This is to certify that the

dissertation entitled

3-D REPRESENTATION AND RECOGNITION USING OBJECT WINGS

presented by

Sei-Wang Samuel Chen

has been accepted towards fulfillment
of the requirements for

Ph.D. degree in Computer Science

GCStockman
Major professor

Date 28 July 1989

3-D REPRESENTATION AND RECOGNITION USING OBJECT WINGS

By

Sei-Wang Samuel Chen

A DISSERTATION

Submitted to

Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Computer Science

1989

ABSTRACT

3-D REPRESENTATION AND RECOGNITION USING OBJECT WINGS

By

Sei-Wang Samuel Chen

A set of 2 1/2-D primitives, called *object wings*, are introduced for representation and recognition of general 3-D rigid objects. The rudimentary philosophy of wing formation is to integrate available contextual shape information of contour and surface into object features. We had several goals in mind when proposing this set of new features. The new features should be able to (1) handle general objects, (2) overcome imperfect feature extraction, (3) index models in a database and (4) compute object pose. In terms of this set of primitives, a theory concerning object representation, called *wing representation theory* is proposed.

There are 34 simple wings used as prototypes. A *simple wing* is defined as a triple including a pair of surface patches separated by a contour segment. Simple wings can be grouped into *composite wings* through non-accidental relationships such as proximity, similarity, connectivity, collinearity, curvilinearity, cotermination, parallelism and symmetry. Both simple and composite wings, together with their spatial structures, can be used to construct internal models of real world objects. Spatial structures include unary and binary geometric properties of object wings.

In terms of object wings, an associated computational framework of wing representation theory is introduced in which techniques of geometric modeling, view generation and wing representation are developed. In geometric modeling, an objects is specified by a set of triangles whose union geometrically approximates the object shape. An object

view can then be generated by projecting geometric model onto a plane perpendicular to a pre-defined viewing direction. A sphere whose surface forms a 2-D manifold of viewing directions (viewpoints) is referred to as *viewsphere*. Object views generated according to the viewpoints of the viewsphere will be clustered into so-called *aspects*. The object views within an aspect possess the same topological structure which is defined in terms of the wing sets of views. An object will be represented by a set of aspects which are in turn sets of object views. Each object view is characterized by wing features and their spatial structures.

In order to test the robustness of wing representations, the properties of uniqueness, stability, compatibility, terseness and locality are investigated. The uniqueness property requires that the representation of an object be distinguishable from representations of other objects. The stability property demands that small variations in the input have little influence on the final results. As for the requirement of compatibility, the object representations in a database should be comparable to the representations derived from real images. Object representation should also be terse enough to make processing efficient. Finally, locality enables the representation to cope with partial occlusion.

A recognition procedure based on wing representation is introduced. This procedure is composed of four steps to achieve the purposes of indexing, consistent labeling, parameter estimation and decision. Wing features detected in an image are first used to index models in the database. A model test process is then invoked to verify candidate models. This process uses interpretation tree search to determine the correspondences between sensed and model features. Afterwards, a view transformation algorithm based on the technique of curve matching is performed to estimate the parameters of object poses. Object recognition is finally accomplished by clustering the set of possible poses.

Experimental results support the claim that wing representations have the properties of uniqueness, stability, compatibility and locality, but not terseness. Further studying the tactics to solve the terseness problem will be needed. Experiments on recognition based on wing representation also reveal that wing features possess enough information to be of use for both indexing models and determining instance poses. In future research, we hope to extend wing representation theory and its associated techniques through composite wings to deal with more complex problems such as high level object representation and object segmentation in real images.

To the memory of my parents for providing me with the best education,

to my lovely wife Szu-Li for her support and encouragement,

and to my children Zouw-Hu and Zouw-Zen for their wonderful love.

ACKNOWLEDGEMENTS

I wish to thank my major professor, George Stockman, for his help and guidance, both in thesis preparation and in his encouragement of my research efforts. Special thanks should also go to Professors Richard C. Dubes and Anil K. Jain, who provide excellent knowledge in pattern recognition, image processing and computer vision. I also thank professors Robert Falco and C.Y. Wang, who serve on my Ph.D. committee and provided me with valuable comments, based on which this thesis was established. I sincerely appreciate the discussions I had with Dr. Mihran Tuceryan on several problems in perceptual grouping and human vision. I would like to give my special thanks to Deborah Trytten, Patrick Flynn and Joe Miller for their kindly enthusiasm about my writting.

Acknowledgement is due also to the other members of the Pattern Recognition and Image Processing Laboratory for diverse assistance: Dr. Nelima Shrikhande, Dr. Gongzhu Hu, Dr. C.C. Chen, Farshid Farrokhnia, Sally Howden, Jezching Ton, Ming-Xia Man, Tim Newman, Sateesha Nadabar and G. Lee. My thanks also go to the system managers Dongyul Ra and David Marks for providing me with reliable research environments.

Acknowledgement should also be made of the financial support I received during my long stay at MSU from NSF grant IRI-8600371 and CDA-8806599, and from Department of Computer Science.

Table of Contents

List of Tables

List of Figures

Ch

Chapter 1

Introduction

The recognition problem which humans solve effortlessly involves a number of difficulties for machine vision systems. According to vision psychologists [She84, Coo84], humans recognize objects because they saw "similar" ones before. Only objects "familiar" to humans can be recognized. Otherwise, humans "learn" the objects through a learning process that forms so-called mental models. A machine vision system, which copies even a small portion of human vision in object representation and recognition, involves tremendous effort. In this thesis, we do not attempt to complete an entire recognition system but focus on the development of theory and procedures which attack several critical problems commonly encountered by vision researchers.

In this chapter, the recognition problem to be solved is first defined. Primary tasks are then proposed. Afterwards, object modeling and recognition are reviewed. Finally, the organization of this thesis is presented.

1.1 Human Visual Processing

Human visual perception, a natural process, involves a series of complex processes which to date are not clearly understood [Gib52, Sek85]. These processes create a "mind's eye" [Wol86] and proceed far beyond what is present on the retina. Research in anatomy, neurophysiology and psychophysics has shown that the visual input goes through a number of associated nerve components. Visual perception arises from the

cooperation of these components which use information previously collected as the basis for inferences about the state of the world.

Since what our eyes receive is just a collection of incomplete information, scene understanding should entail a great deal of vision intelligence in which sensory data are analyzed, fused and interpreted. Therefore, what we see is a creation of our intelligent mind. There are three fundamental steps in a visual process: reception, extraction and inference [Wol86]. The first step is done by receptors in the retina that get external stimuli. Then, the input is forwarded to the visual cortex of the nerve system, where preeminent features such as edges, surfaces, orientation, motion, color and texture, are extracted. These features may further be processed to form higher level features by which an intermediate representation of the scene is established. It is this intermediate representation and the internal mental models that visual intelligence uses to infer the state of the world.

Aside from these mental representations, it is generally believed that the visual system possesses additional capabilities for computations on visual elements. Such computations are called "unconscious inferences" in 19th century by German scientist Hermann von Helmholtz. Although psychologists, psychophysicists and mathematicians have provided substantial means to study inference rules from various fields such as illusion [Kan76, Gil80], hallucinations [Sie77], mental models [Coo84], human brain, visual machinery of animals [Hor77] and differential geometry [Whi55, Koe84], a comprehensive investigation into the rules is nevertheless a nontrivial task.

1.2. Machine Vision Problems

A large assortment of machine vision problems can be classified in terms of several aspects along which one can constrain the problem. Some prominent aspects include:

a) Applications - inspection, automation, manipulation or navigation.

b) Goals - identification, localization or both.

c) Techniques - engineering, biological or both.

d) Environments - indoor or outdoor.

e) Status - moving or static objects.

f) Configurations - single object, multiple objects, macroscopic or microscopic.

g) Objects - natural, manufactured, rigid, nonrigid, regular or sculpted.

h) Sensing systems - monocular, stereo, active or passive.

i) Input data - 2-D or 3-D.

Each group can be divided into finer subclasses; for instance, a stereo sensing system could employ photometric or binocular techniques. In general, to design a system which needs to deal with a complex problem or several different problems simultaneously, it is better first to design a number of system components such that each handles a small problem. The complete system is then created by integrating the components in a proper way. This strategy is commonly referred to as the principle of *modularization*. It can also be found in the biological world, for instance, the compound eye of insects [Hor77] in that each eye has its own channel to perform a particular visual function and then scene interpretations are performed by fusing the information come from individual channels. The spatial frequency multi-channels of human visual system [Mar82] is, in some sense, analogous to this.

1.3 Overview of Problem Area

In this thesis, we limit ourselves to a restricted 3-D recognition problem. This problem can be stated as follows.

> Given a range image of a scene which can be a jumble of either distinct or similar rigid objects, specific objects in the scene are to be recognized based on the given single image and a set of object models.

Examples of scenes are shown in Figure 1.1. They are intensity images for illustrative purposes. Experimental data will be range images of such kinds of scenes. In the left picture of the figure, the scene is composed of a coffee cup, a wooden block and a clay

model of a cobra head. In the jumble on the right the coffee cup has been replaced by a toy tank. The working objects are rigid and opaque but can be arbitrarily shaped.

Range images are a kind of intrinsic image, which contain depth information from scenes. They can be directly obtained using range finders or derived from techniques of *stereopsis* or *shape from approach*. Range values are sensory data with rich shape information from which a set of shape descriptors such as contours, surface normals, gradients and curvatures are readily computed. Although range images provide such explicit shape information, the entire task of recognition is still a formidable task. In this thesis, range images are provided by an active structured light sensing system (available in Michigan State University, Pattern Recognition and Image Processing Laboratory PRIP) and a laser range finder (in Environmental Research Institute of Michigan ERIM). The goal of this research is to deal with both the identification and localization problems.

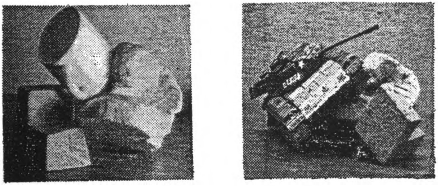


Figure 1.1. Examples of jumbles of objects.

It is believed that there is a learning process in human vision system to tackle unfamiliar objects. If a machine vision system is to simulate the behavior of human visual system, then two questions arise: how to implement the learning process and how to store object models. Aside from these high level problems, a smaller issue states that shapes depicted in images are usually portions of objects because of occlusion and image formation. How can an incomplete appearance of an object be recognized by a machine vision system? These problems have long troubled computer vision researchers.

For the time being, we do not attempt to complete an entire recognition system for dealing with all of these problems, but focus on the development of theory and procedures which attack several critical problems commonly encountered by a machine recognition system.

(1) Introduction to wing primitives and investigation of their properties.

(2) Development of a 3-D object representation theory using the proposed wing primitives.

(3) Development of a computational framework for implementation of wing representation theory.

(4) Development of a recognition procedure based on the wing representations of objects.

Figure 1.2 depicts a proposed recognition system and some major tasks addressed by this research.

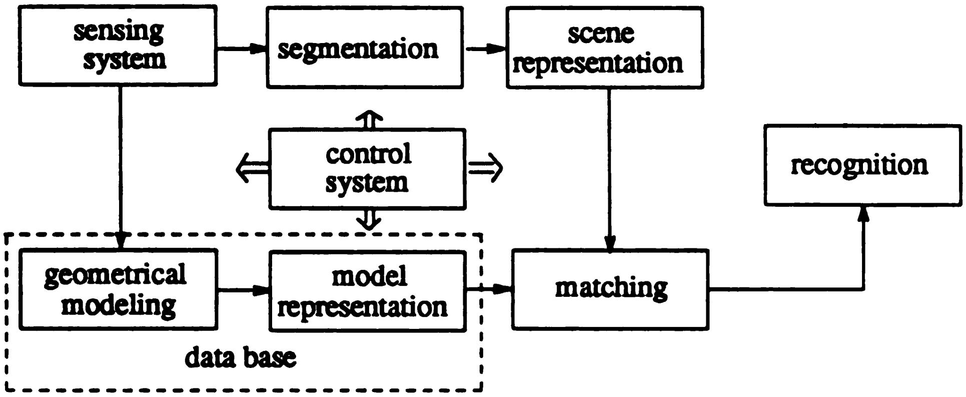


Figure 1.2. Diagram of a presumed recognition system.

1.4 Related Research on Object Modeling

Object modeling and *model-based* recognition have become increasingly important since Roberts [Rob65] used object models to identify a set of blocks. A wide variety of representation schemes have since then been proposed to provide configurations for building databases of objects. Boundary-based, volume-based and parameter-based approaches are commonly adopted. Most schemes favor objects with regular shapes such as planar, quadric surfaces and cylinders. Few people have paid serious attention to the problem of building models for arbitrarily-shaped objects. Possible reasons are that the tasks for modeling such general objects seem implausibly complex, and that researchers hadn't been provided with strong evidence from either psychology or biology to confirm the notion of the model-based recognition paradigm.

1.4.1 Psychological Evidence

Psychologists [She71,84, Coo75,84] have designed a series of experiments using single-stimulus and paired-stimulus techniques to quantitatively demonstrate several interesting aspects of mental process. In their experiments, three-dimensional "armlike" blocks generated from computers are depicted as perspective line drawings. Through the study of mental operations from those line drawings, they came to the conclusion that there are ***mental models***, learned from visual experience, stored in our brain. Object recognition in the human visual system is then performed by matching mental models to real world objects. This immediately gives rise to the question of how such dynamical matching operations proceed in our brain. In order to answer this question, they demonstrated that matchings could be done by *mental transformations* of models that are analogous to their physical counterparts. Evidence of greater importance to computer vision researchers is that the subjects' mental images represented the three-dimensional structure of the objects portrayed and not simply the two-dimensional features of the drawings. This might give us clues to how models are configured.

From experiments executed by Shepard, Metzler and Cooper, the reasoning that representation models play an important role in the tasks of object recognition was confirmed. Their persuasive results also showed that object models should possess spatial relationships among part components. Unfortunately, they didn't give the answers to what object parts are, how the object can be decomposed into parts, and what relational evidence between parts can be used to construct a model. A feasible answer to the first question may be found in the classical work of Gibson [Gib50]: the elementary impressions of the human's visual world are those of surface and edge. This *principle of surface-edge evidence* looks so simple and straightforward, that it has dominated the thinking of researchers within the computer vision community.

1.4.2 Theory of Parts

Hoffman & Richards [Hof84] introduced a *minimum rule* to decompose objects into components, based on the uniformities of nature (the regularity of *transversality*) to partition object surfaces into so called natural parts. This regularity says that when two arbitrarily shaped surfaces are made to interpenetrate, they always meet in a contour of concave discontinuity of their tangent planes. Therefore, the partition rule can be simply stated: object surfaces can be divided into parts at loci of negative minima of each principal curvature along its associated family of lines of curvature.

However, this rule suffers from several weaknesses. First, only surfaces complying with the assumptions of differential geometry can be segmented using this rule. Secondly, contours subject to the minimum rule may not be closed. Partitioning based on such contours will inevitably produce ambiguous choices. Finally, curvature computations relying on second derivatives are noise-sensitive and fragile. Aside from the weaknesses of the minimum partition rule, Hoffman and Richards haven't presented methods for describing object parts once they have been extracted. A solution to this problem was suggested by Pentland [Pen87] and described below.

1.4.3 Lump of Clay and Fractal Model

Pentland proposed a computational theory and 56 descriptive parts characterized by a parameterized family of shapes, called *superquadrics*, which are described by the following equation:

$$X(\eta, \omega) = \begin{bmatrix} \cos^{\varepsilon_1}\eta \cos^{\varepsilon_2}\omega \\ \cos^{\varepsilon_1}\eta \sin^{\varepsilon_2}\omega \\ \sin^{\varepsilon_1}\eta \end{bmatrix}$$

where η and ω are latitude and longitude with respect to the coordinate system of each individual part and ε_1 and ε_2 are parameters which control the surface's shape. This family of functions can describe cubes, cylinders, spheres, diamonds, pyramidal shapes and intermediate shapes. Regardless of size factor, each primitive can be sufficiently represented by the two parameters ε_1 and ε_2. Afterward, these basic primitives can be deformed by bending, stretching, twisting or tapering to form more complex prototypes. High level objects are then represented by a set of prototypes, a collection of deformations, and a sequence of Boolean operations. Clearly, this process of object formation is identical to the process of constructive solid modeling [Req80].

Superquadrics are still constrained by the assumptions of smoothness (differential continuity) and isotropy (symmetrical shape). A *fractal model* [Pen87] was proposed to relieve the restriction. This model is in fact governed by a topological dimension T and a fractal dimension D, which satisfy the relation $D = T + r$, where r is the ratio of the number of features (i.e., the fractal dimension) of one size to the number of features of the next larger size and is a constant. The topological dimension T controls the general outline of an object, whereas the fractal dimension D controls the roughness of object surfaces. Therefore, if ($D = T$ and $r \approx 0$), then shapes are smooth; while if ($D \gg T$ and $r \approx 1$), then shapes are rough.

Although Pentland has proposed efficient models to describe a wide range of part primitives, most of his work focused on model building rather than methods of how to recognize objects using models characterized by his proposed part category. A possible reason that he hadn't proposed the recognition procedure is that the recovery of part primitives (or part parameters) from image data is difficult and the computational method is still unreliable.

1.4.4 Recognition-By-Components

Biederman [Bie87] proposed a theory of *Recognition-By-Components* (RBC) from a psychological viewpoint. He presented a set of 36 volumetric primitives, called *geometrical ions* (geons), which is a sub-collection of generalized cylinders [Bin71]. His experiments on the theory of RBC demonstrated that humans can often recognize objects based on a small set of geons (three will be enough for a moderately complicated object) and their spatial relationships. Although the conclusion is appealing, the extraction of geons from images is still governed by Hoffman and Richards' minimum rule. Inferring volumetric components from 2-D information would easily suffer from viewpoint variation, which means that different primitives might be instantiated for the same component from different object views.

1.4.5 Conclusion on Part Representation

Certainly, the representation problem must eventually be confronted by researchers in model-based recognition. This problem is dependent on the domain of application. In many circumstances, a single representation scheme is not enough to represent all working pieces. A multiple-scheme representation may provide flexible choices but will increase complexity of a system. Furthermore, sophisticated objects may demand more powerful representation schemes.

The part models enjoy several advantages and seem to offer a paradigm with potential for the near future. They can handle assorted configurations of articulated components without substantial adjustments. A finite vocabulary of primitives can also provide tremendous power in dealing with an overwhelming class of objects by combining primitives. The database of a recognition system using part models is usually easier to refine when new objects are encountered that require the addition of primitives, and is therefore flexible. Moreover, these systems can handle objects even though they are transformed, degraded or occluded.

Unfortunately, so far the proposed part models and their associated theories are too idealized to be implemented in a machine vision system. Thus, we fall short of our ambition to directly extract 3-D part components from image data; we instead look for features of *object wings* that are 2 1/2-D primitives (see Section 2.3), and attempt to develop a representation theory, called *wing representation theory*, in terms of this set of primitives.

1.5 Related Background on Model-Based Recognition

Few recognition systems based on part models have been developed. Most existing model-based recognition systems use vertices, holes, edges, arcs and surfaces as features. Several methods adopted in our research are closely related to the previous work and described below.

Goad [Goa83] introduced a system which could automatically construct a fast special purpose program through path search for recognizing and locating objects from a pile. Although consistent labeling is done by exhaustive search, the precomputation of parameter ranges efficiently narrows down a search space and provides potential constraints for determination of viewpoint. The concepts of *automatic programming* and the *locus image* representation of objects are the most notable ideas in his research. Later, these notions were adopted by Ikeuchi and Kanade [Ike88] and extended by introducing techniques for automatically compiling both object and sensor into a recognition process, which uses a *multiple-aspect* scheme to represent objects.

Lowe's work [Low87] pioneered the idea of *perceptual organization* for low level feature groupings and championed the technique of *3-D from 2-D* recognition. Viewpoints are computed through the Newton numerical method, which recursively modifies a vector of view parameters starting from a predicted viewpoint. This method exempts 3-D recognition from the need of recovering depth information. Similar concepts can be found in other work [Fau86, Tho87, Hut87, Lin88, Sho88], where the positions of objects are computed by aligning model features with detected image features.

Jain and Hoffman [Jai88, Hof87] developed an *evidence-oriented* technique for identifying 3-D rigid objects with arbitrary shapes. A *rule-based* database was constructed using evidence conditions based on notable features to represent objects. In their approach, techniques of pattern recognition were used to determine similarities between model and scene representations. They also developed a learning process for automatically extracting evidence conditions from different views of objects. Their recent work has considered pose determination but not multiple-object scenes.

Grimson and Lozano-Pe´rez [Gri87] used an *interpretation tree* search to solve the feature correspondence problem. They adopted a set of constraints on surface patches in terms of their distances and angles to prune the tree search. Pose determination is performed by computing a 3-D transformation having up to six degrees of freedom relative to the sensor.

Other model-based vision systems that have influenced our research include ACRONYM [Bro84], 3DPO [Bol83] and the works of Schwartz & Sharir [Sch87], Lamdan & Wolfson [Lam88], Basri & Ullman [Bas88], and Shoham & Ullman [Sho88]. Schwartz and Sharir developed an elegant 2-D *curve matching* technique. This method transforms matching problems into optimization problems. It uses a complex mapping technique to transform rotations of Euclidean vectors into multiplications of $e^{i\theta}$ terms in the complex space. We extend their method (in Chapter 6) by including a scale factor when computing transformations. Lamdan & Wolfson introduced the *geometric hashing* technique using coordinates of multiple-reference frames as keys to access information of critical points. Through this data structure, both object class and frame basis can be efficiently accessed. Object pose is then determined based on the given frame basis. Basri & Ullman presented a method based on point curvatures to predict the silhouette of a new object pose from the silhouette of a given object pose. This method allows objects to be represented by a small number of object views. A new appearance of an object can be estimated from the representative views. One advantage of this method may be its terseness in object representation because only a few object views are involved. Shoham & Ullman introduced a 3-D from 2-D recognition technique which allows the object pose to be determined without recovery of 3-D information.

1.6 Organization of the Thesis

The remainder of this thesis is devoted to a theory called *wing representation theory*. Starting with this theory, the associated computational frameworks for representation and recognition are designed. Wing representation theory is characterized by a set of new primitives referred to as *object wings*. Definitions and properties of these wings are investigated in Chapter 2.

A computational framework of wing representation forms the major content of Chapter 3 in which techniques for representation are addressed. Then, we turn in Chapter 4 to an implementation of the representation theory. It includes three stages. The first stage is to construct the geometric models of objects. The second stage is to augment models by adding surface characteristics to the geometric models. In the final stage, objects are represented by multiple viewpoints in terms of wing primitives.

In order to examine the adequacy of wing representation, the properties of uniqueness, stability, compatibility, terseness and locality are tested. This is done by analyzing the influences of convolution masks, view and map resolutions on object representations. Techniques of feature extraction with real images are studied to examine the consistency between the wings detected in images and the model wings in the database. All these studies are collected in Chapter 5.

In Chapter 6, a recognition procedure based on wing representation is designed and tested. Recognition must achieve two goals: identification and localization of objects. The recognition process involves four stages: indexing, consistent labeling, parameter estimation and decision. Associated techniques for implementing these stages include feature comparison, interpretation tree search, curve matching and clustering.

Principle contributions, implications of this research, and future work are summarized and discussed in Chapter 7.

Chapter 2

Representation Theory: 2 1/2-D Object Wings

Recently, *part models* have attracted many vision researchers' attention for representation of 3-D real world objects. Among those models, natural components [Hof84, Bie87] have received much appeal because they reflect the physical structure of objects at a high level of abstraction. Such abstraction is generally believed to be enough for object recognition. Meanwhile, associated computational theories of part models have also proliferated [Bar84, Hof84, Tver84, Bra85, Pen87, Bie87, Baj87].

Unfortunately, most models have suffered from a number of difficulties in implementation. The most bewildering concern may be the reliability of extracting 3-D components from 2-D image data, which can easily lead to computational instabilities and problems of uniqueness. These difficulties have hampered the application of part models to representation and recognition problems even though they possess a number of elegant properties.

In this chapter, a set of primitives, called *object wings*, is defined to serve as a feature set for 3-D representation and recognition. Wings are object features which contain shape information of contour and surface. Major considerations for construction of wings are to be able to handle general objects, overcome imperfect feature extraction, index models and compute object poses. Thirty-four simple wings are proposed for practical use. They are defined as triples consisting of a pair of surface patches separated by a contour segment. Composite wings with higher level configurations are then formed by

2.1.

and

be a

Def

be sp

grouping simple wings that comply with certain non-accidental relationships such as curvilinearity, symmetry and parallelism.

Although many researchers have already worked on features of contour and surface, most researchers considered them separately. Object wings are, in fact, compound features of contour and surface. In general, the properties of a compound feature will be different from the properties of individual features which form the compound one. In this chapter, definition and property of object wings are studied.

2.1. What are Wings?

Suppose one can use some underlying segmentation techniques to extract contours and regions from an input intrinsic image. Let **C** denote a set of detected contours and **R** be a set of regions detected from an image.

$$\mathbf{C} = \{c_1, c_2, \cdots, c_{m_1}\}\ ,\quad \mathbf{R} = \{r_1, r_2, \cdots, r_{m_2}\}.$$

Definition 2.1: A region r_i is said to be *adjacent* to a contour c_j if there exists at least a pair of pixels $p \in r_i$ and $q \in c_j$, which are eight-connected. In other words, q is a boundary pixel of r_i. A binary relation, called *cr-adjacency*, between a contour c_j and a region r_i, denoted by α, is defined as follows: $c_j\ \alpha\ r_i$ iff c_j and r_i are adjacent.

Let $\mathbf{C}_a$ represent an N-tuple of contour attributes and **T** be a set of permissible instances for the tuple. Each instance $\mathbf{t}_i$ of **T** is, therefore, an N-dimensional vector in which each component corresponds to an attribute. The properties of a contour can thus be specified by a vector, where

$$\mathbf{T} = \{\mathbf{t}_1, \mathbf{t}_2, \cdots, \mathbf{t}_{n_1}\}.$$

In a similar vein, let $\mathbf{R}_a$ denote an M-tuple of region attributes and S be a set of permissible instances for that tuple. Each instance $\mathbf{s}_j$ of S is an M-dimensional vector of values associated with region attributes defining the properties of a region. Since wing primitives will be defined along detected contours, the adjacent regions of a contour may not be available in some cases. Thus the set S needs to include an additional vector, called the *null* instance, denoted by $\mathbf{s}_0$, to represent an undefined region. Let

$$\mathbf{S} = \{\mathbf{s}_0, \mathbf{s}_1, \cdots, \mathbf{s}_{n_2}\}.$$

Definition 2.2: A *wing primitive* w is configured as a contour with its adjacent regions, and is characterized by a triple, which is in the product set $\mathbf{S} \times \mathbf{T} \times \mathbf{S}$.

2.1.1 Complete Set

Let $\mathbf{W} = \{ (s_i, t_j, s_k) \,|\, i,k = 0,\cdots n_2 \,;\, j = 1,\cdots n_1 \}$ denote the *complete set* of wing primitives. There are $n_1(n_2+1)^2$ distinct members in $\mathbf{W}$. From analyzing the process of image formation, it is not difficult to discover that several of these combinations are impossible.

Proposition 2.1:

The complete set **W** is a superset of valid wing primitives under general image formation processes. Equivalently, not all combinations of contours and surfaces can have corresponding physical wings.

Proof: To prove this, recall Koenderink's theorem [Koe84] about solid shape from occluding contour. Referring to Figure 2.1, consider point P on the transverse curve. Let K denote the Gaussian curvature, K_r represent the radial curvature, and K_t be the transverse curvature, all of which are defined at point P in the 3-D space. Let K_{app} be the apparent curvature at the projection point P' of P on the image plane, and d be the distance from the viewer to P. Then, Koenderink's theorem simply states that

$$K_r K_t = K \quad \text{and} \quad K_{app} = K_t d = \frac{Kd}{K_r}. \tag{2.1}$$

Equation (2.1) indicates that K_{app} has the same sign as K_r since both K and d are positive. If K_r is positive, then the part of the radial curve with negative curvature K_r will be self-occluded, or the corresponding radial curvatures can not change signs when tracing along the apparent curve. For example, if there is a valley (concavity, $K_r < 0$) in a radial curve, there cannot be a hill (convexity, $K_r > 0$) just in front of it. Otherwise the valley would not be seen because of the occluding hill. A similar analysis can be

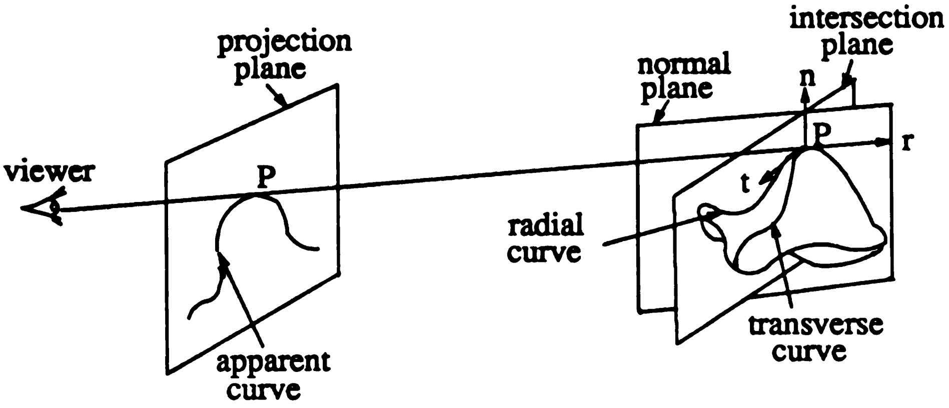


Figure 2.1. Diagrammatical definitions of the Koenderink theorem.

performed for the case of $K_r > 0$. Figure 2.2 clarifies this.

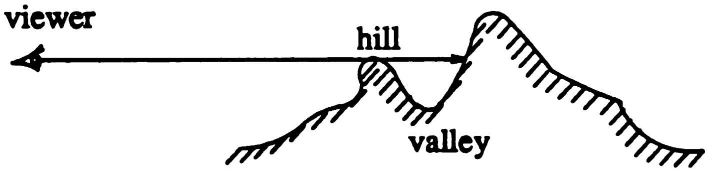


Figure 2.2. If there is a hill appearing just in front of a valley, the valley will be invisible to the viewer.

From the above specific example, it is known that the region adjacent to a concave contour generated by an object limb cannot be a convex surface. We conclude that not all the combinations of contours and surfaces are possible in a projection plane. ■

As a matter of fact, Koenderink's theorem can provide six more rules by which some additional members of **W** can be removed. To collect the complete set of rules is not impossible but also not easy. Potential rules may come from differential geometry [Bra85], physics [Cae82] and mathematics [Nal88]. The proof of Proposition 2.1 indicates a way to find a valid set of wing primitives. For a triple in **W**, if there exists a corresponding physical wing, the triple will be called a *legal triple*. The resulting collection of legal triples forms the *legal set* of wing primitives.

2.1.2 Proper Sets

In many situations, it may be unnecessary to use the whole legal set for practical implementation. For instance, if the direction of contours is not important to a task, the set of wing primitives can be reduced by ignoring the directional attributes of contours. Let us call such a process a *collapsing process* and the resulting set of primitives the *collapsed set*. In contrast to the collapsing processes, there are *augmenting processes* by which one can extend the wing catalogue. Let us call the resulting set the *augmented set*. Both collapsed and augmented sets are proper sets of complete set **W**.

One of the majcr purposes for collapsing wing primitives is to simplify the catalogue. However, when the shapes of objects become sophisticated, more detailed wing primitives will be needed. By collapsing and augmenting processes, one can create as many "scaled" catalogues as needed. Then based on the set of scaled catalogues, a *coarse-to-fine* recognition system can be developed. For some cases, such hierarchical strategies can improve performance and reliability of a recognition system.

2.2. Physical Meaning

As mentioned earlier, a region was defined by a set of attributes. In general, any available information of geometry, topology and statistics can form feasible attributes for regions. However, the availability of feature properties typically depends on the input data, for example range or intensity data. It is usual to include only the necessary constraints for applications. For this reason, in this section attributes of contour and region types are considered. To further simplify the discussion, restricted sets of feature types are employed.

Define the set of contour types as {<<, >>, <, >, +, -}. The first two symbols represent limbs. The next two symbols denote occluding convex edges, and the last two elements represent convex crease and concave crease edges respectively. Here, we have followed the notation used in the analysis of line-drawings [Mal87]. For simplification, the set of contour types can be collapsed into {!, <->, +, -} by ignoring orientations of limbs and occluding convex edges. This set then represents silhouettes, jump edges (self-occluding only), convex crease and concave crease edges. Next, define a set of surface types as {n, p, +, -, s}, which represents *nil*, plane, convex and concave surfaces, and saddle respectively. The *nil* type is included in order to represent each wing primitive as a triple, in case an object wing has an undefined adjacent region. Note that there is no *nil* type for contours because wing primitives have to be defined along known contours.

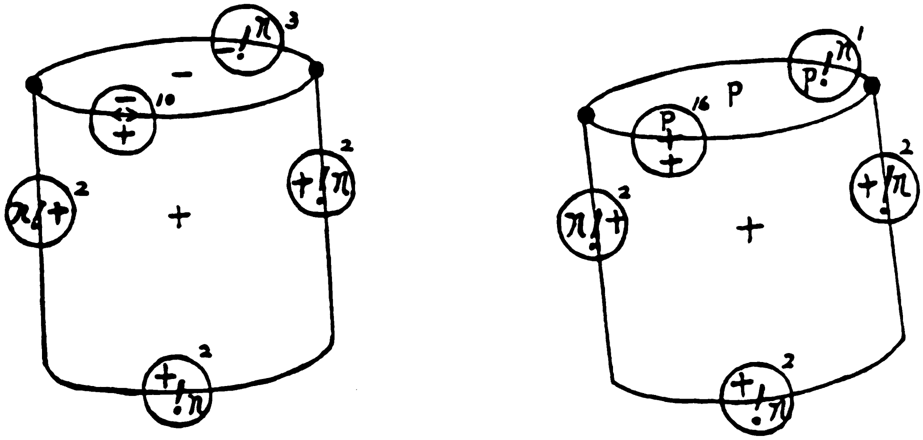
silhouette wing	1 n!p	2 n!+	3 n!-	4 n!s	WING TRIPLE (surface, contour, surface)					
jump wing	5 p↕p	6 p↕+	7 p↕-	8 p↕s	9 +↕+	10 +↕-	11 +↕s	12 -↕-	13 -↕s	14 s↕s
convex crease wing	15 p+p	16 p++	17 p+-	18 p+s	19 +++	20 ++-	21 ++s	22 -+-	23 -+s	24 s+s
concave crease wing	25 p-p	26 p-+	27 p--	28 p-s	29 +-+	30 +--	31 +-s	32 ---	33 --s	34 s-s

surface types: n: null; p: planar; +: convex; -: concave; s: saddle.
edge types: ! : silhouette; ↕ : jump edge;
+ : convex crease; - : concave crease.

Table 2.1. Set of wing types for practical implementation.

Based on the definition of low-level feature types (four contour types and five surface types), in principle, $5^2 4 = 100$ distinct wing types can be defined. As shown earlier in Lemma 2.1, not all these wing types have corresponding physical wings; in other words, not all these wing types are legal. A set of wing primitives for practical implementation is shown in Table 2.1, in which 34 distinct wing types have been filtered out from the complete set of 100 wing types by examining the legality of each wing type. There is no formal proof provided in this thesis for the type filtering.

Let us classify this set of 34 wing types into four subgroups according to the contours along which they are defined: silhouette wings, jump wings, convex-crease wings and concave-crease wings. This results in four distinct silhouette wings and ten wings for each of the other subgroups. Examples of synthetic wing primitives defined on the line drawings of a coffee cup and a coke can is shown in Figure 2.3, where the small black circles denote the boundaries of wing primitives.



(a) coffee cup

(b) coke can

Figure 2.3. Synthetic wing primitives defined on the line drawings of objects.

Wing types listed in Table 2.1 can be easily extended by any suitable augmenting processes. We present three extension principles for future use with objects that have more complicated shapes. First, wing types can be extended through introducing finer classifications of either contour or surface types or both. For example, a convex surface can be further classified into peak and ridge surfaces, and a concave surface can be further divided into pit and valley surfaces. Similarly, for a saddle, it can be classified into subtypes of minimal surface, saddle ridge and saddle valley [Bes86]. Secondly, wing types can also be defined along the projections of any kind of spatial curves such as lines of curvature, geodesic curves, asymptotic lines, parabolic lines or critical lines, as long as they can be detected reliably. Finally, wing types can be further detailed by including orientations and curvilinearities of contours. For example, "+(-" and "+)-" can be categorized as different wing types, where "+" and "-" represent convex and concave surface types respectively, and "(" and ")" denote contours with different directions of curvature.

2.3 Simple Wings

In order to compensate for difficulties of feature extraction, rather than applying restoration procedures to the contour maps, such as cleaning (noise), filling (broken), recovering (missing) and grouping (structure), we simply include the contextual surface information with each detected contour. By this, we mean the surface type of regions adjacent to the contour. Now, a simple wing can be described as follows.

> ***A simple wing** is composed of a pair of 3-D surface patches (one but not both of which can be nil) separated by a 2-D contour segment. Therefore, it contains both 2-D and 3-D shape information and is aptly called a 2 1/2-D object primitive.*

Later on, we will present a rule to extract object wings from images. Here let us first investigate some notable properties of wing primitives. Object wings are entirely characterized by local properties. These properties enable recognition even when objects are transformed, occluded and degraded. It is the contour information of wings that makes the parameter estimation of object poses feasible in the final stage of recognition. This will be discussed further in Chapter 6.

Regions adjacent to a contour in a projection plane may not necessarily adjoin each other on the object surface. For instance, one of the regions adjacent to a detected jump edge will not be adjoining the corresponding contour in 3-D space. Therefore, through an appropriate rotation a jump wing can change to a convex-crease wing, and vice versa. Thus, object wings are not viewpoint invariant. This property is called *type variation* and can be used to delimit aspects [Ike88] of an object in a multiple-view representation.

2.3.1 Extraction Rule

Suppose that a reasonable collection of contours and surfaces have been achieved using some segmentation techniques. There are two stages in labeling wings along contours: *dense labeling* and *sparse labeling*. Dense labeling assigns a wing type for each contour pixel, whereas sparse labeling assigns a wing type for an entire contour segment.

In experiments, dense labeling is performed prior to sparse labeling. Boundaries of wing primitives are then located by the following rule:

Extraction Rule: *Wing boundaries are located on a contour at locations of change in either the contour type or adjoining surface types or both.*

The implementation of this rule can be described as follows. First dense labeling is performed on the detected contour pixels. Then, connected contour pixels with the same label are grouped to form a wing primitive. Note this rule depends only on qualitative types. It should be more reliable than the minimum rule [Hof84] which is based on quantitative curvature values.

2.3.2 Structural Constraints on Wings

Investigating the structural properties of wing primitives not only provides a further understanding of wings, but gives an indication of how a set of unrelated primitives can be organized into more meaningful configurations. There are two kinds of wing constraints: *unary constraints* and *binary constraints*. The unary constraints indicate properties of individual wings, whereas the binary constraints specify relations between two wings. Constraints for more than two wings are possible but beyond the scope of this research.

By using a multiple-view representational scheme, constraints are only required to be invariant within a viewpoint. That means the structural properties of wings exist in 2-D space rather than the 3-D space. For this reason, one has wide flexibility to choose wing constraints. A set of experimental characteristics for defining unary constraints for individual wings is given below. Note here that for explicitness, "curve" and "contour" will be used to denote lines in 3-D and 2-D spaces respectively.

1) *3-D wing length*: the length measured along the corresponding 3-D curve of a 2-D wing contour.

2) *3-D depth range*: the difference between the maximum and the minimum range values along the corresponding 3-D curve of a 2-D wing contour.

3) *2-D angle span*: moving the tangent vector field of a 2-D wing contour to a predefined coordinate system; the angle span of the wing is then defined as the angle sector swept by the tangent vector field.

4) *3-D maximum and minimum turning angles*: a turning angle is defined as the angle swept by the tangent vectors of two adjacent points on the corresponding 3-D curve of the 2-D wing contour.

5) *2-D maximum and minimum turning angles*: similar to Item 4 but directly defined along a 2-D wing contour.

6) *3-D maximum and minimum tangent magnitudes*: measured along the corresponding 3-D curve of a 2-D wing contour.

7) *3-D maximum and minimum curvatures*: measured along the corresponding 3-D curve of a 2-D wing contour.

8) *2-D maximum and minimum curvatures*: similar to Item 7 but directly measured along a 2-D wing contour.

Parameters of the wings are computed in terms of their geometrical properties. Before computation, a circular operator is applied to resample a set of points along each wing contour. Thus, distance intervals between any two points are the same, which equals to the radius of the circular operator. A piecewise cubic B-spline **r**(t) can be used to obtain 2-D parameters of curvature, tangent magnitude, and tangent vector at every sampled point. A point $r(t)$ of a cubic spline curve is defined as $r(t) = \sum_{i=0}^{n} P_i N_{i,k}(t)$, where P_i represents the control points, $N_{i,k}(t)$ denotes the blending function, and k is the order of the curve. The blending functions, also called basis splines, can be constructed via the recursion

$$N_{i,k}(t) = \frac{(t - x_i)N_{i,k-1}(t)}{x_{i+k-1} - x_i} + \frac{(x_{i+k} - t)N_{i+1,k-1}(t)}{x_{i+k} - x_{i+1}},$$

$$N_{i,1}(t) = \begin{cases} 1 & x_i \le t \le x_{i+1} \\ 0 & otherwise \end{cases},$$

$$N_{i,1}(t) = 0, \quad \text{if } x_i = x_{i+1},$$

where x_i's are knots from a predefined knot vector. By defining a uniform knot vector, a point on the piecewise B-spline can be written as

$$\mathbf{r}(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = [P_0, P_1, P_2, P_3]\,\frac{1}{6}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 0 & 4 \\ -3 & 3 & 3 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}, \tag{2.1}$$

where t is the parameter of the spline defined on the knot vector, and P_i's are control points. From this equation we can compute the first and second derivatives of vector function $\mathbf{r}(t)$, such that the tangent vector $\mathbf{T}$ and curvature k are

$$\mathbf{T} = \frac{\mathbf{r}'}{|\mathbf{r}'|}, \quad k = \frac{|\mathbf{r}' \times \mathbf{r}''|}{|\mathbf{r}'|^3}. \tag{2.2}$$

For 3-D cases, the vector function $\mathbf{r}(t)$ of Equation (2.1) includes one more component $z(t)$ and the control point vector becomes a 4 by 4 matrix with the constant matrix remaining unchanged.

A set of relational structures for defining binary constraints between two wings includes:

1) *3-D maximum and minimum distances*: the maximum and minimum distances between the corresponding 3-D curves of two 2-D wing contours.

2) *Ratio of 2-D minimum to maximum distances*: the ratio of the minimum to maximum distances between two 2-D wing contours.

3) *3-D maximum and minimum distances between the convex hulls of centers of gravity*: a center of gravity (or first moment) of a 3-D curve segment is defined as the average of coordinates of the curve points; a 3-D convex hull [Pre85] of centers of gravity of a wing is the smallest convex volume containing centers of gravity of all possible 3-D curve segments of the 2-D wing contour.

4) *Ratio of 2-D minimum to maximum distances between the convex hulls of centers of gravity*: similar to Item 3 but convex hulls are defined in the 2-D space and using the ratio of distances instead of distances.

5) *Parallelism*: the separation and overlapping between the angle spans of two 2-D wing contours. Parallelism is computed using the following equations.

$$MAX = \max \{SP^1_{\max} - SP^2_{\min} \ , \ SP^1_{\min} - SP^2_{\max}\}$$

$$MIN = \min \{SP^1_{\max} - SP^2_{\min} \ , \ SP^1_{\min} - SP^2_{\max}\}$$

where $SP_{\max}$ and $SP_{\min}$ are maximum and minimum tangent angles of contour points with respect to a predefined coordinate system. Therefore, a parallelism measurement of a pair of sensed wings should be within the range of its corresponding model pair.

Note that the proposed sets of wing constraints may have redundancy. For example, in the set of unary constraints, Item 4 is in some sense equivalent to Item 7. An approach to studying the power of constraints will be presented in Chapter 6. In practice, a subset of the aforementioned constraints will be adequate. One should also provide tolerances of parameters and use a penalty strategy to get rid of computational instability.

2.4 Composite Wings

A fundamental objective for forming a composite wing is to look for simple wings which come from a single physical object. Typically, composite wings should possess more potential utility for object recognition than simple wings. Our major purpose is to capture salient component structure of objects while keeping the probability of false grouping very low. To this end, criteria for composite wing grouping must be defined. Feasible criteria include:

1) proximity - neighborhood, spatial relationships.

2) similarity - characteristic properties.

3) collinearity and curvilinearity - orientation continuity.

4) connectivity - reachability, through contours or surfaces.

5) cotermination - junctions.

6) symmetry, and

7) parallelism.

Clearly, the above grouping criteria are similar to those of perceptual organization [Zuc85, Vis85, Low85,87, Tuc86]. A prominent difference between this research and the previous work is the different tokens employed in perceptual organization. Dot patterns [Zuc79, Vis85, Tuc86] or line segments [Low85,87] were often used. Wings serving as tokens have different dimensionality from those of dots and lines. The search for grouping criteria suitable for wing primitives will form an extension of current wing representation theory. Details of definitions and functionals of the aforementioned criteria can be found in the works of Zucker [Zuc85], Lowe [Low85], Tuceryan [Tuc86], Biederman [Bie87] and Malik [Mal87].

A **composite wing** *is composed of a small set of simple wings which are common to a single physical object and are subject to some non-accidental relationship.*

In order to match the objective that simple wings should belong to a single object, the above criteria haven't provided sufficient conditions to follow because there are often competition among criteria [Zuc85, Tuc86]. To alleviate competition, rules for combining wings in terms of physical principles, heuristics and domain assumptions need to be constructed. Once this job has been achieved, category and structure relationships for composite wings can be introduced. Wing representation theory should go further to address the formation of object models in terms of composite wings and their structures. This is also of importance for object separation (high-level segmentation) when dealing with real images with multiple objects.

2.5 Illustrating the Power of Wings in Recognition

Figure 2.4(a), (b) and (c) show edge maps labeled using Malik's extension to the Huffman-Clowes label set [Mal87] for a bowl, a half grapefruit and a clam. Note that junctions at A, B, A" and B" are not included in Malik's junction catalogue because the bowl and clam are not in his object set. It is clear that the 3-D shapes of the objects in the figure can be precisely interpreted through the line labelings. However, since the labeling procedure always starts with a set of junctions and reaches consistency of labeling through hierarchical determination (filtering, backtracking, or derivation of reciprocal figures in gradient space), the connectivity between and within lines is critical for information propagation.

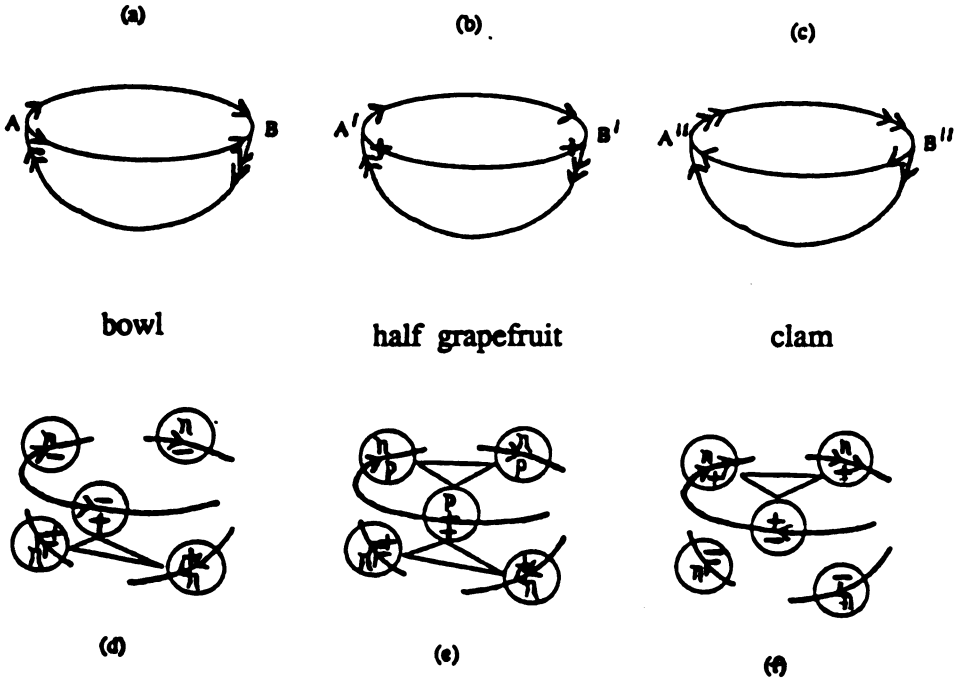


Figure 2.4. Line drawings for a bowl (a), a half grapefruit (b) and a clam (c) using Malik's labels. (d), (e) and (f) show wing features labeled along degraded contour segments for the objects in (a), (b) and (c) respectively.

Figure 2.4(d), (e) and (f) show incomplete edge maps for the same set of objects. To deal with such cases, techniques of *Gestalt based grouping* or *perceptual organization* may provide a solution. However, without appealing to support reasoning, it will be necessary to rely on a premise of non-accidentalness. This greatly reduces the generality of the techniques. To compensate for this, contextual surface information is included for those incomplete contours and results in object wings. As shown in the figure, even under degraded conditions, objects can be easily distinguished based on labeled wings.

We claim that many object wings can be reliably sensed from image data, whereas perfect curves and surfaces cannot be. Were a range image available, the object wings would be readily obtainable. In addition, as indicated in Figure 2.4(d), (e) and (f), surface data can even provide links for contours. This not only removes labeling ambiguity but can form high level composite features, which are critical cues for indexing into a database. The junctions in the figure have the same 2-D shape as the "three tangent" of Malik [Mal87]. While there are $6^3 \times 3^2$ contour/surface label combinations according to our simple labeling scheme, only a handful of them are realizable as real composite wings. Although we haven't sought a formal theory of the structure of the objects we wish to recognize, we do know that, as in classical line drawing analysis, correct detection of a junction will place strong constraints on the object surfaces creating it. Such constraints supported by the capability to sense surfaces is of great benefit to segmentation and recognition procedures.

2.6 Reconstructing Line Drawings from Wing Representations

In the last section, we demonstrated the power of object wings in identifying objects. In this section, we consider the adequacy of wing representation for polyhedral scenes. To this end, we demonstrate that if a wing representation has been derived from the image of a polyhedral scene, the spatial structure of the scene could be recovered from the wing representation. In other words, given the wing representation of a scene containing polyhedral objects, reconstruct all visible parts of objects based on the given wing representation - that is the coordinates of all visible vertices, the equations of all visible faces, and the completely labeled line drawing of the scene.

Here our purpose is not to interpret the 3-D meaning of a 2-D line drawing, but rather to construct the 2-D line drawing and its 3-D interpretation from appropriate fragments of the 3-D scene. It is fairly natural for us to start with the surface equations before reconstructing line drawings. This is because the surface information available in object wings is enough to compute equations of polyhedral object surfaces which are part of the wings. In the following subsections, we study a theoretical approach to interpreting polyhedral scenes based on given wing representations. The details of this study such as assumptions, the definition of object domain, non-accidental scenes, general viewpoints, reconstruction rules and several terms used in the following can be referred to Appendix G.

2.6.1 Reconstruction Algorithm

An associated algorithm characterized by the set of reconstruction rules is presented here. The input data include the range image and its associated wing representation of a polyhedral scene. The output result includes a labeled line drawing, the equations of visible faces and the coordinates of visible vertices. The basic idea of this algorithm is first to recover the line drawings of individual visible object faces; the entire scene present in the image is then reconstructed by integrating the recovered faces.

```
(1) Compute the equations of planes containing object faces using the wing segments in the given wing
        representation;
(2) Cluster wing segments into wing groups according to their associated plane equations; (Rule 1);
(3) For each wing group
        (3.1) generate a line for each wing in the wing group;
        (3.2) compute the intersections among the generated lines;
        (3.3) remove line segments with their endpoints located on the picture frame (Rule 2);
        (3.4) preserve line segments which overlap a wing (Rule 3);
        (3.5) remove any line segment unless points on exactly one side of the line segment are
                located on the given plane (Rule 4);
        (3.6) delete intersections which are not shared by exactly two noncollinear line segments (Rule 5);
        (3.7) record the 3-D coordinates corresponding to the remaining intersections by intersecting
```

line equations;

(3.8) label line segments with the wing types which appear on the line segments (Rule 6);

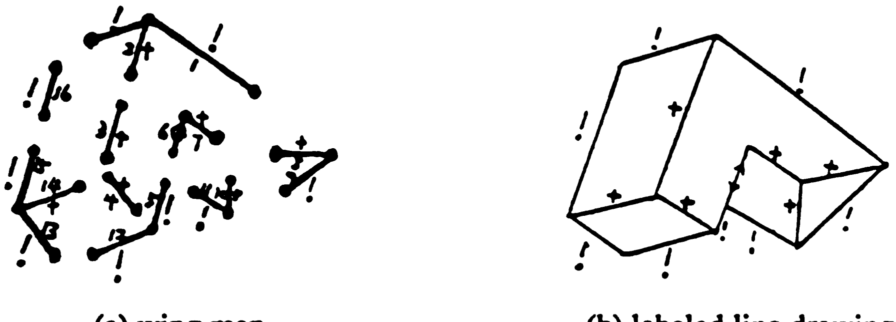
(4) Integrate reconstructed line drawings of individual object faces in order to discover *T* junctions;

(5) If a line segment in the integrated line drawing possesses a compound type, the components of the compound type are separated and are assigned to distinct parts of the edge according to the discovered *T* junctions and the original wing map (Rule 7).

(6) Output the resulting line drawing with edge labels, the plane equations corresponding to object faces, and the coordinates of intersections corresponding to 3-D vertices and 2-D *T* junctions.

2.6.3 Examples

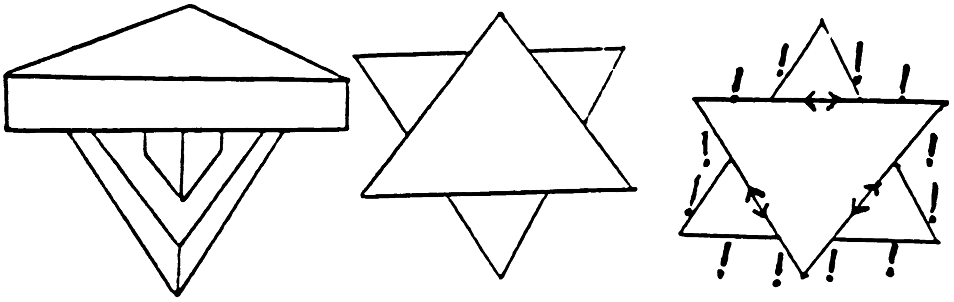
Let us look at the example in Figure 2.5(a). Grouping wing segments based on computed plane equations, we obtain five wing groups: $g_1 = \{1, 2, 3, 4, 5, 6, 7, 8\}$, $g_2 = \{2, 3, 14, 15, 16, 17\}$, $g_3 = \{4, 12, 13, 14\}$, $g_4 = \{6, 7, 10, 11\}$ and $g_5 = \{8, 9, 10\}$. Each group corresponds to an object face in this particular example. The line drawings of individual faces are first recovered based on the information provided by wing groups and the original range image. Recovered faces are then integrated and edges are relabeled by referring to the discovered *T* junctions and the original wing map. The reconstructed line drawing of the object is shown in Figure 2.5(b). Other special examples are shown in Figures 2.6 and 2.7, where the accidental view of a polyhedron and the object with a hole and multi-face vertices can all be reconstructed by the algorithm.



(a) wing map

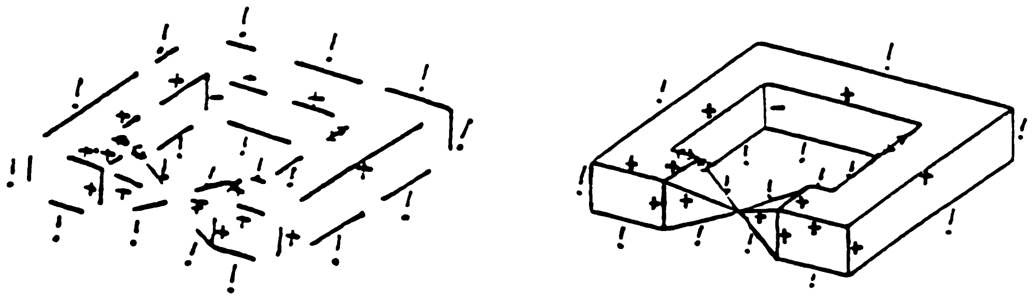
(b) labeled line drawing

Figure 2.5. An example demonstrates the reconstruction algorithm.



(a) (b) (c)

Figure 2.6. (a) A polyhedral object, (b) an accidental view of the object, and (c) the reconstructed line drawing.



(a) wing map (b) labeled line drawing

Figure 2.7. An object with a hole and multiple-face vertices.

2.6.4 Comments

Many previous line-drawing researchers started with a perfect line drawing and attempted to achieve an interpretation of the line drawing in terms of 3-D spatial structures. Unfortunately, this usually leads to a set of possibilities. Moreover, if the initial line drawings are imperfect, the problem becomes extremely difficult. In order to solve the problems of multiple interpretations and imperfections, assumptions and additional information were employed. Although parts of the problems have been successfully solved by introducing extra knowledge, the overall schemes are not efficient. The reason is that most researchers have separated their schemes into two major modules: reconstruction module and interpretation module. Additional information is only used in the interpretation module to reduce ambiguity rather than be used in the reconstruction module. In fact, available information should be used everywhere in the whole scheme as long as the information is helpful at those places.

In this research, we used range values as additional information in both reconstruction and interpretation modules. This not only greatly simplifies the design of procedures but also increases the efficiency of processing. In addition, instead of following the conventional sequence, we separate the modules into several stages and rearrange the stages in a hierarchical manner. For instance, in our algorithm individual faces are first reconstructed and interpreted, and then the entire scene is considered. One apparent advantage of this hierarchical rearrangement is that it increases the flexibility of the algorithm in dealing with a wide variety of object classes.

Another important aspect of this research is the use of wing representation as input data to the reconstruction algorithm. In contrast to perfect line drawings, wing representations are assumed to be imperfect. On the other hand, wing representations contain the information of range values and thus surface normals, which is not available in line drawings. In practice, wing representations are more realistic than line drawings. The additional information not only considerably compensates for certain deficiency of wing representation in interpreting scenes but also leads the reconstruction procedure to a unique result. The judgement of correctness of a labeled line drawing is also greatly simplified due to this additional information.

It is our conjecture that after the wings are extracted, the line drawing can be constructed without further access to the underlying range image. The proposed algorithm applies the first three rules used previously. It then departs into a state-space search for the polygonal circuits that form the faces in the given plane. At each step of the search one line segment is added to the line drawing and all constraints checked. It is easy to see that the search is finite although exponential. The key point to prove is that there is a unique set of circuits satisfying all the constraints that is the goal state of the search. While such an algorithm has less practical value than the one detailed here, it is of theoretical interest and will be pursued in future work.

Currently, we only consider polyhedral scenes. If a scene contains only one object, the proposed algorithm can handle more general objects than those defined in the current object domain. We need to study how to extend this work to include scenes with curved objects.

2.7 Summary

Starting from a general definition of wing primitives, we proceeded to construct a complete set of primitives. Then we developed the concepts of legal, collapsed and augmented sets of wings. By defining the types for contour and region, we were led to a set of 34 simple wings for practical implementation. Their properties are investigated by studying the structural constraints and the extraction rule of object wings. This set of primitives was then clustered to form composite wings under non-accidental relationships. Both simple and composite wings will enable us to explore the machinery of wing representation theory.

The basic idea of the feature proposal was originally motivated by the difficulties commonly encountered by current approaches to feature extraction and defect recovery. The notion of wing primitives has stemmed from a need to restore detected features by integrating available contextual shape information. Via integration of information, both powers in discriminating objects (Section 2.5) and in interpretating scenes (Section 2.6) can be greatly increased.

So far we haven't yet answered two important empirical questions: how to detect object wings in images, and how to construct internal representations of physical objects in terms of wing primitives. A computational framework for dealing with these two questions is given in the next chapter. The associated techniques of implementation are presented in Chapter 4.

Chapter 3

Wing Representation: Computational Framework

Wing representation theory is a computational theory concerning the representation problem characterized by a specific set of wing primitives. Computational theories in computer vision are somewhat unlike the usual mathematical theories. The latter generally involve substantial equations, derivations, theorems and proofs, while computational theories may instead include a set of physical principles, inference rules, algorithms, or simply descriptions. A computational framework in which associated techniques are developed is a realization of an abstract computational theory. In this chapter, a framework for constructing wing representations of real world objects is presented.

The last chapter has implied that wing primitives are not all view-invariant. The representations constructed in terms of wing features, called *wing representations*, will be viewpoint dependent and be composed of a set of views. Therefore, wing representations are a kind of multiple-view representation whose data structure will be referred to as the *viewsphere* throughout this thesis. Major steps for construction of wing representations include geometric modeling, model augmentation, view generation and organization.

Following Marr's thesis [Mar82], the formation of any computational theory should include answers to three questions:

What is the goal of the theory?

How can it be implemented?

Why is it appropriate?

In the last chapter, the primary objective and appropriateness of wing representation theory have been briefly addressed. In this chapter, after a description of secondary goals, we discuss how to construct wing representations for physical objects. The answer to the question of why wing representations are adequate will be scrutinized in later chapters where the implementation and testing are presented.

3.1 What is the Goal of Wing Representation Theory?

According to Shepard, Metzler and Cooper [She71,84, Coo75,84], mental operations are analogous to physical processes by which internal models are aligned with objects displayed in images through a series of transformations. We refer to such transformations as *model transformations* because the models are rotated and translated in order to align them with objects. Looking at the problem from another perspective, we envision that the mind attempts to discover a view of the model which is consistent with a retinal image produced by an object. This behavior (looking for instances of models to coincide with images) is clearly different from the previous one in that the models remain stationary while the mind moves freely. It is equivalent to switching a reference frame from the mental frame to the model frame. We refer to such operations as *mental transformations*. Undoubtedly, both transformations, although different in manner, would have the same influence on perception. However, when they are implemented in machine systems, both the representation schemes and the recognition processes would be different.

In the model transformation paradigm, each object model can be represented by a set of object features together with their structural relations. Those features and relations are described with respect to a pre-defined model frame. It is, therefore, an *object-centered* representation. There are a variety of choices for selecting features and relations, and also a myriad of ways to organize them into models. A critical point is that both features and relations should be view-invariant. Generally speaking, for the paradigm of model transformation, objects are modeled as individual entities within which view-invariant features are connected through a set of view-invariant relations. For a vision system with such representation schemes, techniques of either *3-D to 3-D* [Gri87, Sto88] or *3-D from 2-D* [Low85, Hut87, Lin88] are commonly used. Recognition tasks are then achieved by matching model features to sensed features.

For the mental transformation paradigm, each model is composed of a set of views, called *characteristic views*. Each view is then represented by a set of features and relations present in that view. A requirement for selection of features and relations for a view is that they must be invariant to its viewpoint. Recall that the features and relations used by the model transformation paradigm are invariant to all viewpoints. This is an important difference between these two paradigms. Thus, there is more flexibility within mental transformation paradigm to choose features and relations. A representation constructed by this method is clearly a *viewer-centered* scheme. For a vision system with this representation scheme, the recognition task is typically dominated by searching rather than matching.

In this research, the paradigm of mental transformation is adopted. It immediately gives rise to the problem of having infinite number of views for each object. How can all the views be derived? Of interest is a conclusion made by Cooper [Coo84]:

> *Although the identity of objects displayed in differing orientations can require imagining a rotation through intermediate orientations, it doesn't contend that the rotation is continuous in the strict mathematical sense, which requires that it sweep through all possible intermediate angles.*

Cooper's argument indicates that a finite set of views may be used to represent an object. Let us refer to such views as the *significant views* of an object. More questions immediately emerge. Given an object, how do we define the significant views for that object? Once the views have been defined, what kinds of features and relations can be used to represent views? We have answered this question by proposing a set of wing primitives and structural constraints. A final question is then how can we organize views in a way suitable for the recognition process. Answering these questions is the goal of wing representation theory.

3.2 How can Wing Representation Theory be Implemented?

The overall wing representation theory is depicted as a tree structure in Figure 3.1. The following discussion will start from the node of computational framework and proceed through the tree in a depth-first fashion.

To create the wing representation of a real world object, there are four intermediate tasks: construction of geometric models for objects, augmentation of geometric models, generation of view representation and view organization.

3.2.1 Geometric Modeling

The simplest way to acquire different views for an object may be to put the object in front of a sensor and take images of the object from various directions. This method measures the relative position between the sensor and the object for every view. Without precise tools for measuring distances and orientations, this will produce considerable errors and inconsistency between views with one another. A representation constructed from these error-contaminated views will increase the difficulty of the recognition process. Thus we propose another method that first generates a geometric model for the object of interest; then various views of the object can be easily obtained by repetitions of rotation and projection of the geometric model. A notable advantage of this method is that once a geometric model is constructed, view generation can be done automatically.

We define a geometric model of an object as a set of vertices and triangles formed via registering a set of different sensor views of the object. The triangles are used for convenience. The union of views should ideally cover the whole object surface. It is desirable that each model be a connected single entity composed of a set of vertices and triangles. It should be topologically equivalent to the object shape and geometrically approximate the object surface. Examples of four models and their physical counterparts are shown in Figure 3.2. The construction of a geometric model complying with the above criteria is nontrivial [Pot83, Hen85, Bha87, Wan87]. Models that will be used in

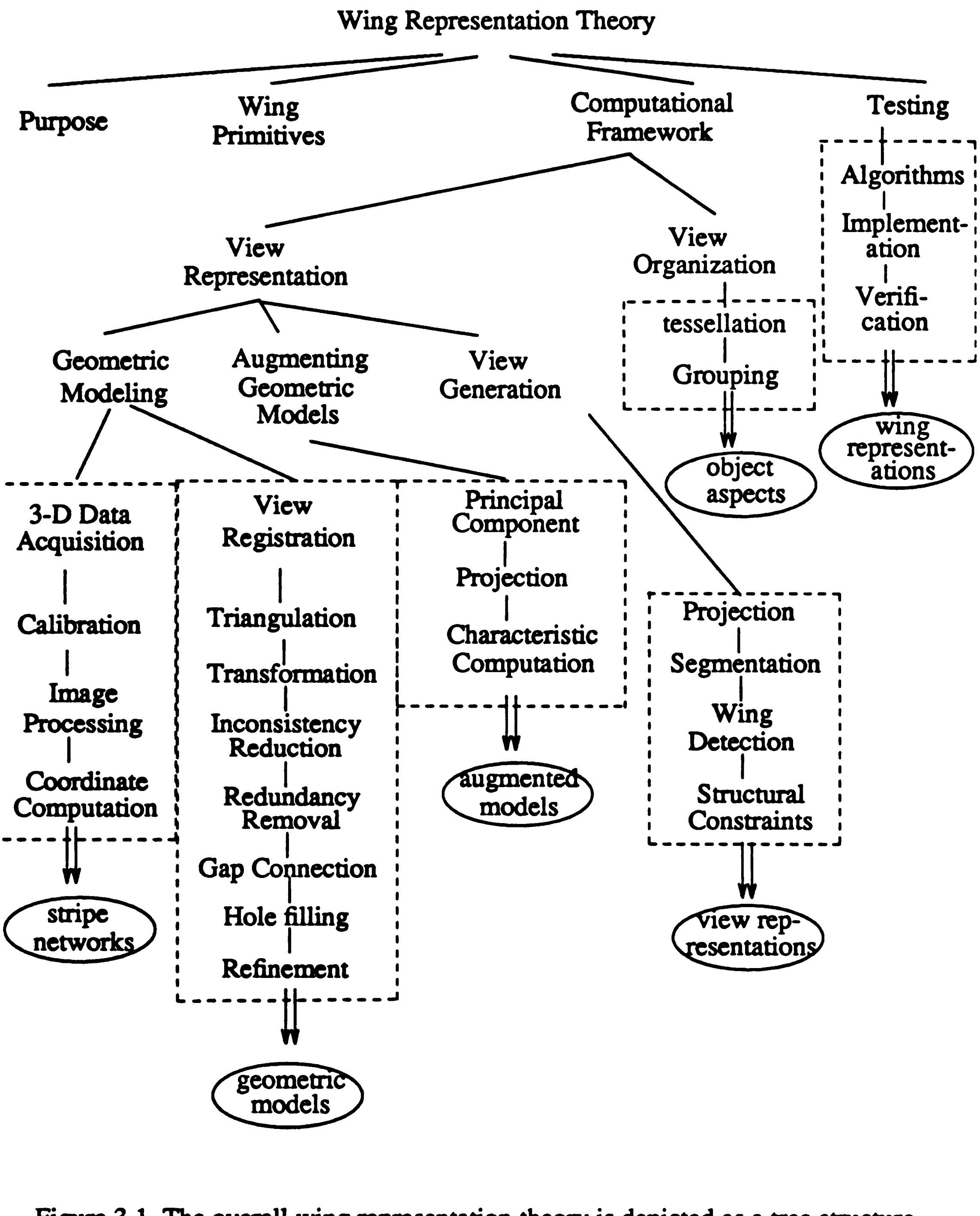


Figure 3.1. The overall wing representation theory is depicted as a tree structure.

some specific applications may fall short of these ambitions. For example, in our research of construction of wing representations, as long as models satisfy the criterion of geometric approximation, they are already enough for the purpose.

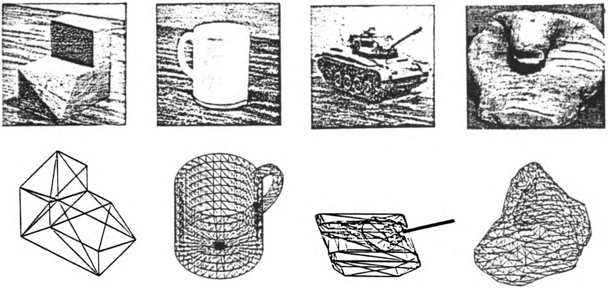


Figure 3.2. Examples of geometrical models for a wooden block, a coffee cup, a toy tank and a clay cobra head.

The construction of geometric models is commonly referred to as a task of *model-building*, which includes two major stages: 3-D data acquisition and view registration. During data acquisition, a set of object views are taken using a range sensor. Each view contains a set of distances to points sampled from the object surface. In general, for the purposes of representation and recognition, it is not necessary that sampling points should be dense in order to achieve models with high similarity to their corresponding objects.

One example of a sensing system that satisfies the above constraints is the active structured light sensor, which is inexpensive and produces precise measurements. A structured light sensing system is composed of a camera, a projector, and an image processor. Before using the system, both camera and projector must be calibrated. With this particular sensing system, each view contains a small set of light stripe networks rather than a set of individual points. Figure 3.3 displays an example of the stripe network extracted in a structured light image. A computational model for deriving coordinates of stripe networks has been formulated by Chen & Stockman [Che85].

After obtaining a set of views for an object, the next stage is to register this set of views to form a connected model. Although our current representation application requires only geometric similarity, the following description includes a more general method since these models may be used for other purposes in the future. One possible additional purpose is automatic manufacturing. Seven procedures are proposed. The first two of these procedures will be enough for our application. These procedures include triangulation, transformation, inconsistency reduction, redundancy removal, gap connection and refinement. The objectives and methods for these procedures are described below. Implementation details and results will be given in the next chapter.

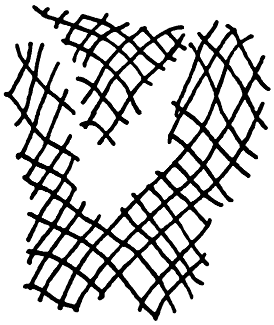


Figure 3.3. An example of the stripe network extracted in a stripe image taken by a structured light sensing system (MSU PRIP Lab).

1. **Triangulation:** As shown in Figure 3.3, stripe networks are formed by a set of distorted meshes. Within a network, there sometimes are holes, and boundaries are usually not connected. Therefore, there are two essential tasks. The first is to sequentially connect the terminations on network boundaries to form closed networks. A strategy previously developed by Hu and Stockman [Hu89], called the turn-right strategy, is employed for this purpose. The second purpose of this procedure is to triangulate networks by applying a 2-D hole filling algorithm to each mesh of the networks. The triangulation procedure is illustrated in Figure 3.4.

2. **Transformation:** Since the networks from each sensor view are derived with respect to a pre-defined global frame known to the viewer (the viewer-centered frame), networks

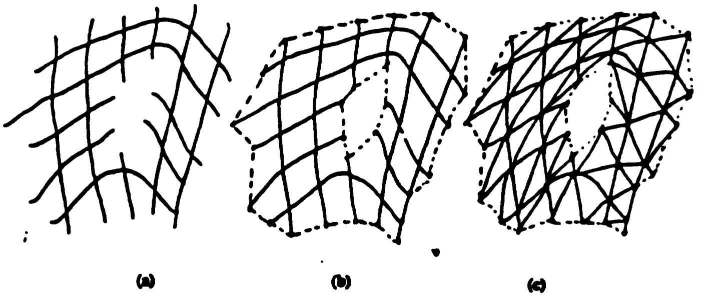


Figure 3.4. The intermediate results of the triangulation process.

have to be transformed to the model frame (the object-centered frame). Transformations are computed based on a set of control points measured *a priori*. Assume that P_i is the subset of control points associated with view *i*. Then at least three points of P_i should be noncollinear in order to obtain a unique transformation. Suppose also that there are N different combinations C = $\{C_j, j=1,N\}$ of three noncollinear points in P_i. For each combination $C_j = \{p_{j_1}, p_{j_2}, p_{j_3}\}$, a transformation $\mathbf{T}_j$ from the global frame to the model frame can be computed.

Let $\mathbf{r}_1$, $\mathbf{r}_2$, and $\mathbf{r}_3$ denote the coordinate vectors of the points of C_j with respect to the model coordinate system, and $\mathbf{r}_1'$, $\mathbf{r}_2'$, and $\mathbf{r}_3'$ be with respect to the global coordinate system. Define

$$\mathbf{v}_{g1} = \mathbf{r}_2' - \mathbf{r}_1', \quad \text{and} \quad \mathbf{v}_{o1} = \mathbf{r}_2 - \mathbf{r}_1.$$

From Equation (A.2.2) in Appendix A, the first rotation transformation is

$$\mathbf{R}_1 = \mathbf{R}(\mathbf{a}, \mathbf{v}_{g1}, \mathbf{v}_{o1}), \quad \textit{where} \quad \mathbf{a} = \frac{\mathbf{v}_{g1} \times \mathbf{v}_{o1}}{|\mathbf{v}_{g1} \times \mathbf{v}_{o1}|}.$$

Define

$$\mathbf{v}_{g2} = \mathbf{R}_1 (\mathbf{r}_3' - \mathbf{r}_1'), \quad \text{and} \quad \mathbf{v}_{o2} = \mathbf{r}_3 - \mathbf{r}_1.$$

From Equation (A.2.2) and (A.2.3) in Appendix A, the second rotation transformation is

$$\mathbf{R}_2 = (\mathbf{v}_{o1}, \mathbf{v}_{g2}, \mathbf{v}_{o2}).$$

The two coordinate systems are now parallel. What we need next is merely a translation T to bring them into a complete alignment.

$$\mathbf{T} = \frac{1}{3}\sum_{i=1}^{3}\mathbf{T}_i\,, \quad \textit{where} \quad \mathbf{T}_l = \mathbf{T}(\mathbf{v}_l), \;\; \mathbf{v}_l = \mathbf{r}_l - \mathbf{R}_2\,\mathbf{R}_1\,(\mathbf{r}_l').$$

Finally we get the transformation $\mathbf{H}_j$ for combination C_j,

$$\mathbf{H}_j = (\mathbf{T}\,\mathbf{R}_2\,\mathbf{R}_1)^{-1}.$$

The resulting transformation H for view i is obtained by averaging all $\mathbf{H}_j$'s,

$$\mathbf{H} = \frac{1}{N}\sum_{j=1}^{N}\mathbf{H}_j.$$

3. **Inconsistency Reduction:** Inconsistencies, which are undesirable displacements between networks, are usually observed after the frame transformations of networks. An illustrative example of displacement between two networks is shown in Figure 3.5. Inconsistency may result in either shape distortions or spurious boundaries on the final model surface. An optimization technique is proposed to reduce the inconsistencies among networks.

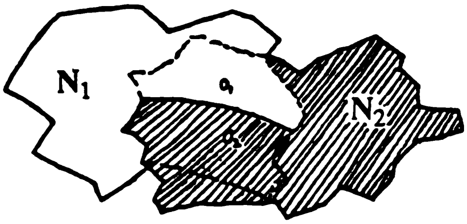


Figure 3.5. An inconsistency phenomenon between two networks.

Consider the two networks shown in Figure 3.5. Suppose there is a 4 by 4 transformation H, which transforms network N_2 so that the inconsistency between N_1 and N_2 is reduced. The transformation H is composed of a 3 by 3 rotation matrix R and a 3 by 1 translation vector t, so that a transformation possesses up to six degrees of freedom.

Let O_1 and O_2 denote respectively the overlap of networks N_1 and N_2. Each contains a set of triangles. The equations, $\mathbf{T} = \{T_i(\mathbf{n}_i, d_i),\ i=1, m_1\}$, of the planes containing the triangles of O_1 can be easily computed. Each plane equation is characterized by its surface normal, $\mathbf{n}_i$, and the perpendicular distance from the origin of the object-centered coordinate system to the plane. Let $\mathbf{V} = \{v_j, j=1,m_2\}$ be the set of vertices of $\mathbf{O}_2$. For each vertex v_j of $\mathbf{V}$, compute the minimum distance $d^j_{\min}$ from the vertex to $\mathbf{O}_1$. The minimum distance of a vertex is determined as follows. Let $\{d^j_i, i=1,\cdots,m_1\}$ be the set of minimum distances from v_j to triangles of O_1. Let

$$d^j = \min\{d^j_i, i=1,\cdots,m_1\}$$

and $T_{jk}(\mathbf{n}_{jk}, d_{jk})$ be the plane containing the triangle which results in d^j. Let

$$d^j_{\min} = \mathbf{n}_{jk} v_j - d_{jk}. \tag{3.1}$$

However, after transforming network $\mathbf{N}_2$, the minimum distance becomes $d^{j'}_{\min}$ where

$$d^{j'}_{\min} = (\mathbf{R}\,\mathbf{n}_{jk})\, v_j - ((\mathbf{R}\,\mathbf{n}_{jk})\mathbf{t} + d_{jk}). \tag{3.2}$$

Now we can formulate the optimization problem: determine a transformation that minimizes the following sum of squared distances.

$$e = \sum_{j=1}^{m_2} d^{j'2}_{\min} = \sum_{j=1}^{m_2} ((\mathbf{R}\,\mathbf{n}_{jk})\, v_j - (\mathbf{R}\,\mathbf{n}_{jk})\,\mathbf{t} - d_{jk})^2, \tag{3.3}$$

To solve Equation (3.3) [Gun87], one can use numerical methods such as the Lagrange formula or the Newton-Raphson approach.

4. **Redundancy Removal**: After transformation and inconsistency reduction, the surface of the object which is located at the object-centered frame should be completely covered by the resulting networks. There are, however, overlaps among them. To achieve a single connected model, the redundant triangles must first be removed. The needed computational techniques of this procedure have been developed by Chen & Stockman [Che85]. An example of the procedure for redundancy removal is shown in Figure 3.6, where the dark area which originally belongs to network N_2 has been removed and results in two smaller networks C_1 and C_2.

5. **Gap Connection**: The above procedures will produce a set of nonoverlapping networks. Any gaps in this set must be bridged to completely connect the networks. There are two problems to be solved in this step. The first is to determine the sets of vertices which belong to different networks but are close enough to be connected. The second

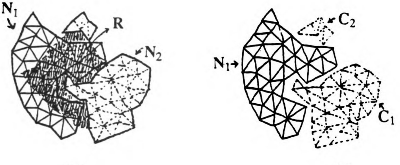


Figure 3.6. An example for demonstrating the procedure of redundancy removal.

issue is then how to connect them properly. These issues are addressed in the next chapter. The output of the gap connection procedure for the example in Figure 3.6 is displayed in Figure 3.7.

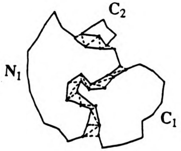


Figure 3.7. The Result of gap connection for the example of Figure 3.6.

6. Hole Filling: Holes may occur in networks for two reasons. First, holes may appear in a network after triangulation due to the absence of stripes in some areas as shown in Figure 3.4(c). Secondly, the procedure of gap connection may also leave holes. For example, using our method, the gap between C_1 and C_2 in Figure 3.7 will eventually form a hole in the resulting network after they are connected to their neighborhood. A procedure similar to the 2-D hole filling algorithm in the triangulation procedure was developed to solve this problem. It is called the 3-D hole filling algorithm to distinguish it from the 2-D hole filling algorithm.

7. **Refinement**: One of the major purposes of refinement is to smooth undesirable bumps on the surfaces of constructed models, which were primarily formed during the connection of networks. Methods of averaging the nearest neighbors of each vertex can be used for this purpose. Also, due to the computational errors, it is possible that the resulting shapes of models may be distorted. This defect can be attenuated by manual editing using a CAD system.

3.2.2 Augmenting Geometric Models

Once a geometric model has been constructed, it has a set of vertices and triangles. One can augment it by adding surface curvatures, color, intensity or other features. For our purpose, each model vertex has surface curvature, surface type, surface roughness, fit error, edge magnitude, and the angle between principal vectors as additional features. These properties are collectively called the characteristics of the model surface. Their mathematical definitions will be given later in this section.

Computation of surface characteristics at each vertex is performed by first looking for the plane tangent to the model surface at that vertex. Then, the object model is projected onto this plane. The resulting projection is a range map. There are several methods to compute desired surface characteristics directly from this range map. One can first fit it with a smooth patch such as a cubic spline, a Coons patch or a quadratic surface. All characteristics can then be computed analytically in terms of the surface function. We present surface equations of B-splines and quadratics for this purpose.

B-splines: A spline surface patch can be defined as a combination of two spline curves. The definition of spline curves has been given in Chapter 2.

$$\mathbf{r}(t,\, s) \;=\; [\, t^3 \;\; t^2 \;\; t \;\; 1\,]\, \mathbf{A} \begin{bmatrix} P_{00} & P_{10} & P_{20} & P_{30} \\ P_{01} & P_{11} & P_{21} & P_{31} \\ P_{02} & P_{12} & P_{22} & P_{32} \\ P_{03} & P_{13} & P_{23} & P_{33} \end{bmatrix} \mathbf{A}^t \begin{bmatrix} s^3 \\ s^2 \\ s \\ 1 \end{bmatrix},$$

where P_{ij}'s are control points, t, s are parameters of the spline patch, and

$$\mathbf{A} \;=\; \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 0 & 4 \\ -3 & 3 & 3 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}.$$

Quadratics: A quadratic surface can be defined in terms of three discrete orthogonal polynomials [Bes86]:

$$\phi_0(t) = 1\,,\;\; \phi_1(t) = t\,,\;\; \phi_2(t) = (t^2 - \frac{A\,(A+1)}{3}),$$

where $A = \dfrac{(W-1)}{2}$, W is the window size, and parameter t,

$$t \in \{\, \frac{-(W-1)}{2},\, \cdots,\, -1,\, 0,\, 1,\, \cdots,\, \frac{(W-1)}{2} \,\}.$$

The normalized versions $a\,(t)$ of functions $\phi(t)$ are:

$$a_0(t) = \frac{1}{W}\,,\;\; a_1(t) = \frac{3}{A\,(A+1)(2A+1)}\,t\,,\;\; a_2(t) = \frac{1}{P\,(A)}(t^2 - \frac{A\,(A+1)}{3}),$$

where

$$P\,(A) \;=\; \frac{8}{45}A^5 + \frac{4}{9}A^4 + \frac{2}{9}A^3 - \frac{1}{9}A^2 - \frac{1}{15}A.$$

Then the surface function becomes

$$r(t,\, s) \;=\; \sum_{i,j=0}^{2} c_{ij}\phi_i(t)\phi_j(s),$$

where

$$c_{ij} \;=\; \sum_{t,s} f\,(t,s) a_i(t) a_j(s).$$

The surface characteristics are then defined as follows.

Fit error: the mean squared error between the fit surface and the measured surface.

Edge magnitude: $e = (r_t^2 + r_s^2)^{1/2}$, where r_t and r_s are the first derivatives of surface function $r(t,s)$.

Roughness of surface: $R = r_{tt}^2 + 2r_{ts}^2 + r_{ss}^2$, where r_{tt}, r_{ts} and r_{ss} are the second derivatives of surface function $r(t,s)$.

Gaussian curvature: $\mathrm{K} = \dfrac{LN - M^2}{EG - F^2}$, where

$$E = r_t \cdot r_t, \quad F = r_t \cdot r_s, \quad G = r_s \cdot r_s,$$

$$L = r_{tt} \cdot \mathrm{N}, \quad M = r_{ts} \cdot \mathrm{N}, \quad N = r_{ss} \cdot \mathrm{N}, \quad \text{and} \quad \mathrm{N} = \frac{r_t \times r_s}{|r_t \times r_s|}.$$

Mean curvature: $\mathrm{H} = \dfrac{EN + GL - 2FM}{2(EG - F^2)}$.

Surface type: determined from the Gaussian and mean curvatures.

Angle between principal vectors: $\mathrm{A} = \tan^{-1}(\dfrac{-B + \sqrt{B^2 - AC}}{C})$, where

$$A = EM - FL, \quad 2B = EN - GL, \quad C = FN - GM.$$

3.2.3 View Generation

By projecting the augmented model of an object onto a synthetic view plane, a range image and a number of registered characteristic maps and a preliminary segmentation of the specific view can easily be generated. The wing representation of this view is then derived via a top-down procedure, which is guided by the object model. The wing detection algorithm proceeds as follows.

First, contour pixels are determined based on an integration of information provided by characteristic maps. For example, jump contours can be determined using range data, and crease edges can be determined using combined information of surface roughness, edge magnitude, fit error and zero crossings of curvature. The final contour map is the integration of those intermediate results. A thinning algorithm is then applied to the contour map. The resulting contours may be shifted from their actual positions due to the thining procedure. Such shifts will not influence the extraction of wing features but will affect the computation of wing constraints. This is one possible source of errors in the final result. The surface type of connected regions from the preliminary segmentation, originally determined from signs of curvatures, are iteratively smoothed. The smoothing procedure replaces the region type of a pixel by the majority of its first order neighbors using a 3 by 3 window. This procedure refers to the contour map in order to prevent counting neighbors located on the other side of a contour. The procedure will stop when no changes of surface type can be made or a maximum number of iterations is reached. Based on the resulting contour and surface maps, wing primitives can be extracted through a labeling procedure, which is an implementation of the **extraction rule** defined in Section 2.3.1. Once object wings are determined, their structural constraints can be computed following the definitions described in Section 2.3.2. The object view is then represented in terms of wing features and structural constraints.

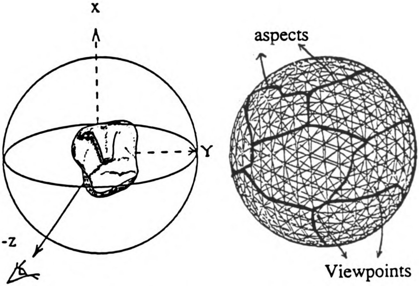


Figure 3.8. A unit viewsphere whose surface forms a 2-D manifold of visual directions. In practice, it is partitioned into a finite set of uniformly distributed viewpoints.

3.2.4 View Organization

Refer to Figure 3.8. Since a 3-D object has an infinite number of 2-D views, a multiple-view configuration characterized by a unit sphere, called the *viewsphere*, is introduced. The viewsphere is composed of a set of *viewpoints* distributed on the surface of the sphere, which form a 2-D manifold of visual directions. It is similar to view characteristics [Dud77], property spheres [Fek83], and locus images [Goa83]. This notion is a natural creation of the mental transformation paradigm -- object recognition through a search of model views. Each object is initially represented by a set of object views created according to viewpoints of the viewsphere. Each view is generated by projecting the geometric model of the object along the direction from the center of the viewsphere to a viewpoint. The representation of this view in terms of object wings and wing constraints is then constructed by the procedure presented in the last section.

There are a variety of approaches to determine the viewpoints of an object. Some are object-independent [Bas88], and some are object-dependent [Kor87]. In the object-independent approach, the viewsphere is tessellated into fixed number of viewpoints. As a result, objects are represented by a fixed set of views. Although this method is quite straightforward, a large number of viewpoints is required to represent complex objects. This will make the representations of simple objects unnecessarily complex. For the object-dependent approach each object has to be processed separately. The determination of optimal viewpoints for an object is not only a formidable task but also inefficient when dealing with a large number of objects. To compromise, we first tessellate a continuous sphere surface into a set of uniform cells, called *viewpoints*. This step is similar to the object-independent approach. Then, viewpoints sharing some object properties are gather into groups called *aspects* [Koe76,79, Ike88]. The result of this step will be analogous to using an object-dependent method. In Appendix B, we present approaches to construct discrete viewspheres with 320 and 240 viewpoints respectively. Other viewspheres with 20, 80 or 140 viewpoins are intermediate results when generating the above viewspheres.

In the last section, we presented a procedure to generate a wing representation for an arbitrary object view. Following that procedure, for each object, a set of view representations with respect to the viewpoints of the viewsphere can be created. Since every view representation contains a set of wings and structural constraints, we can cluster viewpoints into *aspects* [Koe76,79,84,87, Pla87, Gig88, Ike88] according to the topological structure of wing sets detected in viewpoints. The boundary between any two aspects corresponds to a *visual event* which signifies a change of tolpological structure of wing sets across the boundary. A visual event is defined as an appearance or a disappearance of any wing primitives. This means that views within an aspect should possess the same set of wing primitives and be topologically equivalent. As a consequence, an object is finally specified by a set of aspects.

In a polyhedral world, since the faces of objects are all planar, there is no difference between an object wing and a labeled object edge. As a consequence, it is natural for us to treat object wings as object edges in the polyhedral world. Previous research in constructing aspect graphs for polyhedral objects have used either faces [Pla87], edges [Gig88] or k-faces (k=0,1,2,3) [Bow89] as object features. Boundaries of aspect are then determined based on the changes of topological structure of features. In the Plantinga & Dyer work [Pla87], a change of topological structure is defined as a change of visibility of features. However, in the Gigus *et al* work [Gig88], changes of topological structure are defined as distances between junctions and line segments going down to zero. Although wing features are similar to object edges, instead of following the Gigus *et al* approach, we will use the changes of visibility of object wings to define the changes of topological structure.

For a highly symmetrical object, without considering accidental views, the number of aspects is small. Figure 3.9 shows the three generic aspects of the bowl, which are labeled as 1, 2 and 3. To prove this, let us divide the 3-D space into 8 subspaces, which are labeled from Octant I to VIII as shown in Figure 3.10. Let the bowl be located with its top plane aligned with the $x-y$ plane of the spatial frame and the circle is centered at the origin of the space. Suppose a viewer stands in Octant I. There are only two distinct aspects which can be seen by the viewer, i.e. Aspect 1 and 2 of Figure 3.9. The viewpoints of these two distinct aspects are separated by planes tangent to the rim of the bowl. The same set of aspects also occurs in Octant II, III and IV. However, if the viewer stands in either of the remaining octants (V,VI,VII,and VIII). There is only one distinct aspect, which is Aspect 3 in Figure 3.9. There is no way to see the concave surface of the bowl from the lower part of the space until the $x-y$ plane is crossed. Through this exhaustive examination, three distinct aspects are obtained for the bowl. We can even estimate the area on the viewsphere for each distinct aspect. A more detailed study of object aspects will be discussed in Chapter 5.

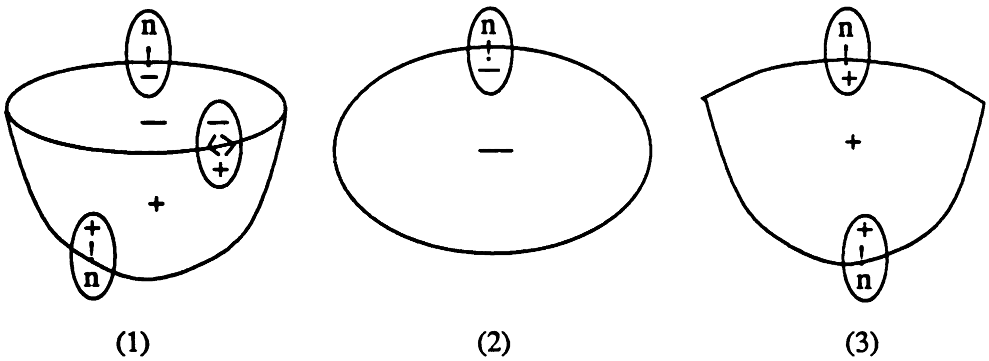


Figure 3.9. All generic aspects of a bowl.

3.3 Summary

This chapter presented a computational framework for constructing wing representations of real world objects. Implementation details will be presented in the next chapter. Experiments and testing of wing representations will be presented in Chapter 5.

Wing representation theory is defined in terms of a set of wing primitives. In order to construct the wing representation of an object, the geometric model of the object is first constructed. There are two major stages involved in the task of geometric modeling: 3-D data acquisition and view registration. Data acquisition includes three steps: calibration, image processing and coordinate computation. In view registration, there are seven steps: triangulation, transformation, inconsistency reduction, redundancy removal, gap connection, hole filling, and refinement.

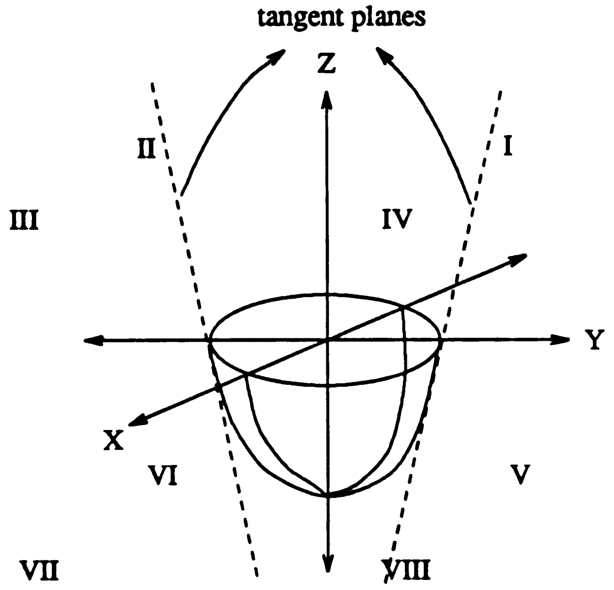


Figure 3.10. The octants of a 3-D space.

Through the geometric model of an object, different views of the object can be readily generated. Techniques of wing detection and computation of structural constraints are then applied to each view to obtain a view representation in terms of wing primitives and their relations. Views are then grouped into aspects according to their wing sets. An object is finally represented by a set of aspects.

Chapter 4

Implementation of Wing Representation Theory

In this chapter, algorithms for implementing wing representation theory are developed. The associated experiments start with physical objects and end with their wing representations. A clay cobra head (Figure 3.2) which has irregular shape will be used throughout this chapter as the primary example object. Later, a coffee cup (Figure 3.2) is used as another experimental object. Further experimental results will be presented in the following chapter.

To construct the wing representation for an object, a multiple-view scheme, called the *viewsphere*, is used. At the beginning, objects are presumed to be located at the center of the sphere. Each object is initially represented by a set of views, which are generated by projecting a pre-constructed geometric model of the object along the directions from the center of the sphere to a set of viewpoints distributed on the surface of the sphere. Each object view is in turn represented by a set of wings and their structural properties. Intermediate results and difficulties of the experimentation will be described. Through this implementation, weaknesses of the theory originally overlooked are discovered and discussed. Remedies will be the basis for future work.

4.1 Model-Building

In the experiment, a geometric model is constructed by integration of a set of sensor views taken by an active structured light sensing system. Views are then registered through a series of procedures to form a connected triangulated surface. Figure 4.1 shows a schematic diagram of the modeling process.

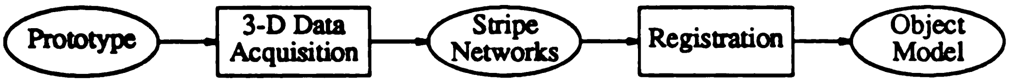


Figure 4.1. The schematic of the modeling procedure.

4.1.1 Sensing System

To acquire 3-D coordinates from object surfaces, an active structured light sensing system is used [Che85]. The configuration of this system is shown in Figure 4.2, which is composed of a camera, a projector and an image processor. A slide with a 10 × 10 square grid is used as the pattern mask and inserted in the front of the projector. It is best to arrange the orientation of the camera such that the optical axis of the camera intersects any light sheet at about 45 degrees in angle. The camera with 50mm focus length is located at about 175cm (the standoff) away from the origin of a global coordinate system. The global frame is fixed at the center of the light grid, with the x and y axes coinciding with the grid axes, and the z axis pointing upward. The projector is positioned above the workbench at about 145cm.

Before the sensing system can be used, both the camera and projector have to be calibrated. The experimental object (the cobra head) is then put in the FOV (Field of View) of the sensor, so that the light sheets created by the projector can be properly projected onto the object surface. Figure 4.3 shows four stripe images taken from scenes where the cobra head has been posed. To construct a complete model, every part of the object surface should be covered by at least one sensor view. Thresholding, thinning, clearing and clustering [Hu89] are applied to the images. Features of the light stripes can thus be extracted. The spatial coordinates of those features are readily computed by

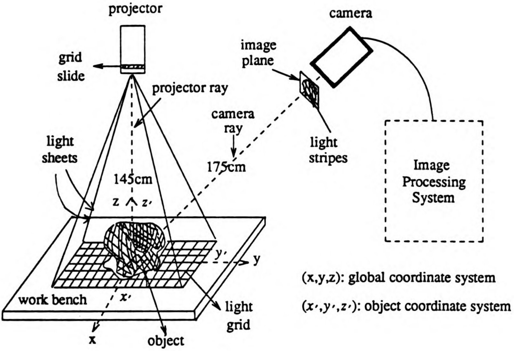


Figure 4.2 The configuration of an active structured light sensing system.

solving the consistent labeling problems and triangulation presented in Appendix A.

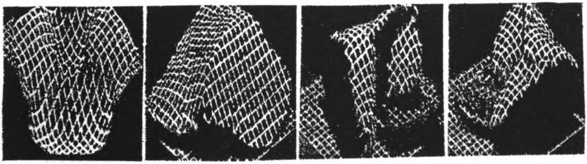


Figure 4.3 Stripe images of the cobra head with four different poses.

4.1.2 Registration

Figure 4.4 displays networks derived from four stripe images corresponding to four different poses of the cobra head. They will be used as the input to the registration process which integrates the networks into a connected model. Procedures in the registration process include triangulation, transformation, inconsistency reduction, redundancy removal, gap connection, hole filling and model refinement, and are depicted in Figure 4.5.

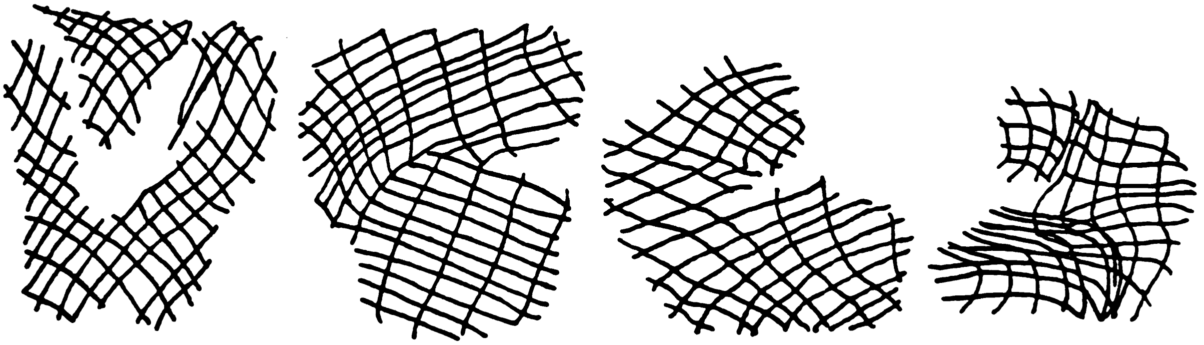


Figure 4.4 Networks derived from four stripe images corresponding to four different object poses (the cobra head) are to be used as the input to the following registration process.

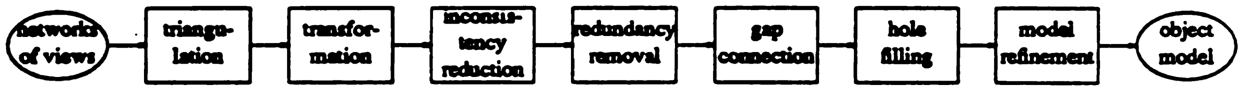


Figure 4.5. The diagram of the registration process.

4.1.2.1 Triangulation

To triangulate a network the terminations of the network, which are endpoints of the stripes, are located and connected to form a closed network. Then a *2-D hole filling* algorithm is applied to each grid cell. The hole filling algorithm proceeds as follows:

(1) Start with any vertex, say v_1, of a grid cell.

(2) If all vertices are collinear or the number of vertices equals two, stop.

(3) Select the next two adjacent vertices from v_1, say v_2 and v_3.

(4) If the vertices v_1, v_2 and v_3 are noncollinear, a triangle is constructed in terms of these three vertices. Otherwise start with the second vertex v_2 (set v_1 = v_2) and go to step 2.

(5) If the constructed triangle overlaps its neighbors, start with the second vertex ν_2 and go to step 2. Otherwise the triangle is preserved and ν_2 is deleted from the list of vertices. Then, start with ν_3 (set $\nu_1 = \nu_3$) and go to step 2.

A detailed algorithm appears below (Algorithm 1).

Algorithm 1: 2-D hole filling

Objective: Triangulates a 2-D polygon into triangles.

Input: A 2-D polygon P_1.

Output: A triangular polygon P_2 of the polygon P_1,

Procedure :

```
begin
  V = the vertex list of $P_1$;
  n = the number of vertices of V;
  v1 = V[1]; n1 = n; times = 0;
  while (n > 2)
    begin
      if (n <> n1)
        begin n1 = n; times = 0; end;
      else
        begin
          times = times + 1;
          if (times = n) return;
        end;
      v2 = the first adjacent vertex of v1;
      v3 = the second adjacent vertex of v1;
      if (v1,v2,v3 are collinear)
        begin v1 = v2; repeat; end;
      T = triangle(v1, v2, v3);
      if (T overlaps its neighbors)
        begin v1 = v2; repeat; end;
      add T to $P_2$;
      delete v2 from V;
      n = n - 1;
      v1 = v3;
    end;
end; (** end of procedure **)
```

An overall procedure for triangulation of a stripe network appears below as Algorithm 2. The result of the example in Figure 4.4 is shown in Figure 4.6.

Algorithm 2: Triangulation

Objective: Triangulates a connected network.

Input: A connected network N_1.

Output: A triangular network N_2,

Procedure :

```
begin
  look for terminations of network $N_1$ using turn-right strategy;
  connect terminations sequentially to form a closed network N;
  for (each polygon $P_i$ of the closed network N)
    begin
      apply 2-D hole filling algorithm to $P_i$;
      add the triangularized $P_i$ to $N_2$;
    end;
end; (** end of procedure **)
```

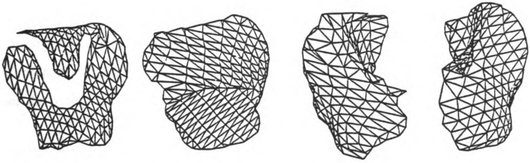


Figure 4.6. The triangulated networks derived from the example in Figure 4.4.

4.1.2.2 Transformation and Inconsistency Reduction

The networks in Figure 4.6 are given in terms of the global coordinate system. They have to be transformed to the object-centered coordinate system so that their union can cover the object surface properly. The transformation of a view is computed through a set of control points detected in that view. It is necessary for at least three points in the set to be noncollinear so that the transformation can be uniquely determined.

Control points should be well separated to reduce the effects of angular error because the computed transformation usually gives better results for the areas close to those points. It is best to mark control points at edges of an object so that they can be easily seen by the sensor from as many viewpoints as possible. If an object can be fixed on a regular external wire frame, then some feature points of the frame can be selected as control points. One could even stick hatpins into the object to provide control points.

A computational method of transformations based on a set of control points has been formulated in Chapter 3. Computed transformations are applied to their corresponding views. The resulting networks approximately cover the object surface. At this moment, there may be some positional shifts among transformed networks. Such shifts will be referred to as the *inconsistency phenomenon.* Serious shifts can result in shape distortion and spurious edges on the final model. Inconsistency can be improved using either CAD systems or a program to interactively adjust the networks.

A mathematical model based on Equation (3.3) can be used to automatically calculate transformations between two inconsistent networks. This model assumes that inconsistency among networks is initially small so that precise matching can be achieved by recursive positional refinements using optimization techniques. An algorithm based on the mathematical model is given as Algorithm 3 below. Figure 4.7 shows the networks after the procedures of transformation and inconsistency reduction.

Algorithm 3: Inconsistency Reduction

Objective: Reduces inconsistency between two networks.

Input: Networks N_1, N_2, and error tolerance ε.

Output: The rectified network N_2,

Procedure :

```
begin
  set e to any value larger than $\varepsilon$;
  set $e_o$ to any value larger than e;
  repeat
    begin
      $e_o$ = e;
      compute the overlaps $O_1$ and $O_1$ of networks $N_1$ and $N_1$;
      compute plane equations T of the triangles of $O_1$,
      $T = \{T_i(n_i, d_i), i=1, m_1\}$;
      let V be the set of vertices of $O_2$;
      compute optimal transformation H by solving Equation (3.3);
      $N_2 = H N_2$;
      compute e by substituting H into Equation (3.3);
    end;
  until ((e > $\varepsilon$) and (e < $e_o$))
end; (** end of procedure **)
```

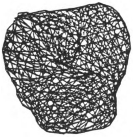


Figure 4.7. Networks after transformation and inconsistency reduction.

4.1.2.3 Redundancy Removal

After transformation, the networks should overlap one another. Referring to the example in Figure 3.6(a), an algorithm called *redundancy removal* is developed and specified in Algorithm 4. In the example, networks N_1 and N_2 partially overlap. The overlapping area of N_2 is called the *redundancy* **R** (hatched area) which is to be removed in order to obtain a larger connected network by connecting the nonoverlapping networks.

(1) For each triangle T_i^2 of network N_2, check whether there is a triangle T_j^1 in network N_1, that is close to T_i^2 and overlaps it.

(2) If such a T_j^1 exists, T_i^2 is a *redundant triangle* and is removed from N_2.

(3) Extract the neighboring triangles S_j^1 of T_j^1 on network N_1, which also overlap triangle T_i^2.

(4) The union of T_j^1 and S_j^1 is referred to as the *contribution set* of triangle T_i^2.

(5) The contribution set of whole redundancy **R** is the union of the contribution sets of triangles in **R**. This set will be used in the gap connection procedure.

In the above algorithm, to check the overlap between triangles, two constraints must hold. These two constraints stem from the assumption that if two triangles overlap, these two triangles should cover nearly the same part of the object surface. Therefore, their positions and orientations should be similar. We examine both the maximum and minimum distances between two triangles. The computation of position and orientation change between two triangles in 3-D space are addressed elsewhere [Che85]. The thresholds of the minimum distance and the angle subtended between two surface normals are 5 mm and 50 degrees respectively, which have been determined from an analysis of error in calibration and transformation to be presented in Section 4.1.3.2 and 4.1.3.3 respectively.

Algorithm 4: Redundancy Removal

Objective: Removes redundant triangles from the second network N_2, and looks for the contribution set of the redundancy.

Input: Networks N_1 and N_2.

Output: A set of surviving components of network N_2, and the contribution set S of the redundancy R.

Procedure :

```
begin
 for (each triangle $T_i^2$ of network $N_2$)
  for (each triangle $T_j^1$ of network $N_1$)
   begin
    if ($T_i^2$ and $T_j^1$ do not satisfy angle constraint) continue the inner for loop;
    if ($T_i^2$ and $T_j^1$ do not satisfy distance constraint) continue the inner for loop;
    if ($T_i^2$ and $T_j^1$ overlap)
     begin
      $T_i^2$ is a redundant triangle;
      delete it from network $N_2$;
      look for neighbors $S_j^1$ of $T_j^1$ in $N_1$;
      for (each triangle $T_k^1$ of $S_j^1$)
       if ($T_i^2$ and $T_k^1$ overlap)
        add $T_k^1$ to the contribution set $S_i^2$ of $T_i^2$;
      break loop j;
     end;
   end;
end; (** end of procedure **)
```

Figure 3.6(b) shows the result of the example in Figure 3.6(a) after the removal algorithm. In a similar vein, the final result for all networks can be achieved by applying the above procedure in parallel to all pairs of networks.

4.1.2.4 Gap Connection

Continuing the example of Figure 3.6, network N_1 remains unchanged. In N_2, removing the redundancy results in two disconnected components C_1 and C_2. Consider connecting N_1 and C_1 into a single network. Before describing the algorithm, let us first define the boundary elements of a network. A triangle is a *boundary triangle* of a network if one of its vertices has an empty sector (no triangle). An edge is a *boundary edge* if it is an edge of a boundary triangle and not shared by another triangle. Finally, a vertex is a *boundary vertex* if it terminates a boundary edge.

According to the above definitions of boundary elements, an algorithm (Algorithm 5), which looks for boundary triangles, edges and vertices of a network, was developed. It is called the *network boundary* algorithm. Note that the boundary elements output from this algorithm are sequentially ordered. Let us refer to the boundary portion of a network to be connected as a *connection boundary*. The procedure will include two stages. The first stage is to determine the connection boundaries of both N_1 and C_1. The second stage is to fill the gap between the networks with triangles in terms of their connection boundaries.

Algorithm 5: Network Boundary

Objective: Looks for boundary triangles, edges, and vertices of a network N, and output them in order.

Input: A network N.

Output: Ordered boundary triangle list TL, edge list EL, and vertex list VL.

Procedure :

```
begin
  (** looks for boundary elements of network N **)
  for (each triangle $T_i$ of network N)
    for (each edge $e_j$ of triangle $T_i$)
      if ($e_j$ hasn't been shared by another triangle of N)
        begin
          $e_j$ is a boundary edge;
          store $e_j$ in BE;  record $T_i$ in BT;
        end;
  (** determine the orders of boundary edges **)
  while (BE is not empty)
    begin
      start with the first edge of BE;
      move this edge to EL and move its corresponding triangle to TL;
      store the edge vertices in VL;
      set the first edge vertex as HEAD;
      set the second edge vertex as TAIL;
      delete the edge from BE;
      while (TAIL not equal to HEAD)
        begin
          look for all boundary branches B of vertex TAIL;
          if (single branch)
            begin
              store the edge in EL;
              store its second vertex in VL;
              set its second vertex as TAIL;
              delete the edge from BE;
            end;
          else (multiple branches)
            return vertex TAIL and boundary branches B;
        end;
    end;
  for (each boundary vertex $v_i$ of VL)
    begin
      look for all triangles $T_i$ sharing vertex $v_i$;
      determine the orders of triangles $T_i$ based on their edges.
      store the ordered $T_i$ in TL.
    end;
end; (** end of procedure **)
```

Using the output of the above algorithm, the gap connection procedure is described below and details are presented in Algorithm 6.

(1) Determine the connection boundary CCB of C_1.

(1a) Apply the boundary algorithm to C_1. Let CBT, CBE and CBV denote the boundary triangles, edges and vertices of C_1 respectively.

(1b) Apply the boundary algorithm to redundancy **R**. Its boundary triangles RBT, edges RBE and vertices RBV can be derived.

(1c) The connection boundary CCB of C_1 is then determined as the intersection of CBE and RBE. $CCB = CBE \cap RBE$.

(2) Determine the connection boundary NCB of N_1.

(2a) Apply the boundary algorithm to N_1. Let NBT, NBE and NBV denote the boundary elements of N_1.

(2b) Look for the triangles CT from the set of redundant triangles **R**, such that each triangle of CT is adjacent to at least one vertex of the connection boundary CCB of C_1.

(2c) Determine the contribution set ST of CT from the contribution set of **R**. Let SE be the set of edges of the contribution set ST.

(2d) The connection boundary NCB of N_1 is the intersection of SE and NBE.

$NCB = SE \cap NBE.$

(3) Connect the gap between N_1 and C_1 by sequentially filling the connection boundaries CCB and NCB with triangles.

Algorithm 6: Gap Connection

Objective: Connects surviving component C of network N_2 to network N_1 after removing the redundancy intersection **R** from N_2.

Input: Networks N_1, C and R.

Output: The connected network N.

Procedure :

```
begin
 (* look for the connection boundary of C *)
 apply the boundary algorithm to C;
 obtain the boundary elements CBT, CBE, and CBV of C;
 apply the boundary algorithm to R;
 obtain the boundary elements RBT, RBE, and RBV of R;
 the connection boundary of C is $CCB = CBE \cap RBE$;
 look for disconnected boundary pieces CBC of CCB;
 apply boundary algorithm to $N_1$;
 obtain the boundary elements NBT, NBE, and NBV of $N_1$;
 for (each boundary piece $CBC_i$)
  begin
    (* look for the connection boundary of $N_1$ *)
    look for triangles $CT_i$ in R which possess at least one
    vertex of $CCB_i$;
    determine the contribution set $ST_i$ of triangles $CT_i$;
    let $SE_i$ be the set of edges of the contribution set $ST_i$;
    the connection boundary of $N_1$ is
    $NCB_i = SE_i \cap NBE.$
    create triangles between $NCB_i$ and $CBC_i$ for filling their gap;
    delete created triangles, which intersect with their neighboring triangles;
  end;
end; (** end of procedure **)
```

4.1.2.5 Hole Filling

Repeatedly applying Algorithm 6 to all pairs of networks, we can complete a connected network. Unfortunately, small holes may still exist in the resulting network. For example, in Figure 3.7, after C_1 and C_2 are connected to the other networks, a hole will eventually be formed between C_1, C_2 and N_1.

We need another algorithm to fill those holes, called the *3-D hole filling* algorithm. This is different from the *2-D hole filling* algorithm addressed in Section 4.1.2.1. The *3-D hole filling* algorithm assumes that the input network is almost complete except for some small holes. Therefore, the boundaries of the network are equivalently the boundaries of the holes. The detail of this algorithm is specified in Algorithm 7.

(1) Apply the boundary algorithm to the network to look for boundary vertices, which are the boundary vertices of holes.

(2) Individual holes are discovered by seeking disconnected sets of boundary vertices.

(3) For each hole, patch it with triangles.

Algorithm 7: 3-D hole filling
Objective: Fills holes of a 3-D network.
Input: A 3-D network N.
Output: The network with holes filled.
Procedure :

```
begin
  apply the boundary algorithm to N;
  obtain boundary elements NBT, NBE, NBV;
  look for disconnected sets BVN of NBV;
  for (each disconnected set $BVN_i$)
    begin
      V = the vertex list of $BVN_i$;
      n = the number of vertices of V;
      v1 = V[1];  n1 = n;  times = 0;
      while (n > 2)
        begin
          if (n <> n1)
            begin  n1 = n; times = 0;  end;
          else
```

```
     begin
       times = times + 1;
       if (times = n) return;
     end;
    v2 = the first adjacent vertex of v1;
    v3 = the second adjacent vertex of v1;
    if (v1,v2,v3 are collinear)
     begin v1 = v2; repeat; end;
    T = triangle(v1, v2, v3);
    if (T intersects its neighbors)
     begin v1 = v2; repeat; end;
    add T to $P_2$;
    delete v2 from V;
    n = n - 1;
    v1 = v3;
  end;
end; (** end of procedure **)
```

The final version of the constructed geometric model of the cobra head is shown in Figure 4.8 with three different orientations.

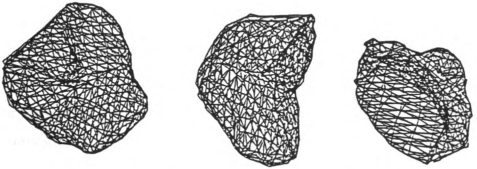


Figure 4.8. The constructed geometric model of the cobra head displayed in different orientations.

4.1.3 Discussion of Model Building

Since data acquired by the sensing system is a set of networks instead of discrete points, the global shape of objects is explicitly preserved in the networks. The registration takes advantage of the connectivity of networks and greatly reduces the workload in the triangulation step. With the benefits of low cost and high precision, the structured light sensing system is suitable for geometric modeling but has some limitations. Later we will present an analysis of error in calibration and transformation. Through this analysis, the thresholds of distance and angle used by the procedure of redundancy removal can be provided. Finally, the difficulties and future improvement to the model building are given.

4.1.3.1 Limitations of the Sensing System

Structured light sensing systems suffer from several limitations. For instance, even under all possible positions, some deep concavities existing on an object surface may not be simultaneously seen by both the camera and the projector. Such concavities are referred to as **blind regions**. To reduce the size of such blind regions, one can move the camera and the projector closer to each other. However, this increases the error in computing the coordinates of 3-D points from triangulation.

The sensing system also has difficulty with some areas on an object surface, such as the top of a typewriter keyboard. We call an area with many texture-like elements a **busy area**. Stripes lying on a busy area will be very disconnected. Their grid coordinates are thus difficult to identify. It is better to remove busy areas from consideration by filling gaps with material or covering with papers.

For some narrow regions, such as the handle of a coffee cup and the barrel of a toy tank, points are either difficult to sample, or the sampled points are not sufficient to cast a significant surface patch. This occurs where the feature widths are smaller than the intervals of light sheets. We call these regions **dead patches**. Grid slides with finer resolution may be used to compensate for this difficulty. One can also use a CAD system to add the features to the model.

4.1.3.2 Calibration Error Analysis

Suppose that the combined error caused by feature point detection and modeling by the camera matrix is assumed to be e pixels in the image plane. We want to predict the error in the computed 3-D point location. First let us consider the one camera case. Refer to Figure 4.9. Suppose a point P is located on the plane $z = z_m$. Clearly, the error ε will be a function of coordinate c, coordinate error e, tilt angle θ, focus length f, and standoff d. The following refers to the configuration in Figure 4.9, and considers the worst case of ε.

Since $\tan\theta_1 = \frac{c}{f}$, and $\tan\theta_2 = \frac{c+e}{f}$,

we get $r_t = \frac{d \sin\theta}{\tan(\theta \pm \theta_1)}$, and $r_c = \frac{d \sin\theta}{\tan(\theta \pm \theta_2)}$.

The error is

$$\varepsilon = |r_c - r_t| = d \sin\theta \left| \frac{1}{\tan(\theta \pm \theta_2)} - \frac{1}{\tan(\theta \pm \theta_1)} \right|.$$

However, $\tan(\theta \pm \theta_1) = \frac{f\, tan\,\theta \pm c}{f \mp c \tan\theta}$, and $\tan(\theta \pm \theta_2) = \frac{f\, tan\,\theta \pm (c \pm e)}{f \mp (c \pm e) tan\theta}$.

Thus, we obtain $\varepsilon = \left| \frac{dfe\,(1 + \tan^2\theta)\sin\theta}{f^2\tan^2\theta - fe\,\tan\theta + c\,(c - e)} \right|$ in the worst case.

This error equation can be further simplified by the fact that the standoff d is usually much larger than the object size, so that $\tan\theta$ can be approximated by $\sin\theta$. This reduces the equation for ε to a much simpler equation,

$$\varepsilon \sim \frac{de}{f \sin\theta}. \tag{4.1}$$

Suppose we have $e \sim 2$ pixel $\sim 0.026mm$, $d \sim 1750mm$, $f = 50mm$, and $\theta \sim 45$ degrees. Substituting these values into (4.1). We get $\varepsilon \sim 1.28mm$. Thus a 2-pixel error in the image plane will cause about 1.28mm radial error on the workbench 1750mm away. If we extend the consideration to different planar levels, an error cone can be formed with the view point as the vertex of the cone.

The previous discussion examined the error which is possible when imaging points on a known plane (the workbench surface). However, we would like to take the projector into consideration. If we assume that the lens of the projector is similar to the lens of the camera, we shall expect similar modeling errors in the calibrated projector matrix. The error cone corresponding to the projector can then be formed (refer to Figure 4.10). Suppose a point P is located within the field of views (FOV) of both the camera and the

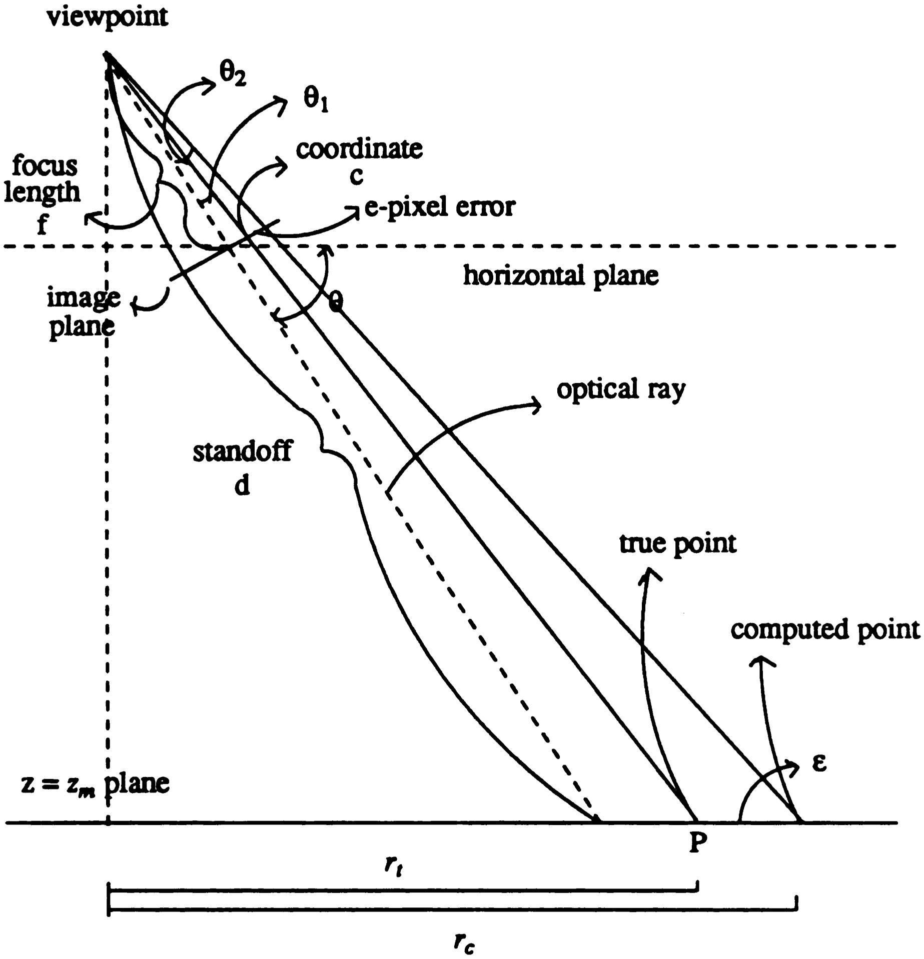


Figure. 4.9. Error of e pixels in image yields radial error r' on given plane.

projector. Let P_a be the image point of P on the camera plane, and P_b be the projected point of P on the projector plane. Then the 3-D coordinates of P can be computed by intersecting the camera ray, which passes through P_a, and the projector ray, which passes through P_b. The mathematical computation could fix the position of P anywhere within the intersection of the two error cones. It is neither simple nor necessary to derive an exact measurement of this 3-D error volume.

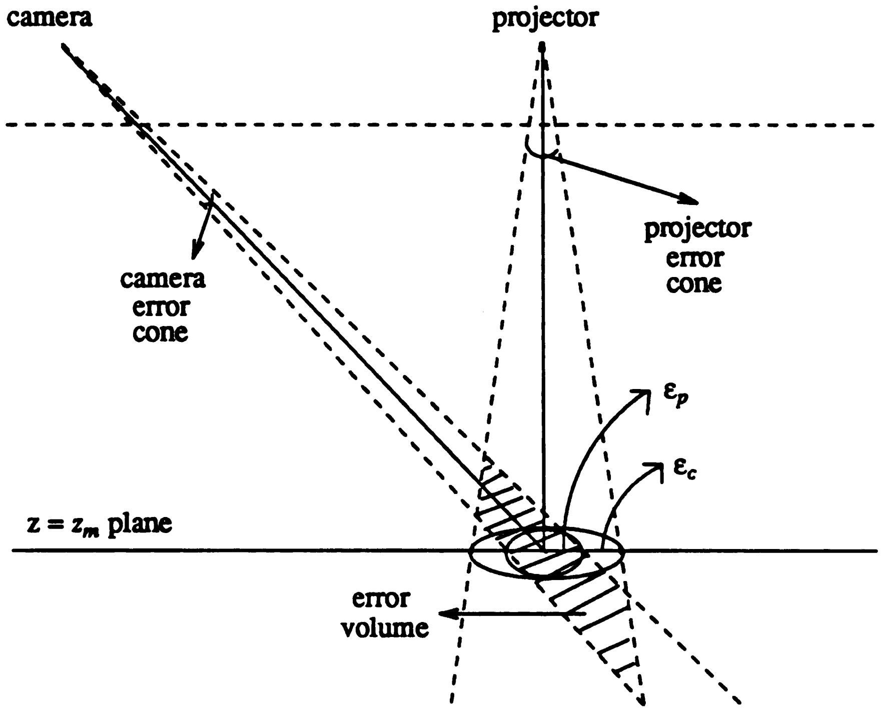


Figure. 4.10. Points computed from triangulation can be anywhere in the intersection of the two error cones.

4.1.3.3 Transformation Error Analysis

The following analysis describes how initial errors of positions and angles will be augmented after transformations. Similar analysis can also be found in [Gri85]. The results of this section and the last section will provide thresholds on distance and angle needed by the redundancy removal procedure.

Theorem 4.1: If a line segment $\overline{o_1 o_2}$ with positional error of at most r_o for each of its endpoints is rotated, the endpoints of the rotated line segment have a relative positional error bound.

$$p' \leq 2\sqrt{1-(1-p^2)^{1/2}} + p\sqrt{3}. \tag{4.2}$$

where $p = \frac{2r_o}{d}$, $p' = \frac{2r_o'}{d}$ and d is the length of the line segment, $d = |\overline{o_1 o_2}|$. r_o' is the new positional error after rotating.

Proof: See Appendix C.

To make this theorem more explicit, let us look at an example. Suppose the original relative error p_0 of points is 1%. From Equation (4.2), the augmented error after a rotation is:

$$p_1 = 2\sqrt{1-(1-p_0^2)^{1/2}} + p_0\sqrt{3} = 2\sqrt{1-(1-0.01^2)^{1/2}} + 0.01\sqrt{3} = 0.0315.$$

After two rotations, the error will have increased to $p_2 = 0.0944$, ie. 9.4% error.

Next, let us consider the error propagation through a translation. As indicated in Section 3.2.1, the translation is computed by averaging three translation vectors. These are obtained by subtracting positional vectors of the starting points from the target points. Suppose the errors of the starting point and the target point are respectively e_1 and e_2. Then for the worst case the resulting error will be increased to $e_1 + e_2$ after a translation. Each transformation is composed of two rotations and one translation, so the final transformation error bound results from their combination.

- Suppose the original positional error of points is r_o.
- The relative error is $p_o = \dfrac{2r_o}{d}$.
- The relative error after performing the first rotation is

$$p_1 = 2\sqrt{1-(1-p_o^2)^{1/2}} + p_o\sqrt{3}.$$

- The relative error after performing the second rotation is

$$p_2 = 2\sqrt{1-(1-p_1^2)^{1/2}} + p_1\sqrt{3}.$$

- The error after performing the translation is $r' = \dfrac{p_2 d}{2} + r_o$.
- r' is the upper error bound or the worst case error.

From the error analysis (1.28mm) of calibrations in the last section, let us assume the positional error of points is about 1.5mm; $r_o \sim 1.5$. Suppose the diameter of an object is about 250mm. Hence the initial relative error is $p_o \sim 1.2\%$. After the first rotation, the error is raised to $p_1 \sim 3.8\%$, and up to $p_2 \sim 11.2\%$ after the second rotation. That is, after two rotation transformations, the initial error 1.5mm will increase to 14mm in the worst case. Adding this error to the translation error (1.5mm), we get the error bound $r' \sim 15.5mm$. Note that the computation of positional error due to rotations is quite dependent on the distances between points to the rotation center. The result of 15.5mm is the worst case. Actual error should be less than this value. In addition, the inconsistency reduction procedure may reduce the positional error.

Now, let us consider how error in the initial angle is increased after the transformation. First we need to figure out by how much the orientation of a line segment will be shifted from its original direction, if a positional error is introduced to its endpoints. Based on the result, we want to estimate the angular shift of the surface normal of a triangle between the before and the after of a positional error being introduced to the triangle vertices.

Proposition 4.2: If the endpoints of a line segment possess a positional error of at most r_o, then the maximum angle shift $\Delta\theta$ from the true orientation is

$$\Delta\theta = \cos^{-1} (1 - p_1^2)^{1/2}$$

where $p = \frac{2r_o}{d}$, and d is the length of the line segment; $d = |\overline{o_1 o_2}|$.

Proof: See [Che87].

Based on the above proposition and the fact that a triangle is composed of three line segments, we have the following result.

Proposition 4.3: If a plane is represented in terms of three points, each point with the largest positional error r_o, then the maximum angle shift of the surface normal from its true direction is

$$\Delta\theta = \cos^{-1} (1 - p_1^2)^{1/2} \qquad (4.3)$$

where $p = \frac{2r_o}{d}$, and d is the length of the line segment; $d = |\overline{o_1 o_2}|$.

Proof: See [Che87].

Note that the value *d* here represents the diameters of triangles of networks. Suppose the observed diameters of triangles range from 15mm to 25mm. According to these values, the angle shift computed from (4.3) will be disastrous because the ratio of error to signal is too large in the worst case. For general cases, the positional error is about 0.7mm. Let us choose 20mm as the general diameter of triangles. From the equation of transformation error bound, the augmented positional error of points after transformations becomes 7.6mm. Therefore the relative positional error p is 76%. Substituting these values into (4.3), we obtain the resulting angular shift of the surface normal is about 49.5 degrees in the worst case. Such large errors, however, are not typically observed. One reason is that worst cases are not frequent. Another reason is that error frequently affects points in a neighborhood systematically yielding a cancelling effect at least for computation of normals. Shrikhande & Stockman [Shr88] showed that normals can be computed to within 3-6 degrees.

4.1.3.4 Difficulties and Suggestions

Since the model shown in Figure 4.8 was constructed using only four sensor views of the cobra head, there was a hole on the top back of the constructed model, which had been patched by the hole filling algorithm. To get rid of this problem, more sensor views will be needed. Currently, the number of sensor views needed to construct a model is determined by the operator. There is no general strategy to figure out *a priori* the most suitable number of views. There is also no optimal method for deciding which views should be registered first. A different ordering of views can result in a different final network.

Typically, using slides with finer grids should improve the quality of a constructed model. This may not be true for objects with simple shapes such as polyhedral objects and quadratic surfaces. The techniques proposed here are suited for objects with moderately complicated surfaces, which can not be easily handled by CAD systems. For such objects, the selection of grid slides becomes important. In general, it depends on aspects such as the complexity of the objects and the purposes of application.

It is believed that the silhouettes of object views can provide clues for registration. Several researchers [Wan87, Pot83] have worked in this direction. However, since their working objects possess symmetrical shapes such as bottles and cars, it is not clear if their techniques can be applied to arbitrarily-shaped objects. An approach developed by Henderson and Bhanu [Hen85, Bha87] using the *k-d* tree data structure of sampling points to connect points into a network has more applicability in dealing with general objects.

Of the seven steps in our registration process, five of them are quite stable and reliable when applied to our object database. However, the gap connection and 3-D hole filling procedures have encountered difficulties. In the gap connection algorithm, there is a subroutine which looks for boundary elements in a network. This routine has no problem finding all boundary elements but has trouble arranging them into ordered sequences if a network possesses multiple edge branches in its boundary. Unfortunately, this situation often occurs at twisted regions on the object surfaces. Holes appearing in such areas are not filled completely using the 3-D hole filling algorithm. These difficulties are due in part to current approaches used in the gap connection and hole filling algorithms, both of which use straightforward heuristics. Methods based on computational geometry may provide better solutions to these problems.

The procedures which follow the inconsistency reduction procedure could be performed by a single procedure which might avoid the aforementioned difficulties. The fundamental idea is that instead of removing redundant overlaps among networks, they could be directly webbed together by connecting the overlapping vertices to their nearest neighbors. A possible drawback of this strategy is that the resulting model will have a different vertex distribution over the model surface. That means the sizes of triangles will be nonuniformly distributed over the model surface. It may be necessary to either rearrange the positions of vertices to adjust sizes of triangles, or to directly tessellate large triangles into smaller ones.

4.2 Augmenting Geometric Models

The geometric models constructed in this research are useful in a variety of applications such as computer graphics, collision avoidance, manipulation, and automatic manufacturing. They are not suitable as prototypes in most recognition tasks. For recognition purposes, objects are usually represented in terms of their notable features and structural relations. To extract features and relations directly from objects has its own difficulties. Geometric models provide an intermediate stepping-stone to facilitate the generation of recognition-oriented object representations. Model augmentation is one of the essential steps toward this end.

A geometric model initially contains a set of vertices and a set of labeled triangles. To augment such a model, we need to include potential surface characteristics for each model vertex. Later, the surface properties of triangles are readily derived from those of vertices via weighting strategies such as barycentric coordinates [Far87]. Surface properties include surface type, curvature, edge magnitude, surface roughness and the angle between principle directions. They have been precisely defined in the previous chapter and will serve as a guide to segment images generated from the model.

Consider a model vertex. An approximate surface normal at this point is estimated by averaging the surface normals of neighboring triangles. Then, we can construct a tangent plane passing through that vertex and perpendicular to the estimated surface normal. This plane will be used as a projection plane on which the object model is projected [Fol82] and a range image can be obtained. A quadratic surface function [Bes88] is then employed to fit range values. Parameters of equation are determined by minimizing a least squares error function. Surface characteristics at the model vertex can then be computed based on the derived parameters. Model augmentation is achieved by applying the above procedure in parallel to all model vertices.

The method proposed here enjoys two advantages. The first is that current techniques of *Computer Aided Geometric Design* (CAGD) haven't provided enough approaches to compute all surface characteristics which we need from a single connected network. Secondly, our method is analogous to the techniques commonly adopted in computing surface characteristics from a real image, so that the characteristic values of augmented models will be compatible with those computed from real images. However, different image resolutions can affect the computational results due to discretization error.

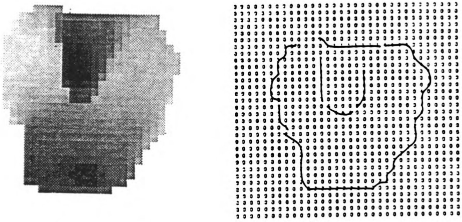
4.3 Object Representation

For representing objects, a multiple-view scheme called the viewsphere, is adopted. Objects are presumed to be located at the center of the sphere. Each object is initially represented by a set of views. These are generated by projecting the augmented model of the object along the directions from the center of the sphere to a set of viewpoints distributed on the surface of the sphere. Each object view is in turn represented by a set of wings and their structural properties.

It is natural for us to attempt to use as few views as possible to represent an object. Researchers [Ike88] have tried to group views into so-called *aspects* such that each aspect possesses views with certain consistent properties such as linear shape changes or the same set of features. This partitions the surface of a viewsphere. Object representations with such partitions may suffer from ambiguity in determination of the optimal viewpoint at the boundaries of partitions.

4.3.1 View Representation

An object view is generated by projecting an augmented geometric model to a plane tangent to the viewsphere at a given viewpoint. The view is composed of a set of arrays containing the range values and other surface characteristics available in the model. Figure 4.11(a) displays a range image (32 × 32) generated from the cobra head model. Based on these maps, the wing representation of that view can be derived. The procedure proposed for wing detection based on the characteristic maps is given as follows.



(a) range image

(b) wing map

Figure 4.11. (a) The range image of a view of the cobra head has been generated from the cobra head model. (b) The resulting wing map of this view.

Given a set of maps corresponding to an object view (including a range image, a preliminary segmentation of the image, and maps of other surface properties) contours are first located based on the characteristics of range values, edge magnitudes, fit errors and angles between principal directions. Contour types of interest include object silhouettes, internal jump edges and crease edges. After extracting contour information, the algorithm proceeds to make the preliminary segmented images uniform by integrating the contour information. This process currently uses a 3 by 3 window to replace the surface type of a region pixel by a new type which is determined as the majority set of the types adjacent to that pixel. Previously detected contours are used to prevent this process from counting neighboring pixels across contours. This process continues iteratively until no change of surface types can be made or the number of iterations exceeds a limit. Based on the resulting contour and surface maps, wing primitives can be found through a labeling procedure, which implements the **extraction rule** presented in Chapter 2. A diagram of the entire procedure is depicted in Figure 4.12. The resulting wing map for the example in Figure 4.11(a) is shown in Figure 4.11(b).

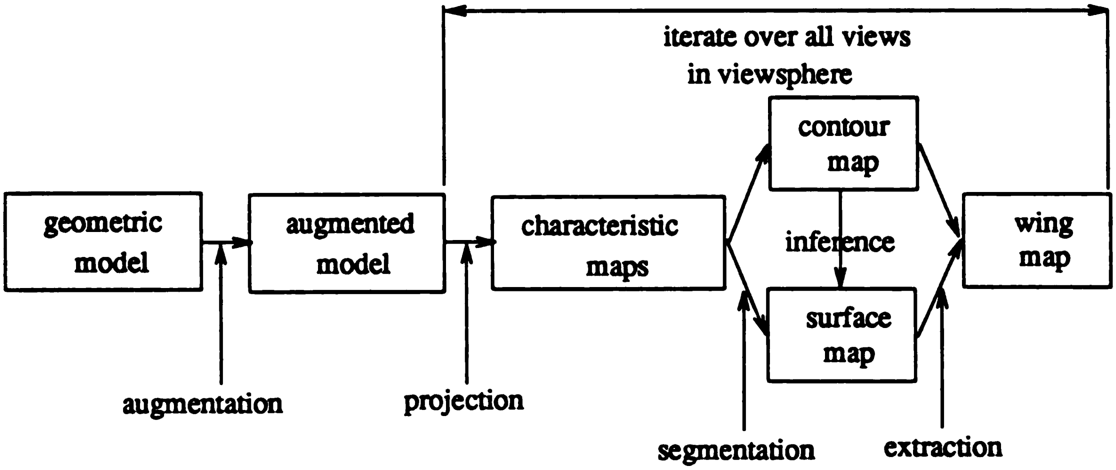


Figure 4.12. The procedure diagram of wing detection.

After deriving the wing features, their structural properties can be computed using the computational methods presented in Chapter 2. The results of computation for the wings in Figure 4.11(b) are illustrated in Figure 4.13. The view information is grouped into three classes: A, B and C. Class A contains the properties of the view in which the number of distinct wing types (6 in the example), the list of wing types (2,3,4,9,10,11, refer to Table 1.1) and the number of wing features (8 wings) detected from the view are indicated. Following Class A is the class B information, where each detected wing is specified by a set of feature properties including length, depth range, tangent magnitude, curvature and so on. This class concerns unary properties of individual wings. In Class C, the binary relations between wings were described. They include distances, ratios and angles defined in Chapter 2. In graphic terms, the data structure in Figure 4.13 is a specification of a relational attributed graph, and is the wing representation of the object view.

4.3.2 Views and Aspects

In the last section, we computed the wing representation for an arbitrary viewpoint. However, a viewsphere is initially composed of 320 viewpoints. The tessellation procedure presented in Section 3.2.4 generates these viewpoints on a unit sphere. Therefore, according to these viewpoints, 320 views of an object and their corresponding wing representations can be derived. Figure 4.14(a) displays 48 of the 320 views of the cobra head. The views in Figure 4.14(b) are those of the coffee cup. For each such object view, the procedures of wing detection and structure computation are applied. Thus, we obtain a set of 320 view representations characterized by wing features and their structures. Examples of range, segmented and wing images of model views of the cobra head and the coffee cup are shown in Figure 5.15. The computation is obviously expensive but it can be done off line.

Recall that the tessellation procedure starts with an icosahedron which has 20 triangular faces. Each face is then divided into four finer triangles to obtain 80 triangles. Thereafter, each triangle is further divided into four smaller triangles. A set of 320 triangles, which correspond to 320 viewpoints, is created. This procedure inherently provides a representational scheme with a hierarchy of three levels. The first level contains 20 faces of the icosahedron, the second level consists of four subtriangles for each of the 20 faces in level one and the last level contains four smaller triangles for each subtriangle in level two. Suppose we are given an unknown view. We want to look for a viewpoint from which the corresponding object view is consistent with the unknown view. We can start by comparing the unknown view with the 20 viewpoints in the first level. Then, the four viewpoints belonging to the best candidate of the last level, which is the most consistency with the unknown view, are compared. This procedure similarly continues to the third level.

In order to reduce the number of object views, we cluster the views sharing the same set of wing primitives into groups called *aspects*. For the example of the cobra head, 139 aspects are obtained from the initial 320 views. However, 36 aspects are needed for the coffee cup. The number of aspects can reflect the complexity of an object shape. The cobra head has higher complexity than the coffee cup. The above hierarchical organization of the viewsphere may be useful for views represented by object wings once there is wing property that complies with the inherent hierarchy of the viewsphere. We also found that some aspects whose sets of wing primitives are subsets of other aspects. Aspects could be arranged into the structure of *lattices* so that they can be tightly organized. The performance of the recognition process might thus be improved. We will investigate these problems in the next chapter.

4.3.3 Preliminary Discussion of Wing Representation

There are several factors that can affect the final object representation. These factors include map resolution, tessellation of a viewsphere and convolution filters. In this section, a preliminary discussion about these factors is presented. A closer examination together with a series of experimental results will be given in the next chapter when we discuss the validity of the representation in terms of wing features. In this way, a better understanding of the characteristics of wing representation will be gained.

During model augmentation, the fundamental task is to compute surface characteristics for model vertices. Computation is based on a set of maps generated by projecting a geometric model to the tangent planes at model vertices. However, different map resolutions will lead to different results in the computed characteristics. This problem also occurs during the generation of the representation for a view. In order to investigate the effects of resolution, we used various map sizes ranging from 32×32 to 128×128 to generate wing representations for the cobra head and the coffee cup. The experimental results and their comparisons are given in the next chapter.

Although the number of aspects is directly related to the complexity of object shapes, it is also possible that the initial resolution of a viewsphere may influence the final representation. In order to understand this, we generated two viewspheres with different tessellation resolutions (240 and 320 viewpoints respectively). For the case of 320 viewpoints, the solid angle of a viewpoint (or the view resolution) is about 0.039 steradians (the angle interval between two adjacent viewpoints is about 12.5 degrees), whereas

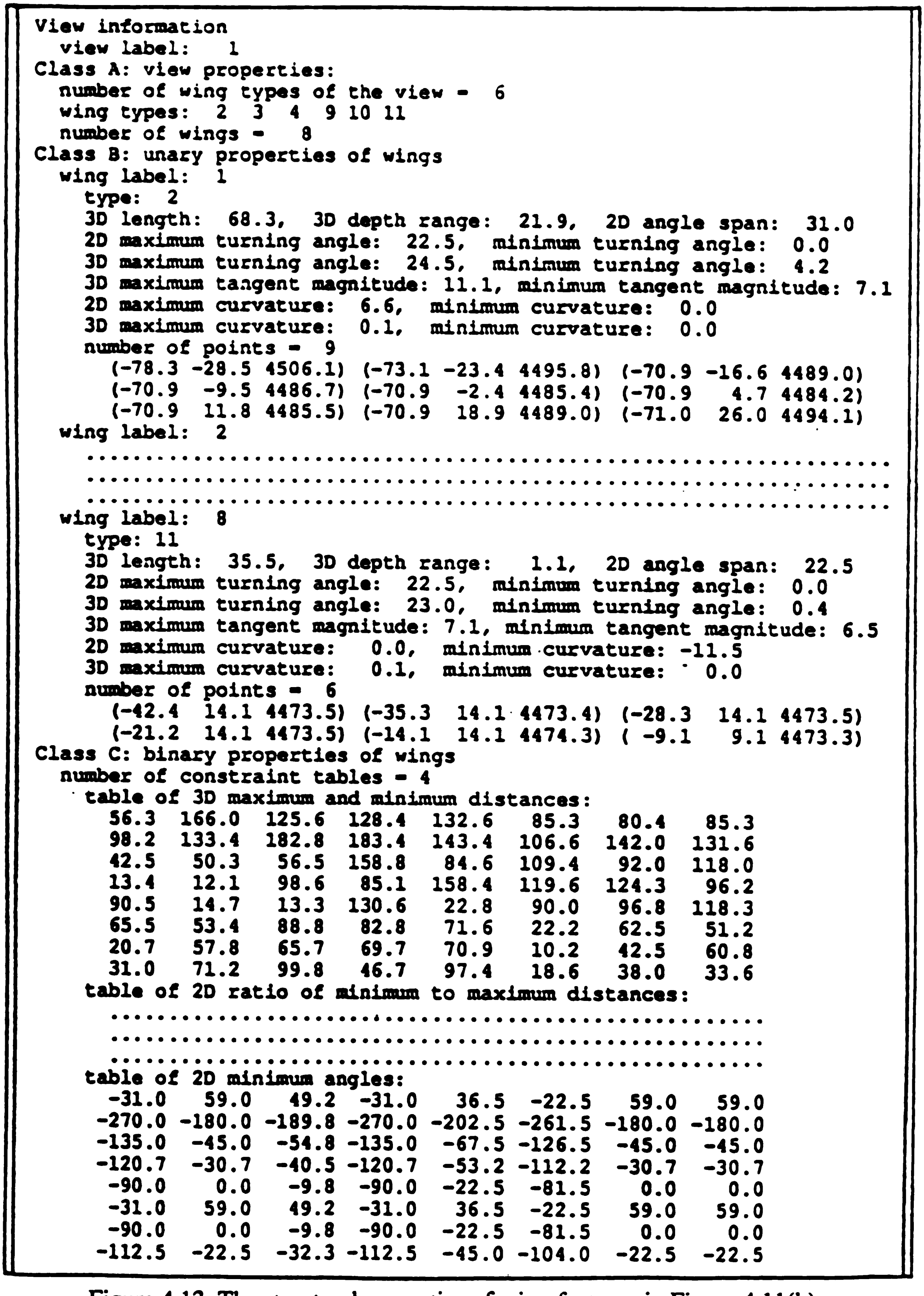
0.052 steradians (angle interval about 14.5 degrees) for the case of 240 viewpoints. For a moderately complicated object, it is possible that different wing features could be obtained as the viewpoint varies within a small neighborhood in a triangulation of the viewsphere. In practice, one often finds that in spite of small differences many features remain the same. Viewpoint perturbation will be further discussed in the next chapter.

Another point of interest, which has been addressed by Besl & Jain [Bes88], is the fact that different sizes of convolution masks used to compute shape information will suffer from computational variations which are a function of mask size and the area covered by the mask. This problem is commonly referred to as a *scale problem*. It is possible for us to construct a model to simulate these computational variations so that better results may be achieved using the estimated variations. We will present a general simulation model for this purpose and give a specific use of it in the next chapter. Several researchers [Low88] have used this concept to estimate computational variations and improve their results.

4.4 Summary

A procedure for construction of wing representations of real world objects was presented. This procedure included four major steps: model building, model augmentation, view generation and view organization. In model building, 3-D data acquisition and view registration were two essential steps. A geometric model was specified in terms of a set of triangles whose union was a topological and geometrical approximation to the object surface. Surface characteristics were computed at every vertex of a model. Geometric models containing this information were called augmented models.

Various views of an object were generated by projecting its augmented model onto a *viewsphere* which was composed of a set of uniformly distributed viewpoints. Object views were then clustered into *aspects*, where the views within an aspect possessed the same set of either distinct or distinguished wing primitives. As a consequence, an object was finally represented by a set of aspects which were in turn sets of object views. Each object view was characterized by wing features and their structural relations. The model construction procedure was demonstrate on the cobra head object. More example objects and the properties of representations will be discussed in the next chapter.



```
View information
  view label:   1
Class A: view properties:
  number of wing types of the view =  6
  wing types:  2  3  4  9 10 11
  number of wings =   8
Class B: unary properties of wings
  wing label:  1
    type:  2
    3D length:  68.3,  3D depth range:  21.9,  2D angle span:  31.0
    2D maximum turning angle:  22.5,  minimum turning angle:  0.0
    3D maximum turning angle:  24.5,  minimum turning angle:  4.2
    3D maximum tangent magnitude: 11.1, minimum tangent magnitude: 7.1
    2D maximum curvature:  6.6,  minimum curvature:  0.0
    3D maximum curvature:  0.1,  minimum curvature:  0.0
    number of points =  9
      (-78.3 -28.5 4506.1) (-73.1 -23.4 4495.8) (-70.9 -16.6 4489.0)
      (-70.9  -9.5 4486.7) (-70.9  -2.4 4485.4) (-70.9   4.7 4484.2)
      (-70.9  11.8 4485.5) (-70.9  18.9 4489.0) (-71.0  26.0 4494.1)
  wing label:  2
    ..............................................................
    ..............................................................
    ..............................................................
  wing label:  8
    type: 11
    3D length:  35.5,  3D depth range:   1.1,  2D angle span:  22.5
    2D maximum turning angle:  22.5,  minimum turning angle:  0.0
    3D maximum turning angle:  23.0,  minimum turning angle:  0.4
    3D maximum tangent magnitude: 7.1, minimum tangent magnitude: 6.5
    2D maximum curvature:   0.0,  minimum curvature: -11.5
    3D maximum curvature:   0.1,  minimum curvature:   0.0
    number of points =  6
      (-42.4  14.1 4473.5) (-35.3  14.1 4473.4) (-28.3  14.1 4473.5)
      (-21.2  14.1 4473.5) (-14.1  14.1 4474.3) ( -9.1   9.1 4473.3)
Class C: binary properties of wings
  number of constraint tables = 4
    table of 3D maximum and minimum distances:
      56.3  166.0  125.6  128.4  132.6   85.3   80.4   85.3
      98.2  133.4  182.8  183.4  143.4  106.6  142.0  131.6
      42.5   50.3   56.5  158.8   84.6  109.4   92.0  118.0
      13.4   12.1   98.6   85.1  158.4  119.6  124.3   96.2
      90.5   14.7   13.3  130.6   22.8   90.0   96.8  118.3
      65.5   53.4   88.8   82.8   71.6   22.2   62.5   51.2
      20.7   57.8   65.7   69.7   70.9   10.2   42.5   60.8
      31.0   71.2   99.8   46.7   97.4   18.6   38.0   33.6
    table of 2D ratio of minimum to maximum distances:
      ..........................................................
      ..........................................................
      ..........................................................
    table of 2D minimum angles:
     -31.0   59.0   49.2  -31.0   36.5  -22.5   59.0   59.0
    -270.0 -180.0 -189.8 -270.0 -202.5 -261.5 -180.0 -180.0
    -135.0  -45.0  -54.8 -135.0  -67.5 -126.5  -45.0  -45.0
    -120.7  -30.7  -40.5 -120.7  -53.2 -112.2  -30.7  -30.7
     -90.0    0.0   -9.8  -90.0  -22.5  -81.5    0.0    0.0
     -31.0   59.0   49.2  -31.0   36.5  -22.5   59.0   59.0
     -90.0    0.0   -9.8  -90.0  -22.5  -81.5    0.0    0.0
    -112.5  -22.5  -32.3 -112.5  -45.0 -104.0  -22.5  -22.5
```

Figure 4.13. The structural properties of wing features in Figure 4.11(b).

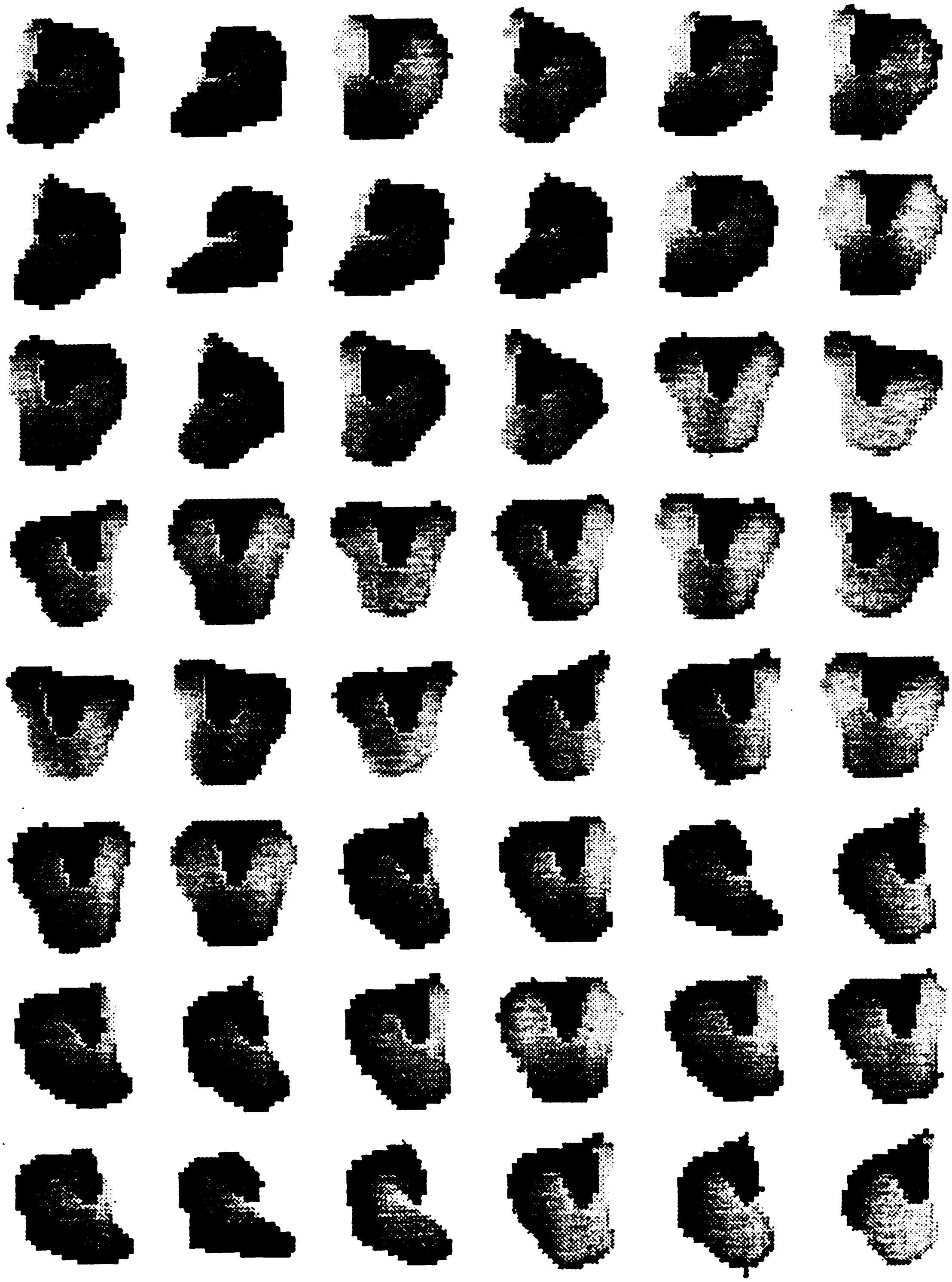


Figure 4.14(a). 48 of the 320 views of the cobra head.

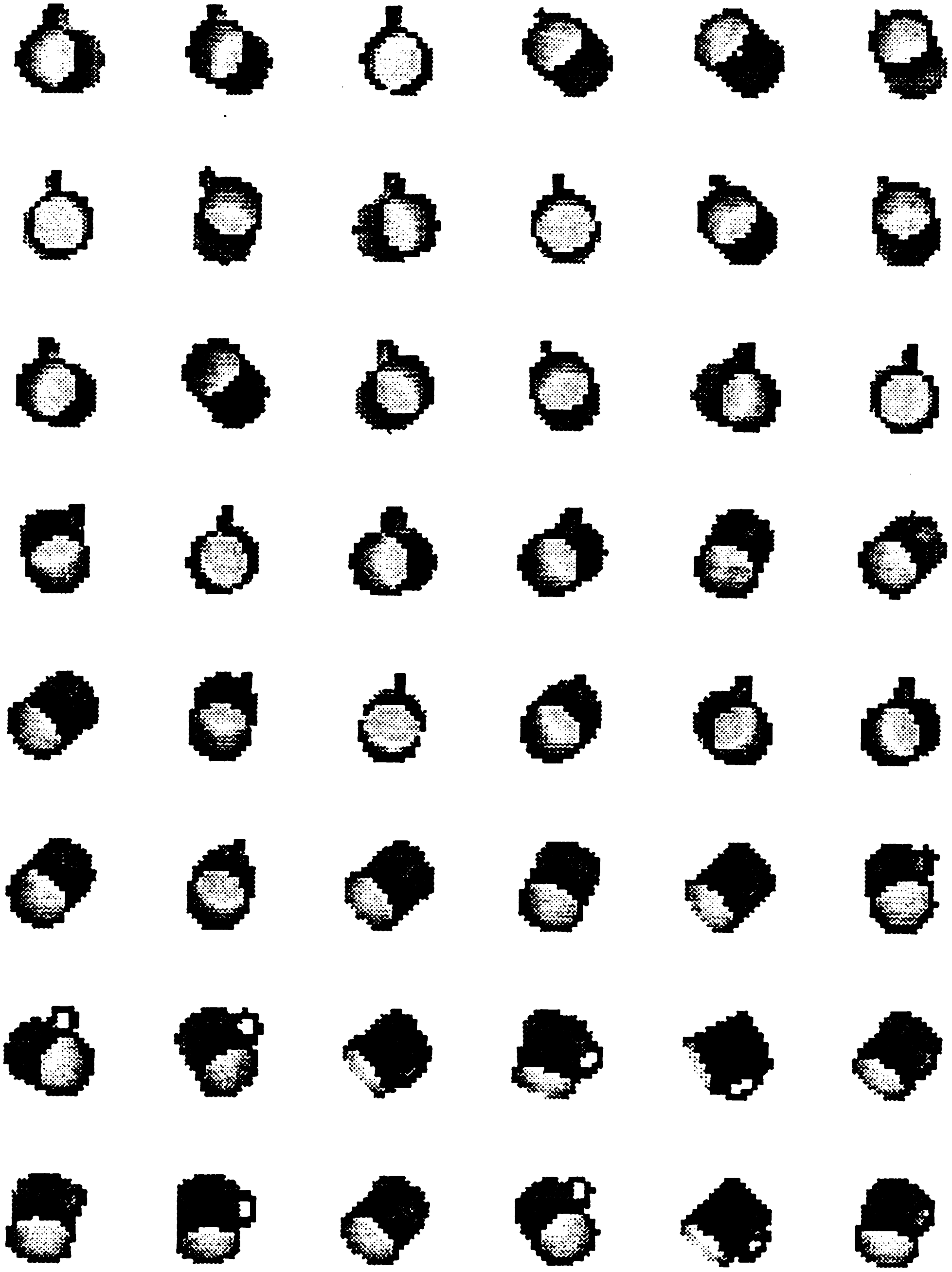


Figure 4.14(b). 48 of the 320 views of the coffee cup.

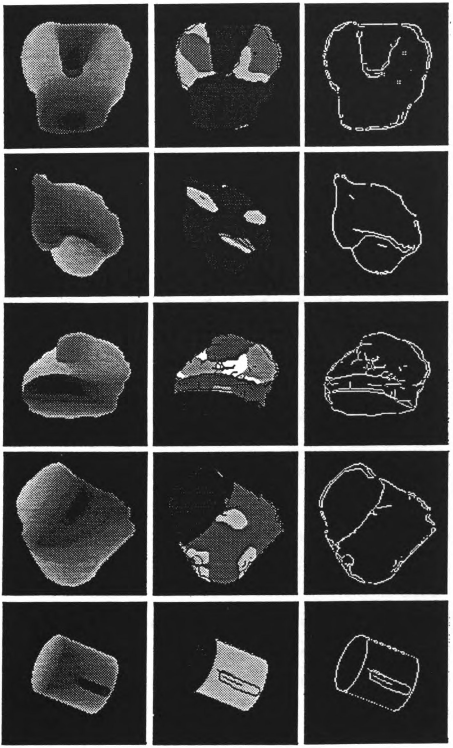


Figure 4.15. Examples of range, segmented and wing images of model views.

Chapter 5

Testing the Wing Representation

A good object representation should have the properties of uniqueness, stability, compatibility, terseness and locality. The property of *uniqueness* requires that the representation of an object be distinguishable from the representations of other objects in a domain. The *stability* property demands that small variations in the input have little influence on the representation. As for the requirement of *compatibility*, the representations in a database should be comparable to the representations derived from real images. In addition, the representations should be also *terse* enough to make processing efficient. Finally, *locality* enables the representation to cope with partial occlusion.

Several procedures have been developed to examine the validity of wing representations with respect to the aforementioned properties. In the following sections, the properties of uniqueness, stability, and compatibility are closely investigated. The property of locality was shown in Chapter 2. However, the current representation scheme does not produce terse representations because it is based on multiple views. To avoid this drawback, we need to either reduce the number of *aspects* [Sho88] or use other representation schemes [Lam88]. Later in this chapter, we show that a strategy based on sets of distinguished wing primitives of individual models can further reduce the number of aspects.

First, the uniqueness of wing representation is investigated by examining the number of aspects, the distinct and distinguished wing sets, and the wing structures of object models. A set of uniqueness criteria will be proposed for this purpose. They form necessary but not sufficient conditions. Next, we examine the property of stability, including investigations of the reliability of algorithms for constructing wing representations under varying resolutions of characteristic maps, tessellations of the viewsphere and convolution filters.

It is essential that object wings extracted from real images be comparable with those of internal representations. Otherwise, the recognition procedure will not succeed in identifying scene objects based on the representations. We call this property compatibility. Our discussion of this property doesn't attempt to present a complete procedure for wing detection. Instead, we will demonstrate that wings detected from real images can be comparable to model wings.

While some broader issues are left for future research, for the time being, we conclude based on the experimental results that for the object domain used wing representation has the properties of uniqueness, stability, compatibility and locality, but not terseness.

5.1 Uniqueness

The uniqueness of representation requires that the representation of an object be distinguishable from the representations of other objects in a domain. Suppose we are given an object domain **O** to be represented. Let f be the mapping from the object domain **O** to an image domain **R** which is the set of object representations. To prove that object representation is unique in the domain, we need to prove f is a reversible function, or equivalently f is an one-to-one and onto mapping between the range and the domain of the function. Suppose R_1 and R_2 are representations of two distinct objects O_1 and O_2. The uniqueness property requires that $R_1 \neq R_2$, iff $O_1 \neq O_2$. For wing representation, there are several ways to discriminate two representations. Recall that a wing representation is composed of a set of object wings and spatial constraints. If two wing representations are equivalent, they should possess the same set of wings and constraints. It is possible that two objects have the same set of object wings, for example, in some cases of polyhedral objects. However, it is very unlikely that two different shaped objects have exactly the same set of both wings and constraints.

There are several criteria in terms of object wings and wing structures, which can be used to justify the uniqueness of wing representations. These criteria form a set of necessary but not sufficient conditions. A formal proof of uniqueness should come along with the algorithm of constructing multiple-view representation. Violation of any criterion will produce a penalty. A similarity function of two representations is then constructed based on the penalties. A similar notion could be employed to construct a priority function for object recognition. For example, given a scene representation $\mathbf{R}_s$ and a set of object representations, the object representation $\mathbf{R}_m$ that has the largest similarity with the scene representation $\mathbf{R}_s$ will be tested first.

5.1.1 Criteria of Uniqueness

A set of criteria is presented to serve as evidence for the uniqueness of wing representations. Some of these criteria will later become important for the recognition procedure to index into a model database once sensed wings are available.

Before describing the criteria, several terms and their symbols need to be defined. First, we want to distinguish between a *wing primitive* and a *wing feature*. A wing primitive p_i simply indicates the symbolic feature type of a wing, whereas a wing feature f_i denotes a wing together with its unary properties. As mentioned in Chapter 2, the unary properties of a wing include wing type, length, depth range, angle span, curvatures, etc.. Therefore, wing features provide detailed attributes of wing primitives.

An *aspect* of a model is a collection of model views which satisfies a grouping condition which states that views with the same set of wing primitives can be clustered. A more restricted definition of the grouping condition can be made by including the relative positions of views and considering the set of wing features rather than just wing primitives. By the relative positions of views, we mean that only adjacent views satisfying the grouping condition can be clustered into an aspect.

The *appearance frequency* ap_j of a wing primitive j is defined as the ratio of the number of viewpoints, in which the wing primitive appears, to the number of viewpoints of a viewsphere. A wing primitive with high appearance frequency in an object model means that the wing primitive is a key primitive of the model. The key primitives of a model k form the *distinguished wing set* D_k of the model.

The criteria for justification of the uniqueness of wing representations are listed in Appendix D in which the sequence of criteria is ordered with larger w_i's representing more rigorous criterion. We can also make $\sum_i w_i = 1$ via normalization. Note that the criteria of uniqueness are necessary but not sufficient conditions because their inverses may not be true. For example, in criterion 1, if $\mathbf{R}_1 \neq \mathbf{R}_2$ it doesn't imply that $n_1 \neq n_2$. Let $S = \sum_i s_i$, where s_i is the penalty of criterion i. Then, the larger S, the more distinguishable the pair of representations. In order to make a similarity function with its value between 0 and 1, we define $F = e^{-s}$, so that if $F = 1$, two representations $\mathbf{R}_1$ and $\mathbf{R}_2$ are exactly the same. On the other hand, if F is close to 0, the two representations are well separated. We believe that by introducing higher level wing features such as composite wings into the representation the probability that two wing representations constructed for two different objects satisfy all the criteria of uniqueness would be very small.

5.1.2 Distinct Wing Set, Appearance Frequency and Distinguished Wing Set

In this section, the wing representations for the cobra head and the coffee cup are used as an example to illustrate that wing representations can be unique with respect to the criteria addressed in the last section.

Table 5.1 displays the lists of distinct wing primitives with their appearance frequencies in 320 viewpoints (Appendix B) using map resolution 96 × 96. There are 27 distinct wing primitives for the cobra head and 23 for the coffee cup. Let us look at the first row of the table in which the wing primitive with type 1 appears in 115 views for the cobra head and thereby has appearance frequency 0.359. In a similar vein, the wing primitive with type 1 is also possessed by the coffee cup and appears in 147 views for the

Viewsphere of 320 Viewpoints, Map Resolution 96 × 96					
Cobra Head (27 distinct wing primitives)			**Coffee Cup (23 distinct wing primitives)**		
Wing Primitive	**#Appearances**	**Frequency**	**Wing Primitive**	**#Appearances**	**Frequency**
1	115	0.359	1	147	0.459
2	303	0.947	2	320	1.000
3	259	0.809	3	145	0.453
4	275	0.859	4	210	0.656
5	27	0.084	5	1	0.003
6	127	0.397	6	181	0.566
7	64	0.200	7	19	0.059
8	72	0.225	8	11	0.034
9	294	0.919	9	155	0.484
10	212	0.663	10	124	0.387
11	256	0.800	11	76	0.237
12	182	0.569	12	42	0.131
14	115	0.359	14	5	0.016
15	119	0.372	15	66	0.206
16	101	0.316	16	6	0.019
17	17	0.053	19	268	0.837
18	25	0.078	20	1	0.003
19	278	0.869	21	10	0.031
20	120	0.375	22	4	0.013
21	175	0.547	24	15	0.047
22	160	0.500	25	38	0.119
24	158	0.494	27	9	0.028
29	15	0.047	32	42	0.131
30	3	0.009	-	-	-
31	104	0.325	-	-	-
32	54	0.169	-	-	-
34	43	0.134	-	-	-

Table 5.1. The lists of distinct wing primitives and their appearance frequencies in 320 viewpoints using the map resolution 96 × 96 for the cobra head and the coffee cup respectively.

coffee cup. Its appearance frequency is 0.459. Among distinct wing primitives, some have high appearance frequencies. There are two ways to define the set of distinguished wing primitives for a model. Either the primitives with highest appearance frequencies or the primitives with frequencies larger than a pre-defined threshold can be used as the distinguished primitives of a model. For example, suppose we select wing primitives with appearance frequencies larger than 0.6 as the distinguished wing primitive. Then the set of distinguished wing primitives of the cobra head is {2, 3, 4, 9, 10, 11, 19} and {2, 4, 19} for the coffee cup. On the other hand, if we select the first three primitives with higher frequencies as distinguished wing primitives, then the cobra head has the distinguished wing set {2, 9, 19} and the coffee cup {2, 4, 19}.

According to the results in Table 5.1 which only show distinct wing primitives, appearance frequencies, and distinguished wing primitives of the cobra head and the coffee cup, the computed similarity value based on these results and a set of criterion weights {1, 2, 3, ····} is about 0.258. In the next section, we will show that their numbers of aspects are different too, and the complete set of criteria produces a similarity value for the object representations that should be close to zero.

5.2 Stability

The property of stability demands that small variations in the input data have little influence on the final results. There are several sources of variations which can affect the resulting wing representations. The first variation comes from using different tessellations of the viewsphere, called the variation of *view resolution*. To study its effect, we use two viewspheres with 240 and 320 viewpoints respectively. In Section 5.2.1, experimental results from the new sphere will be compared with the results in Table 5.1 and 5.2.

The second variation is due to using different map resolutions related to scale space for the algorithms of model augmentation and view generation. A map here denotes a two dimensional array of values. Therefore, range images are a kind of map with depth values, while edge images are another kind of map with edge labels. There are other maps such as characteristic maps with values such as curvature, gradient, etc.. The stability of algorithms can be drastically affected by map resolution. Although using a high resolution map to construct object representations can preserve details of objects, the significant structures of objects may be vague as a result of too many details.

There is another question about what happens with a perturbation of a model with respect to a viewpoint by a small angle before the model is projected. If an object representation is sensitive to view perturbations, a viewsphere with fine tessellation will be needed. We refer to such perturbations as *view variations* and will be explored in section 5.2.3.

Finally, a variation due to using different sizes of convolution filters may be the most important one which exists in many algorithms ranging from smoothing, feature extraction, to computations of characteristic values. The effect of mask size significantly influences the constructed wing representation. A comprehensive investigation into this variation is a nontrivial task. We will present a general model for estimating the computational variations due to using different convolution masks in Section 5.2.4, and give examples of edge detection using LOG filters [Nal86] and smoothing using Gaussian filters [Low88] to illustrate use of this model.

5.2.1 View Resolution

In order to study the influences of using different tessellations of viewspheres on the resulting representations, two viewspheres (with 240 and 320 viewpoints respectively) have been constructed (Appendix B).

Table 5.2 shows the results obtained by repeating the experiments of uniqueness in the last section using the viewsphere with 240 viewpoints and map resolution of 128 × 128. Recall that the results shown in Table 5.1 were obtained using the viewsphere with 320 viewpoints and map resolution of 96 × 96. In these two tables, both view and map resolutions are different. If we compare these two tables by computing their sum d of squared difference of appearance frequencies f_i,

$$d = [\sum_{i=1}^{34} (f_i^1 - f_i^2)^2]^{1/2}, \tag{5.1}$$

and d turns out to be 0.588 for the cobra head and 0.494 for the coffee cup. Note that the wing catalogue has 34 wing primitives. If a primitive doesn't appear in an object model, the appearance frequency of this primitive is assumed zero. Later, we will compare

Viewsphere of 240 Viewpoints, Map Resolution 128 × 128					
Cobra Head (27 distinct wing primitives)			**Coffee Cup (22 distinct wing primitives)**		
Wing Primitive	**#Appearances**	**Frequency**	**Wing Primitive**	**#Appearances**	**Frequency**
1	47	0.196	1	116	0.483
2	157	0.654	2	240	1.000
3	176	0.733	3	53	0.221
4	182	0.758	4	187	0.779
5	18	0.075	5	1	0.004
6	78	0.325	6	135	0.562
7	37	0.154	7	10	0.042
8	52	0.217	8	11	0.046
9	222	0.925	9	122	0.508
10	181	0.754	10	98	0.408
11	186	0.775	11	116	0.483
12	163	0.679	12	41	0.171
14	119	0.496	14	11	0.046
15	89	0.371	15	52	0.217
16	79	0.329	16	8	0.033
17	19	0.079	19	214	0.892
18	34	0.142	21	34	0.142
19	222	0.925	22	5	0.021
20	132	0.550	24	84	0.350
21	155	0.646	25	26	0.108
22	169	0.704	27	4	0.017
24	164	0.683	32	44	0.183
29	31	0.129	-	-	-
30	1	0.004	-	-	-
31	96	0.400	-	-	-
32	56	0.233	-	-	-
34	72	0.300	-	-	-

Table 5.2. Using the viewsphere with 240 viewpoints.

another results (in Table 5.3) obtained using different resolutions.

At present, from the computed *d* values, what we can say is that Table 5.1 and 5.2 are not well consistent with each other. Some frequencies even differ a lot. For example, the frequency of the wing primitive #2 of the cobra head in Table 5.1 is 0.947, whereas it is 0.654 in Table 5.2. We don't know which resolution (view or map or both) causes this difference. On the other hand, if we sort wing primitives according to their appearance frequences, the first few wing primitives (11 for the cobra head and any number for the coffee cup) form a set which is the same for both tables. This reveals that distinguished wing primitives have better stability than that of distinct wing primitives with respect to view and map resolutions.

5.2.2 Map Resolution

In many situations, map resolution is a key factor that can affect the representation results. Although using a high resolution map to construct object representations can preserve details of objects, the significant structures of objects may be vague as a result of too many details. For this reason, it seems desirable that objects with different shape complexities have different map resolutions for their representations. The problems are how can we measure the shape complexity of an object and what is the relationship between shape complexity and map resolution. Is it appropriate to use hierarchical scales to represent objects? This section shows some experimental results for the cobra head and the coffee cup using different map resolutions.

Table 5.3 shows the results using map resolution of 128 × 128 and the viewsphere with 320 viewpoints. Computing d values (Equation 5.1) for Table 5.3 and 5.1, we obtain 0.575 for the cobra head and 0.447 for the coffee cup. These results are close to the results (0.588 and 0.494) of Table 5.1 and 5.2. It means that the map resolution is an important factor which affects object representations. Furthermore, if we compute d values for Table 5.3 and 5.2, we obtain 0.137 and 0.095 respectively. These results indicate that the view resolution has a little influence on the representations. We conclude that using different view resolutions hasn't led to prominent differences in final representations but using different map resolutions may have.

The number of distinct wing primitives for the cobra head and the coffee cup are listed in Table 5.4 along with the map resolution used to create the wing representations. In the experiment, different map resolutions (from 32 × 32 to 128 × 128) have been used to investigate the effects of map resolution. As shown in the table, for the case of 320 viewpoints where using map resolutions larger than 80 × 80, both objects have stable sets

Viewsphere of 320 Viewpoints, Map resolution 128 × 128					
Cobra Head (27 distinct wing primitives)			Coffee Cup (23 distinct wing primitives)		
Wing Primitive	#Appearances	Frequency	Wing Primitive	#Appearances	Frequency
1	49	0.153	1	152	0.475
2	212	0.663	2	320	1.000
3	234	0.731	3	76	0.237
4	239	0.747	4	250	0.781
5	21	0.066	5	1	0.003
6	101	0.316	6	179	0.559
7	60	0.188	7	18	0.056
8	62	0.194	8	21	0.066
9	293	0.916	9	167	0.522
10	238	0.744	10	127	0.397
11	263	0.822	11	152	0.475
12	200	0.625	12	57	0.178
14	144	0.450	14	21	0.066
15	118	0.369	15	66	0.206
16	108	0.338	16	8	0.025
17	21	0.066	19	290	0.906
18	38	0.119	20	1	0.003
19	297	0.928	21	47	0.147
20	168	0.525	22	2	0.006
21	198	0.619	24	90	0.281
22	234	0.731	25	39	0.122
24	228	0.712	27	11	0.034
29	34	0.106	32	47	0.147
30	2	0.006	-	-	-
31	134	0.419	-	-	-
32	69	0.216	-	-	-
34	80	0.250	-	-	-

Table 5.3. Using the map resolution 128 × 128.

of distinct wing primitives. The cobra head possesses 27 (the same for the case of 240 viewpoints) out of 34 wing primitives, whereas the coffee cup has 23 (22 for the case of 240 viewpoints). The variation of distinct wing set due to different map and view resolutions is depicted in Figure 5.1. Note that the handle of the coffee cup changes the shape regularity of the object and makes the global shape of the coffee cup almost as complex as that of the cobra head. In addition, the size of the cup handle is so small with respect to the whole object. Wing features located on the cup handle are small too. Using either a map with low resolution or a viewsphere with coarse tessellation, these wing features can easily be discarded. The high variation shown in Figure 5.1 for the coffee cup reveals this fact. Figure 5.1 also shows that the two curves of the cobra head completely overlapped each other. It means that although the cobra head has complicated shape, the shape complexity distributes uniformly over its surface.

	Viewsphere of 320 Viewpoints		Viewsphere of 240 Viewpoints	
Map Resolution	**Cobra Head**	**Coffee Cup**	**Cobra Head**	**Coffee Cup**
32 × 32	18	13	18	14
48 × 48	26	18	26	18
64 × 64	27	19	27	17
80 × 80	27	22	27	18
96 × 96	27	23	27	22
112 × 112	27	23	27	22
128 × 128	27	23	27	22

Table 5.4. Numbers of distinct wing primitives present in views of the cobra head and the coffee cup *versus* different map resolutions.

Table 5.5 shows the influence of view and map resolutions on the resulting aspects. According to this table, the number of aspects increases as the map resolution increases. This tendency is explicitly revealed in Figure 5.2, which indicates that after a map resolution of 64 × 64 the number of aspects of the cobra head is stable (arriving at 292 for 320 viewpoints and 224 for 240 viewpoints), while the coffee cup gradually attains stability (145 and 122) after resolution 96 × 96. Unfortunately, the numbers of aspects for both object are too high, violating the terseness criterion. As shown in Figure 5.2, to reduce the number of aspects, using low resolution maps (for example 32 × 32) and coarse tessellation viewspheres (for example 240 viewpoints) is preferred (122 aspects for the cobra head and 38 aspects for the coffee cup). A possible way to further reduce the number of aspects is to use weaker grouping conditions, for example using the distinguished wing sets instead of the distinct wing sets. Another method is to select notable representatives from the set of aspects [Bas88]. Then other aspects can be derived from these representatives as needed.

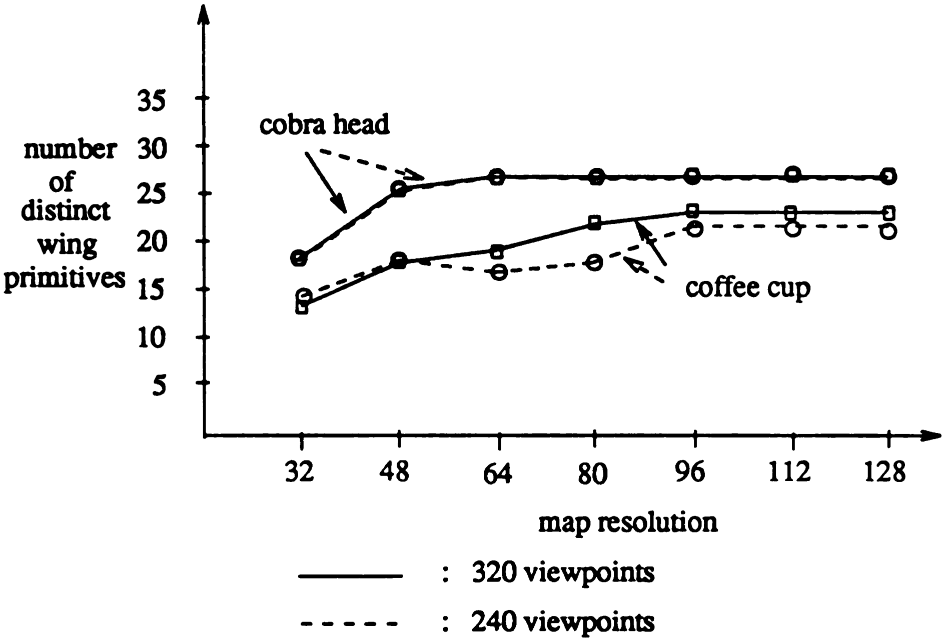


Figure 5.1. The variation of distinct wing set *versus* the map resolution.

	Viewsphere of 320 Viewpoints		Viewsphere of 240 Viewpoints	
Map Resolution	**Cobra Head**	**Coffee Cup**	**Cobra Head**	**Coffee Cup**
32 × 32	139	36	122	38
48 × 48	234	50	178	48
64 × 64	271	76	206	66
80 × 80	292	106	224	86
96 × 96	294	121	224	94
112 × 112	299	145	226	122
128 × 128	296	152	228	128

Table 5.5. Different numbers of aspects resulting from using different map resolutions.

From the wing maps of object views and the sets of wing primitives of individual aspects (not shown here) using a map resolution of 32 × 32 is already good enough for both the cobra head and the coffee cup because most salient structures of objects are preserved. Those discarded detailed wings are not only unnecessary but also harmful for the purpose of object recognition. For other applications, if the details of objects are so important, we suggest use of multiple or hierarchical resolutions to represent objects. A multiple resolution means that using different map resolutions for different parts of

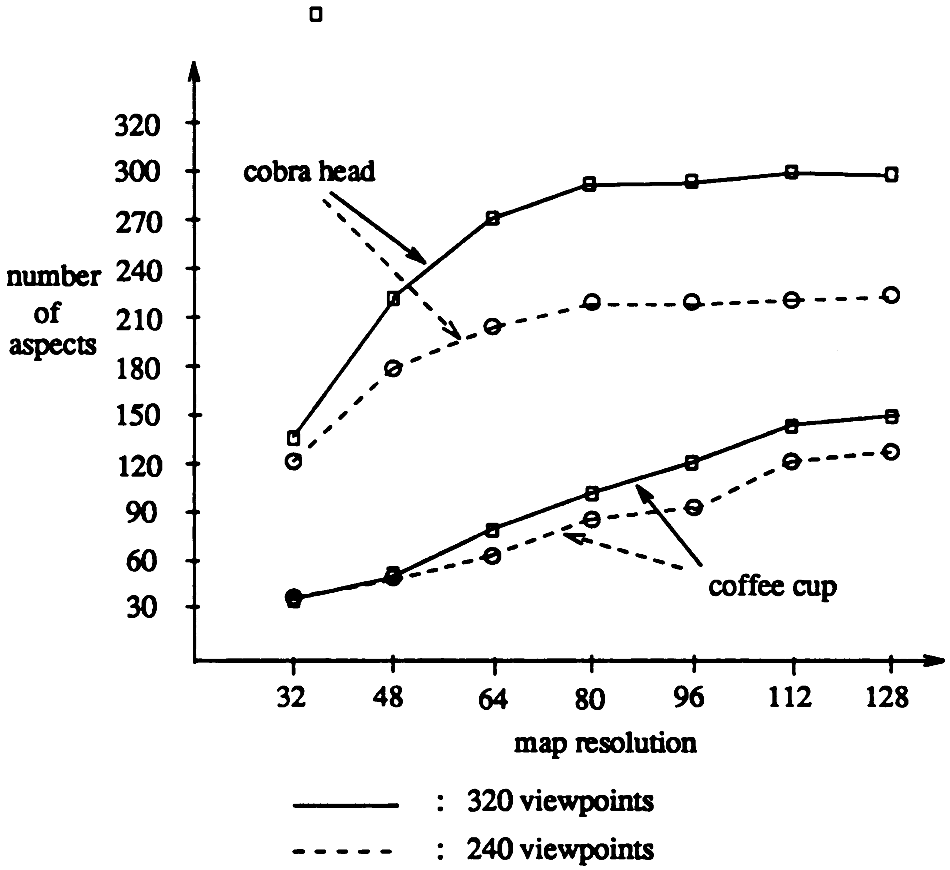


Figure 5.2. The stability of wing representations increases as the map resolution increases.

objects according to their shape complexity. A hierarchical resolution uses coarse-to-fine map resolutions for each view. We also suggest that a complicated object would better use low resolution maps in order to get rid of noisy details, whereas objects with simple shapes can use any map resolution.

Although using a viewsphere with coarse tessellation can result in fewer aspects and better stability than that of using viewspheres with finer tessellations, the constructed representations should be able to tolerate large view range and view perturbation; otherwise some crucial object views may be ignored in the final representations. This problem will be investigated in the next section.

5.2.3 View Variation

As mentioned in Chapter 3, there are an infinite number of viewpoints on the continuous viewsphere. In practice, one would better discretize it. Then by grouping, representative views are determined so that a representation can become more compact. It is desirable that in the beginning the viewsphere is uniformly tessellated into discrete viewpoints. This immediately leads to a question of what angle intervals between viewpoints is adequate. Large intervals may lose important views of an object, while small view intervals require a large amount of storage. Understanding the effect of view variations may provide some clues to this question.

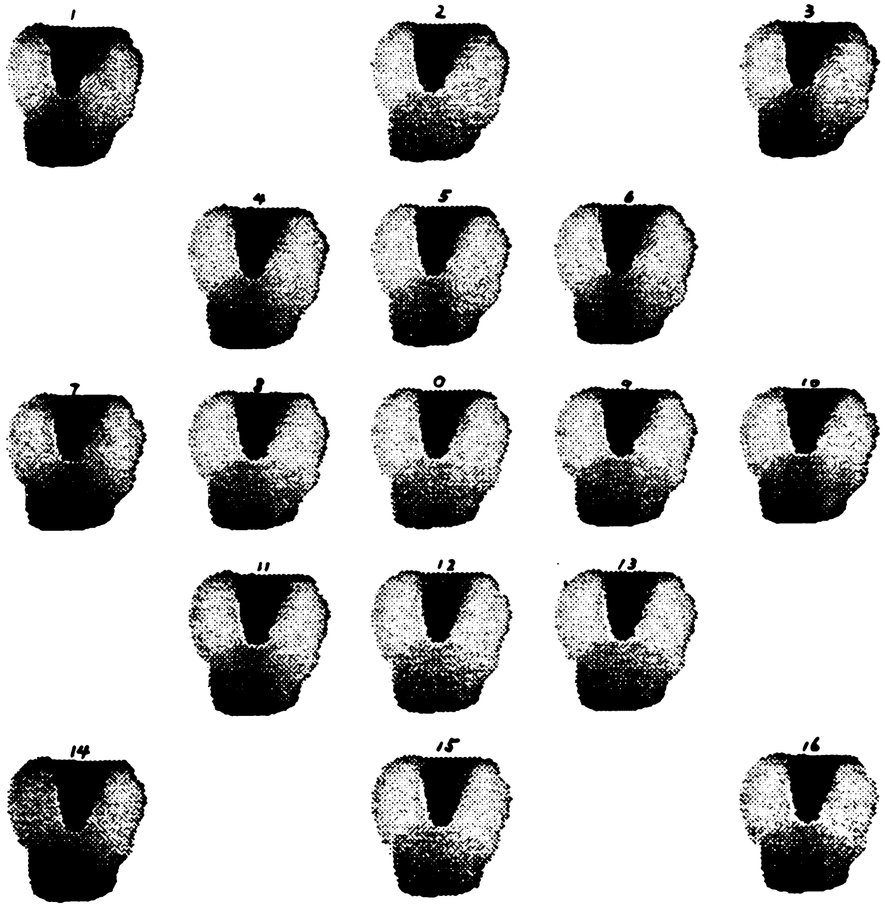


Figure 5.3. 17 views of the cobra head with each pair of adjacent views differing by an angle of 3 degrees. The image resolution is 128 × 128.

17 Views of the Cobra Head, Image Resolution 128 × 128		
View	**#Wing Primitives**	**Wing List**
0	10	3 4 9 10 11 19 20 22 24 34
1	13	3 4 9 10 11 14 19 20 21 22 24 31 34
2	11	3 4 9 10 11 19 21 22 24 31 34
3	12	2 3 4 9 10 11 19 20 21 22 24 34
4	14	2 3 4 9 10 11 19 20 21 22 24 31 32 34
5	11	3 4 9 10 11 19 20 21 22 24 31
6	12	3 4 9 10 11 19 20 21 22 24 32 34
7	11	2 3 4 9 10 11 19 22 24 31 32
8	11	4 9 10 11 19 20 21 22 24 32 34
9	11	4 9 10 11 19 20 21 22 24 32 34
10	11	2 3 4 9 10 11 19 22 24 31 32
11	13	2 3 4 9 10 11 19 20 21 22 24 31 34
12	12	4 9 10 11 12 19 20 21 22 24 32 34
13	12	2 3 4 9 10 11 19 20 21 22 24 34
14	11	2 3 4 9 10 11 12 19 20 22 24
15	11	4 9 10 11 19 20 21 22 24 31 32
16	12	2 3 4 9 10 11 14 19 20 21 22 24

Table 5.6. The lists of distinct wing primitives of 17 views of the cobra head in Figure 5.3.

As mentioned in Chapter 4, for the viewsphere of 320 viewpoints, the solid angle of a viewpoint (or the view resolution) is about 0.039 steradians (the angle interval between two adjacent viewpoints is about 12.5 degrees). The solid angle of a viewpoint is 0.052 steradians (angle interval about 14.5 degrees) for the case of 240 viewpoints. It is desirable that most wing features detected in a viewpoint can also be detected in the adjacent viewpoints so that viewpoints with highly overlapped wing sets can be grouped. Through studying the impact of view variation, we hope to discover some useful criteria for view grouping.

In order to investigate the effect of view variations, a set of model views of the cobra head (refer to Figure 5.3) was constructed. Then, the representation procedure was

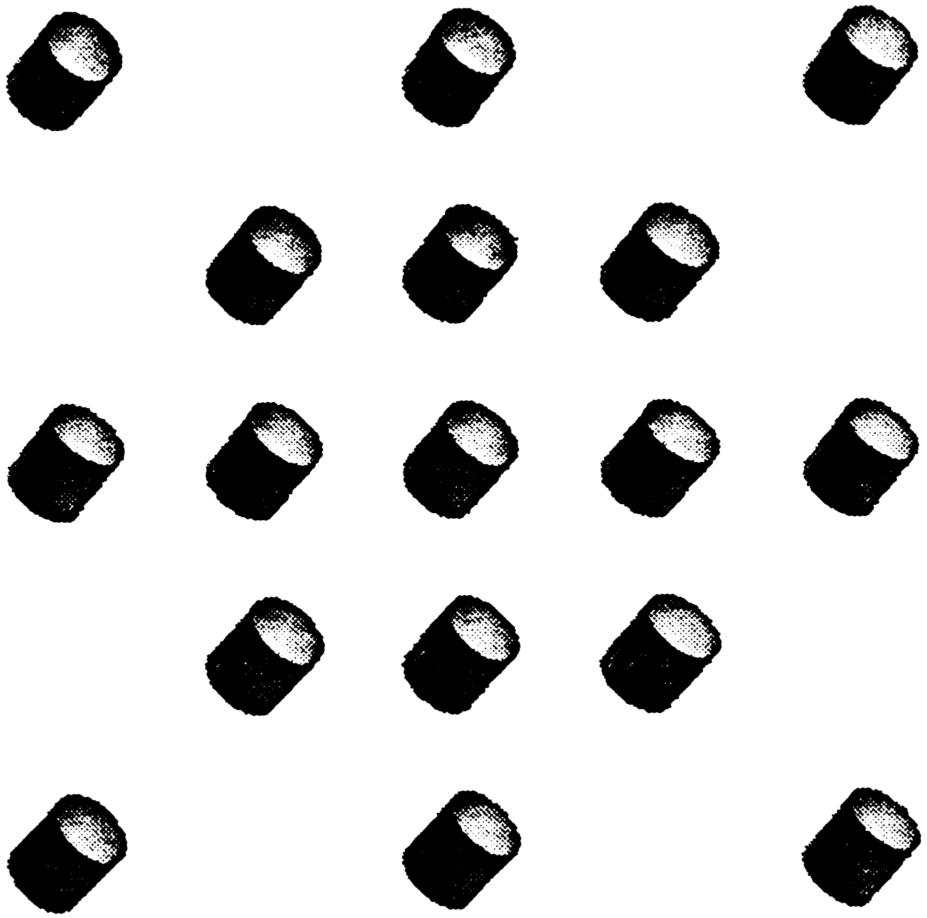


Figure 5.4. 17 views of the coffee cup with each pair of adjacent views differing by an angle of 3 degrees. The image resolution is 128 × 128.

applied to these views, and their resulting list of wing primitives was examined. Each pair of adjacent (including diagonal) views in the figure differs by an angle of 3 degrees. Therefore, the largest angle variation in this test will be 12 degrees, e.g. the angle between view 1 and view 16.

The sets of distinct wing primitives of views are listed in Table 5.6 in which three pairs of views (views 7 and 10, views 8 and 9, views 3 and 13) have exactly the same sets of wing primitives. Most other views have highly overlapped wing lists. The common overlap among wing lists is {4, 9, 10, 11, 19, 22, 24}. Although the best situation is that all views within a small angle variation have the same wing set, we must remember that

17 Views of the Coffee Cup, Image Resolution 128 × 128		
View	**#Wing Primitives**	**Wing List**
0	7	2 4 9 10 11 12 19
1	8	2 4 9 10 11 12 19 32
2	9	2 4 9 10 11 12 19 21 32
3	7	2 4 9 10 11 12 19
4	8	2 4 9 10 11 12 19 32
5	8	2 4 9 10 11 12 19 21
6	7	2 4 9 10 11 12 19
7	6	2 4 9 10 12 19
8	7	2 4 9 10 11 19 21
9	7	2 4 9 10 11 19 21
10	6	2 4 9 10 12 19
11	8	2 4 9 10 11 12 19 22
12	7	2 4 9 10 12 19 32
13	9	2 4 9 10 11 12 19 22 32
14	7	2 4 9 10 11 19 32
15	6	2 4 9 10 11 19
16	6	2 4 9 10 12 19

Table 5.7. The lists of distinct wing primitives of 16 views of the coffee cup in Figure 5.4.

wing sets must vary between some neighbors of the discrete viewsphere if there are several aspects. The situation is much better for the coffee cup (refer to Figure 5.4 and Table 5.7) where more views have the same wing sets (view 0, 3 and 6, view 1 and 4, view 7, 10 and 16, view 8 and 9) and views have more stable common wing list.

In order to further test the effect of view variation using larger angle perturbations, an experiment with angle perturbation range of 50 degrees (Figure 5.5) was performed. Each pair of views now differs by 12.5 degrees. The result is shown in Table 5.8. The common overlap among wing lists is now reduced {9, 10, 11, 19, 22}. We conclude that both distinguished and overlapping wing sets can be used to further reduce the number of aspects.

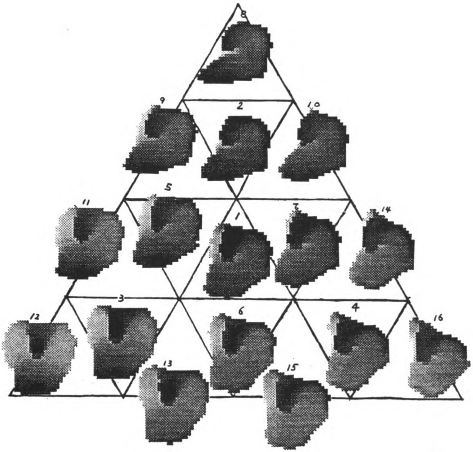


Figure 5.5. 16 views of the cobra head with each pair of adjacent views differing by an angle about 12.5 degrees.

5.2.4 Convolution Masks

In this section, we will consider another factor that can affect the stability of constructed representations. Many image processing procedures ranging from data smoothing and edge detection to computations of surface characteristics can be implemented in terms of convolutions of masks (or filters). Such convolutions have a problem that the input to a finite filter consists of the data points covered by the filter. Different sizes of filters produce different results. This difference is typically a function of the filter and the area covered by the filter. A small filter will suffer by excluding distant feature points from consideration. However, a large one suffers by covering points coming from different features. The latter is sometimes referred to as the problem of *multiple features*

16 Views of the Cobra Head, Image Resolution 128 × 128		
View	#Wing Primitives	Wing List
1	12	3 4 9 10 11 12 14 19 20 22 24 34
2	12	2 3 4 9 10 11 12 14 19 21 22 31
3	11	2 3 4 9 10 11 12 19 20 22 24
4	11	2 3 4 9 10 11 12 19 22 31 34
5	13	3 4 9 10 11 12 14 19 20 22 24 31 34
6	11	2 3 4 9 10 11 14 19 22 24 34
7	11	3 4 9 10 11 12 19 22 24 31 34
8	14	2 3 4 9 10 11 12 14 19 21 22 30 31 32
9	11	4 9 10 11 14 19 21 22 24 31 34
10	12	2 3 9 10 11 12 14 19 22 29 31 34
11	13	2 3 4 9 10 11 19 20 21 22 24 31 34
12	10	3 4 9 10 11 19 20 22 24 34
13	13	2 3 4 9 10 11 12 19 20 21 22 24 34
14	11	3 4 9 10 11 12 14 19 22 31 34
15	12	2 3 4 9 10 11 12 19 22 24 31 34
16	14	2 3 4 9 10 11 12 14 19 20 21 22 24 31

Table 5.8. The lists of distinct wing primitives of 16 views of the cobra head in Figure 5.5.

[Cal88]. Clearly, both situations can considerably degrade the performance of convolution and there is a tradeoff between the two.

To handle the above problem, researchers usually employ a set of multiple-scaled filters. Different areas are convolved with different scales of filters. This approach has two difficulties: either the final result must be obtained through a complicated integration process or the optimal filter must be determined *a priori*. Another method [Cal88] suggests that small, possibly overlapping, neighborhoods of fixed size can be used, which are chosen such that the entire image is covered. However, the problem of what fixed size of covering areas to use and how to cover the entire image (partitioning or overlapping), still exists. Regardless of these issues, both approaches have a common problem that using different sizes of filter mask produces different results. This problem is also encountered in the construction of representations.

We present a general model to evaluate computational differences due to using different filter sizes. Examples of edge detection using LOG filters and smoothing using Gaussian filters will be given later to illustrate use of the model. Let $\mathbf{w}_1$ and $\mathbf{w}_2$ be two square masks with sizes differing by $2n$, i.e. $|\mathbf{w}_2| - |\mathbf{w}_1| = 2n$, where n can be any integer. According to Brady *et al* [Bra85], an n times repeated convolution of a filter is approximately equivalent to convolving the data points with a Gaussian mask g whose standard deviation is proportional to $\sqrt{n}$. Therefore, we can write $\mathbf{w}_2 = \mathbf{w}_1 * \mathbf{g}^n$, where '*' represents the convolution operator and $\mathbf{g}^n$ means an n-fold repetition of g. The 3 × 3 Gaussian mask g is

$$\mathbf{g} = \frac{1}{24}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 12 & 2 \\ 1 & 2 & 1 \end{bmatrix}.$$

Let A_1 and A_2 denote the areas covered by the filters respectively. Then, the difference in computations at an image point (x,y) or map resulting from convolutions of these two filters with their corresponding areas becomes:

$$\begin{aligned} \Delta(x,y) &= (A_2 * \mathbf{w}_2)(x,y) - (A_1 * \mathbf{w}_1)(x,y) = A_2 * \mathbf{w}_2 - A_1 * \mathbf{w}_1 \\ &= A_2 * (\mathbf{w}_1 * \mathbf{g}^n) - A_1 * \mathbf{w}_1 = (A_1 + A_2') * (\mathbf{w}_1 * \mathbf{g}^n) - A_1 * \mathbf{w}_1 \\ &= A_1 * (\mathbf{w}_1 * \mathbf{g}^n) + A_2' * (\mathbf{w}_1 * \mathbf{g}^n) - A_1 * \mathbf{w}_1 \qquad (5.2) \\ &= A_1 * \mathbf{w}_1 * (\mathbf{g}^n - \mathbf{I}) + A_2' * (\mathbf{w}_1 * \mathbf{g}^n)\,, \end{aligned}$$

where the variables x and y have been dropped for conciseness. We assume that the sizes of smaller masks have been enhanced by filling zeros around their circumferences when necessary. I represents the unit impulse mask. A_2' (finite support) is the area difference between A_1 and A_2, i.e. $A_2' = A_2 - A_1$. From this equation, the computational difference can be simply determined from the smaller mask $\mathbf{w}_1$ and the larger area A_2. Note that filters $\mathbf{w}_1$ and $\mathbf{w}_2$ are task-dependent. Before the closed form of Equation 5.2 can be derived, filters must be specified.

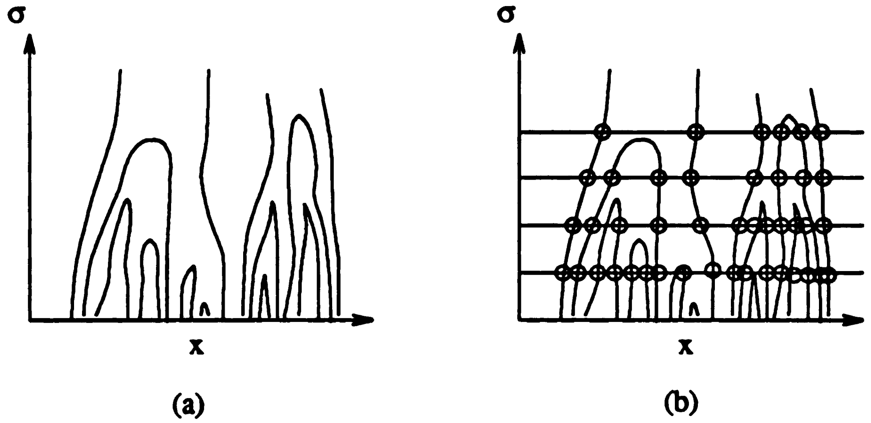


Figure 5.6. (a) A continuous scale space, (b) a discrete scale space.

Let us consider a familiar example of looking for zero crossings from a convolution of the Laplacian of Gaussian (LOG) operator and a range image. It has been shown that there are displacements of zero crossings using different filter scales [Asa86]. Figure 5.6 shows a continuous scale space and a discrete scale space. Much noise can be suppressed using coarse scales whereas better location of depth discontinuities can be achieved using fine scales. Theoretically, an edge map with precise locations of discontinuity and less noise can be obtained by tracking contours in the scale space from coarse to fine scales (Figure 5.6(a)). In practice, only a few discrete scales are used (Figure 5.6(b)), which make the tracking operation difficult.

The following equation derived by Nalwa & Binford [Nal86] for an 1-D step edge indicates the displacements of zero crossings using two filters with different scales σ_1 and σ_2. In practice, one implements a discrete LOG operator using a fixed width w such that $\sigma = \frac{w}{c}$. Therefore, different widths of the mask will lead to distinct scales σ's and in turn different results in distinct LOG filters. In view of the truncation and discretization of the LOG function, zero crossings (which are usually regarded as the positions of edges [Gri86]) are shifted from their actual positions. Moreover, such shifts differ from filter to filter.

Let $\mathbf{w}_1$ and $\mathbf{w}_2$ denote a small and a large filters respectively and w_1 and w_2 be their widths. Let the step function be

$$f(x, y) = \begin{cases} az_1 + d & x \geq 0 \\ bz_2 & x < 0 \end{cases},$$

where z_1 and z_2 are larger than 0 and $z_1 \neq z_2$. The displacement turns out to be

$$\Delta = \frac{b-a}{c^2 d}(w_2^2 - w_1^2). \tag{5.3}$$

The parameters of slopes a and b and step-size d can be estimated from range values. To be more explicit about the use of Equation (5.3), let us refer to the range image in Figure 5.7(a), which is a real image of the cobra head taken using the ERIM range finder. Figure 5.7(b), (c), (d) and (e) show maps of zero crossings derived using sizes 11, 13, 15 and 17 of the LOG filter respectively. The registered result of these zero-crossing maps based on the displacement equation (5.3) is shown in Figure 5.7(f).

Lowe [Low88] has used a similar concept to recover shrinkage error due to convolutions of the Gaussian smoothing filter and detected contours. The amount X of shrinkage at a point is a function of Gaussian scale σ and curve radius r at that point.

$$X = r(1 - e^{\frac{-\sigma^2}{2r^2}}),$$

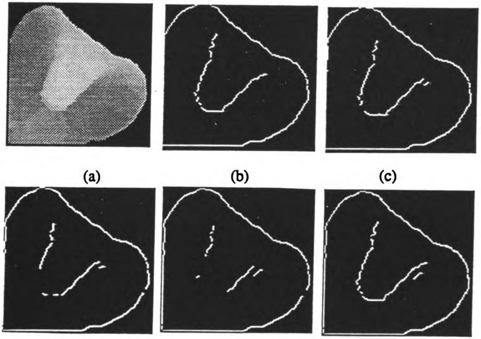
where r is estimated by the measured second derivative of the smoothed curve X''

$$X'' = e^{\frac{-\sigma^2}{2r^2}} / r.$$

Point coordinates of the smoothed curve are then interpolated by the computed shrinkage errors.

5.3 Compatibility of Wing Features

It is essential that object wings extracted from real images should be compatible with those of internal representations. Otherwise, the recognition procedure will not succeed in identifying scene objects based on the representations. Our discussion in the following few sections doesn't attempt to present a complete procedure for wing detection. Instead, it only demonstrates that wings detected from real images can be



(a) (b) (c)

(d) (e) (f)

Figure 5.7 (a) A range image of the cobra head taken using the ERIM range finder. (b),(c),(d),(e) Maps of Zero crossings derived using sizes 11, 13, 15 and 17 of the LOG filter respectively. (f) The registered map of zero crossings using Equation 5.3.

comparable to wings of internal models.

Although many *ad hoc* techniques [Bes85, Lau86, Hof87, Fan87, Par87] of surface and curve detection have been developed, none of them is completely satisfactory. To date, the problem of feature extraction is still nontrivial. Unlike wing detection with synthetic images used for generating object representations, which is a top-down process guided by geometric models, wing detection in real images can only use the sensory data. Without knowledge of models, computations of surface characteristics have to rely on the available data and general domain assumptions.

Current segmentation techniques developed for range images include methods which are edge-based [Lau86, Fan87, Par87] or region-based [Hof87, Bes88]. In the region-based approach, characteristic values are computed through mask convolutions. The pixels close to object boundaries (within a distance about half width of a mask) will result in defective evaluations of characteristics, while the ones far away from boundaries have better results. Thus, a region growing [Bes88] procedure using the central areas of regions as the seeds is necessary to compensate for this defect. Object boundaries are then derived by intersecting the growing regions. Edges detected in this way are usually not perfect. A local analysis [Hof87] of the acquired edges can then be applied to verify them.

On the other hand, the edge-based approaches directly look for object boundaries based on surface normal, slope and curvature, which seems simpler than region-based approaches. Unfortunately, the same problem of computing normals and curvatures is also encountered by the edge-based techniques. To mediate, masks with different scales and oriented preferences [Lau86, Fan87, Par87] are adopted. Edges derived in this way will be further modified using their contextual surface information.

Line-drawings [Tur74, Wal75, Mal87, Cae82, Nal88] and shape from contours [Bin81, Bra85, Koe84, Hu88] have been systematically investigated. The relations between curves and surfaces are becoming clearer through a substantial number of inference rules. These rules are useful for deducing 3-D shapes near curve segments. As a result, inference rules can compensate for some problems with segmentation techniques. The results of segmentation can be verified using these rules.

5.4 Wing Detection from Real Images

Instead of creating new procedures, we mostly use available techniques to carry out partial experiments in wing detection including the methods adopted from Hoffman & Jain [Hof87], Besl & Jain [Bes86], Laurendeau & Poussart [Lau86] and Marr & Hildreth [Mar80]. A primary purpose in this section is to look for feasible approaches to extracting wing features in real images. Feasible approaches should be easy (few thresholds involved), efficient (fast processing) and reliable (little variation with respect to mask size used). On the other hand, they need not be perfect. We can tolerate incomplete and even fallacious wings existing in the final results. The recognition procedure to be discussed in the next chapter should be capable of tolerating these defects. In the wing detection experiment, a number of real images of the cobra head, coffee cup and polyhedral object have been used. Figure 5.8 shows these range images displayed in gray scales. They are taken using the ERIM sensor. Two alternate approaches to feature extraction have been tried in the experiment and are presented in the following.

5.4.1 Hoffman-Jain Technique

The first strategy includes steps of segmentation, jump edge detection and region unification. The Hoffman & Jain technique [Hof87] has been used to segment the range images. The results of segmentation are shown in Figure 5.9. It reveals that for some regions there are too many small patches involved: images are oversegmented. However, except for the triangular region at the right in the last segmented image of the polyhedral object, there is no other patch in the images across the physical edges of objects. This is good since one merging procedure will be enough to obtain most significant regions which are consistent with object surfaces. However, there is another difficulty when using the present result of segmentation to extract wing features especially from nonpolyhedral objects. The boundaries between patches are ragged, which eventually will result in many small wings which will be thrown away because of size or infrequent occurrence. Here, instead of solving the problems of over segmentation and raggedness at once, we will simply detect the silhouettes and jump edges in the images, then apply a unification procedure to the segmented images.

Contours are extracted using both the LOG operator mentioned in the last section and Laurendeau & Paussart's approach [Lau86]. The jump edges and silhouettes of images are shown in Figure 5.10. Afterwards, a symbolic uniform procedure repeatedly groups pixels into the most likely neighboring region according to the majority of eight neighbors of pixels. The neighbors of each pixel may be bounded by contours so that neighboring pixels on the other side of a contour will not be considered.

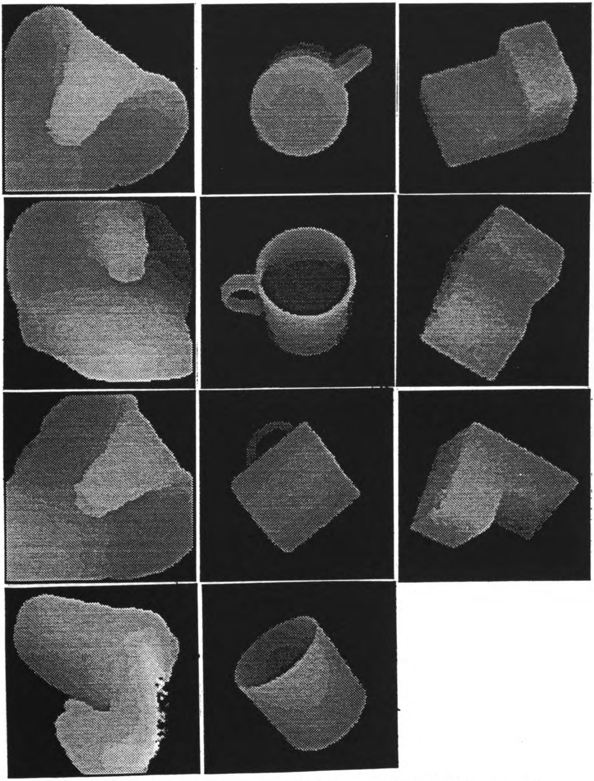


Figure 5.8. Experimental real images of the clay cobra head, coffee cup and polyhedral object taken using the ERIM sensor.

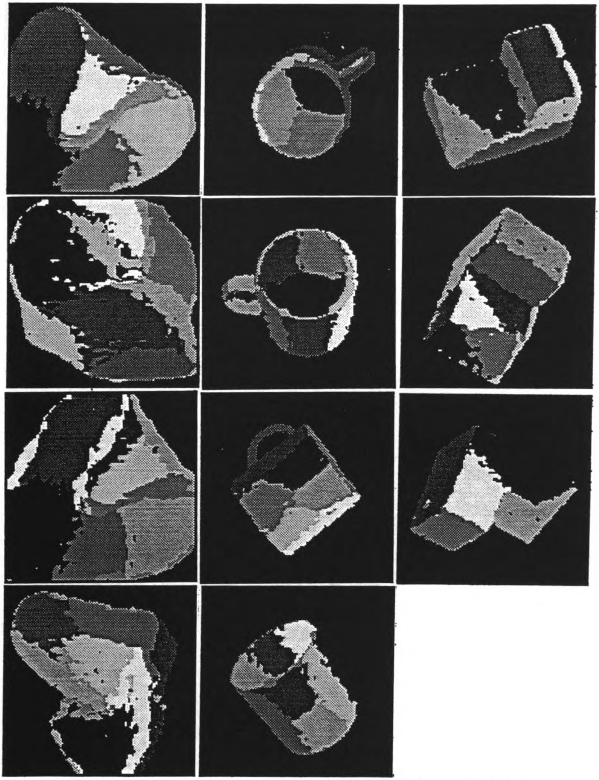


Figure 5.9. Segmentation results using Hoffman & Jain's technique.

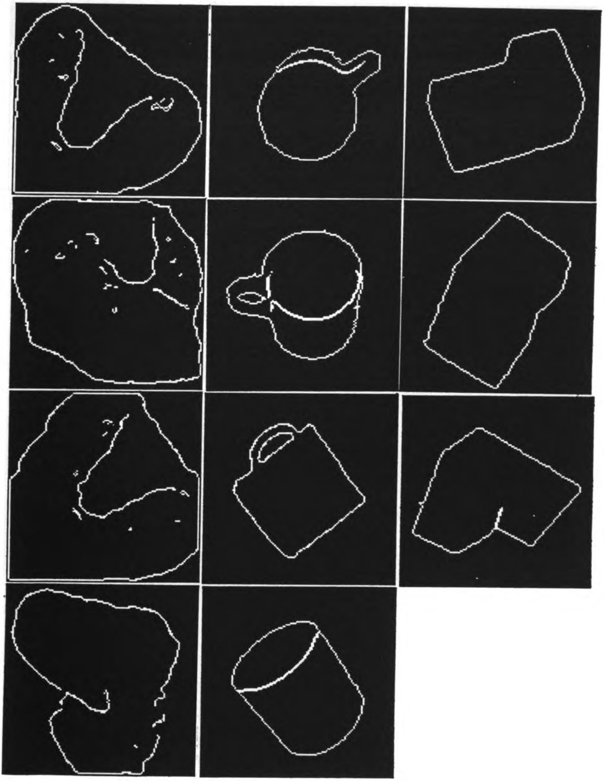


Figure 5.10. Jump edges of images.

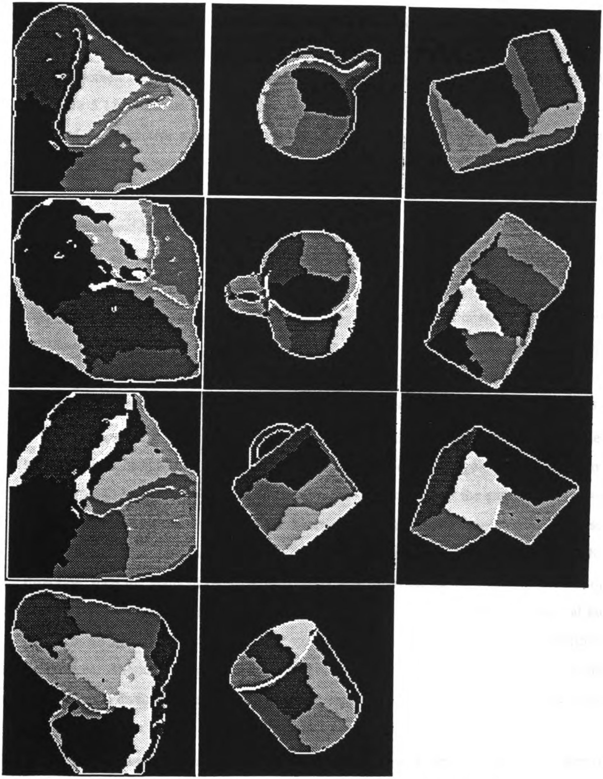


Figure 5.11. The results of the first segmentation method.

Figure 5.11 shows the results of the segmentation imposed by contours. Comparing the results with those in Figure 5.9, it is clear that the boundaries between patches have been smoothed and most noise pixels within patches have been removed. In the images in Figure 5.11, there are a lot of internal edges which are boundaries between patches. Some of them are true crease edges, whereas some are spurious edges. Spurious edges will increase the recognition time but should not affect the recognition results.

5.4.2 Combined Signs of Directional Derivatives

In this section, feature extraction based on directional derivatives is studied. A directional derivative of a function at a point is defined as the first derivative of the function in a certain direction at that point. Clearly, unlike curvatures, directional derivatives are view-variant. A reason that we pay attention on directional derivatives instead of curvatures is that wing primitives can be view-variant. Another reason is that directional derivatives are more noise-resistant than curvatures which depend on second derivatives.

The method is based on the property of range values which represent distances between the sensor and scene points. Suppose there is a range image of an infinite plane whose normal is not perpendicular to the optical axis. If one scans the entire image along a direction (for example the horizontal direction), then the range values either increase, decrease or remain the same. If the first derivatives of range values are computed in this direction, then the signs of the first derivatives will be all the same (either + : increasing range values, - : decreasing range values, or 0 : equal range values). This means that the pixels in an image corresponding to a spatial plane have the same sign of directional derivatives. Although different scanning directions may result in different signs for the plane, once a direction has been selected, the signs of directional derivative of the plane will be the same.

Suppose two intersecting half-planes are depicted in a range image and neither of these two planes is perpendicular to the image plane. If we scan this image along a given direction and compute the first derivatives in this direction, according to the above analysis of a single plane, the derivative signs of pixels of a plane will be all the same.

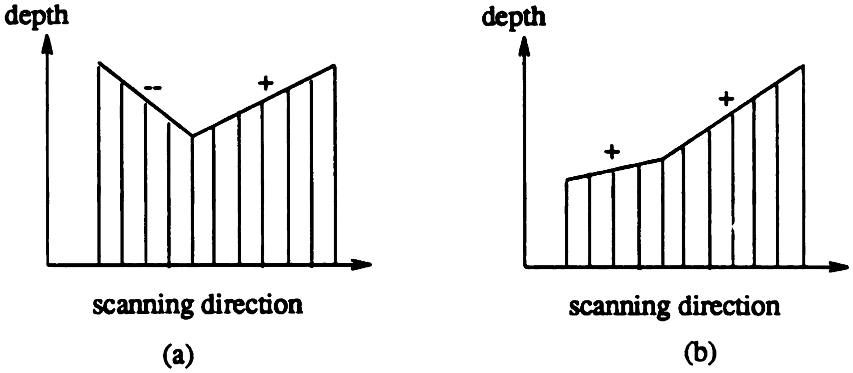


Figure 5.12. (a) Two planes have different signs of directional derivatives.
(b) Two planes have the same sign of directional derivatives.

However, it is possible that two planes in an image may have different signs of directional derivative (Figure 5.12(a)). Of course, both planes may also have the same sign (Figure 5.12(b)). In the first case (two planes having different signs of derivative) the intersection, or the boundary, of planes can be easily determined based on the derivative signs. For the second case (both planes having the same sign), an interesting question arises. Is there a scanning direction along which the computed signs of derivative of the planes can be different. The following lemma answers this question.

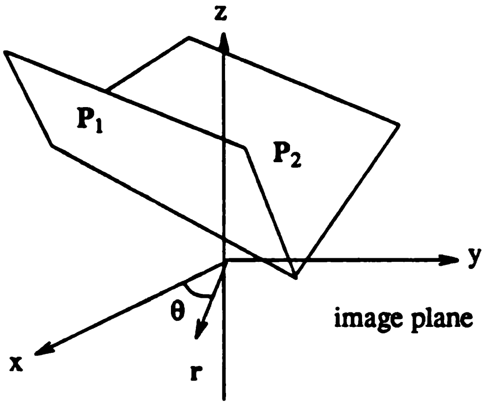


Figure 5.13. Two intersecting half-planes are depicted in a range image.
Neither of them is perpendicular to the image plane.

Refer to Figure 5.13. Let the image plane be the x-y plane. The equation of a plane in E^3 can be written as

$$a'x + b'y + c'z = d' .$$

If the plane is not perpendicular to the image plane, $c' \neq 0$, the plane equation can be simplified as

$$z = ax + by + d ,$$

where $a = -\frac{a'}{c'}$, $b = -\frac{b'}{c'}$, and $d = -\frac{d'}{c'}$.

Lemma 5.1: Suppose there are two intersecting half-planes P_1 and P_2 (as shown in Figure 5.13) such that neither P_1 nor P_2 is perpendicular to the image plane.

$$P_1\!: \ z_1 = a_1x + b_1y + d_1 , \quad P_2\!: \ z_2 = a_2x + b_2y + d_2 .$$

Then there is an angle range in the image plane within which any scanning direction $\mathbf{r}(\theta)$ can make the derivative signs of the planes distinguishable. The angle range is

$$\frac{\pi}{2} + \min(\theta_1, \theta_2) < \theta < \frac{\pi}{2} + \max(\theta_1, \theta_2) , \tag{5.4}$$

where

$$\theta_1 = \cos^{-1}\left(\frac{a_1}{(a_1^2+b_1^2)^{1/2}}\right) = \sin^{-1}\left(\frac{b_1}{(a_1^2+b_1^2)^{1/2}}\right),$$

$$\theta_2 = \cos^{-1}\left(\frac{a_2}{(a_2^2+b_2^2)^{1/2}}\right) = \sin^{-1}\left(\frac{b_2}{(a_2^2+b_2^2)^{1/2}}\right),$$

and $\theta_1 \neq \theta_2$.

Proof: Because the scanning direction $\mathbf{r}$ lies on the image plane, it can be represented by a unit vector $\mathbf{r} = (\cos\theta, \sin\theta, 0)$, where θ is the angle between the scanning direction and the x-axis. The directional derivative of the plane z in direction $\mathbf{r}$ is

$$\frac{dz}{d\mathbf{r}} = a\cos\theta + b\sin\theta .$$

For plane P_1:

$$\frac{dz_1}{d\mathbf{r}} = a_1 \cos\theta + b_1 \sin\theta = (a_1^2+b_1^2)^{1/2}(\frac{a_1}{(a_1^2+b_1^2)^{1/2}} \mathit{cos}\,\theta + \frac{b_1}{(a_1^2+b_1^2)^{1/2}} \mathit{sin}\,\theta)$$

$$= (a_1^2+b_1^2)^{1/2}(\mathit{cos}\,\theta_1\ \mathit{cos}\,\theta + \mathit{sin}\,\theta_1\ \mathit{sin}\,\theta) = (a_1^2+b_1^2)^{1/2}\cos(\theta - \theta_1)\,,$$

where θ_1 is the angle of the projected surface normal of plane 1 with x-axis in the image plane.

$$\theta_1 = \cos^{-1}(\frac{a_1}{(a_1^2+b_1^2)^{1/2}}) = \sin^{-1}(\frac{b_1}{(a_1^2+b_1^2)^{1/2}})$$

For plane P_2:

$$\frac{dz_2}{d\mathbf{r}} = a_2 \cos\theta + b_2 \sin\theta = (a_2^2+b_2^2)^{1/2}\cos(\theta - \theta_2)\,,$$

where θ_2 is the angle of the projected surface normal of plane 2 with x-axis in the image plane.

$$\theta_2 = \cos^{-1}(\frac{a_2}{(a_2^2+b_2^2)^{1/2}}) = \sin^{-1}(\frac{b_2}{(a_2^2+b_2^2)^{1/2}})$$

Due to the image formation, $0 < |\theta_1 - \theta_2| < \pi$. In order to make the derivative signs of planes distinct, $\frac{dz_1}{d\mathbf{r}} \cdot \frac{dz_2}{d\mathbf{r}} < 0$, or equivalently, $\cos(\theta - \theta_1)\cos(\theta - \theta_2) < 0$. By diagrammatical analysis, if

$$\frac{\pi}{2} + \min(\theta_1, \theta_2) < \theta < \frac{\pi}{2} + \max(\theta_1, \theta_2)\,, \tag{5.4}$$

then $\frac{dz_1}{d\mathbf{r}} \cdot \frac{dz_2}{d\mathbf{r}} < 0$. Note that $\theta_1 \neq \theta_2$; the range (5.4) always exists. For example, we can simply choose $\theta = \frac{\pi}{2} + \frac{\theta_1 + \theta_2}{2}$, which is the central value of the range.

$$\cos(\theta - \theta_1)\cos(\theta - \theta_2) = \cos(\frac{\pi}{2} + \frac{\theta_1 + \theta_2}{2} - \theta_1)\cos(\frac{\pi}{2} + \frac{\theta_1 + \theta_2}{2} - \theta_2)$$

$$= -\mathit{sin}\,(\frac{\theta_2 - \theta_1}{2})[-\mathit{sin}\,(\frac{\theta_1 - \theta_2}{2})] = -\mathit{sin}^2\,(\frac{\theta_2 - \theta_1}{2}) < 0$$

■

From the above lemma, we conclude that except for accidental cases (parallel and perpendicular) the crease edges, both convex and concave, of polyhedral objects in an image can be detected by trying an adequate set of scanning directions. Consider the case:

$$P_1: \ z = -x, \qquad P_2: \ z = \frac{-1}{2}x \, .$$

Since $\theta_1 = \theta_2 = 0$, there is no such scanning direction which can detect the edge formed by P_1 and P_2. In the following sections, implementation of the above idea is first presented. Experiments on synthetic images for testing some properties of the algorithm are performed. Afterwards, we turn to real images.

5.4.2.1 Computation of Directional Derivatives

The approach first fits data points with either a piecewise smooth quadratic surface [Bes86] or cubic spline [Yan85]. Directional derivatives are then computed based on the fitting surface. For real images, range data are first smoothed by a square median filter before computations of derivatives. For synthetic images generated from geometric models, the range values need not be smoothed because the projection process has implicitly smoothed the point data.

Let a range image be denoted by the function $f(u,v)$, where u represents the row variable and v the column variable. Two directional derivatives of f_u and f_v have been computed. Once f_u and f_v are available, the directional derivatives can be easily determined for an arbitrary direction (cosθ, sinθ) using the following equation.

$$\frac{df}{d\mathbf{r}} = \frac{\partial f}{\partial u}\cos\theta + \frac{\partial f}{\partial v}\sin\theta = f_u \, cos\,\theta + f_v \sin\theta \, . \tag{5.5}$$

Let us refer to the arrays containing values of f_u and f_v as the f_u and f_v maps respectively. Before using these maps to segment images, they can be smoothed again using a median filter to further remove isolated labels. For each map if an entry has positive value, it is labeled 1; if it is negative, it is labeled 2. Other entries in the map are assigned zero. The results are the sign maps of derivatives. Due to computer discretization error, the probability that a derivative equals zero is very low. Thus, boundaries between regions will mostly be one pixel wide (refer to Figure 5.14). With only one threshold at zero, the method is easy to implement and use. The results below show that it has practical promise.

A symbolic uniform procedure (mentioned in Section 5.4.1) for making the regions more homogeneous is applied to each computed sign map. Since a sign map contains three different values: 0 (background), 1 (positive derivative), and 2 (negative derivative), in order to combine two maps into one and also keep their regions distinguishable, an adequate integration of the sign maps will be needed. A linear combination has been employed. By this, we mean that the first sign map is multipled by a constant and added to the second sign map. The constants adopted shouldn't be 1 or 2 because these two values have been used by sign maps to represent symbolic regions. The constant used in the experiments is 3. After combining the sign maps, the resulting map is the segmentation map. The entire procedure is sketched below.

1. Smooth a range image using a median filter.

2. Compute the directional derivatives f_u and f_v using Besl & Jain's fitting model.

3. Smooth the derivative maps using a median filter.

4. Derive the sign maps based on the derivative maps.

5. Improve the sign maps using a symbolic uniform procedure.

6. Combine the directional derivative sign maps.

5.4.2.2 Synthetic Images

Although the segmentation method is easy to understand theoretically, problems could arise in real data that has noise and that arises from nonpolyhedral objects. To test the previous segmentation procedure, synthetic images generated by projecting geometric models with perturbations, rotations and noise were first used as input data.

(Model perturbation has been defined in Section 5.2.3). Rotating a model means that the model is rotated about the projection direction. Experimental models include the models of a polyhedral object and a clay cobra head (Figure 3.2).

The first experiment investigated the influence of view perturbations on the segmentation results. A set of images was produced by sequentially projecting a model after it had been perturbed by a small angle. Figure 5.14 shows the segmentation results of polyhedral images with different view perturbations within the range of 12 degrees. As we can see, small view perturbations haven't affected the segmentation results.

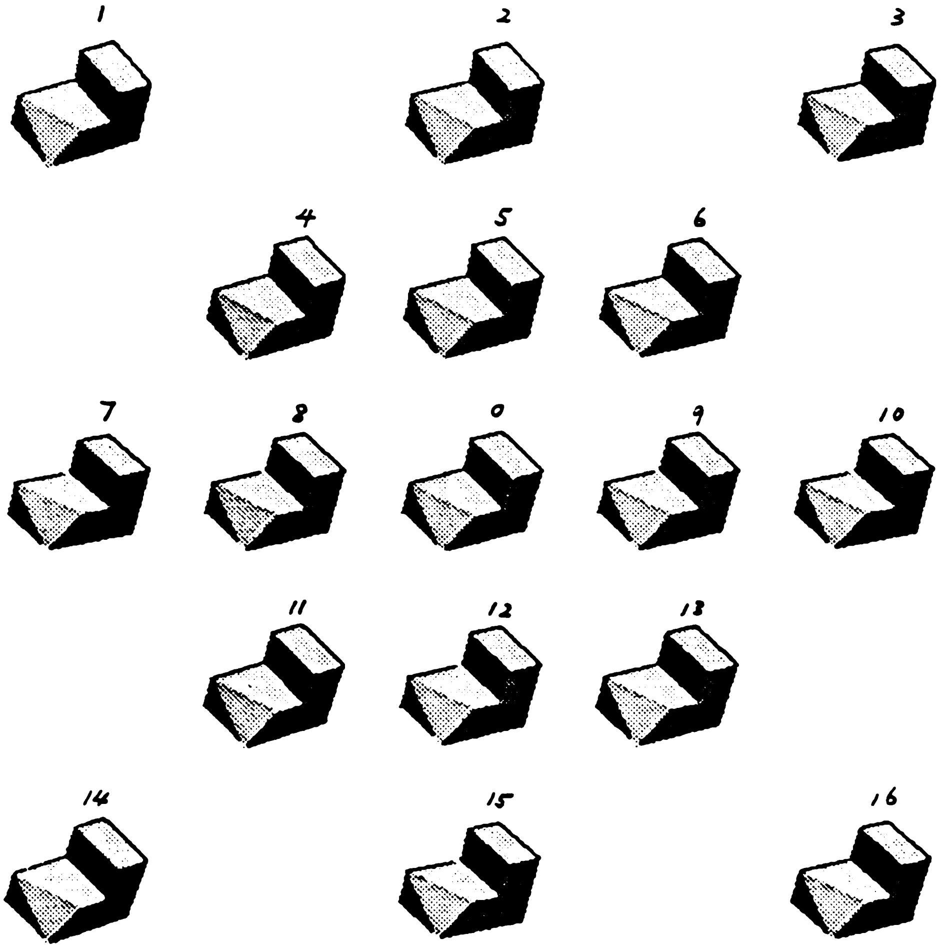


Figure 5.14. Perturbation test using the polyhedral model.

The second experiment explored what would happen when a model is rotated about the projection direction (analogous to the optical axis of a sensor). The model is sequentially rotated 30 degrees each time. Twelve images of the model with different rotated positions are then generated. Figure 5.15 shows the segmentation results of the polyhedral images. Most edges were preserved. Only one was missed in some images, while no fallacious edges appeared. A few missing edges will not harm the object recognition, whereas spurious edges cause more trouble for the recognition procedure.

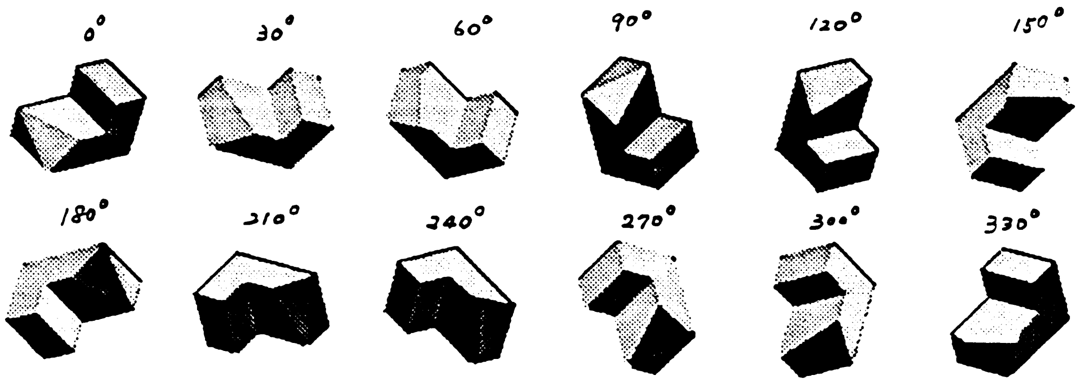


Figure 5.15. Rotation test using the polyhedral model.

The next experiment tested the effect of the size of convolution masks used in data smoothing and computations of directional derivatives. Mask sizes range from 3 × 3 to 11 × 11. The results of polyhedral images are shown in Figure 5.16. According to these results, multiple edges appear around some real edges as mask size increases. However, the multiple edges corresponding to an actual edge are parallel and close to each other. If we find multiple edges corresponding to a real edge in the segmented image, the actual edge can be approximately determined by averaging the multiple edges.

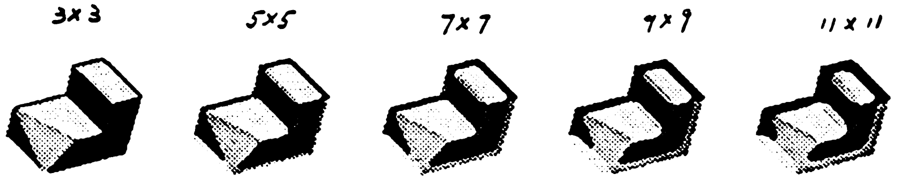


Figure 5.16. Convolution size test using the polyhedral model.

The final experiment examined how well the proposed method can tolerate noise. A uniform random number generator was used to generate noise with ranges from

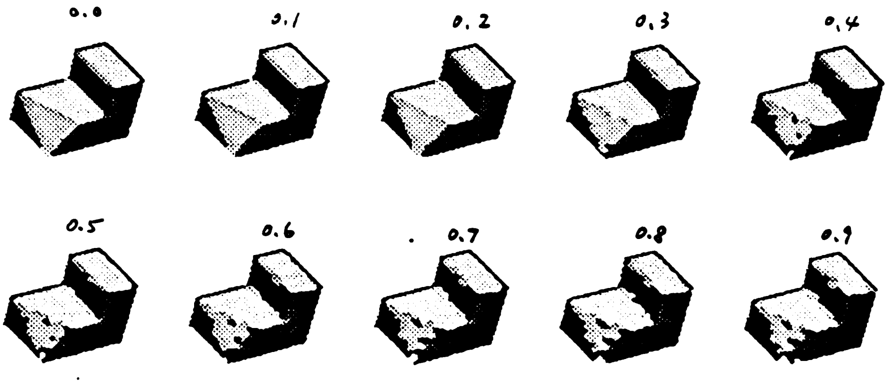


Figure 5.17. Noise test using the polyhedral model.

(−0.1,0.1) to (−0.9,0.9)*mm* which is about (0.1% - 1%) noise with respect to the object diameter. The results of this test are shown in Figure 5.17 for the polyhedral object. Although boundary elements were gradually spoiled, the global shapes of objects were preserved.

Figures from 5.18 to 5.21 display various segmentation results for synthetic images of the cobra head. As shown in these figures, the conclusions made for polyhedral objects apply to curved objects too. A prominent difference may be that unlike polyhedral objects whose segmented contours and regions correspond to physical counterparts, the ones in the segmented images of curved objects may not. Furthermore, those regions and contours are similar to object silhouettes in the property of view-variance. Let us refer to the contours and regions extracted from sign maps of directional derivatives as *critical shape elements*. As mentioned in Chapter 2, wings can be defined along any kind of contour and region as long as they can be reliably detected.

The experimental results reveal that the proposed segmentation approach possesses reasonable ability to resist variations of perturbation, convolution and noise but not rotation. In the rotation test (Figure 5.19), the same segmentation result appears every 90 degrees. For example, the results of views with rotation angle 0, 90, 180 and 270 degrees are exactly the same. Similarly, the views with rotation angle 30, 120, 210 and 300 degrees form another group. This equivalence phenomenon is due to the fact that

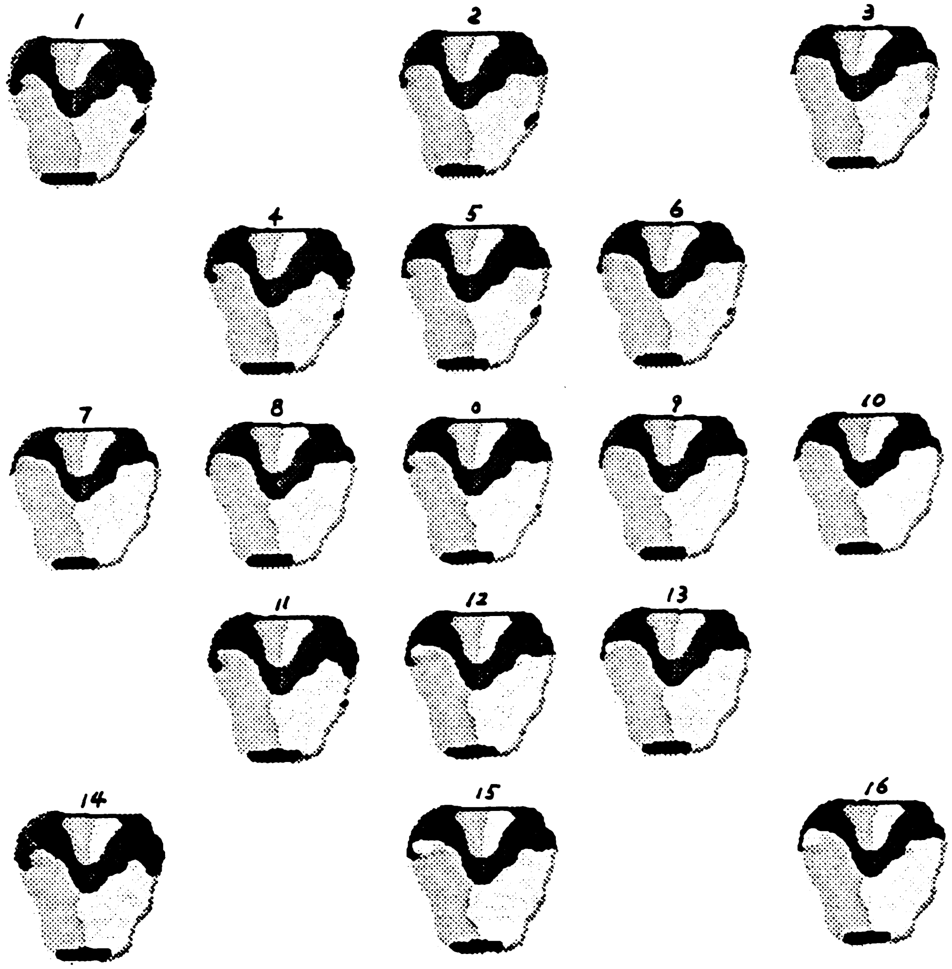


Figure 5.18. Perturbation test using the cobra head model.

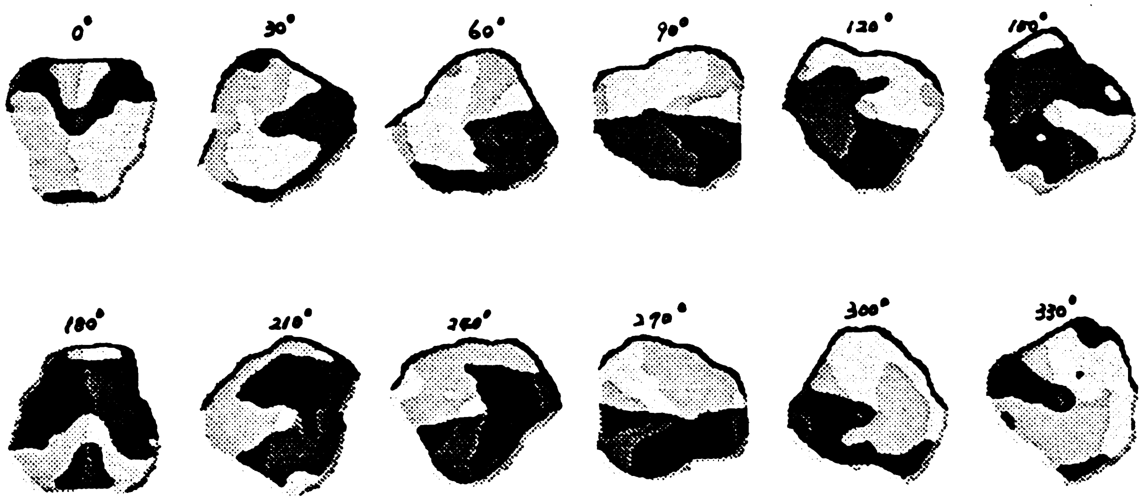


Figure 5.19. Rotation test using the cobra head model.

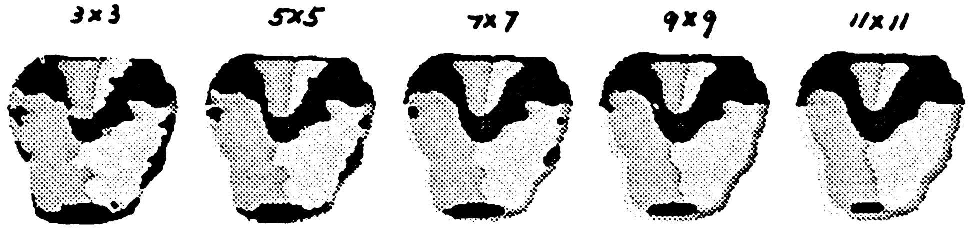


Figure 5.20. Convolution size test using the cobra head model.

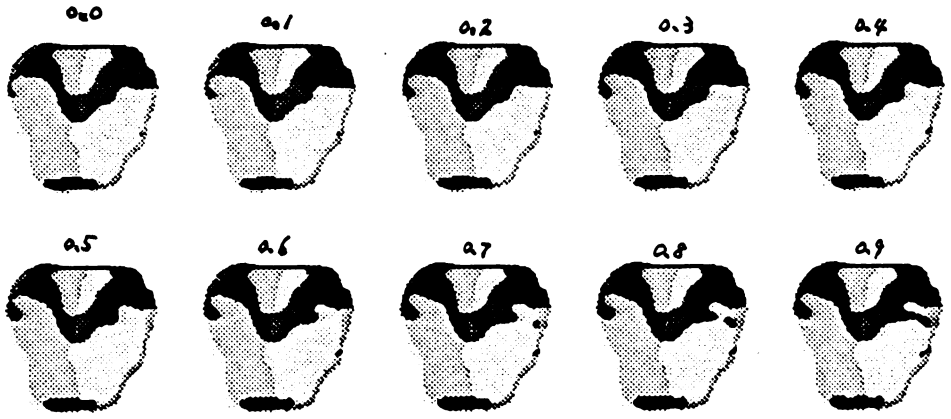


Figure 5.21. Noise test using the cobra head model.

segmentations result from integrating the derivatives computed along two orthogonal directions (90 degrees apart). If we scan an image along four uniformly separated directions (45 degrees apart), the result should repeat every 45 degrees. Let us refer to this phenomenon as the ***equivalence*** property of directional derivatives.

Figure 5.22 shows a set of 36 sign maps of the f_u derivative. Maps are obtained by sequentially rotated the cobra head 10 degrees each time. According to the equivalence property of directional derivatives, there are 18 maps in this set of maps, which are distinct. In fact, if we are given two orthogonal views V_1 and V_2, and a rotational angle A, the view away from view V by angle A can be easily computed using Equation (5.5). This property, the equivalence property, and the property of rotational variance of the segmentation technique might provide some interesting aspects for object representation.

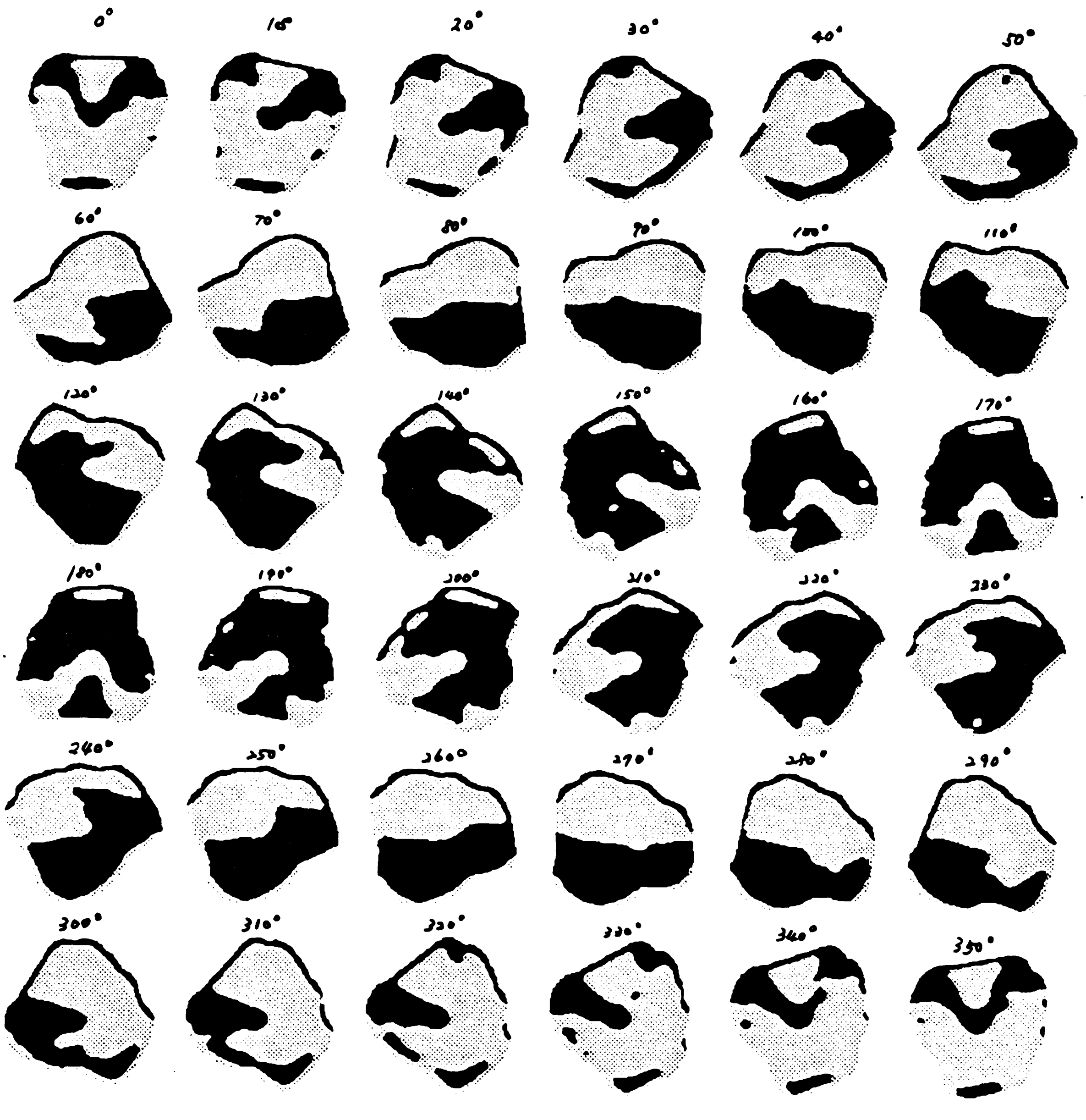


Figure 5.22. The sign maps of f_u derivative of the cobra head.
They are sequentially rotated by 10 degrees.

5.4.2.3 Real Images

In this section, we will apply the proposed segmentation method to real images which were taken using the ERIM sensor. In order to verify the compatibility of segmentation results of real images with synthetic images, their results are computed under different smoothing operators (median filters) and convolution masks (for computing the directional derivatives). Figure 5.23 displays the segmentation results of several real images using a 9 × 9 square median filter to smooth images and a 7 × 7 convolution mask to compute directional derivatives, the results are analogous to those of synthetic images.

What will happen if different masks sizes are used for smoothing and computation. Figure 5.24 displays the results of real images using different sizes of smoothing and computation masks (9 × 9 for smoothing, 3 × 3 for computation). The contours are a little more ragged than those in Figure 5.23, where 9 × 9 and 7 × 7 masks were used. Their basic shapes are consistent. To look at more examples, Figure 5.25 shows the cases of using other mask sizes for the cobra head images, where the results shown in the first row of the figure were obtained using a 9 × 9 smoothing filter and a 5 × 5 convolution mask; the second row displays the results using a 7 × 7 smoothing filter and a 5 × 5 convolution mask, and the last row shows the results using a 7 × 7 and a 3 × 3 respectively. The mask size hasn't significantly affected the segmentation results. The global shapes are preserved. It will not be difficult for a recognition procedure to tolerate such variations.

Finally, Figure 5.26 shows the sign maps of derivatives f_u (column 1), f_v (column 2), and mean curvature (column 3). Instead of fusing maps into one, considering the individual maps will be much easier than a fused one because there are few contours and regions. Besides, other directional derivatives can be easily computed using Equation 5.5. According to the experimental results pertaining to investigations of perturbation, rotation, convolution and noise of the segmentation algorithm, it seems that wings could be nicely defined based on those critical contours and regions. To extract wing features, derivative sign maps are first superimposed by their corresponding mean curvature maps (column 3 of Figure 5.26, where dark gray color means concavity and light gray color represents convexity), and then object wings are defined.

5.4 Summary

In this chapter, we investigated the properties of uniqueness, stability, compatibility, terseness and locality of wing representation. In examination of uniqueness, a function for estimating similarity between two representations was defined in terms of a set of

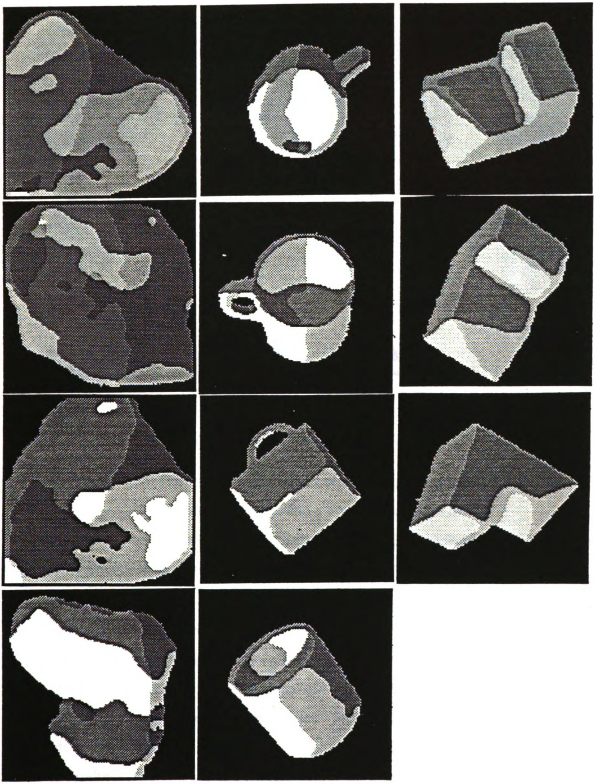


Figure 5.23. The segmentation results of real images in Figure 5.8 using a 9 × 9 smoothing filter and a 7 × 7 computation mask.

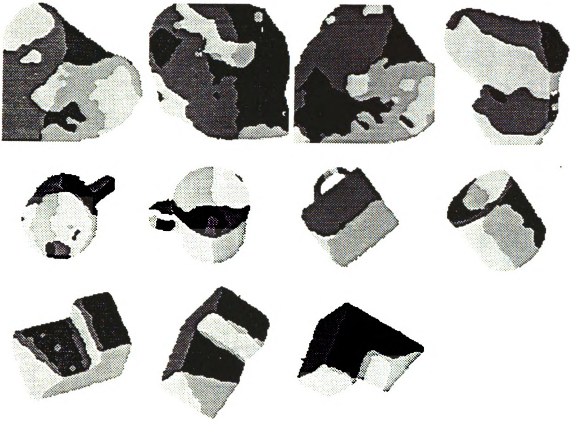


Figure 5.24. The segmentation results of real images in Figure 5.8 using a 9 × 9 smoothing filter and a 3 × 3 computation mask.

criteria. Criteria were characterized by distinct and distinguished wing sets of objects and appearance frequencies of wing primitives.

The stability property was investigated with respect to the impact of input data with variations of resolution, perturbation and convolution on the constructed representations. The algorithms revealed that a small variation in the input data has little influence on the final results. Compatibility of representation was explored by examining consistency between wings detected from real images and those in the database. Two segmentation methods were studied. The first was based on techniques of Hoffman & Jain [Hof87], Marr & Hildreth [Mar80], and Laurendeau & Poussart [Lau86]. The second method was based on the directional derivatives of range values using Besl & Jain's computational model. A series of experiments studying the impact of perturbation, rotation, convolution

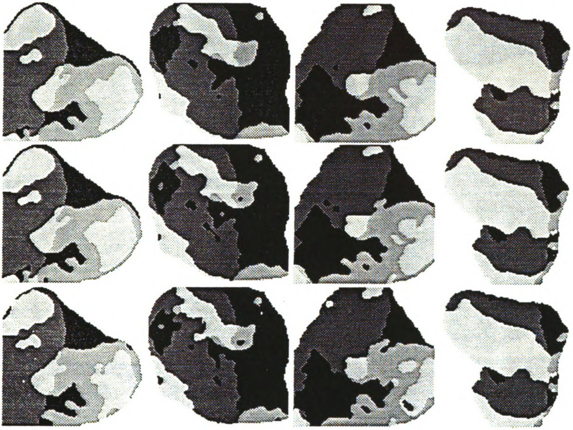


Figure 5.25. The segmentation results of real images using different sizes of smoothing and computation masks. The first row shows the results using a 9 × 9 smoothing filter and a 5 × 5 computation mask. The results in the second row were obtained using a 7 × 7 smoothing filter and a 5 × 5 computation mask. The last row displays the results of using a 5 × 5 and a 3 × 3 respectively.

and noise on the second segmentation approach were performed. The results showed that directional derivatives possesses ability to resist variations of perturbation, convolution and noise but not rotation. We come to the conclusion that the properties of uniqueness, stability and compatibility are reasonably preserved by wing representation, while the property of terseness is not satisfied.

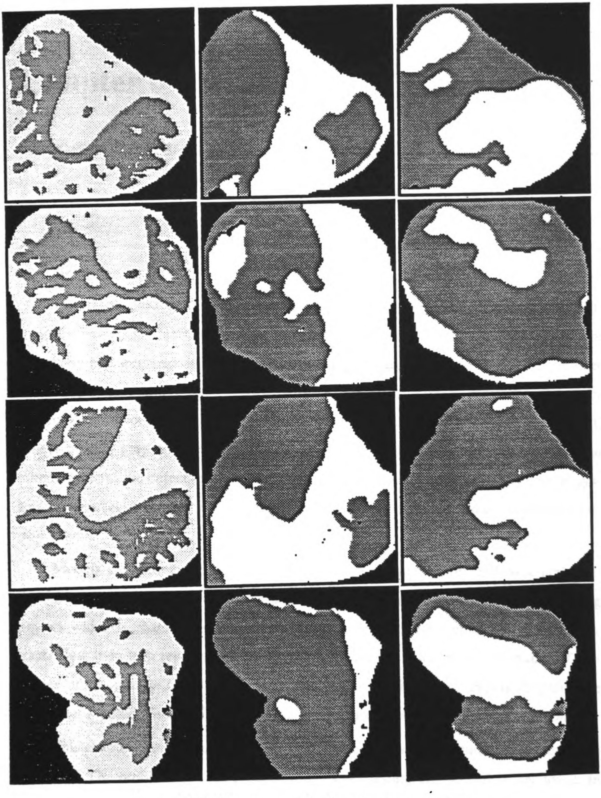


Figure 5.26. Sign maps of f_u, f_v and mean curvatures.

Chapter 6

Recognition Based on Wing Representation

Object recognition involves identifying and locating objects using sensory data. Primary difficulties in object recognition include noisy data and occlusion. Sensory data are inevitably contaminated during acquisition making feature extraction unreliable. Another difficulty is that scene properties are usually distorted during image formation. Recovery of complete object shapes for recognition is neither necessary nor advisable. As a result, object models play an important role; they provide additional knowledge, which can compensate for the defects and irrelevant features of the sensory data. A recognition system using a set of models as a knowledge base is referred to as a *model-based recognition* system.

Models are commonly represented in terms of salient features and structural relations between features. In previous chapters, we presented an approach to representing objects which is characterized by model aspects. Each aspect contains a set of views. Each view is in turn represented by object wings and their structural constraints. Techniques of recognition based on wing representation are developed in this chapter. The major steps include *indexing, consistent labeling, parameter estimation* and *decision.*

This chapter is organized as follows. In Section 1, related recognition techniques are reviewed. The problem and the procedure of model testing are defined in Section 2. In Section 3, a strategy of indexing into models and views based on detected wings is presented. The details of interpretation tree search for consistent labeling are addressed in

Section 4. In view of multiple-view representation, each view portrays a 2-D aspect of an object. A computational model of view transformation in terms of 2-D curves is developed in Section 5. In Section 6, a clustering procedure for determining the class and the pose of an unknown object is presented. A summary of the recognition procedure is given in Section 7.

6.1 Object Recognition Techniques

The recognition problem has long been an important issue in computer vision. The problem can be classified (in terms of dimensionality) into three categories: *2-D to 2-D* [Sto82, Bol82, Tsa85, Tur85, Aya86, Kno86, Sch87], *3-D from 2-D* [Low85, Hut87, Lin88, Sho88, Ohm88] and *3-D to 3-D* [Bol83, Goa83, Sha84, Bro84, Yan85, Fau86, Gri87, Che87, Sto87,88, Ike88]. Several issues are involved in the design of a 3-D object recognition system. The first is the class of features on which matching procedures are based. Features can be points, curves, surfaces, volumes, characteristic parameters, shape descriptors or moments. The second factor involved in the design of a recognition system is the representation scheme used to describe internal models. Relational tables [Sha84], trees [Mea81, Hen85], graphs [Won85], primal sketches [Mar82, Bra85, Gri86], scale spaces [Hum86], evidence rules [Hof87] and characteristic views [Dud77, Fek83, Goa83, Ike88] are a few of the possible schemes.

The third design choice in a recognition system is the control paradigm. First guess [Hof84], hypothesize-and-test [Bro84], relaxation labeling [Wal75], local-feature-focus [Bol82], interpretation tree search [Gri87], generalized Hough transform [Bal83], geometric hashing [Lam88] and RANSAC [Fis81] are some examples of control paradigms. Finally, computational methods are another aspect which influences the design of a recognition procedure. Prominent approaches include least squares [Sch87], relaxation [Pri85], recursion [Low85,87], optimization [Gun87] and closed forms [Gri87, Che87],

Many model-based recognition procedure basically contains three fundamental components: feature correspondence, parameter estimation and decision. We now review some previous work in 3-D object recognition with respect to these three components. A comprehensive survey of most current recognition techniques has been made by Besl & Jain [Bes85], and Chin & Dyer [Chi86].

6.1.1 Feature Correspondence

The simplest approach to solving the feature correspondence problem may be the *Random Sample Consensus* (RANSAC) a paradigm for model fitting [Fis81]. Its basic idea is that given a set of sensed features and model features, a small set of sensed-model feature pairs is randomly selected to instantiate the free parameters of a model. Sensed features consistent with this instantiation are collected. If the number of collected features is greater than a predetermined threshold, it is assumed that the global correspondence is correct and an optimization procedure is then applied to the collected features to determine a final estimate of the pose.

In the RANSAC approach, since feature pairs are randomly selected, no examination of relationships (constraints) among sensed and model features is applied before a parameters are estimated. Bolles & Cain [Bol82] presented a method, called *local-feature-focus*, to select mutually consistent feature pairs using geometric coupling constraints to generate initial hypotheses. This method greatly reduces the number of feature combinations. However, the local-feature-focus method assumes that sensed features and their relations can be reliably computed.

The local-feature-focus approach only uses local information to confirm the consistency of selected sets of features. In order to provide stronger evidence to further reduce the number of combinations and increase the reliability of selected features, global consistency should be incorporated. A mechanism to accomplish this purpose is the paradigm of interpretation tree search. The *interpretation tree* (IT) was first formally proposed by Gaston & Lozano-Pe´rez [Gas83]. By this method, all sensed features are interpreted before they are used to estimate pose parameters.

Since a breadth first search was used by Gaston & Lozano-Pe´rez to implement the control mechanism, a combinatorial explosion forces the system to employ only a small set of features. Grimson and Lozano-Pe´rez [Gri87] used depth-first search and introduced *nil* nodes into the IT. *Nil* nodes are necessary to find all candidates of interpretation when dealing with scenes with multiple objects. Unfortunately, *nil* nodes make the combinatorics worse.

6.1.2 Parameter Estimation

Once a correspondence between sensed and model features has been made, a set of parameters having to do with environments, sensors or models is usually computed. This problem of parameter estimation has been mathematically termed the *generalized inverse set mapping* [Bes85]. Here, we are only concerned with the estimation of rotation, translation and scaling for pose parameters.

The least squares method is the most common technique used to estimate parameters. However, as pointed out by Fischler & Bolles [Fis81], the least squares method is not robust; it assumes the input data are smooth enough to fit a model. It has no ability to reject gross errors *a priori* but simply averages deviations over the data set. This drawback also occurs in many optimization techniques which either maximize or minimize a set of predefined criterion functions. In order to deal with this problem, techniques of iteration and relaxation are used.

The iterative approach requires an initial parameter estimate, often obtained through least squares, then the estimated parameters or an error-correction vector of parameters is resubstituted into the criterion function [Lowe85]. The optimization procedure is then repeated. Through several iterations, better results often can be achieved. The relaxation methods [Wal75, Pri85] are in some sense similar to iterative approaches, which attempt to remove bad data from the data set using statistical updating functions or heuristic constraints. To date, the problem of error recovery in parameter estimation is still a nontrivial problem.

6.1.3 Decision

The parameter estimation step often produces several candidate solutions for the pose parameters. A variety of approaches [Chi86] have been proposed to find the solution from the set of possibilities. The generalized Hough transform [Bal83, Gri88]], clustering [Sto82,87], or geometric hashing [Lam88] have been used to look for so-called optimal solutions. Since the final result is only the most feasible one, the matching is thus often referred to as an *inexact matching* [Boy88].

In the generalized Hough transform approach, a parameter space is divided into discrete bins. Computed parameter vectors are quantized and entered into corresponding bins. Parameter vectors resulting from feature pairs of a correct match of a model to the sensory data are assumed to be approximately equal. The center of gravity of the largest cluster in the parameter space is assumed to represent the correct parameters. There are several difficulties with the use of the Hough transform methods. The first one is selection of the size of bins, which is a function of noise and processing error. The second difficulty is that for a parameter space of high dimension the memory usage will be very large. Accessing and searching entries in a large table is also cumbersome. In addition, according to Grimson & Huttenlocher [Gri88], the probability that false solutions are returned could be very high if the sensory data possess even moderate levels of noise, occlusion and clutter.

Some of the problems of the Hough transform can be partially removed by using the continuous parameter space directly instead of quantizing. The latter methods are generally referred to as clustering approaches. Clustering techniques and their comparisons can be found in Dubes & Jain's classic work [Dub76] in which three categories of clustering methods are presented: minimizing squared error, hierarchical clustering and graph-theoretic clustering.

Stockman [Sto87] introduced a clustering procedure using overlapping volumes in the continuous parameter space. This method simply counts the number of parameter vectors (patterns) within a fixed-size volume centered at each pattern. The pattern producing the largest count together with its neighbors form the most feasible cluster.

Although this method has time complexity $O(N^2)$, where N is the number of patterns, for the Hough transform approach, its time complexity is $O(N')$, where N' is the number of bins. The number of patterns is usually less than the number of bins.

Minimum spanning trees and *k-means* are two popular approaches adopted by computer vision researchers. The minimum spanning tree method, which is a graph-theoretic clustering technique [Zah71], sequentially connects patterns in the parameter space to their unvisited nearest neighbors according to a similarity matrix of patterns. The resulting structure is called the minimum spanning tree. Links in the tree with lengths larger than a pre-defined threshold are deleted resulting in a set of clusters. The center of gravity of the largest cluster is then regarded as the solution of pose parameters. In the k-means approach, patterns are first divided into k groups. Then, points are moved from one group to another in order to optimize a distance metric. Clearly, both minimum spanning tree and k-means approaches need a metric function for pairs of points. However, in the case of transformations, each has three types of parameters: rotation, translation and scaling. Their units are not consistent with each other. Without either normalizing parameters *a priori* or considering them separately, the metric function is nontrivial to design.

6.2 The Definition of the Problem and the Procedure of Model Test

The recognition problem starts with a wing representation of the scene. The scene can be a jumble of objects (Figure 1.1) in which objects may occlude one another. The scene representation may thus be a combination of several incomplete object representations or an incomplete single object representation. Our purpose is to determine the class (identity) and the pose (location) of unknown objects in a scene based on the scene representation.

The proposed recognition procedure includes four stages: indexing, consistent labeling, parameter estimation and decision. In the indexing stage, candidate models and views are predicted based on the set of detected wings extracted from the image. For each candidate view, an *interpretation tree* search using wing constraints is applied to the sensed wings. Once a consistent labeling of sensed wings has been found, transformations of models to objects are readily computed. Each transformation produces goodness of match, mean and maximum errors which indicate how good the match from a model view to an object view is. An object pose is then determined in terms of the predicted viewpoint and the computed transformation. In the final stage of decision, the "best" identity and pose of an object are determined through a clustering procedure. A set of clustering statistics is computed to indicate the confidence of recognition. Thus the final result of an object recognition includes the prediction of class and location of the object, the measurements of goodness of match, mean and maximum errors, and the confidence in this result.

Once a model match has been made, the matched portion is removed from the scene picture. This process could be done in parallel for all models. Model testing will stop when the scene picture is empty or no more models can be matched to remaining portions of the scene picture.

6.3 Indexing into Models and Views

In the early stages of recognition, wing features are used to index into the knowledge base which consists of a set of models and views. As mentioned before, both simple and composite wings possess considerable potential for discriminating objects. In this step, most of the models will be rejected. Only a few models will be considered in the next step.

Suppose we are given a set of sensed wings detected in an image. This set of wings may include some spurious wings because of noise. It may miss some actual wings due to occlusion or imperfect feature extraction. Let S denote the set of sensed wings. Suppose in our model base M there are m models: $\{ M_i, i=1,\cdots,m \}$. Each model may be examined using a preliminary model-test procedure based on wing sets. Models will be ignored if their wing sets don't intersect the sensed wing set. The remaining models are then queued according to a measurement of priority.

Recall that when discussing the uniqueness property of wing representations, a similarity function was introduced for estimating the proximity between two object representations. A similar concept can be used here to construct a priority function to arrange models for testing. A primary difference between the priority function and the similarity function is that the representations used by the similarity function are all complete object representations. The representations used by the priority function is a scene representation which may be an incomplete object representation (single object) or a combination of several object representations (multiple objects). Despite this difference, most criteria defined for the similarity function can be used by the priority function. Rather than directly deriving the priority function, we simply examine the criteria of the similarity function (listed in Appendix D) one by one.

There are eleven criteria used by the similarity function. The first criterion (the numbers of aspects of models) will be ignored by the priority function because a scene representation, which reflects a scene view, is comparable to only one aspect. Similarly, criteria 7, 8, 9 and 11 deal with full model aspects and will be omitted. The second criterion of the similarity function should be replaced by the sets of distinct wing primitives of models corresponding to the set of sensed wing primitives instead of just the sets of distinct wing primitives of models. According to this modified criterion, a model with a distinguished wing set that is more consistent with the sensed wing set will possess high priority and be tested. In a similar vein, criteria 3, 4, 5 and 6 all need to be modified by including this additional condition of correspondence to the sensed wing set. Finally, for the structural relationship criterion, sensed wings are usually not exactly equivalent to model wings. The criterion should be modified to consider the compatibility of relationships subject to tolerance ranges of parameters. After deletion and modification, we obtained six criteria for constructing the priority function $P = e^{-\sum_{i=1}^{6} s_i}$, where s_i is a weighted penalty for violating criterion i.

Let us use an example to illustrate the computation of the priority function. For simplicity, assume that the priority function is only based on the criterion of appearance frequencies of distinct wing primitives. The definition of appearance frequency of a wing primitive has been addressed in Chapter 5 where the cobra head and the coffee cup examples have been shown in Table 5.1. The priority of a model can then be computed by summarizing the appearance frequencies of model wings corresponding to the distinct sensed wings. To make this explicit, let us look at a simple example. Suppose the distinct sensed wing set is $S = \{11, 18, 27\}$, where numbers represent wing primitives. Suppose also that there are two candidate models. The first model M_1 has the distinct wing set {5, 11, 18, 22, 27, 31} with its corresponding appearance frequencies {0.3, 0.2, 0.1, 0.4, 0.3, 0.2}. The second model has the distinct wing set {3, 11, 15, 18, 24, 27, 34} with the corresponding appearance frequencies {0.2, 0.8, 0.4, 0.5, 0.1, 0.6, 0.7}. For model M_1, the summation of appearance frequencies of the model wings corresponding to the sensed wings is 0.2+0.1+0.3=0.6, whereas for model M_2 the summation is 0.8+0.5+0.6=1.9. The second model M_2 has higher priority than that of the first model M_1 because M_2 has wing set that is more consistent with the sensed wing set than that of M_1. M_2 will be tested first based on this simple priority function.

The above example only considered the case of missing wings. To consider spurious wings, other criteria such as 2, 4, and 6 should be included into measurements of priority. Let M_1 denote the set of models in which the wing sets of models are not consistent with the set of sensed wings S. Models with less inconsistency will be examined first. By less inconsistency, we mean that there are fewer sensed wings which are not in the set of model wings. For example, suppose that we have a set of sensed wings $S = \{3, 5, 11, 22, 31\}$. Using the previous model examples, the measurement of inconsistency for model M_1 is 1, and 3 for model M_2. Therefore, M_1 will be tested first in this case. It is clear that to determine the priority of a model one should combine the results of many criteria.

Let us refer to any sequence arranged according to some priority measurement as a *queue*. During the indexing process, models are first queued. Since each model contains a set of aspects, the test sequence of aspects are arranged. Thereafter, each view in a candidate aspect is tested by a view transformation and a clustering procedure to be addressed in Section 6.5 and 6.6 respectively. Views in an aspect need to be queued before they are tested. This procedure will continue until either the unknown object is recognized or failure is reported when the set of candidate models are used up.

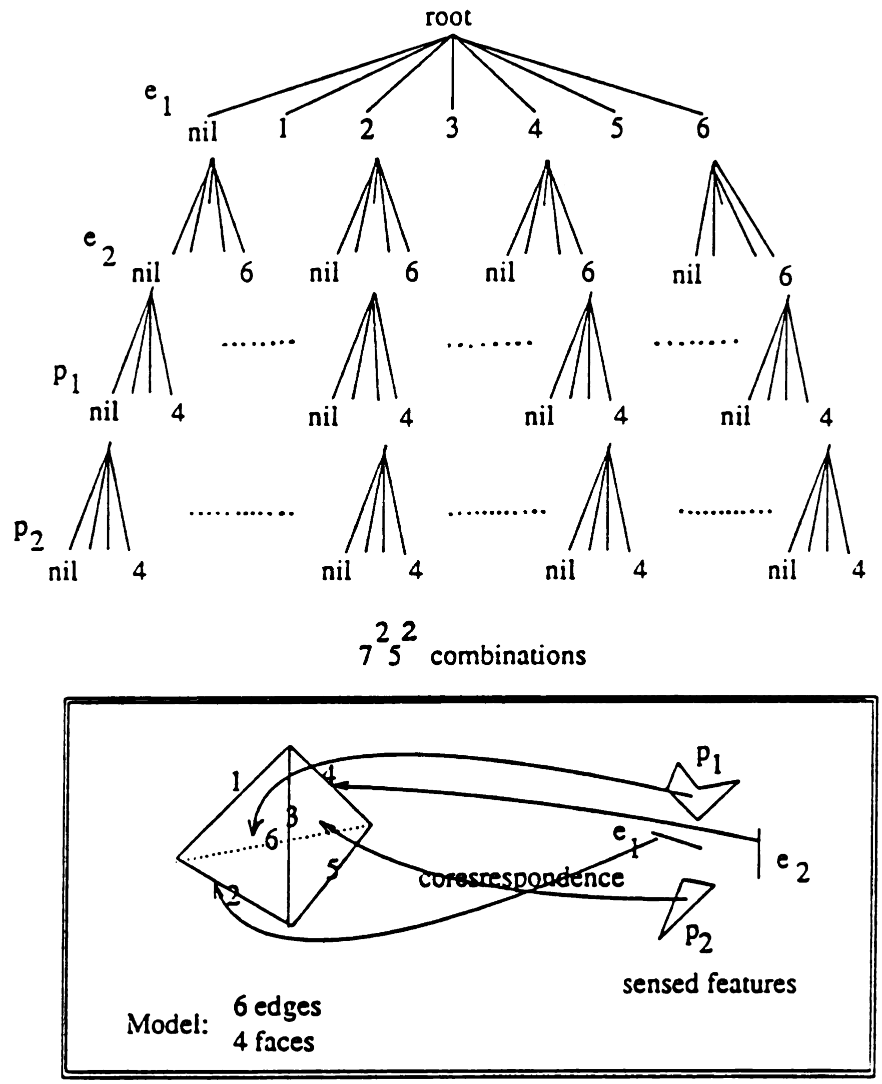
6.4 Interpretation Tree Search

An approach to implementing the control of the interpretation tree search is presented in this section. Before proceeding, let us look at the example in Figure 6.1, which shows a correspondence problem to solve and a tree structure to solve it. In this example, a pyramid having 6 edges and 4 faces is shown on the left side within the square and is used to label the set of sensed features displayed on the right side of the square. Sensed features include 2 edges (e_1 and e_2) and 2 surface patches (p_1 and p_2). The problem asks which model edge should correspond to sensed edge e_i ($i=1,2$), and which model surface should correspond to sensed surface patch p_j ($j=1,2$).

To solve the problem, first we list all possible correspondences between model features and sensed features according to their feature types (edge and face). These correspondences are portrayed in terms of the tree depicted in Figure 6.1. At the first level of the tree, sensed edge e_1 may correspond to any one of the 6 model edges or none at all. Therefore, there are 7 branches spread out from the root: each indicates a possible interpretation of the sensed edge e_1. Proceeding to the next level, each node in the first level is spread out into 7 branches where each branch again specifies a possible interpretation for the second sensed edge e_2. Therefore, to this level, there are a total of 7^2 paths (a path is defined as a sequence of nodes starting from the root node and terminated at a leaf node). Continuing this expansion process for sensed patches p_1 and p_2, we finally obtain $7^2 \times 5^2$ paths. If we consider a model with more features or a large set of sensed features, the number of paths will be large. To alleviate this problem, we introduce constraints to prune some branches of the tree as it grows.

6.4.1 Pruning Algorithm

An algorithm for interpretation tree search which solves the feature correspondence problem of any number of feature types is developed in this section. The algorithm has storage complexity O(N), where N is the number of sensed features. Adopting *Dewey coding* it can be easily implemented by a hardware device, which is similar to a digital counter with a special carry notation.



Tree Pruning Based on Feature Constraints

Figure 6.1 An example of consistent labeling problem.

Suppose we replace the *nil* nodes in an IT tree with 0's. Referring to the previous example in Figure 6.1, the first path of the tree can then be specified by the 4-digit number (0000) and the second path (0001). Following this coding strategy, all paths can be indexed as follows.

```
(0000) (0001) (0002) (0003) (0004) (0010) (0011) (0012) (0013) (0014)
(0020) (0021) (0022) (0023) (0024) (0030) (0031) (0032) (0033) (0034)
(0040) (0041) (0042) (0043) (0044) (0100) (0101) (0102) (0103) (0104)
       ------------------------------------------------------------
(0600) (0601) (0602) (0603) (0604) (0610) (0611) (0612) (0613) (0614)
       ------------------------------------------------------------

       ------------------------------------------------------------
(0640) (0641) (0642) (0643) (0644) (1000) (1001) (1002) (1003) (1004)
       ------------------------------------------------------------

       ------------------------------------------------------------
(6630) (6631) (6632) (6633) (6634) (6640) (6641) (6642) (6643) (6644)
```

There are four digits for every path number, one for each of the sensed features. Look at path number (0642). It means that the first sensed feature corresponds to none of model features (0), the second sensed feature corresponds to model edge 6, the third sensed feature which is a surface patch corresponds to model face 4, and the last sensed feature corresponds to model face 2. Clearly, the above set of path numbers is a set of sequential numbers whose digits have number base (7755) respectively. For this example, we need an array, called the *carry array*, to store the base numbers "7755". To prune a tree (skip some path numbers), we simply change the corresponding digit at some position of a path number. In Appendix F, an implementation of the interpretation tree search for a general case is presented.

The pruning algorithm finds the entire path once a path number has been generated. There is no pointer tracking involved. This not only greatly eases the design of the algorithm but also saves tracking time. It can also be easily implemented by hardware devices. The time complexity of the algorithm is clearly exponential. However, there are heuristic strategies which can be embedded into the procedure to improve its performance. For instance, to compute a transformation, usually we need a small set of feature pairs. Let n be the minimum number of pairs needed to determine a unique transformation. Paths with lengths less than n will be useless and can be skipped. Larger features or features with particular types are generally easier and more reliable to detect. The performance of the algorithm can also be improved by arranging the input sequence of sensed features *a priori*. In addition, an efficient data structure for storing feature constraints, such as hashing tables, may reduce the access time during path checking procedure.

6.4.2 Estimating the Expected Number of Surviving Paths

Analysis of an upper bound on the expected number of nodes at some level of the interpretation tree has been done by Grimson *et al* [Gri87] based on the distance constraint. Instead of estimating the upper bound of nodes using a single constraint, in this section we present a probabilistic model for estimating the expected number of surviving paths $E[N_p]$ characterized by the pruning algorithm and the wing constraints. Wing constraints include eight unary and five binary constraints (Chapter 2). This set of constraints may have redundancy. The importance of the following analysis is that it provides a way to estimate the pruning power of individual constraints. Thus, less important constraints may be abandoned and the remaining constraints can be arranged according to their power.

The analysis will first consider a simple case with one unary and one binary constraint. Later it will be extended to the general case. Suppose we have detected a set of object wings in which n_1 wings are of type t_1, n_2 wings are of type t_2, $\cdots$, and n_w wings are of type t_w. The number N_s of sensed wings is $N_s = \sum_{i=1}^{w} n_i$, where w is the number of distinct wing primitives. Suppose also that a model is to be tested, which has m_1 wings of type t_1, m_2 wings of type t_2, $\cdots$, and $m_{w'}$ wings of type $t_{w'}$. The number N_m of model wings is $N_m = \sum_{j=1}^{w'} m_i$, where w' is the number of distinct model wing primitives. Let p_u denote the probability that a pair consisting of a sensed wing and a model wing satisfies the unary constraint u, and p_b denote the probability that two pairs of sensed wings and model wings pass the binary constraint b. The following proposition provides the expected number $E[N_p]$ of surviving paths, where N_p denotes the random variable representing the number of surviving paths.

Proposition 6.1: Given parameters p_u, p_b, N_s, n_i, m_j and w (as defined above), the expected number $E[N_p]$ of surviving paths with respect to the unary constraint u, the binary constraint b and the given pruning algorithm is

$$E[N_p] = [\prod_{i=1}^{w} m_i^{n_i}]\, p_u^{N_s}\, p_b^{\frac{N_s(N_s-1)}{2}} .$$

Proof: Without loss of generality, assume the sequence of sensed wings has been arranged in advance so that model wings with type t_1 are first branched; then the wings with type t_2 are next, and so forth. Let a_i denote the expected number of branches at level i. Therefore,

$$a_1 = m_1 p_u$$

$$a_2 = [m_1 p_u p_b]\, a_1$$

$$a_3 = [m_1 p_u p_b^2]\, a_2$$

....

$$a_{n_1} = [m_1 p_u p_b^{n_1-1}]\, a_{n_1-1}$$

$$a_{n_1+1} = [m_2 p_u p_b^{n_1}]\, a_{n_1}$$

....

$$E[N_p] = a_{N_s} = [m_w p_u p_b^{N_s-1}]\, a_{N_s-1}$$

By iterative substitution of the expected number of branches at each level, we obtain,

$$E[N_p] = [\prod_{i=1}^{w} m_i^{n_i}]\, p_u^{N_s}\, p_b^{\sum_{j=1}^{N_s-1} j} = [\prod_{i=1}^{w} m_i^{n_i}]\, p_u^{N_s}\, p_b^{\frac{N_s(N_s-1)}{2}} . \qquad (6.1)$$

■

From the above proposition, it is clear that the expected number of branches at the last level (the expected number of surviving paths) is exponential. In order to make $E[N_P]$ small, one or both of p_u and p_b should be small. Since $\frac{N_s(N_s-1)}{2}$ is always larger than N_s (except $N_s = 1$ or 2), binary constraints play a more important role than unary constraints in pruning the interpretation tree.

To estimate the probabilities p_u and p_b, divide both sides of Equation (6.1) by $\prod_{i=1}^{w} m_i^{n_i}$ and take the logarithm of the resulting equation. We obtain a linear equation in $\log p_u$ and $\log p_b$:

$$\log \frac{E[N_p]}{\prod_{i=1}^{w} m_i^{n_i}} = N_s \log p_u + \frac{N_s(N_s-1)}{2} \log p_b . \tag{6.2}$$

Since the left side of the above equation and N_s can be found experimentally, the probabilities p_u and p_b can thereby be evaluated by solving a system of linear equations.

For the general situation, suppose there are n_u unary constraints with pruning probabilities $\{p_{u_j}, j=1,\cdots,n_u\}$ respectively, and n_b binary constraints with probabilities $\{p_{b_k}, k=1,\cdots,n_b\}$ respectively. Then, the expected number of surviving paths N_p for the general case can be obtained by simply replacing the probabilities p_u and p_b in Equation (6.1) by the products of probabilities p_{u_j}'s and p_{b_k}'s:

$$E[N_p] = [\prod_{i=1}^{w} m_i^{n_i}] [\prod_{j=1}^{n_u} p_{u_j}]^{N_s} [\prod_{k=1}^{n_b} p_{b_k}]^{\frac{N_s(N_s-1)}{2}} . \tag{6.3}$$

To estimate the probability of a constraint i, or equivalently the pruning power of a constraint, set all probabilities of constraints except for i to one. Note that this disables the pruning capability of the other constraints. As a result, we can concentrate on the constraint of interest. If we are interested in the combined pruning capability of a set of constraints, keep the probabilities of those constraints unchanged and set the other probabilities to one.

The following experiment uses the housing [Gri85] as an example to illustrate the power of feature constraints in pruning the interpretation tree. The same idea can be used to study wing constraints. Referring to Table 6.1, the results were obtained using the binary constraints of distance and angle ($n_b = 2, p_b = p_{b_1} \cdot p_{b_2}$), no unary constraints ($n_u = 0$) and edge evidence only ($w = 1$). Ten synthetic edges ($N_s = 10$ and $n_1 = 10$) and a

Level	#Path reaching this level	#Die	#Survive	#Survive to this level without *nil*	#Survive to this level with *nil*	#Checks at this level
1	16	0	16	15	1	0
2	256	195	61	30	31	225
3	976	778	198	25	173	950
4	3168	2856	312	3	309	3197
5	4992	4215	777	1	776	5900
6	12432	11495	937	1	936	12188
7	14992	13729	1263	1	1262	14889
8	20208	18267	1941	1	1940	22008
9	31056	27983	3073	1	3072	33755
10	49168	44729	4439	1	4438	56618

Table 6.1 The number of paths at levels using edge evidence.

model with 15 edges ($m_1 = 15$) have been employed for this experiment. The number of surviving paths to level 10 (with *nil*) is 4438 ($E[N_p] = 4438$). Substituting these values into Equation (6.2), we obtain an equation in $\log p_b$.

$$log\,4438 - 10\,log\,15 = 45\log p_b, \quad \text{and } p_b = 0.66.$$

Clearly, this probability is too large because we considered the paths with *nil* nodes which were assumed to satisfy all constraints. For paths without *nil*, the number of surviving paths to level 10 is 1 ($E[N_p] = 1$). Replacing 4438 by 1 in the above equation, we obtain $p_b = 0.548$. Unfortunately, this is still large. A possible reason is that to the level 5 the number of surviving paths has already been narrowed down to 1, which is the correct interpretation path. Constraint checks after this level will all be satisfied and make contributions to the probability p_b. This situation can also be seen in Equation 6.2. After level 5, $E[N_p]$ remains 1, whereas both n_i and N_s keep increasing so as to make the probability p_b increase too. Therefore using a large number of sensed features may degrade the power of constraints and may not be always helpful in saving time finding the correct path. An adequate set of a few sensed features will be good enough for quickly finding the "correct" path.

Substituting $N_s = 5$ and $n_1 = 5$, we get $p_b = 0.258$. If we further consider going to level 4 (4 sensed features employed), the number of paths is 3. Let $N_s = 4$, $n_1 = 4$ and $E[N_p] = 3$. We obtain $p_b = 0.197$. Although this result is already much better than the previous ones, we really expect a smaller value for p_b indicating better power of the constraint. Unfortunately, we don't know in advance which and how many features will be appropriate. In applications, several unary and binary constraints are used simultaneously. Multiple constraints can increase the performance of tree pruning.

Level	#Path reaching this level	#Die	#Survive	#Survive to this level without *nil*	#Survive to this level with *nil*	#Checks at this level
1	16	0	16	15	1	0
2	256	185	71	40	31	225
3	1136	915	221	40	181	1324
4	3536	3049	487	19	468	3865
5	7792	6947	845	2	843	8533
6	13520	11431	2089	4	2085	17902
7	33424	28231	5193	8	5185	44668
8	83088	74783	8305	3	8302	96019
9	132880	119808	13072	3	13069	171333
10	209152	191489	17663	2	17661	247096

Table 6.2 The number of paths at levels using face evidence.

Several experiments have been performed. Table 6.2 shows the results using face evidence only. In Table 6.3, mixed edge and face evidence were used. Substituting these results into Equation (6.2), the combined probabilities of binary constraints (distance and angle) using different feature evidence are shown in Table 6.4. Mixed evidence is a little better than edge evidence, and edge evidence is in turn better than face evidence. We conclude that some constraints are suitable for certain types of evidence, and that using many inadequate constraints or many sensed features may not be helpful in tree pruning. To study integrated influence of multiple constraints, more data will be needed to set up an overdetermined system of linear equations for computing probabilities.

Level	#Path reaching this level	#Die	#Survive	#Survive to this level without *nil*	#Survive to this level with *nil*	#Checks at this level
1	16	0	16	15	1	0
2	256	155	101	70	31	225
3	1616	1457	159	12	147	1567
4	2544	2300	244	2	242	2466
5	3904	3282	622	1	621	4253
6	9952	8680	1272	1	1271	11543
7	20352	17889	2463	1	2462	25980
8	39408	34860	4548	1	4547	51085
9	72768	65370	7398	1	7397	85749
10	118368	108429	9939	1	9938	133737

Table 6.3 The number of paths at levels using mixed evidence of edge and face.

To This Level	*Nil* Nodes	Edge Evidence	Face Evidence	Mixed Evidence
10	with	0.660	0.680	0.672
10	without	0.548	0.556	0.548
5	without	0.258	0.276	0.258
4	without	0.197	0.269	0.184

Table 6.4. The probabilities of combined binary constraints (distance and angle) computed to different levels with or without *nil* nodes using different feature evidence.

6.4.3 Time Complexity

We define the time complexity of the pruning algorithm as the expected number $E[N_c]$ of checks for both unary and binary constraints. Following the definitions in the last section, the expected numbers of checks for levels in the IT are

$$\begin{aligned} b_1 &= m_1 \\ b_2 &= (m_1 + m_1 p_u)\, a_1 \\ b_3 &= (m_1 + 2m_1 p_u)\, a_2 \\ b_4 &= (m_1 + 3m_1 p_u)\, a_3 \end{aligned}$$

....

$$\begin{aligned} b_{t_1} &= [m_1 + (t_1 - 1) m_1 p_u]\, a_{t_1 - 1} \\ b_{t_1+1} &= [m_2 + t_1 m_2 p_u]\, a_{t_1} \end{aligned}$$

....

$$b_{N_s} = (\prod_{i=1}^{w} m_i^{n_i})\, p_u^{N_s-1}\, p_b^{\frac{(N_s-1)(N_s-2)}{2}}\, [1 + (N_s - 1) p_u]$$

The expected number $E[N_c]$ of checks is simply the summation of b_i's.

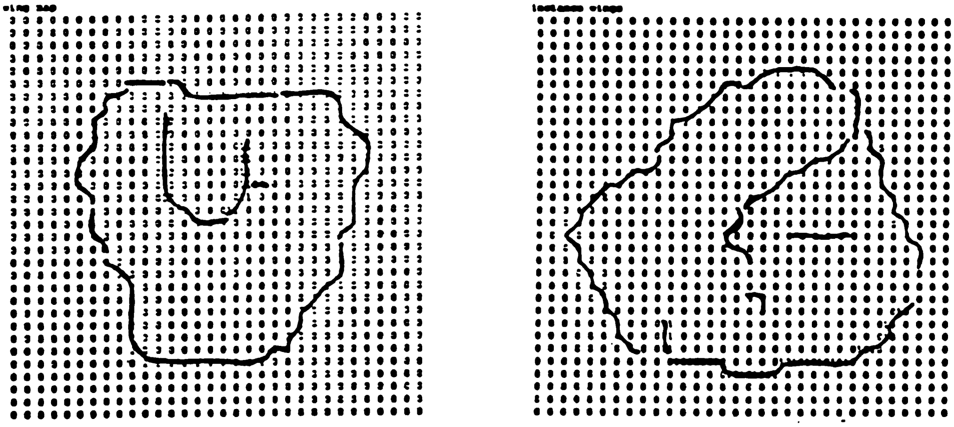
$$E[N_c] = \sum_{i=1}^{N_s} b_i. \tag{6.4}$$

Note that the *nil* nodes will not affect the above result because there are no constraint checks needed. *Nil* nodes are assumed to comply with all constraints (both unary and binary). In order to reduce the number of checks, the probabilities should be small. In other words, the feature constraints should be powerful. For the general case, replace p_u and p_b respectively by the products of their probabilities for individual constraints.

Experimental results are shown in the last columns of Table 6.1, 6.2 and 6.3. The number of constraint checks at certain level are about 5-14 times of the number of surviving paths with *nil* at that level. Equation (6.1) and (6.4) show that both numbers of checks and paths should increase exponentially as the tree level increases, and so does their ratio. In practice, the ratio however increases linearly as the level increases (Table 6.1, 6.2 and 6.3). This reveals that an exponential numbers of paths have been pruned at each level.

6.5 View Transformation

Suppose we are given two views: a model view and a scene view. Let M be the set of wings derived from the model view and S be the set of wings derived from the scene view. An example is shown in Figure 6.2. Part (a) shows the model view and part (b) shows the scene view. Suppose also that the correspondence between model wings and scene wings is known by the IT algorithm. Now, the problem is to determine the transformation T by which the scene view can be matched to the model view.



(a) model view

(b) scene view

Figure 6.2. A model view and a scene view.

6.5.1 Curve Matching

Schwartz and Sharir [Sch87] presented an optimization method to solve the problem of 2-D curve matching. Their method only considered the problems of rotation and translation, and assumed the scale problem has already been solved. The following method extends the Schwartz and Sharir work by including the scale factor.

Consider a consistent pair of a model wing L and a detected wing segment l as shown in Figure 6.3. Let l' be that portion of L which is the corresponding segment of l, with transformation T. Each curve is represented by a set of uniformly spaced points (equal intervals).

$$L = \{ \mathbf{u}'_1, \mathbf{u}'_2, \cdots, \mathbf{u}'_n \},$$

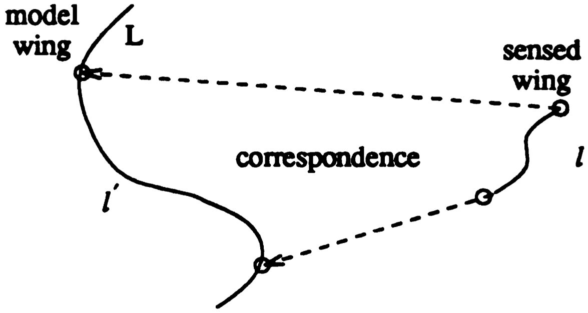


Figure 6.3. A consistent pair of a model wing and a detected wing segment.

$$l = \{\mathbf{u}_1, \mathbf{u}_2, \cdot \cdot \cdot, \mathbf{u}_m\}, \quad \textit{where} \quad m \le n,$$

$$l' = \{\mathbf{v}_1, \mathbf{v}_2, \cdot \cdot \cdot, \mathbf{v}_m\}.$$

As shown in Figure 6.3, we assume that the sensed wing will typically be a subset of the model wing and hence the transformation is "into" and not "onto". The following transformation computation needs to applied to each segment of length m of L, so that it will result in 2(n-m+1) transformations computed from the pair of l and L. The constant 2 here is due to the need to match both directions. Note that the true transformation is included in this set of computed transformations but that there may be several viable candidates because one pair of wings will not be enough to determine a unique transformation. There are usually several consistent wing pairs available. Having computed candidate transformations for all wing pairs, a "best" transformation can be determined through a clustering procedure [Sto82,87, Gri88] to be discussed in Section 6.6.

The matching problem can then be formulated as an optimization problem,

$$\delta = \min_{\mathbf{T}} \sum_{j=1}^{m} |\mathbf{T}\mathbf{u}_j - \mathbf{v}_j|^2 \ , \quad \mathbf{T}\mathbf{u}_j = \mathbf{R}_\theta s \mathbf{u}_j + \mathbf{a},$$

where $\mathbf{R}_\theta$ is the rotation, $\mathbf{a}$ is the translation and s is the scale factor. Our purpose is to look for a transformation T that minimizes the δ function. To simplify the derivation, assume the scale factor s is a constant. The scale factor is a nonlinear function of depth, but can be approximated by a constant when depth variation is small relative to the standoff. By following the derivation presented in Appendix F, the rotation angle which minimizes the δ function is the negative of the polar angle of $\sum_{j=1}^{m} u_j\bar{v}_j$, where u_j and v_j are corresponding complex values of vectors $\mathbf{u}_j$ and $\mathbf{v}_j$. We obtain the minimization δ^* of δ

$$\delta^* = \sum_{j=1}^{m} |\mathbf{v}_j|^2 - m\left|\frac{1}{m}\sum_{j=1}^{m}\mathbf{v}_j\right|^2 - \frac{\left|\sum_{j=1}^{m}\mathbf{u}_j\cdot\mathbf{v}_j\right|^2 + \left|\sum_{j=1}^{m}(\mathbf{u}_j\times\mathbf{v}_j)\right|^2}{\sum_{j=1}^{m}|\mathbf{u}_j|^2},$$

$$s = \frac{\left[\left|\sum_{j=1}^{m}\mathbf{u}_j\cdot\mathbf{v}_j\right|^2 + \left|\sum_{j=1}^{m}(\mathbf{u}_j\times\mathbf{v}_j)\right|^2\right]^{1/2}}{\sum_{j=1}^{m}|\mathbf{u}_j|^2},$$

and $\mathbf{a} = \mathbf{a}_u - \mathbf{R}_\theta\mathbf{a}'$, where $\mathbf{a}_u = \frac{1}{m}\sum_{j=1}^{m}\mathbf{u}_j$, and $\mathbf{a}' = \frac{1}{m}\sum_{j=1}^{m}\mathbf{v}_j$.

The above mathematical model can be implemented in an iterative fashion by modifying the point set of detected wing l using the intermediate result of scale factor s and repeating the procedure until a predefined precision is reached or until a fixed number of iterations has been completed. The inverse value of δ^*/m can be used to measure the goodness of the match.

6.5.2 Experimental Results

To test the curve matching algorithm, a model view is randomly selected. The wing features of this view are shown in Figure 6.2(a). Five sets of synthetic wings are then generated from the model wings to serve as sensed wings. Wings are represented in terms of points whose sequences in a wing are known. Each set of synthetic wings are then respectively contaminated using seven noise levels of 0.0, 0.5, 1.0, 1.5, 2.0, 2.5 and 3.0mm. Noise is generated using a uniform random number generator. Thus, we obtain 35 sets of sensed wings for testing the algorithm. The largest noise (3.0mm) used here is larger than the noise of 2.5mm in the range images taken from ERIM [Hof86].

Tables from 6.5 to 6.9 show the matching results of different noise levels. Results include goodness of match, mean and maximum errors, relative cluster sizes of rotation, translation and scaling components. The goodness of match was defined in the last section. Mean and maximum errors are estimated by a local verification process which applies an estimated transformation to the model wings and compare their resulting locations with the sensed wings. Details of local verification have been addressed elsewhere [Che87]. The relative cluster sizes of transformation components are defined as the ratios of cardinality of the optimal cluster to the total number of candidate transformations.

The correspondence relationships between model wings and sensed wings are assumed to be known by the matching algorithm. As indicated in the last section, each pair i of model and sensed features produces a set S_i of transformations. Suppose we have n such feature pairs. Then the total number of transformations derived from all feature pairs will be $\sum_{i=1}^{n} |S_i|$. Among these transformations, n out of them should closely approximate to the correct transformation. Through a clustering procedure (to be discussed in the next section), the optimal transformation can be determined. Since each computed transformation comes along with the measurements of goodness of match, mean and maximum errors, it is desirable that transformations with poor measurements are deleted beforehand in order to reduce the number of candidate transformations such that the processing time can be improved. Thresholds of goodness and errors are shown in Table 6.10 to 6.14.

As shown in Table 6.5, measurements of goodness, mean and maximum errors all increased as noise increases. Although the maximum errors are a little larger than their corresponding input noise, the mean errors are all smaller than the input noise. More surprisingly, except for Table 6.6 (Data set 2), Table 6.7, 6.8 and 6.9 indicate that both mean and maximum errors are all less than their input noise. It means the input noise hasn't been amplified in the final result through the clustering procedure. In addition, these tables also show that the relative cluster sizes mostly decrease very fast at the beginning of increasing noise and then becomes stable. This situation reveals that noisy data degrades the compactness of clusters. Cluster radii of transformation components, shown in Tables 6.10 to 6.13 explain this too.

Data Set 1

Noise Level	Goodness of Match	Mean Error	Maximum Error	Relative Cluster Size of Components		
				Rotation	Translation	Scaling
0.0	0.00047	0.00056	0.00080	0.7273	0.5455	0.5455
0.5	0.32143	0.46927	0.93073	0.2500	0.2000	0.2000
1.0	0.86586	0.79019	1.59344	0.2500	0.1500	0.1500
1.5	1.36464	1.06956	1.85905	0.3750	0.2500	0.2500
2.0	2.36017	1.38944	2.43219	0.3750	0.3750	0.3750
2.5	3.38219	1.64476	2.86133	0.3333	0.3333	0.3333
3.0	4.17338	1.71092	3.48439	0.2857	0.2857	0.2857

Table 6.5. Matching results using data set 1.

Data Set 2

Noise Level	Goodness of Match	Mean Error	Maximum Error	Relative Cluster Size of Components		
				Rotation	Translation	Scaling
0.0	0.00039	0.00004	0.00007	1.0000	1.0000	1.0000
0.5	0.94880	0.89403	1.27026	0.1250	0.1250	0.1250
1.0	1.61368	1.01382	1.40583	0.0833	0.0833	0.0833
1.5	8.68530	2.47038	4.14518	0.2000	0.2000	0.2000
2.0	24.89006	3.84151	7.33181	0.2826	0.2826	0.2826
2.5	28.78731	4.38987	7.55852	0.2727	0.2727	0.2121
3.0	46.48094	5.41145	8.59683	0.1481	0.1481	0.1481

Table 6.6. Matching results using data set 2.

Among these five sets of experimental data, the second data set behaved strangely because large radii were used. Its measurements of goodness and errors are also poor. The reason is that there is a false cluster with larger size than that of the true one when using a small radius. Until the cluster radius was large enough, the true cluster could override the false one and become the largest cluster. Unfortunately, at this moment, several poor transformations have also been recruited and make the center of gravity

Data Set 3						
Noise	Goodness	Mean	Maximum	Relative Cluster Size of Components		
Level	of Match	Error	Error	Rotation	Translation	Scaling
0.0	0.00067	0.00003	0.00004	0.6000	0.4000	0.4000
0.5	0.00073	0.00906	0.01578	0.2308	0.1538	0.1538
1.0	0.00081	0.01800	0.03137	0.2500	0.1667	0.1667
1.5	0.00215	0.02685	0.04681	0.2308	0.1538	0.1538
2.0	0.00253	0.03560	0.06209	0.2308	0.1538	0.1538
2.5	0.00364	0.04425	0.07721	0.2308	0.1538	0.1538
3.0	0.00588	0.05280	0.09219	0.2143	0.1429	0.1429

Table 6.7. Matching results using data set 3.

Data Set 4						
Noise	Goodness	Mean	Maximum	Relative Cluster Size of Components		
Level	of Match	Error	Error	Rotation	Translation	Scaling
0.0	0.00099	0.00004	0.00007	0.5714	0.2857	0.2857
0.5	0.00703	0.07572	0.14043	0.3333	0.2222	0.2222
1.0	0.02915	0.15036	0.27478	0.3000	0.2000	0.2000
1.5	0.06502	0.22414	0.40350	0.3000	0.2000	0.2000
2.0	0.11426	0.29721	0.52694	0.2727	0.1818	0.1818
2.5	0.11476	0.28309	0.53459	0.2000	0.2000	0.2000
3.0	0.15843	0.33510	0.62948	0.2000	0.2000	0.2000

Table 6.8. Matching results using data set 4.

distant from the true transformation.

From Table 6.10, 6.12, 6.13 and 6.14, both thresholds of measurements and cluster radii vary much little after noise level of 1.5mm. It means that both thresholds and radii are all large enough. Gradually increasing noise gives little influence on the final results. In other words, the matching algorithm can tolerate noise to some level while still produce results within a restricted volume in the feature space. According to Stockman *et al*

Data Set 5						
Noise Level	Goodness of Match	Mean Error	Maximum Error	Relative Cluster Size of Components		
				Rotation	Translation	Scaling
0.0	0.00045	0.00002	0.00004	1.0000	1.0000	1.0000
0.5	0.07124	0.21840	0.38509	0.4000	0.3000	0.3000
1.0	0.17116	0.33315	0.65749	0.1905	0.1429	0.1429
1.5	0.65395	0.62382	1.11667	0.3158	0.2632	0.2632
2.0	0.80223	0.72763	1.24017	0.3333	0.2778	0.2778
2.5	0.87207	0.77968	1.24984	0.3235	0.2647	0.2647
3.0	1.36610	0.99741	1.67153	0.2667	0.2000	0.2000

Table 6.9. Matching results using data set 5.

Data Set 1						
Noise Level	Goodness of Match	Mean Error	Maximum Error	Cluster Radius of Components		
				Rotation	Translation	Scaling
0.0	0.1	0.01	0.01	0.05	0.05	0.1
0.5	1.0	2.00	5.00	0.02	2.00	0.2
1.0	1.0	2.00	5.00	0.01	5.00	0.2
1.5	1.0	2.00	5.00	0.01	5.00	0.2
2.0	3.0	4.00	10.00	0.05	10.00	0.3
2.5	4.0	5.00	12.00	0.07	12.00	0.3
3.0	5.0	6.00	15.00	0.10	15.00	0.4

Table 6.10. The thresholds of parameters and the cluster radii of transformation components for data set 1.

[Sto88] and Grimson [Gri85], in the case of 3-D transformation, the radius of rotation component can be set to be 3-5 times the error in computing angles from local features. Set the radius of translation component to be the tolerance scaled by the object diameter plus the error in sensing points. In these experiments, it seems more adequate using 1-4 times instead of 3-5 times the input error.

Data Set 2						
Noise Level	Goodness of Match	Mean Error	Maximum Error	Cluster Radius of Components		
				Rotation	Translation	Scaling
0.0	0.1	0.01	0.01	0.05	0.05	0.1
1.0	1.0	2.00	5.00	0.02	2.00	0.2
1.5	3.0	4.00	10.00	0.05	10.00	0.3
2.0	40.0	40.00	40.00	0.30	40.00	1.0
2.5	50.0	45.00	45.00	0.40	45.00	1.0
2.5	55.0	50.00	50.00	0.50	50.00	1.0
3.0	70.0	70.00	70.00	0.20	70.00	1.0

Table 6.11. The thresholds of parameters and the cluster radii of transformation components for data set 2.

Data Set 3						
Noise Level	Goodness of Match	Mean Error	Maximum Error	Cluster Radius of Components		
				Rotation	Translation	Scaling
0.0	0.1	0.01	0.01	0.05	0.05	0.1
0.5	1.0	2.00	5.00	0.02	2.00	0.2
1.0	1.0	2.00	5.00	0.01	5.00	0.2
1.5	1.0	2.00	5.00	0.01	5.00	0.2
2.0	1.0	2.00	5.00	0.01	5.00	0.2
2.5	1.0	2.00	5.00	0.01	5.00	0.2
3.0	1.0	2.00	5.00	0.01	5.00	0.2

Table 6.12. The thresholds of parameters and the cluster radii of transformation components for data set 3.

6.6 Clustering Procedure

After view transformation, a set of candidate transformations is obtained. If transformations are represented by points in a feature space, those resulting from feature pairs that correctly match a model to the sensory data should form a cluster in the feature space. Therefore, to look for the "correct" transformation is equivalent to looking for a

Data Set 4						
Noise Level	Goodness of Match	Mean Error	Maximum Error	Cluster Radius of Components		
				Rotation	Translation	Scaling
0.0	0.1	0.01	0.01	0.05	0.05	0.1
0.5	1.0	2.00	5.00	0.02	2.00	0.2
1.0	1.0	2.00	5.00	0.01	5.00	0.2
1.5	1.0	2.00	5.00	0.01	5.00	0.2
2.0	1.0	2.00	5.00	0.01	5.00	0.2
2.5	1.0	2.00	5.00	0.01	7.00	0.2
3.0	1.0	2.00	5.00	0.01	7.00	0.2

Table 6.13. The thresholds of parameters and the cluster radii of transformation components for data set 4.

Data Set 5						
Noise Level	Goodness of Match	Mean Error	Maximum Error	Cluster Radius of Components		
				Rotation	Translation	Scaling
0.0	0.1	0.01	0.01	0.05	0.05	0.1
0.5	1.0	2.00	5.00	0.02	2.00	0.2
1.0	1.0	2.00	5.00	0.01	5.00	0.2
1.5	3.0	4.00	10.00	0.05	10.00	0.3
2.0	3.0	4.00	10.00	0.05	10.00	0.3
2.5	3.0	4.00	10.00	0.05	10.00	0.3
3.0	3.0	4.00	10.00	0.05	10.00	0.3

Table 6.14. The thresholds of parameters and the cluster radii of transformation components for data set 5.

cluster in the feature space.

For each transformation, there are three components: rotation, translation and scaling. In view of different units of the components, the design of metric functions for measuring distances between transformations is formidable. There are two *ad hoc* choices. The first is to normalize transformations by making the sample means of components zero and the sample variances one. The second method is to consider components individually, which means applying the clustering procedure to individual components and then making the decision based on the integration of their results.

We adopt Stockman's clustering approach [Sto87] here. For convenience, from now on let us treat transformations (or their components) simply as *patterns* which are in turn represented by points in a feature space. Also, for simplicity of illustration, the feature space will be depicted as a two dimensional abstract space. Refer to Figure 6.4.

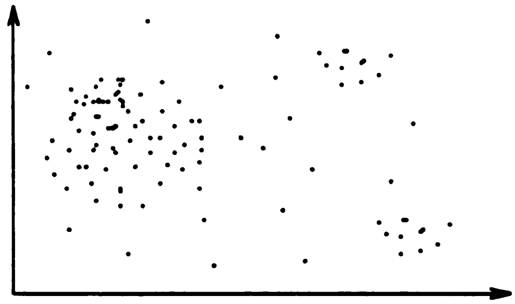


Figure 6.4. An abstract 2-D feature space of transformation for illustration.

The clustering procedure uses overlapping volumes distributed according to patterns in the continuous feature space. This method simply counts the number of patterns within a fixed-size volume centered at each pattern. The pattern having the most neighbors forms the most feasible cluster whose center of gravity is regarded as the correct transformation parameters. An implementation of this procedure can be found in [Che87]. Since the size of volumes is an important factor which can substantially influence the final result, a closer investigation of how the cluster radius is formed is the topic of the following sections. Similar analysis can be found in [Gri88].

6.6.1 Radius of Cluster

Referring to Figure 6.4, it is clear that choosing an adequate radius for the cluster is very important. There are several situations shown in Figure 6.5, which could occur due to an inadequate radius selection. A small radius of cluster can result in multiple clusters (Figure 6.5(a)) and cause selection of transformation parameters that look good but are incorrect. Also, with either a too small (Figure 6.5(b)) or a too large radius (Figure 6.5(c)), the resulting center of gravity of the cluster may shift away from the true position. In order to study the effect of different cluster radii, we use the housing [Gri85] as an example model. Synthetic features are generated from this model to serve as sensed features.

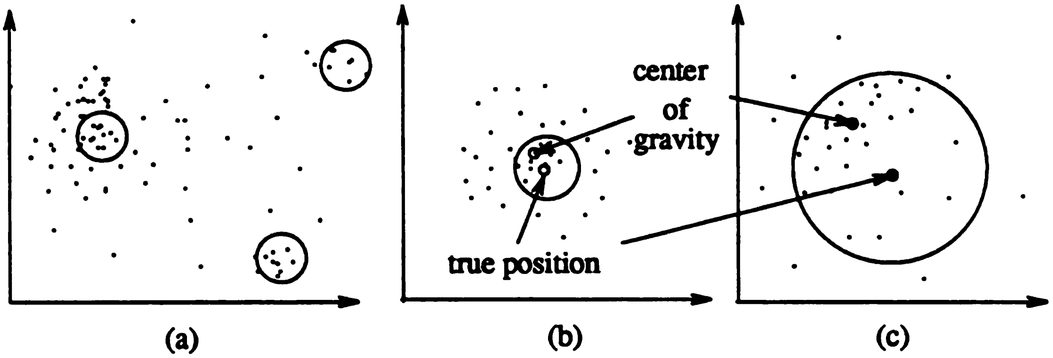


Figure 6.5. Situations could occur due to inadequate cluster radii : (a) a small radius of cluster can result in multiple clusters; (b) using a too small radius of cluster, the resulting center of gravity of the cluster may shift away from the true position; (c) using a too large radius of cluster.

The experiments proceed as follows.

1) Add noise to the sensed features. Noise is generated by a uniform random number generator.

2) Based on the model and the generated sensed features, a set of transformations can be computed. The details of this computation has been addressed elsewhere [Che87].

3) Apply the clustering procedure to the above set of transformations to determine the optimal one. The sizes of cluster for rotation and translation components are also recorded. The size of a cluster is defined as the cardinality of the cluster.

4) Different cluster radii have been used by the clustering procedure. For the rotation component of transformations, the cluster radius ranges from 0.01 to 0.1 (amounts to about 0.6 to 6.0 degrees), whereas for the translation component, the radius ranges from 5.5 to 10.0 mm.

5) The resulting transformation is then applied to the model features. By comparing the transformed model features to the sensed features, a matching error in terms of squared errors of model-sensed feature pairs can be computed.

6) The above steps are repeated using different levels of noise ranging from 0.0 to 4.0 mm.

6.6.2 Experimental Results and Discussion

Since transformation components are considered individually, Table 6.15 shows the matching errors using different cluster radii of rotation component by fixing the radius of translation component at 3.0mm and Table 6.16 shows the cluster sizes. Table 6.17 shows the matching errors using different cluster radii of translation component by fixing the radius of rotation component at 0.088 (about 5 degrees) and Table 6.18 shows the corresponding cluster sizes. Note that the values of 3.0 and 0.088 are estimated based on the error analysis of calibration and transformations presented in Chapter 4, which coincide with our observed experimental errors.

Several critical points of interest in this experiment are summarized below.

(1) As shown in Figure 6.6, for a given noise level, there is always a minimum match error which occurs at some cluster radius, called the *adequate radius*. Let us refer to a curve of matching error *versus* radius of cluster as an *error curve*. The point that corresponds to the minimum error is called the *trough* of the curve. For example, in Table 6.15, for the noise level 1.5, the minimum match error is 0.699, which occurs at radius 0.08. Thus, 0.08 is the adequate cluster radius for the rotation component at noise level 1.5. Using radii either smaller or larger than the adequate radius will result in a larger match error.

Matching Error	Cluster Radius of Rotation Component									
Noise Level	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.1
0.0	0.023	0.023	0.023	0.023	0.023	0.023	0.023	0.023	0.028	0.028
0.5	1.432	0.669	0.604	0.570	0.899	0.987	1.332	1.243	1.194	1.194
1.0	0.573	1.097	1.320	1.234	1.234	1.223	1.071	1.059	1.059	1.059
1.5	0.966	0.972	0.988	0.846	0.841	0.800	0.701	0.699	1.049	1.049
2.0	1.568	1.461	0.849	0.844	0.840	0.839	0.952	0.910	0.887	1.362
3.0	2.728	2.652	2.522	2.553	2.361	2.240	2.185	2.145	2.355	2.253
3.5	2.982	2.982	2.982	2.982	2.973	3.877	3.877	3.877	3.233	2.784
4.0	2.787	2.787	3.095	3.591	3.587	3.207	3.206	3.167	3.167	3.334

Table 6.15. The matching errors using different radii of rotation cluster by fixing the radius of translation cluster at 3.0mm.

Cluster Size	Cluster Radius of Rotation Component									
Noise Level	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.1
0.0	168	168	168	168	168	168	168	168	168	168
0.5	26	47	52	56	103	110	157	164	164	164
1.0	52	84	144	148	148	148	153	153	153	153
1.5	32	35	37	41	46	49	51	52	74	74
2.0	29	34	64	67	70	70	72	75	77	92
3.0	24	27	28	28	29	30	33	37	42	44
3.5	25	25	25	25	26	49	49	49	57	62
4.0	24	24	27	52	74	74	74	75	75	98

Table 6.16. The cluster sizes for rotation component when fixed the cluster radius of translation component at 3.0mm.

(2) If we draw error curves for all different levels of noise in a figure as depicted in Figure 6.7, the troughs of the curves should theoretically tend toward larger radii as the noise level increases because computed transformations get random. This phenomenon hasn't prominently been observed in our experimental results. A possible reason is that the experimental ranges of both noise level and cluster radius hasn't been wide enough.

(3) Matching errors tend to increase as noise increases (look at the tables of match error vertically). There are only a few cases whose matching errors are larger than their

Matching Error	Cluster Radius of Translation Component									
Noise Level	5.5	6.0	6.5	7.0	7.5	8.0	8.5	9.0	9.5	10.0
0.0	0.790	0.997	0.767	0.943	0.943	0.726	0.726	0.726	1.097	1.088
0.5	1.232	1.267	1.243	1.322	1.359	1.338	1.372	1.315	1.354	1.335
1.0	0.994	0.989	1.002	1.024	1.054	1.052	1.055	1.095	1.119	1.225
1.5	2.106	2.019	1.983	2.106	2.086	1.988	1.969	1.969	2.039	2.017
2.0	0.746	0.757	0.772	0.909	0.859	1.033	1.033	0.848	1.275	1.054
3.0	1.974	1.943	2.003	1.984	1.984	1.984	1.984	1.984	1.984	1.988
3.5	3.340	3.436	3.436	3.546	3.523	3.385	3.385	3.416	3.416	3.440
4.0	3.542	3.534	3.750	3.815	3.657	3.550	3.583	3.196	3.298	3.227

Table 6.17. The matching errors using different radii of translation cluster by fixing the radius of rotation cluster at 0.088 (about 5 degrees).

Cluster Size	Cluster Radius of Translation Component									
Noise Level	5.5	6.0	6.5	7.0	7.5	8.0	8.5	9.0	9.5	10.0
0.0	137	139	141	146	146	148	148	148	150	153
0.5	85	90	91	95	98	100	106	113	115	117
1.0	111	114	117	120	124	126	128	130	131	134
1.5	38	39	40	42	42	48	49	49	50	59
2.0	45	47	48	50	53	55	55	56	58	59
3.0	19	23	27	29	29	29	29	29	29	30
3.5	18	20	20	22	24	28	28	28	28	30
4.0	27	31	32	36	38	42	44	46	48	51

Table 6.18. The cluster sizes for translation component when fixed the cluster radius of rotation component at 0.088 (about 5 degrees).

initially given noise, most are smaller (look at the tables horizontally). This situation is particularly notable at high levels of noise. It means that the initial noise (or equivalently the sensory errors) haven't been amplified in the final results. they have been slightly attenuated by the clustering procedure.

(4) Both Table 6.16 and 6.18 reveal that the cluster size (cardinality) is increased as the cluster radius increases. According to the analysis of computational error in Chapter 4, the errors caused by translation are larger than those of rotation given the same initial

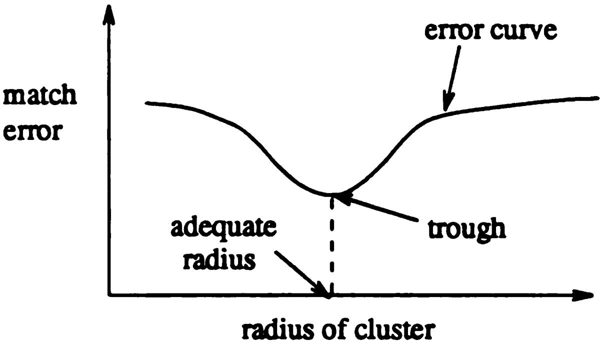


Figure 6.6. For a given noise level, there is always a minimum match error which occurs at some radius of cluster, called the *adequate radius*. The curve is called the *error curve* and the point corresponds to the minimum error is called the *trough* of the curve.

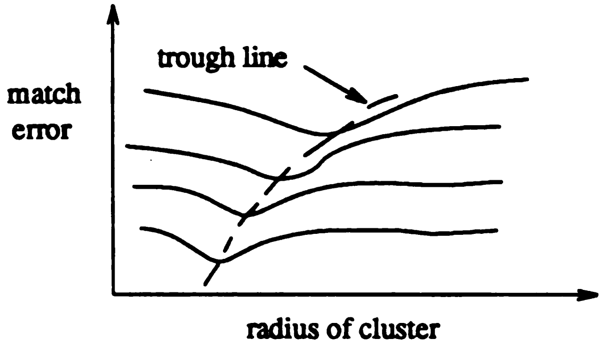


Figure 6.7. The troughs of error curves should tend toward larger radii as noise level increases.

error in feature points. Clusters of the rotation components should be more compact than clusters of the translation components. This has been observed in our experimental results (Table 6.16 and 6.18). As a consequence, clusters of the rotation components provide stronger evidence than the translation components in determining the true cluster.

(5) Another aspect of interest is that the size of a cluster can suddenly increase at some radius. For example, in Table 6.16, at noise level 0.5, there is a jump from 56 to 103 corresponding to radii 0.04 and 0.05. Figure 6.8 shows an example of cluster size jump. The radius at which the cluster size changes abruptly usually coincides with the adequate

radius determined in the figure of error curves. For instance, in Table 6.15, looking at the noise level 0.5, there is a minimum matching error (0.57) at radius of 0.04.

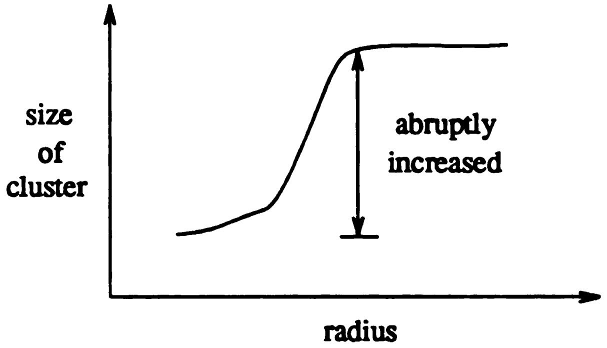


Figure 6.8. The size of a cluster suddenly increases at some radius.

(6) If we connect the troughs of error curves at distinct levels of noise, a *trough line* may be obtained. An analytical equation of the trough line can be calculated by regression methods. Later given a matching error related to noise, an estimate of the adequate radius can be determined immediately and provide some suggestion of where an adequate radius should be. We leave the research on automatic determination of adequate radius for future study.

6.7 Summary

This chapter started with a general review of current recognition techniques and then defined the problem and the steps of object recognition based on wing representations. These steps include: indexing into the model base, consistent labeling, parameter estimation and decision. During indexing, wing primitives play an important role in discriminating objects and predicting sensory views. Each candidate view is then examined by a model-test procedure which is a combination of the interpretation tree search, view transformation and a clustering procedure.

A pruning algorithm has been presented to implement the IT search. The expected number of surviving paths and the time complexity of the algorithm have been closely investigated. It is this investigation that provides a way to study the power of proposed pruning constraints. The pruning power of a constraint is specified by the probability that object features pass the constraint. Probabilities can be obtained by solving a linear system of equations and the smaller a probability is, the more powerful the constraint will be. A computational model was given to estimate the parameters of a view transformation. Each computed transformation comes with measurements of goodness of match, mean and maximum errors. Object pose is computable for sculptured objects provided that a finely tesselated viewsphere is used. At this point in time we do not have a procedure for refining a pose gotten via indexing to a coarse viewsphere. The class and pose of an unknown object were determined by a clustering procedure which also reports a set of statistics to indicate the confidence in recognition results. Finally, a study of cluster radius indicates that the clustering procedure hadn't amplified the initial sensory errors in the final results.

Chapter 7

Summary, Conclusions and Future Extensions

The entire work is summarized and conclusions are drawn from the experimental results. Major contributions are then discussed and possible extensions of the research are addressed.

7.1 Summary and Conclusions

The research in this thesis was concerned with a restricted recognition problem which can be stated as follows. Given a range image of a scene, which can be a jumble of either distinct or similar rigid objects, specific objects in the scene are to be recognized based on the given single image and a set of object models. Two major tasks were included: identification (recognize the object class) and localization (determine the object pose). The objects could be of arbitrary shape.

To solve a recognition problem, researchers must first understand the object domain and create a database containing the representations of these objects. Then algorithms are developed to extract object features in sensory data and construct a representation comparable to the internal representations in terms of the detected features. Finally, procedures are developed to recognize objects based on their representations. There are three important tasks which have traditionally been considered to be formidable in a recognition problem. The first is the representation of objects, especially irregularly shaped objects. The second is the development of feature extraction procedures. The third is to develop a recognition procedure based on the object representations. In this thesis, we have paid more attention to the issues of representation and recognition, and less to the feature extraction.

In order to represent objects, a new set of primitives, called *object wings* was proposed. There are two categories of wings: simple wings and composite wings. A *simple wing* is defined as a triple including a pair of surface patches separated by a contour segment, while composite wings are formed by grouping simple wings through non-accidental relationships such as cotermination, symmetry, parallelism, connectivity, collinearity and curvilinearity. Thirty-four simple wings were defined to serve as prototypes for practical use. This set of new primitives possesses a higher level of structure than other commonly used features such as vertices, edges, surfaces and critical points. We thus make the conclusion that object wings together with their spatial structures provide more information than the aforementioned features for both indexing into a model database and computing object pose.

After closely examining the properties of wing primitives and their structural relations, we studied the adequacy of wing representation for polyhedral scenes. That is, given the wing representation of a scene containing polyhedral objects, we can reconstruct all visible parts of objects based on the given wing representation - that is the coordinates of all visible vertices, the equations of all visible faces, and the completely labeled line drawing of the scene. To this end, we demonstrated that if a wing representation has been derived from the image of a polyhedral scene, the spatial structure of the scene could be uniquely and correctly recovered from the wing representation.

Then, we turned to the central issues of how to represent objects using the proposed wing primitives. A set of objects, whose surfaces range from polyhedral to arbitrary shapes, were used as examples for object representation. This set of objects included a block, a coffee cup and a man-made clay cobra head. Since wing primitives are not view-invariant, a multiple-view scheme called a *viewsphere* was adopted. Several viewspheres with different tessellations of 20, 80, 140, 240 and 320 viewpoints were studied for the purpose of multiple-view representation. It was proposed that each object be represented in terms of hierarchical aspects which are characterized by "scaled" wing sets: common wing set, distinguished wing set and distinct wing set.

We examined the properties of uniqueness, stability, compatibility, terseness and locality of wing representations. The testing of wing representation with respect to these criteria was done by investigating the influence of convolution filters, view resolution and map resolution on the resulting representations. In order to inspect the compatibility of wing representations, we have implemented several segmentation techniques and developed methods to extract wings from sensory data. The results revealed that the wings detected in real images are comparable to those of internal representations in the database which were made from models. We concluded that wing representations have the properties of uniqueness, stability, locality and compatibility for the object domain tested. However, the terseness property is not satisfied by current wing representation due to the multiple-view scheme of viewsphere. It seems possible that this problem can be solved by using low map resolutions and coarse viewspheres. Also, the inherent hierarchy of the viewsphere and the "scaled" wing sets can be combined to achieve more compactness of representation.

In order to show that objects can be identified and located through matching wing representations of scenes and models, we developed a recognition procedure that has been divided into four stages: indexing, consistent labeling, parameter estimation and decision. Techniques for dealing with these stages included feature comparison, interpretation tree search, curve matching and clustering. Several properties of these algorithms were also investigated. The experimental results encourage us to make the conclusion that 3-D object recognition can be achieved based on wing representations.

7.2 Contributions

The most important contribution of this thesis was the introduction of wing representation theory and the associated computational framework. It is within this framework that techniques of representation and recognition of arbitrarily-shaped 3-D rigid objects were developed. A major purpose for proposing this theory is that previous theories were either too restricted to deal with general objects or too idealized to implement in machine vision systems. To compensate for these two difficulties, wing representation theory was proposed, which includes three major components: a set of wing primitives, a set of wing constraints and a multiple-view data structure called viewsphere. This theory was designed to (1) handle general rigid 3-D objects, (2) overcome imperfect feature extraction, (3) index to object models and (4) compute object pose.

Based on wing representation theory, a series of experiments were performed to answer questions on how to extract wing features from synthetic images for construction of the object representation, what kind of geometrical structure is suitable for wing features, how to compute these structures, and how to detect comparable object wings from real images. Although current results are by no means exceptional, they are good enough to show that the construction of representations in terms of object wings is feasible from both theoretical and practical viewpoints.

Another contribution in this research is the design of a procedure to recognize scene objects through their wing representations. There are two requirements which make the development of the recognition procedure difficult. The first was that both problems of identification and localization must be handled. The second requirement was that the recognition procedure should be able to deal with the multiple-view representation scheme together with multiple object features. It is desirable that based on this prototype recognition procedure, better ones could be designed by either improving the current method or designing more advanced approaches such as automatic programming [Goa83,Ike88] and connectionism [Fel81,Sab82].

In wing representation theory, eight unary structural constraints and five binary relational constraints have been defined. A viewsphere combined with this set of wing constraints and wing primitives form the essential ingredients of the theory. In order to test the appropriateness of wing constraints, the probabilistic models for estimating the expected numbers of surviving paths and constraint checks were constructed in terms of a pruning algorithm. These models provided a way to study the pruning power of wing constraints so that redundant and weak constraints may be discarded. These probabilistic models formed a contribution to the design of wing representation theory.

Finally, we discovered that image segmentation based on the signs of the directional derivative has several properties of interest. Since the computation of directional derivatives is easy and the signs of the derivative are stable, segmentation results are fairly reliable. Although contours and regions extracted by this method may not always correspond to physical counterparts, like object silhouettes, these contours and regions are sensitive to view perturbation. Object silhouettes have long been regarded as useful clues for object recognition. Derivatives in any direction can be easily computed from a pair of orthogonal directional derivatives using the following equation

$$\frac{df}{d\mathbf{r}} = f_u \cos\theta + f_z \sin\theta \; ,$$

where $\mathbf{r}$ is the scanning vector and θ is its directional angle. The sign map of the derivative with respect to any direction can be efficiently predicted. This property together with other properties of rotational equivalence (Section 5.4.2.2) and computational reliability reveal the potential that object wings may suitably be defined based on those contours and regions extracted by this particular segmentation technique. To this end, further study of directional derivatives will be needed.

Secondary contributions of this research include the following:

(1) the reconstruction of labeled line drawings from wing representations of polyhedral scenes, a byproduct of demonstrating that wings are an adequate representation for polyhedral objects, (Section 2.6, Appendix G),

(2) the use of a structured light sensing system for constructing geometric models (Che85, Section 4.1.1),

(3) the development of an approach to augmenting geometric models such that model vertices contain 3-D coordinates and a set of surface characteristics (Section 4.2),

(4) the analysis of errors resulting from sensor calibration and transformation computation (Sections 4.1.3.2, 4.1.3.3),

(5) the introduction of a general model for estimating computational errors caused by different sizes of convolution masks (Section 5.2.4),

(6) the extension of Schwartz & Sharir's curve matching by including a scale factor (Section 6.5.1, Appendix F),

(7) the empirical demonstration that initial sensory errors are nut amplified in the final results of pose computation through the clustering procedure (Section 6.6).

7.3 Future Work

Experimental results support the thesis that simple wings can be reasonably extracted from both synthetic images (for representation) and real images (for recognition), and that objects can be recognized through their wing representations. These results encourage us to go further in the proposed representation theory and its implementation. In the previous work, we have deliberately impoverished object wings during experimentation, considering only a collapsed set of thirty four simple wings. Other categories of wings, through either augmenting or collapsing the current wing set, need to be investigated. In order to improve object representation and object recognition, another part of wing theory, concerning composite wings, should be carried out.

The use of composite wings in representation can reduce the storage complexity of the current representation in terms of simple wings because composite wings are higher level features than simple wings. In addition, indexing models using composite wings will be more effective than using simple wings so as to increase the efficiency of recognition. The detection of non-accidental relationships among simple wings, such that simple wings can be grouped into composite wings according to some relationships, will form an important research topic which will be closely related to perceptual organization.

It is known that if object segmentation (high level image segmentation) can be completed for an image, object recognition based on the segmented image will become much easier. Of course, recognition and segmentation can proceed simultaneously. Since current segmentation techniques have difficulties with feature extraction, wing maps derived from images are typically imperfect. By imperfect wing maps, we mean that the wing maps have either missing wings or spurious wings or both. Based on such a kind of wing maps, there are two directions to follow such that high level segmentation of images taken from scenes with multiple objects might be achieved.

The first direction to follow is to look for composite wings in a given wing map. As mentioned in Section 2.4, a composite wing is composed of a set of simple wings which belong to a single physical object. Once composite wings can be successfully extracted, they will be further clustered into groups so that each group corresponds to a single physical object. Object segmentation of images can then be done by recovering visible parts of objects based on groups of composite wings.

The second line to follow is to reconstruct labeled line drawings from wing maps. Since a labeled line drawing represents the important spatial structure of scene, object segmentation can be accomplished by interpreting the labeled line drawing. In this thesis, we have studied the reconstruction of labeled line drawings from wing maps of polyhedral scenes (Chapter 2 and Appendix G). Further study of scenes containing curved objects will be necessary. Unlike line drawings, wing maps suffer from imperfect object wings. On the other hand, wing maps contain such additional information as range values and surface normals, information which is not available for line drawings. In practice, wing maps possess more powerful capability in interpreting scenes and are more realistic than pure line drawings.

There are other potential tasks that will also require significant effort. The first task may be how to remove the terseness problem currently existing in a multiple-view representation scheme. Using "scaled" wing sets along with the hierarchical scheme inherently existing in the viewsphere (Chapter 5) or using fewer aspects as representatives [Bas88] are two potential ways to resolve the terseness problem. The second research topic is how to automatically construct wing representations in terms of multiple views (aspects) for 3-D physical objects. Many researchers [Pla87, Gig88] have succeeded in automatically constructing the aspect graphs for polyhedral objects but only a few [Koe76,79,87, Bow89] have considered curved objects. Another topic of research involves looking for efficient controls and strategies for indexing into a model database.

Tactics from connectionism [Fel81, Sab82], geometric hashing [Lam88], or precompiling and automatic programming [Goa83, Ike88] may provide us with the ideas of how to organize modules in our recognition system using object wings. Another interesting topic of research is to explore methods of pose computation with fewer steps. The technique of *3-D from 2-D* [Sho88], which does not need depth recovery, might accomplish this.

APPENDIX A

Computational Methods

A.1 Computation of 3-D Coordinates

In a structured light sensing system, given the calibration matrices of the camera **C** and the projector **P**, and the coordinates of both image (x_i, y_i) and grid (x_s, y_s) of a point located at a light stripe, the 3-D coordinates (x_w, y_w, z_w), of that point can be computed as follows.

$$\mathbf{C} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \end{bmatrix}, \quad \mathbf{P} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix}.$$

The following relation holds.

$$\begin{bmatrix} tx_i \\ ty_i \\ t \end{bmatrix} = \mathbf{C} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}, \text{ and } \begin{bmatrix} sx_g \\ sy_g \\ s \end{bmatrix} = \mathbf{P} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

After extension and rearrangement, the scale factors s and t can be removed, and the following linear system of equations is obtained.

$$(c_{31}x_i - c_{11})x_w + (c_{32}x_i - c_{12})y_w + (c_{33}x_i - c_{13})z_w = c_{14} - c_{34}x_i$$

$$(c_{31}y_i - c_{21})x_w + (c_{32}y_i - c_{22})y_w + (c_{33}y_i - c_{23})z_w = c_{24} - c_{34}y_i$$

$$(p_{31}x_s - p_{11})x_w + (p_{32}x_s - p_{12})y_w + (p_{33}x_s - p_{13})z_w = p_{14} - p_{34}x_s$$

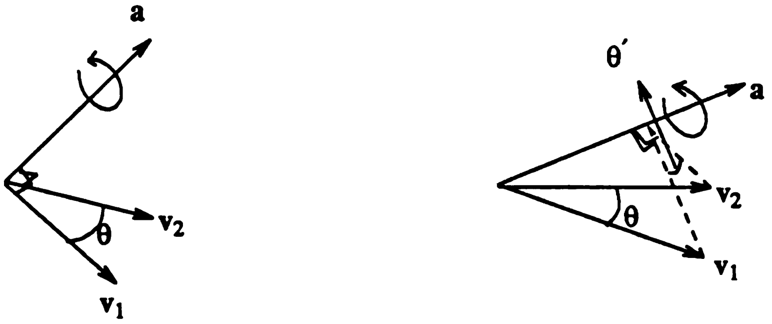
$$(p_{31}y_s - p_{21})x_w + (p_{32}y_s - p_{22})y_w + (p_{33}y_s - p_{23})z_w = p_{24} - p_{34}y_s$$

Here we have four equations to solve the three unknows x_w, y_w, and z_w. However, sometimes a feature point, such as a termination, only provides grid coordinates of either x_s or y_s. Only three equations are available, but still enough to solve for the three unknowns.

A.2. Homogeneous Transformation

Let T(v) denote a translation, which translates a geometrical element from its current position **r** to a new position **r** + **v**, where **v** = (v_x, v_y, v_z).

$$\mathbf{T}(\mathbf{v}) = \begin{bmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{(A.2.1)}$$



(a) perpendicular case (b) nonperpendicular case

Figure A.1. Rotation transformations.

Referring to Figure A.1, let θ be the subtended angle between vectors $\mathbf{v}_1$ and $\mathbf{v}_2$, and $\mathbf{R}(\mathbf{a}, \mathbf{v}_1, \mathbf{v}_2)$ be the rotation, which rotates $\mathbf{v}_1$ to $\mathbf{v}_2$ about the axis $\mathbf{a} = (a_x, a_y, a_z)$. If the rotation axis **a** is perpendicular to the plane spanned by the vectors $\mathbf{v}_1$ and $\mathbf{v}_1$ (Figure A.1(a)), the rotation transformation [Fau79, Pau81] is

$$\mathbf{R}(\mathbf{a}, \mathbf{v}_1, \mathbf{v}_2) = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad (A.2.2)$$

where

$$\begin{aligned}
r_{11} &= a_x^2\,(1-cos\theta) + \cos\theta \\
r_{21} &= a_x a_y\,(1-cos\theta) + a_z \sin\theta \\
r_{31} &= a_x a_z\,(1-cos\theta) - a_y \sin\theta \\
r_{12} &= a_y a_x\,(1-cos\theta) - a_z \sin\theta \\
r_{22} &= a_y^2\,(1-cos\theta) + \cos\theta \\
r_{32} &= a_y a_z\,(1-cos\theta) + a_x \sin\theta \\
r_{13} &= a_z a_x\,(1-cos\theta) + a_y \sin\theta \\
r_{23} &= a_z a_y\,(1-cos\theta) - a_x \sin\theta \\
r_{33} &= a_z^2\,(1-cos\theta) + \cos\theta
\end{aligned}$$

However, if the rotation axis **a** is not perpendicular to the plane spanned by the vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ (Figure A.1(b)), the θ in the above equations should be replaced by θ'.

$$cos\theta' = \frac{[v_2-(v_2 \cdot k)k][v_1-(v_1 \cdot k)k]}{|v_2-(v_2 \cdot k)k||v_1-(v_1 \cdot k)k|}. \qquad (A.2.3)$$

APPENDIX B

Construction of Viewspheres

B.1. Viewsphere with 320 Viewpoints

Without loss of generality, assume the viewsphere is a unit sphere. The tessellation starts with an icosahedron [Bal82, Bas88], which has 12 vertices, 20 faces and 30 edges. Define

$$a = \frac{\sqrt{gr}}{5^{1/4}}, \quad b = \frac{1}{\sqrt{gr} \cdot 5^{1/4}}, \quad c = \frac{1}{b}, \quad d = a + b,$$

where gr is the golden ratio; $gr = \dfrac{1+\sqrt{5}}{2}$. The coordinates of 12 vertices are

$$[0, a, b], [0, a, -b], [-a, b, 0], [-b, 0, -a], [-a, -b, 0], [0, -a, -b],$$

$$[0, -a, b], [a, -b, 0], [b, 0, a], [a, b, 0], [-b, 0, a], [b, 0, -a].$$

For each triangle of the icosahedron, a geodesic tessellation of frequency two splits it into four small triangles. This process is repeated once more after the newly created vertices have been projected onto the sphere surface. As a consequence, a total of $20 \times 4 \times 4 = 320$ cells can be obtained. We call these cells the *viewpoints* of the viewsphere.

B.2. Viewsphere with 240 Viewpoints

Starting with a dodecahedron [Yan85] that has 12 pentagons and 20 vertices, define

$$a = 0.7136, \quad b = 2a \cos 36° \cos 18°, \quad c = \sin(\cos^{-1}\frac{a}{2}), \quad d = a \sin 54°.$$

We can immediately obtain the coordinates of those 20 vertices as

$$V_1 = (\frac{-a}{2}, c, 0), \; V_2 = (\frac{a}{2}, c, 0), \; V_3 = (-c, 0, \frac{a}{2}), \; V_4 = (c, 0, \frac{a}{2}),$$

$$V_5 = (0, \frac{a}{2}, c), \; V_6 = (0, \frac{-a}{2}, c), \; V_7 = (-d, d, d), \; V_8 = (d, d, d),$$

$$V_9 = (-d, -d, d), \; V_{10} = (d, -d, d), \; V_{11} = (\frac{-a}{2}, -c, 0), \; V_{12} = (\frac{a}{2}, -c, 0),$$

$$V_{13} = (-c, 0, \frac{-a}{2}), \; V_{14} = (c, 0, \frac{-a}{2}), \; V_{15} = (0, \frac{a}{2}, -c), \; V_{16} = (0, \frac{-a}{2}, -c),$$

$$V_{17} = (-d, d, -d), \; V_{18} = (d, d, -d), \; V_{19} = (-d, -d, -d), \; V_{20} = (d, -d, -d),$$

and 12 pentagons in terms of the vertices are

(1 2 8 5 7), (1 7 3 13 17), (2 1 17 15 18), (4 8 2 18 14), (5 8 4 10 6),

(5 6 9 3 7), (16 15 17 13 19), (14 18 15 16 20), (6 10 12 11 9),

(19 13 3 9 11), (10 4 14 20 12), (11 12 20 16 19).

For each pentagon, divide it into five triangles according to its five edges. Then, a geodesic tessellation of frequency two splits each triangle into four finer triangles. We obtain 12 × 5 ×4 = 240 cells.

APPENDIX C

Proof of Theorem 4.1

Theorem 4.1: If a line segment $\overline{o_1 o_2}$ with positional error of at most r_o for each of its endpoints is rotated, the endpoints of the rotated line segment have a relative positional error bound.

$$p' \le 2\sqrt{1-(1-p^2)^{1/2}} + p\sqrt{3}. \tag{4.2}$$

where $p = \frac{2r_o}{d}$, $p' = \frac{2r_o'}{d}$ and d is the length of the line segment, $d = |\overline{o_1 o_2}|$. r_o' is the new positional error after rotating.

Pŗoof: Refer to Figure C.1. Suppose a line segment l is to be rotated by an angle θ about the axis $\mathbf{u}$ in order to align with l'. Due to sensing error the observed line segment l'' of the line segment l has an angle error $\Delta\theta$ with l. The above rotational operation is in fact applied to l'' rather than l. In order to align l'' with l', l'' will be rotated about a new axis $\mathbf{u}'$ by an angle about θ+$\Delta\theta$.

Let the correct rotation matrix be $\mathbf{R}$ and the computed rotation matrix be $\mathbf{R}'$. From the equation of a rotation transformation,

$$\mathbf{R} = \begin{bmatrix} u_1^2+(1-u_1^2)cos\,\theta & u_1u_2(1-cos\,\theta)-u_3 sin\,\theta & u_3u_1(1-cos\,\theta)+u_2 sin\,\theta \\ u_1u_2(1-cos\,\theta)+u_3 sin\,\theta & u_2^2+(1-u_2^2)cos\,\theta & u_2u_3(1-cos\,\theta)-u_1 sin\,\theta \\ u_3u_1(1-cos\,\theta)-u_2 sin\,\theta & u_2u_3(1-cos\,\theta)+u_1 sin\,\theta & u_3^2+(1-u_3^2)cos\,\theta \end{bmatrix},$$

then,

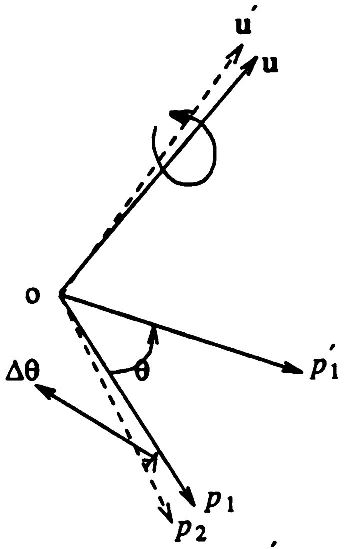


Figure C.1. A rotational operation is to rotate l to l' about the axis **u** by an angle θ.
Due to sensing error the rotation is in fact applied to l'' about a new axis **u**$'$ by an angle about θ+Δθ.

$$\mathbf{\Delta R} = \mathbf{R}' - \mathbf{R} = \begin{bmatrix} \Delta r_{11} & \Delta r_{12} & \Delta r_{13} \\ \Delta r_{21} & \Delta r_{22} & \Delta r_{23} \\ \Delta r_{31} & \Delta r_{32} & \Delta r_{33} \end{bmatrix} . \tag{C.1}$$

where

$$\Delta r_{11} = (1 - u_1^2)\ [(\sqrt{1-p^2} - 1)\ cos\,\theta - p\ sin\,\theta]$$

$$\Delta r_{12} = u_1 u_2\ [(1 - \sqrt{1-p^2})\ cos\,\theta + p\ sin\,\theta] - u_3\ [p\ cos\,\theta - (1 - \sqrt{1-p^2})\ sin\,\theta]$$

$$\Delta r_{13} = u_1 u_3\ [(1 - \sqrt{1-p^2})\ cos\,\theta + p\ sin\,\theta] + u_2\ [p\ cos\,\theta - (1 - \sqrt{1-p^2})\ sin\,\theta]$$

$$\Delta r_{21} = u_2 u_1\ [(1 - \sqrt{1-p^2})\ cos\,\theta + p\ sin\,\theta] + u_3\ [p\ cos\,\theta - (1 - \sqrt{1-p^2})\ sin\,\theta]$$

$$\Delta r_{22} = (1 - u_2^2)\ [(\sqrt{1-p^2} - 1)\ cos\,\theta - p\ sin\,\theta]$$

$$\Delta r_{23} = u_2 u_3\ [(1 - \sqrt{1-p^2})\ cos\,\theta + p\ sin\,\theta] - u_1\ [p\ cos\,\theta - (1 - \sqrt{1-p^2})\ sin\,\theta]$$

$$\Delta r_{31} = u_3 u_1\ [(1 - \sqrt{1-p^2})\ cos\,\theta + p\ sin\,\theta] - u_2\ [p\ cos\,\theta - (1 - \sqrt{1-p^2})\ sin\,\theta]$$

$$\Delta r_{32} = u_3 u_2\ [(1 - \sqrt{1-p^2})\ cos\,\theta + p\ sin\,\theta] + u_1\ [p\ cos\,\theta - (1 - \sqrt{1-p^2})\ sin\,\theta]$$

$$\Delta r_{33} = (1 - u_3^2)\ [(\sqrt{1-p^2} - 1)\ cos\,\theta - p\ sin\,\theta]$$

Without loss of generality, assume point **o** in Figure C.1 is the origin of a reference frame. All line segments have been translated so that each has one of its endpoints is located at the origin **o** of the reference frame. Let p_1 be another endpoint of l, p_1' be another endpoint of l' and p_2 be another endpoint of l''. Referring to Figure C.2, originally p_1 is to be rotated by **R** to align with p_2. In real situation, the observed point p_1' with error r_o is rotated by $\mathbf{R}'$ to the point p_2'. There is an error r_o' between p_2 and p_2', called rotational error, which is to be estimated.

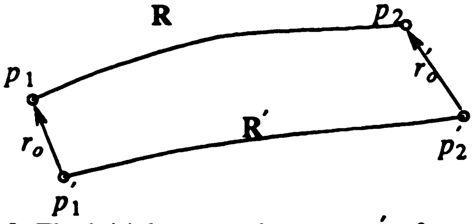


Figure C.2. The initial error r_o becomes r_o' after a rotation.

Let $p_2 = \mathbf{R} \cdot p_1$, and $p_2' = \mathbf{R}' \cdot p_1'$.

Then,

$$\mathbf{r}_o' = p_2' - p_2 = \mathbf{R}' \cdot p_1' - \mathbf{R} \cdot p_1 = \mathbf{R}' \cdot (p_1 + \mathbf{r}_o) - \mathbf{R} \cdot p_1$$

$$= \mathbf{R}' \cdot p_1 + \mathbf{R}' \cdot \mathbf{r}_o - \mathbf{R} \cdot p_1 = (\mathbf{R}' - \mathbf{R}) \cdot p_1 + \mathbf{R}' \cdot \mathbf{r}_o = \Delta\mathbf{R} \cdot p_1 + \mathbf{R}' \cdot \mathbf{r}_o ,$$

and

$$|\mathbf{r}_o'|_2 = |\Delta R \cdot p_1 + \mathbf{R}' \cdot \mathbf{r}_o|_2 \le |\Delta R \cdot p_1|_2 + |\mathbf{R}' \cdot \mathbf{r}_o|_2$$

$$\le |\Delta R|_2 \, |p_1|_2 + |\mathbf{R}'|_2 \, |\mathbf{r}_o|_2$$

Since the rotation matrix is an orthonormal matrix,

$$|\mathbf{R}'|_2 = \sqrt{3} \quad \text{and} \quad |p_1|_2 \sim \frac{d}{2}.$$

We obtain $\dfrac{2r_o'}{d} \le |\Delta R|_2 + \sqrt{3}\dfrac{2r_o}{d}$.

Let $p' = \dfrac{2r_o'}{d}$ 1n' and $p = \dfrac{2r_o}{d}$. Then, $p' \le |\Delta R|_2 + p\sqrt{3}$.

Since $|\Delta R|_2 = (\sum_{i=1}^{n}\sum_{j=1}^{n}\Delta r_{ij}^2)^{1/2}$, from Equation (C.1)

$$|\Delta R|_2 = 2\sqrt{1-(1-p^2)^{1/2}}.$$

Finally we obtain $p' \leq 2\sqrt{1-(1-p^2)^{1/2}} + p\sqrt{3}$.

APPENDIX D

Criteria of Uniqueness

In this appendix, the criteria and their penalties for justifying the uniqueness property of wing representation are listed. A similarity function (in Chapter 5) and a priority function (in Chapter 6) are constructed based on the penalties. Definition of terminologies have been defined in Section 5.1.1.

Let s_i denote a penalty for two representations which violates the criterion i of uniqueness, and w_j be the weight pertaining to criterion j. Let $\mathbf{R}_k$ denote the wing representation of an object k, and A be the entire set of wing primitives.

Two representations $\mathbf{R}_1$ and $\mathbf{R}_2$ are not the same, if one of the following conditions is not the same.

(1) The numbers n_i of aspects of models i :

penalty: $s_1 = w_1 | n_1 - n_2 |$.

(2) The sets D^i of distinguished wing primitives of models i :

penalty: $s_2 = w_2 | (D^1 \cup D^2) - (D^1 \cap D^2) |$.

(3) The sets AD^i of appearance frequencies corresponding to the sets D^i of distinguished wing primitives of models i :

penalty: for $p_j \in$ A, if $p_j \in D^i$, then $A^i_j = AD^i_j$; otherwise $A^i_j = 0$.

$$s_3 = w_3 \sum_{j=1}^{|A|} | A^1_j - A^2_j |.$$

(4) The sets P^i of distinct wing primitives of models i :

penalty: $s_4 = w_4 |(P^1 \cup P^2) - (P^1 \cap P^2)|.$

(5) The sets AP^i of appearance frequencies corresponding to the sets P^i of distinct wing primitives of models i:

penalty: for $p_j \in$ **A**, if $p_j \in P^i$, then $A^i_j = AP^i_j$; otherwise $A^i_j = 0$.

$$s_5 = w_5 \sum_{j=1}^{|A|} |A^1_j - A^2_j|.$$

(6) The sets F^i of wing features of models i :

penalty: $s_6 = w_6 |(F^1 \cup F^2) - (F^1 \cap F^2)|.$

(7) The sets $\mathbf{SD}^k$ of distinguished wing primitives of aspects of models k :

Let SD^k_i be the set of distinguished wing primitives of an aspect i of the model k.

penalty: $s_7 = w_7 |(\mathbf{SD}^1 \cup \mathbf{SD}^2) - (\mathbf{SD}^1 \cap \mathbf{SD}^2)|.$

(8) The sets $\mathbf{SP}^k$ of distinct wing primitives of aspects of models k :

penalty: $s_8 = w_8 |(\mathbf{SP}^1 \cup \mathbf{SP}^2) - (\mathbf{SP}^1 \cap \mathbf{SP}^2)|.$

(9) The sets $\mathbf{SF}^k$ of wing features of aspects of models k :

Let SF^k_i denote the set of wing features of an aspect i of the model k.

penalty: $s_9 = w_9 |(\mathbf{SF}^1 \cup \mathbf{SF}^2) - (\mathbf{SF}^1 \cap \mathbf{SF}^2)|.$

(10) The sets R^k of structural relationships of models k :

penalty: $s_{10} = w_{10} |(R^1 \cup R^2) - (R^1 \cap R^2)|.$

(11) The sets $\mathbf{SR}^k$ of structural relationships of aspects of models k :

penalty: $s_{11} = w_{11} |(\mathbf{SR}^1 \cup \mathbf{SR}^2) - (\mathbf{SR}^1 \cap \mathbf{SR}^2)|.$

APPENDIX E

Interpretation Tree Search

Consider a general case of feature types $\{t_i, \mathrm{i} = 0,\mathrm{N}\}$. Suppose we are given a set of m sensed features, S_0 with type t_0, S_1 with type t_1, $\cdots$, and S_m with type t_m. Suppose also that we want to test this set of sensed features with a given model which has n_0 features with type t_0, n_1 features with type t_1, $\cdots$, and n_m features with type t_m. Thus, the carry array is (n_m+1, n_{m-1}+1, $\cdots$, n_1+1, n_0+1). A sketch of the path generation function is then given below.

Function : Path Generator

Objective: Generates the next candidate path from the current path.

Input: PATH: the current path array.
POS: the position of the path array at which the digit will be increased by one.
CARRY: the carry array.

Output: PATH: the generated path.
MARK: the most significant position of the path array where the digit is modified.
COMPLETE: reports whether or not the current path is the last path.

Procedure :

```
begin
    if (the current path is the last path) COMPLETE = yes; else COMPLETE = no;
    increase the digit at position POS of array PATH by one through referring to the CARRY
    array;
    mark the most significant position of PATH where the digit is changed;
end; (** end of procedure path generator **)
```

Suppose the constraints on both sensed and model features are available. There are two kinds of constraints: unary constraints and binary constraints. Unary constraints specify the properties of individual features and binary constraints illustrate the relational properties between a pair of features. Properties involving more features are not considered here. A function based on feature constraints to examine the legality of a path is sketched as follows.

Function : Path Check

Objective: Checks the legality of a path.

Input: PATH: the path array.
MARK: the highest position of the path array where the digit has been modified.
LENGTH: the length of the path.

Output: POS: the position of the path array where the consistency check failed.
LEGAL: reports whether or not the path is a legal interpretation.

Procedure :

```
begin
   POS = LENGTH - 1;
   for (i = MARK to LENGTH) do
      begin
         check consistency between sensed and model features using unary constraints;
         if (consistency fails) { POS = i; LEGAL = no; return; }
         for (j = i-1 downto 0) do
            begin
               check consistency between pairs of sensed and model features using binary
               constraints;
               if (consistency fails) { POS = i; LEGAL = no; return; }
            end;
      end;
   LEGAL = yes;
end; (** end of procedure path check **)
```

Using the path generator and the path check procedures, the pruning algorithm for interpretation tree search is given as follows.

Algorithm : Interpretation Tree Search

Objective: Looks for candidate interpretations.

Input: MODEL: model features.
SENSE: sensed features.

Output: NO: number of candidate interpretations.
PATHS: candidate interpretations.

Procedure :

```
begin
  LENGTH = number of sensed features; POS = LENGTH - 1; COMPLETE = no;
  set up CARRY array; set up initial PATH array;
  while (COMPLETE is no)
     begin
          call procedure PATH GENERATOR(PATH,POS,CARRY,MARK,COMPLETE);
          call procedure PATH CHECK(PATH,MARK,LENGTH,POS,LEGAL);
          if (LEGAL is yes) store PATH in PATHS;
     end;
end; (** end of algorithm interpretation tree search **)
```

APPENDIX F

Curve Matching

Consider a consistent pair of model wing L and detected wing segment l as shown in Figure 6.3. Let l' be that portion of L which is the corresponding segment of l by transformation **T**. Each curve is represented by a set of uniformly spaced points (equal distance intervals).

$$L = \{\mathbf{u}'_1, \mathbf{u}'_2, \cdots, \mathbf{u}'_n\},$$

$$l = \{\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_m\}, \quad where \;\; m \le n,$$

$$l' = \{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_m\}.$$

The matching problem can then be formulated as an optimization problem,

$$\delta = \min_{\mathbf{T}} \sum_{j=1}^{m} |\mathbf{T}\mathbf{u}_j - \mathbf{v}_j|^2 \;, \quad \mathbf{T}\mathbf{u}_j = \mathbf{R}_\theta s \mathbf{u}_j + \mathbf{a}, \tag{F.1}$$

where $\mathbf{R}_\theta$ is the rotation, **a** is the translation and s is the scale factor. Our purpose is to look for a transformation T such that minimizes the δ function. To simplify the derivation, assume the scale factor s is a constant. In a perspective projection, it is a nonlinear function of depth, but can be approximated by a constant when depth variation is small relative to standoff.

First translate by $\mathbf{a}_u$ the geometrical center of l to the origin of a reference coordinate frame.

$$\mathbf{a}_u = \frac{1}{m}\sum_{j=1}^{m}\mathbf{u}_j. \tag{F.2}$$

Without loss of generality, let us use the same symbols as those before this translation, so that

$$\sum_{j=1}^{m}\mathbf{u}_j = 0. \tag{F.3}$$

The Equation (F.1) becomes $\delta = \min_{\theta,\mathbf{a}',s} \sum_{j=1}^{m} |\mathbf{R}_\theta s \mathbf{u}_j + \mathbf{a}' - \mathbf{v}_j|^2$.

Expanding the above equation and using the result of Equation (F.3),

$$\delta = \min_{\theta,\mathbf{a}',s} [\, s^2\sum_{j=1}^{m}|\mathbf{u}_j|^2 + m|\mathbf{a}'|^2 + \sum_{j=1}^{m}|\mathbf{v}_j|^2 - 2\sum_{j=1}^{m}\mathbf{a}'\cdot\mathbf{v}_j - 2s\sum_{j=1}^{m}\mathbf{R}_\theta\mathbf{u}_j\cdot\mathbf{v}_j \,]. \tag{F.4}$$

Examining this equation, the translation $\mathbf{a}'$ is independent of θ and s and can be determined immediately.

$$\mathbf{a}' = \frac{1}{m}\sum_{j=1}^{m}\mathbf{v}_j \tag{F.5}$$

After substituting Equation (F.5) into (F.4), in order to minimize the resulting equation the following criteria should hold.

$$\nabla\delta = 0\,, \quad \text{and} \quad \det\begin{bmatrix} \frac{\partial^2\delta}{\partial s^2} & \frac{\partial^2\delta}{\partial\theta\partial s} \\ \frac{\partial^2\delta}{\partial s\partial\theta} & \frac{\partial^2\delta}{\partial\theta^2} \end{bmatrix} > 0.$$

Therefore, we obtain the intermediate scale factor:

$$s = \frac{\sum_{j=1}^{m}\mathbf{R}_\theta\mathbf{u}_j\cdot\mathbf{v}_j}{\sum_{j=1}^{m}|\mathbf{u}_j|^2}. \tag{F.6}$$

Substituting Equation (F.6) into (F.1), we obtain

$$\delta = \min_{\theta}[\sum_{j=1}^{m} |\mathbf{v}_j|^2 - \frac{1}{m}|\sum_{j=1}^{m}\mathbf{v}_j|^2 - \frac{(\sum_{j=1}^{m}\mathbf{R}_\theta\mathbf{u}_j\cdot\mathbf{v}_j)^2}{\sum_{j=1}^{m}|\mathbf{u}_j|^2}].$$

Now, the problem is reduced to finding an $\mathbf{R}_\theta$ that minimizes the function δ. Since a rotation in the Euclidean space can be represented by a matrix, if we map 2-D Euclidean vectors into the complex system, the rotation can be simply represented by a factor $e^{i\theta}$. Through this complex mapping, the function δ becomes a scale function of θ. The rotation angle, which minimizes the function, is the negative of the polar angle of $\sum_{j=1}^{m} u_j\bar{v}_j$, where u_j and v_j are complex numbers corresponding to vectors $\mathbf{u}_j$ and $\mathbf{v}_j$. We obtain the minimization δ^* of δ

$$\delta^* = \sum_{j=1}^{m}|\mathbf{v}_j|^2 - m|\frac{1}{m}\sum_{j=1}^{m}\mathbf{v}_j|^2 - \frac{|\sum_{j=1}^{m}\mathbf{u}_j\cdot\mathbf{v}_j|^2 + |\sum_{j=1}^{m}(\mathbf{u}_j\times\mathbf{v}_j)|^2}{\sum_{j=1}^{m}|\mathbf{u}_j|^2}.$$

and

$$s = \frac{(|\sum_{j=1}^{m}\mathbf{u}_j\cdot\mathbf{v}_j|^2 + |\sum_{j=1}^{m}(\mathbf{u}_j\times\mathbf{v}_j)|^2)^{1/2}}{\sum_{j=1}^{m}|\mathbf{u}_j|^2}, \quad \mathbf{a} = \mathbf{a}_u - \mathbf{R}_\theta\mathbf{a}'.$$

The inverse value of δ^*/m can be used for goodness of match. As shown in Figure 6.3, we assume that the sensed wing will typically be a subset of the model wing and hence the mapping is "into" and not "onto". The following transformation computation need to applied to each segment of length m of L, so that it will result in 2(n-m+1) transformations computed from the pair of l and L. The constant 2 here is due to the need to match in two directions. Note that the true transformation is included in this set of computed transformations but that there may be several viable candidates because one pair of wings will not be enough to determine a unique transformation. There are usually several consistent wing pairs available. Having computed candidate transformations for all wing pairs, a "best" transformation can be determined through a clustering procedure.

APPENDIX G

Reconstructing Line Drawings from Wing Representations

In this appendix, we consider the adequacy of wing representation for polyhedral scenes. To this end, we demonstrate that if a wing representation has been derived from the image of a polyhedral scene, the spatial structure of the scene could be recovered from the wing representation. In other words, given the wing representation of a scene containing polyhedral objects, we can reconstruct all visible parts of objects based on the given wing representation - that is the coordinates of all visible vertices, the equations of all visible faces, and the completely labeled line drawing of the scene.

As already pointed out by many line-drawing researchers [Huf71, Clo71, Mac73, Kan80, Bar81, Dra81, Sha79, Fuk83, Sug86, Mal89], without additional information, analysis of line drawings, based on categories of vertices and edges [Huf71, Clo71], using relaxation of consistent constraints [Wal75], can usually lead to a number of coherently labeled line drawings. Geometrical properties such as range values of vertices, edge length, angle between faces, size of face and surface normal, are commonly used as additional information by researchers to reduce ambiguity of interpretations.

Here our purpose is not to interpret the 3-D meaning of a 2-D line drawing, but rather to construct the 2-D line drawing and its 3-D interpretation from appropriate fragments of the 3-D scene. It is fairly natural for us to start with the surface equations before reconstructing line drawings. This is because the surface information available in object wings is enough to compute equations of polyhedral object surfaces which are part of the wings. In the following subsections, we study a theoretical approach to interpreting polyhedral scenes based on given wing representations.

G.1 Assumptions and the Object Domain

Assume that A_1) the object domain contains polygons (2-D) and polyhedra (3-D), A_2) scenes can be composed of multiple objects which may occlude one another, A_3) the entire outer boundary of a scene is in the field of view, A_4) the wing "sensor" produces for each visible edge (in some arbitrary but fixed viewpoint) at least a wing segment associated with the edge, A_5) there is no spurious wing (noise) in wing representations derived from scene images, A_6) the range images from which wing representations are derived are available, and A_7) measurements and computations are infinitely accurate. Clearly, among these assumptions, 4, 5 and 7 are unrealistic but this is a theoretical study. There are interesting practical applications which we will leave to future study.

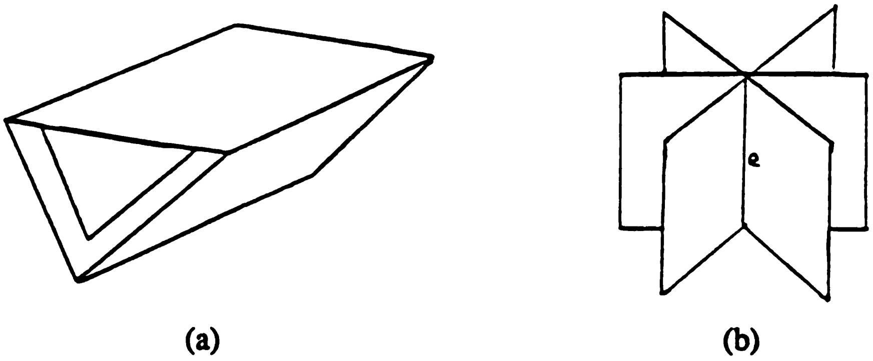


Figure G.1. Example objects are not permissible in our domain: (a) this object is composed of a polygon (2-D) and a polyhedron (3-D), (b) this object is composed of a set of intersecting polygons.

Basically, the approach is based on *faces*, which may arise from either 2-D or 3-D objects. Not all scenes and not all views can be handled. Although the object domain includes both 2-D and 3-D objects, an object can not be composed of both 2-D and 3-D components; in other words, they must be either polygons or polyhedra. Composed objects will not be permissible; an example of such an object, which is made of both polygon and a polyhedron, is shown in Figure G.1(a). Planar panels [Sug86], Origami objects [Kan80], and solid polyhedra are some classes of objects that can be handled by the reconstruction scheme given below. However, objects formed by intersecting a number of polygons, like the one shown in Figure G.1(b), are not considered in this study. Objects may have polygonal holes. Some examples of objects which can be handled are shown in Figure G.2.

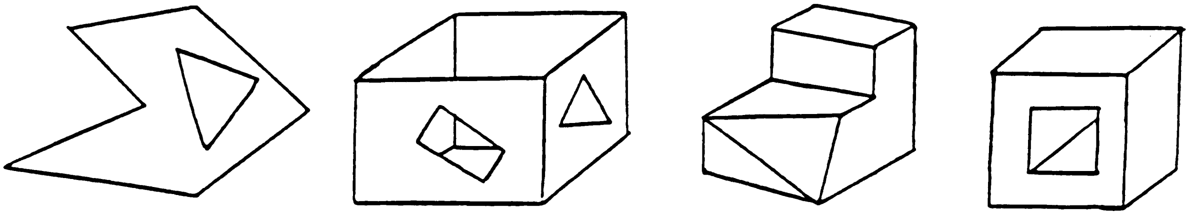


Figure G.2. Examples contained in the object domain: a planar panel with a triangular hole, an Origami object with two holes, a polyhedron with a vertex formed by four faces, and a polyhedron with a hole.

In this paragraph, we will impose three more assumptions on our object domain. A vertex of a polyhedron can be shared by a number of faces and edges (see vertex *v* in Figure G.3), while A_8) a vertex of a polygon should be shared by exactly two noncollinear edges. A_9) Each edge connects exactly two vertices for both polygon and polyhedron. A_{10}) An edge of a polyhedron is formed by two noncoplanar polygons. A face of a polyhedron is a polygon. A polygon is a connected planar area which may have inner polygonal holes. A hole of a face must be within the interior of the face. The interior of a face is an open set bounded by the edges of the face but doesn't include edge points. Thus, holes will not touch boundaries of faces. Refer to the example in Figure G.4, in which area *a* is a hole of face *F*, whereas areas *b*, *c*, *d* and *e* according to our definition are not holes because these areas touch the boundary of *F* by either vertex or edge.

An object face (or polygon) can be represented by a set of simple circuit(s) of edges in a line drawing. In graph terms, a *circuit* is a path whose start and end vertices are the same. A circuit is called *simple* if no vertex, other than the start-end vertex, appears more than once, and the start-end vertex does not appear elsewhere in the circuit [Eve79]. Note

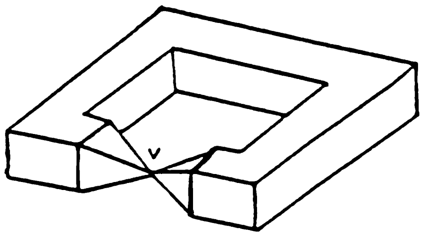


Figure G.3. A vertex of polyhedron can be shared by a number of faces.

that a hole itself is also a simple circuit of edges.

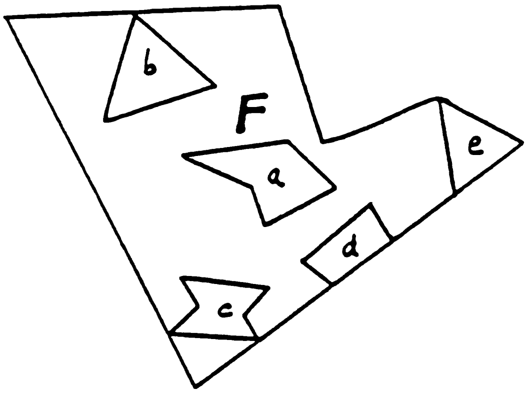


Figure G.4. Area *a* is a hole of face *F*, whereas areas *b*, *c*, *d* and *e* are not holes of *F* because these areas touch the boundary of *F* (violates A_8).

We will consider images which are taken under general viewpoints for 2-D objects (polygons); however, no restriction on viewpoints is imposed for 3-D objects (polyhedra). Thus, a polygon (2-D) cannot project into a straight line in the image plane, whereas a polyhedron (3-D) can form by accident a polygon in the image plane. In short, we demand that for each visible edge at least one face that forms it is visible. For simplicity, if an image is taken from a jumble of objects (including both 2-D and 3-D objects), the image is assumed to be taken from a general viewpoint for all objects. We also assume that there is no accidental arrangement in a jumble of objects; in other words, we only consider general scenes. Accidental arrangements include such situations as touching vertices, touching edges and touching faces, which would violate the vertex-edge-face restrictions A_8, A_9 and A_{10} stated above. An example of accidental alignment is shown in Figure G.5, where two polyhedral objects have their top faces coplanar and have their edges touch each other.

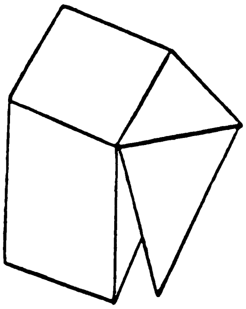
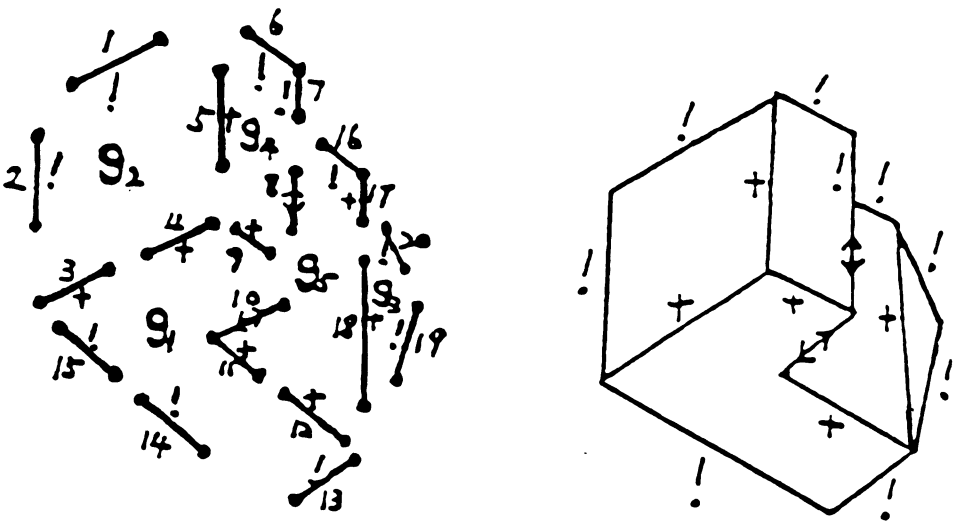


Figure G.5. An accidental arrangement - two polyhedral objects have their top faces coplanar and meanwhile have their edges touch each other (violates A_{10}).

Suppose the background of a scene is a large plane. Any line segment in a wing representation must thus be shared by exactly two distinct regions whose corresponding spatial faces may either separate (jump edges) or connect (crease edges) each other. Therefore, to recognize whether a line segment present in a wing representation map is a real object edge or just a virtual line, we simply inspect points (in both the original range image and the wing representation) on both sides of the line segment to see whether they are located on the same plane.

A wing representation of a polyhedron depicted in Figure G.6(a) purposely made to be imperfect, is an example of input data to a reconstruction procedure to be addressed in Section G.2. In this figure, the surface types of wings are not shown because they are all planar faces, and the small dots indicate the boundaries of wings. Figure G.6(b) shows the labeled line drawing reconstructed from the wing representation, which is expected to be the output of the reconstruction procedure. Here, we follow the convention of line-drawing researchers [Sug86] that a scene is successfully interpreted when coordinates of vertices, face equations, and the labeled line drawing of the scene, or equivalently the spatial structure of scene are reconstructed. Note that line-drawing people [Huf71, Clo71, Mal87] have used {->, +, -} to represent occluding edge, convex and concave edges



(a) wing representation (b) labeled line drawing

Figure G.6. (a) A wing representation of a scene containing only one polyhedral object; (b) the reconstructed labeled line drawing of the object. This is an example of input and output data of the proposed reconstruction procedure.

respectively; however, our wing representation currently uses {!, <->, +, -} to denote silhouette, jump, convex and concave edges. In a polyhedral world, the occluding edge (->) in line drawings actually denotes both silhouette (!) and jump edge (<->) in wing representation.

G.2 Reconstruction Rules

The basic idea of reconstructing labeled line drawings from wing representations of polyhedral scenes (or simply from "wing maps") is first to recover the line drawings of individual visible object faces: the entire scene present in an image is then reconstructed by integrating the recovered individual faces. During reconstruction, only local information is employed to recover individual faces. Global constraints are not necessary provided that we can take sparse samples from the original range image. It is because of this property that a variety of objects and multiple-object scenes with occlusion can be handled by the proposed reconstruction algorithm. By relying more heavily on the general assumptions, a global search procedure can be formulated which does not need to reference the original range image.

Refer to the example wing map shown in Figure G.6(a). Recall that an object wing is composed of a pair of surface patches separated by a contour segment. In a polyhedral world, an object wing is composed of two planar patches separated by a straight line segment. According to assumption 6 - the range images from which wing maps are derived are available, one can immediately compute equations of planes containing object faces which form the wings. Let a plane $ax + by + cz + d = 0$ be represented as a vector (a,b,c,d). Therefore, the data structure of an object wing i can be represented as

$$W_i = ((x_{i1}, y_{i1}), (x_{i2}, y_{i2}), (a_{i1}, b_{i1}, c_{i1}, d_{i1}), (a_{i2}, b_{i2}, c_{i2}, d_{i2}))$$

where (x_{i1}, y_{i1}) and (x_{i2}, y_{i2}) are two endpoints of the object wing. A wing representation (wing map) of a polyhedral scene is a set of object wings.

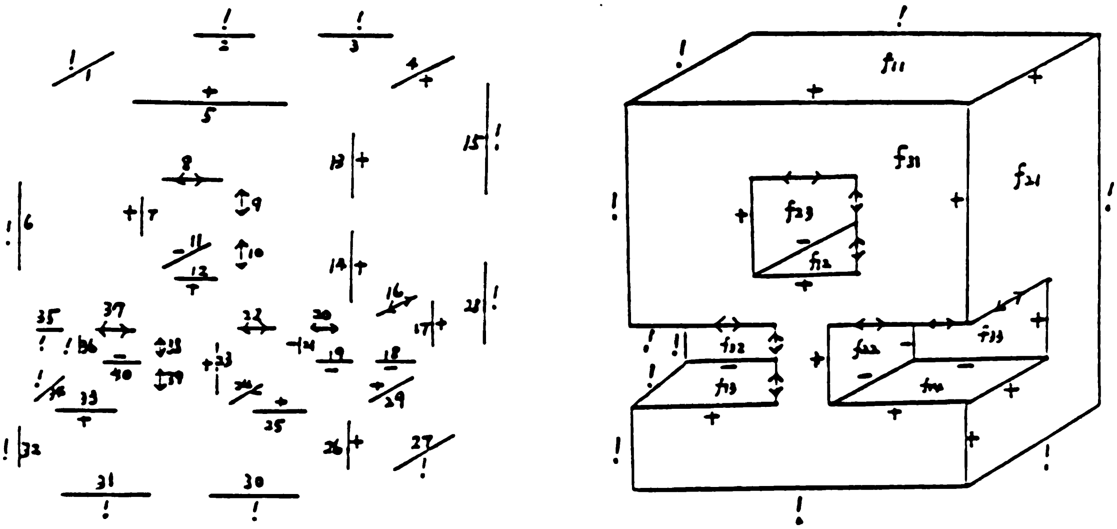
$$\{ W_i \,|\, i = 1, \cdots, n \},$$

where n is the number of object wings. Due to assumption 7 - measurements and computations are infinitely accurate, the computed plane equations are assumed to be perfectly precise. The following rule clusters wing segments based on their plane equations such that wing segments in the same cluster lie on the same plane.

Rule 1: Wings can be clustered into a group, called a wing group, such that wings in the same group lie on the same plane.

Clearly, it is not necessary that wings in a group correspond to the same object face; they may come from several distinct object faces which lie on the same plane. According to assumption 4 - the wing "sensor" produces for each visible object edge at least one wing. Thus, all planes containing object faces will be known as will be all lines containing visible edges. Additionally, due to assumption 5 - there is no spurious wing (noise) in wing maps and there is no superfluous plane derived when computing plane equations from wing segments. In other words, all planes derived are exactly the planes on which object faces lie - no missing and no extra plane.

Consider the example in Figure G.6(a). By following Rule 1, we obtain wing groups: g_1 = {3, 4, 9, 10, 11, 12, 13, 14, 15}, g_2 = {1, 2, 3, 4, 5}, g_3 = {17, 18, 19, 20}, g_4 = {5, 6, 7, 8, 9} and g_5 = {8, 10, 11, 12, 16, 17,18}, Each wing group corresponds to only one object face. However, it is not always so simple. For instance, look at the example shown in Figure G.7(a). After grouping wing segments according to their plane equations, eight wing groups are obtained. They are: h_1 = {1, 2, 3, 4, 5}, h_2 = {10, 11, 12}, h_3 = {33, 34, 39, 40, 18, 19, 24, 25, 29}, h_4 = { 4, 13, 14, 15, 16, 17, 28, 29, 26, 27}, h_5 = {21, 22, 23, 24}, h_6 = {7, 8, 9, 11}, h_7 = {5, 6, 35, 37, 38, 39, 33, 32, 31, 30, 26, 25, 23, 22, 20, 14, 13, 7, 8, 9, 10, 12} and 8_8 = {36, 37, 38, 40, 20, 21, 19, 18, 17, 16}. Among these groups, h_3 has wing segments from faces f_{13} and f_{14}, and h_8 has wing segments from faces f_{32} and f_{33}. There is no trivial way by which a wing group can be separated into subgroups such that each subgroup corresponds to only one object face. The following rules provide the way to reconstruct line drawings of object faces which lie on the single plane from a given wing group.



(a) wing representation (b) labeled line drawing

Figure G.7. A plane may contain several distinct object faces. For example, faces f_{32} and f_{33} lie on the same plane and f_{13} and f_{14} lie on another plane.

Suppose the given group is g_1 of the example in Figure G.6. Figure G.8 shows the picture of g_1. According to assumption 3 - the entire outer boundary of a scene is in the field of view, without loss of generality, we assume that there is a *picture frame* corresponding to the boundary of the wing map as shown in Figure G.8. Then, for each wing segment in g_1, construct a straight line within the picture frame. The broken lines in the figure are the constructed lines and the black line segments in figure are the given wing segments. Note that an object boundary may contain several wing segments, for instance, wings segments (3,4), (11,12) and (14,15). Due to assumption 7, collinear wing segments will lie on the same constructed line. By assumption 4, we know also that there are no extra lines generated and no missing lines; all visible object edges must lie on these constructed lines.

We then compute intersections of lines within the picture frame. This will result in a set of intersections and line segments. Some are real object features (vertices and edges); some are artifact features. Each visible object vertex on the given 3-D plane must correspond to one of these 2-D intersections. The intersections are labeled by letters (a,b,c,..,g) in the figure. Let a *line fragment* be a portion of line, and a *line segment* be a line fragment with two intersections as its endpoints. For convenience, we also define a *semi-line segment* as a line fragment with an intersection at one end and another end located at the picture frame. We now select only those line fragments which form the visible boundaries of the given 3-D object faces in the given plane.

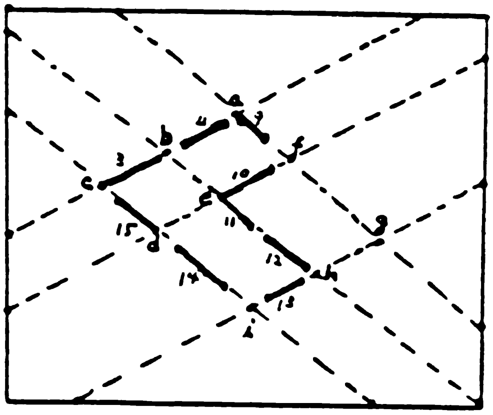
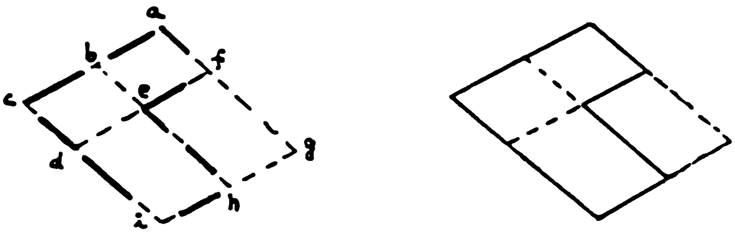


Figure G.8. Straight lines are constructed along wings in g_1.

Recall that each visible object edge should have at least one wing segment associated with the edge (assumption 4). We can remove semi-line segments because such line segments have no wing associated with them and will correspond to no object edges.

Rule 2: Line segments with their endpoints located on the picture frame are removed, or simply, semi-line segments are removed.



(a) (b)

Figure G.9. (a) After removing line segments, the resulting picture of the example in Figure G.8. (b) For each remaining line segment, preserve the line segment if it overlaps at least one wing segment.

After removing semi-line segments, the resulting picture of the example is shown in Figure G.9(a). Then, for each remaining line segment, preserve the line segment if it overlaps at least one wing segment. This is because of assumption 5 - there is no spurious wing (noise) in wing maps. Therefore, a line segment overlapping a wing segment must lie on some object edge.

Rule 3: Preserve each line segment that overlaps a wing.

The recovered object edges (denoted by continuous line segments in the figure) of the example is shown in Figure G.9(b). There are also several undefined line segments (denoted by broken line segments). As we can see, for this particular case, all object edges have been recovered. However, consider another example in Figure G.7(a); after applying Rule 1, 2, and 3 to group h_4, the resulting picture is shown in Figure G.10(a), where line segment $\overline{kl}$, which is a portion of object edge, hasn't been recovered because no wing segment is associated with it. Fortunately, those missing edge segments must lie on some broken line segments as mentioned early in this section. We need to examine broken line segments in order to recover missing edge segments.

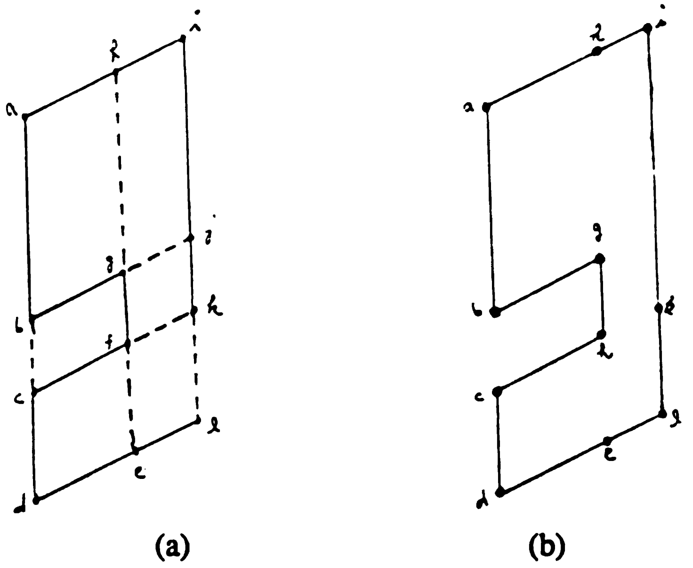


Figure G.10. (a) Consider the example in Figure G.7(a); after applying Rule 1, 2, and 3 to group h_4, 13, line segment $\overline{hl}$ is missing due to no wing segment on it. (b) The missing line segment can be recovered by extending the open ends.

Since any line segment in a line drawing must separate exactly two distinct regions, if points (refer to both the range image and the wing map) on both sides of a candidate line segment are located on the same plane (the plane on which the members of the current wing group lie), the line segment can be deleted because it doesn't correspond to any object edge. Therefore, the line segment between k and l in Figure G.10(a) can be successfully recovered, while segments $\overline{bc}$, $\overline{ef}$, $\overline{gh}$, $\overline{gj}$ and $\overline{fk}$ are identified as non-edge segments.

Rule 4: A line segment is removed from the line drawing unless it has exactly one side with points on the plane of the current wing group.

The result of applying this rule to the data structure of the example in Figure G.10(a) is displayed in Figure G.10(b). Note here that this rule makes Rule 3 redundant because the edge segments recovered by Rule 3 can also be recovered by Rule 4. However, in an implementation, the operation of Rule 3 will run faster than that of Rule 4 in view that Rule 3 only refers to the wing segments in the wing map, whereas Rule 4 needs to refer to the range values of points in the original image and examine whether or not points are located on a given plane. Due to assumption 4, a large portion of object edges can be recovered by Rule 3.

Return to Figure G.10(b). Intersections *e*, *h* and *k* are clearly not vertices of the polygon. As mentioned in the last section, a vertex of a polygon should be shared by exactly two noncollinear edges. However, in some situations, after removing false line segments, some intersections may become isolated. Thus, we have the following rule.

Rule 5: Delete intersections which are not shared by exactly two noncollinear line segments.

After this rule, we are led to the following situation: a) every remaining intersection (vertex) has exactly two noncollinear line segments sharing the intersection (incidence degree 2), b) every line segment (edge) connects exactly two intersections, and c) line segments form a set of simple circuits. These conditions are in fact coincident with the definition of a polygon. Recall that a wing group may contain wing segments coming from several different object faces. The above rules can reconstruct distinct object faces from a given wing group; or equivalently, the rules can recover all disconnected object faces lying on the same plane. An example of group h_8 of the wing map in Figure G.7(a) is shown in Figure G.11, where the results of intermediate steps are presented.

So far, all edges and vertices of faces are recovered. The remaining work is to label the edges with types of {!, <->, +, -}.

Rule 6: A line segment is labeled with the wing type(s) which appear(s) on the line segment.

By this rule, a line segment may have a compound type. A *compound type* means that it is composed of a set of distinct types. For example, a jump wing (<->) and a silhouette wing (!) may appear on the same line segment; in this situation, the line segment is labeled with the compound type (<->!). It is possible that on the same line segment several wing segments with different wing types may appear simultaneously. In practice, only jump wing and silhouette wing can alternatively appear on an image line. Other combinations of wing types are impossible. Figure G.12 shows the result for wing group h_7, where a line segment is labeled (<->!) because there are two different wing

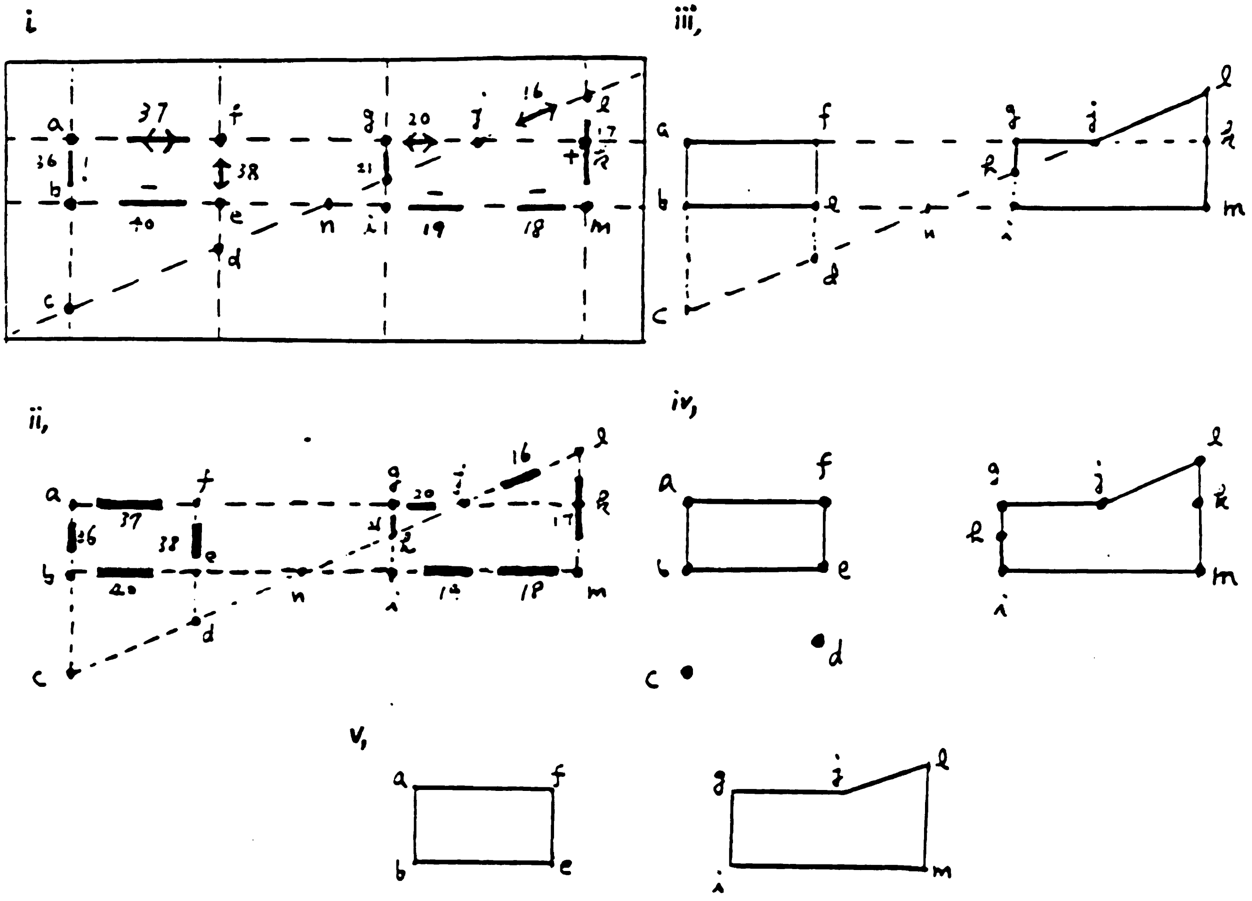


Figure G.11. An example demonstrates that different object faces can be reconstructed from a wing group containing wing segments coming from different object faces.

types appearing on that line segment.

For a line segment with compound type, we don't know where the boundaries of object wings are for the time being. This problem can be easily solved by putting all faces of objects together so that all boundaries of wings can be unveiled. Suppose that applying the above rules to all wing groups, all individual object faces are recovered and their edges are labeled too. To complete the line labeling of a scene, the following rule is invoked after individual faces are integrated, which will need to rely on the wing map and the recovered *T* junctions. The wing map will tell which parts of a line segment should be labeled by which types.

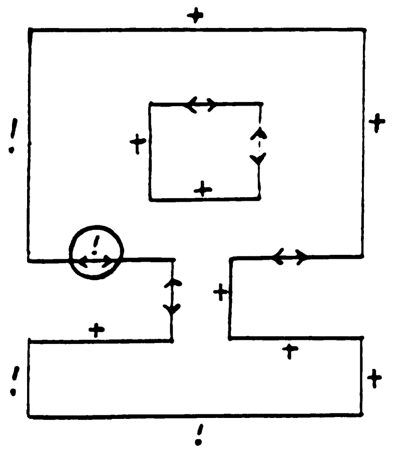


Figure G.12. The result of wing group h_7 contains a line segment with compound type (<->!) which has been highlighted by a circle.

Rule 7: If a line segment possesses a compound type, the components of the compound type are separated and assigned to sub-line segments according to the recovered *T* junctions and the wing map.

The reconstructed labeled line drawing of the example in Figure G.7(a) is shown in Figure G.7(b) in which unique labeling, or equivalently, unique interpretation of the line drawing is obtained in terms of an imperfect wing representation. The reconstructed labeled line drawing of another example in Figure G.6(a) is shown in Figure G.6(b).

G.3 Reconstruction Algorithm

An algorithm characterized by the above set of rules, which reconstructs the labeled line drawing of a polyhedral scene from a wing representation will proceed as follows. The input data include the range image and its associated wing representation of a polyhedral scene. The output result includes a labeled line drawing, the equations of visible faces and the coordinates of visible vertices.

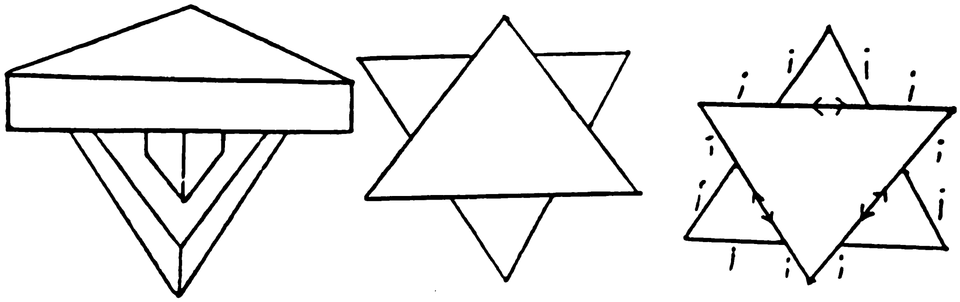
(1) Compute the equations of planes containing object faces using the wing segments in the given wing representation;

(2) Cluster wing segments into wing groups according to their associated plane equations; (Rule 1);

```
(3) For each wing group
    (3.1) generate a line for each wing in the wing group;
    (3.2) compute the intersections among the generated lines;
    (3.3) remove line segments with their endpoints located on the picture frame (Rule 2);
    (3.4) preserve line segments which overlap a wing (Rule 3);
    (3.5) remove any line segment unless points on exactly one side of the line segment are
          located on the given plane (Rule 4);
    (3.6) delete intersections which are not shared by exactly two noncollinear line segments (Rule 5);
    (3.7) record the 3-D coordinates corresponding to the remaining intersections by intersecting
          line equations;
    (3.8) label line segments with the wing types which appear on the line segments (Rule 6);
(4) Integrate reconstructed line drawings of individual object faces in order to discover T junctions;
(5) If a line segment in the integrated line drawing possesses a compound type, the components of
    the compound type are separated and are assigned to distinct parts of the edge according to
    the discovered T junctions and the original wing map (Rule 7).
(6) Output the resulting line drawing with edge labels, the plane equations corresponding to object
    faces, and the coordinates of intersections corresponding to 3-D vertices and 2-D T junctions.
```

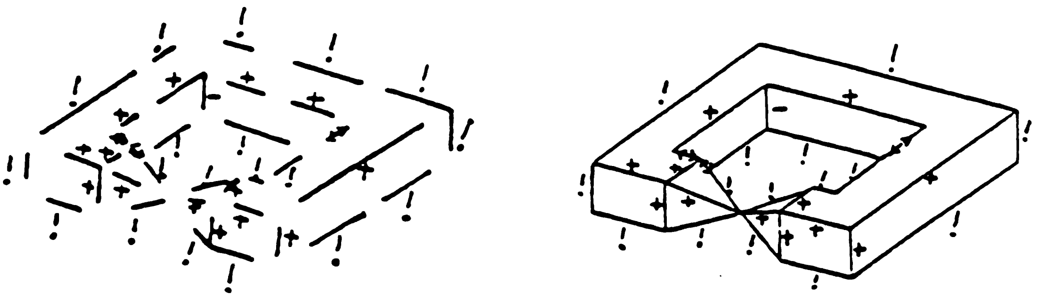
G.4 Further Examples

Since we haven't imposed any restrictions on viewpoints for polyhedra, we may by accident have the situation as shown in Figure G.13(b) for the polyhedron displayed in Figure G.13(a). The above procedure can deal with this accidental case. In addition, objects with holes and multi-face vertices are also permissible. An example of such an object is shown in Figure G.14, which has been taken from Sugihara's book [Sug86]. A few objects from Origami world [Kan80] have been used to study the rules and the procedure (Figure G.15). We claim that the proposed rules and the procedure can uniquely reconstruct from wing representations the labeled line drawings for a wide variety of polyhedral objects. Due to the 3-D information in the wings, a wide variety of objects and viewpoints can be handled than is typical of past research. In fact, current algorithm can interpret the objects in Figure G.1, even though we haven't allowed them in the domain.



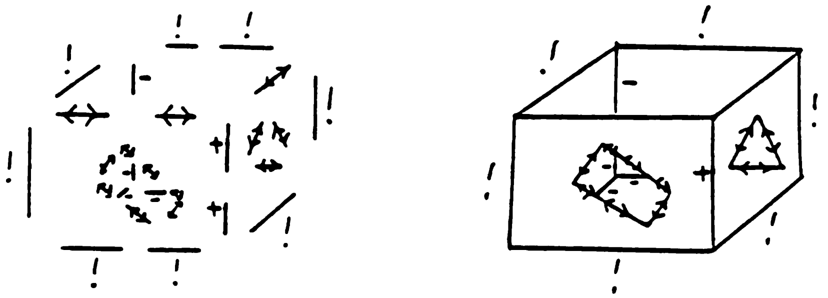
(a) (b) (c)

Figure G.13. (a) A polyhedral object, (b) an accidental view of the object, and (c) the reconstructed line drawing.



(a) wing map (b) labeled line drawing

Figure G.14. An object with a hole and multi-face vertices.



(a) wing map (b) labeled line drawing

Figure G.15. An example of Origami objects.

G.5 Existence, Uniqueness and Correctness of Reconstruction

To justify the validity of the proposed rules for reconstructing line drawings of polyhedral scenes under the assumptions made in Section G.1, we need to cover three aspects of reconstruction: existence, uniqueness and correctness. The existence of reconstruction states that there exists at least one line drawing which can be extracted from the given wing representation. Uniqueness demands that exactly one line drawing is extracted from the given wing representation. As for the correctness of reconstruction, it requires that the extracted line drawing should reflect the spatial structure of the scene. In this section, we will discuss how and why the proposed rules and the algorithm can reconstruct a unique and correct line drawing from a given wing representation.

Recall that the proposed algorithm reconstructs the line drawing of a scene from its wing representation by first recovering all individual visible faces of the scene; then, the entire scene is reconstructed by integrating the recovered faces. During recovery of an object face, the algorithm identifies its edges first and then recovers the face in terms of the identified edges. Note also that this hierarchical reconstruction process - from edges to faces to scene - only uses local information to reconstruct the line drawing of the scene; there is no global constraint involved. In other words, every feature (both edge and face) can be recovered independently. This fact together with assumptions 4 and 5 make the following sequence of proof applicable. That is, if one can show that every object edge present in an image is uniquely recovered, one can prove that every visible object face can be uniquely recovered and in turn prove that the entire scene can be uniquely reconstructed. Consequently, the proof of existence and uniqueness of scene reconstruction can be narrowed down to a proof of the existence and uniqueness of edge recovery. In the reconstruction algorithm, the step 3, which is related to the recovery of edges, includes eight substeps. The major purpose of this step is to reconstruct all object faces lying on a plane. The substeps of step 3 are quite straightforward.

Once object edges has been uniquely recovered from a wing group, if the regions, which are formed by the recovered edges, satisfy the following conditions, these regions possibly correspond to some real world object faces. The conditions include: a) every vertex of the regions has exactly incidence degree 2, b) every edge of the regions connects exactly two vertices, and c) the recovered edges form a unique set of simple circuits which bounds the interiors of the regions. So far, what we can say is that there exists solution(s) for the face reconstruction. We need to show that the face reconstruction is unique. As mentioned early in this section, the uniqueness of face reconstruction will be a natural result because faces are formed by a unique set of edges. In a similar vein, the reconstruction of the entire scene is also unique because it is formed by integrating the unique set of object faces.

G.5 Summary

Many previous line-drawing researchers started with a perfect line drawing and attempted to achieve an interpretation of the line drawing in terms of 3-D spatial structures. Unfortunately, this usually leads to a set of possibilities. Moreover, if the initial line drawings are imperfect, the problem becomes extremely difficult. In order to solve the problems of multiple interpretations and imperfections, assumptions and additional information were employed. Although parts of the problems have been successfully solved by introducing extra knowledge, the overall schemes are not efficient. The reason is that most researchers have separated their schemes into two major modules: reconstruction module and interpretation module. Additional information is only used in the interpretation module to reduce ambiguity rather than be used in the reconstruction module. In fact, available information should be used everywhere in the whole scheme as long as the information is helpful at those places.

In this research, we used range values as additional information in both reconstruction and interpretation modules. This not only greatly simplifies the design of procedures but also increases the efficiency of processing. In addition, instead of following the conventional sequence, we separate the modules into several stages and rearrange the stages in a hierarchical manner. For instance, in our algorithm individual faces are first reconstructed and interpreted, and then the entire scene is considered. One apparent advantage of this hierarchical rearrangement is that it increases the flexibility of the algorithm in dealing with a wide variety of object classes.

Another important aspect of this research is the use of wing representation as input data to the reconstruction algorithm. In contrast to perfect line drawings, wing representations are assumed to be imperfect. On the other hand, wing representations contain the information of range values and thus surface normals, which is not available in line drawings. In practice, wing representations are more realistic than line drawings. The additional information not only considerably compensates for certain deficiency of wing representation in interpreting scenes but also leads the reconstruction procedure to a unique result. The judgement of correctness of a labeled line drawing is also greatly simplified due to this additional information.

It is our conjecture that after the wings are extracted, the line drawing can be constructed without further access to the underlying range image. The proposed algorithm applies the first three rules used previously. It then departs into a state-space search for the polygonal circuits that form the faces in the given plane. At each step of the search one line segment is added to the line drawing and all constraints checked. It is easy to see that the search is finite although exponential. The key point to prove is that there is a unique set of circuits satisfying all the constraints that is the goal state of the search. While such an algorithm has less practical value than the one detailed here, it is of theoretical interest and will be pursued in future work.

Currently, we only consider polyhedral scenes. If a scene contains only one object, the proposed algorithm can handle more general objects than those defined in the current object domain. We need to study how to extend this work to include scenes with curved objects.

List of References

[Asa86] Asada,H., and Brady,M., *"The Curvature Primal Sketch"*, IEEE PAMI, Vol.8, No.1, pp2-14, 1986.

[Aya86] Ayache,N., and Faugeras,O.D., *"HYPER: A New Approach for the Recognition and Positioning of Two-Dimensional Objects"*, IEEE PAMI, Vol.8, No.1, pp44-54, 1986.

[Baj87] Bajcsy,R., and Solina,F., *"Three Dimensional Shape Representation Revisited"*, 1st ICCV, London, England, pp231-240, 1987.

[Bal83] Ballard,D.H., and Sabbah,D., *"Viewer Independent Shape Recognition"*, IEEE PAMI, Vol.5, No.2, pp653-659, Mar., 1983.

[Bar84] Barr,A.H., "Global and Local Deformations of Solid Primitives", Computer Graphics 18(3), pp21-30, 1984.

[Bar81] Barrow,H.G., and Tenenbaum,J.M., *"Interpreting Line Drawings as Three-Dimensional Surfaces"*, Artificial Intelligence, 17, pp75-116, 1981.

[Bas88] Basri,R., and Ullman,S., *"The Alignment of Objects with Smooth Surfaces"*, Proc. 2nd ICCV, Tampa, Florida, pp282-288, Dec. 1988.

[Bes85] Besl,P.J., and Jain,R.C., *"Three-Dimensional Object Recognition"*, Computing Survey, Vol.17, No.1, pp75-145, 1985.

[Bes86] Besl,P.J., and Jain,R.C., *"Invariant Surface Characteristics for 3D Object Recognition in Range Images"*, CVGIP 33, pp33-80, 1986.

[Bes88] Besl,P.J., and Jain,R.C., *"Segmentation through Variable-Order Surface Fitting"*, IEEE PAMI, Vol.10, No.2, pp167-192, 1988.

[Bha87] Bhanu,B., Ho,C.C., and Henderson,T.C., *"3D Model Building for Computer Vision"*, Pattern Recognition Letters, pp349-356, May 1987.

[Bie87] Biederman,I, *"Recognition-by-Components: A Theory of Human Image Understanding"*, Psychological Review, Vol.94, No.2, pp115-147, 1987.

[Bin71] Binford,T.O., *"Visual Perception by Computer"*, Proc. IEEE Systems and Control, Miami, Dec. 1971.

[Bin81] Binford,T.O., *"Inferring Surfaces from Images"*, Artificial Intelligence 17, pp205-244, 1982.

[Bol82] Bolles,R.C., and Cain,R.A., *"Recognizing and Locating Partially Visible Objects: the Local-Feature-Focus Method"*, Int. J. Robotics Res. 1, 3, pp57-82, Fall, 1982.

[Bol83] Bolles,R.C., Horaud,P, and Hannah,M.J., *"3DPO: A Three-Dimensional Part Orientation System"*, Proc. 8th Int. Joint Conf. on Artificial Intelligence. Karlsruhe, West Germany, pp1116-1120, Aug. 1983.

[Bow89] Bowyer,K., Eggert,D., Stewman,J., and Stark L., *"Developing the Aspect Graph Representation for Use in Image Understanding"*, DARPA, IUW, Palo Alto, pp23-26, May, 1989.

[Boy88] Boyer,K.L., and Kak,A.C., *"Structural Stereopsis for 3-D Vision"*, IEEE PAMI, Vol.10, No.2, pp144-166, 1988.

[Bra85] Brady,M., Ponce,J., Yuille,A. and Asada,H., *"Describing Surfaces"*, CVGIP 32, pp1-28, 1985.

[Bro84] Brooks,R.A., *"Model-Based Compute Vision"*, UMI Research Press, Ann Arbor, Michigan, 1984.

[Cae82] Caelli,T., *Visual Perception Theory and Practice*, Pergamon Press, New York, pp93, 1982.

[Cal88] Califano,A., *"Feature Recognition using Correlated Information Contained in Multiple Neighborhoods"*, Exploratory Computer Vision Group, IBM Thomson J. Watson Research Center, Yorktown Heights, New York, 1988.

[Che85] Chen,S.W., and Stockman,G., *"Experiments in 3-D Data Acquisition"*, PRIP Lab. TR. MCS-8304011, Computer Science Dept., Michigan State Univ., East Lansing, Michigan, 1985.

[Che85] Chen,S.W., and Stockman,G., *"Constructing Constraint Tables for Model-Based Recognition and Localization"*, MSU-ENGR-85-34, Computer Science Dept., Michigan State Univ., East Lansing, Michigan, 1985.

[Che87] Chen,S.W., Stockman,G., and Shrikhande,N., *"Computing a Pose Hypothesis from a Small Set of 3-D Object Features"*, MSU-ENGR-87-001, Computer Science Dept., Michigan State Univ., East Lansing, Michigan, 1987.

[Chi86] Chin,R.T., and Dyer,C.R., *"Model-Based Recognition in Robot Vision"*, ACM Computing Surveys, 18(1), pp67-108, 1986.

[Clo71] Clowes,M.B., *"On Seeing Things"*, Artificial Intelligence 2, pp79-116, 1971.

[Coo75] Cooper,L.A., *"Mental Rotation of Random Two-Dimensional Shapes", Cognitive Psychology, Vol.7, No.1, pp20-43, 1975.*

[Coo84] Cooper,L.A., and Shepard,R.N., *"Turning Something Over in the Mind"*, in [*The Mind's Eye*, Freeman and Company, New York, 1986], pp102-108, 1984.

[Dra81] *"The Use of Gradient and Dual Space in Line-Drawing Interpretation"*, Artificial Intelligence, 17, pp461-508, 1981.

[Dub76] Dubes,R., and Jain,A.K., *"Clustering Techniques: The User's Dilemma"*, Pattern Recognition, Vol.8, No.4, pp247-260, Oct. 1976.

[Dud77] Dudani,S.A., Breeding,K.J., and McGhee,R.B., *"Aircraft Identification by Moment Invariants"*, IEEE Computer, Vol.c26, No.1, pp39-45, Jan. 1977.

[Eve79] Even,S., *Graph Algorithms*, Technion, Haifa, Isreal, 1979.

[Fan87] Fan,T.J., Medioni,G., and Nevatia,R., *"Segmented Descriptions of 3-D Surfaces"*, IEEE J. Robotics and Automation, Vol.RA-3, No.6, Dec. 1987.

[Far87] Farin,G.E., *Geometric Modeling: Algorithm and New Trends*, The Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, 1987.

[Fau86] Faugeras,O.D., and Hebert,M., *"The Representation, Recognition and Location of 3D Objects"*, Intl. J. Robotics Research, 5(3), pp27-52, 1986.

[Fau79] Faux,L.D., and Pratt,M.J., *Computational Geometry for Design and Manufacture*, Ellis Horwood Ltd, pp71-73, 1979.

[Fek83] Fekete,G., *"Generating Silhouettes of Polyhedra"*, NB83-NADA-4036, Computer Science Dept., Univ. of Maryland, 1983.

[Fel81] Feldman,J.A., and Ballard,D.H., *"Computing with Connections"*, TR72, Computer Science Department, University of Rocheter, 1981.

[Fis81] Fischler,M.A., and Bolles,R.C., *"Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography"*, Graphics and Image Processing, Editor Foley,J.D., pp726-740, 1981.

[Fol82] Foley,J.D., and Dam,A.V., *Fundamentals of Interactive Computer Graphics*, Addison-Wesley Publishing Company, Massachusetts, 1982.

[Fuk83] Fukui,Y., Hirotani,T., Ohira,T., and Kishi,Y., *"An Input Method for Solid Models by Drawing, Preprint"*, Industrial Products Research Institute, Ministry of International Trade and Industry of Japan, 1983.

[Gas83] Gaston,P.C., and Lozano-Pérez,T., *"Tactile Recognition and Localization Using Object Models"*, AI, Memo-705, MIT, 1983.

[Gib50] Gibson,J.J., *The Perception of the Visual World*, Boston, Houghton Mifflin Comp., Boston, 1950.

[Gig88] Gigus,Z., and Malik,J., *"Computing the Aspect Graph for Line Drawings of Polyhedral Objects"*, CVPR, Ann Arbor, Michigan, pp654-661, June, 1988.

[Gig88] Gigus,Z., Canny,J., and Seisel,R., *"Efficient Computing and Representing Aspect Graphs of Polyhedral Objects"*, ICCV 2nd, Tampa, Florida, pp30-39, Dec., 1988.,

[Gil80] Gillam,B., *"Geometrical Illusions"*, in [*The Mind's Eye*, Freeman and Company, New York, 1986], pp87-94, 1980.

[Goa83] Goad,C., *"Special Purpose Automatic Programming for 3D Model-Based Vision"*, Proc. ARPA Image Understanding Workshop, Arlington, VA, 1983.

[Gri85] Grimson,W.E.L., *"Sensing Strategies for Disambiguating among Multiple Objects in Known Poses"*, AI, Memo-855, MIT, Aug. 1985.

[Gri88] Grimson,W.E.L., *"The Combinatorics of Object Recognition in Cluttered Environments using Constrained Search"*, 2nd ICCV, Tampa, Florida, Dec. 1988.

[Gri86] Grimson,W.E.L., *From Images to Surfaces,* The MIT Press, Cambridge, Massachusetts and London, England, 1986.

[Gri87] Grimson,W.E.L., and Lozano-Pérez,T., *"Localizing Overlapping Parts by Searching the Interpretation Tree",* IEEE PAMI Vol.9, No.4, pp469-482, 1987.

[Gri88] Grimson,W.E.L., and Huttenlocher,D.P., *"On the Sensitivity of the Hough Transform for Object Recognition",* AI, Memo-1044, MIT, 1988.

[Gun87] Gunnarsson,K.T., and Prinz,F.B., *"CAD Model-Based Localization of Parts in Manufacturing",* IEEE Computer Society Magazine, pp66-74, Aug. 1987.

[Hen85] Henderson,T.C., *"Efficient 3D Object Representations for Industrial Vision System",* IEEE PAMI, Vol.5, pp609-617, 1983.

[Hof84] Hoffman,D.D., and Richards,W., *"Parts of recognition",* Cognition, 18, pp65-96, 1984.

[Hof87] Hoffman,R., and Jain,A.K., *"Segmentation and Classification of Range Images",* IEEE PAMI, Vol.9, No.5, pp608-620, 1987.

[Hor77] Horridge,G.A., *"The Compound Eye of Insects",* in [*The Mind's Eye,* Freeman and Company, New York, 1986], pp24-35, 1977.

[Hu89] Hu,G., and Stockman,G., *3-D Surface Solution using Structured Light and Constraint Propagation* IEEE PAMI, Vol.11, No.4, pp390-402, 1989.

[Huf71] Huffman,D.A., *"Impossible Objects as Nonsense Sentences",* Machine Intelligence 8 edited by Meltzer,R. and Michie,D., New York, Elsevier, pp295-323, 1871.

[Huf77] Huffman,D.A., *"A Duality Concept for the Analysis of Polyhedral Scenes",* Machine Intelligence 8, Ellis Horwood, England, pp475-492, 1977.

[Hum86] Hummel,R.A., *"Representations Based on Zero-Crossings in Scale Space"*, Proc. IEEE CVPR, Miami Beach, Florida, pp204-209, June, 1986.

[Hut87] Huttenlocher,D.P., and Ullman,S., *"Object Recognition using Alignment"*, 1st ICCV, London, England, pp102-111, 1987.

[Ike88] Ikeuchi,K., and Kanade,T., *"Towards Automatic Generation of Object Recognition Programs"*, CMU-CS-88-138, May 1988.

[Jai88] Jain,A.K., and Hoffman,R. *"Evidence-Based Recognition of 3-D Objects"*, IEEE PAMI, Vol.10, No.6, pp783-802, Nov., 1988.

[Kan80] Kanade,T., *"A Theory of Origami World"*, Aritificial Intelligence, 13, pp279-311, 1980.

[Kan76] Kanizsa,G., *" Subjective Contours"*, in [*The Mind's Eye*, Freeman and Company, New York, 1986], pp82-86, 1976.

[Kno86] Knoll,T.F., and Jain,R.C., *"Recognizing Partially Visible Objects using Feature Indexed Hypotheses"*, IEEE Journal of Robotics and Automation, Vol.Ra-2, No.1, pp3-13, 1986.

[Koc87] Koch,M.W., Kashyap,R.L., *"Using Polygons to Recognize and Locate Partially Occluded Objects"*, IEEE, PAMI, Vol.9, No.4, pp483-494, July, 1987.

[Koe76] Koenderink,J.J., and van Doorn,A.J., *"The Singularities of the Visual Mapping"*, Biological Cybernetics, Springer-Verlag, 1976.

[Koe79] Koenderink,J.J., and van Doorn,A.J., *"The Internal Representation of Solid Shape with Respect to Vision"*, Biological Cybernetics, Springer-Verlag, 1979.

[Koe84] Koenderink,J.J., *"What does the occluding contour tell us about solid shape"*, Perception, Vol.13, pp321-330, 1984.

[Koe87] Koenderink,J.J., *"The internal Representation for Solid Shape based on the Topological Properties of the Apparent Contour"*, Chapter 9, *Image Understanding* edited by Richards,W., and Ullman,S., 1987.

[Kor87] Korn,M.R., and Dyer,C.R., *"3-D Multiview Object Representations for Model-Based Object Recognition"*, Pattern Recognition, Vol.20, No.1, pp91-104, 1987.

[Lam88] Lamdan,Y., and Wolfson,H.J., *"Geometric Hashing: A General and Efficient Model-Based Recognition Scheme"*, Proc. 2nd ICCV, Tampa, Florida, pp238-251, Dec. 1988.

[Lau86] Laurendeau,D, and Poussart,D., *"3D Model Building using a Fast Range Finder"*, Proc. IEEE CVPR, Miami Beach, Florida, pp424-426, 1986.

[Lin88] Linnainmaa,S., Harwood,D., and Davis,L.S., *"Pose Determination of a Three-Dimensional Object using Triangle Pairs"*, IEEE, PAMI, Vol.10, No.5, pp634-647, 1988.

[Low85] Lowe,D.G., *"Perceptual Organization and Visual Recognition"*, Kluwer Academic Publishers, 1985.

[Low87] Lowe,D.G., *"Three-Dimensional Object Recognition from Single Two-Dimensional Images"*, Artificial Intelligence 31, pp355-395, 1987.

[Low88] Lowe,D.G., *"Organization of Smooth Image Curves at Multiple Scales"*, Proc. 2nd ICCV, Tampa, Florida, pp558-567, Dec. 1988.

[Mac73] Mackworth,A.K., *"Interpreting Pictures of Polyhedral Scenes"*, Artificial Intelligence, 4, pp121-137, 1973.

[Mal87] Malik,J., *"Interpreting Line Drawings of Curved Objects"*, International Journal of Computer Vision, Vol.1, No.1, pp73-103, 1987.

[Mar82] Marr,D., *Vision*, Freeman and Company, New York, 1982.

[Mar80] Marr,D., and Hildreth,E.C., *"Theory of Edge Detection"*, Proc. R. Soc. Lond. B207, pp187-217, 1980.

[Mea81] Meagher,D.J., *"Geometric Modeling Using Octree Encoding"*, Computer Graphics Image Processing 19, 2, pp129-147, June, 1981.

[Mur87] Murray,D.W., *"Model-Based Recognition Using 3D Shape Alone"*, CVGIP, 40, pp250-266, 1987.

[Nal86] Nalwa,V.S., and Binford T.O., *"On Edge Detection"*, IEEE, PAMI, Vol.8, No.6, pp699-714, 1986.

[Nal88] Nalwa,V.S., *"Line-Drawing Interpretation: A Mathematical Framework"*, International Journal of Computer Vision, Vol.2, No.2, pp103-124, 1988.

[Ohm88] Ohmura,K., Tomono,A., and Kobayashi,Y., *"Method of Detecting Face Direction using Image Processing for Human Interface"*, SPIE, Vol.1001, Visual Communications and Image Processing, pp625-632, 1988.

[Par87] Parvin,B., and Medioni,G., *"Adaptive Multiscale Feature Extraction from Range Data"*, Proc. IEEE Computer Society, Workshop on Computer Vision, Miami Beach, Florida, pp23-28, Nov. 1987.

[Pau81] Paul,R.P., *Robot Manipulators: Mathematics, Programming and Control*, The MIT Press, Cambridge, Massachusetts, pp25-28, 1981.

[Pen87] Pentland,A.P., *"Perceptual Organization and the Representation of Natural Form"*, Readings in Computer Vision, Morgan Kaufmann Pub. California, pp680-699, 1987.

[Pla87] Plantinga,W.H., and Dyer,C.R., *"Visibility, Occlusion, and the Aspect Graph"*, TR, No.736, Dept. of Computer Science, Univ. of Wisconsin, Madison, Dec, 1987.

[Pot83] Potmesil,M., *"Generating Models of Solid Objects by Matching 3D Surface Segments"*, Proc. 8th Int'l. Joint Conf. Artificial Intel., Karlsruhe, West Germany, Aug. 1983.

[Pre85] Preparata,F.P., and Shamos,M.I., *"Computational Geometry"*, Springer-Verlag, New York, 1985.

[Pri85] Price,K.E., *"Relaxation Matching Techniques - A Comparison"*, IEEE PAMI, Vol.7, No.5, pp617-623, 1985.

[Req80] Requicha,A.A.G., *"Representation for Solids: Theory, Method, and Systems"*, ACM Computer Survey, 12, 4, pp437-464, 1980.

[Rob66] Roberts,L.G., *"Machine Perception of Three Dimensional Objects"*, Tippett,J.T., *et al*, Optical and Electro-Optical Information Processing, MIT Press, Cambridge MA., 1966.

[Sab82] Sabbah,D., *"A Connectionist Approach to Visual Recognition"*, Ph.D. Dissertation, Computer Science Department, University of Rochester, 1982.

[Sch87] Schwartz,J.T., and Sharir,M., *"Identification of Partially Obscured Objects in Two-Dimensions by Matching Noisy Characteristic Curves"*, Intl. J. Robotics Research, Vol.6, No.2, pp29-44, 1987.

[Sek85] Sekuler,R., and Blake,R., *Perception*, Alfred A. Knopf, Inc., New York, 1985.

[Sha79] Shapira,R., and Freeman, H., *"The Cyclic Order Property of Vertices as an Aid in Scene Analysis"*, Communication of ACM, Vol.22, pp368-375, 1979.

[Sha84] Shapira,R., and Freeman, H., *"The Use of Objects' Faces in Interpreting Line Drawings"*, IEEE, PAMI, Vol.6, pp789-794, 1984.

[Sha84] Shapiro,L.G., Moriarty,J.D., and Haralick,R.M., *"Matching Three-Dimensional Objects using a Relational Paradigm"*, Pattern Recognition, Vol.17, No.4, pp385-405, 1984.

[She84] Shepard,R.N., *"Ecological Constraints on Internal Representation: Resort Kinematics of Perceiving, Imaging, Thinking and Dreaming"*, Psychological Review, Vol.91, No.4, pp417-447, 1984.

[She71] Shepard,R.N., and Metzler,J., *"Mental Rotation of Three-Dimensional Objects"*, Science, Vol.171, No.3972, pp701-703, 1971.

[Sie77] Siegel,R.K., *"Hallucinations"*, in [*The Mind's Eye*, Freeman and Company, New York, 1986], pp109-116, 1977.

[Sho88] Shoham,D., and Ullman,S., *"Aligning a Model to an Image using Minimal Information"*, Proc. 2nd ICCV, Tampa, Florida, pp259-263, Dec. 1988.

[Sto82] Stockman,G., Kopstein,S., and Benett,S., *"Matching Images to Models for Recognition and Object Detection via Clustering"*, IEEE PAMI, Vol.4, No.3, pp229-241, 1982.

[Sto87] Stockman,G., *"Object Recognition and Localization via Pose Clustering"*, CVGIP, 40, pp361-387, 1987.

[Sto88] Stockman,G., Chen,S.W., Hu,G., and Shrikhande,N., *"Recognition of Rigid Objects using Structured Light"*, IEEE Control System Magazine, pp14-22, June 1988.

[Sug78] Sugihara,K., *"Picture Language for Skeletal Polyhedra"*, CGIP, Vol.8, pp382-405, 1978.

[Sug86] Sugihara,K., *Machine Interpretation of Line Drawing,* The MIT Press, Cambridge, Massachusetts, 1986.

[Tho87] Thompson,D.W., and Mundy,J.L., *"Three-Dimensional Model Matching from an Unconstrained Viewpoint"*, Proc. IEEE on Robotics and Automation, Raleigh, North Carolina, pp208-220, 1987.

[Tsa85] Tsai,W.H., and Yu,S.S., *"Attributed String Matching with Merging for Shape Recognition"*, IEEE PAMI, Vol.7, No.4, pp453-462, 1985.

[Tuc86] Tuceryan,M., *"Extraction of Perceptual Structure in Dot Patterns"*, Ph.D. Dissertation, University of Illinois at Urbana-Champaign, 1986.

[Tur85] Turney,J.L., Mudge,T.N., and Volz,R.A., *"Recognizing Partially Occluded Parts"*, IEEE PAMI, Vol.7, No.4, pp410-421, 1985.

[Tur74] Turner,K., *"Computer Perception of Curved Objects using a Television Camera"*, Ph.D. Dissertation, School of Artificial Intelligence, Univ. Edinburgh, 1974.

[Tve84] Tversky,B., and Hemenway,K., *"Objects, Parts and Categories"*, J. of Experimental Psychology: General, 113, pp169-193, 1984.

[Vis85] Vistnes,R., *"Detecting Structure in Random-Dot Patterns"*, Proc. of Workshop on Image Understanding, DARPA, pp350-362, Dec. 1985.

[Wal75] Waltz,D., *"Understanding Line Drawings of Scenes with Shadows"*, The Psychology of Computer Vision, edited by Winston,P.H., McGraw-Hill Book Comp. New York, 1975.

[Wan87] Wang,Y.F., and Aggarwal,J.K., *"On Modeling 3-D Objects using Multiple Sensory Data"*, Proc. IEEE Conf. Robotics and Automation, Raleigh, NC, pp1098-1103, 1987.

[Whi79] Whiteley,W., *"Realizability of Polyhedra"*, Structural Topology, Vol.1, pp46-58, 1979.

[Whi55] Whitney,H., *"Singularities of Mappings of Euclidean Spaces. I: Mappings of the plane into the plane"*, Ann. Math 62, pp374-410, 1955.

[Wol86] Wolfe,J.M., *The Mind's Eye*, Freeman and Company, New York, 1986.

[Won85] Wong,A.,K.,C., and You,M., *"Entropy and Distance of Random Graphs with Application to Structural Pattern Recognition"*, IEEE PAMI, Vol.7, No.5, pp599-609, 1985.

[Yan85] Yang,H.S., and Kak,A.C., *"Determination of the Identity, Position and Orientation of the Topmost Object in a Pile"*, Proc. 3rd Workshop on Computer Vision: Representation and Control, Bellaire, Michigan, pp38-48, 1985.

[Zah71] Zahn,C.T., *"Graph-Theoretical Methods for Detecting and Describing Gestalt Cluster"*, IEEE, Computers, Vol.c-20, No.1, pp68-86, 1971.

[Zuc85] *"Early Orientation Selection:P{Tangent Fields and the Dimensionality of their Support"*, CVGIP, Vol.32, pp74-103, 1985.