, and L.G. Shapiro, "Morphological Edge Detection", *IEEE Journal of Robotics and Automation*, Vol. RA-3, No. 2, April, 1987, pp. 142-156.
8. Narasimhamurthi, N. and R. Jain, "CAD-Based Object Recognition: Incorporating Metric and Topological Information", *Proceedings of the SPIE Conference on Digital and Optical Shape Recognition and Pattern Recognition*, Vol. 938, April, 1988, pp. 436-443.
9. Ponce, J. and D. Chelberg, "Finding the Limbs and Cusps of Generalized Cylinders", *International Journal of Computer Vision*, Vol. 1, No. 3, 1987, pp. 195-210.



This research was sponsored by the National Aeronautics and Space Administration (NASA) through a subcontract from Machine Vision International.

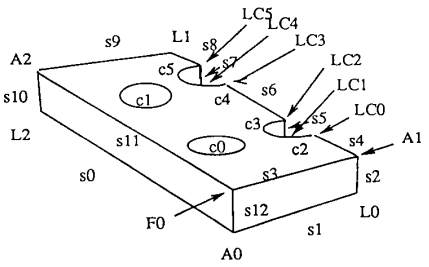



**Figure 1** *illustrates the line drawing representing a view class of a machined part object.*

**Level 0**

Points 13

**Level 1**

Straight 13
Curved 6
Vertical 5

**Level 2**

Junctions

FORK 1
ARROW 3
L 3
LV 2
LAV 2
LAL 2

Loops 7

**Level 3**

Collinear

[(FORK,ARROW),3]
[(ARROW,L),5]
[(ARROW,LV),1]
[(L,LV),2]
[(L,LAL),2]
[(LAV,LAL),2]

Adjacent

[(FORK,ARROW),3]
[(ARROW,L),4]
[(ARROW,LV),1]
[(L,LAL),1]
[(LAL,LAV),2]
[(LV,LAV),2]

Inside Relation

[(LOOP,LOOP), 2]

**Figure 3** *illustrates a summary structure for the relational pyramid of Figure 2.*

LEVEL3:

| Collinear | Adjacent | Inside |
|---|---|---|
| {s0,[(ARROW,A0),(L,L2)]} | {s0,[(ARROW,A0),(L,L2)]} | {(LOOP,LP2),(LOOP,LP5),(LOOP,LP6)} |
| {s1,[(ARROW,A0),(L,L0)]} | {s1,[(ARROW,A0),(L,L0)]} | |
| {s2,[(ARROW,A1),(L,L0)]} | {s2,[(ARROW,A1),(L,L0)]} | |
| {s3,[(FORK,F0),(ARROW,A1)]} | {s3,[(FORK,F0),(ARROW,A1)]} | |
| {[s4,[(ARROW,A1),(LV,LC0)]], {s6,[(LAL,LC2),(LV,LC3)]}, {s8,[(LAL,LC5),(L,L1)]}} | {s4,[(ARROW,A1),(LV,LC0)]} | |
| {s5,[(LAV,LC1),(LAL,LC2)]} | {s5,[(LAV,LC1),(LV,LC3)]} | |
| {s7,[(LAV,LC4),(LAL,LC5)]} | {s6,[(LAL,LC2),(LV,LC3)]} | |
| {s9,[(ARROW,A2),(L,L1)]} | {s7,[(LAV,LC4),(LAL,LC5)]} | |
| {s10,[(ARROW,A2),(L,L2)]} | {s8,[(LAL,LC5),(L,L1)]} | |
| {s11,[(FORK,F0),(ARROW,A2)]} | {s9,[(ARROW,A2),(L,L1)]} | |
| {s12,[(FORK,F0),(ARROW,A0)]} | {s10,[(ARROW,A2),(L,L2)]} | |
| | {s11,[(FORK,F0),(ARROW,A2)]} | |
| | {s12,[(FORK,F0),(ARROW,A0)]} | |
| | {c2,[(LV,LC0),(LAV,LC1)]} | |
| | {c3,[(LAV,LC1),(LAL,LC2)]} | |
| | {c4,[(LV,LC3),(LAV,LC4)]} | |
| | {c5,[(LAV,LC4),(LAL,LC5)]} | |

LEVEL2:

| Fork_Junctions | Arrow_Junctions | L_Junctions |
|---|---|---|
| F0 (s12,s11,s3) | A0 (s12,s1,s0) | L0 (s2,s3,s4) |
| | A1 (s2,s3,s4) | L1 (s9,s8) |
| | A1 (s10,s9,s11) | L2 (s10,s0) |

| LV_Junctions | LAV_Junctions | LAL_Junctions |
|---|---|---|
| LC0 (s4,c2) | LC1 (s5,c3,s6) | LC2 (s5,c3,s6) |
| LC3 (s6,c4) | LC4 (s7,c4,c5) | LC5 (s7,c5,s8) |

Loops:
LP0 (s0,s12,s11,s10)
LP1 (s2,s3,s12,s1)
LP2 (s3,s4,c2,c3,s6,c4,c5,s8,s9,s11)
LP3 (s5,c3)
LP4 (s7,c5)
LP5 (c1)
LP6 (c0)

LEVEL1:
Segments:

| Lines | Curves | Vertical lines |
|---|---|---|
| {s0, (148,299),(187,299),0.0} | {c0 (329,344),(329,344),(331,327),0.4} | s2 |
| {s1, (185,263),(247,211),0.0} | {c1 (291,169),(291,169),(325,138),0.4} | s10 |
| {s2, (247,211),(291,213),0.0} | {c2, (291,213),(247,211),(266,229),0.4} | s12 |
| {s3, (217,427),(233,416),0.4} | {c3, (217,427),(233,416),(202,362),0.4} | s7 |
| {s4, (329,344),(233,416),0.0} | {c4, (187,299),(148,299),(170,317),0.4} | s5 |
| {s5, (329,344),(348,331),0.0} | {c5, (187,299),(185,263),(191,279),0.4} | |
| {s6, (181,456),(217,427),0.0} | | |
| {s7, (121,320),(148,299),0.0} | | |
| {s8, (181,456),(217,427),0.0} | | |
| {s9, (348,331),(406,291),0.0} | | |
| {s10,(181,456),(235,453),0.0} | | |
| {s11,(235,453),(463,292),0.4} | | |
| {s12,(291,213),(291,169),0.4} | | |

LEVEL0:
Points:
13

**Figure 2** *illustrates the pyramid structure for the view class shown in Figure 1 with attribute information suppressed for simplicity.*

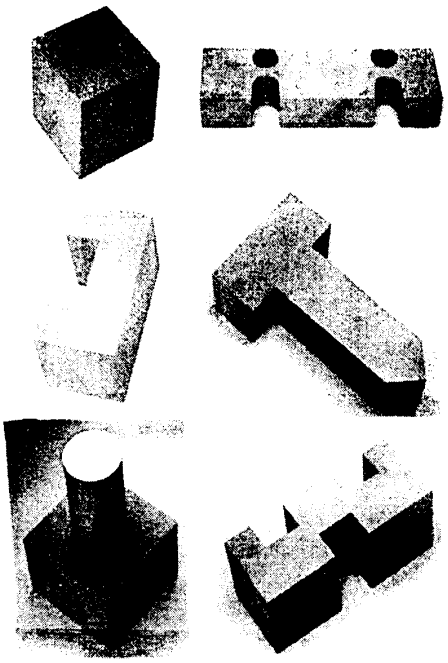
| Gaussian Distribution Using Erroneous Views | | | | | | | |
|---|---|---|---|---|---|---|---|
| Images | Belongs to | Classified as | | | | | |
| | | 1 | 2 | 3 | 4 | 5 | 6 |
| widget_tony1 | 2 | 0.30(31) | **0.49(2)** | 0.31(154) | 0.24(189) | 0.10(235) | 0.20(255) |
| widget_mvi1 | 3 | 0.17(54) | 0.14(81) | **0.56(3)** | 0.35(198) | 0.15(5) | 0.42(262) |
| widget_mvi2 | 4 | 0.42(35) | 0.24(132) | 0.13(3) | **0.57(178)** | 0.14(5) | 0.47(254) |
| widget_mvi3 | 5 | 0.26(35) | 0.20(100) | 0.20(3) | 0.23(174) | **0.75(5)** | 0.22(280) |
| widget_tony2 | 6 | 0.25(11) | 0.26(100) | 0.29(3) | 0.33(219) | 0.34(5) | **0.78(275)** |
| % that matched best to correct views : 100 % | | | | | | | |
| % that matched first or second best to correct views: 100 % | | | | | | | |
| % that didn't match to correct views : 0 % | | | | | | | |

| Gaussian Distribution Using Only Perfect Views | | | | | | | |
|---|---|---|---|---|---|---|---|
| Images | Belongs to | Classified as | | | | | |
| | | 1 | 2 | 3 | 4 | 5 | 6 |
| widget_tony1 | 2 | 0.26 | **0.65** | 0.30 | 0.12 | 0.29 | 0.11 |
| widget_mvi1 | 3 | 0.14 | 0.09 | **0.56** | 0.17 | 0.15 | 0.16 |
| widget_mvi2 | 4 | 0.30 | 0.15 | 0.13 | **0.48** | 0.14 | 0.36 |
| widget_mvi3 | 5 | 0.25 | 0.05 | 0.20 | 0.08 | **0.75** | 0.04 |
| widget_tony2 | 6 | 0.24 | 0.12 | 0.29 | 0.22 | 0.34 | **0.37** |
| % that matched best to correct views : 100 % | | | | | | | |
| % that matched first or second best to correct views: 100 % | | | | | | | |
| % that didn't match to correct views : 0 % | | | | | | | |

| Spike Distribution Using Erroneous Views | | | | | | | |
|---|---|---|---|---|---|---|---|
| Images | Belongs to | Classified as | | | | | |
| | | 1 | 2 | 3 | 4 | 5 | 6 |
| widget_tony1 | 2 | 0.28(8) | **0.49(133)** | 0.10(3) | 0.18(189) | 0.08(235) | 0.13(288) |
| widget_mvi1 | 3 | 0.10(39) | 0.11(77) | **0.43(3)** | 0.20(182) | 0.12(5) | 0.26(239) |
| widget_mvi2 | 4 | 0.28(8) | 0.19(95) | 0.03(3) | **0.50(174)** | 0.05(234) | 0.24(237) |
| widget_mvi3 | 5 | 0.21(9) | 0.15(139) | 0.17(3) | 0.17(179) | **0.37(5)** | 0.17(280) |
| widget_tony2 | 6 | 0.15(22) | 0.19(139) | 0.27(160) | 0.24(177) | 0.32(234) | **0.77(275)** |
| % that matched best to correct views : 100 % | | | | | | | |
| % that matched first or second best to correct views: 100 % | | | | | | | |
| % that didn't match to correct views : 0 % | | | | | | | |

| Spike Distribution Using Only Perfect Views | | | | | | | |
|---|---|---|---|---|---|---|---|
| Images | Belongs to | Classified as | | | | | |
| | | 1 | 2 | 3 | 4 | 5 | 6 |
| widget_tony1 | 2 | 0.21 | **0.23** | 0.10 | 0.04 | 0.08 | 0.00 |
| widget_mvi1 | 3 | 0.09 | 0.05 | **0.43** | 0.03 | 0.12 | 0.03 |
| widget_mvi2 | 4 | 0.22 | 0.12 | 0.03 | **0.36** | 0.05 | 0.15 |
| widget_mvi3 | 5 | 0.21 | 0.01 | 0.17 | 0.04 | **0.37** | 0.02 |
| widget_tony2 | 6 | 0.11 | 0.08 | 0.26 | 0.15 | 0.15 | **0.32** |
| % that matched best to correct views : 100 % | | | | | | | |
| % that matched first or second best to correct views: 100 % | | | | | | | |
| % that didn't match to correct views : 0 % | | | | | | | |

**Table 1** *gives the results of matching five test images to the widget object for each of the four matching variants.*



(1) a cubehole (left top). (2) a widget (right top). (3) a cubecut (left middle).
(4) a tarrow (right middle). (5) a cubecyl (left bottom). (6) a cubeJcut (right bottom).

**Figure 4** *illustrates the six objects used in our experiments.*

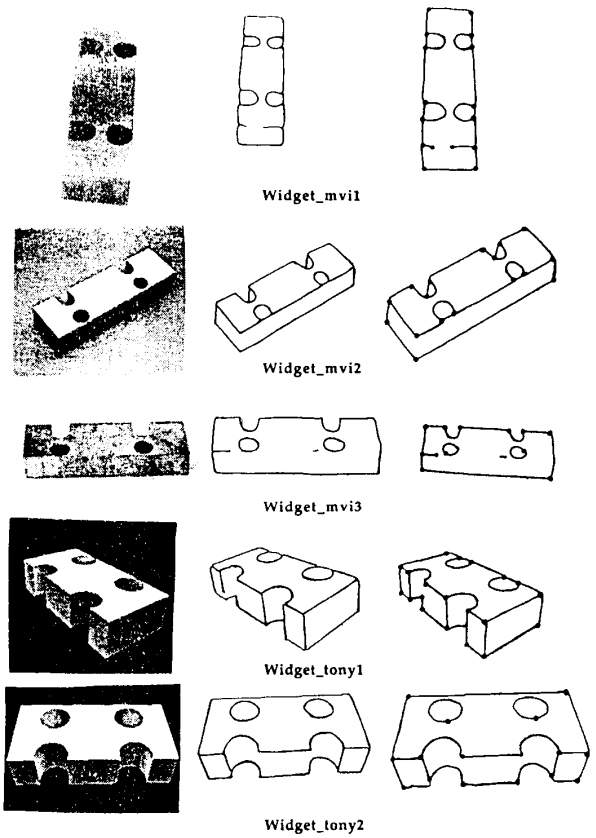



**Figure 5** *illustrates the original test images, the edge images, and the junctions for the widget object.*

| Method | Match best | Match best or second best |
|---|---|---|
| SDEV | 0.86 | 0.90 |
| SDPV | 0.86 | 0.90 |
| GDEV | 0.83 | 0.96 |
| GDPV | 0.73 | 0.90 |

**Table 2** *shows the number of images that matched best or second best to the correct view class for each of the four methods.*