Computer Sciences Department
The University of Wisconsin
1210 West Dayton Street
Madison, Wisconsin 53706

WIS-CS-73-172

Received February 2, 1973

Corrected June 18, 1973

Presented at 1st National Computer Conference and published in Proceedings (without corrections), New York, June, 1973.

THE DESCRIPTION OF SCENES OVER TIME AND SPACE

by

Leonard Uhr

Technical Report #172

February 1973

This research has been partially supported by grants from the National Institute of Mental Health (MH-12266), the National Science Foundation (GJ-36312), NASA (NGR-50-002-160) and the University of Wisconsin Graduate School.

THE DESCRIPTION OF SCENES OVER TIME AND SPACE

Leonard Uhr

ABSTRACT

This paper explores techniques for pattern recognition and description of more than one object, where objects extend over time as well as over space. Three major interrelated issues are examined:

A. Describing, as opposed to simple naming;

B. Time, and the way a system can build up a short-term perceptual memory to handle and make use of changing information;

C. Glancing around, in order to recognize and describe.

Two actual computer programs are presented, to make clear exactly what the issues are and how they are handled, and to make comparisons possible.

Program DESCRIBE-1 uses configurational characterizers, and describes by outputting several highly-implied objects.

Program DESCRIBE-2 handles inputs that extend over time. It glances about and gives structural descriptions of the parts of objects.

A number of variants are examined, to explore how a program can describe with more variety.

Descriptors: Scene Analysis, Describing, Pattern Naming, Time, Flexible Pattern Recognition

THE DESCRIPTION OF SCENES OVER SPACE AND TIME

Leonard Uhr

INTRODUCTION

Most pattern recognition research has been concerned with the assignment of a single name to an input field. But rarely do we find single, isolated objects in the real world.

Describing Scenes of Several Objects that Interact Over Time and Space

Rather, we need programs that describe scenes of several interacting objects, and further describe each object, commenting upon parts, qualities, and other details of interest.

Just as they are not unitary things isolated in space, real-world objects are not isolated, as in a photo, in a static moment of time. But virtually no research has been done on the recognition of objects that come, go, move, and change over time.

Once we introduce time we raise a number of interesting issues: What kind of short-term perceptual memory is needed? How does the system handle, and coordinate, time for perception, and for response (in our case, for describing)? How can the system use information gathered so far during perceptual interaction with its environment in order to help it glance about and attend to objects as they come into view at future times?

These are extremely complex and subtle, and interesting, questions. This paper is a first attempt to tackle them.

The Use of Bare-Bones EASEy Programs to Make Things Clear

Actual computer programs are presented, described, and discussed, in order to make completely clear exactly what is happening, and to allow us to examine a variety of variations. These programs are kept to their bare bones, and coded in a relatively simple English-like variant of SNOBOL called EASEy (an Encoder for Algorithmic Syntactic English that's Easey). Programs and variants are numbered so that they can be compared one with another, and the Appendix and the EASEy primer (Uhr, 1973a) should be helpful when details of the code are not apparent.

These programs are designed to demonstrate a variety of possible mechanisms, to be compared and contrasted one with another. They depend upon the particular set of characterizers given them (by their programmers, or by learning routines). We have not been able to examine within the brief confines of this paper the kind of behavior they will exhibit, and the variety of sensed scenes they will handle, given a sufficiently large and appropriate set.

HISTORY

Relatively little research has been done on general systems that describe the various objects in a scene, along with their structure of parts, qualities, defects, or other characteristics. On the contrary, almost all pattern recognition research has concentrated on the assigning of a single name to an input. When scenes are examined, they tend to be treated in an ad hoc way, using routines designed to find special features of interest.

Systems for Recognition of Continuous Handwriting and Speech

Some research has been done on recognizing handwriting, where several letters continue into one another, to form words and lines (e.g., Frischkof and Harmon, 1961); Mermelstein, 1964; Eden, 1968; Uhr and Vossler, 1963). And some attempts have been made in speech recognition to describe the spoken utterance in terms of their basic components, (e.g., formants, (Forgie and Forgie, 1959), phonemes (Reddy, 1967). But virtually all of this work first decomposes the scene, whether of letters or sounds, into individual units, thus reducing the problem to standard single-name pattern recognition, and then assigns a single name, using standard techniques.

Systems that Build Internal Descriptions to Name

Some programs that name develop a rich internal description of the pattern in order to achieve the name. The best examples of such an approach are probably the "syntactic" recognizers (e.g., Grimsdale et al, 1959; Shaw, 1967; Ledley and Ruddle, 1965; Marrill, et al., 1963; Narashiman, 1966, 1971; Uhr, 1971; Fu and Swain, 1971), since they build up structures of larger and larger wholes from meaningful parts. But in fact almost any naming program that applies a set of characterizers to an input pattern can be thought of as building up an internal description, of those characterizers that succeed, and where, and those that fail. This information might be output, as a description; it would be useful and meaningful to the extent that the characterizers were ones that made sense to the human receiver. And any recognizer could "describe" by outputting some of the alternate implied names that it might have chosen, but didn't.

Systems that Examine Continuous Fields of Objects

A variety of important problem areas confront us with scenes of objects. These include many biological preparations, e.g., blood cells, nerve tissue, chromosomes; X-rays, e.g. of heart, lungs, or bones; and aerial photos of cities, country side, or cloud cover. Up to now such work has concentrated on extracting particular features of interest, e.g., an enlargement or other anomaly of an organ; an aberrant blood cell; a texture of a certain sort; a break in a bone; an edge of a cloud; a boundary between two fields of different crops. Rarely is a complete description asked for or given; rather, a special-purpose program is coded to analyze and search for particular signs of interest. (See e.g., Lipkin, Watt and Kirsch, 1966; Rosenfeld, 1969; Lipkin and Rosenfeld, 1970; Hall, et al., 1971; Sutton and Hall, 1972; Ledley, 1972; Hall et al., 1972; Ausherman et al, 1972).

Kirsch (1964), Londe and Simmons (1965), Fischler (1969), Firschein and Fischler (1971), Sauvain and Uhr (1969), Uhr (1968, 1972, 1973b), are examples of research that attempts to develop more complete descriptions, though usually under the assumption that only one, or at most two, simple, standard, noise-free objects are present in the scene.

Systems for the Description of Three-Dimensional Objects

A good deal of interest has arisen in recent years in the problem of recognizing objects that overlap, often in three dimensions, in the fields of computer graphics (Roberts, 1965, Guzman, 1968) and robots (Brice and Fennema, 1970). But most of this work very carefully attempts to find the edges that are predicted to be present for one of the small number of alternate possible objects.

Recognition Over Time

Virtually no research has been done with objects that move or otherwise change over time, except for special-purpose systems, such as those that track clouds (e.g., Lillestrand, 1972; Smith and Phillips, 1972). Nor is the author aware of any systems that build up a short-term perceptual memory in order to handle such continually changing inputs.

Glancing About, and Conversation

Relatively little research has been done on pattern recognition systems that decide where to look next as a function of what information they have gathered so far. Most "concept formation" systems have a very simple and rigid structure of this sort (see e.g., Kochen, 1960; Hunt, 1962; Towster, 1969). Uhr (1969, 1973b, chapter 8) has examined more flexible systems of this sort.

A PROGRAM FOR TWO-DIMENSIONAL RECOGNITION AND DESCRIPTION

We will now examine two programs that explore the problems of describing objects that change over time. The first, DESCRIBE-1, makes minimal changes to a relatively typical configurational pattern recognizer (see Uhr, 1973b, chapters 2-4) to allow it to describe.

DESCRIBE-1 handles recognition in two dimensions, using configurational characterizers that are sensitive to interactions among their parts. It insists that each part be exactly positioned to match; but characterizers are threshold elements, so that they can succeed when any sufficiently highly weighted subset of their parts succeeds.

It gives a first approximation to a description, since it outputs all names that have been sufficiently implied to exceed a CHOOSE* level.

(OVERVIEW DESCRIBE-1. Uses weighted, positioned configurations in 2 dimensions.

(Outputs all NAMEs* implied above CHOOSE level.	DESCRIBE-1
INITialize the CHOOSE level and the configuration CHARacterizers.	M1-MN
UPDATE Erase the old PRESENT and the ROWPresent	1
IN input the new PRESENT, ROW by ROW: put the CHARacterizers on LOOKFOR	2-5
PERCEIVE Get each CHARacterizer, its THRESHold, DESCRiption, and IMPLIEDS	6-8
Get each PART, its ROW and COLumn location, and WeighT, from the DESCRiption, and look for it, positioned, in the PRESENT; and add the WeighT to TOTAL if the part is found	9-10 11
TEST This CHARacterizer succeeds if TOTAL is above THRESHold.	12
IMPLY Get each NAME and its WeighT from the IMPLIEDS, and add this WeighT into the TOTAL for this NAME on MAYBE	13 14-16
DECIDE Get each NAME and its TOTAL from MAYBE, and, if TOTAL is higher than the CHOOSE level, output it.	17 18-19

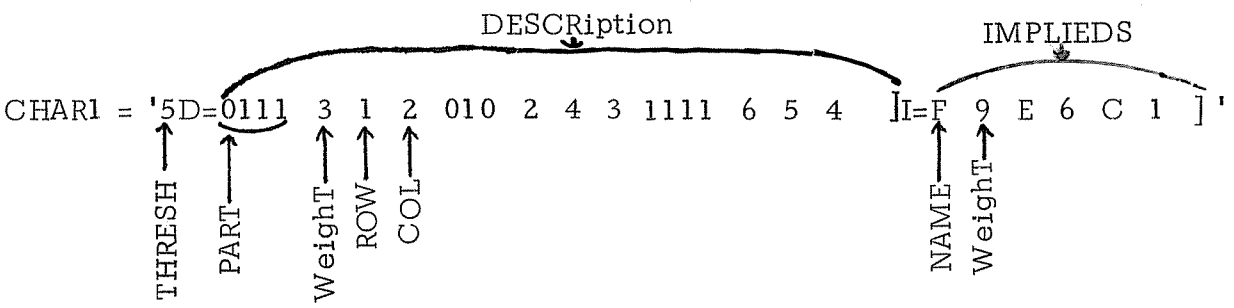
*Capitalized strings in the text and the OVERVIEWs refer to programmer-defined string names in the programs.

```
(DESCRIBE-1. (Handles 2-dimensional patterns. CHARacterizers are WeighTed,  DESCRIBE-1
(positioned configurations that succeed if match above THREshold. Applies
(CHARacterizers till a NAME's TOTAL implied weight exceeds CHOOSE level
        Set CHOOSE = 30                                                    1M1
(Characterizers should go here.
UPDATE2 erase PRESENT, ROWP                                                3 1
(input the PRESENT pattern ROW by ROW)
IN      input TYPE, Row till ] [+to $TYPE -to END]2                        2
S       ROWP = ROWP + 1                                                    3
        On PRESENT set ] ROWP] ROW ] [to IN]                               4 4
R       Set LOOKFOR = 'CHAR1 CHAR2 ... CHARN '                             5
PERCEIVE from LOOKFOR get CHAR = [- DECIDE]                                6
(Look for each OBJect in the DESCRiption at ROW and COLumn specified)
        from $CHAR get THRESH 'D=' DESCR ']I=' IMPLIEDS ] [- PERCEIVE]     5 7
        erase TOTAL                                                        8
P1      from DESCR get PART WT ROW COL = [- TEST]                          6 9
        from PRESENT get ] that ROW ] call COL symbols LEFT, get that PART [-P1] 3 10
            TOTAL = TOTAL + WT [P1]                                        11
TEST    is THRESH lessthan TOTAL? [- PERCEIVE]                             12
(If total WeighT of OBJects got exceeds THRESHold, merge IMPLIEDS NAMEs onto MAYBE
IMPLY   from IMPLIEDS get NAME WT = [- PERCEIVE]                           13
        from MAYBE get # that NAME # TOTAL = [- I1 ]                       4 14
            WT = TOTAL + WT                                                15
I1      on MAYBE list NAME WT [IMPLY]                                      16
(output description with all NAMEs implied above CHOOSE level.
DECIDE  from MAYBE get NAME TOTAL = [- UPDATE]                             17
        is TOTAL lessthan CHOOSE ? [+ DECIDE]                              18
OUT         output 'THERE IS ' NAME [DECIDE]                               19
end2                                                                       --

S 00111110]      (example data cards for an 'F' in an 8 by 8 matrix)       I1
S 00110000]                                                                I2
S 00100000]                                                                I3
S 00011110]                                                                I4
S 00011111]                                                                I5
S 00110000]                                                                I6
S 00110000]                                                                I7
S 00100000]                                                                I8
R ]                                                                        I9
```

[1]NOTE. See the Appendix for a brief description of EASEy programs. (Examples of the [6] major program constructs discussed in the Appendix are superscripted 1 through 6 in this program.) Caps are used in the text to refer to program constructs.

If given good characterizers for the letters, DESCRIBE-1 will output the name F and, probably, a few other names like C, I, and, possibly, E and T. For good performance, the program would need several hundred characterizers, of the following sort:



And it would also need more code to allow these characterizers to succeed within some region (e.g., Uhr and Jordan, 1969; Uhr, 1973b, Chapters 4, 21) rather than in an exact position (alternately, it could be given a separate characterizer for every position; but this would clog memory with far too many characterizers, and slow down processing time). But this kind of program, with an adequate set of characterizers, performs well (e.g., Uhr and Vossler, 1963; Andrews, Atrubin, and Hu, 1968), possibly as well as any approach (see Zobrist, 1971, for comparisons).

Other Possibilities for Descriptive Information

DESCRIBE-1 output as its description all the NAMEs IMPLIED above the CHOOSE level. DESCRIBE-2 and its extensions will explore a variety of richer descriptive information, including the objects' parts, structure, salience, qualities, and location.

DESCRIBE-1 has a bit more descriptive information that it could make available fairly easily. The NAME's TOTAL weight could be output, to indicate its salience, and/or the program's certainty. The names of the CHARacterizers, the PARTs of their DESCRiptions that succeeded, and their locations could be stored with the names, and output as qualifying information (this will be done in DESCRIBE-2).

Weights and Thresholds

DESCRIBE-1 TOTALs the weight of each PART of a DESCRiption (statement 11). The CHARacterizer succeeds if TOTAL exceeds the THRESHold (12). This allows the programmer to design characterizers that have any desired amount of looseness, in the sense of a threshold decision element that succeeds when any of a large number of combinations of subsets of its input PARTs succeed. If the threshold is lower than the weight of any of the PARTs, such a characterizer is equivalent to an "OR" operator; if the threshold equals the sum of all the weights of all the PARTs, it is equivalent to an "AND" operators.

The programs in this paper add weights together, since this is the simplest thing to do. But it is easy to have the program multiply weights, or compute whatever other combining function is deemed appropriate.

Note how similar are the THRESHold for deciding whether a CHARacterizer has succeeded and the CHOOSE level for deciding that a NAME has been sufficiently highly implied (by one or more characterizers) to be output as part of the program's description of the input PRESENT.

Describing Vs. (Merely) Naming

The typical pattern recognition program simply chooses and outputs a single name that it assigns to the input.

This is usually done by having the program choose from MAYBE the single most highly implied NAME, rather than all the NAMEs implied above a CHOOSE level.

Sometimes the program will merely choose and output the first name whose TOTAL implied weight exceeds some minimum level for choosing. This would simplify DESCRIBE-1, which could now put the test that compares TOTAL weight with CHOOSE (statement 18) right after statement 15, and eliminate the DECIDE loop through MAYBE (statement 17) - with suitable changes in gotos.

DESCRIBING SCENES OF OBJECTS, AND THEIR PARTS, OVER SPACE AND TIME

DESCRIBE-2 begins to handle descriptions over time, and descriptions that talk about the parts and the subparts of which the recognized wholes are composed.

To keep it short, it has been over-simplified, so that it handles only 1-dimensional string inputs (e.g., English sentences), and uses characterizers that do not handle position or interactions among parts (except to the extent that parts are explicitly put together, as though

into a rigid template, and all possible combinations of this sort are added to its memory, as separate characterizers - a theoretically possible but practically unfeasible procedure).

In addition to describing the scene using all names implied above a CHOOSE level, it further describes each name by outputting all its parts (each with its column location) and all of each part's parts, until it hits the lowest level. It further does this from the point of view of each name - that is, it says in effect, "If this name is present, then these parts are present."

Time is handled by merging each PRESENT moment into a SEEN list, where OBJects are made salient (by high weights) when they first appear, and to the extent that they move, but then gradually FADE away when they are no longer present.

(OVERVIEW DESCRIBE-2. Builds short-term memory over TIME and space.

		DESCRIBE-2
INITialize memory; UPDATE TIME, input PRESENT		M1-3
SENSE	Merge each OBJect in PRESENT into what has recently been SEEN, its WeighT a function of newness and movement.	4-8
FADE	Down-weight, and erase, OBJects no longer in PRESENT.	9-11
PERCEIVE	Get the IMPLIEDS names for each OBJect still in SEEN,	12-14
IMPLY	Put them on LOOKFOR and merge onto MAYBE, building a list of PARTS.	15-19
EVAL	output each NAME on MAYBE whose TOTAL weight exceeds its THRESHold.	20-23
DESCRIBE	output all PARTS of the NAMEd object, and PARTS2 of each, that have public TRANSforms (giving their COLumn locations).	24-28

```
(DESCRIBE-2.  (Merges short-term SENSEory memory over time and space.
(OBJects are salient when NEW or move; FADE out when no longer PRESENT
(Advance TIME, input the new PRESENT)                                  1     2
                        (characterizers should go here)
UPDATE      erase COLP                                                1.V   1
        set TIME = TIME + 1                                            .1   2
IN      input PRESENT till ] [- END]                                  2.V   3
(Merge each OBJ in PRESENT with those already SEEN)
SENSE   from PRESENT get OBJ = [- FADE]                                .1   4
            COLP = COLP + 1                                            .2   5
        from SEEN get # that OBJ # WT COL = [- NEW]                    .3   6
        on LOOKFOR list OBJ WT + ABS(COLP - COL) COLP [SENSE]         5.V   7
NEW     on LOOKFOR list OBJ 9 COLP [SENSE]                             .1   8
FADE    from SEEN get OBJ WT COL = [- PERCEIVE]                        .2   9
        is WT lessthan 1?  [+ FADE]                                    .3  10
            on LOOKFOR list OBJ WT - 1  COL [FADE]                     .4  11
PERCEIVE  SEEN = LOOKFOR                                               .5  12
(Get NAMEs implied by each OBJect on LOOKFOR)
P1      from LOOKFOR get OBJ WTL COL = [- EVAL]                       6.V  13
        from $OBJ get 'I=' IMPLIEDS ]  [- P1]                         7.V  14
(Put TOTAL of WeighTs for each IMPLIEDS NAME on MAYBE)
IMPLY   from IMPLIEDS get NAME WT = [- P1]                           13.V  15
        on LOOKFOR list NAME WT COL                                    .1  16
        from MAYBE get # that NAME # TOTAL PARTS ] = [-I1]            14.V  17
            WTL = TOTAL + WTL                                              18
I1      on MAYBE list NAME WT + WTL PARTS OBJ COL ] [IMPLY]           16.V  19
(EVALuate total WeighT against NAME's THRESHold, and output  if greater)
EVAL    from MAYBE get NAME TOTAL PARTS ] = [- UPDATE]                17.V  20
        from $NAME get 'T=' THRESH                                     .1  21
            is TOTAL lessthan THRESH?  [+ EVAL]                       18.V  22
(Describes giving overlapping object NAMEs and overlapping parts and parts
(of parts)
        output  'AT TIME = '  TIME  ' IT MIGHT BE = '  NAME  ' DESCRIBED:' 19.V  23
(outputs all the public PARTS, down to the lowest level)
DESCRIBE from PARTS get OBJ COL = [- EVAL]                                     24
        from MAYBE get # that OBJ # TOTAL PARTS2 ]  [- DE1]                    25
            on PARTS set PARTS2                                                26
(If OBJect has an external TRANSform,  outputs  it)
DE1     from $OBJ get 'T=' TRANS ]  [- DESCRIBE]                               27
            output TRANS ' AT ' COL [DESCRIBE]                                 28
END                                                                            --

LEFT-ARM TRUNK LEGS RIGHT-ARM EYE NOSE EYE CHIN ]                              I1
DOG-EAR SNOUT LEG LEG ]                                                        I2
TRUNK ]                                                                        I3
LEG LEG TAIL ]                                                                 I4
```

NOTE that size ABS(...) is a function that computes the ABSolute value of the expression within parentheses; it is defined and written by the programmer (though I don't bother to show the needed code).

DESCRIBE-2 needs a set of characterizers (put before statement 1) of the following simple sort, where each thing points up to the things implied:

```
LEFT-ARM = 'I=PERSON 3 ]'                          M1
TRUCK    = 'I=PERSON 5 DOG 3 ]'                    M2
EYE      = 'I=HEAD 3 ]'                            M3
NOSE     = 'I=HEAD 3 ]'                            M4
HEAD     = 'I=PERSON 5 ]P=EYE 5 EAR 3'
+                 'NOSE 7 CHIN 4 MOUTH 6 ]'        M5
```

These (plus additional characterizers for LEGS, CHIN, DOG-EAR, SNOUT, LEG) would allow it to notice PERSON and DOG. Since EYE, NOSE, etc. point to HEAD, and HEAD points to PERSON, it is also capable of describing a PERSON as having a HEAD, which has an EYE, NOSE, etc. (But Il would not be described as containing an EAR.)

Note how characterizer M5 shows links from HEAD back to its parts (EYE, EAR, etc.) - links that are not actually used. With additional code, DESCRIBE-2 could add these to LOOKFOR, and use them to describe what is missing from a particular object.

To handle an interesting variety of inputs, we would need to give DESCRIBE-2 a much larger set of characterizers. DESCRIBE-1 is actually the much more powerful pattern recognizer since it handles configurations of intereacting parts and does not rely upon a space (which is reasonable for sentences, and in fact typically used by parsing programs) to delimit objects.

Notice how a pattern can be recognized over time, even though at no single moment are all of its parts present, because characterizers look at the SEEN list, which only gradually fades away. Thus e.g., a suitable characterizer would output DOG in response to Inputs I2-I4. The description of the DOG (and of the PERSON Input in I1) would depend upon the particular parts present.

STILL FURTHER POSSIBILITIES FOR DESCRIPTIONS

We are now in a position to make a number of simple variations, as described below.

Describing One Vs. Several Objects

DESCRIBE-2 describes rather exhaustively. For each NAME whose TOTAL weight exceeds THRESHold, and therefore is output, it also outputs all PARTS (that have public TRANSforms), and PARTS2 of parts, down to the lowest level. Thus the description can contain much overlap, of names and of parts. This can be varied in a number of ways.

A. The simplest would have the program output only one NAME of an object and its PARTS description:

(DESCRIBE-2-A. outputs only one object's NAME and description of PARTS. 2-A
DESCRIBE from PARTS get OBJ COL = [-UPDATE] 24.V*

B. A slight change:

(DESCRIBE-2-B. Gives non-overlapping descriptions
from MAYBE get # that OBJ # TOTAL PARTS2] = [-DE1] 25.V

*Numbers like 24.V indicate Variations to the corresponding statements in DESCRIBE-2.

would give non-overlapping descriptions of non-overlapping things, starting with whatever OBJect happened to come first on the MAYBE list. This would make more sense if the NAME with the MAXimum TOTAL weight were got from MAYBE in statement 20 (this entails a simple loop through the NAMEs on MAYBE, to get the one with the highest associated weight).

C. Still another variant would give overlapping descriptions of non-overlapping NAMEd objects:

```
(DESCRIBE-2-C. Describes several non-overlapping objects        2-C
        set COPYM = MAYBE                                       20.1
        from COPYM get # that OBJ # TOTAL PARTS2 ] = [-TO DE1]  25.V
```

Details Vs. Wholes

D. All of these go from top down, from wholes to details. The following simple variant would dip down to details, and then go up:

```
(DESCRIBE-2-D. Dips down to give details first.                 2-D
    at start of PARTS set PARTS2                                26.V
```

This gives a funny kind of order, wandering from top to bottom, and then back up.

E. A slightly more complex program would set all the PARTS2 at the start of an ALLPARTS list, and only then start developing the description, from ALLPARTS.

Keeping Descriptions Short

These all give descriptions that are far too long, since they contain all details. What is really needed is a system that chooses to

output only the pertinent details - but this is an extremely complex matter, as will be discussed below, since it depends upon a deep semantic understanding of the objects in the scene, their import, and their import to the hearer of the description. So for now we can only examine the simplest of methods for keeping descriptions from growing unreasonably long.

F. First, we might have the program put only highly weighted OBJects onto the PARTS list (by checking the weight at statement 18)

G. Second, only parts of a specified QUALity might be output:

```
(DESCRIBE-2-G.  outputs OBJect only if it is of the QUALity specified.   2-G
        QUAL = 'SHAPE'                                                   M1.1
        from $OBJ get that QUAL [-to DESCRIBE]                           27.1
```

H. Third, the program might output only up to a fixed number of object parts:

```
(DESCRIBE-2-H.  outputs only a specified Number of PARTS.                2-H
        ENOUGHP = I2                                                     M1.1
UPDATE  erase COLP, NPARTS                                               1.V
        is NPARTS lessthan ENOUGHP? [-to UPDATE]                         24.1
                NPARTS = NPARTS + 1                                      24.2
```

I. Similarly, fixed numbers of objects, and/or of characterizers to be looked for, could be set (this is best done along with a function that gets the MAXimum implied).

GLANCING AROUND AND ACTING OVER TIME

Glancing, Noticing, and Focussing Attention

DESCRIBE-2 "glaces around", looking for higher-level wholes as a function of things already implied, because statement 16 puts an implied NAME onto LOOKFOR, so that the program will later look for any names that it implies, and so on up the hierarchy. (Note that this feature can easily be added to DESCRIBE-1.)

J. As a variant, we might use:

```
(DESCRIBE-2-J.  Tends to LOOK FOR NAMEs at higher levels.      2-J
    on LOOKFOR list NAME WT + WTL COL                          16.1.V
```

(or some other function of the weights of both the NAME and the OBJect that implied it), so that a name at a higher level has a higher weight, reflecting the weights of all its lower levels.

K. Alternately, we might add the statement:

```
(DESCRIBE-2-K.  Looks only with NAMEs chosen to output.        2-K
    on LOOKFOR list NAME TOTAL                                 22.1
```

so that only at the end, if it has been chosen for output because its TOTAL implied weight has exceeded its THRESHold, is the NAME put onto LOOKFOR. This will put many fewer NAMEs on LOOKFOR, since it requires a more stringent procedure for evaluating the importance of each.

Glancing Over Time

It also has the interesting characteristic that it adds a NAME to LOOKFOR to be processed at the next moment of TIME (since it loops

back to UPDATE in statement 20), whereas the addition after statement 16 will affect processing immediately, on the PRESENT moment.

We are thus beginning to introduce a second source of short-term-memory that gives continuity over time - not only in the SEEN list, that only gradually fades away, but also in the LOOKFOR list.

L. Still other variants would have the program a) add implied names to a NEXTLOOK list at 16.1, and only at UPDATE time set LOOKFOR = NEXTLOOK, so that the casually got names would not be processed until the next time: b) loop back from EVALuation to PERCEIVE some more if new NAMEs have been put onto LOOKFOR (22.1), so that they are processed immediately, at this time.

DISCUSSION

The Short-Term Perceptual Memory

Merging of OBJects from the recent past into the SEEN list, where their salience is a function of their newness and motion, and they fade away only slowly after having disappeared from the environment, appears to be simple, elegant, and sufficient to allow systems to handle environments that continue and change over time. But it may also be useful to introduce further inertia over time by using a separate list of CHARacterizers to LOOKFOR, where LOOKFOR also continues over time. There are many interesting alternative possibilities here, only a few of which have been touched upon in this paper.

Focussing Attention and Noticing

As soon as we let a program add characterizers to its LOOKFOR list we introduce a whole range of possibilities for focussing attention and concentrating on certain things, and type of things. In DESCRIBE-2 what has already been noticed implies new characterizers, and new

objects, which can themselves imply their parts, and characterizers that would imply them.

We can, if we wish, initialize our programs to contain one or more names of objects to LOOKFOR. This will focus attention on these objects, and on the characterizers that imply them, and their subparts. The strength of this focussing will be a function of their weights. Depending upon details of thresholds and choose levels, the system will now find more of the things it has been set to look for, with less certainty thus leading to false positives, and it will tend not to notice other things - all rather reminiscent of human beings.

The Influence of Internal Needs and External Suggestions

These systems are now in a position to have their processes influenced from a variety of sources. Commands and conversational suggestions from the external world can suggest what to look for, and what kinds of descriptions to output. Internal needs and goals can also play a role. In all cases, the various sources of set are merged into the LOOKFOR list, which controls processing.

Changing Points Of View

People will tend to describe scenes as though they are fields of physical objects, with only one object at one place at one time. We will point to and describe a number of faces, but without adding, "oh there's still another face that's made of the left ear of face 3 and the right chin of face 7;" nor will be go on to describe the details of faces, saying "there's a leaf; there are two fish."

But we can very easily shift to different attitudes. For example, if we're shown a drawing and told it contains 82 hidden faces and 212

leaves, we will almost immediately see overlaps.

The germ of this ability appears to lie in the different variant attitudes for outputting overlapping, or non-overlapping,descriptions in DESCRIBE-2-A through 2-E. What still needs doing is to give the program control over which of these attitudes it will take, and let it decide as a function of a variety of pieces of information that it has gathered during its conversational interaction with its environment, including the hearers of its descriptions.

Recognizing and Acting Over Time

It is unrealistic to have a system apply no matter how many characterizers, and output no matter how complex a description, in a single moment of time. Time is needed to recognize, describe, notice, and act. DESCRIBE-2-K and 2-L begin to take this into account, but once again there is a large variety of other possibilities. Ideally, we should consider what is the common real time in which the environment, the program's perceptual processes, and its motor actions must all take place, and we should assign appropriate real times to each separate process. This makes apparent the issues of parallel vs. serial vs. parallel-serial processes, and time needed for feedback loops that monitor action.

What is a Description?

The concept of a "description" is hazy, and not easily defined. Most people will look at a scene and say something like, "There's a man walking a dog through the woods." A fastidious few will say instead, "There's a man with fingers circling around the loop of a leash, whose other end appears to be attached to a dog via a collar; there are also

5 trees in the picture." A detective might say, "There's a tall man in a coonskin cap with a black handle-bar mustache and a smudge on his left cheek," while a dog-nut might say, "There's a siberian husky with eyes that are too slanted," and a nature lover, "there's a mixture of honeysuckle, maple and pine, and it looks like Spring, but I don't see any birds."

It is hard to conceive of a description in which the describer does not 1) make major judgments as to what is important and, further, 2) superimpose his own "understanding" of the objects in the scene, and their interrelationships and their import.

Sometimes the scene will be impoverished to the point where things seem relatively simple, at least on the surface. Thus almost everybody will look at a sheet of paper on which letters have been written and say, "There's an 'E'" or "There's the word 'THE'". If we press further most people will say, "The 'E' has a vertical bar with short horizontal bars extending to the right from top, middle and bottom" - if it is a standard, well-drawn 'E' - and they will think us a bit crazy for asking (why? - I think because such a description feels like a tautology, possibly because it is a constructive definition of an 'E', one that we have pretty generally agreed to use).

If the E is sloppy, and/or we press the describer to say more, we will begin to get statements like, "The top bar wavers and has breaks", "It's long and skinny". A more compulsive person might say, "The top bar angles down 20^{o} for 1/4 inch, then curves up for 1/2 inch, until it is 1/4 inch above its start, then goes straight for 1/2 inch."

This description probably sounds contrived to most readers. But that leads into a third important characteristic of descriptions. Once we are pushed beyond the ordinary level of description we must grope for terms and framework. In general, the describer 3) says that he infers

his hearer will consider pertinent. Thus the diagnostician will tell the neurologist, "the wavering strokes have the quality of palsy rather than brain damage", but he will tell the accountant, "the smudges come because the pencil lead is too soft".

Description is now squarely in the middle of conversational interaction - just where I think it should be, but this raises even more complex and subtle problems. Now we must worry not only about 1) the actual objects in the scene, and their parts and relations, and 2) the describer's understanding of the objects in the scene, but also 3) the hearer's understanding of these objects, 4) the describer's understanding of the hearer's understanding, and even 5) the hearer's understanding of the describer's understanding of the hearer's understanding. For example, the dog-nut might assume that the hearer is a secret dog-nut, or at least realizes that many people are dog-nuts and probably also the describer. And the hearer might infer that the describer knows he the hearer likes dogs, and wants to suck him into an interest in the fine points.

In addition to arguing for the complexity and subtlety of a description, this is to argue that it is intimately related to a rich semantic understanding that describer and hearer have in common -- at least to some extent, along with an understanding by each of the situation of communication, in which one tries to impart suitable information to the other.

We cannot expect pertinent, sensitive descriptions until we have programs that have the necessarily rich semantic understanding of the world whose scenes are being described , and of what this world means to their hearers. But we can still extend pattern recognition programs that merely name to the point where they also describe, albeit either in exhaustive detail or in overly-rigid conventional ways. And we can begin

to tailor their descriptions to their hearers, as a result of simple conversational interactions.

Appendix: A Note on Programs (See Uhr, 1973a for details)

1. Numbering at the right identifies statements, and allows for comparisons between programs. M indicates initializing Memory statements: I indicates cards that are Input by the program. .V indicates a Varient, .1 an additional statement.

2. A program consists of a sequence of statements, and END card, and any data cards for input. (Statements that start with a parenthesis are comments, and are ignored.) Statement labels start at the left; gotos are at the right, within brackets (+ means branch on success; - on failure; otherwise it is an unconditional branch).

3. Strings in capitals are programmer-defined. Strings in underlined lower-case are system commands that must be present (they would be keypunched in caps to run the program). These include input, output, erase, set, list, get, start, call, that, and the inequalities. Other lower-case strings merely serve to help make the program understandable; they could be eliminated.

4. EASEy automatically treats a space following a string as though it were a delimiter; it thus automatically extracts a sequence of strings and treat them as names. The end-bracket] acts similarly as a delimiter, but the programmer must specify it. The symbol # is used to stand for any delimiter (a space,] or #).

5. The symbol $stringI is used to indicate "get the contents of string I, and treat it as a name and get its contents" (as in SNOBOL).

6. Pattern-matching statements work just like SNOBOL statements: there are a) a name, b) a sequence of objects to be found in the named string in the order specified, c) the equal sign (meaning replace), and d) a replacement sequence of objects (b, c, and/or d can be absent). that string I means "get that particular object" - otherwise a new string is defined as the contents of stringI, which is taken to be a variable name.

REFERENCES

Andrews, D. R., Atrubin, A. J., and Hu, K. The IBM 1975 optical page reader: Part III: Recognition and logic development, IBM J. Res. and Devel., 1968, 12, 364-372.

Ausherman, D. A., Dwyer, S. J., and Lodwick, G. S. Extraction of connected edges from knee radiographs. IEEE Trans. Comput., 1972, 21, 753-758.

Brice, C. R. and Fennema, C. L. Scene analysis using regions. Artificial Intelligence, 1970, 1, 205-226.

Eden, M. Handwriting generation and recognition. In: Recognizing Patterns (P. A. Kolers and M. Eden, Eds.) Cambridge: MIT Press, 1968.

Firschein, O. and Fischler, M. A. Describing and abstracting pictorial structures. Pattern Recognition, 1971, 3, 421-443.

Fischler, M. A. Machine perception and description of pictorial data. Proc. 1st Joint Int. Conf. Artificial Intell., Washington, D. C., 1969, 629-635.

Forgie, J. W. and Forgie, C. D. Results obtained from a vowel recognition computer program. J. Acoust. Soc. Amer., 1959, 31, 1480-1489.

Frishkopf, L. S. and Harmon, L. D. Machine reading of cursive script. In: Proc. 4th London Symposium on Information Theory (C. Cherry, Ed.). London: Butterworth, 1961, 287-299.

Fu, K. S. and Swain, P. H. On syntactic pattern recognition. In: Software Engineering, Vol. 2 (J.T. Tou, Ed.) New York: Academic Press, 1971.

Grimsdale, R. L., Sumner, F. H., Tunis, C. J., and Kilburn, T. A system for the automatic recognition of patterns. Proc. Inst. Elect. Engrs., 1959, 106 (pt. B), 210-221.

Guzman, A. Decomposition of a visual scene into three-dimensional bodies. Proc. AFIPS FJCC, 1968, 33, 291-304.

Hall, D. J., Endlich, R. M., Wolf, D. E., and Brian, A. E. Objective methods for registering landmarks and determining cloud motions from satellite data. IEEE Trans. Comput., 1972, 21, 768-776.

Hall, E. L., Kruger, R. P., Dwyer, S. J., Hall, D. L., McLaren, R. W., and Lodwick, G. S. A survey of preprocessing and feature extraction techniques for radiographic images. IEEE Trans. Comput., 1971, 20, 1032-1044.

Hunt, E. B. Concept Formation: An Information Processing Approach. New York: Wiley, 1962.

Kirsch, R. Computer interpretation of English text and picture patterns. IEEE Trans. Comput., 1964, 13, 363-376.

Kochen, M. Experimental study of 'hypothesis-formation' by computer. Trans. 4th Sympos. on Info. Theory (C. Cherry, Ed.). London: Butterworth, 1960.

Ledley, R. S. Analysis of cells. IEEE Trans. Comput., 1972, 21, 740-753.

Ledley, R. S. and Ruddle, F. H. Chromosome analysis by computer. Sci. Amer., 1965, 40, 40-46.

Lillestrand, R. L. Techniques for change detection. IEEE Trans. Comput., 1972, 21, 654-659.

Lipkin, B. and Rosenfeld, A. Picture Processing and Psychopictorics. New York: Academic Press, 1970.

Lipkin, L., Watt, W. and Kirsch, R. The analysis, synthesis, and description of biological images. Ann. N.Y. Acad. Sci., 1966, 128, 984-1012.

Londé, D. and Simmons, R. NAMER: A pattern-recognition system for generating sentences about relations between line drawings. Proc. ACM 20th National Conf., 1965, 162-175.

Marill, T., Hartley, A. K., Evans, T. G., Bloom, B. H., Park, D.M.R., Hart, T. P., and Darley, D. L. CYCLOPS-1: A second-generation recognition system. Proc. AFIPS FJCC, 1963, 24, 27-34.

Mermelstein, P. Computer Recognition of Connected Handwritten Words. Unpubl. Sc.D. Thesis, MIT, 1964.

Narasimhan, R. Syntax-directed interpretation of classes of pictures. Comm. ACM, 1966, 9, 166-173.

Narasimhan, R. and Reddy, V.S.N. A syntax-aided recognition scheme for handprinted English letters. Pattern Recognition, 1971, 3, 345-362.

Reddy, D. R. Computer recognition of connected speech. J. Acoust. Soc. Amer., 1967, 42, 329-347.

Roberts, L. Machine perception of three-dimensional solids. In: Optical and Electro-optical Information Processing (J. T. Tippett et.al., Eds.), Cambridge: MIT Press, 1965.

Rosenfeld, A. Picture Processing by Computer. New York: Academic Press, 1969.

Sauvain, R. and Uhr, L. A teachable pattern describing and recognizing program. Pattern Recognition, 1969, 1, 219-232.

Shaw, A. C. A formal picture description scheme as a basis for picture processing systems. Info. and Control, 1969, 14, 9-52.

Smith, E. A. and Phillips, D. R. Automated cloud tracking using precisely aligned digital ATS pictures. IEEE Trans. Comput., 1972, 21, 715-730.

Sutton, R. N. and Hall, E. L. Texture measures for automatic classification of pulmonary disease. IEEE Trans. Comput., 1972, 21, 667-676.

Towster, E. Studies in Concept Formation, Unpubl. Ph.D. Diss., Univ. of Wis., 1969.

Uhr, L. A Primer for EASEy (An Encoder for Algorithmic, Syntactic English that's easey. Computer Sciences Dept. Tech. Report, Univ. of Wis., 1973a.

Uhr, L. Feature discovery and pattern description. In: Pattern Recognition (L. Kanal, Ed.). Washington: Thompson, 1968, 159-181.

Uhr, L. Flexible linguistic pattern recognition. Pattern Recognition, 1971, 3, 363-384.

Uhr, L. Flexible pattern recognition. Computer Sciences Department Technical Report 56, Univ. of Wis., 1969.

Uhr, L. Layered "recognition cone" networks that preprocess, classify and describe. IEEE Trans. Comput., 1972, 21, 758-768.

Uhr, L. Pattern Recognition, Learning and Thought. Englewood-Cliffs: Prentice-Hall, 1973b.

Uhr, L. and Vossler, C. A pattern recognition program that generates, evaluates and adjusts its own operators. In: Computers and Thought (E. Feigenbaum and J. Feldman, Eds.). New York: McGraw-Hill, 1963, 251-269.

Zobrist, A. L. The organization of extracted features for pattern recognition. Pattern Recognition, 1971, 3, 23-30.