MASSACHUSETTS INSTITUTE OF TECHNOLOGY

ARTIFICIAL INTELLIGENCE LABORATORY

Artificial Intelligence
Memo No. 329

May 1975

PARSING INTENSITY PROFILES

Tomas Lozano-Perez

ABSTRACT

Much low-level vision work in AI deals with one-dimensional intensity profiles. This paper describes PROPAR, a system that allows a convenient and uniform mechanism for recognizing such profiles. PROPAR is a modified Augmented Transition Networks parser. The grammar used by the parser serves to describe and label the set of acceptable profiles. The input to the parser are descriptions of segments of a piecewise linear approximation to an intensity profile. A sample grammar is presented and the results discussed.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under the Office of Naval Research contract N00014-75-C-0643.

I. INTRODUCTION

A significant part of the AI work on low-level vision has dealt with the examination of one dimensional intensity profiles [1,2,3]. I will avoid the issue of the suitability of this approach to low level vision. Rather I will take for granted that the analysis of intensity profiles along rays or circles on the image is a useful tool in Vision's repertoire of techniques. This paper suggests a mechanism for dealing with these profiles in a convenient and somewhat uniform fashion.

This paper is an "intellectual descendant" of my previous working paper [3], in the sense that it addresses some of the issues that I "discovered" in the process of implementing the program described there. Thus, this is an attempt to solve a problem that is not immediately apparent as a hard problem, namely classification and segmentation of one dimensional profiles. The programs described in [3] attempted to characterize the visual structure of electronic components in terms of the spatial relationships between several types of intensity profiles. This kind of characterization, although quite effective, proved to be cumbersome in terms of the sheer amount and ad-hoc nature of the code needed to embody it. The immediate goal of the new system, PROPAR, is to alleviate these problems by presenting a convenient interface to the high level description of intensity profiles. I believe that the system has some other desirable properties, and I will bring these up as they become evident.

PROPAR is essentially an Augmented Transition Networks [4] parser that has been modified to deal with intensity profiles. The grammar used by the parser serves to describe and label the set of acceptable profiles as a sequence of input "words" The "words" that serve as input to the parser are descriptions of segments of a piecewise linear approximation to the intensity profile.

II. ATN GRAMMARS

Augmented Transition Network grammars are an extension of Non-Deterministic Finite State Grammars. The extensions are basically as follows:

1) Arc tests: Associated with each state transition arc is one or more tests that determines whether that state transition is to be made.
2) Arc actions: ATN grammars also allow specification of arbitrary actions to take place once the arc is taken. This allows for side effects such as the construction of a parse tree. All arcs specify what the next state should be and whether to advance the input. The actions TO and JUMP perform these tasks.
3) Registers: The primary type of arc action is the setting of registers whose values can be manipulated and tested on the arcs.
4) Recursion: ATN also allows special transition arcs labelled PUSH whose effect is to save the current state, the register contents and the input pointer. It also transfers control to the state specified on the arc without advancing the input. Arcs labelled POP will resume the parse at the place of the last PUSH. Communication between levels is effected by means of the registers. A SENDR action sets the value of a register which is transmitted

to the next state PUSH'ed to. Another arc act, LIFTR, serves to set a register in the state to be POP'ed to. A POP arc always returns a value which gets bound to a special register called **.

An ATN grammar consists of a list of states, each of these consists of a list of arcs emanating from the state. There are 4 types of arcs:

1) CAT: each input word belongs to a category, this arc is considered if the category of the input matches that specified on the arc. The format is as follows:

   (CAT <category> <test> <list of actions>)

   If the category of the input matches <category> and <test> evaluates to non-NIL the list of actions specified is to be evaluated.

2) TST: this type of arc allows for arbitrary tests.

   (TST <test> <list of actions>)

   The list of actions is executed iff <test> EVAL's to non-NIL.

3) PUSH: this type of arc implements "subroutine calls"

   (PUSH <state> <test> <list of actions>)

   If <test> evaluates to non-NIL the PUSH action is executed, upon return the arc actions are executed with the special register ** bound to the result of the PUSH arc, i.e. the value of the POP arc.

4) POP: this type of arc effects a return from a PUSH

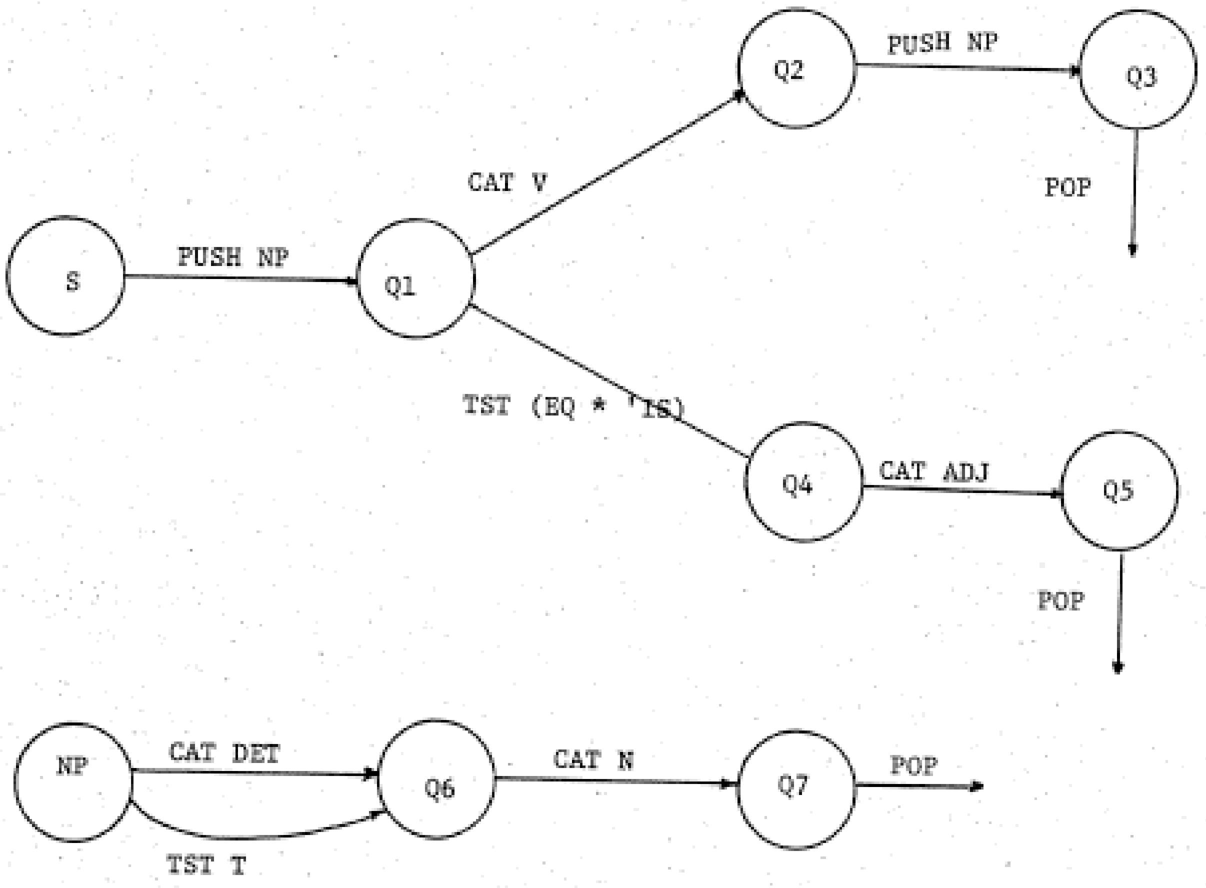
   (POP <value> <test>)

   If <test> evaluates to non-NIL the register ** is set to the value of <value> and execution resumes where last interrupted by a PUSH.

The actions on the arcs can be arbitrary LISP forms although there are a few special ones:

1) (SETR <register> <value>) sets a register.
2) (GETR <register>) returns the value of the register.
3) (SENDR <register> <value>) sets a register in the next state PUSH'ed to.
4) (LIFTR <register> <value>) sets a register in the next state POP'ed to.
5) (GETF <feature>) get the value of an input feature
6) (TO <state>) indicates that a transition to <state> is to be made and that the input is to be advanced (ususally the last action on an arc).
7) (JUMP <state>) similar to TO but does not advance the input.

An example might clear some of this up. The simple grammar shown in FIG 1 "accepts" two kinds of sentences; those of the form "NP V NP" or "NP is ADJ". The * is a special register which holds the input word. The first entry of each of the following lists is the state name, it is followed by one or more arcs.



```
(S (PUSH NP T (SETR SUBJ **) (TO Q1)))
(Q1 (TST (EQUAL * 'IS) (TO Q4))
    (CAT V T (SETR V **) (TO Q2)))
(Q2 (PUSH NP T (SETR OBJ **) (TO Q3))
(Q3 (POP (LIST (GETR SUBJ) (GETR V) (GETR OBJ)) T))
(Q4 (CAT ADJ T (SETR ADJ *) (TO Q5)))
(Q5 (POP (LIST (GETR SUBJ) 'IS (GETR ADJ)) T))
(NP (CAT DET T (SETR DET *) (TO Q6))
    (TST T (JUMP Q6)))
(Q6 (CAT N T  (SETR N *) (TO Q7)))
(Q7 (POP (LIST (GETR DET) (GETR N)) T))
```

FIG 1: A sample grammar

The parser simulates the non-determinism of the grammar by following the elegible arcs in order of their appearance on the state. When a state is reached in which no arcs are elegible, the parser backs up to the last point where there is an unexplored arc which is elegible.

The input "words" used by PROPAR are segments of a piecewise linear approximation of the intensity profile along a ray on the image. The category of a segment is its slope. CAT arcs have therefore been extended so that they not only do an EQUAL test if the category on the arc is an atom, but they can also test if it lies within specified bounds indicated on the arc by a list of length 2. For example:

```
(CAT (1. 3.) T (SETR UP *) (TO DOWN))
```

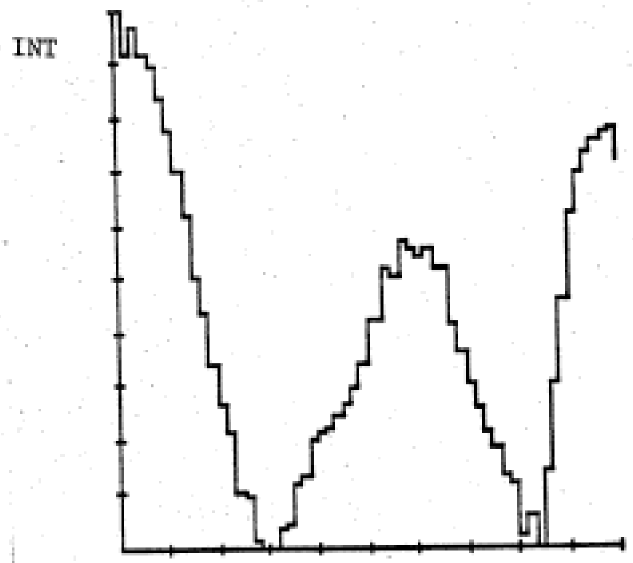
The grammar treats the input pointer, i.e. the place and direction from which input is to be taken, specially. Whenever a PUSH arc is taken, a SENDR of the input pointer is automatically done. This serves to inform that arc of the place in the image where the next input is to come from. A POP arc always does a hidden LIFTR of the input pointer to inform the higher level of where the input has advanced to, this is necessary because before doing a PUSH all the register values are saved and then restored when the state POP's. Treating the input as a register allows automatic backup of the input to happen when a PUSH fails. Both the hidden SENDR and LIFTR of the input pointer can be superseded by explicit actions provided by the parser, SENDPTR and LIFTPTR. The nature of the input function is described in the next section.

III. Characterizing the Input

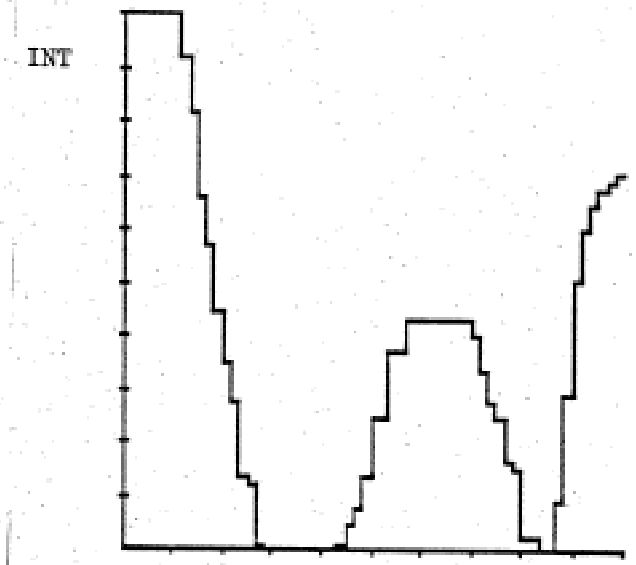
A large measure of the benefit to be gained from the syntactic approach to image analysis stems from the structure it imposes on the input. The first step in any such system is the isolation of the "words" to serve as input. This usually involves segmentation of the input in either the spatial or time domain. PROPAR uses as an input domain a piecewise linear approximation to the smoothed intensity. The segments are represented by their slopes quantized into a small number of integral values (currently nine). The input is taken from a point and direction specified by the grammar.

The input waveform is smoothed by a hysterisis smoothing function [5], FIG. 2 shows an example. The function moves along an input ray and returns the value of the input iff it its magnitude is sufficiently greater than the magnitude of the last point examined. Otherwise, it returns the value of the previous input point. This function has several distinct advantages, it removes low amplitude noise without throwing away the high frequencies as averaging does. Moreover, the nature of its effect on the waveform is easily predictable since it returns stretches of constant intensity whenever a variation in the input is below a given threshold determined by the noise level of the input device.

The input profile is segmented at places where the slope quantization assigned to the slope between two points differs from the quantization assigned to the slope between the previous points. The function returns the slope quantization, the coordinates of the first and last points and



a. The raw data



b. Smoothed data

FIG. 2- Hysterisis Smoothing

the intensities there. These can be accessed as features of the input word by GETF. The features are called CAT, PT1, PT2, I1 and I2 respectively. FIG 3 shows an example of an input profile. FIG 3a shows the raw intensity data, 3b shows the smoothed intensity. Figures 3a and 3b are very similar because the data does not have many small fluctuations . Figure 3c shows the segmented input with the slope quantizations. Notice that a significant data compression has been achieved, while the overall shape remains unchanged.

Several issues are relevant to the characterization of the input. The first is " How good does the characterization have to be?" Obviously, this depends on the distinctions the grammar is capable of making. In the kind of processing I've done using PROPAR, only the rough shape of the input was of interest and thus a very trivial segmentation algorithm was chosen. There are several applications of profile techniques that depend on accurate measurements to be taken on the input rather than on the determination of its general form. For such domains this input characterization is inadequate. Some examples of such applications are precise edge location [1,2] and cardiogram anlysis [6,7]. A fairly large literature [8,9,10] exists on approximation methods for waveforms and any such algorithm might be used in conjunction with the system.

The other and more interesting question is whether the expectations of the grammar can influence the type of processing done on the input. Reference [6] makes a strong and quite valid point that unless this is done the system is not truly a syntactic analysis system but rather a data-compresssion algorithm. The reason being that in general syntactic analysis the aggregation of basic units into constituents depends on the state of the parse as well as on the syntactic class of the basic units. I think the merits of data compression should not be belittled, but I agree with the essence of their argument. Their solution was to define an input function for each of the members of the set of primitive constituents, e.g. a parabola. PROPAR, on the other hand, is based on the principle that as much of the input processing should be explicit as possible. The grammar designer should have control of the processing to any desired level of detail. Theoretically the grammar should look at the input on a point by point basis. This is workable in the ATN grammar because the PUSH arcs allow one to structure the grammar hierarchically. The detail can be added at one level without cluttering the grammar at all levels. The only reason for not going to this extreme are efficiency considerations, namely size of the grammar and speed of execution.

PROPAR is compatible with two extra-grammatical mechanisms for influencing the input process. One is to define several LISP functions that look for the particular features of interest as in [6]. These functions can be called by a PUSH arc since states are treated uniformly as programs that are called with the current contents of the registers as argument. Another is to change the input function as the need arises. The input function used is determined by the value of a register and is thus under control of the grammar. It is important to remember that improving the quality of the representation is only useful when the grammar can make use of the improvement. So far I have found the uniform representation both adequate and efficient.

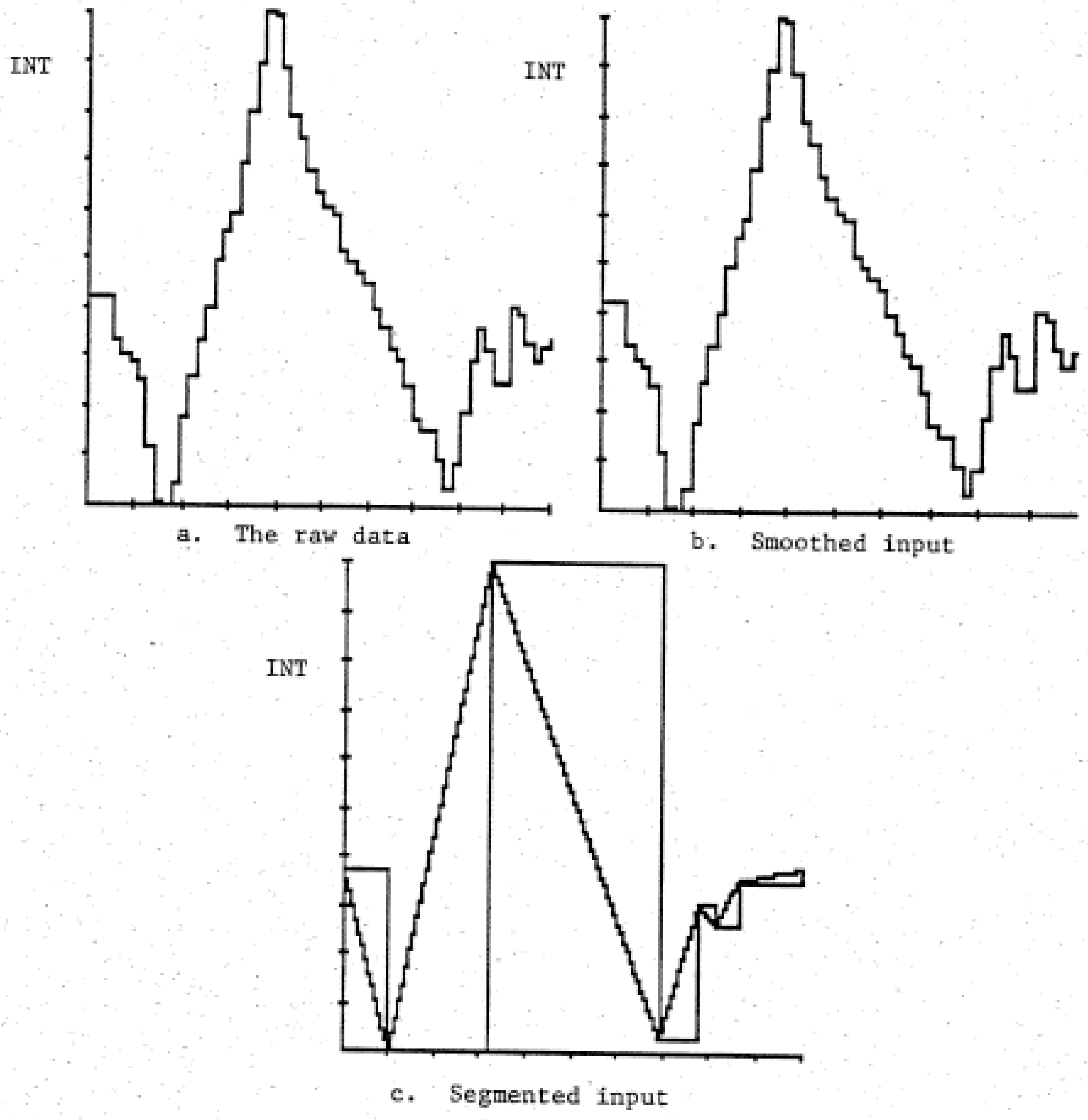


FIG. 3 - Input processing example

IV. A Sample Grammar

What follows is a grammar that attempts to incorporate the basic features of profiles of cylindrical objects with one highlight, namely that the intensity decreases towards the edges of the object. The CYL grammar is shown on FIG 4. This grammar is built up out of PUSH'es to other networks (also shown on FIG 4), STEP+, STEP- and HOLE. The STEP+ network succeeds if the input slope is positive and of sufficient length it also advances the input pointer to the next local maximum in the profile. STEP- does the same thing for negative slopes and HOLE accepts a STEP- followed by a STEP+. These networks will be discussed later.

```
(CYL (TST T (SETR START (GETF PT1)) (JUMP CYL-UP)))
(CYL-UP (PUSH HOLE T (SETR UP **) (JUMP CYL-DOWN))
        (PUSH STEP+  T (SETR UP **) (JUMP CYL-DOWN)))
(CYL-DOWN (PUSH HOLE T (SETR DOWN **) (JUMP CYL!))
          (PUSH STEP- T (SETR DOWN **) (JUMP CYL!)))
(CYL! (POP (LIST 'CYL (GETR START) (GETF PT1) (GETR UP) (GETR DOWN)) T))
```

State CYL just sets the register START to the value of the first point of the current input. CYL-UP sets the register UP either to the value returned by a call to HOLE or to STEP+. CYL-DOWN does the same for register DOWN except that the second choice is STEP-. CYL! just returns a list built by the POP arc which contains the beginning and end points of the profile as well as the values returned by the HOLE or STEP+ and STEP- networks. A PUSH to the state CYL can cause one of two things to happen a) the grammar fails to accept the input, meaning that the current input segment does not begin a legal CYL profile, or b) a CYL profile is returned. This very simple grammar accepts intensity profiles belonging to one of the four classes shown in FIG 5 on the next page. FIG 6 shows the results of the grammar on a typical profile.

A useful extension of this grammar would be a way to examine a specified length of the input sequentially until a CYL profile was found. These three states do precisely that for the CYL grammar.

```
(FIND-CYL (PUSH CYL  T (SETR CYL **) (JUMP FOUND-CYL))
          (TST T (SETR STATE 'FIND-CYL) (TO SEARCH)))
(SEARCH (TST (NOT (BEYOND (GETF PT2) (GETR END-SEARCH)))
             (JUMP (GETR STATE))))
(FOUND-CYL (POP (GETR CYL) T))
```

FIND-CYL does a PUSH to CYL and if it suceeds just returns it. If the PUSH fails then the register STATE is set to the state to go to if the point where the search is to end (END-SEARCH) has not been exceeded. SEARCH checks that the input is within the desired boundary and then either JUMP's to the state specified by the register STATE or fails.

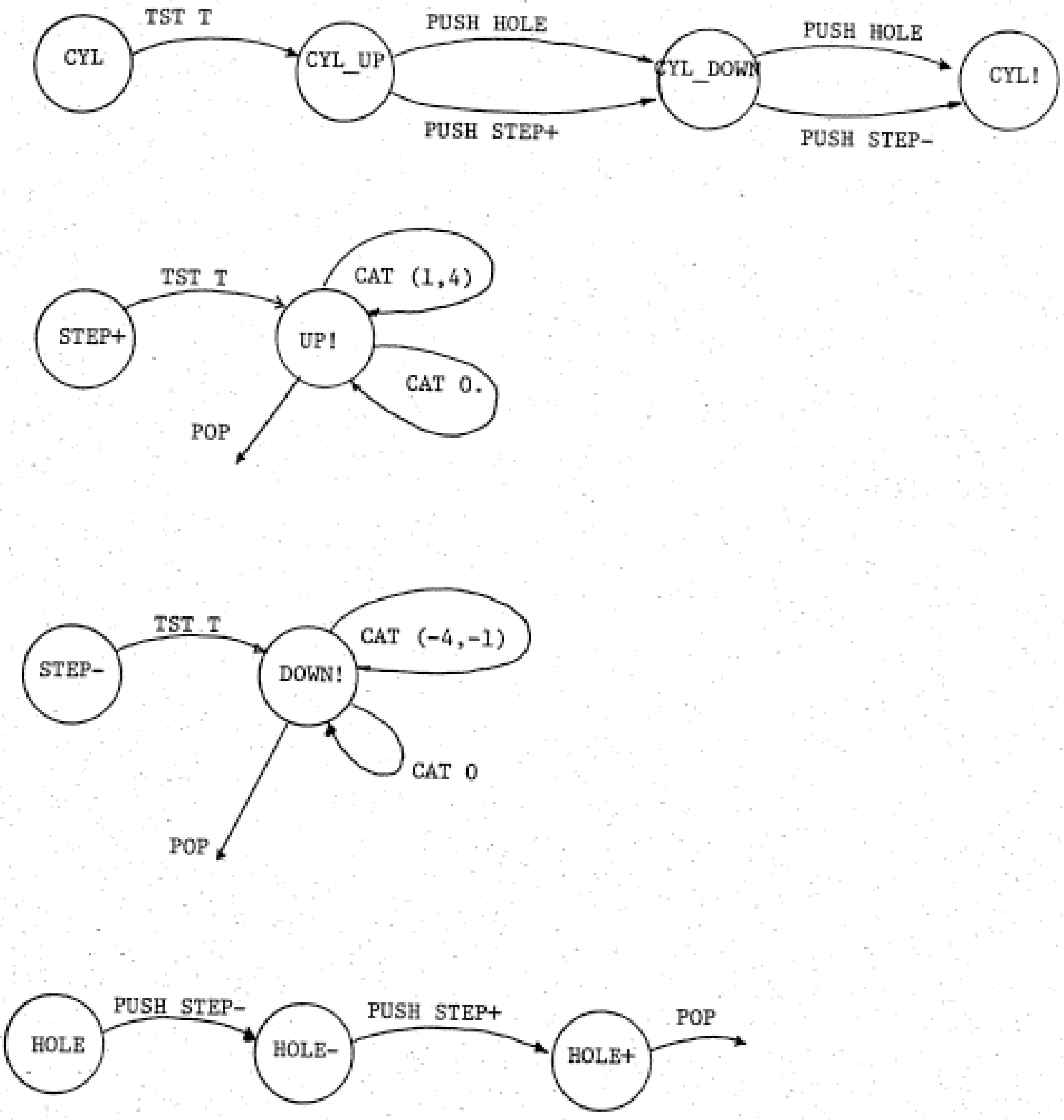


FIG. 4 - The CYL grammar

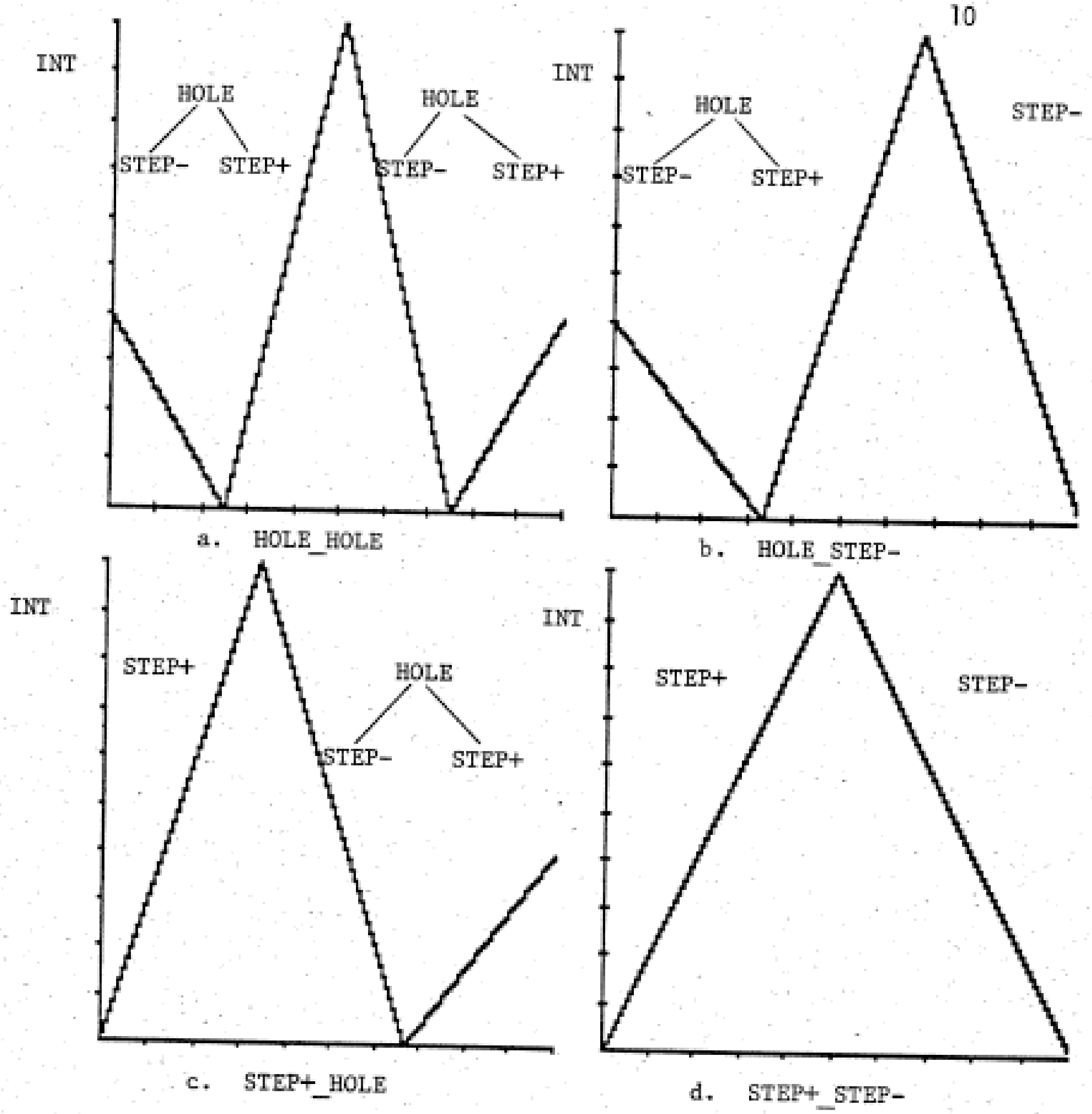


FIG 5. - Four kinds of CYL profiles

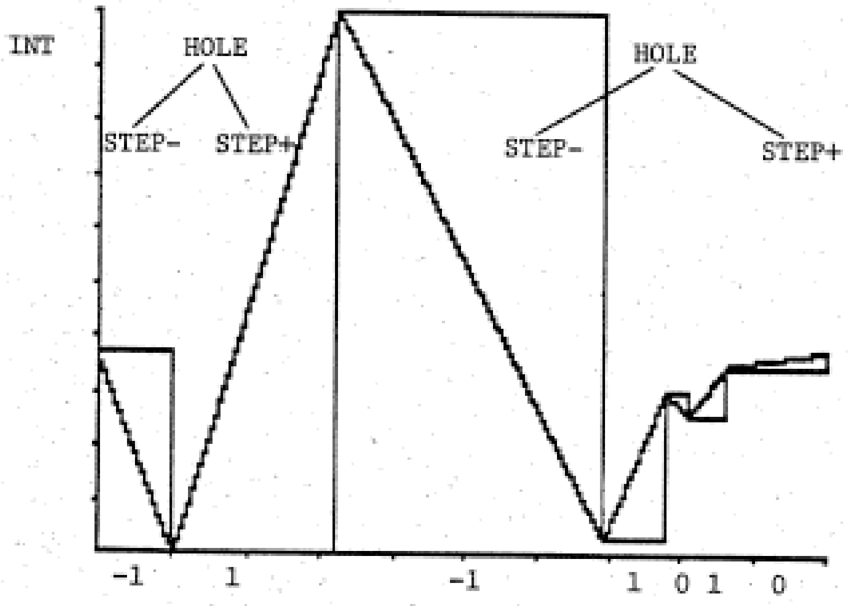


FIG. 6 - Segmented input and Parsing

What follows is the STEP+ network. The method embodied in this network is very crude but it will do for our purposes. In more demanding applications, more elaboration is needed. For instance, this network ignores the actual value of the slope. If one wanted to locate the edges with any degree of accuracy the grammar and possibly the input function would have to be changed. In the examples shown in the appendix one can notice that the position of the edges is not very accurate. I find it somewhat surprising that this simple network works at all.

```
(STEP+ (TST T (SETR START (GETF PT1)) (JUMP UP!)))
(UP! (CAT (1. 4.)  T
          (SETR P (MERGE (GETR P) *))
          (SETR TIP+ (GETF PT2))
          (TO UP!))
     (CAT 0. (LESSP (GETF RUN) SIGTHR)
          (SETR P (MERGE (GETR P) *)))
          (TO UP!))
     (POP (LIST 'STEP+ (GETR START) (GETF PT1) (GETR P))
          (AND (GETR TIP+) (GREATERP (CADDR (GETR P)) SIGTHR))))
```

State UP! accepts sequences of segments whose quantized slope is greater than 0 and segments of slope equal to 0 whose length exceeds SIGTHR. Register P will be a list of the form (SLOPE RISE RUN). The run is the length of the sequence, the rise is the intensity change along the sequence and the slope is the quantization belonging to the quotient of the rise and the run. The function MERGE constructs a new value of P from the old value and the input. Register TIP+ contains the local maxima of the profile so far. The network for STEP- exactly parallels that for STEP+ and the HOLE network just consists of a PUSH to STEP- followed by a PUSH to STEP+.

It should be fairly clear that this grammar is quite general. It has, in fact, been used succesfully to analyze two very different scenes, one of a hammer handle and the other of resistors. The appendix describes these applications.

V. Extensions

The system as described so far fulfills its original goal of providing a convenient descriptive mechanism for one dimensional profiles. After the research reported here had been carried out, a similar effort [6,7] was pointed out to me. These excellent papers describe a system very similar in style to PROPAR applied to the parsing of EKG waveforms. This system focused on several very important issues which I had neglected, namely interaction of the parsing process with the input process and non left-to-right parsing. Their choice of grammar (extended BNF) forced them to introduce extra-grammatical mechanisms to deal with these extensions. I think it indicative of a right choice of grammar for PROPAR that both these mechanisms could be incorporated explicitly in the grammar just as the extension to two dimensions can also be made explicit. This section deals with these three extensions and their embodiment in PROPAR.

A. The other dimension

Real images are two dimensional and the key to their structure can seldom be unlocked from the analysis of one dimensional profiles. Profiles along a ray are typically ambiguous. It is very hard to differentiate meaningless bumps from the significant features. The principal cue to significance is coherence in two dimensions. PROPAR approaches this by means of a simple extension, namely several profiles. Once a particular type of profile has been recognized, the grammar can use the description returned to suggest where to look for other profiles of known types. This serves to characterize two dimensional profiles. The simplest form of this technique is using the same grammar along parallel rays. In this form PROPAR behaves like a generalized tracker such as has been used in line finders [1]. This extension to the CYL grammar was carried out and the results are described in Appendix 1. The obvious next step is profiles along non-parallel rays and this is also demonstrated in Appendix 1.

B. Non left-to-right parsing

In waveforms such as intensity profiles, speech signals or EKG's where noise is prevalent, the commitment to left to right parsing significantly increases the probability of failure. Several people have recently emphasized this fact [7,11]. What one might want is to find significant features first and then look harder, or at least differently, in the contexts defined by these features for other features that bear the desired relationships. This is not as obscure as it sounds, especially in one dimension. PROPAR already has all the right mechanisms needed for this. Let us suppose we wanted to pull out of a scene the flat area between two CYL profiles that might be very close together (maybe the board region between two resistors). This might be very easy to miss because of the crudeness of the states UP! and DOWN!. Let us postulate a state FLAT! which looks at a segment of the profile in a special way and determines whether it is flat. All we need to tell it is where to START by sending it the value of the INPUT-PTR at the end of the first CYL profile and where to END, namely the beginning of the second CYL profile i.e. (CADR **).

```
(CYL1 (PUSH FIND-CYL T (SETR START (GETR INPUT-PTR)) (JUMP CYL2)))
(CYL2 (PUSH FIND-CYL T (SENDR END (CADR **))
                 (SENDPTR (GETR START))(JUMP FLAT)))
(FLAT (PUSH FLAT! T (SETR FLAT **) (JUMP BACK)))
(BACK (POP (GETR FLAT) T))
```

This is a perfectly general mechanism and I believe it to be desirable that such mechanisms be under explicit control of the grammar designer.

VI. Conclusions

In this section I'd like to step back and ask some global questions about the work and its implications. The following issues seem important:

A. The choice of the method: Why parsing?

B. The choice of the tools: Why an ATN parser?

C. The limitations of the method.

Let us examine these in turn.

A. The choice of the method

Parsing is a useful approach to the representation of one dimensional patterns because of all the wrong reasons. Its usefulness stems from the ability to specify legal subsequences in the grammar and essentially ignore all the combinatorics of the interactions by relegating them to the uniform workings of a parsing algorithm. This, of course, makes all the arguments relevant to the choice between LOGIC, PLANNER and CONNIVER [12] relevant to the decision of using parsing as a method. I think the progression from Transformational Grammars to ATN and PROGRAMMAR to Marcus's WASP [13] reflects a concern with the same issues: mainly the inefficiencies of uniform strategies. On the other hand, the attractivenes of uniformity of the PLANNER and ATN variety for limited domains cannot be disputed. One can only make use of more control over processes when one knows more about diagnosis of the interactions; in our case that knowledge is missing and the convenience of writing simple grammars is worth the price of some extra combinatorics.

B. The choice of the tools

Once we have settled on parsing as an approach to the problem, the question remains of which type of grammar is adequate. Adequacy has two dimensions, sufficiency and necessity. The sufficiency need not be argued with respect to the choice of ATN as a tool except as regards the matter of uniform control structure. I believe that, at least for limited applications, the ease of representation is worth the price one pays in loss of direct control. I have to qualify that claim enough to include the possibility that the control structure <u>prevents</u> the solution of the problem. If the size of the class of acceptable profiles grows very large this latter might be the case. The limited number of examples I've currently tried tend to suggest that this is not the case of the uses the system was designed for, namely as a flexible verification system. It is possible, although this merits another paper, to design the grammars to reduce the amount of unguided backup that needs to be done, something which Woods seldom tried to do in his English grammar. The fact that the Woods' LUNAR system was feasible even with no considerations of guided backup seems to me to suggest quite strongly that the control structure problem will be minor at the level PROPAR is meant to operate. I don't believe that the kind of analysis PROPAR does on one dimensional intensity profiles is more complex than the English domain of LUNAR. General two dimensional analysis can be quite explosive, except that PROPAR, by relying on two dimensional coherence, effectively decouples most of the interactions between dimensions. This fact is a source of one of the major limitations of the system but, on the other hand, it reduces the complexity of the problem to approximately the complexity of the one dimensional problem since for any given profile in one dimension only a small number of profiles can be coherent with it in the other dimension, given reasonable smoothness assumptions.

Another important theoretical issue is the necessity aspect of adequacy. We would like the simplest system that the data forces us to accept. I think ATN grammars are justified in these terms given a measure of simplicity that is not just the number of lines of code. I believe that the ease of use should be weighted quite heavily, insofar as this is an experimental system ease of modification and understanding are crucial. Since the problem is not solved, generality is important. One would rather the solution is found before the inherent limitations of the system were reached. As an example of the case in point let us consider the "need" for recursion. I believe that the description of some intensity profiles is most economically captured by a recursive grammar, e.g. a bump on a bump on a bump ... Appendix 2 presents a more elaborate version of the STEP+ grammar which is recursive. It does not prove that recursion is required but I believe it suggests its convenience. On the other hand, there is an independently motivated reason for having subroutine calls, namely clarity and economy of specification. Once we have subroutine calls why not allow them to be recursive? The same line of argument follows for registers.

Considerations of efficiency always color our judgments of even experimental systems, so a word about it might be called for. PROPAR is a fairly modest size MACLISP program, compared to the typical AI MACLISP program. The system allows for compilation of the grammar into LISP programs that are then compilable. In this mode the system is reasonably fast, again compared to other LISP programs, and is mostly bounded by data access speed.

C. The limitations of the method

The major limitation of the use of the system is its dependence on "axes". The problem stems from the view of grammars as a passive representation of the legal sequences; remember that the automata implementing most types of grammars are known as accepters. The most straightforward, ergo useful, way of writing a profile grammar is to encode the profile obtained by scanning the object from a given direction, usually along the line of steepest slope. For most profiles this is quite sensitive to the direction of view, i.e the relationship between the input ray and the profile. It is thus fairly clear that these grammars works best as verifiers rather than as proposers. It is in this guise that PROPAR was used in the two examples discussed in the Appendix. I believe the system in this form to be quite useful as a tool for a more intelligent system such as Freuder's SEER [14].

The system has no inherent limitations that force it to be passive. The extensions to non-left-to-right and two dimensional parsing demonstrate this. The problem now becomes one of designing the right active grammars that decide from a given profile, not only whether it is one of the known views of a profile but, but also what actions should be taken otherwise. For instance, if the desired profile is not found, the grammar might try several scans along different angles from the starting point. With the CYL grammar this would help find profiles when the original scan was nearly parallel to the axis of the profile (the length of the cylinder) I think that PROPAR might be the right environment for the development and coding of such procedures. I can't rule out the possibility that the inherent limitations of the one dimensional approach force a failure before then.

ACKNOWLEDGEMENTS

I would like to thank (alpahabetically) Mike Dunlavey, Berthold Horn, Mark Lavin, David Marr, Patrick Winston, and Robert Woodham for their many comments and suggestions.

BIBLIOGRAPHY

1) Shirai,Y. " A context sensitive Line Finder for Recognition of Polyhedra", Chapter 3, The Psychology of Computer Vision, P. Winston (Ed.), McGraw Hill, 1975.

2) Herskovits, A., Binford, T. "On Boundary Detection" ,MITAIM 183, July 1970.

3) Lozano-Perez, T. "Finding Components on a Circuit Board", MITAI Working Paper 51, Sept. 1973.

4) Woods, W. "Transition Network grammars for Natural Language Parsing", CACM Vol 13, No. 10, Oct. 1970.

5) Duda, R., Hart, P. Pattern Classification and Scene Analysis, Wiley, 1973.

6) Stockman, Kanal, Kyle. "Design of a Waveform Parsing System", TR-266 U. of Maryland, Comp. Sci. Center, Oct. 1973.

7) Stockman and Kanal. "How to Parse a Waveform", Personal Copy.

8) Rice, J. The Approximation of Functions, Addison-Wesley, 1964.

9) Davis, P. Interpolation and Approximation, Blaisdell, 1963.

10) Pavlidis, T. "Waveform Segmentation Through Functional Approximation", IEEE Trans. on Computers Vol C-22 No 7, July 1973.

11) Miller, P. "A Locally Organized Parse for Spoken Input", CACM Vol 17 No 11, Nov. 1974.

12) Sussman and McDermott. "Why Conniving is Better than Planning", MITAIM 255, April 1972.

13) Marcus, M. "Wait and See Strategies for Parsing Natural Language". MITAI Working Paper 75, Aug. 1974.

14) Freuder, E. "Active Knowledge", MITAI Working Paper 53, Oct. 1973. Also forthcoming Ph.D dissertation.

Appendix 1: Results

This appendix describes some applications of the CYL grammar as presented in the body of the paper. In the first example the grammar was used to verify a hypothesis generated by the SEER [14] system. The scene under consideration, consisting of a hammer on a workbench, is shown on the next page (figure A1). The scan lines show where CYL profiles were found. PROPAR was used to "track" the hammer handle to completion. SEER hypothesized a starting point and a search direction and then called PROPAR at state FIND-CYL. The starting point and direction are shown on the picture and the parallel scans used by the system. There are several things to notice about the results. Near the end of the handle notice that the scans change direction. This is done by the tracking grammar, once the scan failed to produce a new CYL profile a line was fit to the points found so far. This line was sufficiently different from the line PROPAR had been following that it merited trying again in the new direction. This proved to be the case as the handle was then tracked to completion. This technique is useful for tracking objects that are supposed to be linear. Another point to notice is that the profiles along the handle (shown on figure A2) are quite different. The grammar was allowed to accept any profile that was of the right type without checking further. In the next example, the verification of resistors, this decision was reversed and the grammar compared the suceeding parallel profiles for similarity. Both these strategies are easily expressed in the system. Again, notice that the endpoints of the profiles do not correspond very well to the edges of the handle. In some of the profiles notice that the small STEP- profiles on the side of the main peak are missed, this happens because the grammar was set to ignore very short profiles. The parser returns a description of what it does find, i.e. the lists returned by the POP arcs, and this information can be used later to determine where to look if more detail was desired.

The second application is to resistors. Figure A3 shows the result of using the CYL grammar to verify a resistor. Once the parallel scans are completed and the endpoints of the resistors are known, a scan perpendicular to the previous scans is taken so as to pinpoint the bands. Figure A4 shows the intensity profile along the length of the resistor. This other grammar consists of a few states that essentially just PUSH to the state PEAK (a STEP+ followed by a STEP-). One also wants to use the scan that finds the bands to identify the ends of the resistor. Since the ends can either be illuminated or in shadow one needs a few extra states.

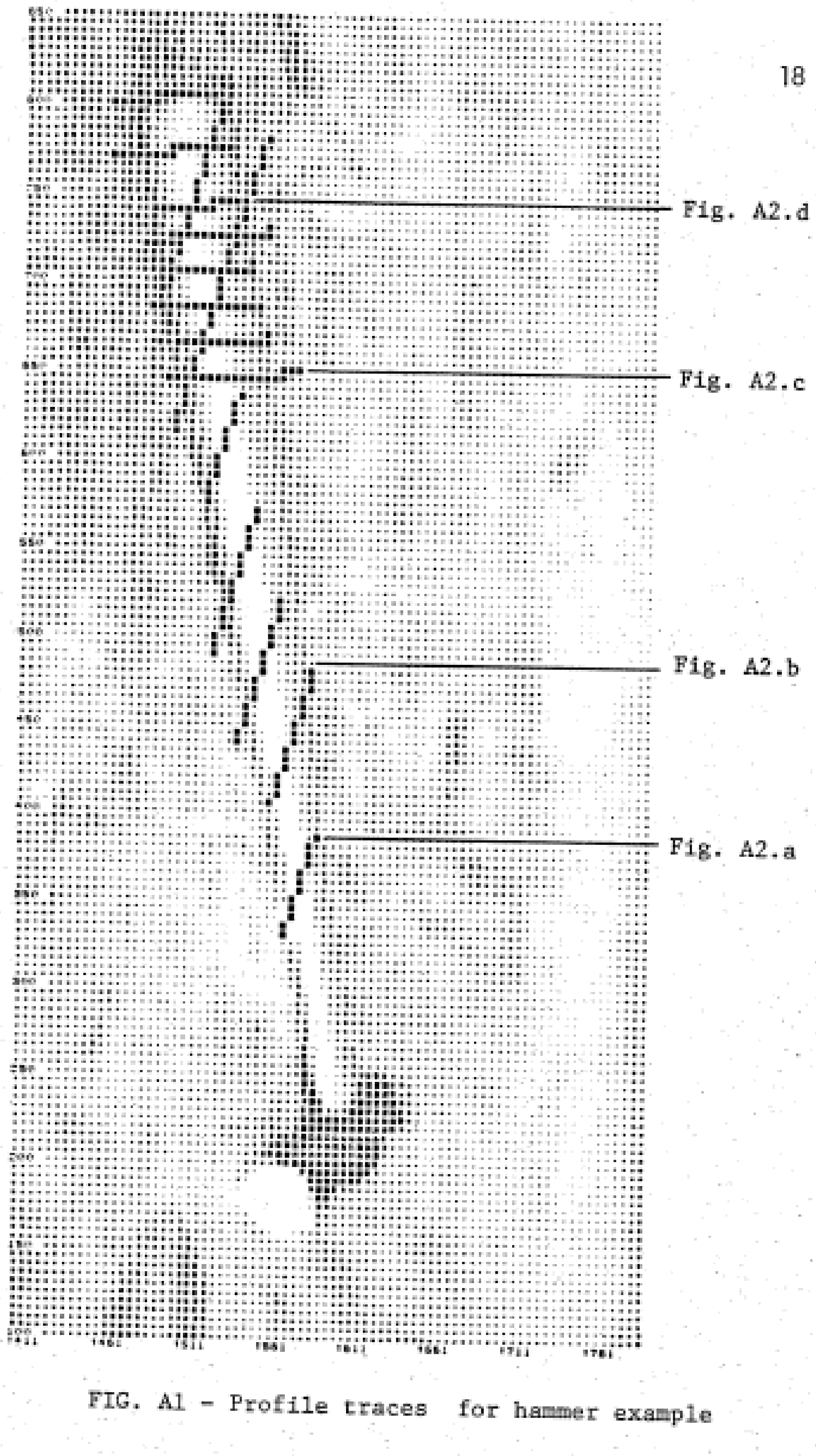


FIG. A1 - Profile traces for hammer example

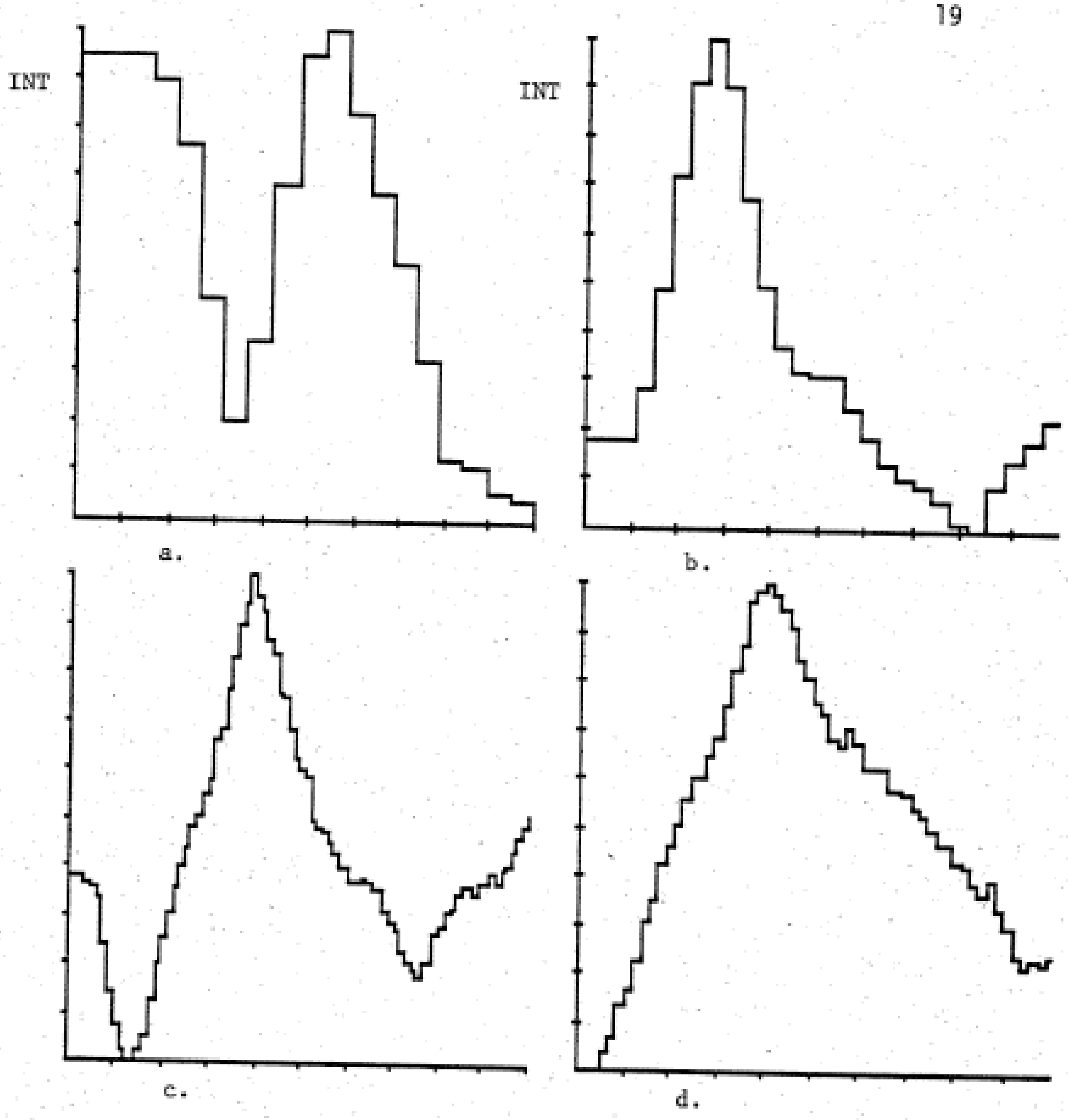


FIG. A2 - Raw intensity profiles across hammer handle

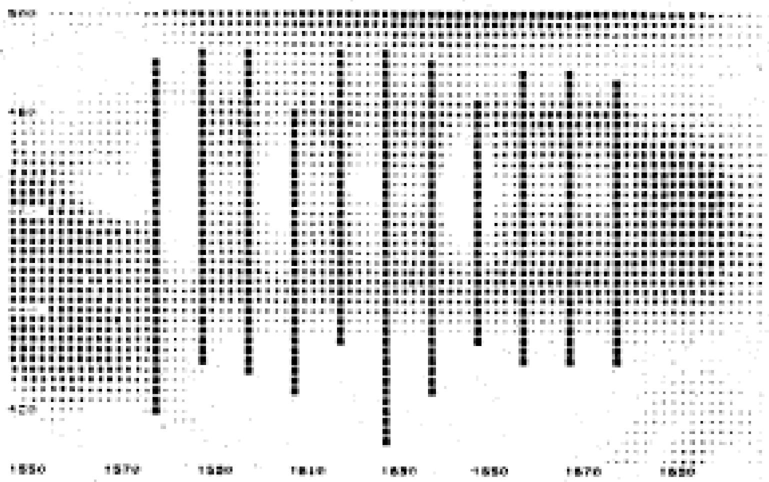


FIG. A3.a - Vertical profile traces
Resistor Example

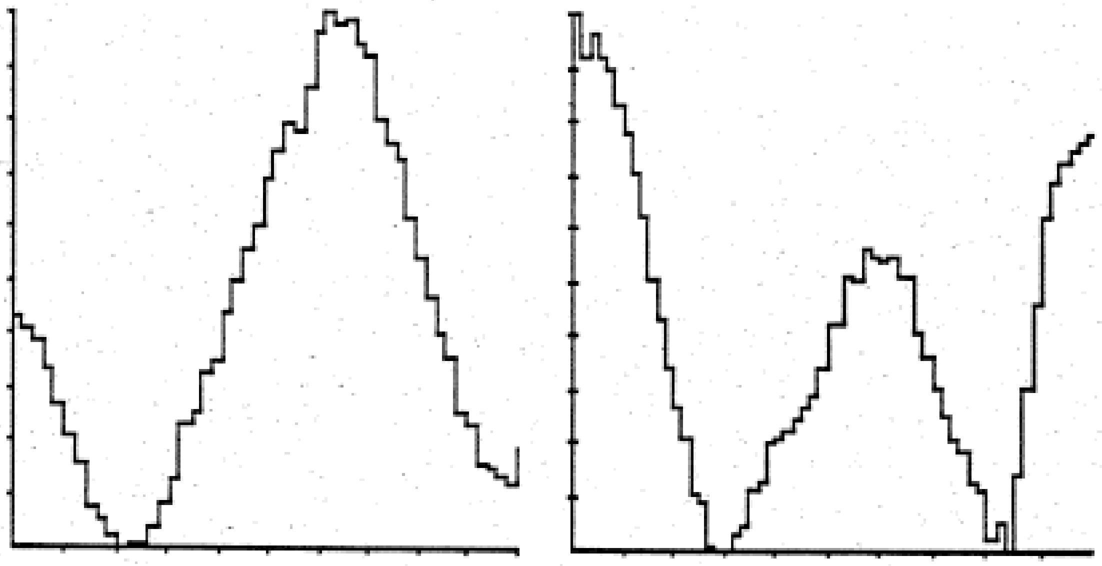


FIG A3.b - Raw intensity profiles

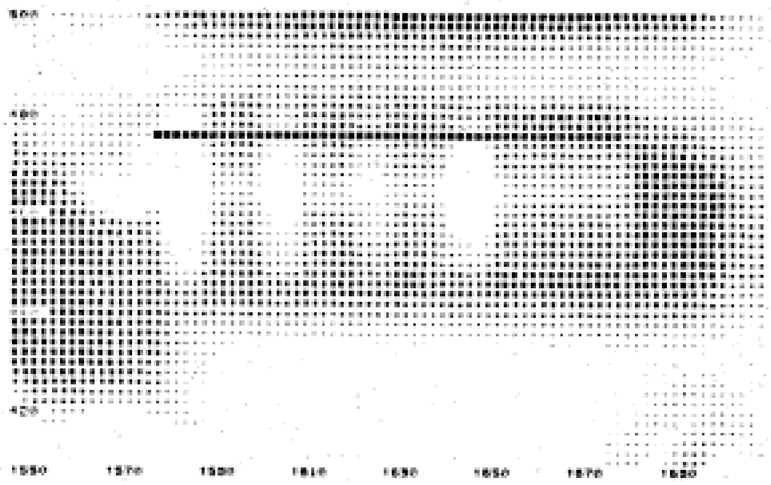


FIG. A4.a - Horizontal profile traces
Resistor Example

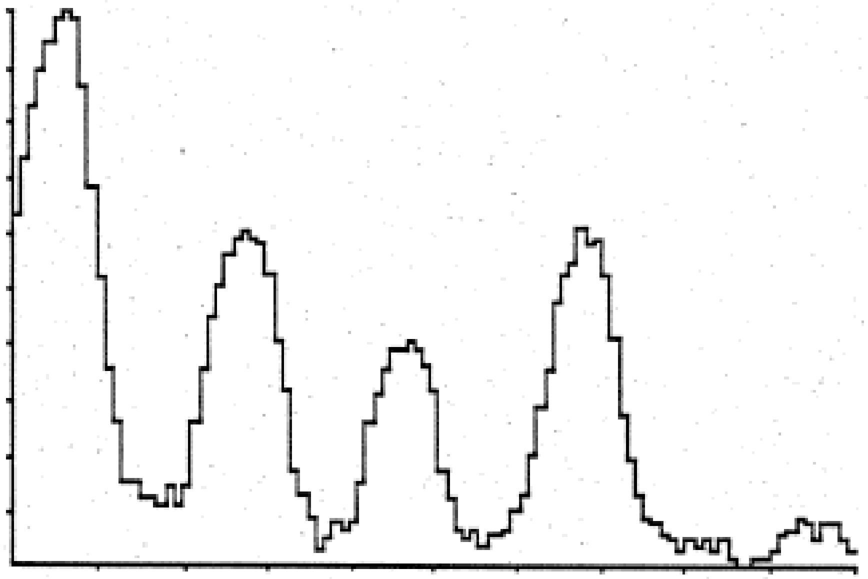


FIG. A4.b - Raw intensity profile

Appendix 2: A More complex STEP+ grammar

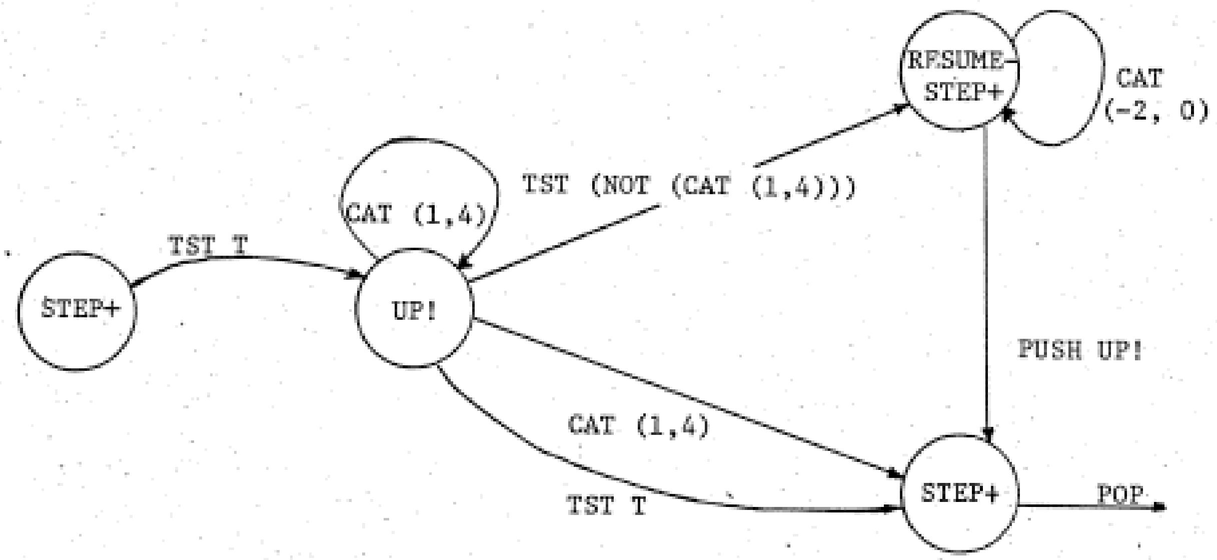
This appendix shows an example of a recursive grammar, this is the next step towards a really adequate STEP+ grammar.

```
(STEP+ (TST T (SETR START (GETF PT1)) (JUMP UP!)))

(UP! (CAT (1. 4.)
          T
          (SETR P (MERGE (GETR P) *))
          (SETR TIP+ (GETF PT2))
          (TO UP!))
     (TST (NOT (CAT (1. 4.) *)) (JUMP RESUME-STEP+))
     (CAT (1. 4.)
          T
          (SETR P (MERGE (GETR P) *))
          (SETR TIP+ (GETF PT2))
          (TO STEP+!))
     (TST T (JUMP STEP+!)))

(STEP+! (POP (LIST 'STEP+ (GETR START) (GETF PT1) (GETR P))
             (AND (GETR TIP+) (GREATERP (CADDR (GETR P)) SIGTHR))))

(RESUME-STEP+ (CAT (-2. 0.)
                   (LESSP (DISTANCE (OR (GETR TIP+) (GETR START))
                                    (GETF PT2))
                          FLATTHR)
                   (OR (GETR TIP+) (SETR START (GETF PT2)))
                   (TO RESUME-STEP+))
              (PUSH UP!
                    (AND (CAT (1. 4.) *) (SENDR START (GETR START)))
                    (SETR P (MERGE (GETR P) **))
                    (SETR TIP+ (GETF PT1))
                    (JUMP STEP+!)))
```



The basic idea of this grammar is to allow small interruptions in a STEP+ profile. The STEP+ grammar in the body of the paper allowed stretches of slope 0 whose length was smaller then SIGTHR in the STEP+ profile. This grammar extends that by allowing sequences of input words whose CAT is 0, -1 or -2 and whose length is smaller than FLATTHR. It also checks to see if the STEP+ profile continues beyond the turning point. The issue is that in the earlier STEP+ network even a very small segment of slight negative slope would stop the profile. If the grammar made provisions for accepting these segments, it might have been the case that they belonged to a complete STEP- profile. What the new grammar does is allow itself the luxury of allowing fairly large (FLATTHR > SIGTHR) deviations from positive slope in a STEP+ profile without commitment to accepting it. The grammar looks ahead to see if the positive slope is resumed further on and if it isn't it returns the profile found so far.

The first arc of the state UP! is identical to that in the previous grammar. It accepts input words whose category (slope quantization) is greater than 0. The next arc goes to state RESUME-STEP+ in case the input slope is less than or equal to 0. The next two arcs are the end conditions on the recursion, the third arc ends the recursion at that level if the last input has positive slope and the last arc ends it for all others. State RESUME-STEP+ accepts input words with negative slope as long as their combined lengths do not exceed FLATTHR. Notice that this grammar, just as the previous one, allows a STEP+ profile to begin with a non-positive slope. This possibility introduces an added consideration. We don't want to allow STEP+ profiles whose positive slope length is less than SIGTHR. This check is performed as the pretest in the pop arc of state STEP+!, except that it checks the distance from START to the beginning of the current input. If the profile started with a non-positive segment of length greater than SIGTHR then our desires would have been thwarted. The arc act `(OR (GETR TIP+) (SETR START (GETF PT2)))` in the first arc of RESUME-STEP+ checks for this possibility, i.e. the register TIP+ would only be set if an input of positive slope had been found, otherwise the register START would be updated to the end of the current input which we know to be non-positive. The second arc of this state is taken when a positive slope is detected and it proceeds to check if the slope continues for a length greater than SIGTHR by doing a PUSH to UP! and sending it the current value of START.

The results of using this grammar on the same hammer handle picture are shown in figure A5. The starting point and slope were again found by SEER. Notice that although the scans still do not pinpoint the edges exactly they tend to cover more of actual profile. Many more of the scans found the small STEP- transitions on the sides of the handle. The fact that the scans are wider than the actual handle are not to be taken as failings of the system but are to be expected from the grammar. The grammar we have been using are meant to find the area of the scene where the general pattern of profile shapes one wants are found without pinpointing the edge. If one used a point in the STEP- profile where the intensity reaches 70% or 80% of the maximum one would have a better idea of exactly where the edges are

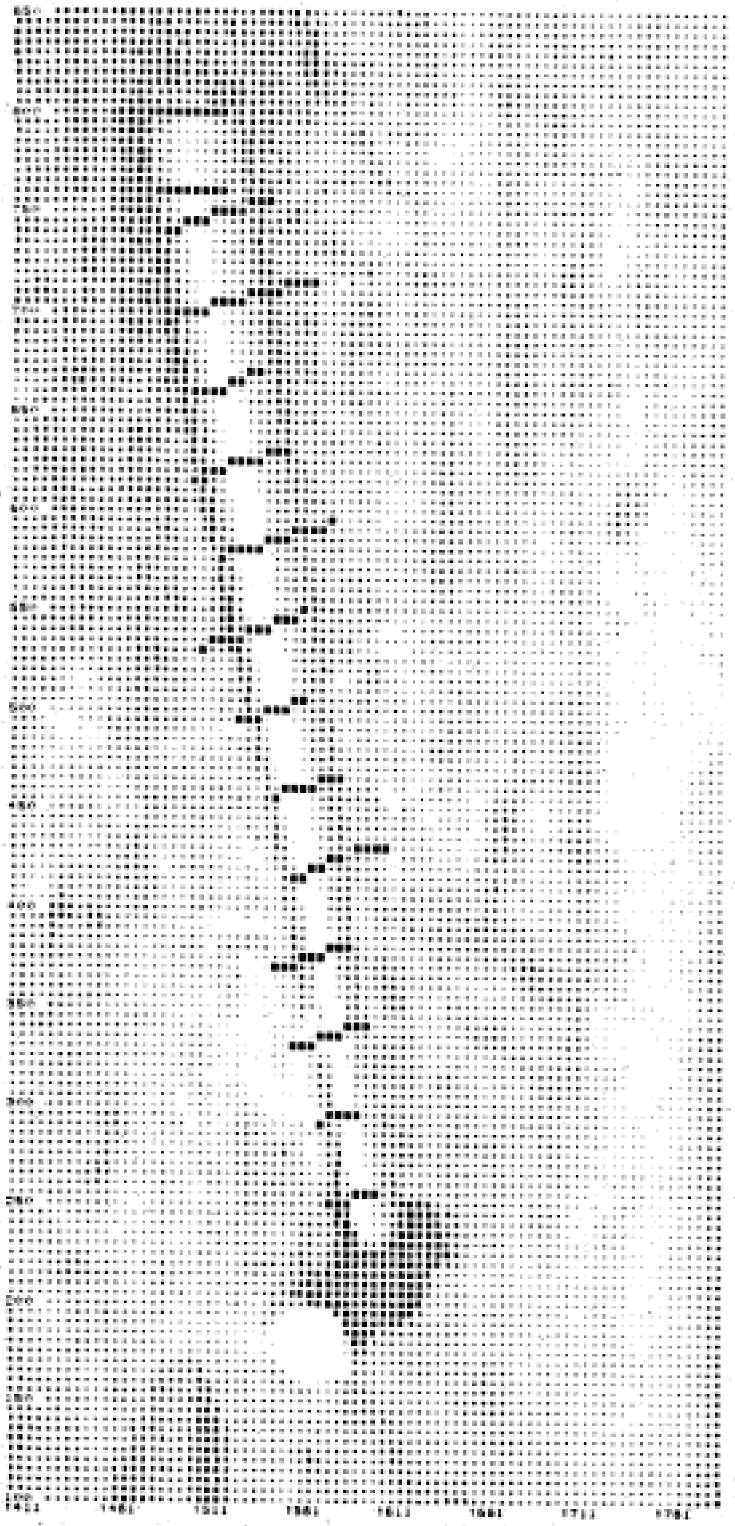


FIG. A5 - Profile traces