CBCV: A CAD-Based Vision System

Thomas C. Henderson, John Evans, Lane Grayston,

Allen Sanderson, Leigh Stoller and Eliot Weitz

UUCS-90-013

Department of Computer Science
University of Utah
Salt Lake City, UT 84112 USA

June 27, 1990

Abstract

The CBCV system has been developed in order to provide the capability of automatically synthesizing executable vision modules for various functions like object recognition, pose determinaion, quality inspection, etc. A wide range of tools exist for both 2D and 3D vision, including not only software capabilities for various vision algorithms, but also a high-level frame-based system for describing knowledge about applications and the techniques for solving particular problems[1].

[1]This work was supported in part by the National Science Foundation under Grant INT-8909596. This work was also supported in part by DARPA (N00014-88-K-0689). All opinions, findings, conclusions or recommendations expressed in this document are those of the author and do not necessarily reflect the views of the sponsoring agencies.

1 Introduction

Computer Aided Design, or CAD as it is better known, has many advantages to offer in the design and development of manufactured items. Computer vision, on the other hand, has yet to make major inroads into the manufacturing domain. We believe that a closer tie between geometric models and computer vision will lead to greater application of computer vision techniques in industry and to a more efficient and effective manufacturing process.

In this report, we examine the role of Computer Aided Geometric Design models in providing support for computer vision techniques. In particular, we examine the requirements placed on CAD systems to achieve useful vision functions, and we take a look at the nature of the representation differences betwen CAD and vision, and we describe AI techniques for synthesizing executable computer vision modules based on an analysis of the task requirements, hardware and software available, and on the geometric object under consideration.

We give detailed examples of both CAD and computer vision systems, as well as synthesis techniques for automatically deriving visual inspection, object recognition, and pose determination modules. We use the Alpha_1 CAGD system, developed by Rich Riesenfeld and Elaine Cohen and their colleagues at the University of Utah for all of our design and 3D data manipulation. Their system is a boundary representation, B-spline based modeler, and provides exceptional design and analysis capabilities.

For the computer vision system, we use the IKS (Image Kernel System). This developed out of an early set of vision tools acquired from Bill Havens at the University of British Columbia. The majority of the current IKS system, however, has been developed over the last few years at the University of Utah. The functions range from low-level image to image processing routines to 3D intrinsic characteristic funstions for the analysis of 3D range data.

The knowledge-based component of the system described here has been developed on top of PCLS (Portable Common Lisp Standard) in FROBS (FRames and OBjectS). These systems were developed by Bob Kessler and the PASS (Portable Ai SystemS) group at the University of Utah.

2 Computer Aided Geometric Design

2.1 General Considerations

Computer vision has been an active research area for over 25 years. In the past, emphasis was on low level processing such as intensity and signal processing to perform edge detection. More recently, models of objects and knowledge of the working environment have provided the basis for driving vision systems. This is known as model-based vision. The pursuit of the fully automated assembly environment has fueled interest in model-based computer vision and object manipulation. This involves building a 3-D model of the object, matching the sensed environment with the known world and determining the position and orientation of the recognized objects. The goal is to provide a solution to the problem of visual recognition in a well-known domain.

In the automation environment, recognition schemes and representations have typically been constructed using ad hoc techniques. Although objects used in the assembly process are designed with a CAD system, generally there is no direct link from the CAD system to the robotic workcell. This means the recognition systems are constructed independently of the CAD model database. What is desired is a systematic approach for both the generation of representations and recognition strategies based on the CAD models. Such a system provides an integrated automation environment. The system is composed of several components: a CAD system, a milling system, a recognition system and a manipulation system. In this paper, the automatic generation of recognition strategies based on the CAGD model is studied. It has also been determined that the use of shape, inherent in CAGD models, can also be used to drive the recognition process. Others have been studying portions of this system. Recent work by Ho has focused on the generation of computer vision models directly from a CAGD model[2,15].

The work described here investigates the use of geometric knowledge in constructing 2D recognition codes. These codes provide a robust mechanism for recognition and localization of two-dimensional objects (occluded as well as non-occluded) in typical manufacturing scenes.

One of the first researchers to study the automatic synthesis of general recognition strategies was Goad[11]. He was concerned with automatic programming for 3-D model based vision. His work generated a recognition scheme for matching edges based on a general sequential matching algorithm. His algorithm proceeded in three steps: (1) predict a feature, (2) observe (match) a feature, and (3) back-project (refine the object hypothesis based on step 2). These three steps form a template which is used by the automatic programming phase. He used a unit sphere to gather loci of view angles (camera positions) which represent orientations of the object. The only features used were straight edges from intensity images and the search trees were generated from a template and ordered by hand rather than automatically. His system didn't consider partial occlusion. However, this was a major contribution since it was one of the first attempts to automate the generation of recognition schemes.

Another influential project was the 3DPO system by Bolles and Horaud[4]. This work is the 3-D generalization of the Local Feature Focus method[3]. Their system annotates a CAD model producing what is called the extended CAD model. From this model, feature analysis is performed to determine unique features from which to base hypotheses. The focus feature in their system is the dihedral arc. When the recognition system finds a dihedral arc, it looks for nearby features which are used to discriminate between model arcs with similar attributes. From these, an object's pose is hypothesized and subsequently verified.

Recently, Ikeuchi has explored the use of interpretation trees for representation of recognition strategies[16]. His system uses the concept of visible faces to generate generic representative views, called aspects. From this set of aspects, an interpretation tree is formed which discriminates among the different aspects. His system uses a variety of object features such as: EGI, face inertia, adjancency information, face shape, and surface characteristics. Most of these features are based on planar faces. A very specific interpretation tree is generated for an object using a set of object specific rules. The rules were selected by hand rather than generated automatically. There doesn't appear to be any algorithmic approach for the application of the rules to discriminate between the aspects. The branching on the tree seems to be a function of the particular aspects chosen rather than being based on the geometric information in the model.

Hansen and Henderson[13] have also proposed a CAD-based 3D recognition and localization scheme based on strategy trees. That system isn't dependent on a certain class of features but rather can be extended to include many classes of features. The system also performs automatic selection of features based on a set of constraints: feature filters. These features are used to form a strategy tree which provides a scheme for hypothesis formation, corroborating evidence gathering and object verification. The flexibility of this approach makes it significantly different from related work.

Our main goal of is the automatic synthesis of recognition system specifications for CAD-based 2D and 3D computer vision[12,13,22]. Given a CAD model of an object, a specific, tailor-made system to recognize and locate the object is synthesized.

To attain this goal, the following problems have been studied:

1. **Geometric Knowledge Representation**: The use of geometric data is central to a strong recognition paradigm. Weak methods can only be avoided when better information is available. The Alpha_1 B-spline model allows the modeling of freeform sculptured surfaces. To obtain the geometric features of interest for 3-D recognition, techniques for the transformation to a computer vision representation have been developed.

2. **Automatic Feature Selection**: The part to be recognized or manipulated must be examined for significant features which can be reliably detected and which constrain the object's pose as much as possible. Moreover, such a set of features must cover the object from any possible viewing angle. In solving the feature selection problem, a technique is available for synthesizing recognition systems. This produces much more efficient, robust, reliable and comprehensible systems.

3. **Recognition System Synthesis**: Once a robust, complete and consistent set of features has been selected, a recognition strategy is automatically generated. Such a strategy takes into account the strongest features and how their presence in a scene constrains the remaining search. The features and the corresponding detection algorithms are welded, as optimally as possible, into a search process for object identification and pose determination. The automatic synthesis of search strategies is a great step forward toward the goal of automated manufacturing. Generation of strategies is constrained, not only by the feature selection process but, by the actual task to be accomplished. Thus, strategies for a specific task might not be as strong when applied to a different task; strategies are task specific.

The remainder of this paper explains how these three components can be exploited to automate the process of selecting proper features and recognition schemes for specific goals. Algorithms are described which were developed for feature selection and which give supporting evidence for their formulation.

Computer vision utilizes object models in a different manner than computer graphics or CAGD. In CAGD, the models must contain information about the 3-D object for rendering, performing finite element analysis, milling and other processes. Computer vision is concerned with recognition of the objects from sensory data. CAGD models must contain information for the local design operations such as what shape to extrude or what is the profile curve for a sweep operation. Features used in construction of models are implicitly rather than explicitly used in the CAGD representation. For example, a dihedral edge formed from two adjoining surfaces isn't modeled as an edge per se but as two surfaces with adjacency information.

Constructive solid geometry (CSG) and boundary representations are the best understood and currently most important representation schemes in computer aided design. Present day 3-D wireframe models used in CAD and model-based vision have many deficiencies including ambiguity – it is easy to build a wireframe model that can be surfaced in several ways[19]. In CSG, the basic idea is that complicated solids can be represented as various ordered "additions" and "subtractions" of simpler solids by means of modified versions of Boolean set operators-union, difference and intersection[18]. For inherent boundary representations a number of different approaches are used. These include Coons patches, bicubic surface patches, Bezier methods and B-splines[1].

Most Geometrical Modeling Systems (GMS) use a limited class of primitives such as rectilinear blocks and conic surfaces (cylinders, cones and spheres). Although these suffice to design a large number of conventional unsculptured parts, a GMS which includes sculptured solids is highly desirable. Also since the sculptured design is surface oriented, it is easier to incorporate it in a boundary based system. In general, boundary modelers tend to support stepwise construction of the models more easily than CSG modelers but require greater data storage. CSG modelers are inadequate for modeling sculptured parts: they have no capability at all for constructing and using sculptured surfaces as part of the boundary of the solid model. Some advantages of boundary representation are: there are many known surface models available from which to choose[1]; the mathematics of surface representation is well developed and complex shapes can often be represented with a single primitive[8,21]; and it results in an intuitive model. A minor disadvantage is that it may be difficult to ensure the validity of a boundary representation of a set. On the other hand, CSG representations are not unique in general, since a solid may be constructed in many ways; the final result may not be easily visualized by looking at the primitives. However, the CSG representation is concise, validity is guaranteed and such a representation can be easily converted to a boundary representation. The comparison of CSG and boundary representation methods can be found in[19,20].

Until recently it was not possible to carry out Boolean operations on sculptured surfaces. Work by Thomas[21] attempts to combine the best attributes of CSG and surface-based representation systems by using subdivision techniques developed by Cohen et al.[10]. He uses a uniform boundary representation. The "primitives" are solids bounded by B-spline surfaces. As compared to the other work in solid modeling, his method does not require that the objects being combined have closed boundaries; they must only satisfy a weak completion criterion. Thus this method results in a powerful shape description system which allows the combination of primitives using set operations into arbitrarily-complex objects bounded by curved surfaces and the production of a model which represents such objects. Adjacency information about surface points and the intersection curve between two surfaces as a polyline can be obtained. Although he has used B-spline surfaces, his techniques are applicable to any surface representation scheme[8]. All this work has been incorporated in the Alpha_1 system[9]. (More details about Alpha_1 are presented below.) Thus, the advantages of both CSG and sculptured surface representation can be obtained in the shape representation of objects and the combination of objects via set operations. As a result of these significant advances in CAGD, we decided to use the Alpha_1 system for exploring the computer vision application.

Alpha_1 is an experimental CAGD based solid modeler system incorporating sculptured surfaces[9]. It allows in a single system both high quality computer graphics and freeform surface representation and design. It uses a rational polynomial spline representation of arbitrary degree to represent the basic shapes of the models. The rational spline includes all spline polynomial representations for which the denominator is trivial. Nontrivial denominators lead to all conic curves. Alpha_1 uses the Oslo algorithm[10] for computing discrete B-splines. Subdivision, effected by the Oslo algorithm, supports various capabilities including the computation associated with Boolean operations, such as the intersection of two arbitrary surfaces[21]. B-splines are an ideal design tool, they are simple yet powerful; many common shapes can be represented exactly using rational B-splines. For example, all of the common primitive shapes used in CSG systems fall into this category. Other advantages include good computational and representational properties of the spline approximation: the variation diminishing property, the convex hull property and the local interpolation property. There are techniques for matching a spline-represented boundary curve against raw data. Although the final result may be an approximation, it can be computed to any desired precision (which permits nonuniform sampling). At present, tolerancing information is not included in the object specification in Alpha_1 system.

Given the CAGD model (perhaps by combining several modeling paradigms), a corresponding set of vision models (with some control structure) is generated. Once these models are available, they provide the basis for standard 2-D and 3-D scene analysis. An early example of such an interactive system is the ACRONYM system[5,6] designed for applications in computer vision and manipulation. The world is described to ACRONYM as volume elements and their spatial relationships and as classes of objects and their subclass relationships. It uses a hybrid CSG and general sweep scheme for the representation of rigid solids. The representations are CSG-like trees whose leaves are generalized cylinders. Like PADL (a geometric modeling system[7]) it allows variation in size, limited variation in structure and variation in structural relationships of the modeled objects. However, in ACRONYM, it may be difficult to design algorithms for computing properties of objects.

The Alpha_1 modeler allows the user to design an object by giving a sequence of commands. These commands define the geometry of the object. At the same time, the result of each command can be viewed in a separate window. Although specific Alpha_1 commands are given here, we use them to describe a more general philosophy of design. The underlying motivation is to exploit wherever possible the geometric modeling system functions to provide the information required to support computer vision applications.

2.2 Front Suspension Shock Linkage

To illustrate our modeling philosophy, we will use the design of a front suspension shock linkage. This part was designed by Samuel H. Drake at the University of Utah, and was actually milled and used in a small off-road vehicle built for the SAE Mini-Baja student competition.

The design is specified in such a way so as to facilitate the machining and automatic inspection of the part. The annotated specification is now given. First, an X window is provided so that the result of every geometry creating command can be viewed. A control window is also provided for viewing transformations (see Figure 1).

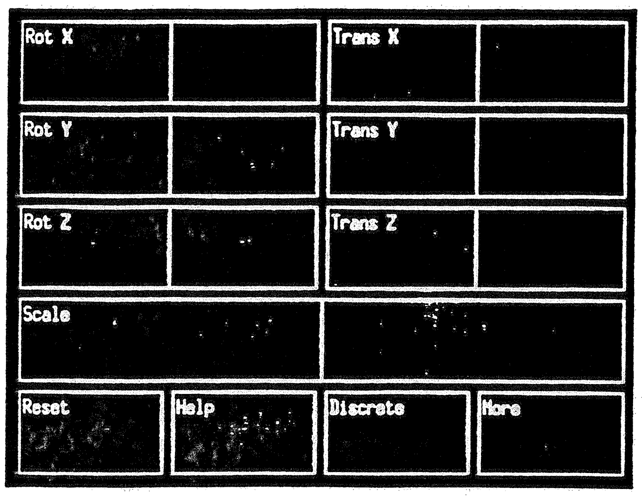
```
% creates geometry for the front suspension shock linkage for Mini-Baja vehicle

grab xgen; – Grab an X window to display the results
```

The next commands bring in the (Lisp) definitions of design features (e.g., pockets, holes, etc.) and set up reference axes.

```
{
load features; – Load feature definitions
setSrfNorms(T);

Xref := XAxis; – Define coordinate axes
Yref := YAxis;
```



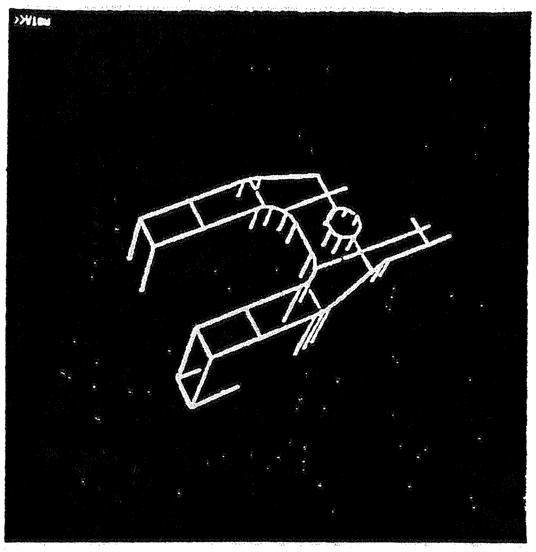


Figure 1: Design and Viewing Control Windows

```
Zref := ZAxis;
};
```

Now, a set of bounding lines can be defined; the *reverseObj* command is used to keep the normal pointing into the shape (i.e., each bounding line has an orientation).

```
{
UpLinkConstLine1 := lineVertical( -0.5 );
UpLinkConstLine2 := lineVertical( 2.5 );
UpLinkConstLine3 := lineHorizontal( 0.75 );
UpLinkConstLine4 := lineHorizontal( -0.75 );
UpLinkConstLine5 := reverseObj( lineHorizontal( 0.5 ));
UpLinkConstLine6 := lineHorizontal( -0.5 );
UpLinkConstLine7 := lineHorizontal( 0.44 );
UpLinkConstLine8 := reverseObj( lineHorizontal( -0.44 ));
UpLinkConstLine9 := reverseObj( lineVertical( 0.875 ));
UpLinkConstLine10 := lineVertical( 1.5 );
```

The following commands define the remaining points, arcs, and segments needed to specify the shape. Note that the shape is defined in the x-y plane.

```
UpLinkConstArc1 := arcRadTan2Lines( 0.275, UpLinkConstLine7, UpLinkConstLine9 )$
UpLinkConstArc2 := arcRadTan2Lines( 0.275, UpLinkConstLine9, UpLinkConstLine8 )$

UpLinkConstPt1 := centerOfArc( UpLinkConstArc1 );
UpLinkConstPt2 := centerOfArc( UpLinkConstArc2 );

UpLinkConstCir1 := circleCtrRad( UpLinkConstPt1, 0.585 )$
UpLinkConstCir2 := circleCtrRad( UpLinkConstPt2, 0.585 )$

UpLinkConstPt3 := ptIntersect2Lines( UpLinkConstLine2, UpLinkConstLine5 );
UpLinkConstPt4 := ptIntersect2Lines( UpLinkConstLine5, UpLinkConstLine10 );
```

```
UpLinkConstPt5 := ptIntersect2Lines( UpLinkConstLine3, UpLinkConstLine1 );

UpLinkConstLine11 := linePtCircle( UpLinkConstPt4, UpLinkConstCir1, T );
UpLinkConstLine12 := linePtCircle( UpLinkConstPt5, UpLinkConstCir1, nil );

UpLinkConstPt6 := ptIntersectCircleLine( UpLinkConstCir1, UpLinkConstLine11 );
UpLinkConstPt7 := ptIntersectCircleLine( UpLinkConstCir1, UpLinkConstLine12 );

UpLinkConstArc3 := arcRadTan2Lines( 0.275, UpLinkConstLine5, UpLinkConstLine11 )$
UpLinkConstArc4 := arcEndCenterEnd( UpLinkConstPt6, UpLinkConstPt1, UpLinkConstPt7 )$

UpLinkConstPt8 := ptIntersect2Lines( UpLinkConstLine1, UpLinkConstLine7 );
UpLinkConstPt9 := ptIntersect2Lines( UpLinkConstLine1, UpLinkConstLine8 );
UpLinkConstPt10 := ptIntersect2Lines( UpLinkConstLine1, UpLinkConstLine4 );
UpLinkConstPt11 := ptIntersect2Lines( UpLinkConstLine6, UpLinkConstLine10 );

UpLinkConstLine13 := linePtCircle( UpLinkConstPt10, UpLinkConstCir2, T );
UpLinkConstLine14 := linePtCircle( UpLinkConstPt11, UpLinkConstCir2, nil );

UpLinkConstPt12 := ptIntersectCircleLine( UpLinkConstCir2, UpLinkConstLine13 );
UpLinkConstPt13 := ptIntersectCircleLine( UpLinkConstCir2, UpLinkConstLine14 );

UpLinkConstArc5 := arcEndCenterEnd( UpLinkConstPt12, UpLinkConstPt2, UpLinkConstPt13 )$
UpLinkConstArc6 := arcRadTan2Lines( 0.275, reverseObj( UpLinkConstLine14 ), UpLinkConstLine6 )$

UpLinkConstPt14 := ptIntersect2Lines( UpLinkConstLine6, UpLinkConstLine2 );
```

The object's profile can now be defined:

```
UpLinkProfile1 := profile( UpLinkConstPt3, UpLinkConstArc3, – Create the 2D shape
UpLinkConstArc4, UpLinkConstPt5,
UpLinkConstPt8, UpLinkConstArc1,
UpLinkConstArc2, UpLinkConstPt9,
UpLinkConstPt10, UpLinkConstArc5,
UpLinkConstArc6, UpLinkConstPt14,
UpLinkConstPt3 )$
```

Now we define the bounding z positions and extrude the shape in 3D:

```
UpLinkExPt1 := pt( 0.0, 0.0, 0.55 ); – Define the z extent and extrude the shape
UpLinkExPt2 := pt( 0.0, 0.0, -0.55 );

UpLinkShape := extrude( UpLinkProfile1, UpLinkExPt1, UpLinkExPt2, T, T )$
};
```

Thus, the basic shape is defined (see Figures 2 and 3). Next, we add a hole:

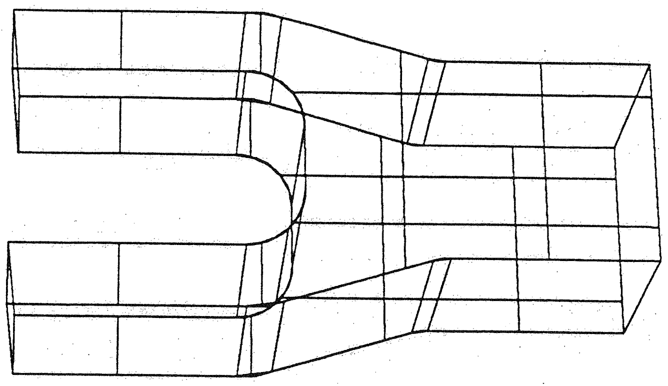


Figure 2: Surface Grid for Main Shape of Linkage

```
UplinkFixHole2 := objTransform( hole( origin, 0.3125, 0.15, 0.0, T ),
                                tx( 1.325 ), tz( 0.075 ))$ - Define a hole feature
```

We can view the hole shape with the profile shape as shown in Figures 4 and 5. However, to actually put the hole through the profile surface, we need to perform a Boolean subtraction. First, we store the two shapes (as defined by the variables UpLinkShape and UplinkFixHole2) to a file named *UpLinkTestOp.a1*.

```
dumpa1File( list( UpLinkShape, UpLinkFixHole2 ), "UpLinkTestOp.a1" )$
```

At the shell command level, we then perform the Boolean set operation which subtracts the hole from the main shape:

```
cs > set_op < UpLinkTestOp.a1 > UpOut.a1
```

The final object is shown in Figures 6 and 7.

2.3 Model Synthesis

The geometric specification described in the previous section facilitates the direct extraction of object features. However, many modeling systems do not allow this. Therefore, the easiest way to provide

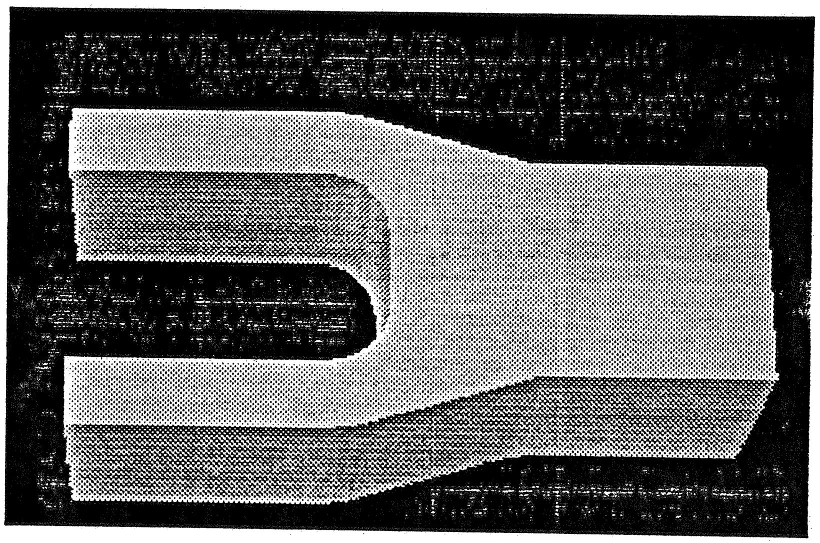


Figure 3: Rendering of Main Shape of Linkage

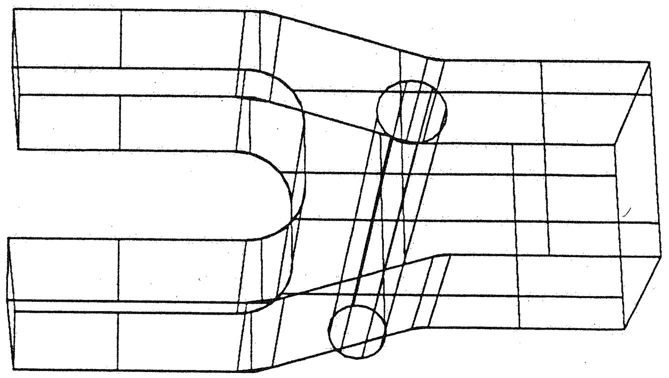


Figure 4: Surface Grid of Shape with Solid for Hole

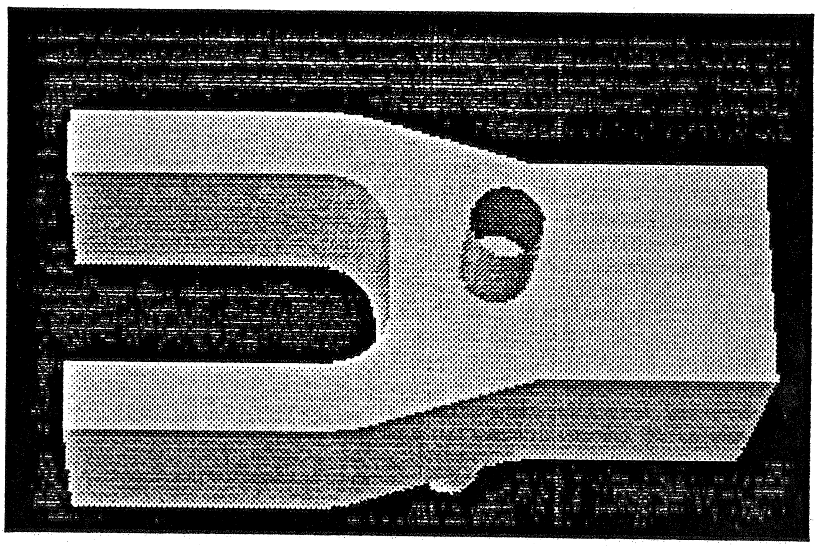


Figure 5: Rendering of Shape with Solid for Hole

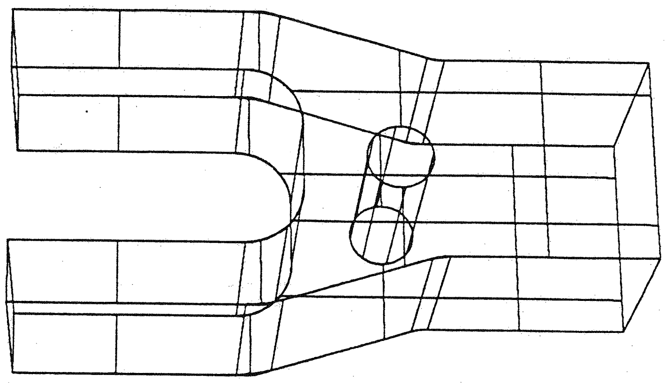


Figure 6: Surface Grid of Boolean Combined Surfaces

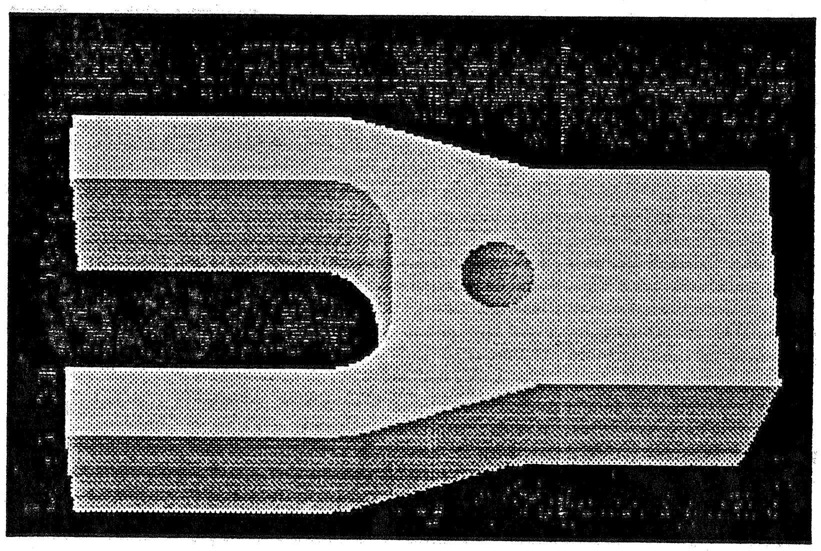


Figure 7: Rendering of Linkage

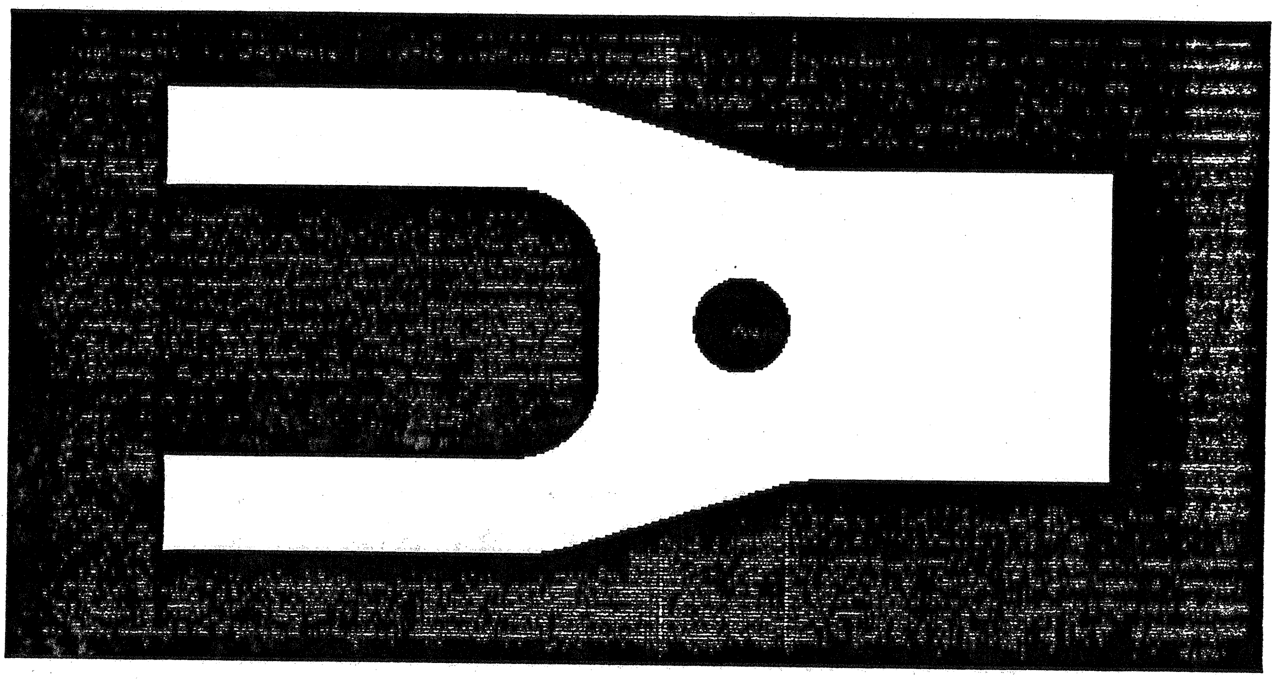


Figure 8: Orthogonal Rendering of the Linkage

models for a 2D vision system is to render an image from the CAD model and provide that image as the training set to the 2D vision system.

That is what we have done here. However, the viewing parameters must be carefully selected in order to produce an orthogonal view of the object. Figure 8 shows such a rendered version of the linkage. This constitutes the input, then, to the computer vision training techniques.

Note that this method also permits the analysis of arbitrary surface geometries, whereas a syntactic approach to discovering surface features in the Alpha_1 specification, may be quite complicated.

```
(def-rule select-lff
          :type ((?req requirements))
          :prem((not (member 'lff (applications ?req)))
                    (equal 'recognition (task ?req)))
          :conc ((assert-val ?req 'applications
                    (cons 'lff (applications ?req)))
          (make class system-specs
                    :task 'recognizer
                    :method 'lff
                    :time (time ?req)
                    :space (space ?req)
                    :accuracy (accuracy ?req))))
```

Figure 10: A FROB Forward Chaining Rule

3.1.1 System Support

The knowledge-based system must have utilities for supporting the networking of logical sensors and objects. These utilities provide the foundation from which the system is built. Higher level utilities are built on top of lower level ones for sophisticated system operations. The lowest level utility functions should have a maximal amount of flexibility since it is not known what or how more powerful constructs built upon them will be used. In the prototype system they are implemented as methods attached to FROB classes which define major components of the system. These classes and their methods form the templates from which application systems are synthesized. The application specific rules use knowledge of these templates to apply the line interpretation rules by relying on the transparent nature of the methods to handle lower level hardware or operating system specific tasks.

An example is the FROB representing the class of cameras. Knowledge about operating this class of camera is represented in a "run" method which is local to the class. It executes operating system commands which are not of concern to the object using the method. A "run" method is also provided to other sensors which have other operating system commands which are transparent to the caller of the method.

3.1.2 Language Issues

Since the application system is created from FROBS in the same environment as the knowledge-based system, the application system runs in the Common Lisp environment. To require that all of the algorithms in the system be written in Common Lisp would be a severe restriction to its flexibility. The object-based approach allows algorithms written in any language to be incorporated into the system as an algorithm object.

Methods are used to run the algorithm and provide it with the necessary I/O. Since the internal representation of the object is transparent to I/O from the outside, algorithms written in any language can be incorporated into the system as long as there are low level utilities in the system to support the methods which run them.

3 The FROBS Knowledge and Rule Base

The knowledge-base component of CBCV is written in FROBS (FRames + OBjectS) which is an object-oriented frames package that runs on top of CommonLisp and provides:

- object oriented programming
- frame based programming
- daemons
- rule based programming.

An overview of the system is given here; for more details, see the FROBS Manual[17].

3.1 Overview of FROBS

The basic building block of the FROBS package is called a module. Modules consist of a class FROB and all of its associated methods. This provides for total method and data access hiding with no distinction between methods and slots. The organization of class FROBs can be viewed as a tree structure, although more complicated schema-type structures are possible through multiple inheritance. FROB class instances are leaves of the tree.

The class frob is used to define the structure of instance frobs of that class. It is also the frob that **daemons** and **methods** are defined over. Inheritance of **methods** is done through the class frobs. A special feature of the class frob is that it is an instance of itself. It can be used like any other instance of the class. Figure 9 shows how an algorithm class FROB and a subclass FROB are defined.

FROBS are used to build both the knowledge-based vision system and the application system it synthesizes. This allows templates in the knowledge-based system to be directly used in the application system. The concept of logical sensors is implemented easily using objects to form logical sensors[14]. Class FROBS represent logical sensor templates to be instantiated for application system synthesis.

Most importantly, the FROBS package provides forward chaining rules as well as slot daemons. Slot daemons are useful for automatic data consistency checking and hidden slot calculations. The forward chaining rules provide the mechanism needed to create the knowledge base.

```
(def-class algorithm nil
          :slots (name
                  size
                  language
                  machine

(def-class feature-calculator ({class algorithm})
          :slots (feature-type
                  focus-type))
```

Figure 9: Example of FROB Class Definition

3.1.3 Object Communication Protocol

When designing a CBCV vision system using objects, there must be a well defined way for one object sensor to pass information to another. Logical sensors address this problem in an abstract sense, but a specific protocol must be chosen which has the flexibility to accept all kinds of data. The protocol is represented in the slots and methods of the logical sensor objects. There must be a way to pass information from machine to machine as well as an efficient way to pass information in the Lisp environment itself. We separate the two as different types of information passing, file piping and S-expression passing.

Passing S-expressions between objects is a trivial task. All that is required is a slot in the algorithm object which stores the expression to be passed. This slot is read by any object requiring the expression as input. To perform file piping on any host in the system, the simplest approach is to use the Unix pipe facility which allows executables to work as filters passing their output to the next program in the pipe. This is the easiest implementation since it is supported by the remote shell command "rsh" which is used to perform tasks on remote machines. It requires, however, that most programs written for the knowledge-based system be written in filter form on machine supporting Unix. This is not an unreasonable requirement since Unix is a widely supported operating system and it is good modular style to have a system designed with filters. Other programs can be run as well as filters although it is up to the user to supply names and flags in slots of the object which contains the program. Only filters are handled "automatically" by the piping method.

At some point in the CBCV system's operation, information from a remote machine will have to be read by an object in the Lisp environment or vice-versa. This requires some special processing on the part of the methods performing the pipe. To send the S-expression output of a Lisp algorithm to a non-Lisp algorithm, certain conventions must be adopted. The program receiving the S-expression must know that its input is in such a form. Each algorithm in the knowledge base must have information regarding what format it expects its input to be in and what format is produced as output. This is done with methods using slot information in the algorithm object. These methods determine what format conversions are necessary for information piped between algorithms. Information transfer between machines is performed when objects have slots indicating that their executables are on different machines.

4 2D Capabilities Applications

Although the general goal of CAD-Based Computer Vision is to recover the 3D nature of the objects in a scene, many tasks can be handled using 2D techniques. This amounts to extracting regions of interest from the 2D image and analyzing the features of those regions. There are many approaches to this problem, and we present two simple techniques which are available in CBCV: global feature matching and Local Feature Focus.

Both techniques require that objects in the image be separated from the background and that distinct connected regions have unique labels. Then, features are computed for each region. Global features are derived from some measure of the entire set of pixels of the region; for example, area is a global feature. Local features are those which are restricted in spatial extent and only require a small percentage of pixels from the region; corners and holes are examples of local features.

Figure 11 shows the steps involved in applying these matching techniques. The training data is used to construct a model of the object under consideration. As indicated earlier, such a model is usually based on visual features of the object; other kinds of features could be used, such as weight or surface roughness, but those will not be considered here. The standard approach to get the visual features of the object is to examine several views (digital images) of the object. These then constitute the training data, and after the features of the object are extracted, then some statistical analysis of the features is performed. Robust features are then selected to represent the object; that is, the mean values of the features are determined as well as their variances. Finally, some sort of comparison measure is selected; this includes both distance functions (e.g., Euclidean, Mahalanobis, Manhattan), as well as similarity measures (e.g., the correlation coefficient).

The images used as training data can be obtained several ways. Sometimes an actual part is available and images are taken from the camera and image acquisition system. When a CAD model is available, the test images can be produced by rendering several views of the object. Alternatively, one view is sufficient if the statistical properties of the feature calculation processes are known.

It may be possible to determine the vision model directly from the CAD model without resorting to the use of images. For example, the surface area of a face can be calculated from the definition of the profile curve. When using manufacturing features such as pockets or holes, their dimensions are usually part of the definition of the part. In this case, ideal values are obtained, and it is necessary to take into account the error introduced by the manufacturing process and the image acquisition system.

4.1 Global Feature Matching

The *feature class* is defined as a very simple class consisting of these slots:

- name: the name of the feature,
- command: the executable command line, and
- features: a list comprised of:
  - a select switch (T or NIL)
  - the mean value of the feature, and
  - the variance of the feature.

The **FROB** definition for this is:

```
(def-class feature nil
:slots (name                                  ;; Feature name.
        command                               ;; Command line for executable.
        features                              ;; Feature list (T/NIL mean var).
))
```

Once the *feature class* has been defined, individual features can be defined. For example, the *area feature* is defined as:

```
(def-class area class feature
:init ((nil (name 'area)
            (command "path/area")))
)
```

Thus, *area* is a *feature* and has the name 'area' and is invoked by running '*path*/area' at the shell level. Of course, '*path*' must be expanded into the correct path to the area binary file.

Other features are defined similarly: *aspect, diameter, n1, n2, n3, n4, n5, n6, n7, perimeter* and *thinness*. These frames comprise the knowledge (at this level of the *ISA* hierarchy) about features. However, to create executable instances requires the definition of the 'make' method. for example:

```
(def-method (class area make) ()
   (let ((instance (new-instance $self)))
     instance))
```

defines a method for making instances of the *area* feature.

Now we can define useful methods which operate on features. The most basic operation is to run the feature operator on an image to produce a feature value for each connected region in the image:

```
(def-method (class feature run) (inputdata)
  (let ((stream (start-feature-process inputdata (command $self))))
    (unwind-protect
      (read-feature-output stream)
      (close stream))))
```

The *start-feature-process* function takes inputdata as the segmented image and passes it to the feature command line. The output is then piped into the variable *stream* and is read from there by the *read-feature-output* function which is returned as the value of the method.

For example, to run the area feature on the image *scene.img*, the following command is issued:

```
(run {class area} "scene.img")
```

4.1.1 An Example

We now give an example of CBCV using the global feature matching technique to inspect the linkage.

The training command is given first:

```
(setq *r* (make-recognizer :hint 'global :training-data *crocfiles*))
```

where *crocfiles* is a FROB variable that has the name of the training files (see Figures 12, 13 and 14). The result is an algorithm:

```
ALGORITHM 0 has the following values:
(ALGORITHM ALGORITHM) ⇒ GLOBAL 0
(ALGORITHM HINT) ⇒ GLOBAL
(ALGORITHM OCCLUSION) ⇒ NIL
(ALGORITHM SPACE) ⇒ NIL
(ALGORITHM TIME) ⇒ NIL
(ALGORITHM NAME) ⇒ NIL
```

All features are tested for robustness, and in this case, only perimeter, n7, diameter, and area survive:

```
THINNESS 2 has the following values:
(FEATURE NAME) ⇒ THINNESS
(FEATURE COMMAND) ⇒ " A/thinness"
(FEATURE FEATURES) ⇒ NIL


PERIMETER 2 has the following values:
(FEATURE NAME) ⇒ PERIMETER
(FEATURE COMMAND) ⇒ " A/perimeter"
(FEATURE FEATURES) ⇒ ((T 824.333333333333 4016.88888888889))


N7 2 has the following values:
(FEATURE NAME) ⇒ N7
(FEATURE COMMAND) ⇒ " A/moments — A/n7"
(FEATURE FEATURES) ⇒ ((T -1.17363033333333 2.85431257070555))


N6 2 has the following values:
(FEATURE NAME) ⇒ N6
(FEATURE COMMAND) ⇒ " A/moments — A/n6"
(FEATURE FEATURES) ⇒ NIL


N5 2 has the following values:
(FEATURE NAME) ⇒ N5
(FEATURE COMMAND) ⇒ " A/moments — A/n5"
(FEATURE FEATURES) ⇒ NIL


N4 2 has the following values:
(FEATURE NAME) ⇒ N4
(FEATURE COMMAND) ⇒ " A/moments — A/n4"
(FEATURE FEATURES) ⇒ NIL
```

```
N3 2 has the following values:
(FEATURE NAME) ⇒ N3
(FEATURE COMMAND) ⇒ " A/moments —  A/n3"
(FEATURE FEATURES) ⇒ NIL


N2 2 has the following values:
(FEATURE NAME) ⇒ N2
(FEATURE COMMAND) ⇒ " A/moments —  A/n2"
(FEATURE FEATURES) ⇒ NIL


N1 2 has the following values:
(FEATURE NAME) ⇒ N1
(FEATURE COMMAND) ⇒ " A/moments —  A/n1"
(FEATURE FEATURES) ⇒ NIL


DIAMETER 2 has the following values:
(FEATURE NAME) ⇒ DIAMETER
(FEATURE COMMAND) ⇒ " A/diameter"
(FEATURE FEATURES) ⇒ ((T 285.096666666667 10.4814888888889))


ASPECT 2 has the following values:
(FEATURE NAME) ⇒ ASPECT
(FEATURE COMMAND) ⇒ " A/aspect"
(FEATURE FEATURES) ⇒ NIL


AREA 4 has the following values:
(FEATURE NAME) ⇒ AREA
(FEATURE COMMAND) ⇒ " A/area"
(FEATURE FEATURES) ⇒ ((T 12527.0 1248.66666666667))
```

Next, the global matcher can be run on a scene image (see Figure 15):

```
(match *r* :image-data "scene-image.img"))
```

Then the results of the global analysis are reported:

```
Feature match success: "scene1-seg.img"
T
```

4.2 Local Feature Matching

The Local Feature Focus method proposed by Bolles[3] is a robust 2D shape recognition and localization scheme. The method is organized as follows:

- Model Building

   1. Enumeration of potentially useful features
   2. Location of structurally equivalent features
   3. Enumeration of secondary feature groups
   4. Selection of secondary feature groups
   5. Ranking of focus features

- Recognition and Localization
   1. Get scene features
   2. Match focus graph
   3. Generate hypothesis (including pose transformation)
   4. Verify match

4.2.1 An Example

We will now use this technique to locate the linkage part. The CBCV system invokes a sequence of filters to obtain the features from the CAD image of the object (see Figure 16). First, a boundary file is produced (see Figure 17). Next, the center of mass of the object is found: (104.326981 104.861526); this is used to produce feature descriptions located with respect to the object's center of mass. The corners are found next:

```
18   66    90.00045.000
174  79    90.000135.000
174  131   90.000225.000
18   144   90.00045.000
18   129   90.00045.000
18   82    90.00045.000

#
```

From this information, the feature types are determined:

```
0  circular    hole    3  x y   diameter
1  noncircular hole    3  x y   diameter
2  external    corner  4  x y   angle     orientation
3  internal    corner  4  x y   angle     orientation

@

0  0  diameter   0.00
1  0  diameter   0.00
1  1  diameter  19.35
2  0  angle      0.00
2  1  angle     90.00
3  0  angle      0.00
```

```
 @

  2   1  -86.33  -38.86   90.00  -45.00
  2   1   69.67  -25.86   90.00 -135.00
  2   1   69.67   26.14   90.00 -225.00
  2   1  -86.33   39.14   90.00   45.00
  2   1  -86.33   24.14   90.00  -45.00
  2   1  -86.33  -22.86   90.00   45.00
  1   1    9.17    0.45   19.35    0.00
```

From this information, the focus features are determined:

This then constitutes the model.

Given a scene, as shown in Figure 18, the system must discover the transformation. The features in the scene are determined in much the same way as for the model image, and are as follows:

```
0  circular    hole      3  x y      diameter
  1  noncircular hole      3  x y      diameter
  2  external    corner    4  x y      angle     orientation
  3  internal    corner    4  x y      angle     orientation

 @

  0   0   diameter    0.00
  1   0   diameter    0.00
  1   1   diameter   19.35
  2   0   angle       0.00
  2   1   angle      90.00
  3   0   angle       0.00

 @

  2   1   36.48  -85.03   97.13 -138.81
  2   1   26.48   68.97   97.13 -228.81
  2   1  -24.52   69.97   90.00   45.00
  2   1  -40.52  -84.03   90.00  -45.00
  2   1  -24.52  -83.03   97.13 -138.81
  2   1   20.48  -85.03   97.13  -41.19
  1   1    0.01    8.85   21.89    0.00
```

After matching model features to scene features, the system identifies the transformation, and produces the match shown in Figure 19.

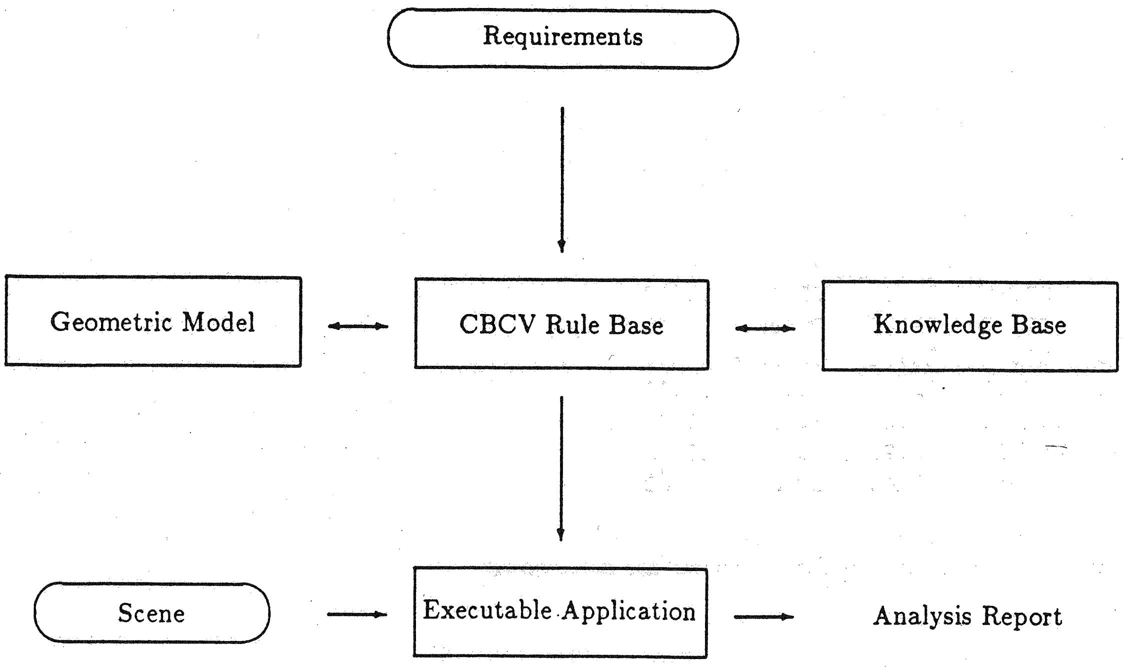


Figure 11: Vision Paradigm

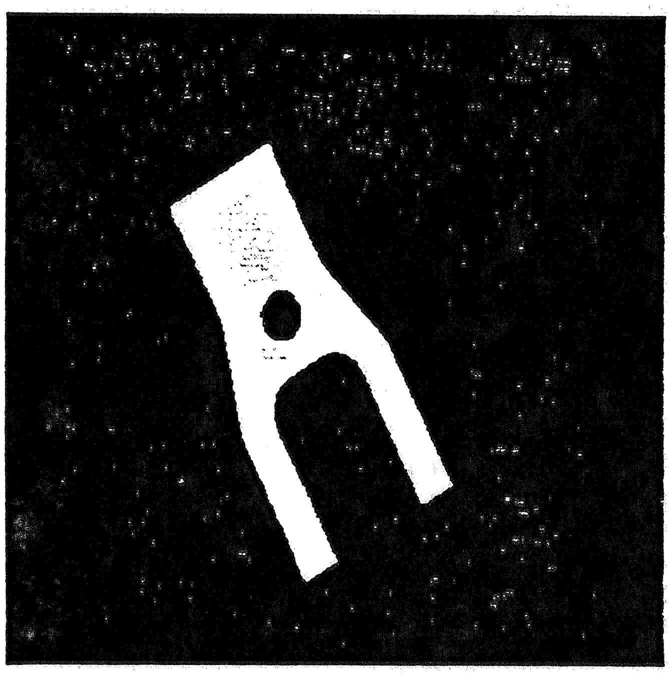


Figure 12: Training Image 1

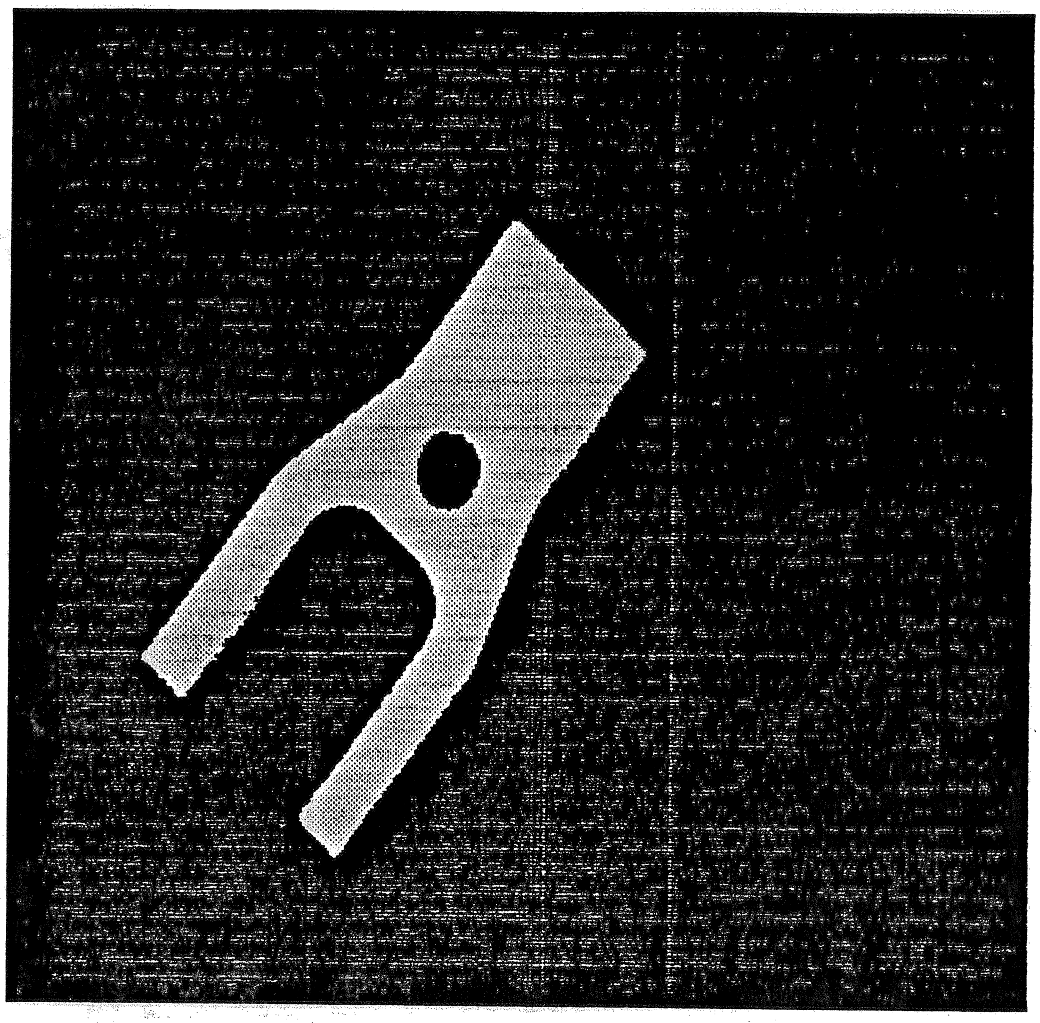


Figure 13: Training Image 2

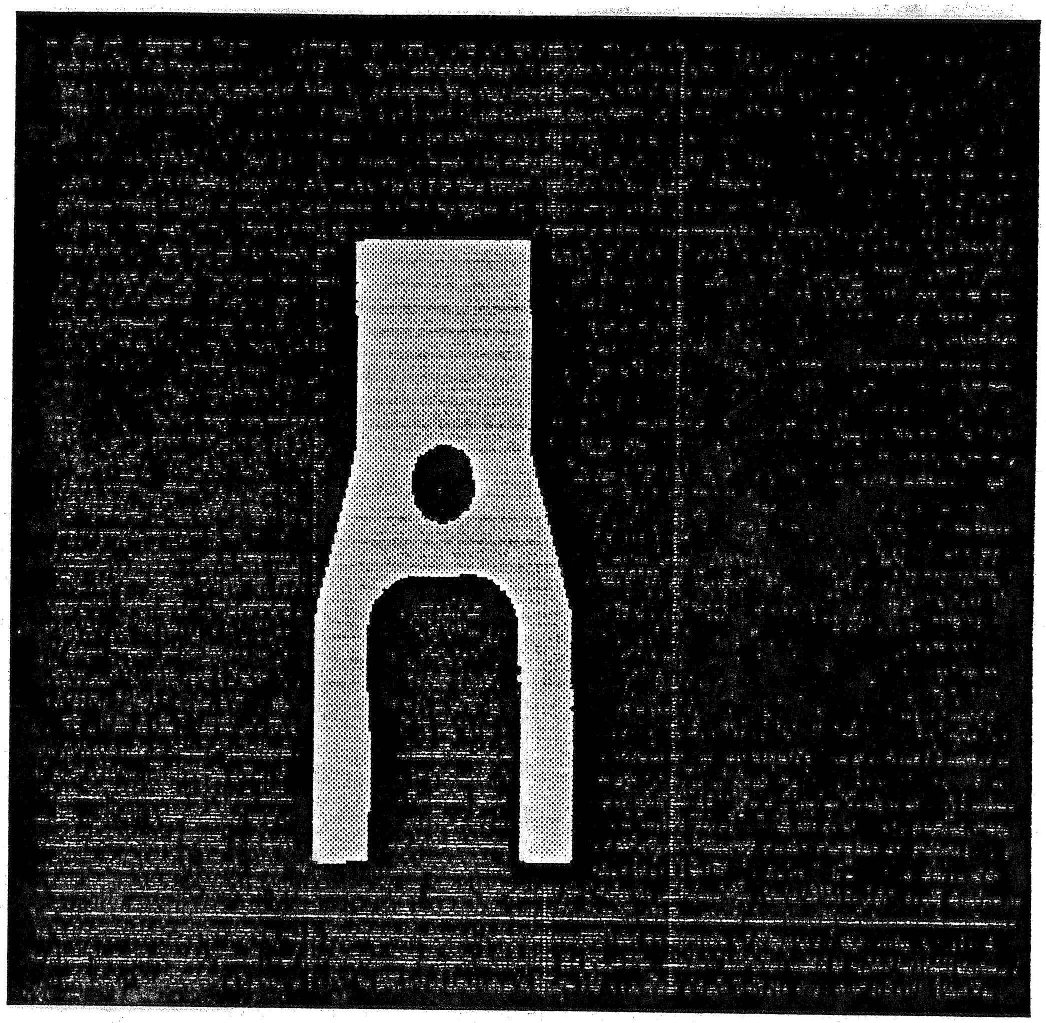


Figure 14: Training Image 3

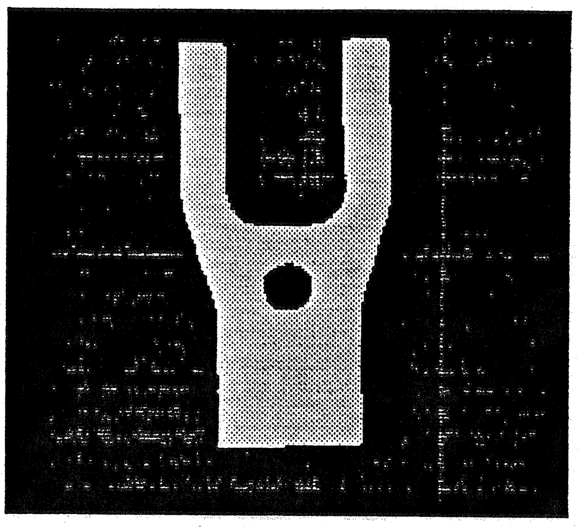


Figure 15: Scene Image for Global Matching

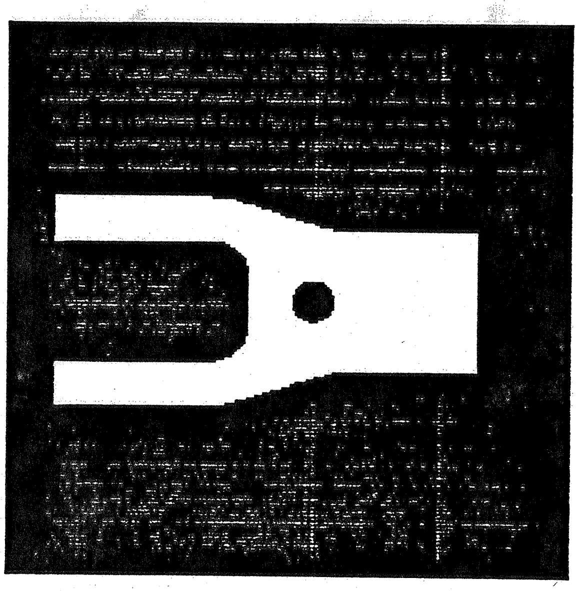


Figure 16: CAD Model Image

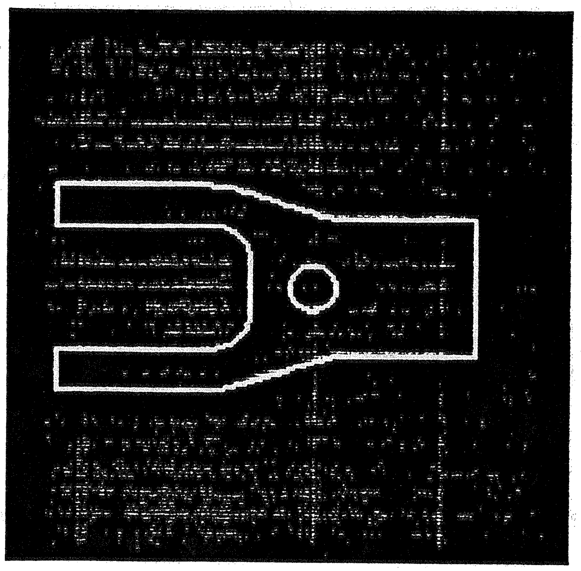


Figure 17: CAD Model Boundary Image

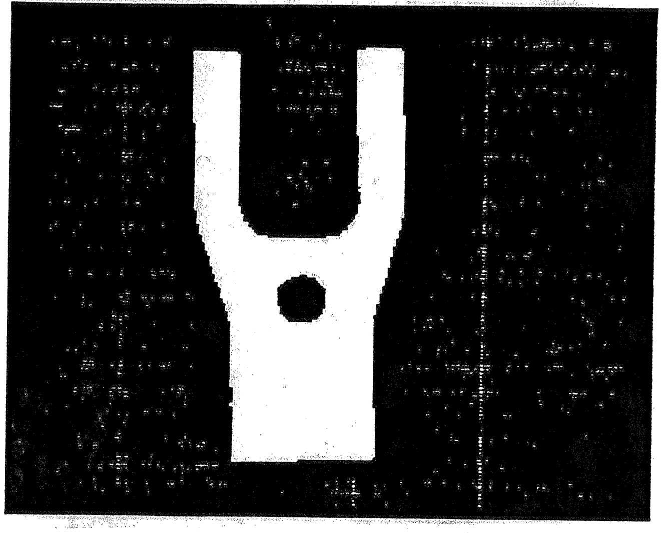


Figure 18: Scene Image

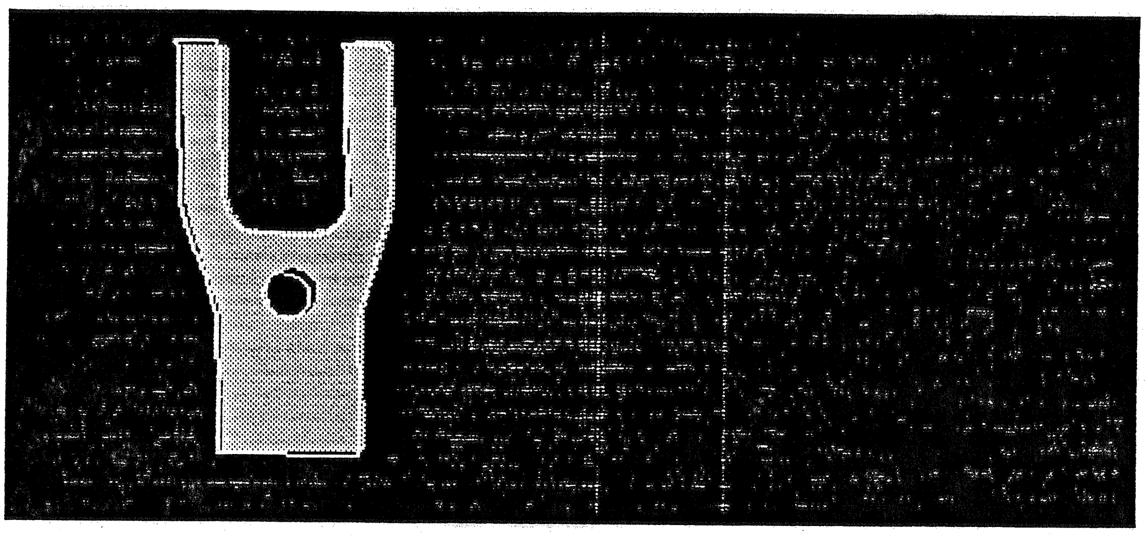


Figure 19: Model Superimposed on Scene

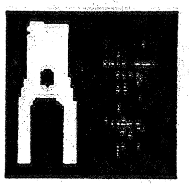


Figure 20: Example image

5 Filters

In this section we give brief descriptions of the filters available for image analysis and manipulation. Examples are given to illustrate each. The examples will be given in terms of the image shown in Figure 20. (This is a 64x64 version of the original 512x512 image shown in Figure 22.)

5.1 AREA

The area filter takes in an image that has been run through segment and outputs the area for each region in the segmented image. The output is organized with one area per line with the first line the area of region 1, the second line the area of region 2, etc. The output for the region in Figure 20 is:

```
626
```

5.2 ASPECT

The aspect filter produces the aspect ratio for each object in a segmented image and outputs them one per line. The output for the region in Figure 20 is:

```
2.571429
```

5.3 BF

The bf filter implements a boundary following routine and given a segmented image as input, it outputs:

```
* x y c1 c2 c3 ... cn
```

for each region found, where the plus symbol signifies an exterior boundary, the boundary starts at (x,y) and the ci's are the chain codes of the boundary. For every hole within a connected region, the output is:

```
# x y c1 c2 c3 ... cn
```

The output for the region in Figure 20 is:

```
* 9 4  0  0  0  0  7  0  0  0  0  0  0  0  0  6  6  6  6  6  6  6  6  6  6
       6  6  6  6  6  6  6  6  7  6  6  6  7  6  6  6  6  7  6  6  6  6  6  6
       6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  4  4  4  2  2  2  2
       2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  3  3  3  3  4  4  4
       4  4  4  4  5  5  6  5  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6
       6  6  6  6  6  4  4  4  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
       2  2  2  2  2  2  2  2  1  2  2  2  1  2  2  1  2  2  2  2  2  2  2  2
       2  2  2  2  2  2  2  2  1  3  2  1  2
# 13 22  5  6  6  6  6  7  0  7  1  0  1  2  2  2  2  3  3  4  4  5
```

Thus, we see that there is one region and its external boundary is given after the , while there is one hole within that region and its boundary is given after the #.

5.4 CENTER_OF_MASS

The center_of_mass filter takes in a segmented image and returns the (x,y) location of the center of mass for each region, one pair of numbers per line. The output for the region in Figure 20 is:

```
15.001597 27.215654
```

Thus, the x mean is 15.001597, while the y mean is 27.215654.

5.5 CORNER

The corner filter takes in the output of the bf filter and produces a list of all the corners of each region, one set of corners per line. The output includes:

```
x y theta alpha
```

where (x,y) is the location of the corner, theta gives the angle of the corner, and alpha gives the orientation of the corner (i.e., the direction of the bisector of the angle). The output for the region in Figure 20 is:

```
*
    9    4   116.565    -63.435
   22    5    90.000   -135.000
   25   57    90.000   -225.000
   22   57    90.000     45.000
    8   57    90.000   -225.000
    5   57    90.000     45.000
    8    6   126.870      0.000

#
   12   27   243.435    206.565
   18   27   243.435    -26.565
```

This indicates that there are 7 external boundary corners (the first and last are actually both generated by the real corner, as can be seen by the angle assigned to each corner), and there are two corners found in the hole boundary (these are artifacts of the small image size).

5.6 DIAMETER

The diameter filter takes in a segmented image and produces the greatest cross section of each region and outputs them one per line. The output for the region in Figure 20 is:

```
64.11
```

This is the longest chord across the object.

5.7 FIX-IMAGE

This shell script massages the IKS version image from the VICOM into a usable format for the rest of the filters; in particular, it is:

```
iwindow -r60 -c60 -w380 -d380 | \
threshold 53 | \
iffmap -t /n/sunset/c/cs537/cbcv/lib/fliplut | \
segment
```

Thus, this is a hardwired set of transformations from VICOM to segmented images.

As an example, given the image shown in Figure 21 brought in by get-image, then fix-image produces the image shown in Figure 22.

5.8 GET-IMAGE

This shell script calls up the VICOM to obtain an image from the Fairchild camera in the CED lab. It returns a 512x512 IKS image and requires that the user has an account on the croc vax. The script is:

```
rsh croc /n/croc/u/tch/vsh/real_stuff/hp/get-image
image 512 512 < /n/croc/tmp/scene.img > $1
```

See Figure 21 for an image acquired with get-image.

5.9 HOLE_FEATURES

The hole_features filter takes as input either a segmented image (or the output from bf with the -cc option) and produces a set of features for each hole in each region. For each hole the the output is:

```
center_of_mass diameter area perimeter
```

The output for the region in Figure 20 is:

```
  15.000    24.837      9.327      49      20
```

Thus, the center of the hole is at $x = 15.0$ and $y = 24.837$, while the diameter is 9.327, the area is 49, and the perimeter is 20.

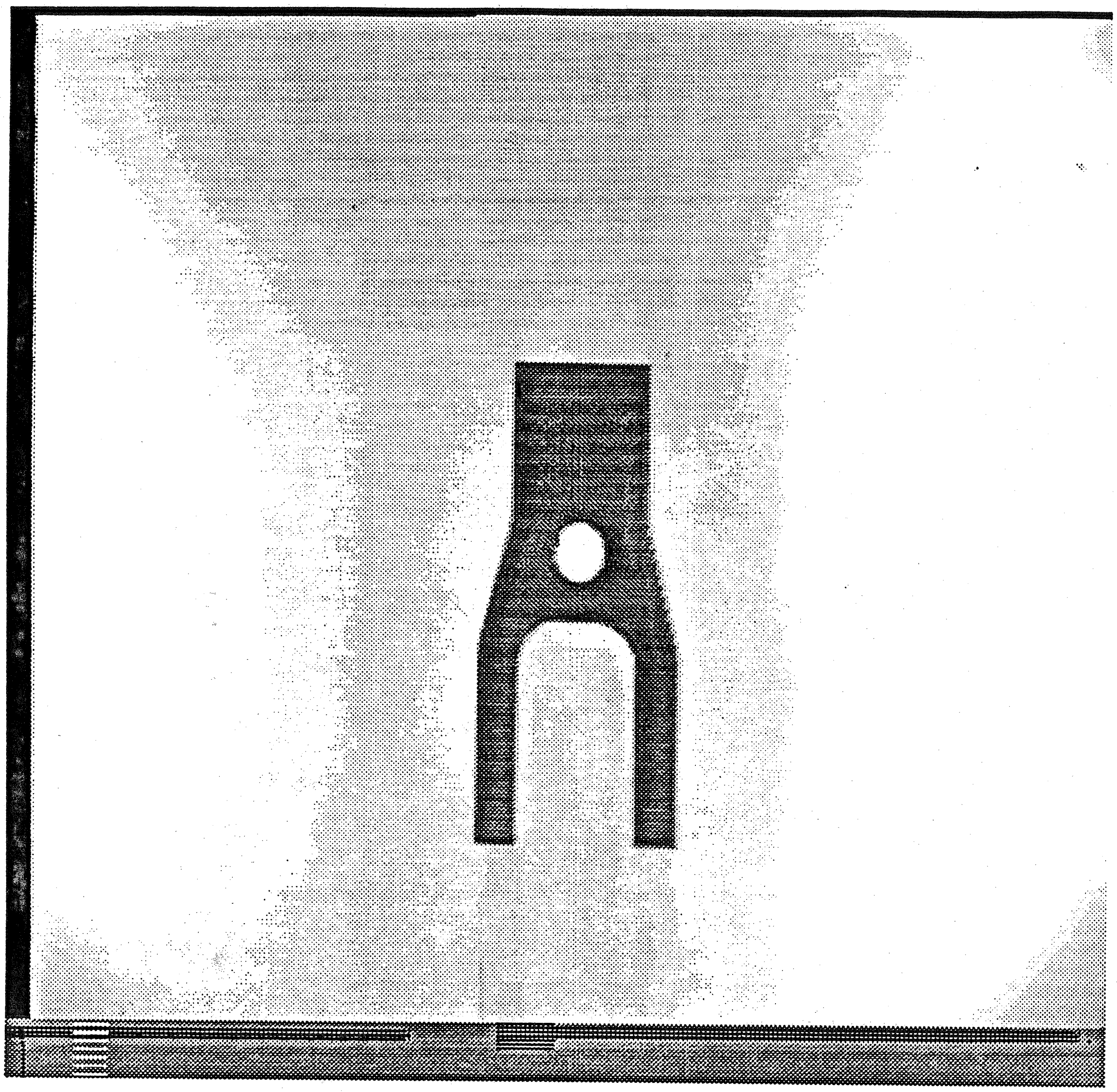


Figure 21: VICOM image

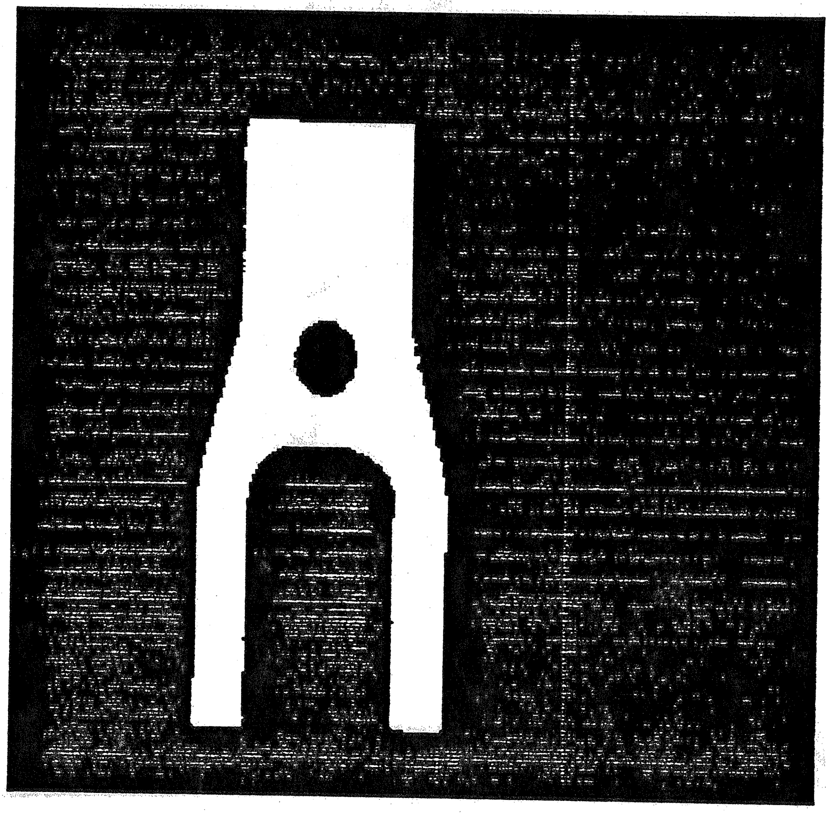


Figure 22: Fixed image

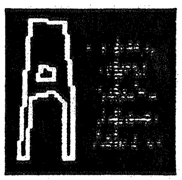


Figure 23: Linear Approximations to the Shape

5.10 LINAPP

The linapp filter takes in the output of bf and generates a sequence of linear segment approximations to the boundary. Each line of output is:

```
x1 y1 x2 y2
```

The output for the region in Figure 20 is:

```
*
   9    4   22    5
  22    5   25   34
  25   34   25   57
  25   57   22   57
  22   57   22   36
  22   36   18   32
  18   32    9   34
   9   34    8   57
   8   57    5   57
   5   57    9    4
#
  13   22   12   27
  12   27   18   27
  18   27   18   23
  18   23   13   22
```

Thus, there are 10 segments generated on the external boundary and 4 on the internal hole boundary. Figure 23 shows the segments overlaid on the original scene.

5.11 MOMENTS

This filter takes in a segmented image and produces the following moments: μ_{00}, μ_{01}, μ_{10}, μ_{11}, μ_{20}, μ_{02}, μ_{21}, μ_{12}, μ_{03}, μ_{30}, where

$$\mu_{pq} = \int_x \int_y x^p y^q dx dy$$

The output for the region in Figure 20 is:

```
626.000000 22401.000000 9391.000000 335895.000000 165175.000000 936743.000000
5707821.000000 14036605.000000 42547212.000000 3206647.000000
```

Thus, the area is 626, for example.

5.12 N1

The n1 filter takes in a segmented image and for each region produces the following invariant moment:

$$\varphi_1 = \eta_{20} + \eta_{02}$$

where

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}}$$

and

$$\gamma = \frac{p+q}{2} + 1$$

The output for the region in Figure 20 is:

```
0.406845
```

5.13 N2

The n2 filter takes in a segmented image and for each region produces the following invariant moment:

$$\varphi_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2$$

where

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}}$$

and

$$\gamma = \frac{p+q}{2} + 1$$

The output for the region in Figure 20 is:

```
0.080006
```

5.14 N3

The n3 filter takes in a segmented image and for each region produces the following invariant moment:

$$\varphi_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2$$

where

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}}$$

and

$$\gamma = \frac{p+q}{2} + 1$$

The output for the region in Figure 20 is:

```
0.000029
```

5.15 N4

The n4 filter takes in a segmented image and for each region produces the following invariant moment:

$$\varphi_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2$$

where

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}}$$

and

$$\gamma = \frac{p+q}{2} + 1$$

The output for the region in Figure 20 is:

```
0.007399
```

5.16 N5

The n5 filter takes in a segmented image and for each region produces the following invariant moment:

$$\varphi_5 = (\eta_{30}-3\eta_{12})(\eta_{30}+\eta_{12})\left[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2\right]+(3\eta_{21}-\eta_{03})(\eta_{21}+\eta_{03})\left[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right]$$

where

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}}$$

and

$$\gamma = \frac{p+q}{2} + 1$$

The output for the region in Figure 20 is:

```
0.000003
```

5.17 N6

The n6 filter takes in a segmented image and for each region produces the following invariant moment:

$$\varphi_6 = (\eta_{20} - \eta_{02}) \left[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03})$$

where

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}}$$

and

$$\gamma = \frac{p+q}{2} + 1$$

The output for the region in Figure 20 is:

```
0.002093
```

5.18 N7

The n7 filter takes in a segmented image and for each region produces the following invariant moment:

$$\varphi_7 = (3\eta_{12}-\eta_{30})(\eta_{30}+\eta_{12}) \left[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2\right]+(3\eta_{12}-\eta_{03})(\eta_{21}+\eta_{03}) \left[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right]$$

where

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}}$$

and

$$\gamma = \frac{p+q}{2} + 1$$

The output for the region in Figure 20 is:

```
0.000041
```

5.19 ORIENTATION

The orientation filter takes a segmented image as input and for each region outputs the difference angle between the y-axis and the principle axis of the region. The output for the region in Figure 20 is:

```
-89.91947
```

5.20 PERIMETER

The perimeter filter takes in a segmented image and for each region produces on output the perimeter of the region, one per line. The output for the region in Figure 20 is:

```
201
```

5.21 RLE2IFF

The rle2iff filter takes as input an rle format image and produces on output an IKS image.

5.22 ROTATE

The rotate filter takes a segmented image as input and for each region rotates the region the specified angle. It translates it the specified amount in x and y after centering in the image. The output is a similar-sized IKS image.

5.23 THINNESS

The thinness filter takes a segmented image as input and for each region outputs the thinness measure of the region. The output for the region in Figure 20 is:

```
81.591057
```

References

[1] R.E. Barnhill and R.F. Riesenfeld eds. *Computer Aided Geometric Design*. Academic Press, New York, 1974.

[2] B. Bhanu and C.C. Ho. CAGD-Based 3-D Object Representations for Computer Vision. *IEEE Computer*, 20(8):19–36, August 1987.

[3] R.C. Bolles and R.A. Cain. Recognizing and Locating Partially Visible Objects: The Local-Feature-Focus Method. *Robotics Research*, 1(3):57–82, 1982.

[4] R.C. Bolles and P. Horaud. 3DPO: A Three-Dimensional Part Orientation System. *Robotics Research*, 5(3):3–26, 1986.

[5] R. Brooks, R. Greiner, and T.O. Binford. The ACRONYM Model Based Vision System. In *Proc. 6th IJCAI*, pages 105–113, Tokyo, 1979.

[6] R.A. Brooks. Symbolic Reasoning Among 3-D Models and 2-D Images. *Artificial Intelligence*, 17:285–348, 1981.

[7] C.M. Brown. PADL-2: A Technical Summary. *IEEE Computer Graphics and Applications*, 69–84, March 1982.

[8] Elizabeth Cobb. *Design of Sculptured Surfaces Using the B-Spline Representation*. PhD thesis, Univserity of Utah, Salt Lake City, Utah, June 1984.

[9] E. Cohen. Some Mathematical Tools for a Modeler's Workbench. *IEEE Computer Graphics and Applications*, 63–66, October 1983.

[10] E. Cohen, T. Lyche, and R.F. Riesenfeld. Discrete B-Splines and Subdivision Techniques in Computer Aided Geometric design and Computer Graphics. *Computer Graphics and Image Processing*, 14(2):87–111, October 1980.

[11] C. Goad. Special Purpose, Automatic Programming for 3D Model-Based Vision. In *Proceedings of the DARPA Image Understanding Workshop*, pages 94–104, DARPA, 1983.

[12] C.D. Hansen. *CAGD-Based Computer Vision: The Automatic Generation of Recognition Strategies*. PhD thesis, The University of Utah, Salt Lake City, Utah, July 1988.

[13] Charles D. Hansen and Thomas C. Henderson. CAGD-Based Computer Vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-11(10):1181–1193, 1989.

[14] Thomas C. Henderson, Eliot Weitz, Chuck Hansen, and Amar Mitiche. Multisensor Knowledge Systems: Interpreting 3D Structure. *International Journal of Robotics Research*, 7(6):114–137, 1988.

[15] C.C. Ho. *CAGD-Based 3-D Object Representations for Computer Vision*. Master's thesis, University of Utah, Salt Lake City, Utah, June 1987.

[16] K. Ikeuchi. Model-Based Interpretation of Range Imagery. In *Proceedings of the DARPA Image Understanding Workshop*, pages 321–339, DARPA, 1987.

[17] Eric Muehle. *FROBS Manual*. Technical Report PASS-note-86-11, University of Utah, October 1986.

[18] A. Requicha. Representations for Rigid Solids: Theory, Methods, and Systems. *Comp. Surv.*, 12(4):437–464, December 1980.

[19] A. Requicha and H.B. Voelcker. Solid Modeling: A Historical Summary and Contemporary Assessment. *IEEE Computer Graphics and Applications*, 9–24, March 1982.

[20] A. Requicha and H.B. Voelcker. Solid Modeling: Current Status and Research Directions. *IEEE Computer Graphics and Applications*, pp. 25–37, October 1983.

[21] Spencer Thomas. *Modelling Volumes Bounded by B-Spline Surfaces.* PhD thesis, University of Utah, Salt Lake City, Utah, June 1984.

[22] Eliot Weitz. *Knowledge-Based 2D Vision System Synthesis.* Master's thesis, University of Utah, Salt Lake City, Utah, June 1987.