N89-19867

CAD-Model-Based Vision for Space Applications

Linda G. Shapiro

Department of Electrical Engineering, FT-10
University of Washington
Seattle, WA 98195
USA

ABSTRACT

A pose acquisition system operating in space must be able to perform well in a variety of different applications including automated guidance and inspections tasks with many different, but known objects. Since the space station is being designed with automation in mind, there will be CAD models of all the objects, including the station itself. The goal of our work is to construct vision models and procedures directly from the CAD models. The system we are designing and implementing must convert CAD models to vision models, predict visible features from a given view point from the vision models, construct view classes representing views of the object, and use the view class model thus derived to rapidly determine the pose of the object from single images and/or stereo pairs.

Keywords: matching, view class, CAD model, relational pyramid

1. Introduction

CAD systems are increasingly becoming an integral component of the manufacturing process in the factories and plants of the United States. Parts can be designed interactively and then automatically machined from the newly created models. As part of the automation process, some manufacturers are buying machine vision systems to inspect the machined parts and to provide the three-dimensional guidance for robots that handle the parts. Currently, most of these systems have to be reprogrammed to recognize each new part; this can involve weeks or months of effort, depending on the task. Since many manufacturers have the CAD models of the parts, the most desirable approach is to produce the vision algorithms directly from the CAD models. This approach has already been recognized as important, and some preliminary work in the area has been done. Examples include the work of Henderson and Bhanu at The University of Utah (Henderson et al., 1986) and that of Ikeuchi at Carnegie-Mellon (Ikeuchi, 1987). One 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.

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 processing and one for online processing. The offline processing subsystem will handle 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 will use the vision model and procedure to perform the inspection/guidance task.

CAD models range from the standard three-view, orthographic engineering drawings to the more recent constructive solid geometry (CSG) and boundary models produced by solid modeling systems. Since there are algorithms for converting three views to solid models, we will not consider the three view representation, but are working with solid modeling systems. The purpose of most solid modeling systems is to provide an interactive environment in which an engineer can design a part, with procedures by which he or she can display the part in different orientations. A few systems also produce the specifications for the automatic machining of the part. None of these representations is suitable to be directly used by a machine vision system.

A machine vision system needs a model of a part that specifies the features by which it can be recognized from one or more images. The features that will show up and can be extracted from an image are dependent on the

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

view, the material of the object, and the lighting setup. Machine vision companies that sell software to customers for particular inspection or guidance applications typically have programmers and engineers who spend months developing the best lighting and optics and the algorithms for determining the exact pose of the part and then performing the inspection or guidance task. All of the knowledge and experience of these people is missing in the bare CAD models. Thus the offline system must contain enough knowledge of lighting and materials and the physics of imaging, that it can perform the same task automatically and reliably.

In our overall design, the input to the offline system consists of 1) the CAD model from the geometric modeling system, 2) a knowledge base of information on materials, lighting, and imaging, in a rule-based form, and 3) the specifications for the particular task to be performed. The offline system will transform this information into a vision model and a procedure for that task, to be used by the online system. This paper deals with the vision models being produced and the matching algorithms that will use these models. The topics to be discussed include 1) prediction of images features from object models, 2) representation of the features predicted for a given view and their interrelationships, 3) generation of viewing clusters, and 4) matching an unknown view to the representatives of the view classes derived from the clustering.

2. Predicting Image Features from Object Models

Prediction of image features from object models is important in determining the machine vision algorithms that will detect and extract the features and match to stored models. Ikeuchi (1987) has included prediction in his system that matches range data derived from photometric stereo. Ponce and Chelberg (1987) have worked out the mathematics for predicting the appearance of limbs and cusps of generalized cylinders. We have chosen to work with standard geometric modeling systems of the type used in computer aided design of industrial parts in order to be most compatible with current and potential industrial applications.

There are two major ways of representing three-dimensional objects in todays geometric modeling systems. Constructive solid geometry (CSG) models are built from primitive solids such as spheres, ellipsoids, cylinders, rectangular parallelepipeds, and cones. The primitive solids are combined using operations of union, intersection, and set difference to produce a binary tree structure that represents the object. The idea is conceptually simple and easy for a designer to learn. Furthermore, the objects produced are guaranteed to be physically possible three-dimensional objects. The tree structures can be used to generate wire frame drawings of the object or, with the use of a ray casting procedure, gray tone and color images of different views with different lightings. Boundary models represent an object by its surfaces and edges. They are more difficult to use for construction of an object, but potentially more flexible, since the surfaces and edges may be represented by B-splines, allowing much more generality. It is possible to construct a boundary representation (BREP) that does not correspond to any physically possible object. Boundary models can also be used to generate wire frame, gray tone, or color images.

We are using three different systems in our present work. The first system, Renaissance, is an experimental CSG modeller being developed by Tony DeRose of the Computer Science Department at the University of Washington. This system accepts CSG input and applies ray casting to produce shaded images. It allows spheres, ellipsoids, cylinders with spherical or ellipsoidal faces, rectangular parallelepipeds, cones, and half planes as primitives. It also allows the user to specify the reflectivity of the surface material and to include any number of point light sources at different positions and with different intensities. Ongoing research on this system is producing the ability to specify areal light sources of various shapes that are more realistic than the point light sources. We are using this system mainly to generate artificial images for display and for image processing.

The second system we are using is PADL-2 (Voelcker and Requicha, 1977) which was developed at Rochester and is now being distributed by Cornell University. PADL-2 is a CSG system, but has the ability to convert the CSG models to BREP. Since vision systems deal with surfaces and edges, this converstion is essential to our work. PADL-2 allows spheres, cylinders, rectangular parallelepipeds, cones, wedges, and tori as primitives, and it does not have as much flexibility as WART in selection of lighting. We are mainly using it for fast wire frame drawings of the objects and for the conversion to BREP.

The third system we are using is CATIA, a joint venture of IBM and Dassault. CATIA can produces a BREP with not only simple surfaces and curves, but also B-spline surfaces and curves. Although we cannot access the CATIA system ourselves, we will be given data from that system via IGES format tapes.

The prediction work consists of extracting data from various geometric modeling systems, storing it in a three-dimensional vision model of our own design, and using the vision model to predict the features that will appear in images. The extraction process is merely a programming task, made difficult or easy by the particular system from which the data must come. The vision model we are currently using is shown in Figure 1. It is a variation of the hierarchical, relational structure we designed for collision avoidance and matching several years ago (Shapiro and Haralick, 1984). The basic data structures employed are the *relational data structure*, which represents entities, and

the *relation*, which represents attributes of and relationships among entities. A relational data structure is a set of named relations. Each relation is a set of N-tuples for some positive integer N. The components of an N-tuple may be atomic or may themselves be relational data structures. Each relational data structure has one distinguished relation called the attribute-value table. The attribute-value table stores the values associated with global attributes of the entity represented by the relational data structure. The other relations often depict the relationships among the primitive pieces of the entity.

The entities shown in Figure 1 are the world, the object, the face, the boundary, the simple arc, the compound arc, the simple surface and the compound surface. The world is made up of objects. The Objects Relation, which is a part of the World Relational Data Structure, is a list of the objects in the world. Each object in the list has a name, a type, a pointer or reference to the relational data structure for that object, and a transformation that can be applied to the points of the object to position it in the world. The attribute-value table of the world contains global information about the world, including, but not limited to the bounding box shown in the figure. An object has a set of faces and two important relations that embody the topology of the object. The Edge/Surface Topology Relation is a list of the three-dimensional (and possibly curved) edges of the object. Each edge in this relation is associated with its two endpoints, the faces to its left and right, the arcs that represent the edge in the boundaries of the two faces, and the angle between the two faces (or an approximation if they are not planar). The Vertex Relation is a list of three-dimensional vertices. Each vertex has associated a name, a location, a list of edges that meet at that vertex, and a transformation.

Conceptually, a face has a surface equation or equations and a set of boundaries that tell what portions of the surface actually belong to the face. These boundaries are listed in the Boundaries Relation, and the surface is referenced as the value of the Surface attribute in the attribute-value table of the face. This is consistent, because the face has only one surface, but it has a set of boundaries. Each boundary is a list of arcs, which can be simple (represented by an equation) or compound (represented by a Compound Arc Relational Data Structure which itself has an Arcs Relation or list of arcs). Similarly, surfaces are either simple surfaces or Compound Surface Relational Data Structures. At all levels of the structure, entities have associated transformations and bounding boxes.

We have used data structures of this general form for model-based vision systems and geographic information systems and have found them to be very flexible and very natural representations of complex structures. The particular variant discussed here stores all the information usually found in any BREP plus the topological information that can help generate constraints for matching in a vision system. Later in the work, inspection specifications and other attributes will be added to the models.

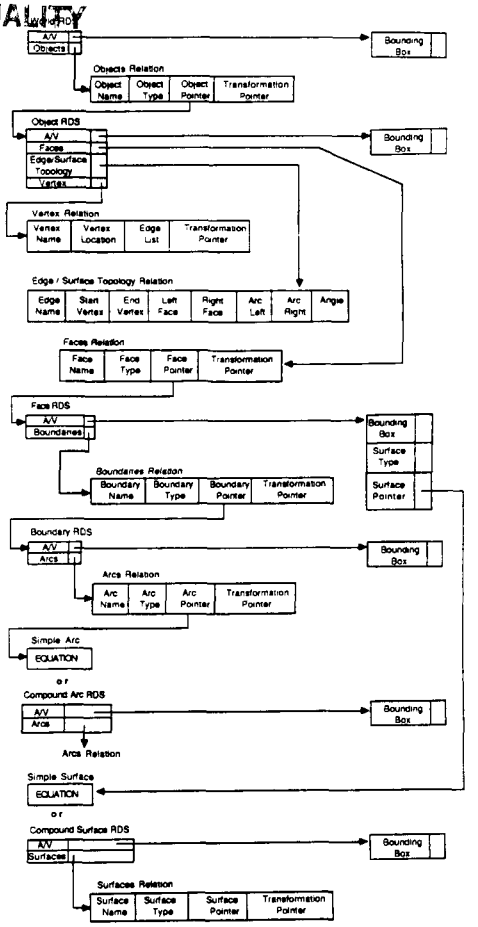

Figure 1 *illustrates the relational data structure for a three-dimensional vision model.*

Given a relational data structure representing a three-dimensional object, the goal of the prediction module is to predict the features of that object that will appear in an image of the object from a given viewpoint and with a given set of light sources. Visible features can consist of edges between separate surfaces of the object, limbs (loci of points where the line of sight is tangent to the surface), corners, holes, imprinted characters, and anything else we might be able to detect with low- and mid-level vision algorithms. Our initial system will predict edges and limbs that should appear in an image of the object from a particular viewpoint and independent of the lighting. The result will be similar to a wire-frame rendering of the object, and the algorithms used will be variations of standard hidden line/hidden surface algorithms used to produce wire-frames. The main difference will be the maintenance of the relationships between the predicted two-dimensional structures and the three-dimensional entities which produced them. This information will be used later

in the pose estimation procedures.

We are currently beginning the implementation of this part of the system. We have derived the equations of the limbs for the simple CSG primitives (ellipsoid, cylinder, and cone) in closed form. Yet to be derived are the equations and procedures for working with B-spline data. The next step is to investigate and characterize the reasons that an edge or a limb may not show up in an image at all. Not enough contrast between two surfaces due to illumination or even the color and patterns of the surfaces and not enough difference in the orientations of the two surfaces are some of the possible causes.

3. Representation of Features

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 viewing cluster. 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.

The level-0 primitives in our current system are straight and curved line segments. Thus in our formalism, $D_0 = <$ straight_segments, curved_segments $>$. The attribute vector for a straight line segment contains its starting point and its ending point, and the attribute vector for a curved line segment contains its starting point, its ending point, and an interior point which is used in later calculations of relationships.

The level-1 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. Because we wish to distinguish between the separate lines of each junction, we define a numbering scheme that selects the first line in each junction as the one closest to vertical and then numbers the remaining lines consecutively in clockwise order.

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

C-5

the first segment. 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 1 of the pyramid represent each of the junction types just described. The LOOP relation, which is also being implemented, consists of sets of segments that together form a minimal closed boundary. Other feasible level-1 relations for view classes would be parallel line segments, colinear 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-2 relations use level-1 tuples representing attributed junctions and loops and level-0 tuples representing line and curve segments as their primitives. The relations currently being implemented at level 2 are PARALLEL, ANTIPARALLEL, REVERSE, COLINEAR, ADJACENT, and INSIDE. Because these relations are being defined on junctions rather than on line segments, they have special definitions.

The PARALLEL relation consists of attributed sets of parallel junctions. Two straight line junctions $J_1 = (l_1, l_2, \ldots, l_n)$ and $J_2 = (l'_1, l'_2, \ldots, l'_m)$, $n \leq m$, are *parallel* if there is an order preserving injection $f : J_1 \rightarrow J_2$ satisfying parallel$(l_i, f(l_i))$, $i = 1, \ldots, n$. Two arbitrary junctions $J_1 = (l_1, l_2, \ldots, l_n)$ and $J_2 = (l'_1, l'_2, \ldots, l'_m)$ are parallel if the nonempty subsequences $K_1 \subseteq J_1$ and $K_2 \subseteq J_2$ consisting of only the straight lines are parallel, where the number of straight lines is greater than one.

While two ARROW junctions may be parallel, it is impossible for a FORK junction and an ARROW junction to be parallel, since one line segment of the FORK will point in the opposite direction as the corresponding line segment of the ARROW. Yet this relationship is also important in describing the line drawing as a whole. For this reason, we define the ANTIPARALLEL and REVERSE relations as follows. Two straight line junctions J_1 and J_2 are *antiparallel* if there is a function $f : J_1 \rightarrow J_2$ satisfying parallel$(l_i, f(l_i))$ or *antiparallel*$(l_i, f(l_i))$. Two straight line junctions are *reverse* if they are antiparallel and if antiparallel$(l_i, f(l_i))$ is true for exactly one pair of corresponding segments. Two junctions are antiparallel (or reverse) if the nonempty subsequences consisting of only straight lines and consisting of at least two straight lines are antiparallel (or reverse). In our implementation, the REVERSE relation corresponds to this definition of reverse, and the ANTIPARALLEL relation corresponds to pairs of junctions that are antiparallel, but not reverse.

The COLINEAR relation consists of attributed sets of colinear junctions. Two junctions are considered colinear if they are parallel, antiparallel, or reverse and 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-0 primitive and a level-1 loop, where the primitive lies inside the loop. Figure 2 illustrates these concepts for one view class of a three-dimensional object.

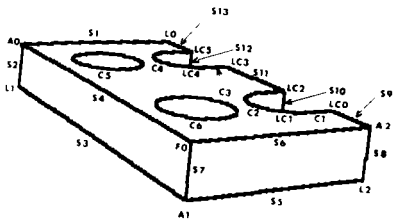

Figure 2a *illustrates the line drawing representing a view class of a machined part object.*

4. Constructing Viewing Clusters

A view class model of an object consists of a set of characteristic views of that object, represented by a suitable data structure. In his range-data-oriented system, Ikeuchi's view classes are the aspects of the object. Two views are part of the same view class if all the same surfaces are visible in each of them. While this is a very reasonable approach for range data, where segmentation into surfaces is relatively easy, it is not reasonable for intensity images of complex parts, where it may be very difficult to accurately segment into individual surfaces. Korn and Dyer suggested that the view classes could be constructed by considering a large number of views around a tesselated Gaussian sphere and grouping together those that share some property. The important question here is what that property might be.

The most reliable features that can be automatically extracted from intensity images are still line segments: both straight and curved. If we apply Ikeuchi's thinking to Korn and Dyer's suggestions, we would say that two views belong to the same cluster if the same line segments are visible in both. But, even under conditions where background and lighting are carefully controlled, one can still expect some missing or extra line segments due to effects of illumination and other environmental factors. Thus two line drawings of an object, taken from approximately the same viewpoint, may be a little bit different. Furthermore, there may be a set of viewpoints which all produce similar, but not identical line drawings, and which we would like to group together in order to control the number of view classes. All of this

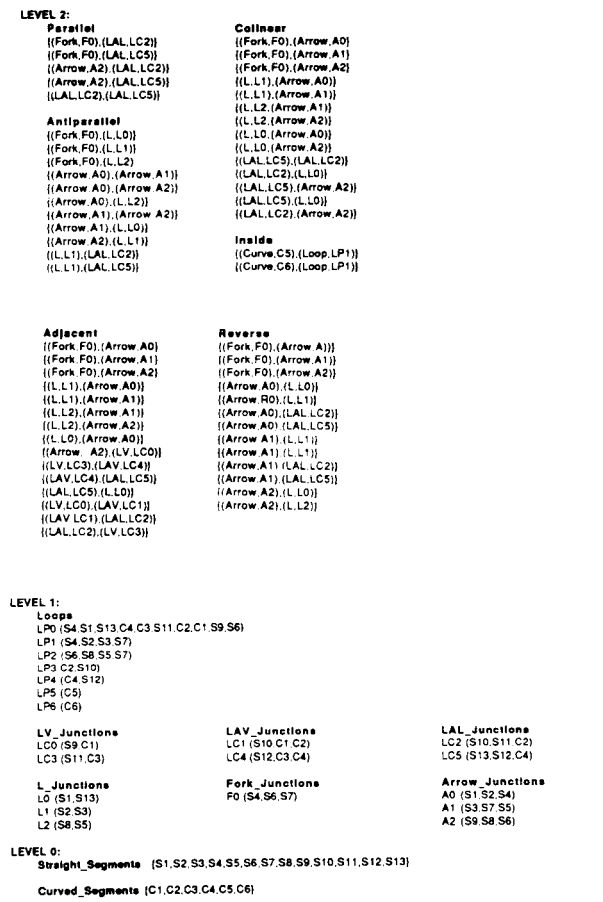

Figure 2b *illustrates the pyramid structure for the view class shown in Figure 2a with attribute information suppressed for simplicity.*

suggests that two views belong in the same view class if their relational distance (Shapiro and Haralick, 1985) is similar enough. Thus finding view classes consists of finding relational distances between pairs of pyramid structures representing view points selected from a tesselated Gaussian sphere and clustering them, based on these relational distances. This is all part of the offline processing of the object.

5. 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 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 parallel relation has 4 tuples of the form $((FORK, f), (ARROW, a))$, then the parallel 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 2b. Note that we have simplified the level-1 summary structures for readability. The names of these level-1 relations actually indicate the types of the primitives.

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 would traverse the summary structure derived from the unknown view, and for each tuple in the summary structure, it would add 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 would be selected.

Exact matching will produce erroneous results, due to missing and extra segments. Our current solution is to actually model the "erroneous" views associated with a view class, producing a separate summary for each. We also associate a probability with each perfect and each erroneous

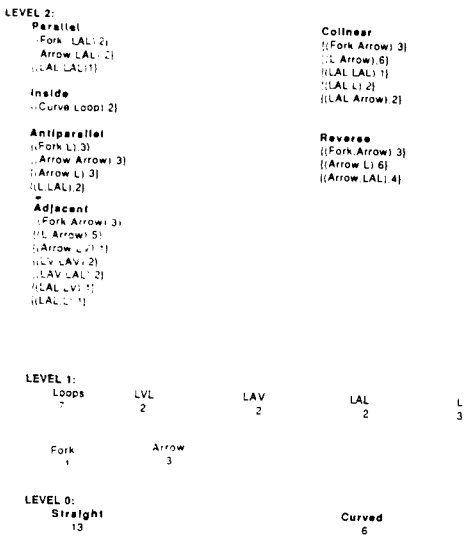

Figure 3 *illustrates a summary structure for the relational pyramid of Figure 2b.*

view; the probabilities are now chosen by the experimenter, but will eventually come from the prediction module. Using the summary matching described above with the enlarged set of summaries, we can select one or more view classes. A Bayesian analysis will then tell us which is the most probable view.

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, we expect that a backtracking tree search, using discrete relaxation, will be able to rapidly find the best mapping from view class structure to view structure. In particular, the higher-level relations are expected to 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.

6. Summary

We have described a system for pose acquisition. The system consists of modules for predicting features that will appear in 2D images from 3D models, clustering views into view classes, and using the view classes to rapidly determine the pose of the object. The system is currently being implemented in the Intelligent Systems Laboratory at the University of Washington.

References

1. 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.
2. 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.
3. 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.
4. 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.
5. Korn, M. and C. Dyer, "3-D Multiview Object Representations for Model-Based Object Recognition", *Pattern Recognition*, Vol. 20, No. 1, 1987, pp. 91-103.
6. 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.
7. Shapiro, L.G. and R. M. Haralick, "A Metric for Comparing Relational Descriptions", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-7, No. 1, 1985, pp. 90-94.
8. Shapiro, L.G. and R. M. Haralick, "A Hierarchical Relational Model for Automated Inspection Tasks", *First IEEE Conference on Artificial Intelligence Applications*, Denver, Dec. 1984, pp. 207-210.
9. Voelker, H.B. and A. A. Requicha, "Geometric Modeling of Mechanical Parts and Processes", *Computer*, Vol. 10, No. 12, Dec. 1977, pp. 48-57.