A Robot Vision System for Recognizing 3-D Objects in Low-Order Polynomial Time

C. H. CHEN AND A. C. KAK, MEMBER, IEEE

Abstract —**The two factors that determine the time complexity associated with model-driven interpretation of range maps are: 1) the particular strategy used for the generation of object hypotheses; and 2) the manner in which both the model and the sensed data are organized, data organization being a primary determinant of the efficiency of verification of a given hypothesis. 3D-POLY, a working system for recognizing objects in the presence of occlusion and against cluttered backgrounds is presented. The time complexity of this system is only $O(n^2)$ for single object recognition, where n is the number of features on the object. The most novel aspect of this system is the manner in which the feature data are organized for the models; we use a data structure called the feature sphere for the purpose. Efficient constant time algorithms for assigning a feature to its proper place on a feature sphere and for extracting the neighbors of a given feature from the feature sphere representation are present. For hypothesis generation, we use local feature sets, a notion similar to those used before us by Bolles, Shirai and others. The combination of the feature sphere idea for streamlining verification and the local feature sets for hypothesis generation results in a system whose time complexity has a low-order polynomial bound.**

I. Introduction

THE TASK at hand is to locate and identify instances of known model objects in a range image. The objects are assumed to be rigid. This task can, in general, be accomplished by matching features extracted from the image (scene features) with those describing models (model features). We will assume that the features are geometric in nature and can be characterized by shape, position and orientation; such features may include surfaces, edges, points, etc.

What are the desirable traits of a good system for object recognition and location? Clearly, it should be robust enough to work in multiobject scenes where the objects may be occluding one another. The complexity of the system should exhibit a low-order polynomial dependence, on, say, the number of features involved. The system should also be easy to train, meaning that it should be amenable to "learning by showing." In our context, that means that if we showed the system an object in all its external entirety then the system should automatically extract the relevant information that it would subsequently use for recognition and determination of pose.

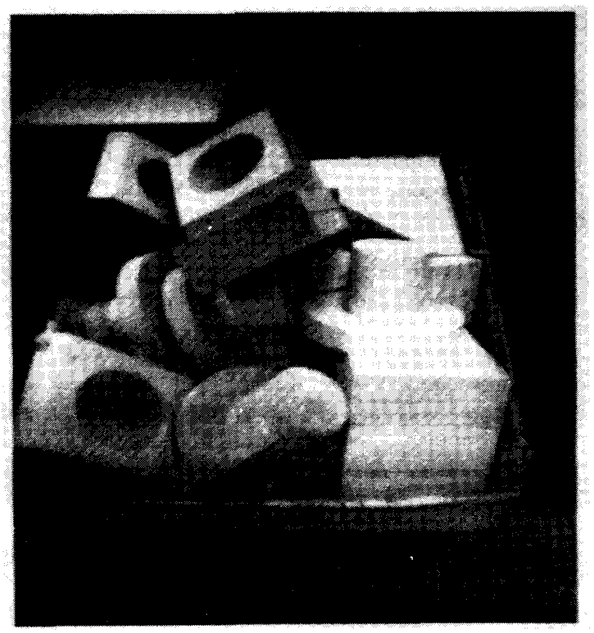


Fig. 1. Scene consisting of pile of objects.

A system with these traits is in operation in the Robot Vision Lab at Purdue. The system, called 3D-POLY, has been tested on scenes consisting of mutually occluding objects.[1] Fig. 1 is an example of such a scene which is made of two different types of objects shown in Fig. 2. Evidently, these objects, whose surfaces are of planar and conical types in convex and concave juxtapositions, do not exemplify the most difficult that can be found in the industrial world; however, their recognition in occluded

Manuscript received September 20, 1988; revised March 30, 1989.
This work was supported by the National Science Foundation under Grant CDR 8803017 to the Engineering Research Center for Intelligent Manufacturing Systems.
C. H. Chen was with the Robot Vision Lab, School of Electrical Engineering, Purdue University, West Lafayette, IN 47907. He is now with SRI International, Menlo Park, CA.
A. C. Kak is with the Robot Vision Lab, School of Electrical Engineering, Purdue University, West Lafayette, IN 47907.
IEEE Log Number 8930355.

[1]The major modules of 3D-POLY are 1) a calibration module based on the principles described in [13], 2) a range data pre-processor, 3) a recognition system, and 4) a module for automatically building models by showing. All the modules are described in detail in [12].

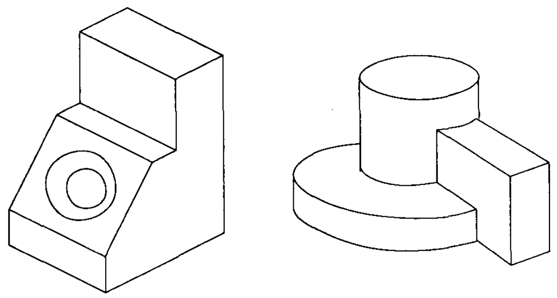


Fig. 2. Two objects that make up pile in the scene shown in Fig. 1.

environments represents a significant advance in our opinion. The various frames in Fig. 3 illustrate a successful determination of pose and identity of one of the objects whose surfaces are sufficiently visible in the scene of Fig. 1, and the manipulation of this object by the robot. For the objects in the heap, the models were automatically generated by the model-building module of 3D-POLY by placing each object in a computer controlled scanner; each object was shown in many different configurations so that the system could build an integrated "whole view" model of the object.

Although we prefer to think of the our work as a major development in 3-D robot vision;[2] more objectively, 3D-POLY should only be perceived as an evolutionary step of progress founded upon the research reported by other workers in the past [2]–[4], [6], [9], [11], [14]–[16], [18]–[21], [23]–[26], [29], [30], [32], [36]–[38], [41], [42]. A critical survey of this previous research appears in [12].[3]

The aim of this paper is not to describe all those aspects of 3D-POLY that result in the type of sensor-guided manipulation shown in Fig. 3, but only those that deal with strategies for hypothesis generation and the manner in which the model information is organized to facilitate verification. In the next section, we state more formally the problem of determining the identity and location of an object. Against a background of this problem statement, in Section III we describe features, their attributes, and criteria for feature matching. Section IV then presents the main points of our hypothesis generation and verification strategy and provides a rationale for the reduction of the computational complexity involved. In Section V, we then discuss the data structure used for representing objects and show how this data structure makes it possible for 3D-POLY to access features and their neighbors in constant time. Section VI describes how the strategy expounded in the previous section can be applied to objects in the presence of occlusions. Finally, we discuss experimental results in Section VII.

[2]Consider that on a Sun3 workstation, starting from the preprocessed range map of Fig. 5 it takes 3D-POLY less than 9 s to generate and refute about 156 pose/identity hypotheses for the scene of Fig. 1 before a verifiable hypothesis is found.

[3]The manuscript as originally submitted did include this survey of previous work, however this survey was later dropped in response to referees' demands that the manuscript be shortened.

II. Problem Statement

Before we formally state the problem of object recognition, we would like to define the more important symbols used in our presentation.

- S or S_i: A scene feature will be denoted by S. When more than one feature is under discussion, the ith feature will be denoted by S_i.
- M or M_j will denote model features.
- O_S will denote a scene object. For the purpose of explaining our hypothesis generation strategies and verification procedures, we will assume that the scene consists of a single object. However, as will be pointed out in Section VI, the entire method is easily generalizable to multiobject scenes. (Of course, its success in multiobject scenes would depend upon the extent to which an object is visible.)
- O_M will denote a candidate model object. Selection of a model object from the library of objects available to the system is part of hypothesis generation.
- n will denote the number of features extracted from the scene object.
- m will denote the number of features in the model object.
- Tr will represent the location (orientation and position) of the scene object; it is in fact a transformation consisting of a rotation $\boldsymbol{R}$ and a translation $\boldsymbol{t}$.[4] The transformation takes the object from the model coordinate system to its actual location in the world coordinate system (see Fig. 4). Actually, the model coordinate system is the same as the world coordinate system, the only difference being that in the former all model objects are supposed to reside at the origin in some standard orientation. When we say a model object resides at the origin, we mean that some reference point on the object is coincident with the origin.
- c will denote a one-to-one mapping function from scene features to model features. Although there can certainly be situations where a one-to-one mapping may not be appropriate—for example, when scene edges are broken, one may have to map more than one scene edge to the same model edge—our segmentation algorithms do manage to produce features at almost the same level of connectivity as the model features, at least for the types of scenes depicted in Fig. 1. For example, Fig. 5 shows the edge and surface features of the objects in a heap. In our implementation, a one-to-one mapping from scene features to model features is guaranteed by insisting that a model feature be used only once in the mapping process. This point should become clearer in our discussion in Section VI.

[4]We will also use Tr to denote the set of all possible solutions for the location of an object given the currently known constraints. The type of usage should be clear from the context.

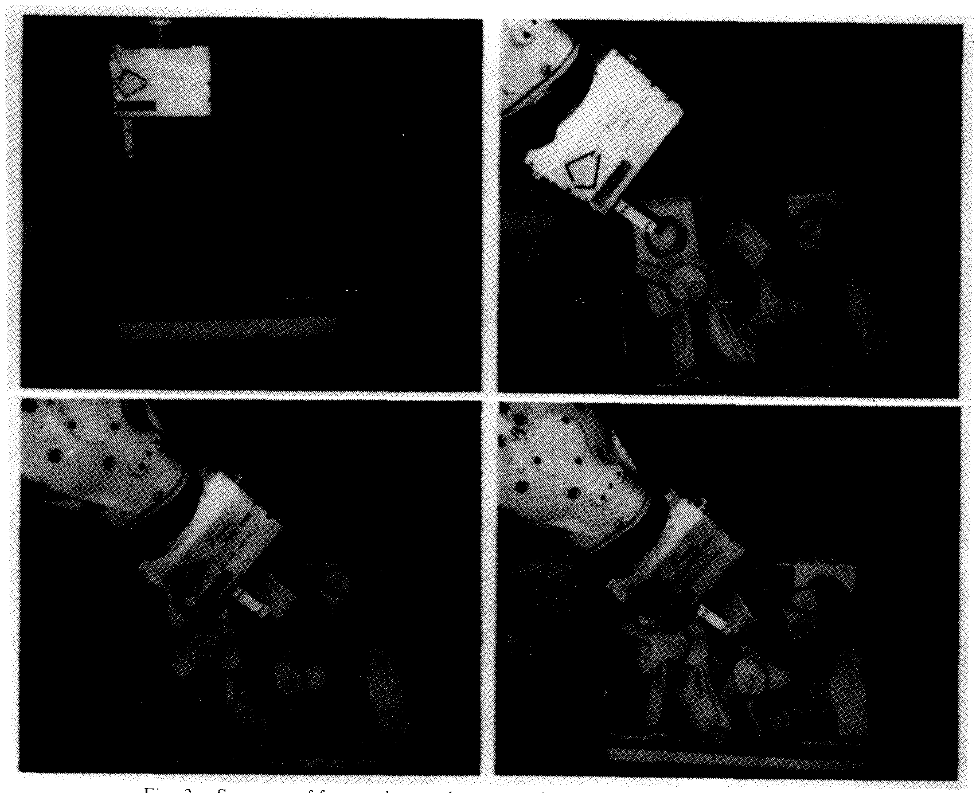


Fig. 3. Sequence of frames shows robot successfully picking up recognized object.

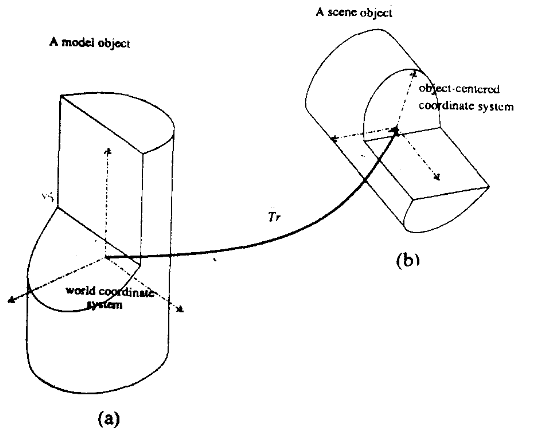


Fig. 4. Transformation *Tr* takes (a) model object into (b) scene object.

With this notation, a scene object may be represented by

$$O_S = \{ S_i | i = 1, \cdots, n \}$$

and a model object by

$$O_M = \{ M_j | j = 1, \cdots, m \}$$

where the ordering of the features is unimportant at this time; we will have more to say about the subject of ordering later, since it plays an important role in the interpretation of multiobject scenes.

Since all objects will be assumed to be rigid, if object O_S is an instance of model O_M placed at location Tr in 3-D space, then every observed scene feature of O_S must be an instance of some model feature of O_M transformed by the rotation and translation specified by Tr. One may now state our problem more formally by saying that the aim of our system is to find a model object, O_M, determine its position and orientation transform Tr and establish a correspondence c: $O_S \rightarrow O_M$ such that

$$S_i \Leftrightarrow Tr \cdot M_{c(i)} \tag{1}$$

for all $S_i \in O_S$. Here $\Leftrightarrow$ symbolizes a match between the two features S_i and $M_{c(i)}$. The criteria under which two features may be considered to match will, in general, depend on factors such as the types of features used, capabilities of the sensor, occlusion, noise, etc., and will be addressed later. Equation (1) provides a framework for discussing the problem of feature matching in general terms.

The problem of recognition and localization of a scene object may be decomposed into the following three subproblems:[5]

a) Select a candidate model O_M from the library; this generates the object identification hypothesis.

[5]We do not wish to give the reader an impression that strategies for range image interpretation must be founded on the problem decomposition shown here, only that it is the preferred approach for us (and a few other researchers in other laboratories). Approaches that do not conform to our problem decomposition do exist and some of these have been extensively investigated. See the critical survey of the literature in [12].

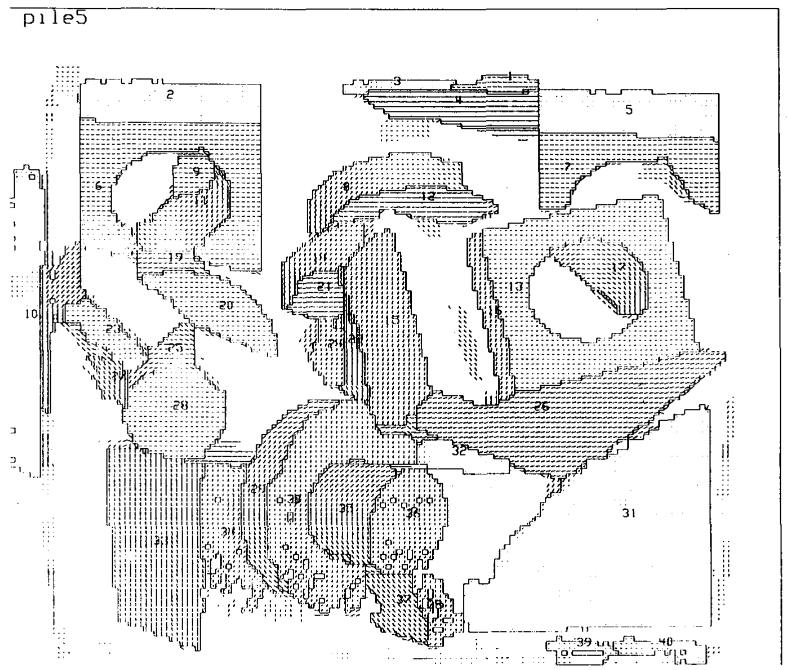


Fig. 5. This preprocessed range map for scene of Fig. 1 shows segmented regions. Needle orientations show computed local orientations of surfaces in scene.

b) Determine (estimate) the location Tr; this is the location hypothesis.
c) Establish a correspondence $\boldsymbol{c}$ between O_S and O_M that satisfies (1). Such a correspondence constitutes verification of the hypothesis.

The size of the solution space for the first subproblem is proportional to the number of model objects in the model library. For the second subproblem, one has to contend with the six degrees of freedom associated with the transformation Tr, three of these being associated with the position vector $\boldsymbol{t}$ and the other three with the orientation matrix $\boldsymbol{R}$. If we use $\mathbb{R}^{Tr}$ to denote the solution space associated with the second subproblem, it is given by

$$\mathbb{R}^{Tr}=\mathbb{R}^3\times[0,2\pi]\times[0,2\pi]\times[0,\pi]$$

where $\mathbb{R}$ stands for the real line, and, therefore, $\mathbb{R}^3$ stands for the solution space corresponding to all possible solutions for the translational vector $\boldsymbol{t}$. The solution space associated with the third subproblem is obviously of size m^n, since, in general, one must allow for all possible ways in which the n scene features can be matched with the m model features. Therefore, we can write the following expression for the solution space associated with the problem represented by (1):

$$\#\_\text{of\_models}\times\mathbb{R}^{Tr}\times m^n$$

In general, any solution to the problem of matching a scene object O_S with a candidate model O_M can, for the purposes of analyzing complexity and efficiency related issues, be conceived of as a tree search, as for example shown in Fig. 6. A traversal through the search tree may be referred to as a recognition process, each arc in the traversal representing an attempt at scoring a match between a scene feature and a model feature. Each node in a traversal may be considered to represent the current state of the recognition process, where the current state is specified by $(\boldsymbol{c}^*, Tr^*)$ with $\boldsymbol{c}^*$ being a partial correspondence list established so far and Tr^* representing the partial solution to the determination of object location. Clearly, the initial state of tree search, represented by the root, should be $(\boldsymbol{c}^*=\varnothing,\ Tr^*=\mathbb{R}^{Tr})$. Through the tree search, we incremently construct the correspondence c^* on the one hand

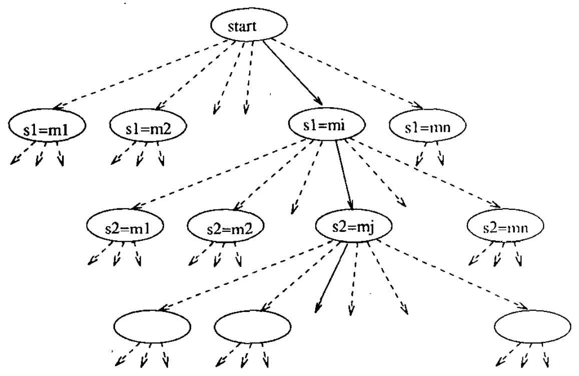


Fig. 6. Data driven tree search where s is scene feature and m is model feature.

and contract the solution space of Tr^* on the other. A model object is considered to be an acceptable interpretation of a scene object if the traversal reaches one of the terminal nodes. Since reaching a terminal node merely means that all of the n scene features have been matched, it clearly does not constitute a sufficient condition for object recognition; all that can be said is that the model is an acceptable interpretation. If no traversal in the tree search is able to arrive at a terminal node then the candidate model object must be rejected. Note that the tree search depicted in Fig. 6 represents a data-driven approach to the recognition process, the search being data driven because the sequence of matches is controlled by the order of the scene features. Alternatively, the recognition process may be cast in a model-driven framework, as shown in Fig. 7, where the sequence of matches is controlled by the order in which the model features are invoked. The time complexity associated with model-driven search is $O(n^m)$. For recognizing isolated single objects, n must be less than m; for the sake of argument, if we assume that the objects are shaped such that only half the features are visible from the viewpoint used, n will be approximately equal to $m/2$. Since $m^{(m/2)}$ is less than $(m/2)^m$, one could therefore argue that for such cases a data-driven search would be more efficient than a model-driven search.

Going back to the data-driven tree search shown in Fig. 6, since each path in the tree corresponds to a possible solution to the correspondence $\boldsymbol{c}$, and since in the worst case the search may have to sweep through the entire space (via, say, backtracking), and since the total number of nodes in the space is of the order of m^n, the time complexity of an exhaustive search on the tree is equal to $O(m^n)$. This exponential complexity, which is unacceptable for practically all applications, can be substantially reduced by using hypothesize-and-verify approaches. In this paper, we will show that it is possible to establish a hypothesize-and-verify approach in such a manner that the time complexity reduces from exponential to a low-order polynomial.

III. Features for Object Recognition

The concept of a feature normally implies some saliency which makes it especially effective in describing objects and in matching. Since the recognition of objects will be solely based on shape, of primary interest are geometric features, such as edges, vertices, surfaces together with their mathematical forms, etc. Such features specify three dimensional shape, in contrast with features like surface texture, color, etc. These latter features, although important for recognition of objects by humans, will not be addressed in this paper, since they can not be detected in range images. In 3D-POLY we consider only those geometric features with which we can associate positions or orientations. For example, a vertex feature has associated with it a position vector, which is the vector from the origin to the vertex. Similarly, a cylindrical-surface feature has associated with it an orientation, which is the unit

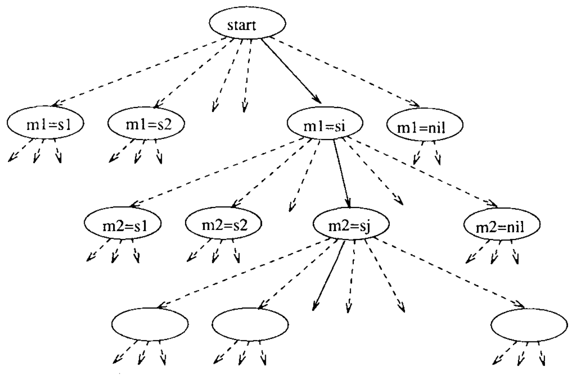


Fig. 7. Model driven tree search where s is scene feature and m is model feature.

vector parallel to the axis of the cylinder. We will categorize the geometrical features into three different classes: primitive surfaces, primitive edges, and point features.

Primitive surfaces include planar surfaces, cylindrical surfaces and conic surfaces, which are three special cases of quadric surfaces. Primitive edges refer to straight-line features or ellipsoidal-curve features. Point features consist mainly of object vertices and those surface points that have distinctive differential-geometrical properties; surface points falling in the latter category exhibit maximal or minimal curvatures or can be saddle points. These three classes of features are effective in describing the shape of an object, and, what is more, they can be reliably extracted from range images. Note that we are only using simple geometrical features, as compared to more complex ones, like generalized cylinders [31] and primitive solids [34], that are often difficult to extract from range imagery.

A. Attributes of Features

We represent a feature by a set of attribute-value pairs, each pair being denoted by $\langle a: v\rangle$, where a is the name of the attribute and v its value. The value of an attribute can be a number, a symbol, a set of labels of other features, or a list of some of these, depending on the nature of the attribute. For example, the surface feature $s2$ of the model object in Fig. 8 may be described by

⟨surface_type: cylindrical⟩
⟨number_of_pixels: ⟩
⟨radius: 3⟩
⟨normal: ⟩
⟨axis: (0.0, 0.0, 1.0)⟩
⟨area: 3⟩
⟨moment_direction: ⟩
⟨point_on_axis: (0.0, 0.0, 0.0)⟩
⟨number_of_adjacent_regions: 3⟩
⟨types_of_edges_with_adjacent_regions: (convex, convex, convex)⟩
⟨adjacent_regions: $\{s1, s3, s4\}$⟩ ⋯

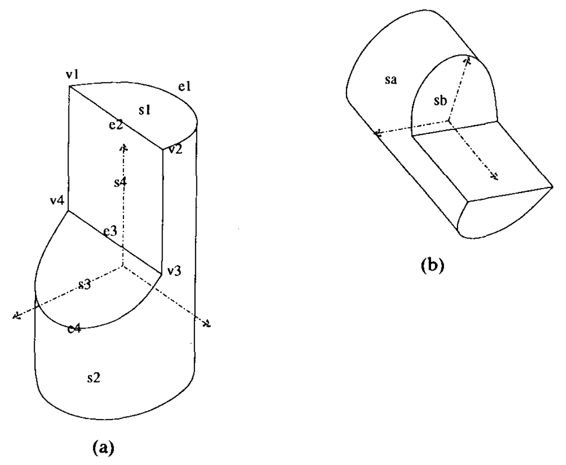


Fig. 8. Labels of primitive surfaces, primitive edges and vertices of object shown in Fig. 4. (a) Model object. (b) Scene object.

As the reader can see, not all the attributes may be instantiated for a given feature. For the cylindrical feature shown here, the attributes normal, moment_direction, etc., are undefined and are therefore left uninstantiated. The attributes of a feature, according to their geometric and topological characteristics, can be categorized on the bases of shape, relations and position/orientation. Each of these categories will be discussed in greater detail as follows.

Shape Attributes: A shape attribute, denoted by *sa*, describes the local geometric shape of the feature. For example, "surface_type", "radius", and "area" are some of the possible shape attributes of surface feature $s2$. Ideally, a shape attribute should be transformation invariant, i.e., independent of the object's position and orientation, in practice when a feature is seen through a sensor, some of its shape attributes may "look" different from different viewpoints. Therefore, we make a distinction between two different types of shape attributes, those that are viewpoint independent and those that are not. For example, the area of a surface and the length of an edge are viewpoint dependent, because they may vary with viewing direction due to occlusion. On the other hand, the attributes surface-type and radius are viewpoint independent. (Of course, we do realize that in high-noise and high-occlusion situations, a precise estimation of, say, the radius of a cylindrical surface may be difficult and may even become viewpoint dependent.) Clearly, when comparing shape attributes for matching a scene feature to a model feature, we should take this viewpoint-dependency into consideration.

Relation Attributes: A relation attribute, denoted by *ra*, indicates how a given feature is topologically related to other features. For example, for the surface feature $s2$, the attribute "⟨adjacent_to: $\{s1, s3, s4\}$⟩" indicates its adjacency with three other surfaces. Relation attributes should also be independent of transformation. An attribute "⟨on_top_of: s1⟩" is not a proper relation attribute for feature $s2$ because it depends on the pose of the object.

Position/Orientation Attributes: A position/orientation attribute, denoted by *la*, specifies position and/or orientation of a feature with respect to some coordinate system. In general, the position/orientation attributes of a scene feature are measured with respect to a world coordinate system, and those of a model feature are measured with respect to a model-centered coordinate system. As with shape attributes, some position/orientation attributes may be viewpoint-dependent, such as the centroid of a surface, or the midpoint of an edge; while others may be viewpoint-independent, as with the attributes surface-normal of a planar surface or the axis of a cylindrical surface, etc. The viewpoint-dependent position/orientation attributes should be avoided in the estimation of *Tr* during hypothesis generation (see next section), however, they can be useful for rapidly eliminating the unmatched features during the verification process.

Since a feature may possess more than one shape attribute—for example, the feature $s2$ in Fig. 8 possesses the shape attributes surface-type, area, radius, etc.—we will use the symbol *SA* to denote the set of shape attributes associated with a feature. Similarly, we will use symbols *RA* and *LA* to denote the sets of relation and position/orientation attributes of a feature, respectively. Therefore, the feature edge $e2$ of the object in Fig. 8 may be described by

$$SA(e2) = \{\langle \text{shape}: \text{straight}\rangle, \langle \text{length}:\ 3\rangle, \cdots\},$$
$$LA(e2) = \{\langle \text{direction}: (0.0, 1.0, 0.0)\rangle,$$
$$\langle \text{point\_on\_edge}: (1.0, 0.0, 3.0)\rangle, \cdots\}, \text{ and}$$
$$RA(e2) = \{\langle \text{end\_vertex}: v1\rangle, \cdots\}.$$

In practice, we may not need to use all the three categories of attributes but only those useful for a particular application. For example, Faugeras and Hebert in their geometric matching approach [16] have used only shape and position/orientation attributes. The set of attributes used in describing features should also depend on the sensor capability and the performance of the feature extractors. Using attributes which can not be reliably detected or measured by a sensor will not contribute much to solving the problem of object recognition. In any event, a minimal requirement in deciding which attributes to use for describing features is that no two features in an image or in a model be allowed to have the same set of attribute-value pairs. This, of course, does not imply that a model or a scene not contain multiple instances of a particular feature type. To elaborate, the vertex features $v1$ and $v2$ for the model object in Fig. 8 are identical, but their attribute-value pairs will be different because of the differences in their position/orientation attributes.

B. Principal Directions of Model Features

Empirical observations show that an important characteristic associated with an object feature is what we call its principal direction. In an object-centered coordinate system, the principal direction of a feature gives us a fix on the directional position and/or orientation of the feature with respect to the other features on the object. Since, one must first establish an object-centered coordinate frame, a principal direction can only be assigned to a model object feature, or to a scene object feature if the scene object has been embedded in an object-centered coordinate frame via prior matches of some of its features with model features.

In Section V, we show how a useful data structure—we call them feature spheres—can be defined using the concept of principal directions. Here, we will define the principal directions more formally and make the reader aware of the fact that the manner in which such definitions are made is different for different classes of features, and within each class, for different types of features. As was mentioned before, different classes of features correspond to primitive surfaces, primitive edges and primitive points. Within the class primitive surfaces, we may have different types of features such as cylindrical, planar, spherical surfaces, and so on. While for some class and type of features, the principal direction represents their orientations, for others it represents their directional position on the object.

We will now formally define the principal direction, denoted by Φ, for the three classes of features and for types within each class.

1) Primitive Surfaces

Planar surface: $\Phi \equiv$ The direction of the outward surface normal.

Cylindrical surface or conic surface: $\Phi \equiv$ The direction of the axis. The 180° ambiguity associated with this direction is resolved by choosing that direction which subtends an acute angle with the positive z-axis. For axes that are perpendicular to the z-coordinate, that direction is retained that is closest to the x-coordinate. And, if the axis is perpendicular to both z and x, then we choose the $+y$ direction.

Spherical surface: Let $\boldsymbol{o}$ be the position vector to the center of the sphere in the object-centered coordinate system. The principal direction is defined by the normalized form of this position vector

$$\Phi \equiv \frac{\boldsymbol{o}}{|\boldsymbol{o}|}.$$

Note that this principal direction is different in character from that defined for a cylindrical surface. We choose this form for Φ because it is not possible to associate an orientation vector with a spherical surface.

Quadric surface: A quadric surface has the form $\boldsymbol{x}^t A \boldsymbol{x} + \boldsymbol{x} \cdot B + C = 0$

$$\Phi \equiv \frac{\boldsymbol{a}_p}{|\boldsymbol{a}_p|}$$

where $\boldsymbol{a}_p$ is the principal eigenvector of the matrix $\boldsymbol{A}$.

In general, the eigenvector associated with the largest eigenvalue is the principal eigenvector of $\boldsymbol{A}$ and the direction associated with this eigenvector usually defines the major axis of a surface. Since a quadric description includes the cases of planar, cylindrical and spherical surfaces defined above, the definition of the principal axis here must be used with care. The quadric definition is used only if a surface cannot be classified as being either planar, cylindrical or spherical.

2) Primitive Curves

Straight line: $\Phi \equiv$ line direction. The 180° ambiguity associated with the direction of a line is resolved using the same criterion as for the axis of a cylinder.

Circular or ellipsoid curve: $\Phi \equiv$ surface normal of the curve's plane. The 180° ambiguity here, too, is resolved as for the case of a cylinder.

3) Point features

Let $\boldsymbol{p}$ be the position vector of a point feature with respect to the object-centered coordinate system. The principal direction is defined by normalizing the position vector

$$\Phi \equiv \frac{\boldsymbol{p}}{|\boldsymbol{p}|}.$$

The important thing to note is that the parameters used in the definitions of principal directions are all extracted with relative ease from range maps. For example, if from a given viewpoint in a range map only about 40 percent of the round part of a cylindrical surface is visible, in most cases it would be possible to make a good estimate of the direction of the axis of the cylinder.

C. Criteria for Feature Matching

We will now provide matching criteria for the matching problem represented by (1) and express the criteria separately for each of the three attribute classes. In other words, we will express the conditions for each attribute class, conditions that must be satisfied for a scene object to match a model object. Our conditions are applicable strictly only under the noiseless case. For actual measurements, the comparisons implied by our conditions would have to treated in relation to some user specified thresholds, the magnitudes of the thresholds depending upon the extent of noise and other uncertainties in the system.

Matching Criteria for Shape Attributes: The reader will recall that we have two different types of shape attributes, those that are viewpoint independent and those that are not. A viewpoint independent shape attribute sa of a scene object feature is said to match the corresponding shape attribute of a model object feature if

$$sa(S_i) = sa(M_{c(i)}). \tag{2}$$

where the function $sa(.)$ returns the value of the attribute sa for the feature that is its argument; S_i is a feature from the scene and $M_{c(i)}$ is the candidate model feature that is under test for matching with the scene feature. The aforementioned equality must be satisfied for each $sa \in SA(S_i)$. For viewpoint dependent shape attributes, clearly we cannot insist upon an equality in the previous equation. In general, for such attributes, we require[6]

$$sa(S_i) \leqslant sa(M_{c(i)}).$$

Note that since all the viewpoint dependent shape attributes are numerical in nature, we only have to use numerical inequalities and not, say, subsets, as would be case for symbolic features. For example, we would expect the length of a scene edge to be equal to or less than the length of the corresponding model edge due to possible occlusion. Therefore, the matching criterion for this attribute can only be expressed as

$$\text{edge\_length}(S_i) \leqslant \text{edge\_length}(M_{c(i)}).$$

Matching Criteria for Relation Attributes: Relation attributes are also transformation invariant; thus if a scene feature S_i has relation ra with, say, scene feature S_l, then the model feature $M_{c(i)}$ must have the same relation ra with the model feature $M_{c(l)}$.[7] More precisely, for every $ra \in RA(S_i)$

$$c(ra(S_i)) \subseteq ra(M_{c(i)}). \tag{3}$$

To justify the nature of this comparison, consider that the model object is as shown in Fig. 8. Further, suppose that in the scene the model surfaces s2 and s3 are visible and have been labeled as, say, S_a and S_b, respectively. Then, from the model description, we have

$$\text{adjacent\_to}(s3) = \{s2, s4\}$$

and, from the scene information,

$$\text{adjacent\_to}(S_b) = \{S_a\}$$

Let's say that during the hypothesis generation phase, an estimate was made for the transformation that takes the model object into the scene object. Let's further say that using this transformation, we have already established the correspondence of the scene surface S_a with the model surface s_2, and, now, we are testing the correspondence of the scene surface S_b with the model surface s_3. We see that since adjacent_to(S_b) = S_a, and since $c(S_a) = s_2$, a substitution in (3) yields for ra = adjacent_to

$$\{s2\} \subseteq \{s2, s4\}$$

which being true implies that the scene surface S_b can indeed be matched with the model surface s_3, at least from the standpoint of satisfying relational constraints. The point to note is that in the matching criterion of equation (3) the features participating in a relation at a given scene feature S_i should be a subset of the features participating at corresponding model feature $M_{c(i)}$; it is not possible to replace the "subset" comparison with a strict equality because not all of the model surfaces may be visible in the scene.

Matching Criteria for Position/Orientation Attributes: For a viewpoint independent position/orientation attribute la, such as the location of a vertex or the direction of an edge, the matching criterion is described by

$$la(S_i) = \boldsymbol{R} \cdot la(M_{c(i)}) \qquad \text{if } la \text{ is an orientation vector,} \tag{4a}$$

$$la(S_i) = \boldsymbol{R} \cdot la(M_{c(i)}) + \boldsymbol{t} \qquad \text{if } la \text{ is a position vector.} \tag{4b}$$

for every $la \in LA(S_i)$. These criteria play a vital role in the localization of a scene object. Recall that the matching process starts with $Tr = \mathbb{R}^{Tr}$ and should end up with a unique solution which is the location of the scene object, else it should fail. Before Tr has converged to a unique solution, (4a) and (4b) provide a system of equations to solve for Tr, but after that, they are nothing but two predicates which confirm or reject a match between the scene and the model features.

Many important position attributes are not viewpoint independent, yet they are important. For example, the position attributes point_on_axis and point_on_edge shown in Section III-A are both viewpoint dependent since these points can be at arbitrary locations along their respective directions. Despite their arbitrary locations, these points play a vital role during the verification phase of matching. To illustrate, let's say a scene edge is under consideration for matching with a model edge under a given pose transformation. Now, if the directions of the two are identical, that's not a sufficient criterion for the match to be valid, since the identity of directions merely implies that the scene edge is parallel to the model edge. To impose the additional constraint that the two edges be collinear, we need the point_on_edge attribute even if the point is arbitrarily located on the edge in the model space. The point_on_edge attribute is used in the following fashion: First the difference vector between the vector to point _on_edge and a vector to some arbitrary point on the scene edge is computed. Then the cross-product of the difference vector with the direction attribute of, say, the model edge is calculated. The magnitude of this cross-product should be close to zero for the match to be acceptable. Note that while the cross-product being zero guarantees the collinearity of the model and scene edges, it still allows one degree-of-freedom between the two. It is impractical to completely constrain this remaining degree-of-freedom since in the presence of occlusions the model edge may not be completely visible in the scene.

Before concluding this subsection, we should mention that, in a manner similar to edges, the position attribute associated with a planar surface, specified by giving the coordinates of an arbitrary point on the surface, can be

[6]The reader beware that it is possible to define attributes that would be viewpoint dependent and whose values may actually increase with occlusion. As was pointed out to us by one of the reviewers, one such attribute would be *compactness*, calculated by using the formula perimeter^2/area. No such attributes are used in 3D-POLY.

[7]The reader might question our assertion by saying that in the presence of occlusions, adjacent surfaces in an image of a scene may actually not be adjacent at all in the model space. But note that 3D-POLY uses range maps for input and wherever there are self or other kinds of occlusions, there will also be jump discontinuities in the data. Two scene surfaces are not allowed to satisfy adjacency relationship if they are connected by only a jump discontinuity in the range map.

TABLE I
SUMMARY OF HGF SETS

Configuration of Features	Position/Orientation Attributes
Three unique, noncollinear points.	The three positions vectors associated with the three points.
One straight edge and one noncollinear point.	The orientation attribute associated with the direction of the edge, the position attribute associated with a point on the edge, and the position attribute associated with the noncollinear point.
One ellipsoidal edge and one noncollinear point.	The orientation attribute associated with the edge, the position attribute associated with a point on the axis of the ellipsoidal edge, and a position attribute associated with the noncollinear point.
Two primitive surfaces and one point.	The two principal directions associated with the two surfaces, and a position attribute associated with the extra point.
Three noncoplanar primitive surfaces.	The orientation attributes associated with any two of the surfaces, and three position attributes associated with some three points, one on each surface.

used to make sure that a model surface is coplanar with the corresponding surface from the scene; again, the identity of surface orientations is not sufficient, and to cope with occlusions, it is not possible to constrain the two any further.

IV. Matching Strategy

As mentioned in the introduction, the recognition method employed in 3D-POLY is based on hypothesis generation and verification. In this section, we will explain how hypotheses are generated and then how each hypothesis is verified.

In what follows, we will first show that if hypotheses are generated by exhaustive search, meaning that a scene feature is tested against every possible model feature, then the time complexity of the recognition procedure is $O(n \times m^{(h+1)})$, where n is the number of scene features, m the number of model features, and h the number of features used for hypothesis generation. Unfortunately, this complexity reduction is not sufficient for most practical purposes. We then proceed to show how by using the notion of local feature sets for generating hypotheses and using the feature sphere data structure for verification, the complexity can be improved to $O(n \times m \times h!)$. Finally, we will show that when we use vertex features to organize the local feature sets, the complexity is further improved to $O(n^2)$.

A. Hypothesis Generation and Verification

It is rather well known that only a small number of features is necessary to estimate the transformation matrix Tr that gives the pose of a scene object in relation to the corresponding model object [8], [37]. In our work, this small number of features will be referred to as a hypothesis generation feature set (HGF). Clearly, it is the position/orientation attributes for the features in an HGF that must be used for the estimation of Tr. We have shown some possible HGF sets and the position/orientation attributes used for determining Tr in Table I. Note that the table is not an exhaustive listing of all possible HGF sets, but only those which are rather frequently used. How exactly a Tr may be constructed from the position/orientation attributes of the different possible HGF sets may, for example, be found in [8], [16], [21], [37]; each of these references discusses the method used to calculate a Tr for the type of HGF used.

The reader might like to note that for each HGF set in Table I, the position/orientation attributes shown constitute the least amount of information that is required for the calculation of the pose transform using that set. That this is so should be obvious for the first set. For the second set, we need to know the coordinates of at least one point that is arbitrarily located on the straight edge. Without this extra information, the rotation transform computed from just the edge directions would not also "move" the edge from its model space to the scene space; the point on the straight edge helps us make the model edge become collinear with the scene edge. Then, we can use the extra noncollinear point to constrain the rotation of the object around the edge. The same argument applies to the third HGF entry. The attributes listed for the fourth HGF set should be obvious. Finally, for the last HGF set, while two orientation vectors are sufficient to give us the rotation transform, coordinates to three points, located on each of the surfaces, are needed to constrain the translation vector. Note that these three points can be at arbitrary locations on the surfaces in the model space, the same being true in the scene space.

Let's assume that a recognition procedure needs a maximum of h features to construct a hypothesis for the pose transform Tr for a candidate model object. We will further assume that we have somehow selected a subset of cardinality h of the scene features, this subset will constitute the HGF set and will be represented by $\{S_1, S_2, \cdots, S_h\}$; we wish to generate hypotheses by using the features in this subset. We may then divide the search tree in Fig. 6 into two parts as shown in Fig. 9, the division occurring at level h on the tree. Note that at the first level of the tree, we try to match the scene feature S_1 against all possible model features from the candidate object. Then, at the second level, at each node generated by the first level, we try to

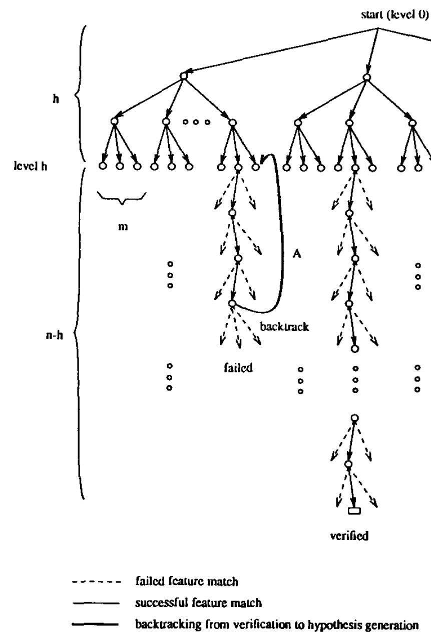


Fig. 9. Data driven search tree is divided into two parts at level h of tree. First part represents hypothesis generation stage while the second part represents verification stage.

match the scene feature S_2 with every possible model feature; and so on.

As depicted in the figure, after a hypothesis is formed with h features, we use the remaining $n-h$ features for verification. In principle, if a hypothesis is correct, i.e., the scene object O_S is indeed an instance of the candidate model O_M at location Tr, we should then be able to find all the remaining $n-h$ matched feature pairs using the transformation Tr. This implies that in the verification phase the scene feature at each level will match with exactly one model feature. This uniqueness is guaranteed by the requirement that no two features of a model have the same description. To reiterate what was said in Section III, on account of the different position/orientation attributes this requirement is easily met even for those features that might otherwise be identical, say, by virtue of their similar shapes.

On the other hand, if any one of the remaining $n-h$ features can not be matched to a model feature, that implies the current hypothesis is invalid, because either O_M is not the right object model or Tr is not the correct transformation. Therefore, when a scene feature, say, S_k, $k > h$, does not match any model features under the candidate Tr, it serves no purpose to backtrack to level $k-1$ or higher. Instead, the system should go back into the hypothesis generation phase, and by backtracking over the first h levels, try to generate another hypothesis for Tr, as illustrated by arc A in Fig. 9. Clearly, if repeated backtracking over the first h levels fails to produce a valid Tr, the candidate model object should be discarded and new tree search begun with a new candidate object. In the rest of this subsection, we will explore the time complexity associated with this type of search.

This search process is exhaustive over the model features in the sense that at every node shown in Fig. 9, a scene feature must be compared with all the unmatched features of the candidate model object. Therefore, at each node, the complexity is equal to $O(m)$, where m is the number of features in the model object. The number m is also the fan-out at each node encountered during the hypothesis generation phase, i.e., in the first $h-1$ levels of the search space.[8] However, the fan-out in the verification phase equals 1 because of our requirement that a match failure during verification implies going back into hypothesis generation.

Since backtracking is allowed to be exhaustive during the hypothesis generation phase, the time complexity associated with hypothesis generation is $O(m^h)$. The time complexity associated with the worst case verification scenario can be estimated by noting that each verification path has at most $n-h$ nodes, and, since at each node we must make m comparisons, the complexity of verification is $O(m \times n)$. Therefore, the overall complexity associated with this recognition process is

$$O(m^h) \times O(m \times n)$$

which is the same as

$$O(n \times m^{h+1})$$

For rigid objects, h will typically be equal to 3, although its precise value depends upon how carefully the HGF sets are constructed. Therefore, the expression for the complex-

[8]Strictly speaking, the fan-out will be m at the root node during hypothesis generation, $m-1$ at the next level, and so on. Notwithstanding this fact, our complexity measures remain valid.

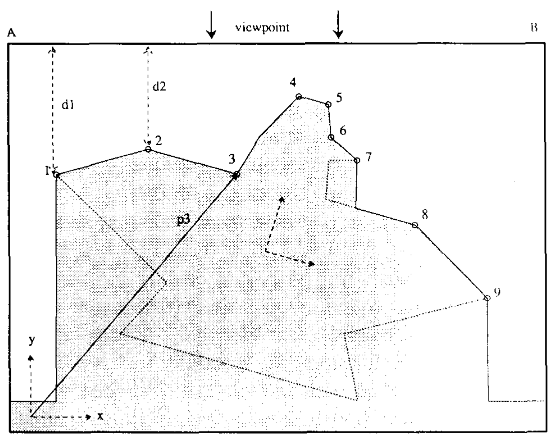


Fig. 10. Two-dimensional range image of hypothetical polygon. Range values are proportional to perpendicular distance from line AB. For example, range corresponding to scene point 1 is equal to distance $d1$.

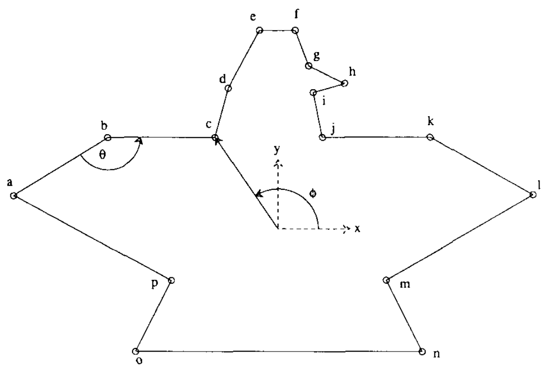


Fig. 11. Model of polygon whose 2-D range map is shown in Fig. 10.

ity function becomes

$$O(n \times m^4)$$

Although one may consider this complexity function to be a substantial improvement over the $O(m^n)$ function associated with the search tree of Fig. 6, it is still not acceptable for practical applications. In the next subsection, we will show how by constraining the selection of model features for matching we can make further reductions in the complexity.

B. How to Constrain the Selection of Model Features

In this subsection, we will explore the question of what constraints one should invoke to select model features for matching with a scene feature. Given that the comparison of attribute values plays a central role in the matching process, the constraints we are looking for should be derivable from the attributes. But, since we have three different kinds of attributes, namely, shape, relation, and position/orientation, the question that arises is which of these attributes are best suited for the required constraints.

To answer this question, we will take the reader through a two-dimensional (2-D) example shown in Fig. 10. With the help of this example we will convince the reader that the attributes used for constraining the selection of model features should depend upon whether or not we know the transform Tr. In other words, the constraints used in the hypothesis generation phase must, of necessity, be different from those used in the verification phase. We will show that for the hypothesis generation phase, we must take recourse to an idea suggested and used by other researchers: the model feature that is invoked for comparison against a scene feature should depend upon its relations with the previous model features in the path traversed so far in the search space of Fig. 6. And, for the verification phase, we will show that remarkable reductions in computational complexity can be achieved by using constraints derived from the principal directions of features—recall that the principal direction of a feature is derivable from its position/orientation attributes. We will then show that the invocation of constraints on the principal directions is greatly facilitated if the features are organized according to a special data structure we call the feature sphere.

To help explain our points, in Fig. 10 is shown a 2-D range image of a polygonal object. The viewpoint is from the top, as illustrated. Range mapping is orthogonal, meaning that the lines of sight for the determination of range values are parallel; for example, the range at scene point 1 is equal to the distance $d1$, and $d1$ is parallel to $d2$, the range at scene point 2. The model polygon is shown in Fig. 11. The problem is to recognize and locate the polygon in Fig. 10 given its model in Fig. 11. We will assume that the recognition system is using only vertex features (an example of primitive point feature type). Scene vertices will be denoted by integers $1, 2, 3, \cdots$, and those of the model by letters, $a, b, c, \cdots$. From the viewpoint shown, only the model vertices $a, b, c, d, e, f, g, h, k, l$ are visible to the sensor. For the sake of argument, we will assume that of these vertices, the model vertex d is not detectable; therefore, its correspondent in Fig. 10 has not been given a label. The undetectability of d in the sensor data could be due to the fact that the angle defining that vertex is not very sharp. As a result, feature extraction from the sensor data will only yield the vertices $1, 2, 3, 4, 5, 6, 7, 8, 9$. For the viewpoint shown, we must further assume the unavailability of angle measurements at vertices 1, 7, and 9.

We will subject the recognition of the polygonal object in Fig. 10 to the kind of hypothesis and verify approach depicted in Fig. 9, except that we will add constraints on the selection of model features at each node. We will first examine the possibility of using constraints derived from shape attributes.

1) Using Constraints Derived from Shape Attributes: Let us say each scene feature is characterized by the following set of shape attribute-value pairs:

$$SA = \langle sa_i, v_i \rangle.$$

In the absence of uncertainties, perhaps the most straightforward way of constraining the selection of model features in the matching process is to invoke only those model features whose sa_i values are the same as v_i. For the 2-D example, say that at a node in the search space the scene vertex 2 is under consideration. Now, a possible shape attribute for a 2-D vertex is the dihedral angle θ shown for one of the vertices in Fig. 11. Let's say the measured dihedral angle at the scene vertex 2 is θ_2. Given this shape information, it should be necessary to invoke only those model vertices whose dihedral angles, denoted here by θ_x, satisfy the constraint

$$\{x: |\theta_x - \theta_2| < \epsilon\},$$

where ϵ represents the uncertainty in angle measurement. Given a judicious choice for ϵ, such a constraint might only invoke the model vertices b and k—a considerable improvement over having to compare, in the worst case, of course, the scene vertex 2 with all the 16 model vertices.

A practical implementation of the above idea would require that we organize the model features according to the shape attributes. One could do so, for example, by sorting the model features by the values of the attributes. Then given a desired attribute value for a scene feature, the candidate model features could be retrieved by a binary search. Another way is to use an array with each array cell representing an interval of the attribute value; a model feature could then be assigned to an appropriate cell on the basis of the value of the attribute. The latter method would, in general, be more demanding on memory requirement, but the retrieval of candidate model features for a given scene feature attribute value would be more efficient.

Although, in some cases, it would certainly be possible to benefit from the ideas outlined in the previous two paragraphs, we have chosen to not use shape attributes for constraining the selection of model features. Our most important reasons are that the viewpoint independent shape attributes, for the most part, do not contain sufficient discriminatory power for adequately constraining the selection of model features; and the viewpoint dependent shape attributes are too prone to getting their values distorted by occlusion and, of course, the change in viewpoint.

For example, planarity of a surface is a viewpoint independent shape attribute. Now consider the example illustrated in Fig. 8 and assume that we are matching scene surfaces with model surfaces using planarity as a shape attribute. Clearly, all the model surfaces but $s2$ would become candidates for scene surface s_b, and there would be almost no gains in the computational complexity. On the other hand, a viewpoint dependent shape attribute, like the area of a surface would obviously be useless because the problems that could be caused by occlusion. For another illustration of the difficulties caused by using viewpoint dependent shape attributes, let's go back to the matching of vertices in Fig. 10. Although it may not seem so, the dihedral angle is viewpoint dependent, as, for example, evidenced by the vertices 1, 7, and 9. The dihedral angles at these vertices can not be measured from the viewpoint shown in Fig. 10 because of self-occlusion. Therefore, it would be impossible to use the most obvious shape attribute—the dihedral angle—for constraining search, as any of the nodes could suffer from self-occlusion, depending upon the pose of the object.

2) Using Constraints Derived from Relation Attributes: Let's say a scene feature S has the following relation attribute-value pair $\langle ra\colon \{S_1, S_2, \cdots, S_k\}\rangle$, meaning that the scene features $S_1, S_2, \cdots, S_k$ are participating with S in the relation named ra. We will assume that of $S_1, S_2, \cdots, S_k$, the features S_1 through S_p, $p < k$, have already been matched with model features. Then we may consider only those model features for matching against S that enter into relation ra with the model features that match S_1 through S_p. More formally, a model feature M will be selected for matching with S provided one of the relation attribute for M is $\langle ra\colon \{M_{c(1)}, M_{c(2)}, \cdots, M_{c(p)}\}\rangle$. Remember, the mapping function c gives us the correspondences between the scene features and the model features.

In the example of Fig. 10, assume that the scene vertices 4 and 5 have already been matched with the model vertices e and f, and that the scene vertex 6 is now under test for possible match with a model vertex. Since vertex 6 has relation attribute $\langle$adjacent_to: $\{5,7\}\rangle$, a candidate model feature for matching with vertex 6 should possess the relation attribute $\langle$adjacent_to: $\{f,*\}\rangle$, where we have used an asterisk to act as a place-holder for the yet unknown corresponding model vertex of vertex 7. For symmetric relations, such as adjacent_to, a search for those model features that satisfy the desired constraint can be easily conducted by examining the relations at the vertex f; we may thus conclude that the model vertex g is a candidate model feature for the scene vertex 6. For nonsymmetrical relations, unless care is exercised in organizing the model feature with respect to their relations, in the worst case one may have to search through all the model features to determine those which satisfy the required constraints. However, even with such a search, one would gain from the subsequent savings in not having to match all the model features with a scene feature. In addition to having to search for the model features, there are other issues that play an important role in matching scene features with model features under relational constraints, especially when one also has to contend with the uncertainties associated with real data. Over the years, much has been done in this area and the reader is referred to [7], [27], [35], [39] for further details.

Although what we have said so far in this subsection may be construed as implying the appropriateness of relational constraints, the reader beware. We will now show that there can be situations when relational constraints may not help at all with the pruning of model features, and, further, in some cases they can lead to results that may be downright incorrect.

Going back to our example of Fig. 10, we just showed how the prior matches at scene vertices 4 and 5 help us constrain the search at vertex 6. One may similarly show that the matches at the vertices 5 and 6 help us with the selection of candidate model vertices at 7. Now, let's say, that the scene vertex 7 has been successfully matched with the model vertex h. Our next task is to find a list of candidate model vertices for the scene vertex 8. However, because of self-occlusion, there does not exist at vertex 8 the relation attribute ⟨adjacent_to: ..,7,..⟩. This means that prior matching history along the path being traversed in the search space would not help us at all at the scene vertex 8.

Now to show that relational constraints may lead to erroneous matches, consider the scene vertex 3, where we have the relation ⟨adjacent_to: 2,4⟩. We will assume that the vertex 4, being closest to the viewpoint line AB, has already been matched with model vertex e. The candidate model vertices for the scene vertex 3 must satisfy the relation ⟨adjacent_to: *, e⟩, since e is the correspondent of 4 and since the vertex 2 has not been matched yet. This constraint would cause 3 to be matched with the model vertex d—an obviously incorrect result which would eventually cause an erroneous rejection of the model object.

3) Using Constraints Derived from Position/Orientation Attributes: If a scene feature possesses a position/orientation attribute-value pair $\langle la: v\rangle$, then it follows from (4a) and (4b) that a potential candidate model feature must be characterized by the either or both of the following position/orientation attribute-value pairs

$$\langle la: \boldsymbol{R}^{-1}\cdot v\rangle \qquad \text{if } v \text{ is an orientation vector} \quad (5a)$$

$$\langle la: \boldsymbol{R}^{-1}\cdot (v-\boldsymbol{t})\rangle \qquad \text{if } v \text{ is a position vector} \quad (5b)$$

where $\boldsymbol{R}$ and $\boldsymbol{t}$ are the rotation and translation components, respectively, of the transformation Tr that takes the model object into the scene object. Clearly, an estimate of the transformation Tr is required before a position/orientation constraint can be invoked.

We must again address the issue of how one might organize model features in order to efficiently invoke the position/orientation constraints. One approach would be to partition the space of all possible positions and orientations into cells and to assign model features to appropriate cells. Since it takes three parameters to specify an orientation, two to specify the direction of the axis of rotation and another one to specify rotation around this axis, the space of all possible orientations will consist of either the volume of a unit sphere, or, using the quaternion notation, the surface of a four-dimensional unit sphere. On the other hand, the space of all possible positions will be the 3-D Cartesian space; note that by positions we mean translations.

While it would indeed be possible to use the position/orientation constraints in this manner to prune the list of candidate model features, difficulties arise in practice on account of the fact that it may not be possible to develop a unified organization of model features on the basis of position/orientation information, since some features may

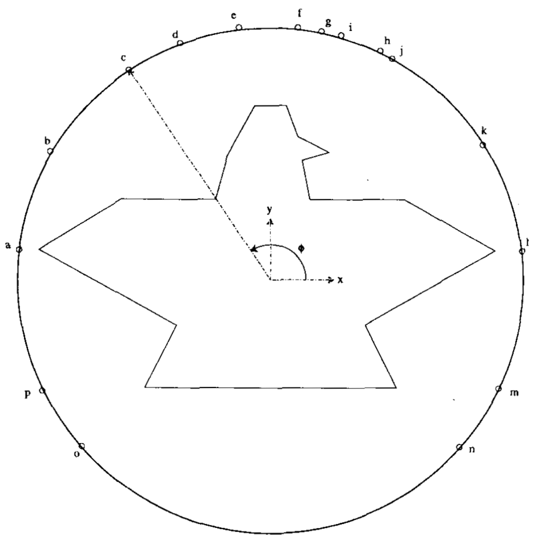


Fig. 12. Vertices of model polygon are pushed out to unit circle; we call this circle the feature circle of the model.

have only position attributes, others only orientation attributes, and still others both.

Fortunately, there is a way out of this impasse, by the use of principal directions defined in Section III. For every feature, as shown in that section, we can derive its principal direction from either the position information or the orientation information. The principal direction can then be used, by the method discussed below, to organize the model features for efficient retrieval subsequently. In the rest of this subsection, we will use the 2-D example of Fig. 12 to introduce the idea of a *feature circle*, which is a means to organize, on the basis of their principal directions, the model features for the 2-D case.

For the 2-D example, we first compute the principal direction of each model vertex according to the definition in Section 3.2. Since the space of direction vectors in 2-D space is a circle, we organize the model vertices along a unit circle as shown in Fig. 12. This constitutes the feature circle for the model object. Suppose that the orientation $\boldsymbol{R}$ and position $\boldsymbol{t}$ of the scene polygon have been hypothesized by matching vertices 4 and 5 to model vertices e and f, respectively. Now, suppose we want to find the candidate model vertices for the scene vertex 3. Using (5b), the position vector of a candidate model vertex for possible matching with the scene vertex 3 should be

$$\hat{\boldsymbol{p}} = \boldsymbol{R}^{-1}\cdot \boldsymbol{p}_3 - \boldsymbol{t}.$$

The principal direction associated with this position vector is

$$\hat{\boldsymbol{\Phi}} = \frac{\hat{\boldsymbol{p}}}{|\hat{\boldsymbol{p}}|}.$$

We can then access the feature circle of Fig. 12 and pull out those model features whose principal directions lie in the interval $[\hat{\boldsymbol{\Phi}} - \epsilon, \hat{\boldsymbol{\Phi}} + \epsilon]$, where ϵ depends upon the magnitude of uncertainty in the sensed data.

Of course, for the 3-D case, the organization of the model features would not be as simple as what is shown in Fig. 12, since the features would now have to be mapped onto the surface of a sphere on the basis of their principal directions. To handle the resulting complications, in Section IV-D, we will introduce the notion of a feature sphere which use a special indexing scheme for the tessellations on the surface of the sphere. The indexing scheme chosen reduces the complexity associated with finding the neighbors of a particular cell on the surface of a sphere, and for assigning a cell to a given principal direction.

4) Conclusion Regarding the Choice of Constraints: Before we present our complete feature matching strategy, we would like to summarize the conclusions that can be drawn from the preceding three subsections.

- Relatively speaking, shape attributes are not that useful for the purpose of selecting candidate model features because, when they are view independent, they often do not carry enough discriminatory power, and, when they are view dependent, they cannot be used for obvious reasons.
- When the pose transformation *Tr* is unknown, relation attributes can provide strong constraints for selecting model features; however, the extraction of relation attributes may be prone to artifacts.
- When the pose transformation *Tr* is given, the principal direction attribute, which can be derived from the position/orientation attributes, probably provides the best constraint for selecting model features. We use the adjective "best" to emphasize, in a qualitative sense admittedly, that this attribute can be calculated in a fairly robust manner for most features, and, to emphasize its ability to provide strong discrimination amongst competing model features.

These conclusions form the foundation of our overall matching strategy, which we now present:

During hypothesis generation:
In this phase, we will use constraints on relation attributes to prune the list of model features. To get around the problems associated with exhaustive backtracking in the upper *h* levels of the search space shown in Fig. 9, we will group immediately related model features into sets, to be called local feature sets (LFS). Each LFS will be capable of generating a value for the transformation matrix *Tr*. The idea of using feature sets for constructing hypotheses about pose transformations is akin to the local feature focus idea used by Bolles and Cain [5] for the 2-D case and to the notion of kernel features used by Oshima and Shirai [32] for the 3-D case.

During verification:
In this phase, we will use the principal direction constraint to select model features. For efficient retrieval on the basis of their principal directions, the model features will be organized on feature spheres.

In the next subsection, we will elaborate on the notion of local feature sets for hypothesis generation. We will subsequently present a formal definition of the feature sphere data structure and present expressions for the complexity functions associated with our matching strategy.

TABLE II
LFS of the Object in Fig. 13

Vertex	Surfaces
a	1,10,2
b	1,2,8
c	2,10,3
d	2,3,8
e	3,10,4
f	3,4,8
g	7,12,8
h	7,10,12
i	4,10,7
j	4,7,8
k	1,8,9
l	9,8,12
m	1,9,10
n	10,9,12

C. Local Feature Sets for Hypothesis Generation

Ideally, an LFS is a minimal grouping of features that is capable of yielding a unique value for the pose transform which takes the model object into the scene object. The features in such a minimal grouping could, for example, correspond to one of the rows in Table I.

More practically, it is desirable that the features in an LFS be in close proximity to one another, so that the probability of their being simultaneously visible from a given viewpoint would be high. In our implementation, we have found useful the following variation on the above idea, which seems to lead to particularly efficient hypothesis generation strategies for objects that are rich in vertices, such as the objects of Fig. 2. We allow our LFS's to be larger than minimal groupings and insist that each grouping contain a vertex and all the surfaces meeting at that vertex. [It would be equally easy to use edges in place of surfaces.] In Fig. 13, we have shown the labeled features for one of the objects of Fig. 2. For this object, the LFS's generated with the specification that all the surfaces meeting at a vertex be included are shown in Table II. To explain the advantages of our approach, consider the LFS corresponding to the vertex *c* of the object in Fig. 13. The data record for this LFS will look like the following.

```
Vertex c
flag: -1
xyz: ##
surfaces: 2 10 3
adjacent_vertices: a e d
edge_type: v v c
```

The flag value of -1 means that one of the three edges meeting at the vertex is concave. The variable *xyz* is instantiated to the coordinates of the vertex in the model coordinate system. In edge_type, *v* denotes convex and *c* concave, as there are two convex and one concave edges meeting at this vertex. This LFS subsumes at least three minimal feature groupings that are also capable of generating a unique value for the pose transform. For example,

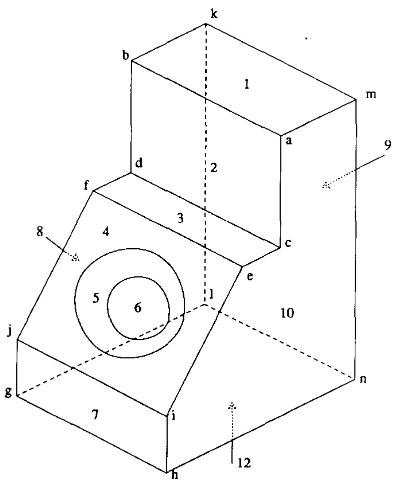


Fig. 13. Labels of surfaces and vertices of object in Fig. 2(a).

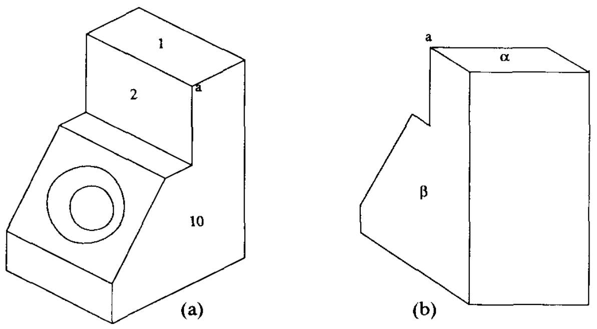


Fig. 14. Completely visible vertex of (a) object in one view, such as vertex labeled on left-hand side, becomes only partially visible in (b) another view.

the grouping consisting of the surfaces 2 and 10, together with the coordinates of the vertex a, can yield a unique value for Tr. To answer the question why we use this particular construction for LFS's, we will first define a completely visible vertex.

In the scene, a vertex will be called *completely visible* if no occluding edges meet at the vertex. An example of a completely visible vertex is shown in Fig. 14(a), while (b) shows the same vertex when it is not completely visible. Note that occluding edges in a range map are characterized by range discontinuities.

We believe that a completely visible vertex in a scene provides the strongest constraints for calculating the Tr associated with an object in a scene. Of course, theoretically, any two of the non-parallel surfaces coming together at a vertex, in conjunction with the vertex itself, are capable of specifying uniquely the Tr associated with a scene object. Therefore, theoretically at least, for the vertex labeled in Fig. 14(a), any two of the surfaces, together with the coordinates of the vertex, can yield Tr. However, in practice, it is difficult to calculate with great precision the position of the vertex itself, primarily because of the nature of discontinuities of some of the spatial derivatives at such a point. Therefore, our approach is that if a completely visible vertex can be found in a scene, it should immediately be used to calculate a Tr.

Of course, it is entirely likely that we may not find any completely visible vertices in a scene, meaning that all the visible vertices may be like the one labeled in Fig. 14(b) for which we are able to see only two of the three surfaces due to self-occlusion. In such a case, the LFS for a model vertex can still be used by assigning appropriate labels to the scene surface. In general, if h surfaces meet at a vertex in the model and only k of these are visible in the scene, then there are only h possibilities for matching the k scene surfaces, this happens because of the rotational adjacencies that have to be maintained. For example, as illustrated in Fig. 14, the vertex a is formed by three surfaces 1, 10 and 2; if we see only two of the surfaces α and β as in (b), there are only three different labeling patterns for the two scene surfaces, namely,

$$\{(1 \rightarrow \alpha, 10 \rightarrow \beta, 2 \rightarrow \text{nil}), (10 \rightarrow \alpha, 2 \rightarrow \beta, 1 \rightarrow \text{nil}), (2 \rightarrow \alpha, 1 \rightarrow \beta, 10 \rightarrow \text{nil})\}.$$

In each of these patterns, the labels must maintain the same adjacencies that are in the model. Therefore, we can say that in matching k scene features meeting at a scene vertex with the h features of an LFS, the overhead is k, which is incurred in matching the k scene features with the potential correspondents from the LFS. Since this can only be done in h ways, the overall complexity associated with matching with an LFS is $O(h \times k)$.

Therefore, the complexity associated with generating hypotheses for an object which has N_{LFS} LFS's is $N_{\text{LFS}} \times O(h \times k)$. In practice, $N_{\text{LFS}} = O(m)$, where m is the total number of model features. Therefore, the overall complexity associated with generating all the hypotheses is

$$O(m \times h \times k) = O(m).$$

Before we conclude this subsection, we would like the reader to note that the gains achieved with the use of LFS's as nonminimal feature groupings is at the cost of a more complex flow of control during hypothesis generation. While with minimal groupings, it is possible to institute uniform control, with nonminimal groupings special cases must be handled separately depending upon how many of the features in an LFS can be matched with the scene features.

Also, we have said nothing about the mathematics of how to actually compute a Tr given that we have a match between some scene features and model features. For lack of space, this mathematics can not be presented here. However, the interested reader might wish to go through what we believe is a very readable derivation in Appendix A of [12] which employs quaternions to demonstrate that an optimum Tr can be computed by solving an eigenvalue problem. This derivation is based on the principles first advanced by Faugeras and Hebert [17].

D. Feature Sphere for Verification

We want to organize model features of an object such that, given a candidate principal direction $\hat{\Phi}$ computed for a scene feature, all the model features with the principal direction $\hat{\Phi}$ can be accessed efficiently. Since a particular direction corresponds to a unique point on the surface of a unit sphere, in a manner similar to the organization of model vertices on a circle in the previously discussed 2-D example, a natural way is to record the model features on a unit sphere as a function of their principal directions. We shall call such a sphere a feature sphere. There can, of course, be multiple number of features corresponding to a given point on the feature sphere, especially if more than one feature class is used for describing models. In our experience, programming becomes more efficient if a separate feature sphere is used for each class, meaning that we represent all the primitive surface features on one sphere, all the primitive edge features on another sphere, and all the primitive point features on yet another. Fig. 15 shows the vertex feature sphere and the surface feature sphere for the 3-D model object in Fig. 8.

After a pose transform hypothesis Tr is generated, we want to verify or reject the hypothesis by matching the rest of the scene features to the rest of the model features under Tr. Of the different scene features which will be used for verification, consider a scene feature S. According to (4a) and (4b), a model feature that is a candidate for matching with the scene feature S should be characterized by a principal direction $\hat{\Phi}$ that is equal to the following for the different types of S.

- If S is a primitive surface (spherical surface excluded) or a primitive curve:

$$\hat{\Phi} = \boldsymbol{R}^{-1} * \boldsymbol{v}(S), \tag{6a}$$

where $\boldsymbol{R}$ is the rotation component of Tr, and $\boldsymbol{v}(S)$ is the orientation direction of feature S, defined similarly as its principal direction but with respect to the world coordinate system.
- If S is point feature or a spherical surface:
Let $\boldsymbol{p}(S)$ be the position vector of feature S with respect to a world coordinate system.

$$\hat{\boldsymbol{p}} = Tr^{-1} * \boldsymbol{p}(S) = \boldsymbol{R}^{-1} * (\boldsymbol{p}(S) - \boldsymbol{t}); \tag{6b}$$

$$\hat{\Phi} = \frac{\hat{\boldsymbol{p}}}{|\hat{\boldsymbol{p}}|}.$$

where $\boldsymbol{t}$ is the translation component of Tr.

As previously mentioned, principal directions provide a very strong constraint for the selection of candidate model features, i.e., each candidate principal direction computed from (6a) or (6b) will lead to a small number of candidate model features. This is especially true for point features as should have been apparent from the 2-D example discussed previously. For primitive surfaces or primitive edges, the discrimination provided by principal directions depends on the configuration of object surfaces. In general, we may assume that the principal directions of a model's features are randomly distributed over the unit sphere. Although, the probability of any two features occupying the same spot on the unit sphere will be very low, for the sake of argument we may assume that at most there will be k features for each principal direction, where $k \ll m$. Then the worst case time complexity for matching for verification will be $O(n \times k) = O(n)$. When combined with the complexity of hypothesis generation, as discussed in Section IV-C, this implies an overall complexity level of $O(mn)$. Since, $m = O(n)$, we can conclude that with our approach the overall complexity for single object recognition is $O(n^2)$.

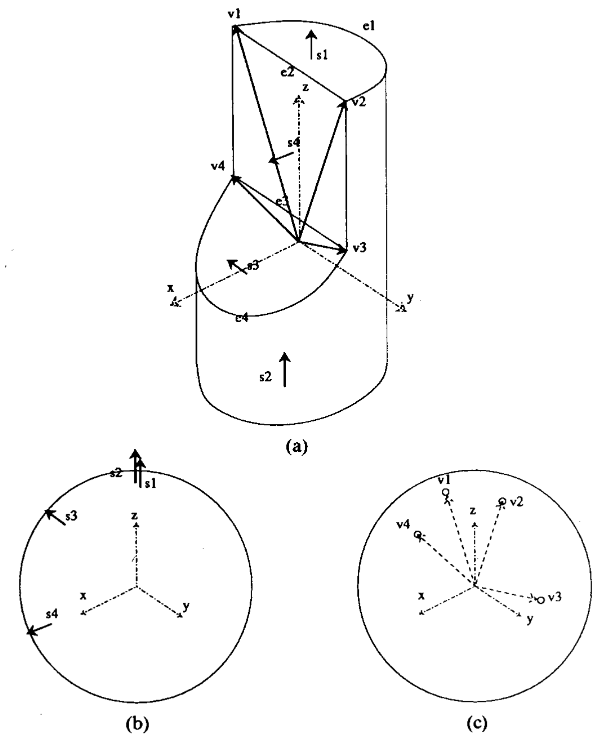


Fig. 15. (a) Model object. (b) Surface feature sphere. (c) Vertex feature sphere.

In the next section we will discuss in detail the feature sphere data structure implemented in a computer. It is interesting to note that if a model object is a convex polyhedron then its surface feature sphere representation is equivalent to its EGI (extended Gaussian image) [24], [26], and if a primitive curved surface is allowed to be added to a polyhedron then the surface feature sphere is similar to CSG-EESI representation proposed by Xie and Calvert [40]. In addition, if every surface point is regarded as a point feature, then the point feature sphere of a star-shape object is equivalent to the well-tessellated surface representation proposed by Brown [10].

V. A Data Structure for Representing Feature Spheres

In order to implement feature spheres in a computer, we first need to tessellate the sphere and then create an appropriate data structure for representing the tessellations. In our case, each cell on the sphere will be repre-

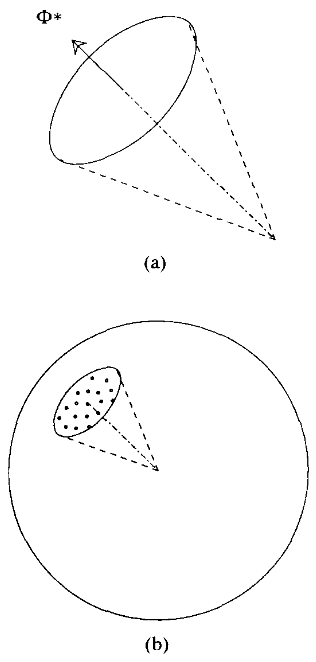


Fig. 16. (a) Directional uncertainty cone associated with principal direction Φ^*. (b) Sampling points of sphere that lie within cone.

sented by its center point, and the purpose of the data structure will be to allow us to efficiently access these points. In what follows, we will use the term *tessel* to refer to both a cell created by tessellating a sphere and to the central point of the cell. Before a data structure can be created for representing the tessels, we must bear in mind the following two kinds of operations that will be performed on the data structure for the purpose of feature matching.

First, during the model building process model features must be assigned to their respective tessels on the basis of their principal directions. Clearly, it is unlikely that the direction corresponding to one of the tessels would correspond exactly to that of a feature. For a given model feature, we must, therefore, locate the nearest tessel. In other words, we need a tessel assignment function, which will be denoted by $\boldsymbol{L}(\boldsymbol{\Phi})$, that should return the label of a tessel to which a model feature with principal direction $\boldsymbol{\Phi}$ is assigned.

Second, given a scene feature S during the verification stage, we want to examine whether there is a corresponding model feature with direction $\boldsymbol{\Phi}^* = Tr^{-1}(\boldsymbol{\Phi}(S))$ in the model under consideration. Assuming the hypothesis is correct, ideally, we should be able to find such a model feature at $\boldsymbol{L}(\boldsymbol{\Phi}^*)$ on the feature sphere of the model. However, due to noise and other artifacts associated with the estimation of Tr, $\boldsymbol{\Phi}^*$ will only be accurate to within some uncertainty interval. This directional uncertainty associated with $\boldsymbol{\Phi}^*$ can be expressed as a cone whose axis is the computed direction itself, as shown in Fig. 16. This implies that potential model features for matching with S should be all those that are within this cone. If we could assume the error processes associated with the uncertainties in $\boldsymbol{\Phi}^*$ to be of zero-mean type, from within the cone one would first select that feature that is closest to $\boldsymbol{L}(\boldsymbol{\Phi}^*)$, and, if that match were to fail, select the next closest, etc. Clearly, this is a breadth first search rooted at $\boldsymbol{L}(\boldsymbol{\Phi}^*)$, and the depth of search (the farthest neighbors to examine) should correspond to the maximal allowable direction uncertainty.

It should be obvious that for implementing the above strategy for the selection of model features, we need a function that would be capable of directly accessing the immediate neighbors of a given tessel; we consider two tessels to be neighbors if they share a common edge in the tessellation. This function will be called *find-neighbors* function and will be denoted by $\boldsymbol{N}$. So, we want

$$N(L_0) = \{L_1, L_2, \cdots, L_k\}$$

where $L_1, L_2, \cdots, L_k$ are the labels of the immediate neighbors of the tessel labeled L_0.

A. Previous Approaches To Data Structuring of Sphere Tessellations

In their work on EGI representation, Horn [24] and Ikeuchi [26] have discussed a hierarchical tree structure for representing a tessellated sphere based on icosahedron or dodecahedron. A drawback of this hierarchical data structure is that the adjacency relationship between neighboring tessels is not preserved. To get around this difficulty, Fekete and Davis [18] used a fairly complex labeling scheme, in this scheme each tessel is labeled by the pathname of its corresponding node in the tree. The neighbors of a tessel within one of twenty main icosahedral triangles are found by examining the pathname of the tessel, symbol by symbol, and synthesizing the pathnames of its neighboring tessels by the use of complicated state-transition rules and lookup tables. This procedure requires at least $O(n)$ operations, where n is the number of levels in the hierarchy. When the neighbors lie in an adjacent triangle, a different procedure is needed. Korn and Dyer [29] have also proposed a data structure for a tessellated sphere with a fixed number of subdivision levels. Twenty 1-D arrays, each of size 4^n, are used to represent the sampling points on the sphere, which implies that a sampling point is labeled by a number from 0 to $4^n - 1$. Their find-neighbors algorithm is essentially the same as that of Fekete and Davis. In another approach that allows for a quick determination of adjacencies in a spherical tessellation is by Goad [19] who projects a cube onto a sphere. A disadvantage of the Goad method is the unevenness of the resulting tessellations.

In this section we will present a new data structure for representing a tessellated sphere based on the icosahedron. Its main merit is that logical adjacency between elements of the data structure corresponds to physical adjacency between sampling points on the sphere. We will show that the neighbors of a given tessel can be found with a constant time complexity algorithm, regardless of the sampling resolution. Furthermore, via the find-neighbors function, the tessel-assignment function L can be implemented efficiently, too.

B. Tessellating a Unit Sphere

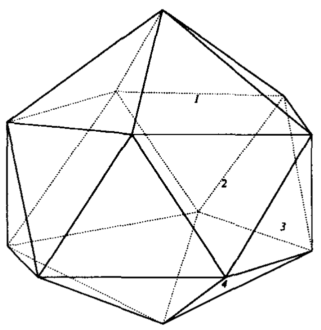
In this subsection, we will present the tessellations on which our data structuring is based. Subsequently, it should become evident to the reader that the regularity of the neighborhood patterns in the tessellations used allows us to devise a simple scheme for neighbor finding. However, first we will quickly review the considerations that go into the design of tessellations.

When a sphere is tessellated into cells, ideally we would like the cells to be symmetrical, be identical in shape, and possess equal areas; also, ideally, the tessellation scheme should maintain these attributes over a wide range of cell resolutions. However, it is well known that a tessellation scheme with these attributes does not exist. The best one can do is to use the techniques of geodesic dome constructions [28], [33]; these techniques lead to triangular cells that are approximately equal in area and shape. The geodesic tessellations are obtained via the following three steps.

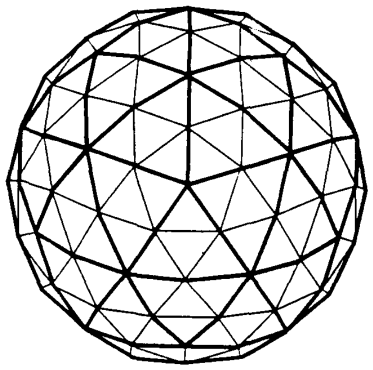
1) Choose a regular polyhedron, which usually is an icosahedron or a dodecahedron, and inscribe it in a sphere to be tessellated. If a dodecahedron is used, each of its pentagonal faces is divided into five triangular faces around its center to form a pentakis dodecahedron. Thus each face of the regular polyhedron will be a triangle.
2) Subdivide each triangular face of either the icosahedron or the pentakis dodecahedron into subfaces by dividing each edge of a triangular face into Q sections, where Q is called the frequency of geodesic division. Using these sections, each triangular face is divided into Q^2 triangular subfaces. Finer resolution can be obtained simply by increasing the frequency of geodesic division. Usually, Q is a power of two.
3) Project the subdivided faces onto the sphere. In order to make the projected triangle sizes more uniform, the edges of the triangles of, say, the icosahedron should be divided into sections such that each section subtends the same angle at the center of the sphere; as a consequence the lengths will be the same for the edge sections after they are projected onto the sphere [28].

To generate the tessellations used in 3D-POLY, we start out by implementing the above approach with an icosahedron. The geodesic polyhedron thus produced contains $20 \times Q^2$ cells and $10 \times Q^2 + 2$ vertices. Fig. 17 shows an icosahedron and a geodesic polyhedron for $Q = 4$.

Our next step, for the purpose of delineating cells that would contain pointers to feature frames on the basis of their principal directions, is to construct a dual of the



(a)



(b)

Fig. 17. (a) Icosahedron, and (b) frequency four geodesic polyhedron derived from icosahedron.

geodesic polyhedron produced by the above method. The dual of a polyhedron is defined as a polyhedron obtained by connecting the centers of the adjacent cells of the original polyhedron, as shown in Fig. 18. For example, a dodecahedron is the dual of an icosahedron. The dual of a frequency Q geodesic polyhedron derived from an icosahedron consists of $10 \times Q^2 + 2$ cells, of which $10\ (Q^2 - 1)$ are hexagonal and the remaining twelve pentagonal. The twelve pentagonal cells of the dual polyhedron correspond to the twelve vertices of the original icosahedron. This dual polyhedron is then projected onto the unit sphere to produce the desired tessellations. It is important to note that the cells of the dual polyhedron are used, in a sense, as bins for principal directions and that the centers of these cellscorrespond to the vertices of the geodesic polyhedron derived originally from the icosahedron.

As illustrated by the dashed lines in Fig. 18, our tessels can be either pentagonal or hexagonal, the former has five neighbors, and the latter six. The average area of a tessel is given by $4\pi/(10 \times Q^2 + 2)$. The radial angle between adjacent sampling points, which is an indication of sampling resolution, can be roughly estimated by

$$\operatorname{atan}(2)/Q$$

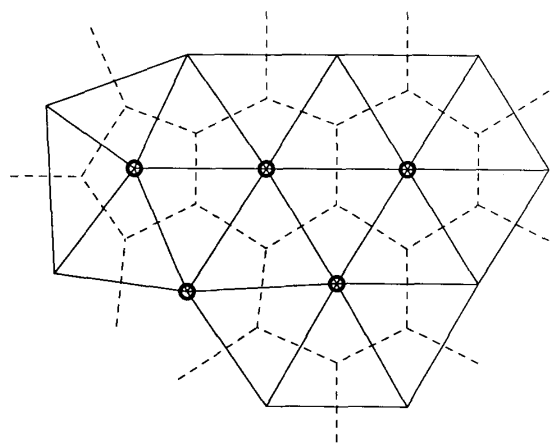


Fig. 18. Dashed lines define cells on feature sphere; these cells constitute a tesselation that is generated by dual polyhedron. Centers of cells, called sampling, correspond to vertices of geodesic tesselations, shown by solid lines, derived from icosahedron.

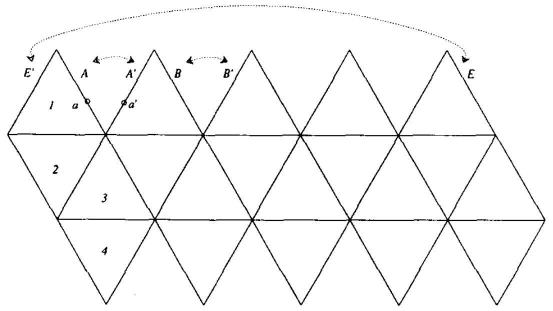


Fig. 19. Original icosahedron is flattened out to form five connected parallelograms, each of them consisting of four triangular faces.

where atan(2) is the angular spread of an icosahedron's edge.

C. A Spherical Array for Representing the Tessellation

We will now present a spherical array data structure for the computer representation of the tessellation. This array will lead to easy and efficient implementations of the find-neighbors function N and tessel-assignment function L. The data structure will be constructed by noting that the vertices of the underlying geodesic polyhedron, shown in Fig. 17 for the case of $Q = 4$, are the sampling points of the dual polyhedron whose cells are the "bins" for principal directions. As was mentioned before, in our case the underlying geodesic polyhedron is built from an icosahedron.

We note that when an icosahedron is flattened out, we obtain five connected parallelograms, each parallelogram consisting of four triangular faces (Fig. 19). We have indexed four of the triangular faces in the icosahedron shown in Fig. 17(a) and their correspondents in the flattened out version shown in Fig. 19 to help the reader visualize the process of flattening out.

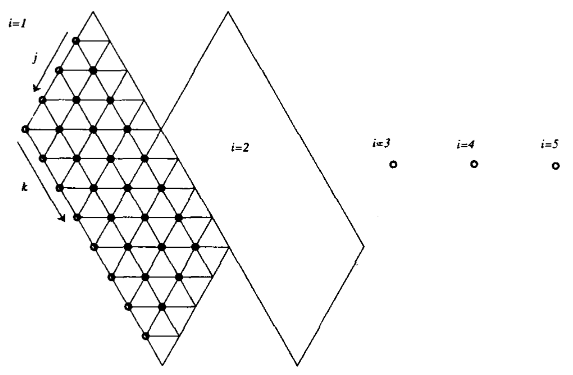


Fig. 20. Flattened-out spherical array representation using indices i, j, and k.

The flat representation of a frequency Q geodesic polyhedron may now be constructed by subdividing each parallelogram of Fig. 19 into $4 \times Q^2$ triangular cells. The vertices thus obtained, as shown in Fig. 20 for the case of $Q = 4$, correspond to the vertices of the geodesic polyhedron. In other words, the vertices shown in Fig. 17b correspond to the vertices obtained in the flat representation if we were to divide all the parallelograms in a manner similar to what was used for the leftmost parallelogram in Fig. 20. Therefore, the sampling points shown in Fig. 20 correspond to the centers of the cells used for discretizing the space of principal directions. The flattened-out representation, of which Fig. 20 is an example, will be referred to as the *spherical array*.

Each parallelogram in a spherical array consists of $(Q + 1) \times (2Q + 1)$ vertices. Obviously, the vertices in each parallelogram separately could be represented by a two dimensional array; however, note that the vertices on the borders of the parallelograms are shared, meaning, for example, that the vertices a and a' on the edges A and A', respectively, are really the same vertex on the geodesic polyhedron. In other words, before the icosahedron is unfolded to form the spherical array, edge A is connected to edge A', edge B to edge B', edge E to edge E', and so on (Fig. 19).

The fact that each border vertex should appear only once in an overall indexing scheme for the vertices in a spherical array implies that the size of the index array for representing each parallelogram need only be $Q \times 2Q$. For example, for the case shown in Fig. 20, each parallelogram need only be represented by a 4×8 array. The assignment of array indices for the parallelograms is depicted in Fig. 20 for the $Q = 4$ case. The index i specifies a parallelogram and the indices j and k specify a vertex within the parallelogram. Clearly, we have five $Q \times 2Q$ arrays, for a total of $10 \times Q^2$ indexed points on the spherical array, this number being two less than the total $10 \times Q^2 + 2$ vertices on the geodesic polyhedron. The two missing vertices correspond to the two common vertices of the five parallelograms, one at the top and the other at the bottom. We shall allocate two additional distinguished sets of indices to represent these two vertices and refer to them as the zenith and the nadir (see Section V-C-2) for explanation) of the tessellated sphere.

The proposed indexing implies the following ranges for i, j and k:

$$[i,j,k] \quad 1 \leqslant i \leqslant 5, \; 1 \leqslant j \leqslant Q, \; 1 \leqslant k \leqslant 2Q.$$

The zenith and the nadir are assigned the distinguished indices $[0,0,0]$ and $[-1,0,0]$, respectively.

1) The Find-Neighbors Function: As pointed out before, the simplicity of the proposed data structure lies in its preserving the physical adjacencies between the tessels. We will now show that simple relationships exist that yield a tessel's neighbors, regardless of the location of the tessel, and, more important, regardless of whether the tessel possesses six or five neighbors. Most tessels possess six neighbors, except for the twelve that correspond to the twelve vertices of the original icosahedron, each of latter type possessing five neighbors only. In general, the six neighbors of a tessel $[i, j, k]$ that is not on the border of any of the five parallelograms are given by:

$$\left\{\begin{array}{l}[i,j,k+1], \\ [i,j+1,k], \\ [i,j+1,k-1], \\ [i,j,k-1], \\ [i,j-1,k], \\ [i,j-1,k+1].\end{array}\right. \tag{7}$$

Therefore, for the set of indices in (7) to give us the neighbors, the indices j and k must obey the constraints $1 < j < Q$ and $1 < k < 2Q$. If also used to find the neighbors of a border tessel, some of the above indices would take out-of-range values, implying that those neighboring tessels are vertices shared by another parallelogram and should really be assigned to the array for that parallelogram. To convert the out-of-range labels to the legitimate ones, we apply the following substitution rules:

$$\left.\begin{array}{ll}[i,j,0] \Rightarrow [i-1^{m5},1,j] & j=1,\cdots,Q \\ [i,Q+1,k] \Rightarrow [i-1^{m5},1,Q+k] & k=0,\cdots,Q \\ [i,Q+1,k] \Rightarrow [i-1^{m5},k-Q,2Q] & k=Q+1,\cdots,2Q \\ [i,0,k] \Rightarrow [i+1^{m5},k-1,1] & k=2,\cdots,Q \\ [i,0,k] \Rightarrow [i+1^{m5},Q,k-Q] & k=Q+1,\cdots,2Q+1 \\ [i,j,2Q+1] \Rightarrow [i+1^{m5},Q,j+Q+1] & j=1,\cdots,Q-1 \\ [i,0,1] \Rightarrow [0,0,0] & \\ [i,Q,2Q+1] \Rightarrow [-1,0,0] & \end{array}\right\} \tag{8}$$

for $i=1,\cdots,5$, where $i^{m5}=(i-1)\bmod(5)+1$.

Except for the zenith and the nadir tessels, it can be verified that (7) and (8) are also applicable to ten of the twelve five-neighbor tessels. At a five-neighbor tessel, two of the six labels returned by (7) will turn out to be identical after applying the substitution rules in (8). The five neighbors for the zenith and the nadir are

$$[i,1,1] \quad i=1,\cdots,5$$

and

$$[i,Q,2Q] \quad i=1,\cdots,5$$

respectively.

The following two examples will illustrate the neighbor finding scheme. The examples are for the case of $Q=4$.

Example 1:

Find the neighbors of tessel [1,3,1]

$$[1,3,1] \rightarrow \left\{\begin{array}{l}[1,3,2] \\ [1,4,1] \\ [1,4,0]=[5,1,4] \\ [1,3,0]=[5,1,3] \\ [1,2,1] \\ [1,2,2].\end{array}\right. \rightarrow \left\{\begin{array}{l}[1,3,2] \\ [1,4,1] \\ [5,1,4] \\ [5,1,3] \\ [1,2,1] \\ [1,2,2]\end{array}\right. .$$

Example 2:

Find the neighbors of tessel [2,4,5]

$$[2,4,5] \rightarrow \left\{\begin{array}{l}[2,4,6] \\ [2,5,5]=[1,1,8] \\ [2,5,4]=[1,1,8] \\ [2,4,4] \\ [2,3,5] \\ [2,3,6]\end{array}\right. \rightarrow \left\{\begin{array}{l}[2,4,6] \\ [1,1,8] \\ [2,4,4] \\ [2,3,5] \\ [2,3,6]\end{array}\right. .$$

It is worth noting that [2,4,5] happens to be a vertex of the original icosahedron and has only five neighboring tessels, exactly what the rules returned.

2) Directions of Sampling Points: In order to specify the tessel-assignment function, we will need formulas for the directions of the tessels, meaning the directions associated with each of the vertices on the spherical array. For that purpose, we will take advantage of the symmetry of the icosahedron and use a sphere-centered coordinate system whose positive z axis passes the zenith at ([0,0,0]) and

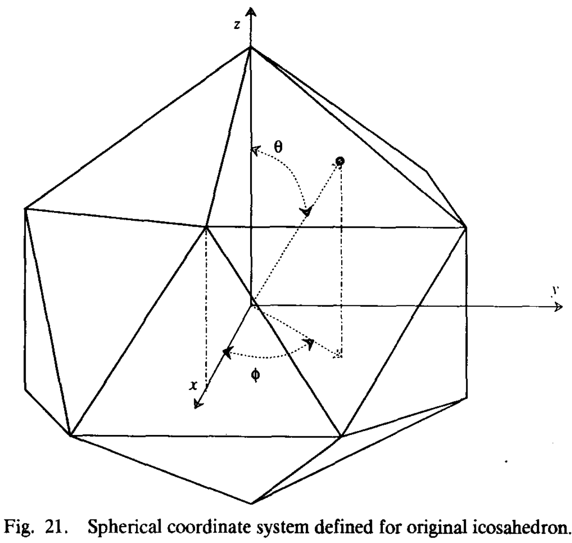


Fig. 21. Spherical coordinate system defined for original icosahedron.

whose z-x plane passes the icosahedral vertex $[1, Q, 1]$, as shown in Fig. 21. The direction of each tessel, denoted by $\Phi[i, j, k]$, in this coordinate system will be expressed in terms of the longitude and latitude angles (ϕ, θ). Because of the symmetry of the icosahedron, we have

$$\begin{aligned} \theta[i,j,k] &= \theta[i-1,j,k] \\ \phi[i,j,k] &= \left(\frac{2\pi}{5} + \phi[i-1,j,k]\right) \bmod (2\pi) \end{aligned} \tag{9}$$

for $i = 2, \cdots, 5, \quad j = 1, \cdots, Q, \quad k = 1, \cdots, 2Q$. Therefore we only need to compute the direction of the tessels in the first parallelogram (array).

It is easy to see that the direction of the five vertices of the first parallelogram are:

$$\Phi[1,0,1]^\dagger = (0, -)$$

$$\Phi[1,Q,1] = (\operatorname{atan}(2), 0)$$

$$\Phi[1,0,Q+1]^\dagger = \left(\operatorname{atan}(2), \frac{2\pi}{5}\right)$$

$$\Phi[1,Q,Q+1] = \left(\pi - \operatorname{atan}(2), \frac{\pi}{5}\right)$$

$$\Phi[1,0,2Q+1]^\dagger = \left(\pi - \operatorname{atan}(2), \frac{3\pi}{5}\right)$$

$$\Phi[1,Q,2Q+1]^\dagger = (\pi, -).$$

Recall that in the derivation of the geodesic polyhedron, we subdivided each edge of the triangles of the inscribed icosahedron into Q sections of equal radial angle. When $Q = 2^r$, the result is equivalent to recursively subdividing r times a triangle into four triangles. Therefore, we can compute the direction of a new tessel by taking the averages of the known directions of the two tessels that are the end-points of the edge whose division led to the formation of the new tessel. This procedure can be applied recursively to compute the direction of every tessel. As an example, the three tessels that are the midpoints of the three edges of the upper triangle have directions:

$$\Phi\left[1, \frac{Q}{2}, 1\right] = \operatorname{mid}(\Phi[1,0,1], \Phi[1,Q,1])$$

$$\Phi\left[1, \frac{Q}{2}, \frac{Q}{2}+1\right] = \operatorname{mid}(\Phi[1,0,Q+1], \Phi[1,Q,1])$$

$$\Phi\left[1, 0, \frac{Q}{2}+1\right] = \operatorname{mid}(\Phi[1,0,Q+1], \Phi[1,0,1]).$$

Here $\operatorname{mid}(\Phi_1, \Phi_2)$ means the average of the two directions Φ_1 and Φ_z, on the unit sphere. To save runtime computation, we may precompute the directions for all the tessels and store them in a lookup table.

3) The Tessel-Assignment Function: Given a particular direction Φ, its corresponding tessel in the spherical array should be the one whose direction is closest to Φ. In other words, for a given Φ, we want that index triplet L that satisfies

$$\max_L \, (\Phi \cdot \Phi[L]).$$

The finding of the tessel L would thus involve a search for the maximal dot product. Because the dot product is a monotonically increasing function toward the desired tessel, a local maximum must also be the global maximum. The local maximum can be found by an iterative climbing method from any tessel guessed initially. Since a good initial guess can reduce considerably the computations required to reach the maximum, we have provided in the Appendix a linear approximation that translates a given Φ into a triple (i, j, k). This approximation has proved to be fairly good, and the resulting indices are quite close to their actual values. Starting with these indices, one can then find the actual ones by conducting local search, as depicted by the following algorithm.

```
assign_tessel($\Phi$){
    $L^0$ = get_initial_guess($\Phi$)
    $L$ = get_closer($L^0$, $\Phi$)
    return $L$}
get_closer($L$, $\Phi$){
    among all $L'$ in $N(L)$
        find $\hat{L}$ that maximizes $(\Phi(L') \cdot \Phi)$
    if $(\Phi(\hat{L}) \cdot \Phi^* > \Phi(L) \cdot \Phi)$
        get_closer($\hat{L}$, $\Phi$)
    else return $\hat{L}$}
```

4) Building Feature Spheres on the Spherical Array: Note that a feature is described by a list of attribute-value pairs; this list is referred to as a feature frame, an example of which was shown in Section III-A. Each such frame structure is identified by a pointer that is stored at the corresponding tessel in the spherical array. The tessel address, as represented by the indices i, j, and k, is computed by

†Note that these labels are not legitimate in the spherical array data structure; we use them just to make the derivations clearer. The legitimate versions of these labels can be obtained by using the substitution formulas in (8).

applying the tessel-assignment-function to the principal direction of the feature. It may happen that two or more neighboring features, neighboring in the sense of their possessing nearly identical principal directions, may have their pointers assigned to the same tessel. This conflict can be resolved by recording in the first registered feature a list of pointers for the features that share the same tessel address.

VI. Recognition of Objects in the Presence of Occlusions

The discussion presented so far could be used directly for the recognition of single isolated objects. However, our main interest in 3D-POLY lies in recognizing objects under occluded conditions, as would be the case when objects are presented to the sensory system in the form of heaps.

In general, when the range images to be interpreted are of scenes containing piles of overlapping objects, one has to contend with the following two problems: 1) The number of features extracted from a scene will usually be very large; and, 2) since different objects may be made of similar features, it would generally not be possible to set up simple associations between the scene features and the objects. To get around these problems in dealing with multiple object scenes, researchers previously have either performed object segmentation by exploiting range discontinuity information [15], or have used a *model driven* (to be contrasted with the *data driven* procedure to be described in this section) approach to group together scene features belonging to single objects [6], [16]. However, the former approach usually fails to work especially when the juxtapositions of multiple objects are such that there are no range discontinuities between them; and the latter is inefficient for reasons described in Section II.

We will now present a data-driven approach for aggregating from a complex scene those features that belong to single objects. The cornerstone of our approach is the idea that only physically adjacent scene features need be invoked for matching with a candidate model object. For this purpose, the notion of physical adjacency will be applied in the image space as opposed to the object space, implying, for example, that two surface regions sharing a common boundary, even if it is a jump boundary, will be considered adjacent to each other. Using this idea, we will now describe the complete method:

The algorithm uses two sets, UMSFS and MSFS, the former standing for the unmatched scene feature set and the latter for matched scene feature set. Initially, the algorithm assigns *all* the scene features to the set UMSFS. The process of object recognition starts with a local feature set (LFS) extracted from the UMSFS. The matching of this scene LFS with a model LFS generates a hypothesis about object identity and a pose transformation. The features in the scene LFS are then taken off from the UMSFS and assigned to MSFS; note that MSFS keeps a record of all the scene features matched so far with the current candidate model. Then during the verification stage, only those scene features in the UMSFS that are adjacent to the features in MSFS are selected for matching with the candidate model. During the verification state, if a UMSFS feature does match the candidate model feature, the scene feature is taken out of the UMSFS and added to the MSFS; otherwise the feature is marked as tested under the current hypothesis and left in the UMSFS.

The verification stage terminates when MSFS stop growing. Once the verification process terminates, the algorithm determines whether or not the features in the MFS constitute enough evidence to support the hypothesis on the basis of some predefined criterion. This criterion may be as simple as requiring a percentage, say, 30 percent, of model features to be seen in the MSFS; or, as complicated as requiring a particular set of model features to appear in the MSFS; or, at a still more complex level, some combination of the two. If a hypothesis is considered verified, the features currently in MSFS are labeled by the name of the model and taken out of further consideration; otherwise, the hypothesis is rejected and every feature in the MSFS is put back into the UMSFS and the process continued with a new LFS. The entire process terminates after all the LFS's have been examined. The algorithm is presented as follows in pseudo *C* language:

```
Interprete_scene ($I$){
    extract feature set $\{S\}$ from $I$
    UMSFS = $\{S\}$
    while (there exists a local feature set $LFS_s$ in
        UMSFS)
      for each $LFS_m$ in the model library
        if ($LFS_s$ matches $LFS_m$){
          estimate $Tr$ by matching LFS$_s$ with
          LFS$_m$
          candidate model $O_M$ is the model
          corresponding to LFS$_m$
          MSFS = LFS$_s$
          mark every $M_j$ in LFS matched
          Verify ($O_M$, MSFS, UMSFS, $Tr$)}}
Verify ($O_M$, MSFS, UMSFS, $Tr$){
tag for each untested $S_i$ in UMSFS adjacent to MSFS{
      compute principal direction $\Phi$ of $Tr^{-1}(S_i)$
      for each unmatched $M_j$ registered in the neigh-
      borhood of $L(\Phi)$ on the feature sphere of $O_M$
        if ($Tr^{-1}(S_i)$ matches $M_j$){
          add $S_i$ to MSFS
          mark $M_j$ matched
          go to tag}
        else
          mark $S_i$ tested
if (MSFS satisfies the recognition criterion){
      UMSFS = UMSFS − MSFS
      label every $S$ in MSFS by the name $O_M$
      write_result ($O_M$, $Tr$)
      unmark every $M_j$
      return (true)}
    else {unmark every $M_j$
```

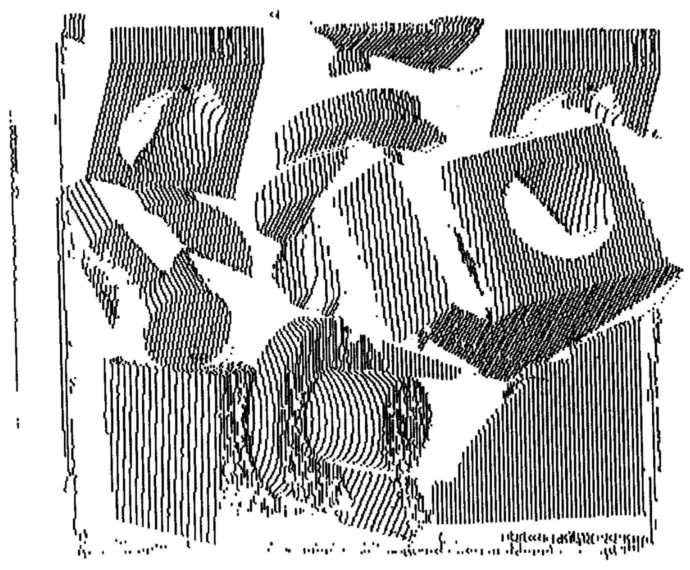


Fig. 22. Structured light image of scene #1 (pile 5).

In our current implementation of this algorithm, the recognition criterion requires that at least 33 percent of a candidate model's features be present in the MSFS for a hypothesis to be considered valid. Note that the acceptance threshold can be no greater than 50 percent for most objects, especially those that have features distributed all around, since from a single viewpoint only half of an object will be seen. Therefore, 50 percent is a loose upper limit on the acceptance threshold. On the lower side, the threshold can not be set to be too low, since that would cause misrecognition of objects. We have found 33 percent to be a good compromise.

VII. Experimental Results

This section presents experimental results obtained with our matching strategy; the results will also demonstrate in action the algorithm for recognizing objects in heaps. Although we have done experiments on a large number of scenes with 3D-POLY, only two such experiments will be presented to discuss the behavior of the algorithms.

A. The Models

The model library used consisted of two object models shown in Fig. 2. The object in Fig. 2(a) is given the name "square" and the one in 2(b) "round." The model knowledge was obtained by a "learning system" consisting of a special scanner in which the object is automatically rotated while illuminated by a number of translating laser beams. The data thus generated from many viewpoints is integrated and directly transformed into a feature sphere representation. Further details on the methods used for viewpoint integration and the transformations involved are presented in [12]. For the two experiments discussed here, model data was generated by integrating six views for "square" and five for "round." For "square" object this resulted in a feature representation consisting of 14 vertex and twelve surface features. The model representation derived for "round" object consisted of twelve vertices and ten surfaces.

Two feature spheres were derived for each model, one for surface features and the other for vertex features. The frequency of geodesic division, Q, of the spherical array discussed in Section V-C was chosen to be 16; this gave a resolution of about 4° per tessel in the spherical array representation. The vertex and surface features were used for the generation of hypotheses, while only the surface features were used for verification.

As described in Section IV-C, each object model must be associated with a list of LFS's for the purpose of hypothesis generation, an LFS being a set of surface features meeting at a vertex. In this prototype system, we have chosen to organize LFS's around convex vertices only, that is those whose edges are all convex. For "square" object, there are 12 LFS's that correspond to the twelve convex vertices, and for "round" object there are only four LFS's corresponding to the convex vertices.

B. The Data

For the results that will be shown here, we had 10 overlapping objects, five of each type, in each of the two scenes. The objects were placed in a tray and, before data collection, the tray shaken vigorously to randomize the object placements. A typical scene was as was shown earlier in Fig. 1. Range images of the two scenes, shown in Figs. 22 and 23, were acquired by using a structured-light range sensing unit that is held by a PUMA robot for dynamic scanning; these images will be referred to as stripe images. Each stripe image consists of 150 stripes, with the inter-stripe spacing being 0.1″; this spacing is the

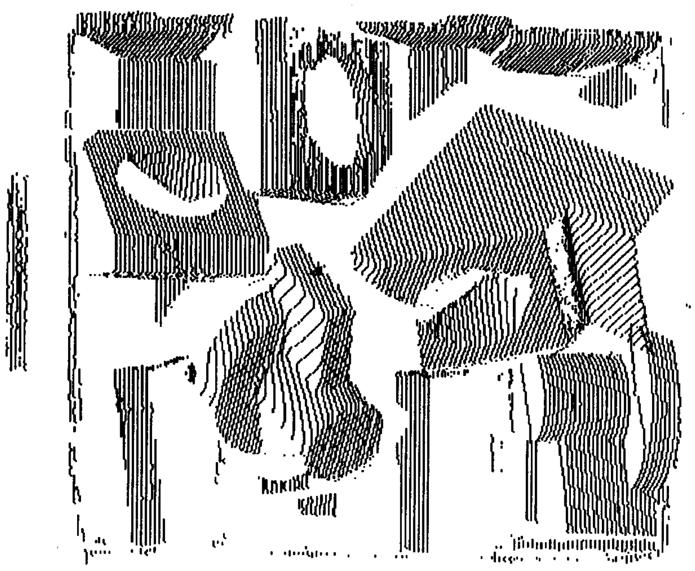


Fig. 23. Structured light image of scene #2 (pile 4).

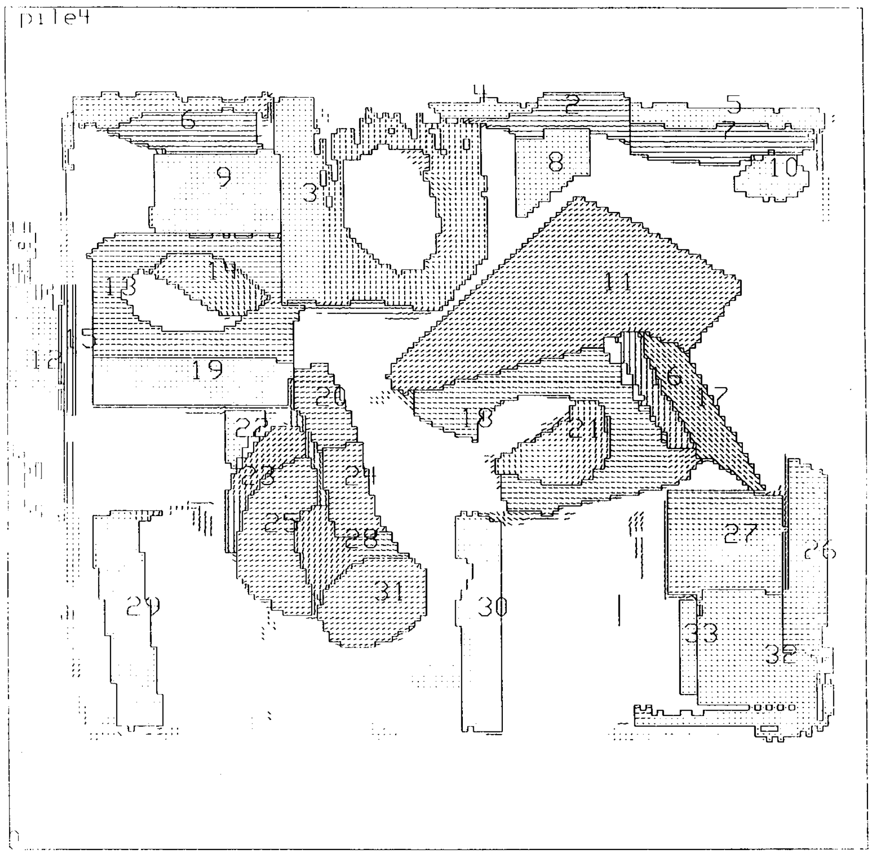


Fig. 24. Result on feature extraction from stripe image of scene #2.

distance the robot end-effector travels between successive projections.

Range maps for the scenes are obtained by converting each stripe point, which exists in image coordinates, into world coordinates using a calibration matrix by the method discussed in [13]. Features are extracted from the range maps by a battery of low level procedures developed specifically for this research project. These procedures carry out surface normal computations, segmentations of surfaces of different types, surface classifications, etc., and are discussed in greater detail in [12]. The output of preprocessing for the range map corresponding to the stripe image of Fig. 22 is shown in Fig. 5 in the form of a needle diagram and segmented surfaces. Fig. 24 shows the results for the stripe image of Fig. 23. Figs. 5 and 24 also display the labels given to the different surfaces.

C. Hypothesis Generation

For the purpose of hypothesis generation, each detected vertex in a scene is given a rank depending upon the number of surfaces meeting at the vertex and the convexity/concavity of the edges convergent at the vertex. The rank is greater, the larger the number of surfaces meeting at a vertex. Also, since we only use convex vertices for constructing the LFS's of a model, if a concave edge is found to be incident at a vertex, the rank of the vertex is made negative.

To generate hypotheses, the system first chooses the highest positively ranked vertex and then constructs an LFS by collecting all the surfaces meeting at the vertex. The scene LFS thus generated is matched with the LFS's of all the models, one by one. This matching between a scene LFS and a model LFS is carried out by a special procedure that tests the compatibility of the shape and relation attributes of the corresponding features in the two LSF's. Note that the maximal number of surfaces in an LFS for the objects in the experiments reported here is 3, thus there are three possible ways of establishing the correspondences between a scene LFS and a model LFS; all the three possibilities must be tested, each accepted possibility will lead to a different pose hypothesis.

For a given match between a scene LFS and a model LFS, the viewpoint independent position/orientation attributes of the features in the two LFS's are used for generating a candidate pose *Tr* for the scene object; further details on how exactly this is done can be found in [12]. For a candidate pose to translate into a pose hypothesis, the system checks the fitting error computed from the estimation of *Tr*; the error must be less than a predefined threshold.

In the preprocessed output shown in Fig. 5, there are 68 vertices, but only 36 of them are of convex type; in the output shown in Fig. 24 there are 22 convex vertices out of a total of 49 vertices. So, supposedly, in the worst case one would have to check 36 LFS's in the former case, and 22 in the latter. Since there are a total of 16 LFS's in the model library, 12 for "square" and 4 for "round", in the worst case one would have to carry out $16 \times 36 \times 3 = 1728$ LFS matches for the scene of Fig. 5, where the number 3 takes care of the aforementioned different ways of establishing correspondences between a model LFS and a scene LFS. Similarly, in the worst case situation, there may be $16 \times 22 \times 3 = 1056$ LFS matchings to be tested for the scene of Fig. 24. In practice, however, the number of LFS matches actually carried out is far fewer on account of the following reason: An object hypothesis can be generated by any one of many LFS's, and when a hypothesis thus generated is verified, the system does not need to invoke any of the other LFS's for that object.

To give the reader an idea of the number of hypotheses generated, the system generated 156 hypotheses for the scene of Fig. 5, and 75 for the scene of Fig. 24.

D. Verification

Given the pose transformation *Tr* associated with a hypothesis, verification is carried out by computing the feature sphere tessel indices of those scene features that are "physically adjacent" to the LFS features, the notion of physical adjacency being as explained before, and matching each such scene feature with a model feature assigned to that tessel, assuming such a model feature can be found. [If more than one model feature is assigned to a tessel, the scene feature must be matched with all of them.] Of course, since measurement noise and other artifacts will always be present to distort the attribute values of a scene feature, the scene feature must be matched with all the model features belonging to tessels within a certain neighborhood of the tessel computed from the scene feature principal direction. The size of the neighborhood reflects the uncertainty in the feature measurements. For most of our experiments, we use all the model features within two tessels of the tessel assigned to a scene feature, corresponding approximately to a directional uncertainty of 8.0°.

To illustrate the behavior of the algorithm, Table III shows the hypothesis generation and verification procedure in action. Each line entry, printed out upon the formation of a hypothesis, identifies the LFS used by the vertex chosen, and shows the surface correspondences established when the scene LFS was matched with a model LFS. For example, for the first hypothesis, marked hyp#1 in the table, the LFS matching established correspondences between scene surface 7 and model surface 2; and between scene surface 5 and model surface 1. (See Fig. 13 for a labeling of the surfaces of the model object "square.") The number 2 at the end of the line in the table indicates that the first hypothesis was failed during the verification stage after failures along two different paths in the search space, each path involving only one feature test that failed, each failure caused by a mismatch of a scene feature that was physically adjacent to one of the hypothesis generating LFS features, and the model feature located within the uncertainty range of the tessel corresponding to the scene feature. This is not to imply a fan-out of only 2 at the end of the hypothesis generating segment for hyp#1; only that

TABLE III
OUTPUT LISTING OF THE INTERPRETATION OF SCENE #2

```
Verify hyp #1 Model: square Vert: 12 Reg: (7→2)(5→1)...failed-2
Verify hyp #2 Model: square Vert: 12 Reg: (7→1)(5→2)...failed-2
Verify hyp #3 Model: square Vert: 12 Reg: (7→10)(5→4)...failed-2
Verify hyp #4 Model: square Vert: 12 Reg: (7→4)(5→8)...failed-1
Verify hyp #5 Model: square Vert: 12 Reg: (7→10)(5→12)...failed-1
Verify hyp #6 Model: square Vert: 12 Reg: (7→4)(5→10)...failed-1
Verify hyp #7 Model: square Vert: 12 Reg: (7→12)(5→9)...failed-1
Verify hyp #8 Model: square Vert: 12 Reg: (7→9)(5→10)...failed-1
Verify hyp #9 Model: square Vert: 12 Reg: (7→9)(5→12)...failed-1
Verify hyp #10 Model: square Vert: 12 Reg: (7→12)(5→10)...failed-1
Verify hyp #11 Model: square Vert: 12 Reg: (7→10)(5→9)...failed-1
Verify hyp #12 Model: round Vert: 18 Reg: (17→9)(11→7)...failed-0
Verify hyp #13 Model: round Vert: 18 Reg: (17→9)(11→1)...failed-0
Verify hyp #14 Model: round Vert: 18 Reg: (17→7)(11→9)...failed-0
Verify hyp #15 Model: round Vert: 18 Reg: (17→1)(11→6)...failed-0
Verify hyp #16 Model: round Vert: 18 Reg: (17→6)(11→7)...failed-0
Verify hyp #17 Model: round Vert: 18 Reg: (17→7)(11→6)...failed-0
Verify hyp #18 Model: square Vert: 18 Reg: (17→2)(11→1)...failed-0
Verify hyp #19 Model: square Vert: 18 Reg: (17→1)(11→10)
    scene region 18 matched to model region 4...failed-2
Verify hyp #20 Model: square Vert: 18 Reg: (17→10)(11→2)...failed-0
Verify hyp #21 Model: square Vert: 18 Reg: (17→8)(11→1)...failed-0
Verify hyp #22 Model: square Vert: 18 Reg: (17→1)(11→2)...failed-0
Verify hyp #23 Model: square Vert: 18 Reg: (17→2)(11→8)
    scene region 18 matched to model region 4
    scene region 21 matched to model region 5...SUCCEED!!!-2
Verify hyp #24 Model: square Vert: 26 Reg: (19→4)(13→3)...failed-0
Verify hyp #25 Model: square Vert: 26 Reg: (19→3)(13→4)...failed-0
Verify hyp #26 Model: square Vert: 26 Reg: (19→7)(13→4)...failed-0
Verify hyp #27 Model: square Vert: 27 Reg: (13→4)(19→3)...failed-0
Verify hyp #28 Model: square Vert: 27 Reg: (13→3)(19→4)...failed-0
Verify hyp #29 Model: square Vert: 27 Reg: (13→7)(19→4)...failed-0
Verify hyp #30 Model: round Vert: 36 Reg: (24→7)(20→10)...failed-0
Verify hyp #31 Model: round Vert: 36 Reg: (24→10)(20→9)...failed-2
Verify hyp #32 Model: round Vert: 36 Reg: (24→9)(20→7)...failed-0
Verify hyp #33 Model: round Vert: 36 Reg: (24→9)(20→1)...failed-0
Verify hyp #34 Model: round Vert: 36 Reg: (24→1)(20→7)...failed-2
Verify hyp #35 Model: round Vert: 36 Reg: (24→7)(20→9)...failed-0
Verify hyp #36 Model: round Vert: 36 Reg: (24→7)(20→1)...failed-0
Verify hyp #37 Model: round Vert: 36 Reg: (24→1)(20→6)...failed-2
Verify hyp #38 Model: round Vert: 36 Reg: (24→6)(20→7)...failed-0
Verify hyp #39 Model: round Vert: 36 Reg: (24→6)(20→10)...failed-0
Verify hyp #40 Model: round Vert: 36 Reg: (24→10)(20→7)
    scene region 28 matched to model region 4...failed-3
Verify hyp #41 Model: round Vert: 36 Reg: (24→7)(20→6)...failed-0
Verify hyp #42 Model: square Vert: 36 Reg: (24→2)(20→1)...failed-0
Verify hyp #43 Model: square Vert: 36 Reg: (24→1)(20→10)...failed-0
Verify hyp #44 Model: square Vert: 36 Reg: (24→10)(20→2)...failed-0
Verify hyp #45 Model: square Vert: 36 Reg: (24→8)(20→1)...failed-0
Verify hyp #46 Model: square Vert: 36 Reg: (24→1)(20→2)...failed-0
Verify hyp #47 Model: square Vert: 36 Reg: (24→2)(20→8)...failed-0
Verify hyp #48 Model: square Vert: 36 Reg: (24→3)(20→10)...failed-0
Verify hyp #49 Model: square Vert: 36 Reg: (24→10)(20→4)...failed-0
Verify hyp #50 Model: square Vert: 36 Reg: (24→4)(20→8)...failed-0
Verify hyp #51 Model: square Vert: 36 Reg: (24→8)(20→3)...failed-0
Verify hyp #52 Model: square Vert: 36 Reg: (24→12)(20→7)...failed-0
Verify hyp #53 Model: square Vert: 36 Reg: (24→7)(20→10)...failed-0
Verify hyp #54 Model: square Vert: 36 Reg: (24→10)(20→12)...failed-0
Verify hyp #55 Model: square Vert: 36 Reg: (24→4)(20→10)...failed-0
Verify hyp #56 Model: square Vert: 36 Reg: (24→10)(20→7)...failed-0
Verify hyp #57 Model: square Vert: 36 Reg: (24→9)(20→1)...failed-0
Verify hyp #58 Model: square Vert: 36 Reg: (24→1)(20→8)...failed-0
Verify hyp #59 Model: square Vert: 36 Reg: (24→8)(20→9)...failed-0
Verify hyp #60 Model: square Vert: 36 Reg: (24→12)(20→9)...failed-0
Verify hyp #61 Model: square Vert: 36 Reg: (24→9)(20→8)...failed-0
Verify hyp #62 Model: square Vert: 36 Reg: (24→8)(20→12)...failed-0
Verify hyp #63 Model: square Vert: 36 Reg: (24→10)(20→1)...failed-0
Verify hyp #64 Model: square Vert: 36 Reg: (24→1)(20→9)...failed-0
Verify hyp #65 Model: square Vert: 36 Reg: (24→9)(20→10)...failed-0
Verify hyp #66 Model: square Vert: 36 Reg: (24→9)(20→12)...failed-0
Verify hyp #67 Model: square Vert: 36 Reg: (24→12)(20→10)...failed-0
Verify hyp #68 Model: square Vert: 36 Reg: (24→10)(20→9)...failed-0
Verify hyp #69 Model: round Vert: 37 Reg: (20→7)(24→10)...failed-2
Verify hyp #70 Model: round Vert: 37 Reg: (20→10)(24→9)...failed-0
```

TABLE III (CONTINUED)

```
Verify hyp #71 Model: round Vert: 37 Reg: (20 → 9)(24 → 7)...failed-0
Verify hyp #72 Model: round Vert: 37 Reg: (20 → 9)(24 → 1)...failed-2
Verify hyp #73 Model: round Vert: 37 Reg: (20 → 1)(24 → 7)...failed-0
Verify hyp #74 Model: round Vert: 37 Reg: (20 → 7)(24 → 9)...failed-0
Verify hyp #75 Model: round Vert: 37 Reg: (20 → 7)(24 → 1)
    scene region 28 matched to model region 4
    scene region 31 matched to model region 2
    scene region 25 matched to model region 8...SUCCEED!!!-6
Total number of feature matching tests for verification: 37
Process completed
Recognized_objects:
  square
  round
```

for the other branches the scene features, again physically adjacent to one of the LFS features, had no corresponding model features on the feature sphere. This is also the reason for 0 at the end of many of the line entries in the table.

As mentioned in Section VI, the acceptance of a hypothesis is predicated upon our finding at least 33 percent of the model features from amongst those that are adjacent to the features in an LFS. As shown in the table, from among the 75 generated hypotheses only the hypotheses #23 and #75 are verified and lead to the recognition of an instance of "square" in the first case, and to that of "round" in the other. During the verification of hypothesis #23, scene surface 16 fails to match model surface 3 of the square model, although scene regions 18 and 21 do match model regions 4 and 5, respectively. As is evident from the stripe image of Fig. 23, the difficulties with scene surface 16 are due to problems with the robust detection of stripes over that surface; these problems are probably caused by the rather very acute angle between the stripe projection direction and the surface. It is entirely possible that the surface labeled 16 in the scene is made of reflections of the stripes seen in adjoining surfaces. In other words, surface 16 is most likely a spurious surface and not matchable with its potential candidate model surface 3. During the verification of hypothesis 75, scene region 23 is not matched to any model region. This is because only a small portion (less than 25 percent) of the cylindrical surface is visible in the scene, and the computed radius is off too much from its correct value to match to the candidate model region 5.

Note from Table III that most of the 75 hypotheses are rejected immediately during verification, without the computational burden of any feature matching. For each line entry in the table that ends in a 0, no features had to be matched during the verification stage; the hypothesis failed simply because no model features could be found in the vicinity of the tessels for the scene features used during verifications. In fact, as depicted at the end of the table, for the scene of Fig. 24, only 37 feature matching tests had to be carried out during the entire verification process. So, on the average, the system had to match only 0.49 features during each hypothesis verification. The largest number of features invoked for matching during any verification was 6, of these only three proved successful, as shown in Table III for hypothesis #75, confirming our $O(n)$ measure for the time complexity of verification.

This prototype system is programmed in *C* language and runs on a Sun-3 workstation. The CPU time for interpreting a processed range image was 9 sec. for the scene of Fig. 5 and 4 sec. for the scene of Fig. 24. The CPU time is approximately proportional to the number of generated hypotheses, which in turn depends on the complexity of the scene.

VIII. CONCLUSION

In this paper, we have presented feature matching and recognition strategies in 3D-POLY. For recognition, the system used an approach based on hypothesis generation and verification. The strategies used in the system lead to a polynomial time algorithm for the interpretation of range images.

The polynomial bound on the time complexity was made possible by two key ideas, one for hypothesis generation and the other for verification. The key idea in the former was the use of special feature sets, the spatial relationships between the features in these sets being such that the number of possible ways in which the scene features could be matched to those in the sets was substantially curtailed. The key idea in the verification stage was the association of a principal direction with a feature and, after the establishment of a pose transform, comparing a scene feature with a model feature only if the two agreed on the basis of their principal directions. This sharply reduced the number of scene and model features that had to be actually matched, leading to great savings in the computations involved.

To embed the notion of feature principal-direction in a computationally efficient framework, we represented model features on a feature sphere. We advanced a data structure for feature spheres and presented efficient algorithms for finding neighborhoods on the sphere and for assigning a tessel on the sphere to a measured principal direction.

We showed how our object recognition framework should be applied to scenes consisting of multiple objects in a heap. Finally, we discussed experimental results validating our complexity measures.

Appendix
Initial Guess for Tessel Assignment

Given a principal direction Φ, in this appendix we show how its corresponding tessel indices (i, j, k) can be computed from a linear approximation. Note that the indices thus computed are only supposed to place us in the vicinity of the true tessel. As explained in Section VI-C-3, the approximately located tessel is used as a starting point for getting the exact tessel corresponding to Φ.

We will present our approximation for the first of the parallelograms shown in Fig. 20, the approximations for the other parallelograms are identical in their j and k dependences by virtue of symmetry; the dependence on i is different and will be shown below.

The approximation for the first parallelogram in Fig. 20 actually consists of three separate approximations, one for each of the three zones that we will now identify. The first zone consists of the triangle marked 1 in Fig. 19, the second zone of the triangles marked 2 and 3, and the last zone of the triangle marked 4.

According to (9) in Section VII, the index i is independent of indices j and k in the computation of θ and ϕ for a given triplet (i, j, k). For a given (θ, ϕ), we can therefore separate the determination of index i from that of j and k. The procedure that follows consists of three steps:

1) First determine the identity of the zone to which the direction belongs.
2) Next, determine the index i corresponding to the parallelogram in which the direction (θ, ϕ) lies.
3) Estimate the indices j and k.

For the first two steps, the following formulas are used: (Let $\kappa = (2\pi/5)$, $\tau = \operatorname{atan}(2)$ and assume $0 \leqslant \theta \leqslant \pi$ and $0 \leqslant \phi \leqslant 2\pi$.)

```
if $(0 \leqslant \theta < \tau)$    /*$\Phi \in$ zone 1*/
        $i = \left\lceil \frac{\phi}{\kappa} \right\rceil \bmod (5)$
else if $(\tau \leqslant \theta < \pi - \tau)$    /*$\Phi \in$ zone 2*/
        $i = \left\lceil \frac{\phi - (\theta - \tau) \times \kappa/(2\pi - 4\tau)}{\kappa} \right\rceil \bmod (5)$
else /*$\Phi \in$ zone 3*/
        $i = \left\lceil \frac{\phi - \kappa/2}{\kappa} \right\rceil \bmod (5)$
```

For step 3, we will allow j and k to take non-integer values in the following formulas. During computations, the non-integer values are truncated to yield the integer values. First, let

$$\phi' = \phi - (i-1) \times 2\kappa.$$

If $\Phi \in$ zone 1,

$$k = \phi' \times \frac{(\theta \times Q/\tau)}{\kappa} + 1$$
$$j = \theta \times Q/\tau - k + 1.$$

If $\Phi \in$ zone 2, assume

$$\theta = aj + bk + c$$
$$\phi' = dj + ek + f.$$

Solving for a, b, c, d, e, f at the four corners of zone 2, we have

$$\theta = (j + k - 1)\frac{\pi - 2\tau}{Q} + 3\tau - \pi$$
$$\phi' = (k - j - 1)\frac{\kappa}{2Q} + \frac{\kappa}{2}.$$

Then j and k can be obtained by

$$j = Q \times \left[\frac{(\theta + \pi - 3\tau)}{(\pi - 2\tau)} - \left(\frac{2\phi'}{\kappa} - 1 \right) \right]$$
$$k = j + 1 + Q \times \left(\frac{2\phi'}{\kappa} - 1 \right) \bigg].$$

And for zone 3, we use formulas similar to those for zone 1, except that j and k are swapped, and angles θ and ϕ are appropriately offset.

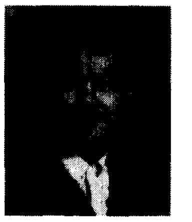
Acknowledgment

The authors would like to thank Matt Carroll and Jeffery Lewis, both research engineers in the Robot Vision Lab at Purdue University. Without their support, many experimental and software development aspects of the research reported here would not have been possible. Thanks are also owed to Whoi-Yul Kim for many valuable discussions on many aspects of range data processing and to Seth Hutchinson for his valuable contributions to our knowledge of robot manipulation.

References

[1] K. S. Arun, T. S. Huang, and S. D. Blostein, "Least-squares fitting of two 3-D point sets," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 9, no. 5, pp. 698–700, 1987.
[2] P. J. Besl and R. C. Jain, "Invariant surface characteristics for 3D object," *Computer Vision, Graphics, and Image Processing*, 33, pp. 33–80, 1986.
[3] ——, "Three-dimensional object recognition," *Computing Survey*, vol. 17, no. 1, pp. 75–145, Mar. 1985.
[4] B. Bhanu, "Representation and shape matching of 3-D objects," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 6, no. 3, pp. 340–350, 1984.
[5] R. C. Bolles and R. A. Cain, "Recognizing and locating partially visible objects: the local-feature-focus method," *Int. J. Robotics Res.*, vol. 1, no. 3, pp. 57–82, 1982.
[6] R. C. Bolles and P. Horaud, "3DPO: a three-dimensional part orientation system," *Int. J. Robotics Res.*, vol. 5, no. 3, pp. 3–26, Fall 1986.
[7] K. L. Boyer and A. C. Kak, "Structural stereopsis for 3-D vision," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-10, no. 2, pp. 144–166, Mar. 1988.
[8] B. A. Boyter, "Three-dimensional matching using range data," in *Proc. 1st Conf. Artificial Intell. Appl.*, Dec. 1984, pp. 221–216.
[9] R. A. Brooks, "Model-based three-dimensional interpretations of two-dimensional images," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 5, no. 2, pp. 140–149, 1983.

[10] C. M. Brown, "Fast display of well-tessellated surfaces," *Computer & Graphics*, vol. 4, pp. 77–85, 1979.
[11] I. Chakravarty and H. Freeman, "Characteristic views as a basis for three-dimensional object recognition," in *Proc. SPIE Conf. Robot Vision*, vol. 336, pp. 37–45, 1982.
[12] C. H. Chen and A. C. Kak, "3D-POLY: A robot vision system for recognizing objects in occluded environments," Dept. Elec. Eng., Purdue Univ., Tech. Rep. 88-48, 1988.
[13] ____, "Modeling and calibration of a structure light scanner for 3-D robot vision," in *Proc. IEEE Intl. Conf. Robotics and Automation*, pp. 807–815, Apr. 1987.
[14] R. T. Chin and C. R. Dyer, "Model-based recognition in robot vision," *Computing Survey*, vol. 18, no. 1, pp. 68–108, Mar. 1986.
[15] T. J. Fan, F. Medoni and R. Nevatia, "Matching 3-D objects using surface descriptions" in *Proc. IEEE Int. Conf. Robotics and Automation*, pp. 1400–1406, Apr. 1988.
[16] O. D. Faugeras and M. Hebert, "The representation, recognition, and locating of 3-D objects," *Int. J. Robotics Res.*, vol. 5, no. 3, pp. 27–52, 1986.
[17] ____, "A 3-D recognition and positioning algorithm using geometrical matching between primitive surfaces," in *Proc. 8th Int. Joint Conf. Artificial Intell.*, pp. 996–1002, 1983.
[18] G. Fekete and L. S. Davis, "Property spheres: A new representation for 3-D object recognition," *IEEE Workshop on Computer Vision*, pp. 192–201, 1984.
[19] C. Goad, "Special purpose automatic programming for 3D model-based vision," in *Proc. DARPA Image Understanding Workshop*, June 1983, pp. 94–104.
[20] W. E. L. Grimson and T. Lozana-Perez, "Localizing overlapping parts by searching the interpretation tree," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 9, no. 4, pp. 469–482, 1987.
[21] ____, "Model-based recognition and localization from sparse range or tactile data," *Int. J. Robotics Res.*, vol. 3, no. 3, pp. 3–35, Fall 1984.
[22] W. R. Hamilton, *Elements of Quaternions*. New York: Chelsea Publishing Co., 1969.
[23] C. Hansen and T. Henderson, "CAGD-based Computer Vision," *IEEE Workshop on Computer Vision*, pp. 100–105, 1987.
[24] B. K. P. Horn, "Extended Gaussian image," *Proc. IEEE*, vol. 72, no. 12, 1984, pp. 1671–1686.
[25] K. Ikeuchi, "Generating an interpretation tree from a cad model for 3D-object recognition in bin-picking tasks," *Int. J. Computer Vision*, vol. 1, no. 2, pp. 145–165, 1987.
[26] ____, "Determining attitude of object from needle map using extended Gaussian image," MIT AI Lab Memo no. 714, Apr., 1983.
[27] A. C. Kak, A. J. Vayda, R. L. Cromwell, W. Y. Kim, and C. H. Chen, "Knowledge-based robotics," in *Int. J. Production Res.*, vol. 26, no. 5, pp. 707–734, 1988.
[28] H. Kenner, *Geodesic Math and How to Use it*. Berkeley, CA: Univ. California Press, 1976.
[29] M. R. Korn and C. R. Dyer, "3-D multiview object representations for model-based object recognition," *Pattern Recognition*, vol. 20, no. 1, pp. 91–103, 1987.
[30] Y. Lamdan and H. J. Wolfson, "Geometric hashing: a general and efficient model-based recognition scheme," in *Proc. 2nd Int. Conf. Computer Vision*, pp. 238–249, Dec. 1988.
[31] R. Nevatia and T. O. Binford, "Description and recognition of curved objects," *Artificial Intelligence*, vol. 8, no. 1, pp. 77–98, 1977.
[32] M. Oshima and Y. Shirai, "Object recognition using three-dimensional information," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 5, no. 4, pp. 353–361, 1983.
[33] A. Pugh, *Polyhedra: A Visual Approach*. Berkeley, CA: Univ. California Press, 1976.
[34] A. A. Requicha, "Representations for rigid solids: theory, methods, and systems," *Computing Surveys*, vol. 12, no. 4, pp. 437–465, 1980.
[35] L. G. Shapiro and R. M. Haralick, "Structural descriptions and inexact matching," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 3, no. 5, pp. 504–519, 1981.
[36] Y. Shirai, *Three-Dimensional Computer Vision*. Berlin: Springer-Verlag, 1987.
[37] G. Stockman, "Object recognition and localization via pose clustering," *Computer Vision, Graphics, and Image Processing*, vol. 40, pp. 361–387, 1987.
[38] F. Tomita and T. Kanade, "A 3D vision system, generating and matching shape descriptions in range images," in *Proc. 2nd Int. Symp. Robotics Res.*, pp. 35–42, 1984.
[39] A. K. C. Wong and S. W. Lu, "Representation of 3-D objects by attributed hypergraphs for computer vision," in *Proc. Int. Conf. Syst. Man Cybern.*, pp. 49–53, 1983.
[40] S.-E. Xie and T. W. Calvert, "CSG-EESI: a new solid representation scheme and a conversion expert system," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 10, no. 3, pp. 221–234, 1988.
[41] H. S. Yang and A. C. Kak, "Determination of the identity, position and orientation of the topmost object in a pile," *Computer Vision, Graphics, and Image Processing*, 36, pp. 229–255, 1986.
[42] ____, "Determination of the identity, position, and orientation of the topmost object in a pile: Some further experiments," in *Proc. 1986 IEEE Int. Conf. Robotics and Automation*, 1986, pp. 38–48.



C. H. Chen was born in Kaohsiung, Taiwan, in 1960. He received the M.S. degree from the State University of New York at Stony Brook in 1984 and the Ph.D. degree from Purdue University, West Lafayette, IN, in 1988, both in electrical engineering.

He is currently with the Robotic Laboratory at SRI International, Menlo Park, CA. His areas of research interest include computer vision and sensor-based robotics.

A. C. Kak (M'71), for photograph and biography please see page 810 of the July/August 1989 issue of this TRANSACTIONS.