THE ICON COMPUTER PERCEPTION LABORATORY USERS GUIDE AND REFERENCE MANUAL

by

Lorenz A. Schmitt

Computer Sciences Technical Report #421

March 1981

The ICON Computer Perception Laboratory Users Guide and Reference Manual

Lorenz A. Schmitt

University of Wisconsin - Madison

ABSTRACT

The ICON Computer Perception Laboratory is an integrated collection of software designed to allow a user to easily research and design computer vision systems. The ICON system is based upon the recognition cone paradigm and uses these cones to perform the visual tasks. It allows the user to interactively create, maintain, manipulate, test and refine systems to recognize real and artificial scenes. The objective of the laboratory is to provide an easy to use but sophisticated interactive environment for both the naive and experienced computer user. The system provides not only the capability of creating and maintaining recognition cone systems, but also attempts to analyze the run time characteristics of these systems in order to give the user feedback about the efectiveness of his design.

TABLE OF CONTENTS

APPENDICIES

The ICON Computer Perception Laboratory Users Guide and Reference Manual

Lorenz A. Schmitt

University of Wisconsin - Madison

SECTION I
INTRODUCTION

1.

1.1. Design Overview

The ICON Computer Perception Laboratory is designed to allow a user access to a wide variety of hardware and software for the purposes of performing computer vision tasks. The software described here is a part of a larger body of software which provides the device (ie. TV camera, graphics terminals) dependant software as well as general routines to handle pictures (ie. intensity arrays). This document describes the software which is specific to the implementation of a computer vision laboratory employing the recognition cone paradigm [1,2,3]. This software is available for use on the Computer Science Departments VAX 11/780 and provides the user with the means to develop, maintain and interface all of the necessary data structures, software routines and hardware required by the Recognition Cone operation. The ICON Laboratory software package consists of five major programs and the related files necessary for their operation. All five programs work together to allow the user to interact with the recognition process in such a way that the development of an effective recognition program comes easily and naturally.

The first major program is the TRANSFORM EDITOR. This is a program which provides an interactive means of creating and maintaining specific recognition cone configurations. These configurations can then be interpreted and monitored all under control of the TRANSFORM EDITOR program. This piece of software uses the Transform Library Interface Module (see APPENDIX G) to allow a user to create transforms, catalog them with meaningful attributes and assign them to specified layers of a Recognition Cone to create a specific configuration. The editor is easy enough to use so that a person unfamiliar with computer vision techniques in general and Recognition Cones in particular can create meaningful transforms in a very short time. The

editor is also powerful enough so that the advanced user can utilize all of the inherent capabilities of a Recognition Cone system. Section III describes the operation of the TRANSFORM EDITOR.

Section IV describes the second program, the RECOGNITION CONE INTERPRETER. This program takes a configuration developed by the Transform Editor program and interprets it. This program is activated as a separate process which in initiated by the TRANSFORM EDITOR program. The user can thus continue the monitoring, analysis and development of other configurations concurrently with the interpretation of a recognition cone configuration.

The third piece of software is the RECOGNITION CONE MONITOR program. The operation and results of the RECOGNITION CONE INTERPRETER are displayed by this program when requested by the user. The monitoring and display of the recognition cone results are requested through the TRANSFORM EDITOR program which passes these requests to the RECOGNITION CONE MONITOR program. These results can be displayed upon graphics or character terminals or upon hard copy devices. Section V contains a description of the RECOGNITION CONE MONITOR program.

The fourth program is the RECOGNITION CONE STATISTICS program. The information gathered during the interpretation of a recognition cone are analyzed by this program which will produce reports indicating the effectiveness of all parts of the designed cone configuration. These reports can be used to refine and enhance the vision process. Section VI describes this program.

The fifth program is the RECOGNITION CONE COMPILER. This program will take a configuration and output a parallel pascal program [4,5] which is then compiled using the parallel pascal precompiler. This program will be a direct compiled implementation of the configuration designed by the user. This program will execute much faster than the RECOGNITION CONE INTERPRETER program but usually is much larger than the interpreted version. This program is described in section VII.

These five pieces of software are designed to provide a freindly interactive environment for the development of recognition programs by both naive and experienced users. The design of the system allows a user to develop, test, change, retest and finally execute his design of a recognition system.

1.2. Operating Environment

The vision laboratory is designed to be an interactive system which allows the user a great deal of flexibility in

creating and testing vision programs. An integral part of any such laboratory is the software and hardware environment in which it runs. For vision research, this environment must include mechanisms to manipulate, store and retrieve images which are going to be used. The handling of images is not done directly by the ICON programs. Instead it is done by a package called PICT . This PICT software allows one to take pictures with a TV camera, save and retrieve these pictures, display the pictures upon several different graphic devices and to manipulate the pictures in various ways. All of these things can and must be done independantly of the other programs in the ICON system. The RECOGNITION CONE INTERPRETER depends on receiving an image produced by PICT. For this reason the user should become familiar with dealing with images using the PICT software.

One of the primary concerns addressed by this laboratory is the necessity of providing automatic methods of analyzing the effectiveness of the vision process. The monitoring of the recognition cones and the gathering of statistical information is integrated into the system in such a way that a user will get the information he needs to more effectively refine and improve the recognition process.

SECTION II
RECOGNITION CONES

2.

2.1.

In order to work effectively with this software, the user must be familiar with the principles and workings of a recognition cone as a means of visually analyzing a scene. A very brief description will be given here. For more detailed information the user should refer to the articles mentioned in the bibliography.

A recognition cone is the name for a specific class of computer vision paradigms. The cone consists of a particular type of data structure along with specific algorithms operating upon that structure. The data structures used are commonly called TRANSFORMS and LAYERS. The algorithms which use these data structures are called TRANSFORMATIONS.

2.2. Transforms

A transform consists of three parts, a global part, a lookfor part, and an implied part. The objective of a transform is to store knowledge about a particular pattern and what that pattern means. These patterns can be as complex as necessary, but the basic structure of the transform is always the same and very simple. The global part of a transform contains such information as the name of the transform, the size of the pattern being looked for and a threshold to be used in the TRANSFORMATION process. The lookfor part of a transform contains the pattern. This pattern is represented as an array of either numeric (integer) or symbolic (names) attributes. The objective of the TRANSFORMATION process is to find all occurrences of a transforms pattern in the processed image. The implied part of a TRANSFORM is a list of numeric and/or symbolic attributes. If the lookfor pattern is found, then this list of names will be implied into a layer by the TRANSFORMATION process.

2.3. Layers

The layers of a recognition cone hold the results of all of the TRANSFORMATIONS that are taking place. There can be any number of layers of any size in a recognition cone. Commonly, the layers are arranged in a sequence of decreasing size from the retina (most often 256 x 256) to the apex. The layers are always sequentially ordered. This arrangement of layers is what gives a cone appearance and hence gives rise to the name 'recognition cone'.

A layer consists of a two dimensional array of cells.

Each cell in a layer is a list of numeric and/or symbolic attributes. These attributes are placed in the cells of a layer by the TRANSFORMATION process when it has found a pattern, and consist of the implied list of the TRANSFORM whose lookfor pattern has been found. It is these cell lists which are examined when a lookfor pattern is being looked for during a TRANSFORMATION. Typically, a TRANSFORMATION will lookfor a pattern in one layer and put the implied list of the TRANSFORM in another layer. This action is what gives TRANSFORMS their name. The are used by the TRANSFORMATION proces to transform (change) the contents of one layer into a different pattern of attributes in another layer. It is through tthis TRANSFORMATION process that visual recognition takes place. With a well designed set of transforms, the TRANSFORMATION process will discover patterns which imply visually meaningful objects.

2.4. The Transformation Process

The TRANSFORMATION process consists of what is called 'applying' TRANSFORMS to the layers of a recognition cone. The transforms which can be applyed are of two type called numeric and symbolic. Numeric transforms are used on integer arrays like the kind obtained from the digitized input to the system. These transforms are used to convert the integer array into a symbolic or integer representation. Symbolic transforms are used on the symbolic attributes contained in the cells of a layer and convert symbolic patterns in one layer into other symbolic patterns in another layer. The action of these two types of transforms in the TRANSFORMATION process is discussed in more detail below.

2.4.1. Numeric Transformation

The ICON system has two types of predefined transforms called 'ratio' and 'absolute'. The ratio numeric transform is designed to locate patterns of relative intensities (values) in an integer array. Such transforms are useful in locating such things as gradients and spatial frequencies regardless of the absolute intensities of these patterns. The formula used in calculating the strength of the pattern being looked for as it exists in the actual integer array is the following.

$$W = \sum_{i=1}^{i=n} w_i * e^{\left|\frac{ev_i - tv_i}{max}\right| * A}$$

A - exponential decay constant
tv - transform lookfor value at location i in transform pattern
ev - expected integer array value at location i in window
max - maximum integer array value within window

w - weight at location i in transform window
n - size of the window

The window mentioned above is the portion of the integer array which is being looked at by a transform. This window will be the size of the transform being applied which can be anywhere from 1 x 1 up to 5 x 5. The formula above has the following properties.

[1] The weight contributed by a particular location of the transform (when matched against its particular location in the integer array) decreases exponentially as the difference between the actual retina value (av) and the expected retina value (ev) increases.

[2] The expected retina value (ev) is itself a function of the transform window applied at a particular point on the integer array, and is calculated with the following formula

$$ev_i = \left| \frac{\sum_{i=1}^{i=n} tv_i}{\sum_{i=1}^{i=n} av_i} \right| * av_i$$

tv - sum of all transform lookfor values

av - sum of all integer array values within the transform window

[3] The value of w will be multiplied by a term varying from between Ø and 1. This means that the strength of a particular transform pattern is estimated by a weight which is decreased exponentially the farther the transform pattern is from the actual integer array pattern.

The absolute numeric transform is designed to locate patterns of absolute intensities within an integer array. The formula used for calculating the strength of an absolute pattern is identical to that for calculating a ratio pattern except for the replacement of the ev term with av (the value of the transform at location i in the window). An example of the numeric transformation process is given in appendix I.

2.4.2. Symbolic Transformation

A symbolic transform is very similar to a numeric transform except that a pattern of symbols (names) is looked for instead of a pattern of integers. These symbols are looked for in a layer containing the symbolic attributes produced by other transformations. The matching process consists of searching a window in a layer for the attributes in the lookfor pattern of a transform. When an attribute is found in the layer, the weight of that attribute is added to a running total for the transform. If this running total exceeds the transform threshold (contained in the global portion of the transform) then the transform is said to have succeeded over the particular window in the layer. When a transform succeeds, the attributes in the transforms implied list are put into another layer of the recognition cone. For each transform in a recognition cone configuration, this process is repeated at every cell in the layer to which the transform is assigned. The symbolic pattern can represent virtually any kind of knowledge the user wishes. The discovery of these patterns in the layers of a recognition cone is what drives the recognition process.

SECTION III
TRANSFORM EDITOR

3.

3.1. Introduction

Several functions are performed by this program. The major functions are to create and maintain transforms and to develop a configuration for later interpretation or compiling. On top of this, the TRANSFORM EDITOR also initiates and interfaces with the RECOGNITION CONE INTERPRETER and the RECOGNITION CONE MONITOR program thus controling the functioning of the entire system. The following paragraphs describe each of these functions in detail.

3.2. Transform creation and maintenance

The creation and maintenance of transforms and their associated attributes is done by the TRANSFORM EDITOR using the TLIM routines (see APPENDIX G). These routines are invoked by the TRANSFORM EDITOR when they are requested by the user through the use of commands to the editor. The TLIM routines do the actual directory and transform file maintenance, the TRANSFORM EDITOR is designed to provide a friendly interface between the user and the TLIM routines. When a user executes the TRANSFORM EDITOR he has access to the full set of commands described below. These commands are designed to aide in developing transforms (ie the 'ADD' command) and to aide in cataloging transforms (ie the 'LOCATE' and 'DISPLAY' commands). Many of the commands provided cause the current transform and current attribute lists maintained by the TLIM routines to be updated. The user can, at any time, display the current contents of these lists (see DISPLAY command). APPENDIX G describes the current transform and current attribute lists and should be read before any of these commands are tried.

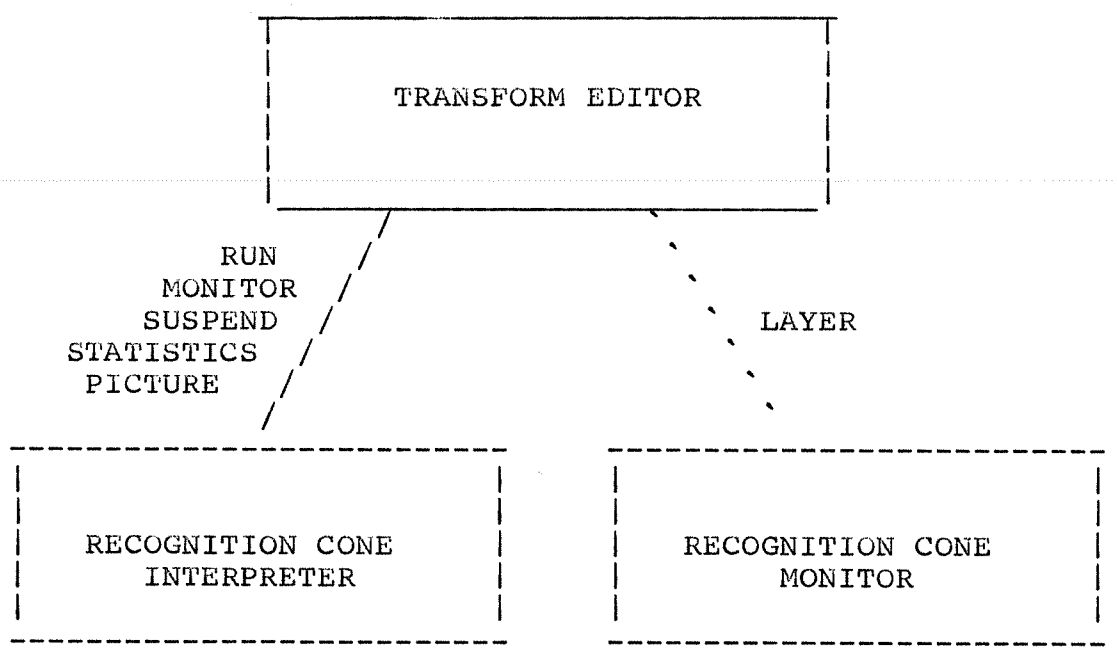
The transform editor itself always maintains a directory of the transforms which have been loaded from the transform library. This directory is empty when the program first gets executed and has transforms loaded into it when the user does a GET, LOAD or ADD command (see below). All commands that affect transforms (CHANGE,DELETE,SHOW) operate upon previously loaded transforms. Many commands (CREATE, DISPLAY, DESTROY, LINK, LOCATE, RESET, OPEN CLOSE, INITIALIZE, QUIT) do not effect the current loaded directory and may be entered no matter what it contains.

3.3. Configuration Development

One of the basic functions of the TRANSFORM EDITOR program is to specify the configurations of the recognition cones that the developed transforms are to be used with. A configuration consists of a specification of the layers and transforms involved in a recognition cone. The user can define an arbitrary number of layers (currently 20 maximum) of arbitrary sizes (currently 256 x 256 maximum) and describe which transforms are assigned to each layer. A user can name and describe as many configurations as he wishes. The configuration name is what distinguishes the specific recognition cone to the RECOGNITION CONE INTERPRETER. This name is given to the INTERPRETER when the user issues a RUN command. Although the user may describe and save as many configurations as he wishes, only one such configuration can be RUN at any one time. The commands which develop, maintain and display configurations are the CONFIGURE, SHOW, LOAD and SAVE commands. The LOAD and SAVE commands allow a user to save configurations across sessions.

3.4. Controlling the Interpreter and Display programs

The third major function of the TRANSFORM EDITOR program is to provide a means of initiating and controlling the other programs in the system. These programs communicate with the TRANSFORM EDITOR by means of messages passed between them. There are a number of commands which are designed to provide the interface with the other programs. The RUN, MONITOR, STATISTICS, END, SUSPEND, and LAYER commands deal solely with the control of these other programs. The messages passed between the programs are sent by means of a communications module which always runs in the background when the TRANSFORM EDITOR is running. The following diagram shows the communications interfaces between the three major programs in the system and the commands which effect each program.



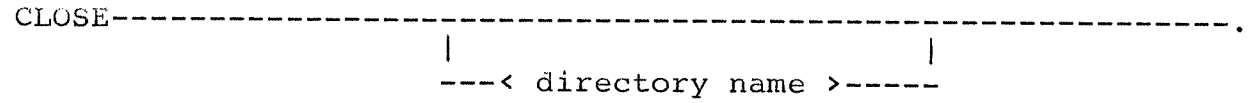
3.5. TRANSFORM EDITOR Commands

All commands are described using a railroad syntax. In order to determine how to use the command just follow one of the paths through the diagram for the particular command. The characters in parentheses are the command abbreviations which may be used in place of the commands. All commands may be entered in either upper or lower case. Words enclosed in '<>' represent a class of objects which must be replaced by the name of a particular object of the class. Words not enclosed must be entered the way they are shown. In all cases, keying RETURN will cause a prompt for the next input.

Any system command can be executed through the shell by having an exclamation ('!') be the first character of the input.

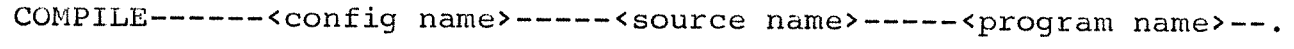
example: !ps

CLOSE(CL)



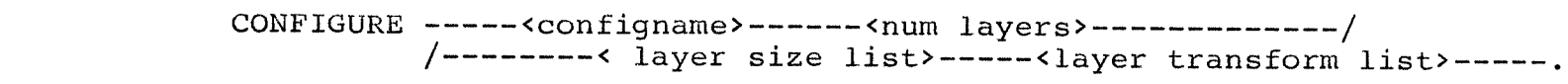
Closes the currently opened directory. If a name is given then the name must match the currently opened directory or it will not be closed.

COMPILE(CP)

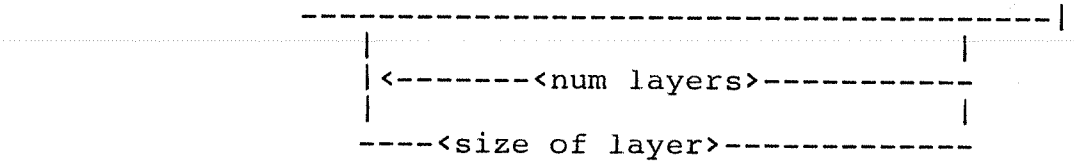


Initiates the parallel pascal code generation program which will produce a source file of parallel pascal code implementing the named configuration.

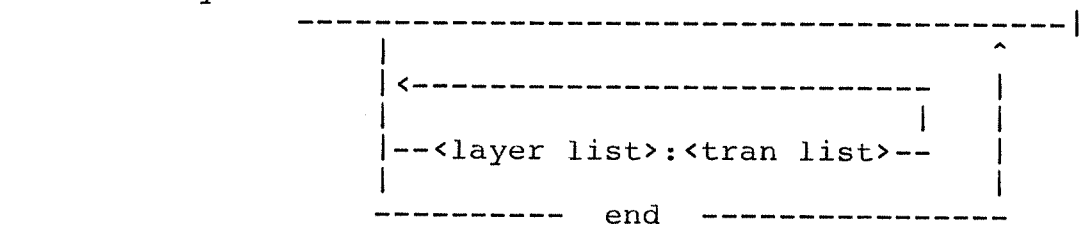
CONFIGURE(CF)



<layer size list> :

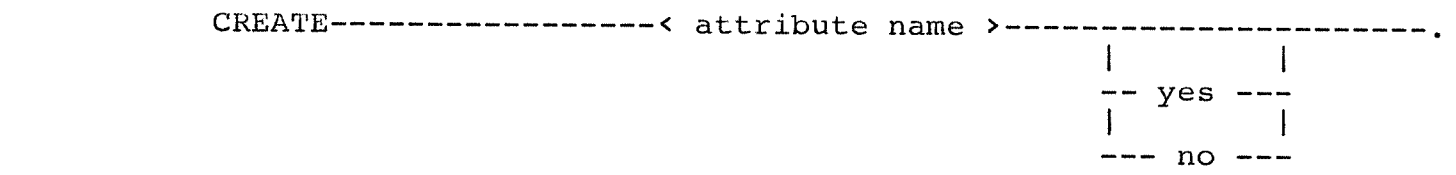


<layer transforms> :



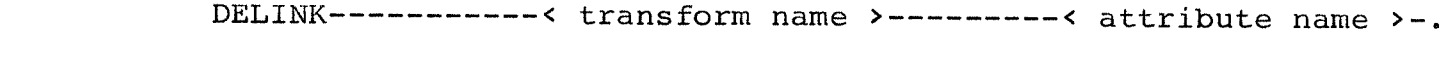
A cone configuration with the specified name is created. The configuration specifies the number and size of each layer along with the transforms assigned to each layer.

CREATE(CR)



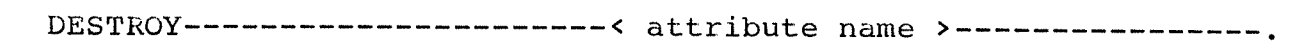
Creates a new attribute and adds it to the directory. The attribute can be made current by specifying 'yes'.

DELINK(DL)



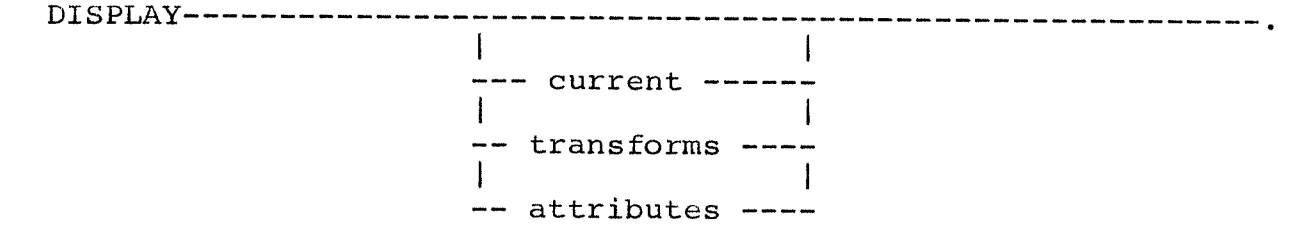
Removes the association between a transform and an attribute. This is implicitly done when a transform is deleted or an attribute is destroyed.

DESTROY(DS)



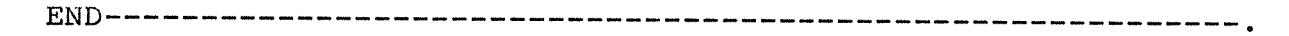
Deletes an attribute from the directory and from any transforms with which it is associated.

DISPLAY(DI)



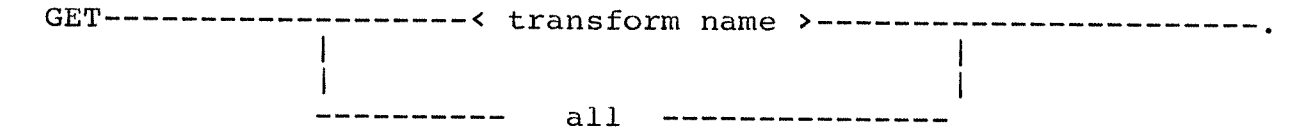
Displays the lists of current attributes and transforms or the set of all attributes or all transforms in the directory. The default parameter is 'current'.

END(ED)



Ends the currently executing recognition cone.

GET(GE)



Gets a transform from the current transform list and loads it into the directory. If 'all' is specified, all of the transforms having the current attributes are loded.

HELP(HE)

```
HELP-------------------------------------------------------.
                        |                 |
                        ---<command>------
```

Prints the list of available commands or the syntax and function of a particular command.

INITIALIZE(IT)

```
INITIALIZE----------< directory name >-------------------------.
```

Creates and initializes a transform directory with the given name. The directory must be opened in order to be used.

LAYER(LA)

```
LAYER----<layer number>-----------------------------------------.
                        |                 |  |                 |
                        -- on <device> ----  --<attribute list>-
```

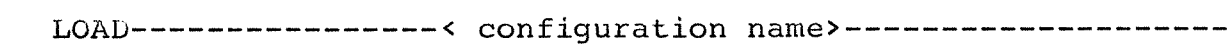
The specified layer is displayed on the specified device. This display is done by the RECOGNITION CONE MONITOR program as a separately running process. The allowed devices are 'term', 'terak', 'printer', 'tektronix', and 'versatec'. The default is 'term'. If an attribute list is specified the location and weights of the given attributes will be displayed. If no attribute list is given, the count of the existing attributes at each point in the layer will be given.

LINK(LI)

```
LINK------------< transform name >---------< attribute name >--.
```

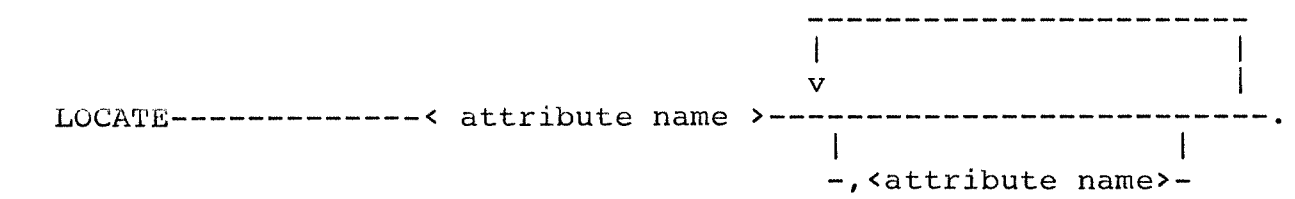
Associates an attribute with a transform. Both the attribute and the transform must have been created or added previously.

LOAD(LD)



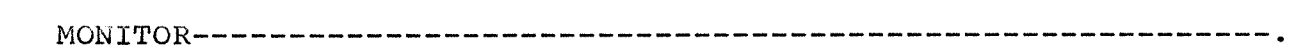
A previously saved configuration is loaded from the named file. All transforms in the file are loaded automatically and assigned to the layers they were assigned to at the time of the SAVE command.

LOCATE(LO)



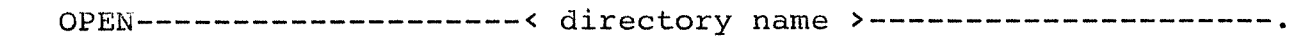
Locates the given attribute in the directory and updates the current transform list to reflect the changed attribute list.

MONITOR(MO)



The monitor command causes the TRANSFORM EDITOR to tell the currently executing configuration to start monitoring the contents of the layers as transforms are being applied. Each layer will be dumped once each cycle. These layer dumps permit the RECOGNITION CONE MONITOR program to go in and analyze the results of the transformations. (see LAYER command).

OPEN(OP)

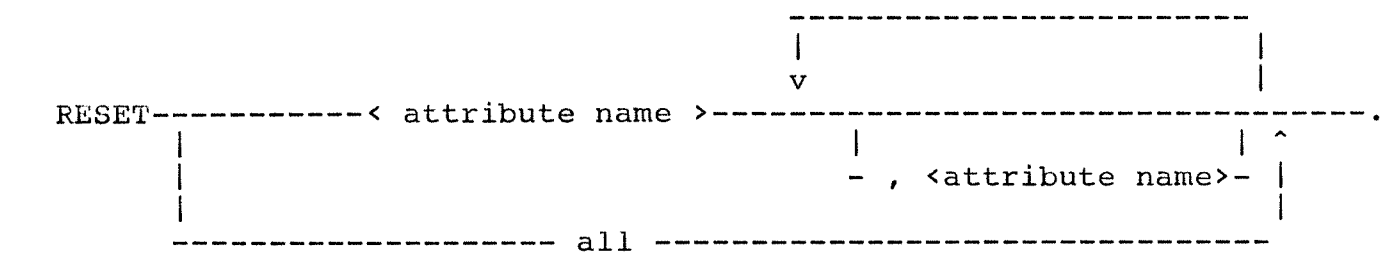


Opens the named directory and reads the directory file. Any changes made to transforms or attributes will not be made permanent until the directory is closed (see QUIT and CLOSE command).

QUIT(QU)

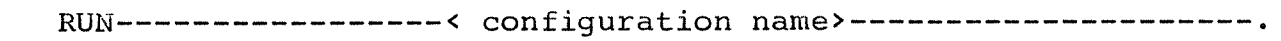
Stops the editor program and saves the directory and all changes.

RESET(RE)



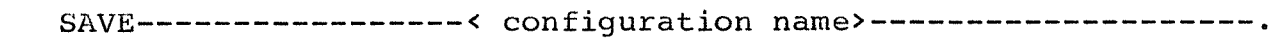
Takes an attribute (or all attributes) out of the current attribute list and adjusts the list of current transforms accordingly.

RUN(RU)



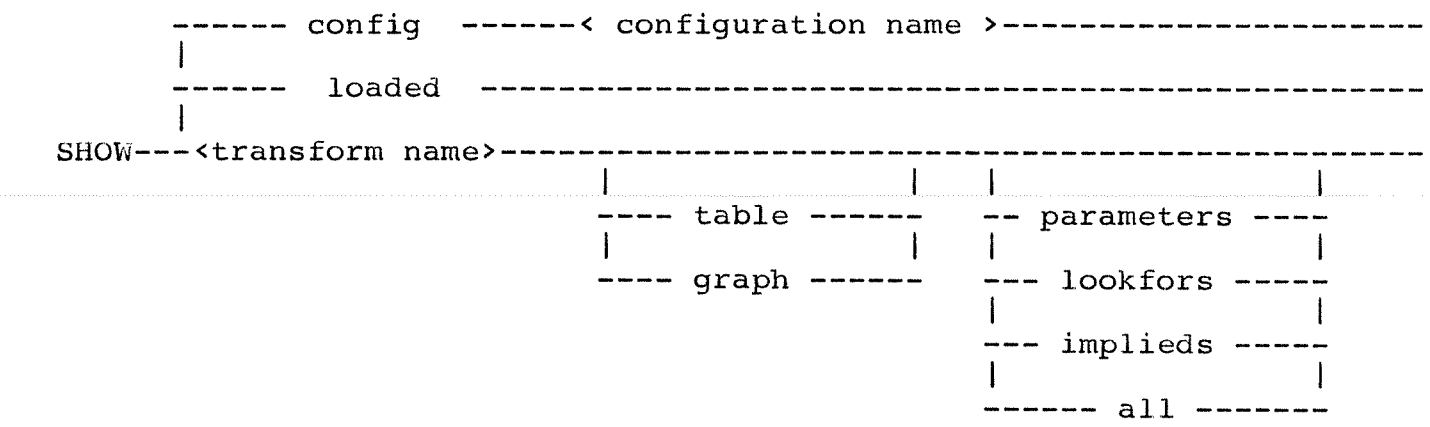
A recognition cone will be executed with the named configuration. The cone program will be a separate process executing independantly of the transform editor program.

SAVE(SV)



The current configuration is saved in the named file, if one is given, or in the default file CONFIG.DAT if no file is named. All existing loaded transforms, their active state, and the layers they are assigned to are saved so that they can be recreated at a later date with the LOAD command.

SHOW(SH)



Shows a particular transform in several formats, a particular configuration or the directory of loded transforms. The particular part of the transform to be shown can be selected with the second parameter. The first parameter indicates how the lookfors are to be displayed. The defaults are 'graph' and 'all'.

STATISTICS(SS)

STATISTICS----------.

Signals the RECOGNITION CONE INTERPRETER to start gathering statistics on the currently executing configuration.

STATUS(ST)

STATUS----------.

The current status of the currently executing configuration is displayed.

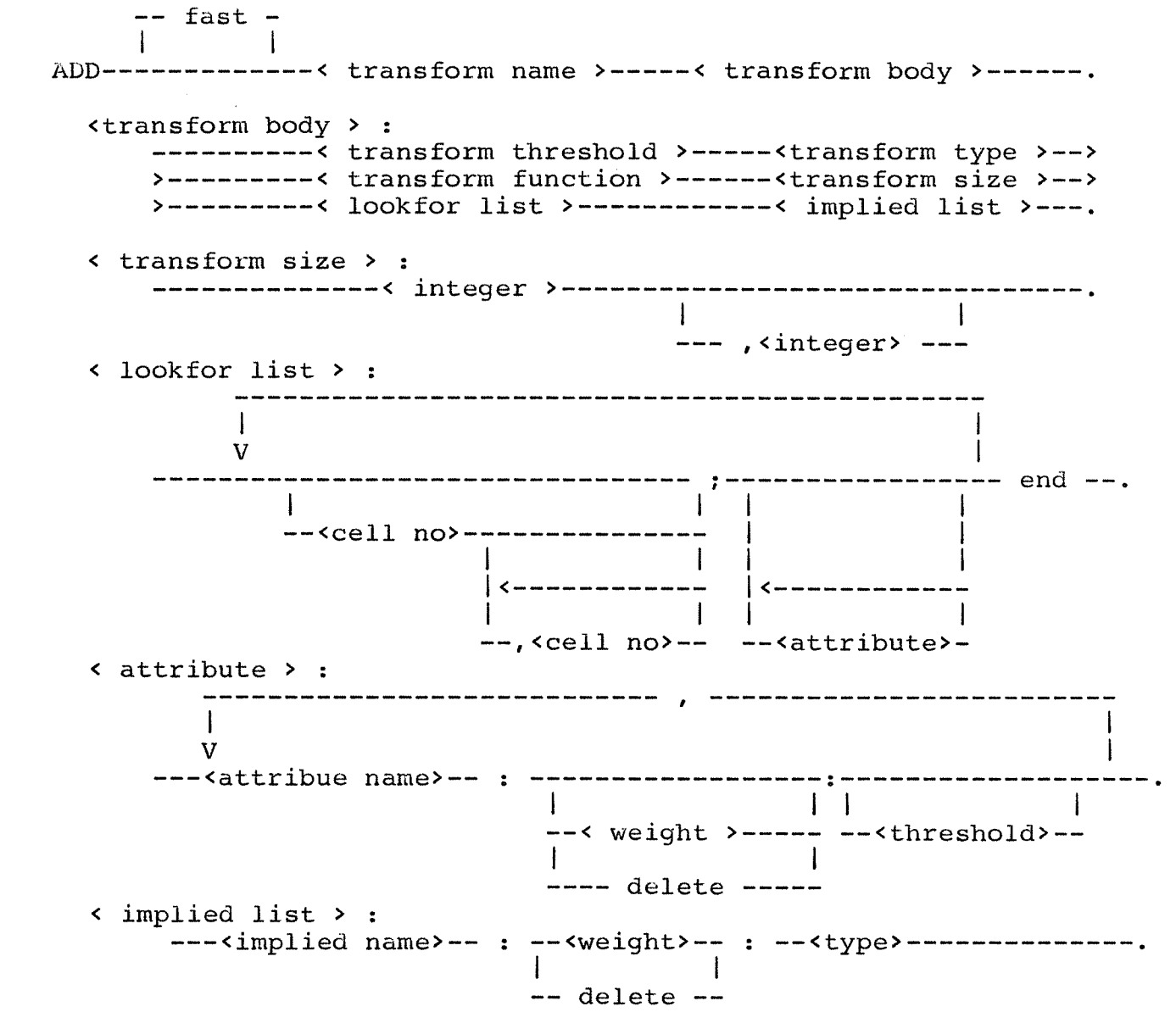
SUSPEND(SU)

SUSPEND----------.

The currently executing configuration is suspended at the next interruptable point.

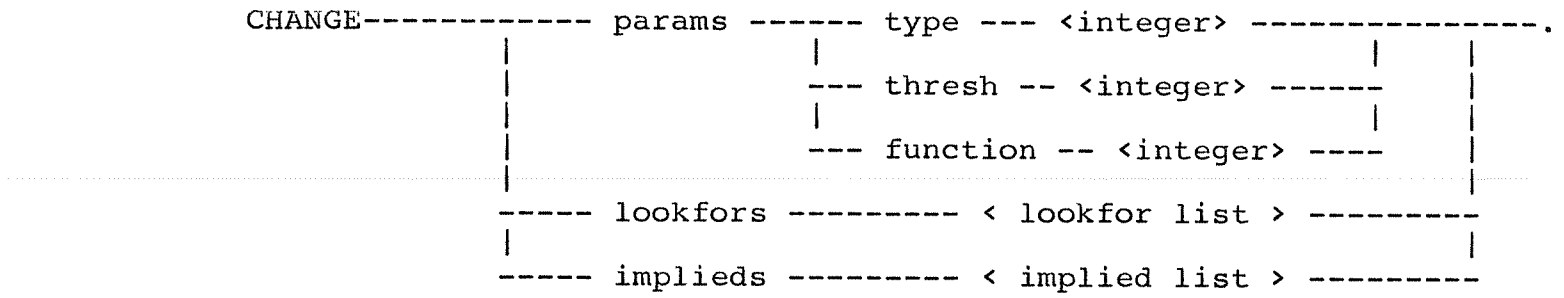
The following commands are used to add and update transforms. These functions are interactive in that the TRANSFORM EDITOR will prompt the user for any needed information. The user can also put in the information without being prompted by using the syntax described below.

ADD(AD)



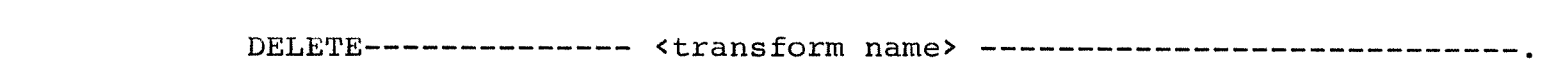
Add a transform to the system with the given attributes. A description of all of the fields of a transform is given in appendix G. The add fast form of this command will inhibit the display of intermediate stages of transform creation.

CHANGE(CH)



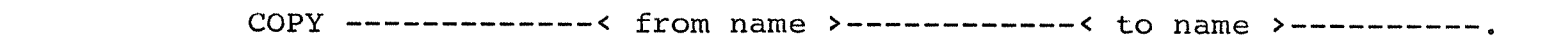
Allows one to add change or delete a transforms components. The transforms type, thresshold or function can be changed along with the lookfors and implieds. The changes to the transforms lookfors or implieds has the same format as that in the ADD command.

DELETE(DL)



Deletes the current transform from the directory file and removes the transform file so that it can no longer be accessed.

COPY(CO)



Copies a transform from one name to another. An exact copy is made and put into the transform directory.

SECTION IV
RECOGNITION CONE INTERPRETER

4.

4.1. Introduction

The RECOGNITION CONE INTERPRETER program allows the user to test his designs in a simple interactive manner to see if they are valid. The recognition cone configurations developed by the user using the TRANSFORM EDITOR program are interpreted by this program resulting in the desired recognition process. The interpretive functioning of this program means that it is slow, however great flexibility is provided to the user in monitoring the functioning of the cone and in the displaying of the results. In this manner a user can see the effectiveness of the transforms he designs and rapidly modify his transforms to achieve better success.

4.2. Operation

The program itself is initiated by the TRANSFORM EDITOR when a user enters a RUN command. The program gets the name of the transform directory and the configuration file from the TRANSFORM EDITOR and sets up the desired configuration. This configuration is then executed until the user enters and END command to the TRANSFORM EDITOR at which time it tells this program to stop executing.

The operation of this program can be thought of as a series of cycles through the cone layers starting at the retina. At the beginning of each cycle an image must be supplied to the cone. This is done by the PICTURE command in the TRANSFORM EDITOR. The name of the picture must be the name of an image file produced by the PICT software. This image should match the size of the retina in the configuration being interpreted and should be scaled to have intensity values in the range of 1 to 64. The user can either provide one picture at a time to the interpreter, or he can set up a sequence of pictures which are provided when the interpreter requests them. During each cycle, the transforms which are assigned to each layer are applied to these layers. Before any transform is applied to a layer, the interpreter checks if it has received any messages from the TRANSFORM EDITOR. The user can thus turn on and off the statistics gathering and monitoring functions of the RECOGNITION CONE INTERPRETER during its execution. The interpreter can be instructed to dump the contents of any layer or its current executing status into a file for analysis and display by the RECOGNITION CONE MONITOR program. After dumping the requested information the interpreter continues executing,letting the user examine the results at his leisure. The user requests results to be dumped and displayed through the MONITOR, STATUS and LAYER commands of the

TRANSFORM EDITOR.

SECTION V
RECOGNITION CONE MONITOR

5.

5.1. Introduction

Results of the execution of the RECOGNITION CONE INTERPRETER are displayed by this program. These results are requested by the TRANSFORM EDITOR when a user types in a LAYER command. This request is given to the RECOGNITION CONE MONITOR program which then reads the correct monitor file produced by the RECOGNITION CONE INTERPRETER and displays the information requested by the user with the LAYER command. Once the information has been displayed, control is returned to the TRANSFORM EDITOR.

5.2. Operation

The file produced by the RECOGNITION CONE INTERPRETER contains the raw layer or global data representing the current status of the interpretation. This data is analyzed by the RECOGNITION CONE MONITOR program to produce meaningful displays on a variety of devices. There are several display devices currently available for display of both graphic and symbolic information produced by the interpreter. Both the display device and the mode of representation (graphic or symbolic) are specified by the user with the LAYER command. The display devices available are the users terminal, the tektronix graphics terminal, the terak or the line printer. The users terminal can be used as either a symbolic or graphic device. The terak and the tektronix terminals can be used only as graphic devices and the line printer can be used only as a symbolic device.

There agraphic devices and the line printer can be used only as a symbolic device.

There are several different types of display which can be produced by the RECOGNITION CONE MONITOR program. In general, the type of display is determined by the choice of symbolic or graphic representation and by the presence of an attribute list in the display request (see LAYER command). For symbolic display, if no attribute list is given, the display program will list, for every row and column position in the layer, the attributes currently occupying that position. If an attribute list is given, the display program will list, for every attribute chosen, the locations it occupies within the chosen layer. Thus for symbolic display mode there exist the following two types of displays.

attribute list present

<attribute name> : <loc 1> <loc 2> . . .

```
<attribute name> :  <loc 1> <loc 2> . . .
                          .
                          .
                          .
```

attribute list not present

```
<loc 1>   : <attr 1> <attr 2> . . .
<loc 2>   : <attr 1> <attr 2> . . .
                     .
                     .
                     .
```

Graphic displays also depend on the presence or absence of an attribute list. When an attribute list is present, a display is produced which represents the sum of the weights of the named attributes at each point in the layer. When the attribute list is not present, the display produced represents the strict count of the number of attributes existing at each point in the layer. In both these cases, the sum of the weights or the count of the atributes, the integer obtained at each point in the layer is used as a direct indication of the intensity of the display at the point. More intense portions of the display thus indicate higher weights or greater numbers of attributes at the point on the display which represents a location in the layer.

SECTION VI
RECOGNITION CONE STATISTICS

6.

6.1. Introduction

The operation of a recognition cone depends upon many subtle influences in the design of the transforms used. The RECOGNITION CONE STATISTICS program is designed to take some of the guesswork out of transform design by providing information about the effectiveness of particular transforms in the various vision processes they are involved in. This program will produce three reports from the information gathered during the interpretation of a particular configuration when a user issues the STATISTICS command via the TRANSFORM EDITOR. These reports can be requested in a number of ways, by the image name , by the layer number, by the transform name or by any combination of these three factors. The three reports produced are described in the following sections.

6.2. Transform Hit Report

This report details the number of times a particular transform 'succeeded' in a particular layer for a particular image. The report will look like the following.

TRANSFORM HIT REPORT

IMAGE = <image name>
LAYER = <layer number>

TRANSFORM NAME	NUMBER OF HITS
<transform name>	<number of hits>

This report is useful in performing a gross culling of the transforms which might or might not be effective, in determining those images for which more transforms need to be developed or in determining a more effective assignment of transforms to layers.

6.3. Individual Transform Report

This report contains detailed information about the action of a particular transform. The format of the report is the following.

INDIVIDUAL TRANSFORM REPORT

TRANSFORM NAME = <transform name>

HITS PER LAYER

LAYER NUMBER	NUMBER OF HITS
<layer number>	<number of hits>
.	.
.	.
.	.

HITS PER IMAGE

IMAGE NAME	NUMBER OF HITS
<image name>	<number of hits>
.	.
.	.
.	.

$$\text{LOOKFOR CONTRIBUTION} = \frac{\text{\# HITS WHERE THIS LOOKFOR CONTRIBUTED}}{\text{\# HITS FOR THIS TRANSFORM}}$$

LOOKFOR NAME	PERCENTAGE CONTRIBUTION
<lookfor name> FOR IMAGE <name>	TOTAL = <contribution>
.	<contribution>
.	.
.	.
	.

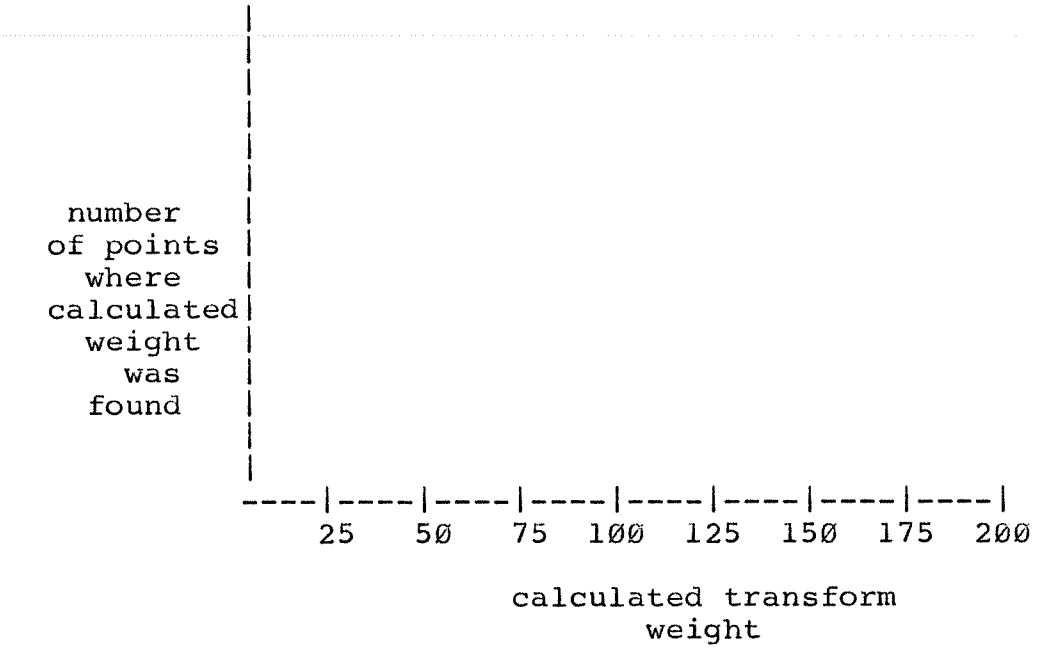
The Individual Transform Report is useful in determining if the transform design is good for the task it is used for. The lookfor contribution can be used to analyze the lookfor pattern to make it more effective.

6.4. Transform Threshold Histogram

This histogram is used to determine the most effective value for a particular transforms threshold. The report has the following form.

TRANSFORM THRESHOLD HISTOGRAM

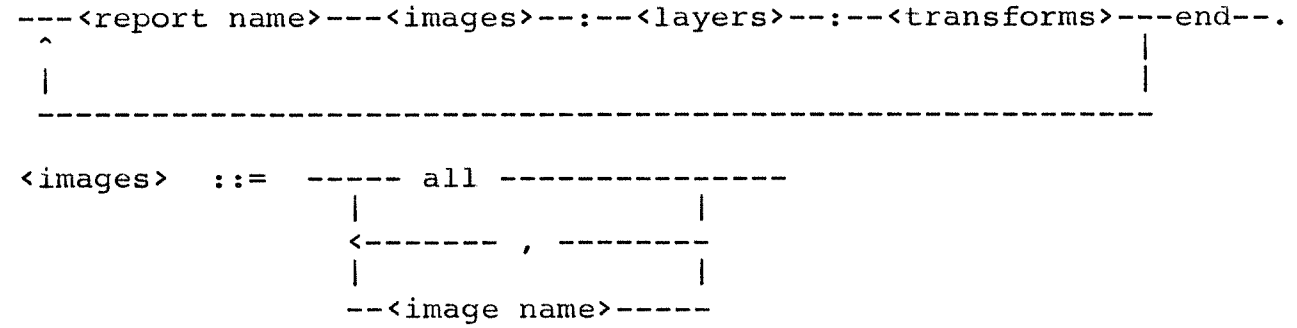
TRANSFORM NAME = <transform name>
IMAGE NAME = <image name>
LAYER NUMBER = <layer number>

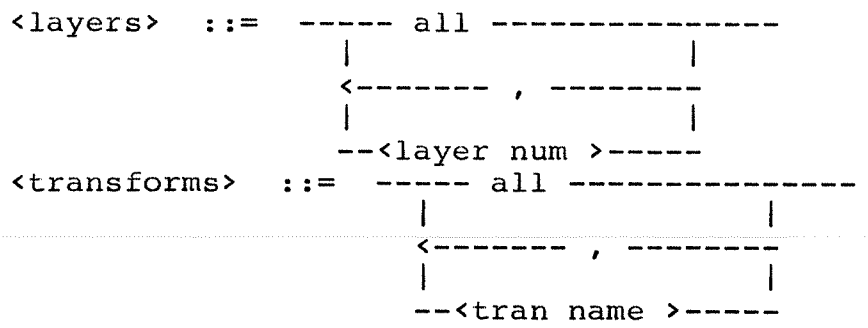


With this histogram the user can see the distribution of the calculated transform weights for a particular image in a particular layer. The transform threshold value which is most effective can be determined from this histogram

6.5. Running the Program

The RECOGNITION CONE STATISTICS program is called 'conestats'. To use it the user merely executes the program and informs it of the reports desired. This is done by entering the images,layers, transforms and reports to be used. The command format for doing this is shown below.




<layers> ::= ----- all ----------
| |
<------- , --------
| |
--<layer num >-----
<transforms> ::= ----- all ----------
| |
<-------- , --------
| |
--<tran name >-----

SECTION VII
RECOGNITION CONE COMPILER

7. Introduction

The cone compiler is designed to take a configuration the user has designed and create a specific pascal program for the execution of that configuration. This results in a program which is specifically designed with a particular layer configuration and a particular set of transforms. The purpose of compiling the configuration is to provide a significant increase in execution speed over the interpreted approach. The cone compiler is meant to be invoked after the user has 'debugged' his configuration with the RECOGNITION CONE INTERPRETER.

7.1. Operation

The recognition cone compiler is initiated with the COMPILE command to the TRANSFORM EDITOR. The compiler is given, as arguments, four names. These names are the name of the configuration file, the name of the transform directory to be used, the name of the source file which is to be produced by the compiler, and the name which is to be given to the program when it is compiled. The compiler uses the configuration file to determine the number of layers in the desired recognition cone, and the transforms which are assigned to each of these layers. The specified transforms are then read from the transform library and parallel pascal code is generated for each transform in each layer. Routines are also generated to execute the layer and transform procedures and to provide input into the retina layer. A sample configuration file and the resultant parallel pascal code generated by the compiler is given in appendix E.

When the parallel pascal code has been generated, it is put into the file named when the compiler was executed. This file can then be used as input to the parallel pascal preprocessor which will convert the parallel pascal constructs to standard pascal. This preprocessor is called parpas and must be invoked by the user after the parallel pascal code has been generated by the cone compiler. Detailed instructions for using the parpas precompiler can be found in (Uhr 1979).

SECTION VIII
OPERATING INSTRUCTIONS UNDER VAX/UNIX

8. Introduction

The TRANSFORM EDITOR program is the initiator and driver of all of the other pieces of software which make up this system. The TRANSFORM EDITOR program is initiated by typing in the command

icon

This invokes a shell file which will then execute both the TRANSFORM EDITOR program and the communications module. In order to effectively use the system, the user must be familiar with both the TRANSFORM EDITOR and PICT commands. The user must use PICT to retrieve, manipulate and supply images to the RECOGNITION CONE INTERPRETER. This is done by calling PICT using the exclamation (ie. !pict) getting the image you want, rescaling it to a range of 1-64, resizing it to the correct size of the retina, and storing it in a disk file. The TRANSFORM EDITOR command PICTURE can then be used to pass this picture to the currently running RECOGNITION CONE INTERPRETER.

When the TRANSFORM EDITOR is initiated and a transform directory is opened, this directory will reside in the directory the user was in when he invoked the program. All transforms which are created will reside in files within this same directory. There is currently no provision for accessing transform directories or transform files which reside in another Unix directory. This means that a user who wishes to use the transforms created by another person in another directory must sign on as that person or copy them into his own directory.

APPENDIX A
TRANSFORM DIRECTORY FILE FORMAT

The directory file is kept on disk and read in to memory by a TLIM routine when the opentransforms procedure is called. The directory contains the names of all transforms and attributes and the associations of each transfform to its attributes. Several flags indicate various conditions in the file. The meaning of these flags is described after the diagram.

Directory File

```
                                  tlibfile
<name>    <flag>
<name>    <flag>
      .
      .
      .
<name>    <flag>
$$$$$$$$$$
<attribute name>   <transform name>
<attribute name>   <transform name>
                  .
                  .
                  .
<attribute name>   <transform name>
$$$$$$$$$$
EOF
```

The first part of the file (before the first '$$$$$$$$$$') contains a list of names. These names are either attribute names or transform names. The flag indicates if the name is an attribute or a transform. A flag of Ø means the name is an attribute name, a flag of 1 means the name is a transform name.

APPENDIX B
TRANSFORM FILE FORMAT

A Transform file is stored as a sequence of strings and integers representing the various parts of the transform. The flags put in as part of the file signal things such as the start and end of the lookfor and implied lists as well as the lookfor type. These flags are described after the diagram.

Figure 3
Transform File

```
<transform name> <threshold> <type> <function> <refcell> <rowsize> <cc
   <lookfor list flag>
   <x disp> <ydisp> <dist 1> <dist 2> <dist 3> <dist 4>
   <lookfor type flag>
      <attribute list flag>
      <attribute present flag>
      <attribute name> <weight> <threshold>
      <attribute present flag>
      <attribute name> <weight> <threhold>
           .
           .
           .
      <attribute list flag>
   <xdisp> <ydisp> <dist 1> <dist 2> <dist 3> <dist 4>
   <lookfor type flag>
                .
                .
                .
   <lookfor list flag>
   <implied present flag>
   <implied name> <weight> <factor> <type> <xdisp> <ydisp> <function>
   <implied present flag>
   <implied name> <weight> <factor> <type> <xdisp> <ydisp> <function>
            .
            .
            .
   <implied present flag>
   EOF
```

The lookfor list flag is 1 at the beginning of the lookfor list and Ø at the end. The lookfor type flag is Ø for an attributelookfor and 1 for a lookforlookfor. The attribute list flag marks the end of the attribute list and is always Ø. The attribute present flag is 1 if another attriibute remains and Ø if there are no more attributes. The implied present flag is 1 if another implied remains and Ø otherwise. A sample transform file is shown below.

Figure 4
Sample Transform File

```
tranl          40    1   1   5   3   3
1
 Ø   Ø   Ø   Ø   Ø   Ø
Ø
1
sky             25    5
1
cloud           1Ø    3
Ø
 -1   -1   Ø   Ø   Ø   Ø
Ø
1
tree            15    4
Ø
Ø
1
outdoor         4Ø    Ø    Ø    Ø    Ø    Ø
1
forest          3Ø    Ø    Ø    Ø    Ø    Ø
Ø
EOF
```

APPENDIX C
CONFIGURATION FILE FORMAT

The configuration file contains the configurations developed by the user with the CONFIGURE command. These files are created by the TRANSFORM EDITOR program and are used by the RECOGNITION CONE INTERPRETER to set up its cone. The following diagram shows the format of this file.

Figure 5
Configuration File

```
configfile
<number of layers>
<layer number>    <layer size>
      <layer transform name>
                    .
                    .
                    .
      ##########
<layer number>    <layer size>
      <layer transform name>
      ##########

     .
     .
     .
```

APPENDIX D
STATISTICS FILE FORMAT

```
#<image name>
  *<layer number>
  $<transform name>  <transform threshold>
        <threshold bucket count>  .  .  .
     %<attribute name>  <hit count>
              .
              .
              .
  $<transform name >  <transform threshold>
       .
       .
       .
     EOF
```

<threshold bucket count> - buckets are from Ø-2ØØ in increments of 5. The count denotes the number of times the transform would have succeeded if the threshold was in the bucket range.

<hit count> - This count denotes the number of times the attribute was found when the transform was successful.

APPENDIX E
INTERPROGRAM COMMUNICATION

MESSAGES

TRANSFORM EDITOR ----> RECOGNITION CONE INTERPRETER

command	message
RUN	<configuration file name> <transform directory n
PICTURE	tranedit picture <picture name>
SUSPEND	tranedit suspend
STATISTICS	tranedit statistics
MONITOR	tranedit monitor

TRANSFORM EDITOR ----> RECOGNITION CONE MONITOR

command	message
LAYER	tranedit layer <layer number> <device> <attribute

FILES

STATUS FILE

```
<current cycle>
<current image name>
<current layer>
<current transform name>
<current status>   - WAITING PICTURE,SUSPENDED,APPLYING TRANSF
      EOF
```

LAYER FILE

```
<layer number>  <layer size>
<x coord>  <y coord>
      <attribute name>
              .
              .
              .
 <x coord>  <y coord>
      .
      .
      .
         EOF
```

APPENDIX F
TRANSFORM LIBRARIES

9.

A Transform Library consists of a directory file and a set of transform files. The Transform Library Interface Module (see section III) contains routines to create and maintain these files. The TRANSFORM EDITOR program (see section IV) provides an interactive interface to the TLIM routines and thus to the Transform Library. The Vision program uses the Transform Library to get the transforms necessary for processing.

9.1. DIRECTORIES

A directory file contains the names of all transforms in the Library, the names of all attributes which have been created (see Editor program documentation) and a list of transform-attribute pairs which specify the attributes that have been associated with each transform. A transform can have any number of attributes associated with it. These attributes can convey any information which the designer of the transform wishes. For example, a transform designed to 'recognize' a green leaf might have the following attributes

```
transform: grnleaf
attributes: myname,test,outdoor,highlevel,tree,leaf
```

signifying that this is a transform in the test stage belonging to 'myname' for outdoor scenes etc.

It is through the use of these attributes that a user can browse through the transform library, creating or selecting transforms which he wishes to use. Attributes should be created so that they are meaningful to the user and also to other persons who might use the transform. It is the attribute list associated with the transform which can define what the transform can be used for.

A directory file will contain all transform names and the transform-attribute associations which have been defined by the user. There will be only one directory file per Transform Library, although a user can create as many libraries as he wishes. Appendix A shows the format of the directory file.

9.2. TRANSFORMS

A transform in a Transform Library is kept as a disk file, one transform per file. The name of the file is the same as the name given to the transform by the user, therefor transform names must be unique in the first eight (8) characters. Transform files are created and maintained by routines in the TLIM. TLIM routines also read transform files into memory for use by the Editor and Vision programs. Appendix B shows the format of a Transform file.

APPENGIX G
TRANSFORM LIBRARY INTERFACE MODULE

10.

10.1. Introduction

The Transform Library Interface Module (TLIM) contains routines to create , maintain and search a transform library. A user interfaces to a transform library by using the appropriate routines from the TLIM. Through calls to these routines, a user can generally manipulate transforms and attributes of the transform in any way he wishes. A user should avoid updating the transform or directory files directly by always using the routines provided in the TLIM. The following sections describe the operation of the routines in the TLIM. Sections 3 and 4 can be skipped by those users who do not wish to write their own programs utilizing these routines. Section 2 should be read by everyone.

10.2. Current Attribute and Transform Lists

From the time a transform library is opened (see opentransforms procedure) until it is closed (see closetransforms procedure) the TLIM routines maintain two lists called the Current Attribute list and the Current Transform list. Attributes are put into the Current Attribute list by the locateattribute function and taken out of the list by the backup function. Because transforms are located via their attributes, the current transform list is automatically updated any time the current attribute list changes. There are no specific routines to manipulate the current transform list.

At any point in time, the transforms in the current transform list all share the attributes in the current attribute list. This means that each transform in the current transform list has associated with it (in the directory file) at least all of the attributes in the current attribute list (it may have more attributes associated with it than are in the list but never less). Take as an example the following transforms and their associated attributes.

```
transform : tranl
attributes: test,retina,myname,brite

transform : tran2
```

```
attributes: test,layer1,myname,longline

transform : tran3
attributes: layer1,brite,region
```

If the current attribute list contained just the attribute 'test' then the current transform list would be ;

```
current attributes: test
current transforms: tran1,tran2
```

since both of these transforms share the attribute 'test'. If, however, the attribute 'brite' was added to the current attribute list (by calling the locateattribute procedure) then the current transform list would become ;

```
current attributes: test,brite
current transforms: tran1
```

(note: the modification of the current transform list happens automatically and is beyond the control of the user of the routines)

As another example, suppose the user removed the attribute 'test' from the current attribute list by using the backup function. The current transform list would then look like ;

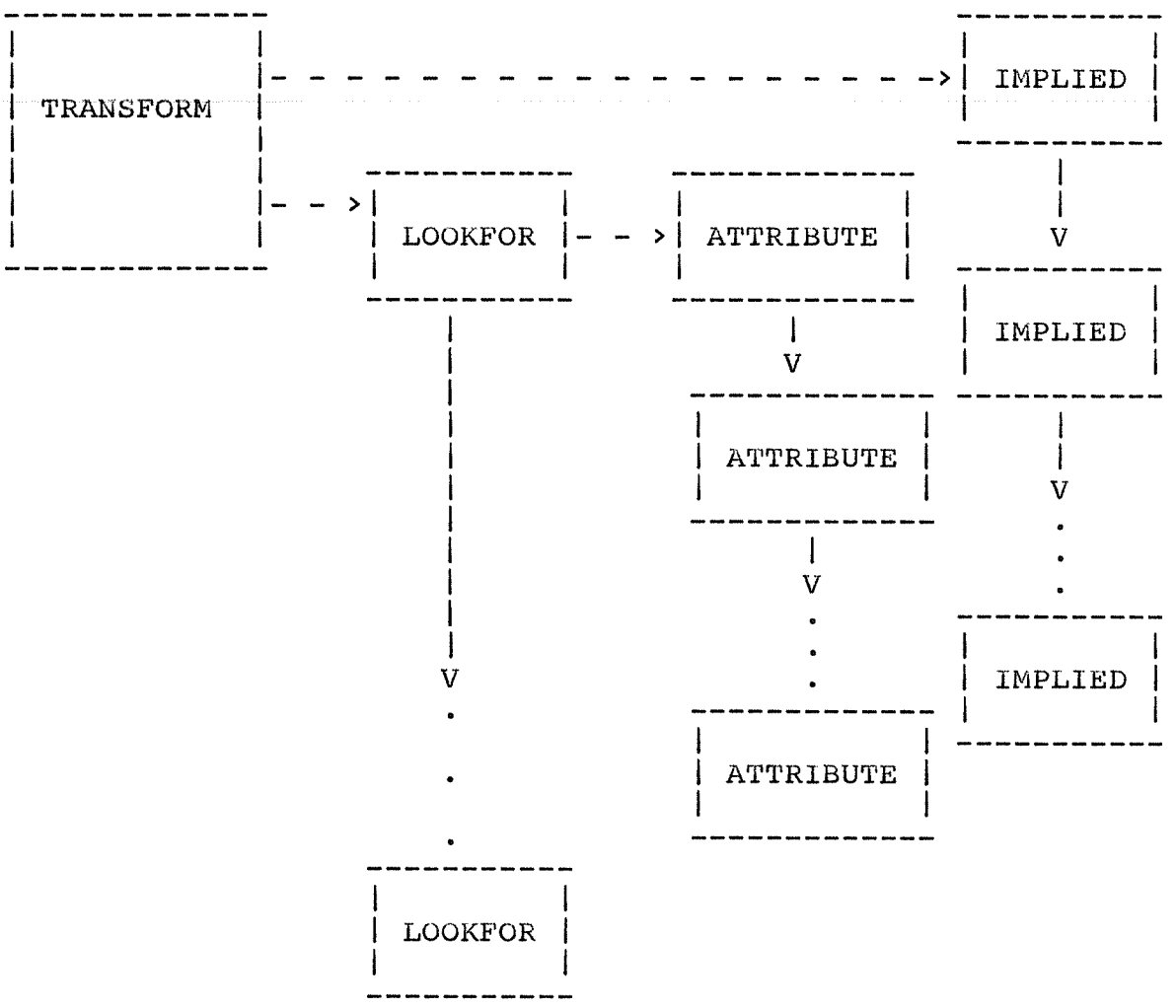
```
current attributes: brite
current transforms: tran1,tran3
```

It can be seen from these examples that the list of current transforms will contain all existing transforms which have all of the attributes in the current attribute list as a subset of the set of attributes associated with them in the directory. Many of the TLIM procedures may affect the contents of the current transform list by either updating the current attribute list or changing an association between a transform and an attribute (see link and delink procedures). In the discussion of the routines below, it is noted when the routine may cause a change in the current transform or attribute lists.

10.3. Interface Area

The main objective of the TLIM routines is to allow users to retrieve and store the transforms they desire. Transforms are maintained in memory as a data structure consisting of four different pascal record types. When a transform is to be retrieved or stored (see gettransform and puttransform functions) a pointer to the transform memory data structure is passed between the calling program and the TLIM routine. When a transform is requested by the calling program, the TLIM routine gettransform initializes the data structure and fills in the data from the transform disk file. When a transform is to be stored by the TLIM routine puttransform, it is the responsibility of the calling program to pass the pointer to a valid transform data structure. A transform memory data structure can be diagramed as follows.

Transform Memory Data Structure



The following are the pascal record definitions which implement this structure. These record definitions must be included in any program wishing to interface with the TLIM routines.

```
{*************************************************************}
{                        INTERFACE AREA                       }
{*************************************************************}
   transformptr              = ^transformrecord;
   lookforptr                = ^lookforrecord;
   impliedptr                = ^impliedrecord
   attributeptr              = ^attributerecord;
{*************************************************************}
{                        TRANSFORM RECORD                     }
{*************************************************************}
```

```
   transformrecord = record
      transformname                  : string;
      transformthreshold             : integer;
      transformtype                  : integer;
      transformfunction              : integer;
      transformrefcell               : integer;
      transformrowsize               : integer;
      transformcolsize               : integer;
      centerlookfor                  : lookforptr;
      implieds                       : impliedptr;
   end { transform record };
{*****************************************************************}
{                          LOOKFOR RECORD                         }
{*****************************************************************}
   lookforrecord = record
      lookfornext                    : lookforptr;
      lookxdisp,lookydisp            : integer;
      lookdist                       : array [1..4] of integer;
      case lookfortag : integer of
           ATTRIBUTELOOKFOR          : (attributelist : attributeptr);
           LOOKFORLOOKFOR            : (lookforlist : lookforptr);
   end { lookfor record };
{*****************************************************************}
{                        ATTRIBUTE RECORD                         }
{*****************************************************************}
   attributerecord = record
      attributenext                  : attributeptr;
      attributename                  : string;
      attributeweight                : integer;
      attributethresh                : integer;
   end { attribute record };
{*****************************************************************}
{                         IMPLIED RECORD                          }
{*****************************************************************}
   impliedrecord = record
      impliednext                    : impliedptr;
      impliedweight                  : integer;
      impliedfactor                  : integer;
      impliedtype                    : integer;
      impliedname                    : string;
      impliedxdisp,impliedydisp      : integer;
      impliedfcn                     : integer;
   end { implied record };
```

These record definitions are in a file called 'interecs.i' which can be included in a pascal source program. Several of the fields of these records can have values which are predefined (ie they have special significance to the Vision program described later). The field and the predefined values are described below. These constants are in the file 'tranconst.i' and should be included in the constant section of any program using the TLIM routines.

```
{ values for field lookfortag in lookfor record }
   ATTRIBUTELOOKFOR          = 1;
   LOOKFORLOOKFOR            = 2;
{ values for field impliedtype in implied record }
   ATTRIBUTEIMP              = 1;
   TRANSFORMIMP              = 2;
{ values for field transformtype in transform record }
   RETINATRAN                = 1;
   LAYERTRAN                 = 2;
{ values for field transformfunction in trasform record }
   RATIOFCN                  = 1;
   ABSOLUTEFCN               = 2;
```

In order to store or retrieve transforms using the TLIM routines the user must use a pointer such as the following.

```
transformptr    = ^transformrecord
```

10.4. Procedures

The following TLIM routines are ones that can be called from a user program. Each procedure is listed along with its formal parameters as it would appear in a pascal program.

```
function backup(attr : string) : boolean;
```

Removes the specified attribute name from the current attribute list.

```
function changetransform(name : string;trans : transformptr) : boolean;
```

Changes the named transform to be what is pointed to by the transform pointer without modifying the current transform or attribute lists

```
procedure closetransforms;
```

Closes the transform directory after writing out to it all updated information.

```
function deleteattribute(attr : string) : boolean;
```

Removes the named attribute from the directory as well as all of its links to transforms. The attribute is unavailable for location of transforms.

```
function deletetranattr(tran,attr : string) : boolean;
```

Removes the link between an attribute and a transform and updates the current transform list if necessary.

```
function deletetransform(name : string) : boolean;
```

Removes the named transform from the transform directory and deletes the transform file the transform is stored in.

```
function errortype : integer;
```

Returns an integer indicating the type of error encountered. Always returns an error or a Ø indicating last call did not produce an error.

```
function getdirectory(var ptype : integer;var pname : string) : boolean;
```

This procedure will, upon repeated calls, return all attribute and trnasform names in the transform directory. This function returns false when there are no more names left.

```
function getname(var name : string;t : integer) : boolean;
```

This procudure will, upon repeated calls, return the contents of the current attribute list (if t = Ø) or the current transform list (if t = 1) in name. The function will become false when the last name in the list is returned.

```
function getattribute(var name : string;tran : string) : boolean;
```

This procedure will, upon repeated calls, return all attributes associated with the named transform tran. The names are returned in name. The function will be false when the last attribute name is returned.

```
function getcurtransform(name : string) : transformptr;
```

A pointer to the memory structure of the named transform will be returned. This function will return nil if the named transform is not in the current transform list.

```
function getanytransform(name : string) : transformptr;
```

A pointer to the memory structure of the named transform is returned. The transform does not have to be in the current transform list. A nil is returned is the transform is not in the directory.

```
procedure inittransforms(name : string);
```

A transform directory with the given name is initialized.

```
function locateattribute(attr : string) : boolean;
```

The named attribute is found and the current attribute and current transform lists are updated appropriately. False is returned if the attribute daoes not exist in the directory.

function opentransforms(name : string) : boolean;

Opens the named directory. This procedure must be called before any other procedures.

function putattribute(attr : string) : boolean;

Puts a new attribute into the directory.

function puttranattr(tran,attr : string) : boolean;

Forms a link between the named attribute and the named transform.

function puttransform(name : string;trans : transformptr) : boolean;

Inserts the named transform into the transform directory and creates the transform file to store the transform.

procedure resetattributes;

Removes all attributes from the current attribute list.

procedure resettransforms;

Removes all transforms from the current transform list.

10.5. Use of Interface Module

In order to use the TLIM routines the user must include several source files in his program. These files provide the necessary constant,type,variable and external definitions necessary to interface with the TLIM routines. The following files should be included in the appropriate sections of the user program.

```
const
%include 'errconst.i'    {constants for error flags }
%include 'tranconst.i'   {constants for interface records}

type
%include 'intetype.i'    {types for interface records }
%include 'interecs.i'    {interface record descriptions}

var
```

```
%include 'intevar.i'     {globals for interface }
%include 'intertns.i'    {defines of interface routines}
```

APPENDIX H
Parpas GENERATION EXAMPLE

The RECOGNITION CONE COMPILER uses the configuration file created by the TRANSFORM EDITOR to create a parallel pascal program. The following configuration file was created when the COMPILE command was given to the TRANSFORM EDITOR. For example the following command

```
compile configl trandir sourcel rconel
```

Configuration File (configl)

```
          3
          1        150
          2         75
          3         25
trcross              1          3
trline000            1          1          2
trline090            1          1          2
```

The following Parallel pascal program is generated from the above configuration file and is put into a file called sourcel.

```
program rconel    (input,output);
const
type
var
{ feature arrays for layer  1        }
      grad0009          : array[0..149,0..149] of integer;
      grad0007          : array[0..149,0..149] of integer;
      grad0005          : array[0..149,0..149] of integer;
      grad0909          : array[0..149,0..149] of integer;
      grad0907          : array[0..149,0..149] of integer;
      grad0905          : array[0..149,0..149] of integer;
{ feature arrays for layer  2        }
      grad0009          : array[0.. 74,0.. 74] of integer;
      grad0007          : array[0.. 74,0.. 74] of integer;
      grad0005          : array[0.. 74,0.. 74] of integer;
      grad0909          : array[0.. 74,0.. 74] of integer;
      grad0907          : array[0.. 74,0.. 74] of integer;
      grad0905          : array[0.. 74,0.. 74] of integer;
{ feature arrays for layer  3        }
      cross000          : array[0.. 24,0.. 24] of integer;
      cross010          : array[0.. 24,0.. 24] of integer;
```

```
      cross010          : array[0.. 24,0.. 24] of integer;
      cross350          : array[0.. 24,0.. 24] of integer;
      line100           : array[0.. 24,0.. 24] of integer;
      line080           : array[0.. 24,0.. 24] of integer;
      line090           : array[0.. 24,0.. 24] of integer;
      line350           : array[0.. 24,0.. 24] of integer;
      line010           : array[0.. 24,0.. 24] of integer;
      line000           : array[0.. 24,0.. 24] of integer;


{*************************************************}
{  parallel procedure for layer  1                }
{*************************************************}
||procedure layer1    ;
||begin
   ||ROWMIN := 0;
   ||ROWMAX := 149;
   ||COLMIN := 0;
   ||COLMAX := 149;
   ||ROWSHRINK :=  2;
   ||COLSHRINK :=  2;
  {*********************************************}
  { transform trline090  in layer  1              }
  {*********************************************}

  ||if  grad0909  [+(-1:0* 10,1:0* 10,0:0* 10)] +
        grad0907  [+(-1:0*  8,1:0*  8,0:0*  8)] +
        grad0905  [+(-1:0*  6,1:0*  6,0:0*  6)] >  40
  ||then
     ||begin
           ||let line100    := line100    +  15;
           ||let line080    := line080    +  15;
           ||let line090    := line090    +  40;
     ||end;
  {*********************************************}
  { transform trline000  in layer  1              }
  {*********************************************}

  ||if  grad0009  [+(0:-1* 10,0:1* 10,0:0* 10)] +
        grad0007  [+(0:-1*  8,0:1*  8,0:0*  8)] +
        grad0005  [+(0:-1*  6,0:1*  6,0:0*  6)] >  40
  ||then
     ||begin
           ||let line350    := line350    +  15;
           ||let line010    := line010    +  15;
           ||let line000    := line000    +  40;
     ||end;
||end;


{*************************************************}
{  parallel procedure for layer  2                }
{*************************************************}
||procedure layer2    ;
```

```
||begin
   ||ROWMIN := Ø;
   ||ROWMAX :=  74;
   ||COLMIN := Ø;
   ||COLMAX :=  74;
   ||ROWSHRINK :=  3;
   ||COLSHRINK :=  3;
  {***********************************************}
  { transform trline0900  in layer  2             }
  {***********************************************}

  ||if  grad0909  [+(-1:Ø* 10,1:Ø* 10,Ø:Ø* 10)] +
        grad0907  [+(-1:Ø*  8,1:Ø*  8,Ø:Ø*  8)] +
        grad0905  [+(-1:Ø*  6,1:Ø*  6,Ø:Ø*  6)] >  40
  ||then
     ||begin
           ||let line100    := line100    +  15;
           ||let line080    := line080    +  15;
           ||let line090    := line090    +  40;
     ||end;
  {***********************************************}
  { transform trline000  in layer  2              }
  {***********************************************}

  ||if  grad0009  [+(Ø:-1* 10,Ø:1* 10,Ø:Ø* 10)] +
        grad0007  [+(Ø:-1*  8,Ø:1*  8,Ø:Ø*  8)] +
        grad0005  [+(Ø:-1*  6,Ø:1*  6,Ø:Ø*  6)] >  40
  ||then
     ||begin
           ||let line350    := line350    +  15;
           ||let line010    := line010    +  15;
           ||let line000    := line000    +  40;
     ||end;
||end;


{*****************************************************}
{  parallel procedure for layer  3                    }
{*****************************************************}
||procedure layer3    ;
||begin
   ||ROWMIN := Ø;
   ||ROWMAX :=  24;
   ||COLMIN := Ø;
   ||COLMAX :=  24;
  {***********************************************}
  { transform trcross    in layer  3              }
  {***********************************************}

  ||if  line100   [+(Ø:-1* 10,Ø:1* 10,Ø:Ø* 10)] +
        line080   [+(Ø:-1* 10,Ø:1* 10,Ø:Ø* 10)] +
        line090   [+(Ø:-1* 20,Ø:1* 20,Ø:Ø* 20)] +
        line350   [+(-1:Ø* 10,Ø:-1* 10,Ø:1* 10,1:Ø* 10,Ø:Ø* 10)] +
        line010   [+(-1:Ø* 10,Ø:-1* 10,Ø:1* 10,1:Ø* 10,Ø:Ø* 10)] +
```

```
          line000   [+(-1:0* 20,0:-1* 20,0:1* 20,1:0* 20,0:0* 20)] >  40
   ||then
      ||begin
             ||let cross350   := cross350   +  15;
             ||let cross010   := cross010   +  15;
             ||let cross000   := cross000   +  40;
      ||end;
 ||end;

 {******************** start of main procedure ******************}
 begin
    layer1    ;
    layer2    ;
    layer3    ;
end.
```

APPENDIX I
Numeric Transformation Example

Numeric Transform Lookfor Pattern

1Ø tv1	2Ø tv2	3Ø tv3
1Ø tv4	2Ø tv5	3Ø tv6
1Ø tv7	2Ø tv8	3Ø tv9

Integer array

4	4	12	17	16	13
6	av1 5	av2 1Ø	av3 15	2Ø	18
7	av4 2Ø	av5 1Ø	av6 5	12	1Ø
5	av7 6	av8 7	av9 16	18	13
6	8	11	11	9	14

$$\frac{\sum_{i=1}^{i=n} tv_i}{\sum_{i=1}^{i=n} av_i} = 1.9149$$

Ratio transform calculation

```
max  =  20
 ev1  =  1.9149  *  5    = 9.5745
 ev2  =  1.9149  *  10   = 19.149
 ev3  =  1.9149  *  15   = 28.7235
 ev4  =  1.9149  *  20   = 38.298
 ev5  =  1.9149  *  10   = 19.149
 ev6  =  1.9149  *  5    = 9.5745
 ev7  =  1.9149  *  6    = 11.4894
 ev8  =  1.9149  *  7    = 13.4043
 ev9  =  1.9149  *  16   = 30.6384
```

For a value A = 3.0

|ev1 - av1| = .4255 : multiply factor = .93817

$$e^{-\frac{.4255}{20} * 3} = .93817$$

```
|ev2 - tv2|  = .851 : multiply factor = .88016
|ev3 - tv3|  = 1.2765 : multiply factor = .82574
|ev4 - tv4|  = 28.298 : multiply factor = .01436
|ev5 - tv5|  = .851 : multiply factor = .88016
|ev6 - tv6|  = 20.4255 : multiply factor = .04671
|ev7 - tv7|  = 1.4894 : multiply factor = .79979
|ev8 - tv8|  = 6.5957 : multiply factor = .37182
|ev9 - tv9|  = .6384 : multiply factor = .90868
```

Depending upon what the weights of the particular look-for attributes are and the transforms threshold, this transform might or might not succeed.

BIBLIOGRAPHY

[1] Douglass, Robert; Recognition and Spatial Organization of Objects in Natural Scenes; CSD Tech Report 5420 (UW Madison); 1978

[2] Uhr, Leonard and Douglass, Robert; A Parallel-Serial 'Recognition Cone' System for Perception : Some Test Results; CSD Tech Report 4894 (UW Madison) 1977

[3] Uhr, Leonard; 'Recognition Cones' And Some Test Results : The Imminent Arrival of Well-Structured Parallel-Serial Computers, Positions and Positions on Positions; CSD Tech Report 5099 (UW Madison); 1977

[4] Uhr, Leonard; A Language for Parallel Processing of Arrays Embedded in Pascal; CSD Tech Report 6730 (UW Madison); 1979

[5] Uhr, Leonard; A Higher-Level Language for a Large Parallel Array Computer; CSD Tech Report 6295 (UW Madison); 1979