

READING A SKETCH BY HUNCH

by

James Richard Taggart

Submitted in Partial Fulfillment of the Requirements
for the Degree of
Master of Science

at the
Massachusetts Institute of Technology

May, 1973

Signature of Author _____
Department of Electrical Engineering, May 11, 1973

Certified by _____
Thesis Supervisor (Academic)

Accepted by _____
Chairman, Departmental Committee on Graduate Students



Valve Exhibit 1067
Valve v. Immersion

READING A SKETCH BY HUNCH

by

James Richard Taggart

Submitted to the Department of Electrical Engineering on
May 11, 1973 in partial fulfillment of the requirements for
the Degree of Master of Science.

ABSTRACT

This thesis describes the development of a computer system, HUNCH, intended to provide a simple means for a person to communicate with a computer his ideas through the medium of sketching. The emphasis is not on developing a computer system which produces finished quality drawings from sketched input, but rather on having the computer understand what is meant by the sketch. An overview of the intended goals of such a system is described, along with a comparison to other systems of sketch recognition. A history of the development of HUNCH is given to show the reader the evolution of the ideas invoked by HUNCH as it currently stands. A description of how HUNCH performs a data reduction pass to simplify and structure the sketch is given. Finally, a proposal for a graphical compiler is made to permit development of a system which would be able to understand sketches of a predefined class.

Thesis Supervisor: Nicholas P. Negroponte
Title: Associate Professor of Architecture

ACKNOWLEDGEMENTS

It is traditional to thank everybody in the world in this section. Special reference must be made to Vick Negroponte for providing the inspiration, and to the NSF and Project MAC for providing the means for keeping body and soul together while the project was carried out. There are also nameless millions who have contributed over the years to the development of HUNCH, especially Doris Ju and Cindy O'Connell who had to work under my direction, and Mike Miller and Chris Herot, who have been bricks throughout. Leon Groisser should be thanked for providing a needed boot at a crucial moment. Finally, a nod to Archie the Architect, for whom this system has been developed.

TABLE OF CONTENTS

ABSTRACT

ACKNOWLEDGEMENTS

TABLE OF CONTENTS

ORGANIZATION

SECTION I: OVERVIEW; Why am I here?

GOALS

OTHER SYSTEMS

LETDOWN

WHAT HUNCH DOES AND DOES NOT

HARDWARE USED BY HUNCH

SECTION II: PAST; How did I get here?

READING AND REDISPLAYING--DRAW/SHOW

SQUIGGLES

EXTRACTING LINE SEGMENTS--STRAIT/STRAIN

RATE/PRESSURE

INTERSECTIONS--INSECT

HORIZONTALIZING AND VERTICALIZING--LEVEL

CURVES

EDITING

SECTION III: PRESENT; Where am I now?

SECTION IV: FUTURE; Where do we go from here?

DESCRIPTION OF A SKETCH

EXISTING KNOWN SETS

LINE SET ATTRIBUTES

OTHER KNOWN SETS

FEATURES AND SET ATTRIBUTES

RECURSE

REFERENCES

APPENDIX I: EXISTING DATA STRUCTURE

APPENDIX II: THE GRID FACILITIES

APPENDIX III: LISTINGS OF STRAIN

ORGANIZATION

The body of this paper is divided into four parts: overview, past, present, and future. The first part of the paper is a look at the goals for developing an interactive sketching system. Other implementations of computer systems are described, and how they matched up to the defined goals is discussed. The second part is a history of events from its inception to HUNCH in its present state. Such a description seems important for two reasons. First, while the way HUNCH works can be understood without the history attached, why HUNCH exists as it is, and the motivation for the fourth part of the paper needs as part of its explanation how the ideas were derived and which ideas were discarded. HUNCH in its present form appears to be a regression from earlier successes. The motivation for this change of state is best explained by describing the sequence of events which led to the current state. Second, some of what has been learned about sketch recognition through the development of HUNCH is represented only by certain elements which are NOT included, perhaps despite earlier versions with these features. The knowledge gained by failures and changes of attitude over time is almost as great as that which currently is known. This history of HUNCH, then, is an attempt to apprise the reader of this knowledge.

The third part of the paper is a description of how the system as it currently exists is run. An extensive description of how

straight line data is extracted from the raw sketched input is included. This description not only shows how the more complicated portions of HUNCH work, but also indicates the philosophy of how operations are performed in a HUNCH-like way. From this outlook, one can see how other functions, which might be added to the HUNCH system later, would be implemented.

The last section of the paper may be looked on as conclusions and indications for future work. As it exists now, HUNCH falls short of its stated goals by quite a distance. Its biggest shortcoming is in its inability to derive a higher level description from a sketch. The last section provides the frame of mind which one might need in order to begin solving this inability. The solution proposed is neither complete nor rigorous, and in that sense, it seems strangely inconclusive. The only explanation I can offer is that the solution proposed seems to solve all the challenges I can think of, although sometimes the thinking required seems unnecessarily baroque. There must be simpler solutions, but finding them can only come with further experience.

I. OVERVIEW

Why am I here?

GOALS

There have been many computer systems developed which purport to let the user sketch using a computer. The user is placed before a console display of some sort, handed something which looks like a pen, perhaps is given some instruction in how to use the system, and is told to draw. It is reasonable to wonder in abstract, if one had such a system, what would he want it to do. There seem to be two answers: first, the system should help in the construction and storage of graphical images. Second, the system should act as an aide in the development of the information the sketch is meant to convey.

Under the first goal, when pictures can be constructed out of elements, saved, modified, and recalled at a later time, the designer has a useful time-saving tool for handling pictorial data. Repetitive elements need only be described once to the system. Representations of the complete structure can then be evoked with only a single stroke of the pen. Thus, instead of laboring hours over a drawing, the designer can describe the whole drawing to the computer using many previously defined elements, and the computer can construct the complete, finished work. Similarly, if two drawings are the same, with only minor

variations, the designer can construct one of them and store it away. He can then modify a copy of the saved image to match it to the second intended drawing, saving himself the trouble of constructing the drawing essentially twice.

More complicated than the first goal of a sketching system, one might ask a system to perform some more abstract operations on the sketch. A person uses sketches for two purposes: to convey information to other people which is difficult to transmit verbally, and to act as a sort of physical memory, in a sense, conveying information to himself. Once the sketch has been committed to paper, the user can modify it to change the information it contains. This act can be prompted either by the ebb and flow of the dialogue with the observer, or by a change in the sketcher's own idea brought about by the feed-back loop running between brain, hand, paper, and eye. In either case, the sketch is important because of the intended meanings it contains.

In a similar way, it is useful for a computer system being used as a sketching tool to be able to attach some meaning to the objects being sketched.

The result of such a dialogue is that the information contained in the interaction is greater than the amount of information which could be contained in the sketch alone, or which the user could carry around in his head. Thus, one would like a computer system for sketching to be alert enough to be able to affect a

dialogue with the user. It would need to be knowledgeable enough about the subject matter being sketched to be able to ask reasonable (intelligent?) questions, and perhaps offer some information of its own. In short, the computer should be able to enter into a dialogue with the user, in much the same way as a person observing the sketch being created might interact. Such a provocative system would tend to maximize the amount of information generated in a sketching session.

OTHER SYSTEMS

Computer systems which have been developed to date tend to be divided into two classes, reflecting to a certain extent the two goals for a computer sketching system. The first class, historically, is that which uses some specialized set of functions, keys, or symbols in the process of sketching. Input was accomplished by invoking a function (key, e.g.), which told the computer what the user was intending to do, with the ultimate goal of permitting the computer to store, retrieve, and assist in modifying a drawing. In response to the user's request, the system performed some output which accomplished the action specified. The second sort of computer system seen uses a limited set of known symbols, and attempts to map the user's sketched ikons, usually drawn on a data tablet, into these symbols. In this case, the computer does not know in advance exactly what the user is going to do. What the user intended

must be inferred from the match of the sketched item to the known symbols, and the computer is usually expected to take some action as a result of the recognition of the symbol. Because of this level of guessing, such systems are not infallible, but this objection is matched by a comparable improvement in the ease of input (In the first class of sketch handling programs, it should be noted, the computer is incapable of making a mistake; only the user).

The most notable example of the first sort of program is Ivan Sutherland's SKETCHPAD (Sutherland, 1963), particularly since it was the first attempt at communicating a visual image between user and computer in an interactive manner. In its stated goals, however, SKETCHPAD was to be a system unlike drawing with pencil and paper, because interacting with a computer was seen to be a totally different kind of experience. Using primitives common to all line drawings (line segments, points, and arcs of circles), the user creates symbols, structures, and composites of these images. To increase the power of the interaction, certain functions could be applied to previously defined images. Thus, when the user laid down two lines, he could indicate to the system that he wanted them to be parallel or perpendicular, and the system performed the requisite steps to make them exactly that way. Thus, the user could be inaccurate in his original layout and yet get a highly specific output of his final image. Furthermore, since the user was not drawing symbols for the

system to recognize, the kinds of graphic images the system could accept was unrestricted.

For such freedom, the user pays a penalty, however. The system of light pen on display and function keys utilized by SKETCHPAD bears no relation to the normal means of communicating an idea graphically. The fairly demanding system of input required by such a system would tend to interfere with the creative thinking process. The user is concentrating so hard on getting the drawing into the machine that is difficult to think about what he is drawing. IEM is marketing a new system similar to SKETCHPAD which uses a tablet instead of a light pen (Saderholm, 1973). While this hardware is an improvement over the old setpoint-rubber-band-line, it still relies on a set of function buttons on the tablet to relay commands to the system. The degree of explicitness required in such systems quickly generates tedium sufficient to offset any preference over the less complicated job of digitizing the data. Any sense of natural graphical communication is lost. Furthermore, since the computer is operating continuously in "slave" mode, it can add no information to the dialogue. Thus, an important potential is lost.

In the second class of computer systems, the computer does add information of its own to the dialogue in its interpretation of the sketched symbols of the user. Although this approach is

primarily found in character recognition programs , perhaps the most notable example is the GRAIL system developed by the Rand Corporation (Ellis, et. al., 1969). Besides recognizing the alphabet and decimal digits, it could also handle a set of flow charting symbols (rectangles, triangles, and so forth), and lines connecting these symbols. Using the Rand data tablet, the user drew his flow chart and labeled it. As each symbol was drawn, the system identified it, and the rough display of the user's line was replaced by the machine's representation of the symbol, appropriately scaled and positioned. Because of the level of inference making, the program was capable of making mistakes. In order to allow for errors and to permit the user to change his mind, one of the symbols recognized by the system was the scribbling out motion normally used by people to cross out an error or a misplaced line. The symbol was called a "squiggle," and caused any line or symbol which appeared beneath it to disappear. Once the flow chart was completed, the user could ascribe specific functions to the symbols of the flow chart and see what happened when the flow chart was "run." It provided a neat way of seeing information which might otherwise have been too difficult to visualize.

Systems in the GRAIL class are quite attractive, since they provide a sort of interaction which is very natural and familiar to the experience of the probable user. Drawing with a pen on paper is an experience common to most people, and GRAIL's

replication of this experience is not bad. On the other hand, these pattern recognition systems can only handle a limited class of inputs. Given an unrecognizable symbol, the system is lost. This problem is partially overcome in many character recognition systems by having a "learning" mode, where the system samples the individual user's representation of the symbols it knows, thereby adapting its models for the symbols to the habits of the individual user. There are two limitations usually imposed, however. First, the user's representation usually can not deviate beyond some accepted boundary conditions. For example, a character recognition program normally accepts either script or printed characters, but not both. Thus, a user who mixes his characters would inevitably be mis-understood. Second, it is usually impossible for the user to define symbols of his own. Thus, if a mathematician wished to use a character recognition system for the alphabet, he might be hampered by the inability of the program to accept Greek symbols; similarly, a Russian translator would have to start all over. Furthermore, as the number of symbols recognized by the system is increased, typically, the frequency of error increases at a much faster rate. This phenomenon occurs because the system can not use clues about the interaction between elements in a sketch. If a character recognition system had difficulty distinguishing between U's and O's, for example, it would be useful to look to see if the preceding character was a Q (in English, anyway).

LETDOWN

It would be nice to be able to claim to have developed the alert, provocative, interactive system mentioned in the earlier section. The system which has been developed, HUNCH, falls short of this goal, however. Provocative it is, although not in the manner described above. It is also moderately interactive. It does not, however, carry on anything which can be called a dialogue. . .yet. Dialogue implies purpose and a developing context, and although HUNCH does know a few tricks, once it has performed, all it can do is walk off stage.

The name HUNCH is derived from the methods it uses to achieve its ultimate goal. It uses guesses about implied intentions to determine what the sketcher PROBABLY meant. In that sense, it is similar to the character recognition systems discussed. It does not have a set of patterns it is trying to match, however. Rather, it attempts to extract from the stream of input data the primitives which make it up: line segments, arcs of curves, end points of lines. Once the data has been so compressed and structured, these components can be combined to form objects of a higher order. This second step is not as well understood, since it requires extensive knowledge about the subject matter being sketched to accomplish this goal.

Because its knowledge is somewhat more limited than a human's, it would appear that HUNCH is at a disadvantage when it comes to reading a sketch. If it was limited to those cues available to human, that claim would probably be true. However, because of the way the data is collected, HUNCH can use some information unavailable to the human onlooker. Inherent in the way the data is sampled is the sequence in which the sketch was drawn, the pressure on the pen at a particular time, and the rate at which the user was drawing.

These cues provide additional information which the program can use to make decisions. In general, for example, the faster the user draws, the less accurate he is. Furthermore, if he is drawing rapidly, it can usually be inferred that he is not interested in the fine detail of his line, but rather in the grosser features of what he is drawing. When it encounters a rapidly drawn line, then, HUNCH is prepared to make bigger assumptions and to permit greater inaccuracies before declaring one line segment ended and a second one begun. Similarly, if the user is drawing slowly with great deliberation, then nearly every bend or tweak in the line is preserved.

Pressure can also be sensed to provide cues to the intentions of the user. Here, the role variations of pressure in a sketch is not so clear. It appears, however that pressure variations occur in quanta; a user typically draws in no more than three or four

pressure ranges. An initial inference is that pressure behaves something like inverse rate--that is, the harder a person pushes, the greater the detail implied. The cue to look for, however, is the quantum pressure change, not the small variations across a line.

WHAT HUNCH DOES AND DOES NOT

Sketching can be considered to be a kind of graphical language. A person can read a sketch if he knows the rules for making a sketch, and if he knows the symbols used in the sketch--syntax and semantics. In order to carry on a useful dialogue, you have to have both. The HUNCH system does a reasonable job at providing a large portion of the syntax. It is one of the goals of this paper to outline a means of supplying some of the semantics.

The sketch, as received by HUNCH, is one long serial stream of data. The system tries to apply some structure to this data, to make the search for meaning more manageable. In a sense, the solution developed so far still leaves the computer doing what it does well--number crunching. In the process of discovering the structure of the sketch, massive amounts of data are reduced to a collection of points and relations between points. It performs these operations with uncanny accuracy, using only local information about the dynamics of the line.

The relations formed are only those based on information explicit in the data, such as the aforementioned rate and pressure, continuity of line, and sequence. Attempts to apply further relations to the data failed for various reasons described later, and in fact, appear inevitably doomed without the application of some semantic guides. Some of those relations attempted include latching of two known points, and horizontalizing and verticalizing lines in a sketch which appear nearly so. These functions failed largely because HUNCH was unable to judge those situations in which those relations might or might not apply. Thus, it applied them indiscriminately to any set of lines which fell within its guide-lines. The result inevitably was a severe distortion of the original sketch. Parenthetically, it should be noted that if input was limited to those sketches where the rule was always appropriate, HUNCH solved the problems with distinction. Thus, the problem was not in the rule, but in when to apply the rule. Given any rule, there is always a condition where, applied indiscriminately, the rule will fail (including this one). Thus, some means is needed to guide the system about when a particular relation might be appropriate.

The final output of the system is not intended to be a "working drawing" with all extraneous lines eliminated, all corners squared, and all lines straight and parallel. Rather it is intended that the output be the description of the sketch which

might correspond in some way to the verbal description a human might make of the sketch having observed it. The structure of this description would be hierarchical, having at its top the major features of the sketch, at its interim levels the elements which combine to make up these features, and at the bottom the individual line segments of which the sketch is made. The interaction of the various elements in the description provides contextual information. This information can be used to augment the rules about relations between lines which got us into trouble before to provide the guide-lines about when a particular rule might be applied. Furthermore, it helps to avoid the difficulties systems of the GRAIL class get into when called upon to recognize a large number of different elements. The context limits the number of plausible elements which may be used, preventing the system from drifting too far afield.

Unfortunately, this desirable description has not been implemented in any form. In order to derive such a description, the system needs to know what the elements are for which it should be searching (wired in to most systems). While in most types of sketches the number of these elements is not large, it has never been clear how one would specify these elements for the system. The description offered in the last section of this paper is a first attempt at making such a specification possible.

HARDWARE USED BY HUNCH

HUNCH runs on the Architecture Machine, a family of Interdata mini-computers, running under a disk resident operating system. The original sketch is read from a Sylvania data tablet, and can be stored on a variety of mass storage devices. The Sylvania tablet is (was) the Cadillac of data tablets, offering resolution to three thousandths of an inch, constant rate data sampling, and a clear tablet. This clear tablet means that it can either be drawn on as with other tablets, or it can be placed in front of the display and used in a manner similar to a light pen. Both of these modes are used by various parts of HUNCH. The tablet samples data at a constant rate (two hundred times per second), sending off to the computer twelve bits of x- and twelve of y-coordinate data at each sample. The tablet can also sense a limited capability for a z-dimension (three bits), such that it can tell if the pen is touching, is in the near field (about one half inch), is in the far field (up to four inches), or is away from the tablet. This feature suggested a logical extension, and the pen was modified to be able to sense pressure--how hard the penman is pressing on the paper. A load cell (a sort of transverse strain gauge) has been built into the shaft of the pen, taking the thrust from the top of the ball point pen cartridge. It can measure pressure from a fraction of an ounce up to a pound. This load is converted into a digital signal which is sent as a six-bit number to the computer. Each pressure

sample is associated with the point read when the sample was taken.

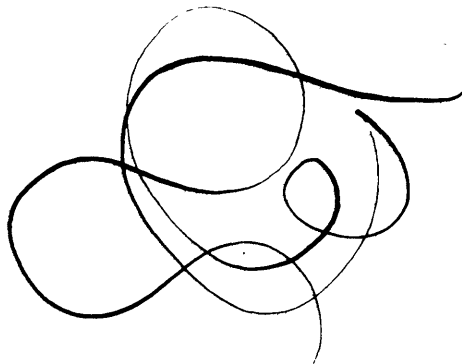
The display is not crucial to HUNCH, although it is useful for demonstration and debugging purposes. Because of the amount of data developed by the data tablet and the complicated pictures possible, it would be impossible to maintain a flicker free image on a refreshing display. After ten seconds of drawing, the screen would have two thousand vectors on it. HUNCH uses an ARDS storage tube, which effectively avoids this difficulty. Although it is difficult to dynamically modify the image on a storage tube, there is very little need to do so in a sketching environment. The difficulty of erasing is not unlike that the user experiences when drawing with pen on paper, anyway. Rather than erasing, the user just gets a clean sheet of paper. The ARDS has a limited dynamic mode, called write-through, which permits the dynamic alteration of a limited number of lines. This feature is adequate for those rare occasions when a picture must be modified.

While the sketch is being initially stored, it can be displayed in an exact mimic of the original. The ARDS is a relatively slow display, however, and the time taken to display the image reduces the sample rate which can be obtained from the tablet. Thus, the resulting stored sketch is less detailed. To overcome this difficulty, display while drawing can be suppressed. The stored

sketch can, of course, be replayed to the display. During times when the pen is not touching the tablet, a real time clock is sampled, so that the length of pauses in drawing can also be stored. The replayed sketch, then, can be an exact replica of the dynamic development of the original.

The addition of pressure sensitivity demanded an additional feature for the data display--some method for showing variations in pressure. The ARDS was altered such that its focus could be modified under computer control. Thus, while the tube normally displayed a thin, sharp line, by defocusing the beam slightly, the width of that line could be increased up to an eighth of an inch. This feature is integrated with the load cell in the pressure pen such that the width of the line varies as a function of the (original or redisplayed) pressure (Figure 1). This line variation greatly enhances the visual effect of the display, since it provides a better feel for pressure than the line output of a ball point pen can provide.

Figure 1.



II. PAST

How Did I Get Here?

READING AND REDISPLAYING--DRAW/SHOW

In the spring of 1970, the Architecture Machine Group obtained its Sylvania tablet, and embarked on an experiment to discover about reading a sketch by computer. The tablet was an ideal device for this experiment, having the natural feel of pen on paper, while at the same time providing a fast, accurate, time-dependent sampling of the sketch as it was created. The first programs written, naturally enough, were programs to read and save the data from a sketch, DRAW, and to redisplay the stored data, SHOW. DRAW senses the z-position of the pen, only recording data when the pen touches the tablet. The maximum distance the pen is away from the tablet (near or far field) is recorded as a flag in the stream of data whenever the pen leaves the tablet. The distinction of the z-fields is not used by any part of the program to date. It is thought, however, that the degree of pen lift may be useful for providing some clues into logical separation of the sketch into sub-sections, divided by higher lifts of the pen.

Where the pen went while it was not in contact with the tablet could be read from the data, and there was some discussion at the

time about whether or not this information should be saved. Since at the time, we did not know what information was going to become important, we were leery of discarding any obtainable information. The use to which pen-up information could have been put was unclear at the time (it is still so), however, and space limitations for storage of data were relatively severe at the time. It was decided, therefore, to discard this data.

DRAW begins by sensing the position of the pen in the z-field. When the pen touches down, DRAW records a far-field pen-lift flag and the x- and y-coordinates of the first point. With the recent addition of pressure sensitivity, the value of pressure is also saved. It then continues to read successive points and pressures, storing them away and (optionally) displaying them on the storage tube. When the pen is lifted from the tablet, DRAW waits for the pen to be replaced and saves the pen-up flag recording the farthest field reached by the pen and the time the pen was lifted. DRAW continues to read and save data in this manner until the pen is lifted away from the tablet field and then signals that the drawing is complete.

Once the data has been saved, it can be redisplayed by a call to the program SHOW. When this program was run for the first time, it caused the sort of serendipitous discovery which occasionally provides direction for research. Although it seems obvious in hind-sight, the effect the time based sampling of data would have

on the data itself had not occurred to anyone. Since the tablet samples data at a constant frequency (200 times per second), the distance the pen covers between samples is a direct function of how fast the pen is moving. Obviously--now--the faster the pen is going, the greater the distance it will cover in a two hundredth of a second--the farther apart the recorded points will be. The effect of this fact, of course, is that SHOW not only redisplayes the original sketch, but also it replays the sketch at exactly the same rate it was originally drawn. Inherent in the way the data is stored is the data is stored is the RATE at which the line was drawn.

This fact provided the ground on which HUNCH is built. It may be assumed that the speed at which a person draws reflects in some way his degree of purposefulness, his detailed interest in exactly what he is sketching. More specifically, it is usually true that if a person is drawing quickly, he is not as interested in detail as he is when drawing slowly. In a quick sketch, the person is usually interested in the general impression his lines make, rather than in the exact reproduction of those lines. Conversely, a slowly drawn sketch may often be painstakingly detailed. In this case, the position of each line becomes important, and the sketcher wants his drawing to be seen exactly as drawn.

SQUIGGLES

One special purpose kind of line is detected "on the fly." As mentioned in the description of the GRAIL system, the scribbling out motion has a special meaning. If drawn over previously drawn lines, it means that the earlier lines are to be crossed out. If filling an open area, the scribble implies that the area is to be shaded. In either case, the exact configuration of the line is not as important as the area it covers. Thus, it is not critical to submit the line to exact analysis. In order to extract these scribbles from the raw data, a "squiggle" recognizer was devised as a part of DRAW. Coincidentally, Rand's use of a squiggle, even the to the name itself, was not discovered until after the one in HUNCH had been developed.

A squiggle is characterized by several things. First of all, it has many changes of direction. It is usually drawn at a fairly high rate of speed, however, so it is not confused with the curving line of a driveway, for example. Finally these changes of direction form a sawtooth pattern (they are neither too spread out nor too sharp), so that a squiggle is not confused with a wobbly fast straight line nor with a line which has been heavily overtraced (Figure 2).

When the pen is placed on the paper, the squiggle recognizer begins searching the data as it is read for sharp changes of

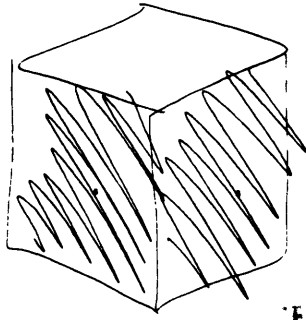


Figure 2.

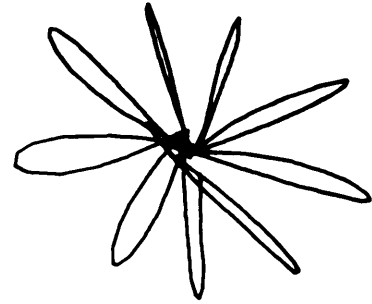
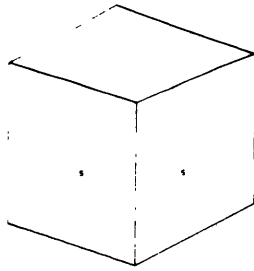


Figure 3.

direction, called extremes. Since a squiggle must be drawn quickly, the program expects to find many of these extremes before many points have been read. In fact, if the required number of extremes have not been found before a maximum number of points have been read, the program decides that the line is not a squiggle, and it quits looking. If the requisite minimum extremes do fall within the limit, then the position of these extremes is examined for the sawtooth pattern. If the extremes are too spread out or too close together (separated by angles greater than 9 degrees or less than 10 degrees), then the squiggle is rejected. Finally, the total rotation of lines connecting the extremes is compared to some maximum allowable value. If it falls above this maximum, the squiggle is rejected (this check is added because the person who

implemented the algorithm objected to the fact that flowers were recognized as squiggles (Figure 3)). Having passed all these tests, the line is recognized as a squiggle. The beginning point of the line is tagged, so that subsequent programs can recognize the line as a squiggle to be treated as a special case.

EXTRACTING LINE SEGMENTS--STRAIT/STRAIN

After the discovery of the rate dependence of the data, it was decided that the next step `FUNCH` should undertake would be to try to extract from the original data the straight line segments of which it consists. It would make its decisions by searching the sequential, raw data for "significant" changes of direction, tempering these decisions by taking into account the rate (and later the pressure) at which the line was produced. A more detailed account of how this program works follows in Section III. This section covers how the program arrived at the state it is in now.

The original attempt at a solution was a set of programs which eventually became known as `STRAIT`. This original version calculated the tangents of segments defined by connecting pairs of points in the raw data. It then looked for differences in these tangents, comparing the change in tangent to some value. If the change was greater than this threshold value, it was determined that one segment ended and another began. This solution quickly turned out to be a mistake. Because the tangent is so non-linear (going to infinity for a vertical line), the threshold level had to vary as a function of the direction of the segment. Furthermore, when dealing with infinity on a finite state machine, one quickly becomes embroiled in roundoff and overflow difficulties. As a result, this original approach was

abandoned, and a calculation of the arctangent of the segment was substituted. Except for a discontinuity in the arctangent function around zero radians, it has the attractive feature of being linear everywhere else. Thus, the threshold problem became direction independent. Furthermore, since the arctangent is limited to values between zero and two pi, the difficulties with infinity were eliminated.

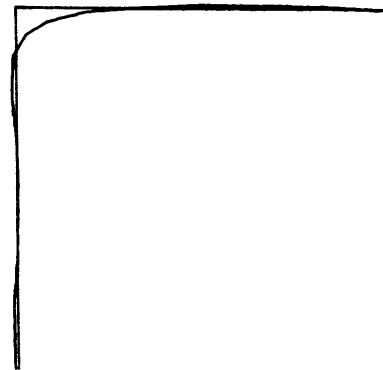


Figure 4.

It is the nature of a sketch that although a sharp corner is intended, it is rarely achieved. Instead, the two segments meeting at the corner are connected by some circular arc. Furthermore, since the drawer has to slow down in order to negotiate the corner and still be drawing where he intends afterwards, points tend to collect at corners. As a result of these facts, the first point falling above the threshold for a corner could not be assumed to be the actual intended position of the corner. In fact, because the (sharp) corner might actually be represented by an arc, there is no guarantee that the intended corner exists in the data at all (Figure 4). The only reliable way of determining the intended position of a corner is to determine the two segments lying on either side of the corner, and then to calculate algebraically the intersection of these two

lines. This was the approach taken in STRAIT. Once a segment has been discovered by an instance of a change of arctangent greater than the threshold, its endpoints are saved. When a second segment is found, then, the intersection of the two segments is calculated and used as the common endpoint of the two segments.

The result of such calculations is the creation of points and links between points which represent lines. When a pen-lift flag is found, the points on either side of it are saved as the end and the beginning of a segment. When the position of a corner is calculated, that point too is saved. To represent a line between two points, a link is created which contains information about which two points are connected, the rate at which the particular line was drawn, and the greatest pressure reached across the line. This structure is the output of the program STRAIT.

In order to cut down even further on the amount of data to be saved, and to maximize the inferred information in the final structure, STRAIT had a program which looked for implied "latches." When two points fall near each other, people will often mentally connect them as if they were a single point. STRAIT tried to do the same thing. In Figure 5, the first and last point of the line fall near each other. STRAIT decided that they were intended to be the same point, and latched them (The raw sketch appears at the beginning of Section IV).

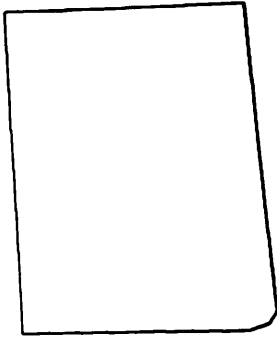


Figure 5.

The method for performing this operation is similar to that for finding corners. Each time a new point is determined, the list of existing points is searched for points nearby. In order to be considered near, the distance between the new point and each point on the list is compared to some threshold limit. If the distance is less than that threshold, the point is considered near. If no point falls below the threshold, then the new point is added to the structure with the appropriate link to any points which may be related by lines. If at least one point is found below the threshold distance away, the nearest point to the new point is considered to be the intended match. Links are created between this point and any related points, and the new point is not saved. Initially it was thought that the tendency would be for people to draw lines to known points, so the initial position of a known point was unmodified, and the direction of the new line was modified to take it to the known point. Subsequent experience seems to indicate that people tend to correct earlier errors in positioning points by drawing subsequent lines to where

the point should preferably have been. A somewhat better treatment, therefore, would have been to move the old point to the position of the new point, or at least to average the two points somehow. Difficulties with the whole latching scheme later tended to render the whole approach suspect, however, so this minor modification was never implemented.

The output structure of STRAIT, then, represented the minimum number of line segments and points which could describe the sketch, subject to some threshold values. For certain classes of sketches, this assumption proves to be entirely adequate; using these simple principles, STRAIT's handling of such a sketch is remarkable (Figure 6). STRAIT had several severe short-comings, however, which tended to point away from its existing mode of operation to some more complex handling of the data.

One difficulty came in the problem of handling overtracing. The tendency to retrace a line already drawn is a normal behavior on the part of a human user of the system. The method for finding corners was inadequate for handling the small angles commonly resulting from retracing a line. The precision of the computer was inadequate for sharp angles, due to roundoff errors and a tendency to wind up dividing by zero. Calculating the algebraic position of a corner between two lines which are nearly colinear resulted in frequent, severe misplacing of the common point. To combat this difficulty when calculating a corner, an additional

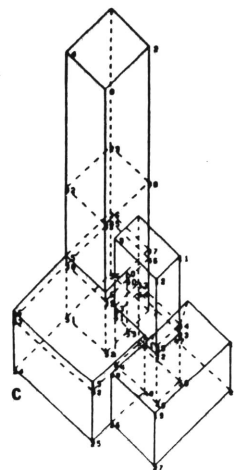
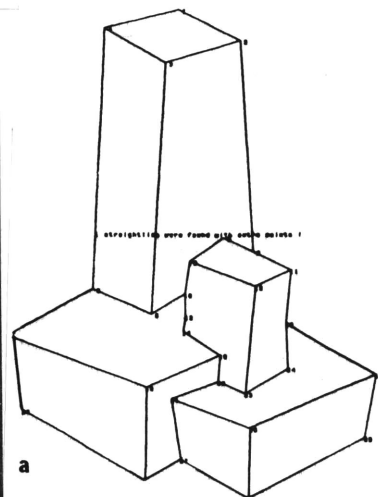
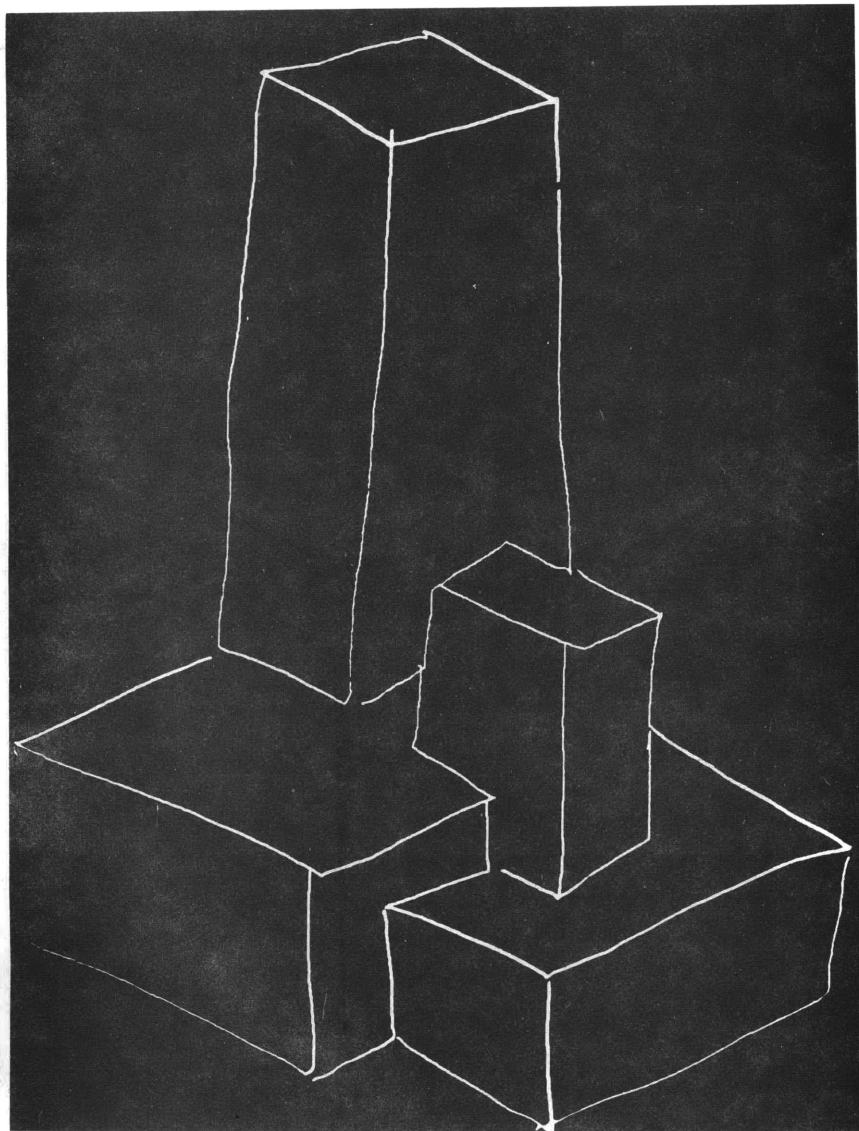


Figure 6.

routine was added to check specifically if the angle between the two lines was very small. For a small angle, using the calculated intersection is dangerous. Instead, an effort is made to find the point at which the line changed direction--the locally extreme point on the line. In such an instance, this extreme is used as the common point between the two segments. In the case of a sharp angle, the corner can not be very rounded, so the error induced by using a real data point on a corner is small. Unlike the case of a wider angle, furthermore, such an error causes little change in the direction of the line (this fact, after all, is what makes calculating the intersection so difficult).

The introduction of overtracing causes a vast proliferation of segment endpoints in a sketch. One of the restrictions on the class of sketches STRAIT could handle well is that the sketch must have a fairly diffuse distribution of corners and endpoints. When the density of points increases locally, then segments begin to become wrongly latched (see Figure 7, the proverbial sketch of Aunt Fifi's house). The addition of overtracing only

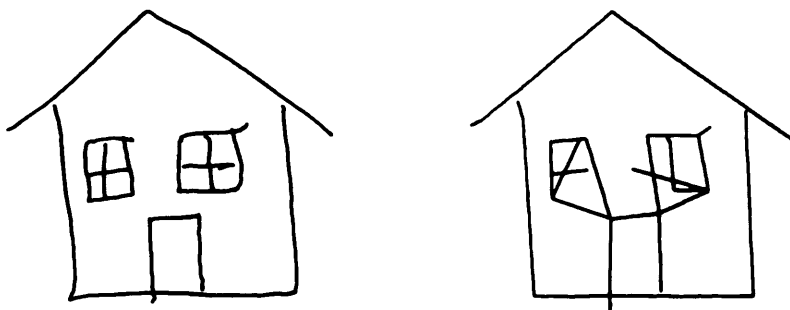


Figure 7:

aggravates the problem. The decision to latch or not to latch is not a purely local decision. Who should have authority to make such a decision is a point which is still under debate. The last part of this paper is one proposal at a solution. At any rate, it seemed futile to try to continue with the solution used by STRAIT, so a variation was developed which eliminated the latching step (STRAIN, STRAIGHTen with No latching). This seemingly backward step is justified as a basis for the groundwork for future development. We already know one method which will not work.

RATE/PRESSURE

Mildly glossed over in the above discussion of finding straight line data was the role that rate and pressure played. The original version of STRAIT had no measurement of rate or pressure; the various threshold values applied were constant throughout the program. After the basic routines were functioning more or less correctly, a method for figuring rate was determined (see description in Section III). Once the rate that a line had been drawn was calculated, that value could be applied to a function for figuring the various thresholds. Nominally, for a faster line, the thresholds were higher. The effect of this additional function on STRAIT was striking. There was an immediate, marked improvement in the decisions STRAIT was making about the data. In order for earlier versions of STRAIT

to work, the thresholds had to be set quite low to allow for the comparatively small changes of direction which occur when a line is being drawn slowly. Similar difficulties arose for latching. The effect of this limitation was to cause STRAIT to permit many more corners than were actually intended. If the thresholds were increased, STRAIT began to miss intended corners on slow lines. The addition of a rate measurement permitted application of a more liberal threshold for fast lines, a more conservative one for slowly drawn ones.

One side effect of this treatment was that STRAIT (and also STRAIN) became fairly sensitive to the "hand" of the user. Different individuals have different styles of sketching. Some people can sketch quite accurately at a very high rate of speed; others are not so accurate. Thus, the value of the thresholds for one person at a particular rate might not be appropriate for another person at that rate. In a pathological case, one can imagine a person who drew quite smoothly when moving his hand rapidly, but who suffered from palsey when moving his hand slowly. In order to work properly for an individual, then, STRAIT has to be tuned to each user's hand. Several methods for discovering the proper tuning for a particular person have been tried. The only one which works at all successfully is intuitive manual adjustment of parameters.

One program was written which tried to do the tuning job implicitly. The way the value of a threshold is determined is by applying the rate to a polynomial function:

$$TH=A*(Rate)**3+B*(Rate)**2+C*(Rate)+D$$

Rate varies between a value of zero and a value of fifteen. The various parameters (A,B,C,D) are a function of the hand of the individual user. Because of the complexity of tuning the parameters manually, the value of A was normally set to zero, reducing the polynomial to a quadratic. In such a case, the effect of the various parameters can be seen to be as follows:

At low rates, A predominates.

At high rates (assuming the value of B is a small fraction), the C term is dominant.

For the middle ranges, the value of the fraction B determines which way the function curves and the degree of curvature.

It was thought, then, that the various parts these parameters played at various rates could be separated and treated individually. Thus, the tuning program asked for a set figure (a square) drawn at a slow rate. It then juggled its B parameters until it got a four-lined, four-cornered figure. The program then asked for a quickly drawn square, and modified its C

parameter until it got a value that made the figure fit. Finally, it took a square drawn at a moderate speed and set the B parameters. While the program frequently could come up with a solution, almost equally often it could not find a value for one or more of its parameters which was satisfactory. Part of the problem derived from the limits which had to be placed on the values the parameters could take. In order to provide some starting point for the program and to prevent the arrival at some totally unreasonable parameter values, each of the parameters had an upper and a lower bound for the values it could take. In many cases, however, rather than settling on an intermediate value for a parameter, the program had a tendency to slide to one limit or the other. To compensate, the other parameters would become equally skewed. It is difficult to speculate on why this error tended to occur, but it appears likely that it was caused in part by the inter-relation of the parameters. Experience in manual tuning of the parameters seems to indicate that it may not always be true that the parameters can be separately tuned. Occasionally a better set of parameters was arrived at if, while increasing the value of one parameter, another parameter was comparably reduced. Since the implicit tuning program knew nothing about this technique (which appears to be largely intuitive, anyway), its results were often inadequate. At any rate, that particular experiment has been abandoned.

With the ability to sense pressure, a new variable has been introduced into the system. How this parameter should affect the behavior of HUNCH is not certain yet, as we have not lived with it for very long. It seems reasonable to assume that a heavily drawn line requires more detailed analysis than a light line. This effect can be accomplished by using pressure to offset rate.

As the pressure increases, it applies more drag to the line, slowing down the calculated rate. Thus, a quickly drawn line, drawn at great pressure, receives as detailed an examination as a slow and deliberately drawn line at any pressure. The rate is made to vary as a function of pressure according to the equation:

$$\begin{array}{l} \text{16-Pressure} \\ \text{Rate} = \text{Rate} * \text{-----} \quad (0 < \text{Pressure} < 15) \\ \text{16} \end{array}$$

This function has the interesting side effect that its impact varies as the rate it is operating on changes. At high rate, a moderate change in pressure (say from zero to four) cause a comparative change in rate (from fifteen to eleven). At a slow rate, the effect is less (three to two, e.g.). This fact is rather attractive, intuitively, but the overall effect has never been evaluated.

INTERSECTIONS--INSECT

Because of the way the data is generated and stored, there is no reason to believe that if two lines cross each other, their intersection actually exists in the raw data. This problem aside, in the process of searching for straight line segments described above, it is impossible to find intersections at the same time. Nonetheless, it is not unreasonable to want to know about the existence of intersections. In fact, for some applications, the finding of crossings and related T-intersections, is a great aid toward solution of the particular problem (Negroponte 1972). In order to locate these points, then, a program to find them was developed, INSECT (from INterSECTION).

INSECT uses a brute force method for locating intersections of lines in a sketch. The first line discovered is compared to all the other lines found, and the algebraic intersection of the line and each subsequent line is calculated. As these points of intersection are found, they are compared to the endpoints of the two lines being worked on to see if the intersection falls between, or within some delta (varying according to rate, as usual) of the endpoints. Intersections falling outside these limits are discarded. Those falling within the limits are considered to be discovered intersections (Figure 8). If the intersection falls within the threshold of one of the endpoints

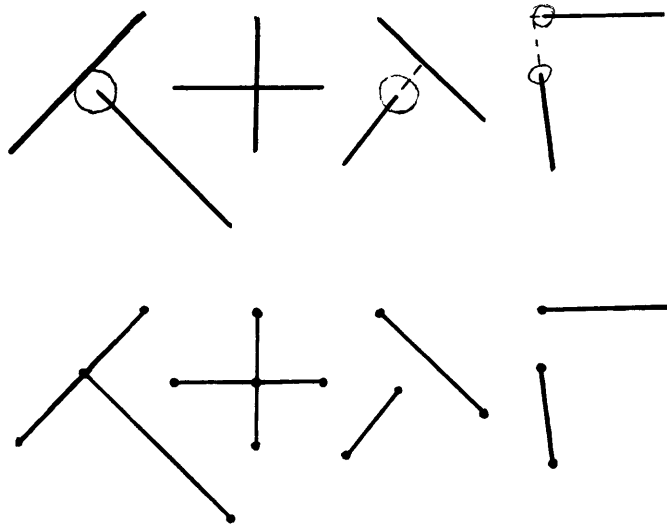


Figure 8.

of the two lines, that point is moved to the coordinates of the intersection, the other line of the pair is broken, and the endpoint is made the common end of the two new segments. If the intersection falls well within the ends of the two segments, a new point is added to the structure, and both lines are broken and attached to it. The second line in the structure is then compared to all subsequent lines; the third; and so on. The resulting structure has all intersections and T-joints inserted in appropriate places in the straightened version of the sketch.

It can be seen that the amount of calculation which must be done by this method goes up as the factorial of the number of lines in the sketch. This load is clearly unacceptable. As a sketch becomes more complicated, with overtracing for example, the amount of calculation goes up astronomically while the number of useful results does not. It seems reasonable to say that we are

willing to make N linear passes through the data (where N is a small integer--we are willing to look at each line N times), but we do not wish to pay the cost of even one pass through the data which is worse than linear. It took a long time to come up with a solution better than INSECT for finding intersections, and the better method has not been implemented yet.

We have a method for mapping a sketch, raw or straightened, on to a large array (existing on a fast-access fixed-head disk). For a complete description, see Appendix II. With a bit of cleverness, the straightened data can be mapped into two grids (or one two-bit grid), such that if the mapping program discovers that the bit in the first grid is already on (a bit being a point on a line in the straightened data), it turns on the equivalent bit in the second grid. It can be seen that, if every line in the straightened data is mapped on to the grid once, the only points which could appear on the second grid would be those points where two lines intersected. Thus, in order to include the intersections in the data structure, it would be simply a matter of reading points from the second grid which relate to lines in the straightened data. If a bit is found to be on, then, it is an intersection, and the coordinates of that point could be inserted into the line currently being tested. The extraction of all intersections would require only two linear passes through the data for a complete solution: one to map on to the grid, one to read for intersections.

It should be obvious that this solution does not find T-joints which were near misses (Figure 8A). This objection is not entirely bad, however. Since the method for finding T-joints in INSECT used the same kind of "latching" described for STRAIT, it was prone to the same kinds of difficulties. Essentially, this problem comes up whenever there is an attempt to apply a local decision to a situation which requires more global information. This error occurs where there is an attempt either to remove or to add information which can not be directly derived from the available raw data, where the data is extrapolated across empty space. Thus, the loss of this "missed T-joint" capability only parallels the removal of latching which turned STRAIT into STRAIN. If one wished to recover this ability for some particular application, one could take windows off the grid around the points in a sketch and look for lines which cross these windows.

HORIZONTALIZING AND VERTICALIZING--LEVEL

In an architectural context, lines which are horizontal or vertical have special meaning. This effect occurs because we live in a gravitational system which makes building horizontally or vertically a more reasonable way to construct buildings than any other way. Since the applications for HUNCH were considered to be primarily architectural, it was decided that it should be able to place a special meaning to lines which were horizontal or

vertical. The program LEVEL was written to search the straightened data for lines which could be implied to be horizontal or vertical. It calculated the restricted arctangent (between zero and $\pi/2$) of each straightened line segment, and compared it to a pair of rate-dependent threshold values. If the line fell above the upper threshold, it was considered to be a vertical line, and the coordinates of its endpoints were adjusted so that the line was forced exactly vertical. Similarly, if it fell below the lower threshold, it was eventually forced exactly horizontal. This program attempted to take into account the implied continuity of lines. If one line was found to fit one of the thresholds, the lines connecting to the points on either end of the segment were examined to see if any of them fit the threshold the same way. If a line was found continuing in the same direction, its second endpoint was also searched for a continuing line. This process was repeated until no further continuing lines were found. Then the positions of all of the points found to be part of the continuous horizontal or vertical were adjusted at once, so that the

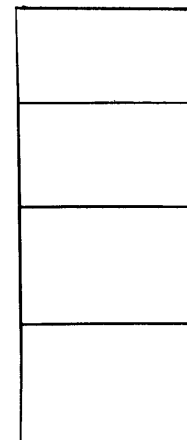
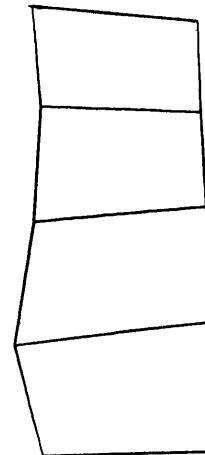
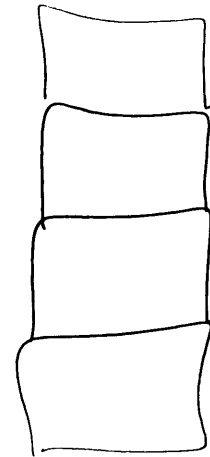


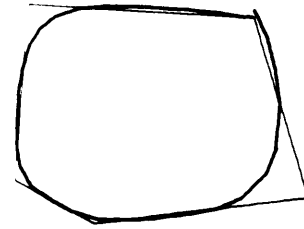
Figure 9

continuity was preserved (Figure 9). This need arises from the desirability of preserving continuity, and from the danger of moving one point without examining the points around it. There could exist a condition where a line which falls outside of the thresholds is moved within them by the leveling of one of its endpoints being acted on at another segment. Thus, if the points were adjusted independently, information from the original data might be lost by the partial treatment of a line segment. This approach made LEVEL perform as well as it could be expected to, but since it used the same extremely local information about line segments that latching and intersection finding do, it suffered from the same kind of indiscriminating errors that those two functions make. Thus, while serving as an educational and jazzy exercise, its usefulness is questionable.

CURVES

One of the first decisions made in the development of HUNCH was on the subject of curves. Curves are somewhat more difficult to handle than straight lines, since they are more difficult to define. A straight line, after all, can be represented by two points. A curve requires at least three, and it is not at all clear which three are appropriate. Furthermore, in a sketch, it was difficult to come to grips with the problem of differentiating a curve from a wobbly straight line or from a very sloppily drawn corner. In Figure 10, for example, it is

not even clear to a human whether that is a sketch of badly drawn rectangle or of a super-ellipse.



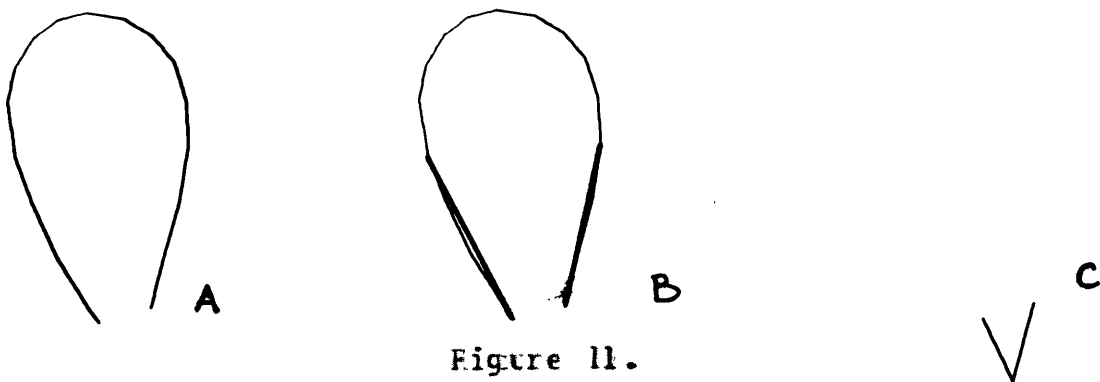
As a result of these difficulties, it was initially decided to side-step the issue by refusing curves as valid input. This decision can be partially justified on the grounds that, in the assumed architectural context, curves just do not occur that frequently. Avant garde architects aside, the vast majority of buildings have straight walls and flat ceilings (Negroponte, 1973). Thus, while ignoring the problem of curves imposed a limitation on HUNCH, the resulting simplification of the goals seemed to get us a long way before it became a problem.

Figure 10.

The immediate goal of HUNCH, then, was to look for straight lines. The approach used has a rather interesting side effect if the program is presented a curve. Since corners are defined by finding two line segments and calculating their intersection, a curve becomes simply a very long corner. Once a curve starts, no further action is taken by the straightening program until the curve ends. This fact makes the operation of HUNCH on curves rather unpredictable. In fact, it may be said that the way HUNCH operates is the worst possible way to handle curves there is: HUNCH will make worse decisions about curves than any other method of data reduction from sketches. The most extreme case of

failure is shown in Figure 11. In 11A, HUNCH found a straight line segment at the beginning at at the end of the stroke. They have been emphasized in 11B. It decided that everything in between the two segments was a corner, and calculated the intersection of the two segments, to define the position of the corner between the two segments. The results are shown is 11C.

While the initial assumption about the importance of curves still appears valid, over the years it has become a bit of a thorn in the side. The first thing anyone does in a demonstration of HUNCH is to throw a curve at it. As a result, it was decided to try to find a method for at least recognizing curves. If HUNCH knew a curve existed, that would be sufficient to keep it from being confused. Furthermore, in a sketch, the actual shape of a curve is not as important as is the recognition of its existence. The user is not likely to care whether the curve is a parabola, circular arc, sine curve, or part of a complex polynomial. He will care if it gets straightened, however.



The initial approach taken to try to recognize curves was to use the mathematics of the line to cause the curves to stand out. The slope of a sketched line (its first derivative) changes along its length. In the case of a straight line, the variations are small around some constant value. At a corner, the first derivative undergoes a discontinuity. For a curve, the first derivative is constantly changing, smoothly. If one looks at the second derivative of a line, then, that of a straight line will be zero, or nearly so. A corner would have a spike around the discontinuity, and a curve would be identified by some fairly uniform, non-zero value. Unfortunately, due to the method of sampling data, the theory does not work when put into practice. Local variations in the data tend to over-ride the actual data from the second derivative. Figure 12 shows some samples of sketches and the first and second derivatives associated. Any positive value in the second derivative is lost in noise from the data.

Another approach which shows more promise is to capitalize on the poor curve handling ability of the straightening pass of HUNCH. One way to make HUNCH handle a curve better has always been to draw it more slowly, since that would cause the curve to be segmented into smaller lines. In fact, small variations in the parameters used to determine the minimum bend which defines a corner result in radically different behavior of HUNCH on curves while having only minor effects on the analysis of straight

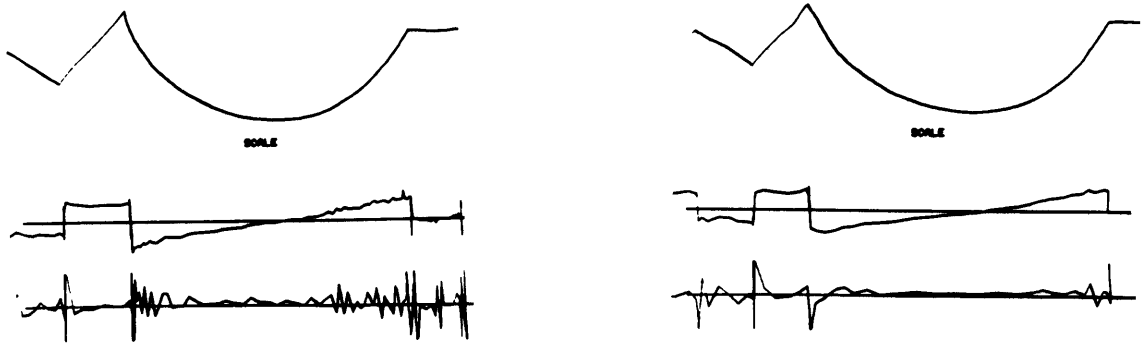


Figure 12

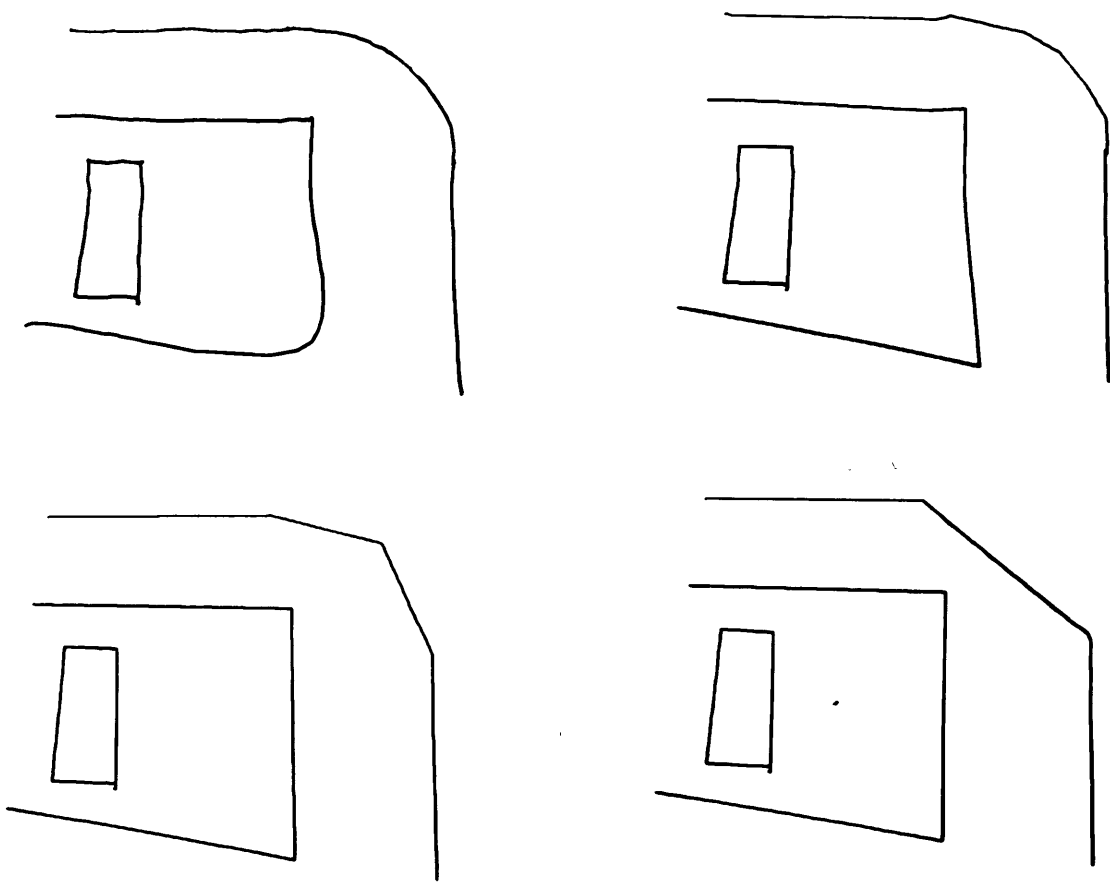


Figure 13

lines. Figure 13 shows a sketch consisting of both lines and curves, and three analyses by HUNCH of the sketch, using different rate parameters. It can be seen that while the variation of the analysis of the straight lines is slight, there are vast differences in the handling of the curves. Note that the handling of the rounded corner (at (a)) is done correctly in all cases. The implications of this discovery are not fully investigated yet. Although this approach was first proposed over a year ago, it was not tried until recently, after the first approach had been thoroughly discredited.

EDITING

Since HUNCH frequently makes mistakes (largely due to objections previously discussed), it seemed desirable to implement a means of editing the resulting structure. Since this editing was ultimately accomplished by explicit commands, their implementation has no direct bearing on the philosophy behind HUNCH. Their description is included for completeness, however.

To edit a sketch one needs the ability to perform four functions: the ability to add a point; the ability to remove a point; the ability to link a line to a point; and the ability to break such a link. The workhorse of the editing package is a program called MOVE. By pointing with the stylus of the tablet, the user can grab a point in the displayed structure and move it

to some other position in the display. If the new position falls close to another point, all the lines attached to that point are latched to the newly positioned point, and the other point is deleted. A point can be slid down a line that it is on until it becomes latched with the other end of the line and the extra point deleted. Points may also be deleted by a program called CLEAN, which eliminates slight bends in otherwise straight lines.

Since this process over-rides a decision made in the straightening pass, however, this program is extremely timid

To add a point, there is a program called BEND. If the stylus is pointed at the line to which the point is to be added, the line is broken, and a new point is added to the line at the location indicated. This point may then be moved or latched, as in MOVE

Finally, there is a program called DETACH which breaks links between points. The user draws a line across those lines in the display which he wants detached from a given point. A new point is created, offset slightly from the point in question, and the crossed lines are detached from the old point and latched to the new one. The program decides which point is to be detached from by finding the common endpoint of the lines drawn across. If there is only one line crossed, it decides which end of the line segment the intersection is closer to, and detaches from there.

III. PRESENT

Where Am I Now

Let us step through HUNCH's resolution of a sketch. The user invokes the command DRAW, and draws Figure 14, a rectangle. The drawing was done in one stroke in a counter-clockwise direction from the upper right. He signifies that the drawing is complete, and the file containing the data is closed. The actual data points stored by the program are shown in Figure 15. The distance between the points gives some indication of the rate at which the line was drawn. It can be observed that the first half of the square was drawn at a fairly rapid rate, while the last half was drawn more slowly. Since the pressure sensitive pen was not available when this sample was taken, the pressures associated with the data in this sketch are all zero.

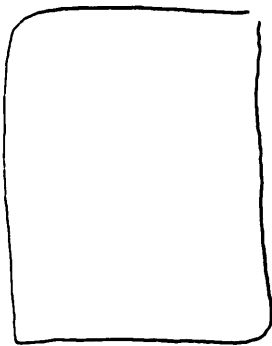


Figure 14.

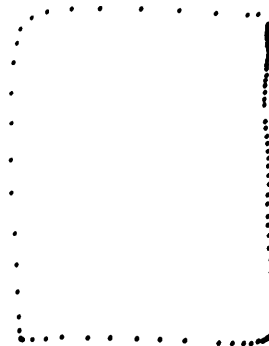


Figure 15.

The sketch is now ready to be examined for straight line segments. This pass at the data is described as separate from the DRAW routine, but there is no inherent reason for the two operations to be separated. With minor modifications, the command STRAIN could be integrated with the draw phase. In that case, the search for line segments could be done "on the fly." Under the present scheme, the straight line finding pass at the data is invoked by the command STRAIN.

The body of the work in finding line segments is bound up in three routines: NOEND, TANGNT, and TRNTST. These routines provide the main interface between the program and the raw data, and they are usually called in sequence, in the order they were named above. NOEND performs lookahead, data management, and parameter setting. TANGNT, a slight misnomer from historical reasons, calculates the arctangent of a line over a segment. TRNTST determines the difference between two successive arctangents.

NOEND is perhaps the most subtly complicated program in the system, since it apparently does so little and is responsible for so much. The first thing NOEND does is to call the routines which calculate the current applicable rate and pressure. RATER does not calculate the rate at every point, since the rate does not fluctuate greatly over a very small area and since the method of calculating rate works over a range of points around the

current one. In order to minimize the effect one anomalous point could have on calculation of rate, a method of calculation was chosen which examines points near the current point. Rate is considered to be an inverse function of the number of points sampled before the line has traveled some set distance, approximately three eighths of an inch. The current point pointer is temporarily backed up a few points, so that the sample will likely fall across the current point, rather than one side or the other of it. Then a points traversed counter is bumped, and the distance between the new current point and its following point is calculated. This length is compared to the fixed distance, and if greater, the calculation is complete. Otherwise, the current point counter is incremented, the counter bumped, and the next distance is calculated. The length of this segment is summed with the length from previous calculations, and the sum is compared to the fixed length. This process is repeated until either the fixed length is exceeded or until the points traversed counter exceeds sixteen. The difference between the points traversed counter and sixteen equals the rate (0-15, a reputable computer-based numbering system). The pressure at the original current point is then normalized to fall in the 0 to 15 range as well, and this value is applied inversely to the rate (see Section II, Rate/Pressure).

This calculated rate is then applied to the user's parameter polynomial,

$$TANDIF=A*Rate**3+B*Rate**2+C*Rate+D$$

where A, B, C, and D are parameters unique to the user for this application of the function. TANDIF is the maximum allowable change in arctangent before STRAIN decides that a corner was intended. Thus, the degree of turn which determines a corner is a function of the individual user and the local rate he is drawing at any given time.

The next task NOEND has is to check to see if it is going to run out of data. STRAIN is going to calculate the arctangent of a segment connecting the current point to a point some interval down the data (usually two points away). The reason for using this interval, rather than simply calculating for the segment between the current point and its successor is that such an approach would make the program too susceptible to local jigs in the line. By skipping a point or two in between, extremely local variations in the line tend to be smoothed out, while major changes in direction are unaffected. However, there are two events which must be watched for across this interval. First, since the data exists in a fixed buffer, NOEND must check to make sure that the end of the buffer has not been reached. If this is the case, the remaining data in the buffer is flushed, and more data is fetched. Second, and more important, the data across the interval must be checked for "pen-up" flags. Such a flag indicates that the line being looked at has ended. If a pen-up

flag is encountered, a special exit is required to a subroutine called ENDSEG. At the end of a segment, no more calculation can be done on the current line, so loose ends must be tied up, and the program must be reinitialized to begin a new segment. ENDSEG will be discussed later.

NOEND has one more function to perform; it checks for extremes on a line. As it examines the points on the line, it sends each point to a subroutine called EXTRMR. This routine checks to find out if the line has changed direction more than ninety degrees over the last few points. Such an occurrence is called an extreme. If searching for extremes seems redundant to the main-line search for corners, it is. The information developed by EXTRMR is used later by subroutine CFSA (Check For Small Angles), however, and therefore must be found. Note, this EXTRMR is the same as the routine called in the search for extremes defining squiggles.

Finding no ends of lines across the next interval, NOEND loads the x- and y-coordinates of the current pointer value it was handed (or the first point in the new buffer, if the end of the buffer was reached), and returns. The next step is a call to subroutine TANGNT. This routine increments the current point counter by the interval (two points), and gets the x-length and the y-length for the segment between the previous and the new current points. Using these values, TANGNT calculates the

arctangent of that segment, resulting in a value between zero and two-pi. This number is added to the top of a circular list of arctangents for later access, and TANGNT returns.

The third member of the trilogy, TRNTST, obviously can not be called until TANGNT has been called at least twice. After the second call to TANGNT, the list of arctangents is handed to TRNTST, which calculates the difference between the top two arctangents on the list. Because of the discontinuity of the arctangent function around two-pi, TRNTST has to worry about lines in this vicinity (a pair of nearly colinear, horizontal lines could return an arctangent difference of nearly two-pi). The exact difference of the arctangents is not as important as the magnitude of the difference, so TRNTST gets around the discontinuity by returning the lesser value of the following functions:

$$\begin{aligned} \text{Difference} &= |\text{Arctangent}(1) - \text{Arctangent}(2)| \\ &= 2 * \text{Pi} - |\text{Arctangent}(1) - \text{Arctangent}(2)| \end{aligned}$$

It can be seen that in the horizontal line case, the routine would return the correct value.

The power of these three routines can be shown by seeing how STRAIN uses them in its search for line segments. The first buffer of data is fetched and the current point pointer is set to

the first point in the buffer (194,x;540,y in the sample sketch).

STRAIN is going to try to deal with two line segments at a time.

1) Since the first point obviously begins a segment, its x- and y-coordinates are saved in an array called THOLD; a pointer to this location is stored as the first element in a second array called CORNER; and a pointer to the point's sequential position in the original data is stored as the first element in an array called DATP. Furthermore, a flag is set in RATE to signal that a new TANDIF parameter should be calculated, and a call is made to NOEND. Since the interval is currently set to two points, no endpoints or end of buffer is encountered. The rate is found to be 14, the x-coordinate 194, and the y-coordinate 540. When stored, the maximum rate reached while drawing the line segment is associated with the line for future reference. Since this is the first data examined, the associated rate must be the greatest found, so it is set aside for later comparison. A call is made to TANGNT, and the arctangent of the first segment, between the first and third points is calculated (6.27) and stored on the circular list TAN. This arctangent is the only one on the list, so a call to TRNTST would be futile at this point.

2) The current point counter is incremented to the second point in the buffer and NOEND is called again. RATER does not need to recalculate yet; there is no endpoint across the next interval, which lies entirely in the buffer. Therefore, NOEND returns with x equal to 98, y at 547, and rate the same as before. The rate is not greater than that previously determined, so the saved

value is unchanged. TANGNT is called and adds the arctangent of the second segment (6.22) to TAN. TRNTST is now called; it returns the magnitude of the difference between the two arctangents, DIFFERENCE=.05. This difference is compared to TANDIF, which for this user drawing at rate 14 has a value of .19. The change in direction falls well below this value, so no start of corner is determined. STRAIN returns to 2), increments the current point counter, and continues.

3) This loop continues until point 7 is reached. The x- and y-coordinates of point 7 are -687 and 531, respectively. The arctangent of the segment between points 7 and 9 is .25. At point 8, however, the arctangent has changed to .62. Meanwhile, the rate has changed to 15, so the value of TANDIF is .20. Thus, when TRNTST returns a change of arctangent of .37, something happens.

4) The significant bend in the line indicates that the line is entering a corner. The current point counter is backed up one point, so that it points to the end of the first straight segment. The x- and y-coordinates of this point are stored in THOLD, and a pointer to this data is stored in the second element of CORNER. The current point counter then goes into DATP.

5) A call is made to subroutine LINE, which calculates the slope of the line segment defined by the points indicated by the first two elements in CORNER. This information will be of use later, so the slope is stored in the third element of CORNER.

The maximum rate reached across the segment (15) is stored in a location called RMAX1.

Because of a tendency to make rounded, rather than sharp, corners when sketching, it can not be assumed that a corner occurs at a single point in the original sketch. To cope with this problem, STRAIN searches for the straight line segments on either side of the corner. Using these segments, then, the algebraic intersection of these lines is calculated, and this intersection is defined to be the corner connecting the two segments.

6) This method for calculating corners renders the data between line segments valueless. The collection of points which make up the corner in the raw data are useful only where they help to define where the second segment begins. Just as the corner was defined to begin where the change in arctangent was greater than TANDIF, the corner is defined to end and a new segment to begin where the delta arctangent falls below this limit. Thus, once the start of the corner is found, a new cycle of calling NOEND, TANGNT, and TRNTST is begun. Because the current point counter was backed up one point, the first results of the first cycle are the same as those described in 3) above. Since .37 is greater than TANDIF, the corner is continued. The next cycle, on point 9 results in an arctangent of 1.066. The difference between this value and the arctangent of the previous segment is .44, which is still greater than TANDIF.

7) This cycle is repeated until between points 10 and 11 the difference in arctangents has fallen to .13. This small change in direction indicates that a new segment has started.

8) The current point counter is backed up one point to get it to the first point in the segment (since the difference in arctangents was small, the segment must have started with the previous point). This point is preserved in THOLD and CORNER. CORNER is examined to see if it contains information about two segments, so the corner between the segments can be calculated. In fact, only one segment has been found and the starting point of a second one. Since no calculation can be made until the second segment is found, STRAIN returns to 2) to search for it, saving the data in the appropriate locations in CORNER and THOLD.

TABLE A

DATP	CORNER	THOLD	
		X	Y

1	->	194	540
7	->	-687	531
-	1	-	-
11	->	-743	431
20	->	-727	-750
-	-4668	-	-
23	->	-719	-745

By the time the routine gets back to 8), the current point counter has made its way up to 23, and a second corner has been delimited. DATP, CORNER, and THOLD have the values shown in Table A.

There are now two segments, so their intersection can be calculated. An examination of Table A shows that one would expect the corner to fall between -687 and -743 in the x-direction, and between 531 and 431 in the y-direction--approximately. Once this intersection has been determined, the exact endpoints of the first line segment in the sketch are known, and may be saved. Therefore, let us digress from the original sketch and see how this step is performed.

9) First, the angle between the two segments defined in CORNER is checked to see if it is very small, in subroutine CFSA (Check For Small Angles). For very tight angles, such as occur in overtracing, the precision of the arithmetic permitted by the computer was found to be inadequate. The calculated intersections of lines at small angles were frequently found to be highly inaccurate. As a result of this difficulty, the actual raw data was deemed preferable to calculated data for determining corners at small angles. Calls to EXTRMR from NOEND have previously determined if any extremes exist along the line so far. CFSA first determines the arctangents of the two segments defined in CORNER, then checks the difference between these arctangents to see if the two segments qualify. It then looks to

see if there are any extremes. If so, the point of interest must have an associated point number which falls somewhere between the indices of the midpoints of the two segments in CORNER. The "average" point numbers of the two segments are calculated, therefore, and the list of extremes is searched for one with an index which falls between these values. If one is found, and if the angle between the segments is small enough, then CFSA returns the x- and y-coordinates of these extremes as the location of the corner between the two segments. Parenthetically, if no such extreme is found, but the angle is still small, then the value returned is the second end of the first segment. If the angle is not small, then CFSA just does a little house-cleaning, discarding extremes already passed, and returns.

In the case of the sample sketch, while an extreme was found, the angle between the segments is nearly ninety degrees. This angle is too large to be considered, so CFSA returns no value. D0) The corner must be calculated, a task performed by CFIX. Since two points on each of two line segments in CORNER are known, the formulas for each of the lines ($Y=A*X+B$) can be calculated. "A" for each line is the slope, calculated by LINE, contained in slot 3 and slot 6 of CORNER for the first and second lines respectively. "B" can be determined by: $B=Y-A*X1$. This equation can be calculated by taking either of the known points in CORNER, getting its x- and y-coordinates, and substituting. Once the formulas for the two segments are known, another fact

can be applied--at the intersection, $X_1=X_2$ and $Y_1=Y_2$. Therefore:

$$A_1 * X + B_1 = A_2 * X + B_2$$

and:

$$B_2 - B_1$$

$$X(\text{intersection}) = \text{-----}$$

$$A_1 - A_2$$

Once $X(\text{intersection})$ is known, it can be substituted into the formulas for either of the lines to determine $Y(\text{intersection})$. In the interest of accuracy, in fact, $X(\text{intersection})$ is substituted into both formulas, and the resulting Y which causes the least change in arctangent from those of the two segments in CORNER (there is bound to be some slight adjustment) is used as $Y(\text{intersection})$.

The above description is true in most cases. However, there is a special case which arises when one of the lines is nearly vertical, as is the case in the second line segment of the sample sketch. When a line becomes vertical, its slope becomes infinite. Dealing with infinite, or large, numbers in a computer becomes difficult, since one begins to encounter round-off and overflow difficulties. There is one saving grace, however. A line is vertical because there is only a small variation in its x-coordinate across its length. In this case, there can be little error in assuming that the x-coordinate of the

intersection is the same as the x-coordinate of the point of the vertical line segment nearer the proposed corner (in the case of the sample sketch, the x-coordinate of point 11, $x=-743$). This value can then be substituted into the formula for the other line in CORNER, to determine the y-coordinate of the intersection. This approach is the one which CFIX takes, returning the values $X(\text{intersection})=-743$, $Y(\text{intersection})=530$. If both lines are nearly vertical, then the discovery of an implied corner was a mistake. The two segments are merged into one, and the program returns to look for a second segment.

11) Since both ends of the first segment have been determined, the first segment can be saved by a call to LINER. The data about straight line segments is stored in a variably sized structure; a more complete description of the data structure can be found in Appendix I. Generally, the data which is stored about the endpoints is as follows: the x- and y-coordinates of the two endpoints of the line segment; the index of the position in the raw data where the point was first discovered (the first point in the stroke, or the beginning point of a corner); and a flag indicating whether the point was immediately preceeded or followed by a pen-lift flag. The line is stored as a relation between the two endpoint, holding information about the maximum rate and pressure attained while the line was drawn. Absence of such a relation means that no line was discovered between two points. A point can have as many line relations associated with it as necessary to represent the picture, but in the current

approach, a point can have either one or two lines related to it after the initial pass at finding straight lines. On the segment just discovered, the first point will have a single associated relation, one with the second point. The second point, being a corner, is related to the first point (the same relation, in fact), and eventually to a third point which will define the second corner discovered. Since no latching is done at this pass, there can be no more than two lines associated with a point.

12) Once the segment has been saved, it can be discarded from CORNER, THOLD, and DATP. The beginning point of the second segment is shifted to the first position in CORNER, and the remaining data is shifted down accordingly in the various arrays.

Then the pointer in CORNER to the first point in THOLD is changed to point to the second point just saved by LINE. This second point will be used instead of the numbers in THOLD as one end of the next segment defined, the next time LINE is called. The common endpoint will thus be defined.

The state of the program has now returned to where it was after the first corner was discovered. The only difference is in the data in the various arrays (see Table B).

TABLE B

DATP	CORNER	THOLD	
		X	Y

11	->POINT(2)		
20	->	-727	-751
-	-4568	-	-
23	->	-719	-745

As it did before, therefore, STRAIN loops back to 2) and continues. The second corner is determined, and because the rate slowed toward the end of the line, the third turn in the line is broken into several corners.. STRAIN has now gotten to point 84 looking for a corner to end, so it can go to determine the seventh segment. When it picks up point 84 in a call to NOEND, however, it is discovered that the data element is a pen up flag; the line has ended. Since there are no more corners to be found on this line, the program branches to ENDSEG, which deals with end conditions of lines..

13) ENDSEG basically deals with the end of a line the same way the beginning of the line was handled in 1). Since the last point in the line obviously ends a line segment, the current point counter is backed up one before the pen-up flag. A pointer to this point is placed in CORNER, and its index is saved in

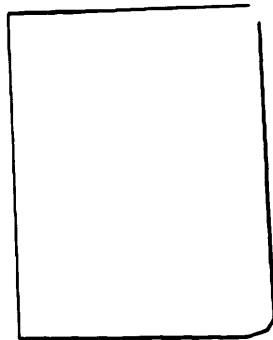
DATP, along with the pen lift flag (Note that since this point will be disposed of immediately, there is no need to save its coordinates in THOLD. There is no danger of the data being overlaid by a refill of the buffer). LINE is called to figure the slope. Two segments have been determined thus, so the corner between them can be calculated by a call to CFSA or CFIX. Then the first segment can be saved by LINER.

14) Finally, since the end point of the stroke is known exactly, from the data, no calculation needs to be done to determine the last point of the last segment. It can be saved directly by LINER. Once this operation has been done, all the information which can be determined directly from the stroke has been dealt with. CORNER is cleared, and STRAIN returns to 1) to look for more data.

There are two other possible cases which might have to be dealt with by ENDSEG, however. One is that the beginning of a corner has been found, but the curve of the corner has not terminated at the point where the pen lift flag is encountered, as is the case in the sample sketch. In this instance, there is little choice but to ignore the data occurring after the corner began. There is no way to calculate a position for this final corner, so it is simply dropped. This rarely results in any major distortion of the sketch, but occasionally, an entire final line is dropped if the line is sufficiently curved to fool the program (See the discussion on curves, in Section II). The calculation for this

case is essentially the same as in 13). The second case, which occurs more frequently, is that of a single straight line segment. If the pen is lifted before any corner is drawn, ENDSEG has no interim corner to find. In this case (i.e. CORNER only has two or three cells filled in), LFSA and CFIX are not needed, and the two points (first and last on the stroke) can be saved directly by LINER. In either case, all the information which can be found on the stroke is done with and CORNER is cleared.

When STRAIN returns to 1) to pick up another line, it discovers that there is no more data; it is done. The completely STRAINED sketch appears in Figure 16. The data is stored in a structure of points and lines described in Appendix I, and is accessible to other programs for further analysis. The above description covers essentially all the steps required to create this structure, covering all the special cases. The only likely extension is that which will enable the system to recognize and describe curves.



IV. FUTURE

Where Do We Go from Here?

DESCRIPTION OF A SKETCH

Having examined all that has occurred since HUNCH was begun, let us go back a bit to look at the original goals, particularly, the development of a hierarchical description of the sketch. The output of the straightening pass at the data is a two level structure of lines and points. While this structure makes the data much more manageable, it is not much of a step toward a description of the sketch in any normal human sense.

To elucidate what is meant by "a normal human description," let us try to generate one from a particular sketch. Figure 17 is a sketch of a house plan, much simplified. A verbal description which one might expect from a human is as follows:

1. It has 5 rooms
2. Rooms 1 and 6 have access to the outside; room 6 through one door, room 1 through two.
3. All the rooms are rectangular, having four walls, except rooms 3 and 6 which have six walls each.
4. Room 1 is connected to the outside through (doors in) its left and right walls, and to room 2 through its bottom wall.

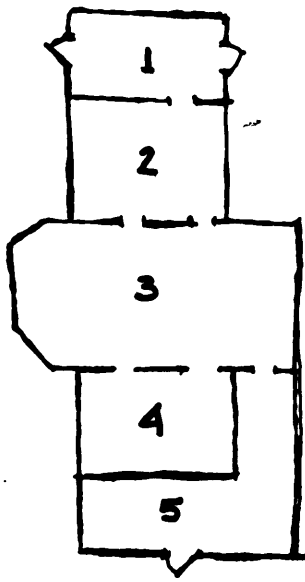


Figure 17a.

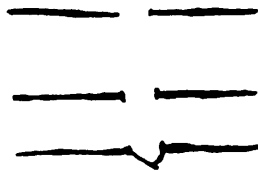


Figure 17b.

5. Room 2 has access to room 1 through its top wall, and to rooms 3 and 4 through its bottom wall.

6. Room 3 is connected to room 2 through its top wall, and to rooms 4 and 5 through two doors and one door, respectively, in its bottom wall.

And so forth.

This verbal description is not complete. One might wish to describe the exact configuration of rooms 3 and 5, to enumerate which walls were on the perimeter of the building and which walls were shared by the various rooms. The verbal description does not even specify a unique house plan. However, it does supply sufficient information such that some specific questions could be asked about the relationships between the rooms ("What rooms would I cross going from room 1 to room 5?"), about the overall accounting for the building ("How many rooms? doors?"), and about specific parts of the sketch ("How big is room 3?"). (Note that while the last question is not specifically answerable from the description, the area which has to be examined to determine the answer is considerably restricted.

In order to generate such a description, a person (or program) needs to know what kinds of elements might be found in a sketch, what kinds of questions are going to be asked about these elements, and how to recognize the various elements represented in the sketch. In the sample sketch, there are basically three elements: walls, doors, and rooms. The kinds of questions one

might be asked are about position, relative size, and relationships between elements. If the information is stored in some reasonably structured way, these questions ought to be easy to answer, once the various elements have been found.

The recognition of elements is perhaps the most difficult requirement. In the sample sketch, three different representations were used for doors (Figure 17b). In this simple case, then, what are some of the cues people use to identify the three elements which comprise the sample sketch?

To find the rooms in a plan, one would look for areas enclosed by sets of lines which define walls and which do not enclose other rooms.

Walls are characterized by sets of fairly long colinear, continuous lines which appear at the edges of areas which are rooms. They may have doors in them. They comprise most of the lines in a plan sketch.

Doors may be characterized by the fact that they appear in walls. They may be represented by a break in an otherwise colinear wall or a set of (probably) short lines bracketing a break in the perimeter of a room. These short lines will probably be perpendicular to the lines which make up the wall in which the break occurs. Finally, there may be a line about the same length as the break in the wall which lies at a small angle

to the wall in which the break occurs, and which shares a common end point with those points making up one end of the break.

The seeming circularity of some of the definitions (a wall is an edge of a room, and a room is limited by its walls) is an asset of such a description. One can apply one rule tentatively (take a guess at a room, for example), and then use the consequence of this guess to draw other conclusions. Eventually, one will either come to a solution, or because of the circularity of definition, one will arrive at a contradiction. If the description were completely directed and acyclic, one could drift farther and farther from the facts after an erroneous guess without ever noticing that anything was wrong.

In order to have a computer program which can construct a descriptive structure for a particular class of sketches, then, one must be able to specify to the program what features to look for and how the program is to go about recognizing these features. The process of recognizing features seems to require the application of those functions already discredited in earlier discussion: latching, horizontalizing, verticalizing, parallelizing, perpendicularizing, conlinearizing, normalizing, and continuing. While this is true, in this case the functions are applied in some context. Earlier attempts were simply exercises in investigating how the HUNCH approach could be applied to solving these various functions. The result was a

program which knew a rule and which applied that rule under all conditions. The effect of this method was that the rule was applied under conditions where it was possible but not appropriate. In the proposed solution, the program would have some idea about when the rule could reasonably be applied while searching for a specific item.

Using this system, the user would name a set of features which would be searched for in a particular class of sketches. The names of these features will provide the means of describing the sketch, and relationships between the features will provide clues to the hierarchy for the description (a kind of precedence relation). Once the features have been named, the user would describe the attributes of each feature which would be searched for in a given sketch. In general, a feature consists of a set of points, or lines, or perhaps other features, or a combination of these elements. Identifying whether a part of a sketch is a particular feature involves matching elements of the sketch to attributes of the set describing that feature. For example, an attribute of a one element set consisting of a straight line segment is the direction in which the segment lies; another attribute is the length of the segment. The process of telling the program how to recognize a particular class of sketch involves naming a set of features and then describing attributes of the sets which determine a feature. This description will provide a structure against which sets of lines and features will be matched by the program providing the description of the sketch.

EXISTING KNOWN SETS

After a sketch has been processed by STRAIN, the output of the operation can be considered to be two sets: a set of lines and a set of points. These two sets are the primitive sets from which the rest of the description will be derived. Furthermore, the elements of these sets each have their own set of primitive attributes.

The most basic element of a sketch is a point. It has two primitive attributes: position and sequence. The initial value of position is the obvious one--simply, where in the coordinates of the tablet the point lies. Later analysis might require changing this value (a three-dimensional mapping, for example). Sequence can be taken to be the temporal position of the point. In the process of straightening the sketch, the index of a point, in the original stream of data, which is the start of a corner or an end of a stroke is saved and associated with that point in the straightened data structure. These two basic attributes, then, provide a one-to-one mapping from a point number into a three-dimensional space/time volume.

There is another attribute of a point which should be considered; that is, it can define one end of a line segment. In the straightened data structure, the role this attribute can play is fixed, since there are at most two line segments which can have a given point as an endpoint. This attribute was not

included in the discussion of primitive attributes, however, because in defining features, one might wish to construct lines not appearing in the original sketch. Thus the manifestation of this attribute might change under varying circumstances. More properly, this attribute of a point will be reserved for the discussion of the relationships between points.

A larger set of attributes is associated with a line segment. As with points, there are two obvious primitive attributes: direction and magnitude. The direction can initially be taken to be the arctangent of the line relative to the positive horizontal direction of the tablet. Its magnitude is its length, using the tablet coordinate system. As with the position of a point, the value of either of these attributes might have to be changed if the line is mapped into three-space. In the process of straightening, the fastest rate and the greatest pressure reached while drawing a particular line are associated in the structure with the line. These attributes suggest others which might be considered at some later date, if the means of input should be modified. One can imagine a line having width, color, texture, even thickness. Finally, as the inverse of the point attribute, two attributes of a line segment are the points which define its ends. Unlike the point example, the two endpoints of a segment are fixed, so these attributes are unchanging.

Given these two sets, some analysis of a sketch is possible, as

long as that analysis is related to some absolute scale. One might wish to examine all the lines longer than a certain minimum length, or, those lying in a particular direction. The process of looking for horizontal and vertical lines requires just this sort of analysis. Most analysis requires the ability to distinguish relationships between sets of points or lines, however; there is no absolute background against which the measurement takes place.

Because of this need, there must be the capability to describe relationships between points and lines which can be assigned as attributes to sets. These primitives are discussed in the next section.

LINE SET ATTRIBUTES

In describing the features of a sketch, one uses words like parallel, perpendicular, angle, corner, near, and so forth. It is proposed in this section to define a set of primitives which can be applied to lines to test them for these descriptors, and to describe how the procedures which establish the existence of these attributes might be implemented. Given these primitives (and some other, more general ones described in a later section), one can describe the attributes of a set of elements which would make up a feature of a sketch.

PARALLELISM: In most sketches, there are preferred directions. In the sketch of the house plan of the average house, the great

majority of the lines would fall into two perpendicular directions. In an axonometric sketch, there would be three preferred directions. A histogram of the direction of the lines in a sketch versus either the number of occurrences of a given direction, or the total length of line in a particular direction, or a combination of both, will show peaks at the various preferred directions with a fairly narrow standard deviation (Figure 18). It is a simple matter to divide the sketch into families of lines, using the valleys between the preferred directions as division guide-lines. One could use rate and pressure to influence the decision, as far as allowable deviation is concerned. Some sort of average direction could be used to define the family which was implied, and this identifier for the family could be associated with each line in the straightened data structure. A set of lines could be considered parallel, then, if they all belonged to the same family.

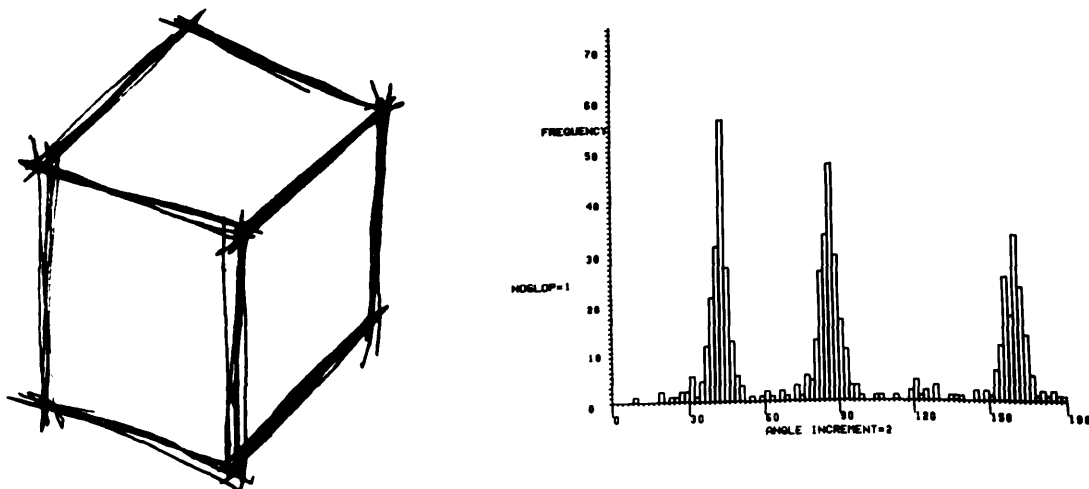


Figure 18.

COLINEARITY: Once the family of a line is determined, the family identifier can be used to calculate the x- or y-intercept of that line. The result of this calculation could also be associated with each line in the structure. It is likely that intercepts would tend to fall in groups as arctangents do. A set of lines could be said to be colinear, then, if the set had the attribute of parallelism, and if each element of the same, or nearly the same, x- or y-intercept. Again, the definition of near could be subject to rate and pressure data influence.

CONTINUITY: In the strictest sense, a set of lines can be defined to be continuous if the set is conlinear, and if there are no "breaks" in the line. More specifically, a line is unbroken if there is never a case where, scanning from left to right (or from top to bottom), the right-most end of the line segment is encountered before the left-most end of another line segment is found, if there are still line segments to the right. Normally, one would wish to include instances of "near continuity," where the break in the line is small, in this definition of continuity. This modification of the definition is somewhat risky, since it is prone to the same difficulties encountered in earlier latching attempts. A way to get around this objection would be either never to apply the weak continuity rule until other analysis had failed to provide a satisfactory result, or to flag a weakly continuous set, such that later difficulties might be resolved with a minimum of searching for

the error. In any case, it should be noted that this form of latching is only applied in a fairly explicit set of circumstances: the distance is small, and the lines to be latched are colinear..

With the above primitives, one could make a generous stab at reducing the complexity of a heavily overtraced sketch. A "Meta-line" could be defined as a feature of the sketch having the attribute that each Meta-line in the sketch would consist of a set of continuous line segments. The endpoints of the Meta-line would be the left- and right-most points which were attributes of the lines making up the set (in some cases, the top- and bottom-most points would be used), and the deviation in the y-intercept of the lines in the set could be used to define a width for the Meta-line. Once this information had been determined, further analysis could proceed exclusively at the Meta-line level. It would not usually be the case, however, that an analysis would proceed in an exclusively bottom-up manner, as described above. The example does show how the attributes of a set might simply be related with fairly powerful results.

SEQUENCE: In the set of points, a point which ends a line segment must be preceeded immediately by the point which began the segment. Similarly, the following sequential point is either the end of a line emanating from the point in question, or it is the beginning of the next line to be started. At any rate, the straightened data structure contains the complete sequence in

which all the lines in the sketch were drawn. Two lines are in sequence, then, if the second point in one line is either the first point on the other one, or it immediately precedes the first point in the second line--or vice versa. A Dashed-line could be defined as a set of lines which are colinear, non-continuous, and in sequence.

ANGULARITY: The intersection of two lines forms an angle. Angles seem to perform two functions in a sketch: they permit definition of relationships between non-parallel lines ("line A is perpendicular to line B"), and they indicate intended changes of direction (as at corners). An angle provides a relationship between two (sets of) lines, and it consists of two defining elements: a pair of lines or line sets and a magnitude. The position of the angle can be taken to be the point of intersection of the two lines. The magnitude of the angle between line A and line B can be taken to be that solution of the following two equations which is non-negative and less than π :

$$\text{Magnitude} = B.\text{direction} - A.\text{direction} \quad \text{or}$$

$$\text{Magnitude} = \pi + B.\text{direction} - A.\text{direction}$$

Note that this method of measuring the magnitude makes the value of the result depend on which line is mentioned first; $\text{Angle.Magnitude}(\text{lineA}, \text{lineB}) = \pi - \text{Angle.Magnitude}(\text{lineB}, \text{lineA})$. Since an angle has two sides, this distinction is important. To compare the magnitudes of two angles, he must first be certain that he is measuring the angles from the same side.

Where the corner between two lines exists, this definition of an angle is fairly simple to handle. In those cases where two lines do not physically intersect, however, the problem becomes more complex. One would like to create a point for the intersection, and then construct lines from it to the ends of the two lines whose angle is to be measured ("Construct point P, such that line(P,B) is colinear with line(A,B) and line(P,D) is colinear with line(C,D). . ."). This desire leads to the need for the ability to describe a "working point" and a "working line." Two functions are needed to modify the point-line structure: the first, $P = \text{Setpoint}(X,Y)$, adds a point to the structure at location (X,Y) on the tablet (or perhaps some modification of the above to allow for three-dimensional positioning). The arguments of the function would serve to establish the position attribute of the element. Since the point did not appear in the original data, it has no sequence attribute. A pointer to this new point would be returned in P. The second function is $L = \text{Line}(A,B)$; it constructs a line between points A and B and assigns L a pointer to it. The direction and length attributes can be calculated, while other attributes (rate, pressure, and so forth) are undefined. Once a point or a line has been created, it can be taken as an element in any set defined by the attributes described above. Thus, one might examine a sequence of points to see if they were colinear by constructing lines between them and then examining the lines. Similarly, the distance between two parallel lines segments could be found by constructing a line perpendicular to the lines from

one of the endpoints of the lines. By adding to the structure the intersection point of this line with the other of the parallel lines, the length of the constructed segment is determined. The value of this length is the distance between the two lines.

OTHER KNOWN SETS

While experimenting with primitives to describe features, one keeps arriving at a need to define a set with an attribute of area. For example, in the house plan, a room is an area enclosed by walls. Similarly, in an axonometric sketch, a surface of a solid can be described as an area enclosed by edges. It seems necessary, then, to define an extension to the current structure for a set of areas.

The attributes of an area are similar to those of a line. Corresponding to the line's attribute of length is the area's attribute of magnitude--the amount of area encompassed. This attribute can be defined by the number of tablet coordinate squares enclosed by the area. Corresponding to the endpoints associated with a line is the perimeter of the area. Unlike the endpoint attribute, which has a fixed number of elements, the perimeter of an area consists of an indefinite number of elements; it is a variable length list of pointers to features in the sketch which defines the set of elements containing the area.

Note that the set of elements defining the perimeter might change as the analysis of the sketch proceeds. The perimeter of an area might initially consist of a set of line segments. As these segments are absorbed into the description of features of the sketch, these features would be substituted for the lines. Thus, when the lines defining the room in a house plan are included in the description of the walls in the plan, the walls become the defining elements of the room. With the perimeter attribute of an area, however, it seems reasonable to extend the attributes of a line to include the attribute that a line is an element of the perimeter of an area, just as a point was described as one end of a line segment.

Other attributes which might be associated with an area are reflected in those of a line. An area could have an attribute of orientation, which would be defined as the direction perpendicular to the plane in which the area lies. Initially, the orientation of all areas would be the same--perpendicular to the plane of the tablet. In the case of a sketch of a three-dimensional object, however, this orientation might change (as might the magnitude attribute). While an area does not have either the attributes of rate or pressure, it could have color, texture, or thickness.

Having gone this far, it seems reasonable to propose that a further extension to the structure be added to permit the

definition of volumes. Just as the description of features in two dimensions led to the necessity of describing areas, it is inevitable that the need for descriptions of volumes will arise. Like areas, a volume would have the attributes of magnitude and perimeter, although the perimeter would consist of a set of areas. Orientation does not seem to be relevant in the description of a volume, but color and texture seem possible, along with density, center of mass, opacity, and just about anything else.

FEATURES AND SET ATTRIBUTES

Features may be defined by enumerating the attributes which must hold for elements of the set comprising the feature. Some of these features may be defined in terms of those attributes already described, as the Meta-line was defined. There are other attributes which will prove useful, however, which operate on sets in general, rather than on the specialized set described above.

BOOLEAN OPERATORS: The three operators AND, OR, and NOT specifically, may be used either as modifiers to attributes or as operators on sets. As an attribute modifier, the operators can specify that a particular condition may NOT occur, that either of two conditions may hold, or that a pair of attributes must hold

("colinear & ~continuous V ~colinear. . ."). As a set operator, the booleans would act as the Union, Intersection, and Set-Subtraction operators, permitting the concatenation of sets and the subsetting of sets (Wall::=wall & Door V Wall & Wall. . .). The effect of such an operation on the description structure would be the addition of an element with links down to those elements being concatenated (or in the case of a subtraction, the creation of two elements linked to the original set from below).

NUMBER: A set may have to have a particular number of elements (or features). There must be a means of specifying that number, then, and of comparing a number of elements against that number. A rectangle, for example, implies an area with four edges in its perimeter set. Once an area has been isolated, the number of its edges must be compared to the number of edges specified for the set, to see if it is less than, equal to or greater than the required number.

Similarly, since some attributes return values other than boolean values, the relative magnitude of these values should be comparable, addable, subtractable, and so forth. This requirement implies that either numbers of elements or value of attributes must be countable and able to be operated on by the normal arithmetic functions.

SIMILARITY/EQUALITY: Two sets are equal if all the attributes

of both sets and of all the elements of each set are equal. Two sets are similar if the values of the specified attributes are the same. In the absence of an attribute, a particular feature is similar to a feature name if it is in that feature class. Thus:

$rect1 = rect2$ if everything is the same.

$rect1 \leq rect2$ if $rect1.width = rect2.width$ &
 $rect1.length = rect2.length$

$rect1 \leq Rectangle$ if $rect1$ is in the class of features called Rectangle.

These working definitions permit the comparison of sets by permitting the user to define what he means by two sets being similar. In a case where it is desirable to define features which are similar but which may have minor variations, this ability can greatly simplify the definition process.

MEMBERSHIP: A particular element has the property that it either does or does not belong to a particular set. In the hierarchical description, if an element belongs to a set, there will be a vertical path from that element linking it to the set name (Wall e Area.Perimeter).

RECURSE

In order to demonstrate how the functions defined above might operate, the features mentioned in describing the sketch at the beginning of this section will be defined in terms of these functions. The sketch had basically three features: Rooms, Walls, and Doors. The third paragraph on page 72 gives a verbal description of what a person might look for in recognizing these features. Table C gives a translation of this description into the set of attributes defined in this section. In order to complete this definition, three auxiliary features were defined: Breaks, Clusters, and Metalines. The description of a door mentioned that it was indicated by a break in a wall. This description implies that what is meant by a break is known, so it has to be defined as a feature as well. The other two features are defined merely to help simplify the descriptions. A metaline is defined formally in the same way it was described earlier in the text. A cluster is simply a collection of points no farther distant from one another than some distance, whose magnitude is defined by the rates of the lines which the points define.

The notation used to define the features is a sort of bastardized set notation. A glossary of the notation is given in Table D. While the circularity of the definitions for Door indicates that more work is desirable to make this method of

describing features more humane, it does show that such a method is a viable approach to the problem. This statement is reenforced by the fact that I had no idea about how the formal definition was going to be implemented until after I had defined all the sets described in this section. Since I was able to do the formal definition using only those sets, it can be argued that they are at least sufficient to accomplish the job at hand.

TABLE C

Room::=R | R=>Area & (^R1 | R1=>Room & (R & R1)^= R)

*A room is an area which contains no other rooms

Room.Perimeter=>R.Perimeter

Wall::=W | W=>Metaline

V Door

V W1 & W2 | W1=>Wall & W2=>Wall & W1^=W2
& l1 e W1 & l2 e W2 & {l1,l2}.Continuous

*A wall is the concatenation of two continuous walls

V W | R=>Room & W e Room.Perimeter

Metaline::=S | S={L | L=>Line} & S.Continuous

Metaline.Direction=>L.Direction

Metaline.Endpoint={P1,P2} | l1 e S & l2 e S
& P1 e l1.Endpoint & P2 e l2.Endpoint
& (^L3 | l3 e S & P1 e l3.Endpoint
& P1.X < P3.X & P2.X > P3.X)

*The endpoints of a Metaline are those with the Maximum
and Minimum X-coordinates

Metaline.Length=>line(P1,P2).length

Cluster::=C | C={P | P=>Point} & P1 e C & P2 e C
& l1 | P1 e l1.Endpoint & l2 | P2 e l2.Endpoint
& L=Line(P1,P2) & l.Magnitude < F(l1.Rate,l2.Rate)

*A cluster is a set of points such that the length of a line
between any two points in the set is less than some
function of the rates of the lines of which those two
points are endpoints

(Table C, continued)

Break::=B | B=={C1,C2 | C1<=>Cluster & C2<=>Cluster & C1^=C2

& (¬L | L<=>Line & L.Endpoint e C1 & L.Endpoint e C2)

& (¬C3 | C3<=>Cluster & P1 e C1 & P2 e C2 & P3 e C3
 & L1=Line(P1,P2) & L2=Line(P1,P3) & L3=Line(P2,P3)
 & {L1,L2,L3}.Collinear
 & L1.Magnitude > L2.Magnitude
 & L1.Magnitude > L3.Magnitude)

*A break is a set of two clusters with no line having
 one endpoint in each cluster, and with no cluster
 lying between the two clusters

Break.Direction<=>L1.Direction

Door::=D | D=={C1,C2 |
 (C1<=>Cluster & (P | P e C1 & P e L1.Endpoint
 | L1 e W & W<=>Wall)
 & C2<=>Cluster & (P | P e C2 & P e L2.Endpoint | L2 e W & W<=>W
 & {C1,C2}<=>Break
 & {L1,L2}.Collinear)

*A door is a pair of clusters which have as elements the
 endpoints of a pair of collinear walls surrounding
 a break

V ((S1 | S1<=>Metline) & (S2 | S2<=>Metline)
 & {S1,S2}.Parallel
 & C1<=>Cluster
 & C2<=>Cluster
 & {C1,C2}<=>Break
 & P1 | P1 e S1.Endpoint & P1 e C1
 & P2 | P2 e S2.Endpoint & P2 e C2
 & Line(P1,P2) e R.Perimeter | R<=>Room
 & D.Direction-S1.Direction= +-Pi/2 }

*or a pair of clusters forming a break which are endpoints
 of a pair of parallel metalines such that a line
 connecting the endpoints is on the perimeter of a
 room and perpendicular to the direction of the
 metalines

(Table C, continued)

V D1 | D1=={D,L | D<=>Door & L<=>Metaline
 & C1 e I & C2 e I
 & P | P e L.Endpoint & (P e C1 V P e C2)
 & L.Direction-Door.Direction < Pi/4}

*or a door with a line projecting from one of its clusters
 at an angle less than 45 degrees

Door.Direction<=>Break.Direction

TABLE D

Symbol	Meaning
::=	
::=	is defined to be
{. . .}	is the set
	such that
V	or
&	and
~	not
e	is an element of the set
xxx.yyy	the attribute yyy of set xxx is true
==	the set is equal to
<=>	the set is similar to
(. . .)	group delimiters (to aid in reading this stuff)

REFERENCES

Ellias, T.O., J.F. Heafner, and W.L. Sibley, THE GRAIL
PROJECT: AN EXPERIMENT IN MAN-MACHINE COMMUNICATION, The
Rand Corporation, Memorandum RM-5999-ARPA, September, 1969.

Negroponte, Nicholas P., THE ARCHITECTURE MACHINE, MIT
Press, 1970.

-----, MACHINE RECOGNITION AND INFERENCE MAKING
IN COMPUTER AIDS TO ARCHITECTURE, Proposal to the National
Science Foundation, 1973.

-----, and James Taggart, "MUNCH--An Experiment
in Sketch Recognition," EDRA PROCEEDINGS, edited by William
Mitchel, January 1972.
(also in COMPUTER GRAPHICS, edited by W. Giloi, Berlin,
1971).

Saderholm, B.V., "Paper 'Keyboard' Runs Experimental IBM
System," IBM Release #33-030873, Yorktown Heights, New York,
March, 1973.

Sutherland, I.E, "Sketchpad: A Man-Machine Graphical Communications
System," AFIPS Proceedings, 1963.

Appendix I
EXISTING DATA STRUCTURE

Appendix I is reprinted from
MACHINE RECOGNITION AND INFERENCE
MAKING IN COMPUTER AIDS TO
ARCHITECTURE

EXISTING DATA STRUCTURE

In any complex system, it usually becomes true sooner or later that there is a demand for a data structure with a varying number of elements of varying size. The data structures available from Fortran are inadequate. Consequently, the General Purpose Structure was developed for users of the operating system. The structure is created and modified by a set of function calls which can be made from Fortran to the system. These permit creation and deletion of structures, and they permit the addition, subtraction, modification and accessing of data elements in the structure. The structure is created in free storage made available by supervisor calls to the operating system, and, therefore, is limited in size only by the physical size of the memory of the machine in which it resides.

The structure has three constituent elements: a Structure Pointer Block, a set of points, and a set of links. They are related in a somewhat hierarchical fashion. Since there may be more than one structure in the system being accessed at a given time, they are separately identified by a Structure Pointer Block. Each Block consists of six data elements containing information about the particular structure: a pointer to the first point of the structure; a count of the number of points in the initial point block of the structure; a flag indicating if any points have been dropped; the size of a link, or zero, if link size is not fixed; the virtual point number of the last point created; and a pointer to the next Structure Pointer Block available in the system, if there is one.

Points are currently allocated in blocks of seven at a time, in order to increase the ease with which they are manipulated. Each point consists of four data items, two of which may be accessed by the user as general purpose arguments. (Typically, they are used as the x- and y-coordinates of a point in

a display of some sort. Extending the number of arguments from two to some higher number is a simple extension being contemplated.) The other two arguments are a pointer to the first link associated with the point, and the point's number. As points are created, they are assigned a point number, sequentially from one for the first point created. In order to avoid confusion as more points are added and deleted, this number is fixed. Thus, even though a slot in a point allocation block may be reused, the numbers of the points on either side of it do not change, and the numbers of new points as they are created are monotonically increasing. The virtual point number of any newly created point will be one more than the number of the last point created as indicated by the Structure Pointer Block, and the Block will be modified accordingly.

A link is a means of establishing a relation between two points. A request to add a link between two points appends a copy of the link requested to the link chain pointed to by the third data field in a point. A link consists of at least two fields: a pointer to the point being linked to, and a pointer to the next link on the chain. The size of the links in a structure may be constant or varying. If at creation time, the structure is declared to have uniform links, the size of a link is stored in the Structure Block Pointer, and that size is used throughout. Otherwise, the size of a link is declared at the time the link is created and stored in a field of the link itself. Links may have between four and sixty addressable fields. A field is four bits wide (and thus may contain a number between 0 and 15), and fields may be addressed singly or in groups of up to four (generating a sixteen bit wide field). The first four fields in a link are used for the pointer to the point linked to. The next four are for the next link pointer, but they are not addressable. The next field contains the link size if it is non-uniform. The remaining fields are free to be used for storage of any information about the relationship between the two points desired.

In order to create a structure, the

user makes a function call of the form: PTR=CONSTR(size) where size is the size of the links in the structure if uniform, or zero if varying. CONSTR sets up a Structure Pointer Block, allocates space for the first seven points, and returns to PTR the address of the Structure Pointer Block created. All further calls referencing this structure take this pointer as their first argument. Since any call with an improper Structure Pointer Block specified can only lead to disaster, the other calls check to see whether the address handed in future argument list points to a valid Block (hence the pointer as the last element in a Block). An invalid block causes the system to type out an error message and halt. This is the only fatal error. Other errors, if they occur, are returned as results from the function, and a message is printed on an output device available to the user consisting of a two letter code. The first letter specifies in which function the error occurred, and the second letter specifies what the error was. Most errors are caused by faulty argument passing, either by requesting an impossible change or asking for information from non-existent links. In the case where a change is requested, if an error is detected, the change does not occur. In some instances the returned error code can be useful in programming. For instance, since links can be referred to by sequential number, in order to access every link associated with a given point, a counter specifying the number of the link to be accessed could be increased by one until an error returns saying there is no such link. Then the program knows it has finished. Similarly, to establish if a link between two points exists, any reference to that link specified by those two points will return an error code if no such link exists.

Other function calls are as follows:

Function Calls

To add a point to the structure
 PN=PADD(PTR,VALUE,VALUE1)
 returns the virtual point number of the point added or -1 if there was an error. Since each point can have two arguments associated with it directly, they are set by VALUE & VALUE1. The error code is Pn.

To add a link to the structure
 x=LADD(PTR,PN,PN1,SIZE)
 PN & PN1 specify the points between which the link is to be added. SIZE specifies the number of fields in the link (required whether or not the SIZE is uniform). The error code is Ln.

To access or modify an argument in a point
 x=GPOINT(PTR,PN,ARGN,VALUE)
 sets VALUE equal to the contents of ARGN of point PN. The error code is Hn.

To access or modify a set of fields in a link
 x=GPFLDR(PTR,PN,LN,NO,*WIDTH,VALUE)
 x=PPFLDR(PTR,PN,PN1,NO,WIDTH,VALUE)
 returns in VALUE the contents of the specified field(s) in the link specified by PN, LN or the link between points PN, PN1. Note that the difference between the two calls is simply that GPFLDR specifies a link by link number while PPFLDR specifies a link by passing the two points the link connects. The error code is kn.

x=GPFLDF(PTR,PN,LN,NO,*WIDTH,VALUE)
 x=PPFLDF(PTR,PN,PN1,NO,WIDTH,VALUE)
 puts VALUE truncated to the size specified by WIDTH into the specified field. The difference between the two calls is the same as above. The error code is Fn.

To delete a link from the structure
 x=LDROP(PTR,PN,LN)
 x=LLDROP(PTR,PN,PN1)
 removes the specified link. The error code is En.

To delete a point from the structure
 x=PDRGP(PTR,PN,[COLLECT])
 drops the specified point from the structure ONLY if the point has no remaining links. If COLLECT is requested, points are garbage collected after every 8 drops. The error code is Dn.

Appendix II
THE GRID FACILITIES

Appendix II is reprinted from
MACHINE RECOGNITION AND INFERENCE
MAKING IN COMPUTER AIDS TO
ARCHITECTURE

DISK GRID PACKAGE

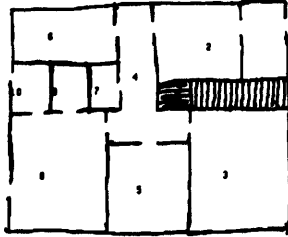
The grid package provides the FORTRAN user with a set of x-y addressable grids (up to 1024 by 1024 bits) which are used to store sketches and more generally, data from a number of graphics input terminals including the Sylvania Data Tablet for drawings made by hand and a television camera for input of predrawn sketches. This type of storage medium allows a sketch to become completely independent of time and provides a computer scratch pad for analysis of complex configurations.

Each grid is stored as a bit map on a fixed head disk storage device, with each 1024 point line represented by a 128 byte record. A grid can consist of between 1 and 1024 such lines, and any number of separate grids may be used, limited only by the capacity of the disk. When a 1024 by 1024 grid is used to represent a sketch drawn with the Sylvania Tablet, which addresses 4096 by 4096 points, a bit is set on the grid if a line drawn on the tablet passed through a square 4 by 4 tablet coordinates in size. When used with the television camera, each line of video data is stored one after another until the entire sketch or drawing is scanned.

The data on the grid is accessed by means of an assembly language program which transfers a "window" of arbitrary width and height from the grid to a FORTRAN user's array in core. Likewise, data may be transferred from a FORTRAN array to the grid. In addition, a scale may be specified so that one element of the array can represent anything from one bit in the grid to the entire grid, with the value returned equal to the number of bits set within that portion of the specified window. At the largest scale, with the array containing the entire grid, most details are too small to affect the mapping into the array, leaving only the outlines of the major forms, as when the human eye views a scene from a distance. Since an image of the original sketch is now in core,

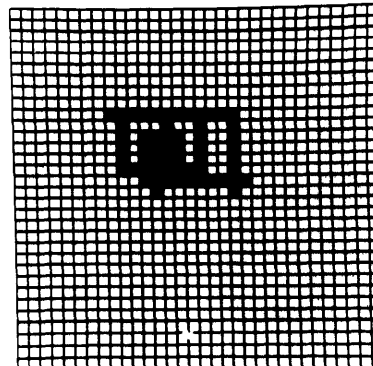
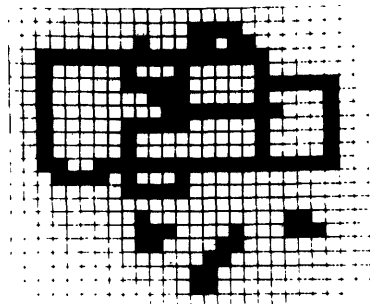
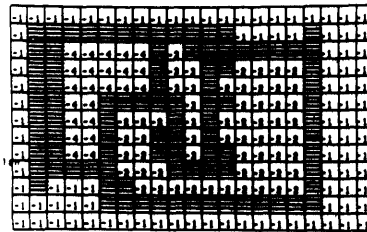
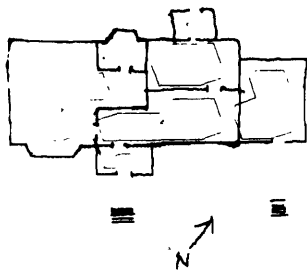
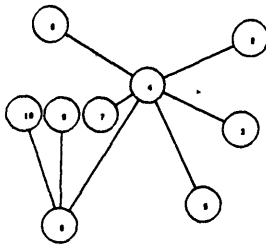
the entire sketch can be examined very easily, in much less time than it would take to scan a magnetic tape containing the original positions. Based upon the shapes, voids, bodies, etc. found by scanning the array, a program can select areas of the sketch to be examined in greater detail. As the scale, hence the size of the window, is decreased, more and more details appear. Usually there is some scale, about 3 or 4, at which most of the important features are present but at which noise from minor movements of the pen is absent. Features of the sketch found at smaller scales are usually of no significance as far as position-dependent interpretations are concerned, since they are usually the kind of noise ignored by a human examining a drawing.

Data can be entered into a grid from a variety of sources, including magnetic tape, a vidicon camera, and directly from a FORTRAN program. The most common method takes its input from a magnetic tape or disk file produced by the DRAW program in HUNCH, consisting of a list of pen coordinates measured at constant intervals. These coordinates are converted to grid coordinates and the appropriate bits set on the disk. If two successive points are more than one grid unit apart, as often happens with lines drawn at high velocity, the intervening points are interpolated, so that the bit image on the disk approximates the appearance of the completed sketch. Thus the programmer need not burden himself with the problem of connecting points, which would be very difficult in the time-independent context of the grid. Since most drawings are smaller than the maximum size of the grid, the conversion program automatically reduces the size of the grid to the size of the sketch, saving both disk space and access time. Another input source is a television camera which enables the grid to replicate a completed paper sketch. Data conversion in this case is very simple, as the vidicon scans line by line, just as the grid is organized on the disk. It is also possible to set bits in the grid directly from a FORTRAN program, by means of an assembly language routine which maps a FORTRAN array onto the disk, in a manner analogous to the



window program described above. Finally, there is a complete set of entry points which enable the assembly language programmer to set and retrieve single bits and to access individual lines in the grid.

Following are examples of 5 of the more important of the 15 FORTRAN callable entry points in the grid package.



Subroutine DISPLW

Purpose

The purpose of the subroutine is to display the contents of an array on the ARDS.

Usage

```
INTEGER*2 ARRAY(XDIM,YDIM), DENSTY, SDIM, YLIM, GRIDF, DISPF  
CALL DISPLW(ARRAY, DENSTY, XDIM, YDIM, GRIDF, DISPF)
```

The subroutine draws a grid XDIM by YDIM and fills in those squares whose corresponding array elements are not equal to zero.

ARRAY is the array.

DENSTY is the number of fill-in lines to be drawn per square. Optional default is 4.

XDIM describe the array. Both are optional. XDIM defaults to 32. YDIM defaults to XDIM.

YDIM

GRIDF specifies if the grid is to be drawn.
 = 0 no grid
 = 1 grid (the default)

DISPF allows the grid to be drawn without filling in the squares.
 = 0 no fill in display
 = 1 fill in (the default)
 - 1 fill in with the number of hits

Subroutine GRDSK

Purpose

The purpose of the subroutine GRDSK is to convert a drawing from time-dependent magnetic tape to position-dependent disk.

Usage

```
CALL GRDSK(IDISPL,ICNVRT,IERASE,GCB)
```

All arguments are optional, if omitted, the default is assumed.

If IDISPL = 0 drawing will not be displayed (default).
 = 1 drawing will be displayed as it is converted.

If ICNVRT = 0 no conversion will take place.
 = 1 conversion will take place (default).

The ICNVRT option allows the displaying to original drawing from tape.

If IERASE = 0 the drawing will be placed on the grid along with any previous drawing.
 = 1 any previous drawing will be erased (default is to the value of ICNVRT)

GCB is the Grid Control Block (see INIGRD)

DIMENSION ARRAY (XDIM, YDIM)

CALL WWWINDOW (ARRAY, SCALE, XBIAS, YBIAS, IERR, DRAW, XIIP,
YDIM, GCB)

All arguments have the same meaning as in subroutine WINDOW.

For any element of ARRAY equal to zero, no action is taken.

For any non-zero elements the corresponding bit in the grid
is turned on.

Note

Since no action is taken for ARRAY entries of zero, the
original bit value is retained for that entry in the grid.

Subroutine INIGRD

Purpose

The purpose of the subroutine INIGRD is to initialize
the disk constants table and to define one or more grids.

Usage

INTEGER*2 TAPE(66), GCB1(271), SIZE1, GCB2(79), SIZE2
SIZE1=4
SIZE2=1
CALL INIGRD (TAPE,GCB1,SIZE1,GCB2,SIZE2,...GCB4,SIZE4)

Only the first three arguments are required, allowing
the definition of one to four grids.

TAPE is an array 132 bytes, long used for magnetic tape.
GCB1 is an array (128*SIZE1) + 30 bytes long.
SIZE1 specifies the length of the buffer in lines.

Note

This program must be called before any other grid system
subroutines.

The first 6 elements of a grid control block (GCB) specify
information about the corresponding grid. If the user calls
INIGRD, these values are filled in automatically, and the
first GCB specified is established as the default for other
fortran-callable subroutines. Alternatively, the user may
fill in these values himself as follows:

BIAS	DC x'2000'	Disk address of start of grid
BADTRK	DC x'FFFF'	Can be used to specify bad track on disk
NLIB	DC 4	Number of lines in buffer (SIZE)
BOTLIN	DC 0	Number of first line in grid
TOPLIN	DC 1023	Number of top line in grid
LEFT	DC 0	
RIGHT	DC 1023	
BFST	DC A(BUFFER)	Address of buffer
	DS 14	Used by the system
BUFFER	DS SIZE*128	

Subroutine WINDOW

Purpose

The purpose of the subroutine WINDOW is to transfer a portion of the grid to a FORTRAN array.

Usage

INTEGER*2 ARRAY (XDIM,YDIM), SCALE, XBIAS, YBIAS, IERR,
DRAW, XDIM, YDIM

CALL WINDOW(ARRAY, SCALE, XBIAS, YBIAS, IERR, DRAW, X'LM,
YDIM)

The grid consists of 1024 by 1024 squares numbered as shown:

Thus one square of the grid corresponds to four units on the Sylvania tablet.

SCALE specifies how many grid squares correspond to one element of the array. The value of each element of the array is the number of grid squares "turned on" within the corresponding spot on the grid.

XBIAS specify the location of the lower left hand corner of the window.

YBIAS
IERR = 0 if all is well
= 1 if the specified parameters would force the window off the tablet. The values of XBIAS and YBIAS will be modified in the FORTRAN program to make the window fall within the range of the tablet.
= 2 if the SCALE*LARGEST_DIMENSION 1024 (the size of the tablet). The value will be adjusted.

DRAW is an optional variable.
1 = A square will be drawn on the ARDS corresponding to the portion of the original drawing included in the window.
0 = No square will be drawn (the default).

XDIM is an optional variable which specifies the dimension in the X direction. Default is 32.

YDIM is an optional variable which specifies the dimension in the Y direction. Default is to the value of X'LM.

GCB is an optional variable which specifies the grid to be used (see INIGRD).

Subroutine WWWDOW

Purpose

The purpose of subroutine WWWDOW is to write out an array on the grid.

Usage

INTEGER*2(ARRAY, SCALE, XBIAS, YBIAS, IERR, DRAW, XDIM, YDIM)

Appendix III
COMPUTER LISTINGS FOR
STRAIN

```

0000R      EXTRN  DSKTAP,SKIP,BSTORE
0000R      EXTRN  DISKST,INITAF
0000R      EXTRN  PTMAX,JNAME
0000R      EXTRN  ENCN,GETLAT
0000R      EXTRN  MARK,IODEV
0000R      EXTRN  OFFSET,TANLRV,CONSTR,PTMAX
0000R      ENTRY  PTCALC,PERROR,PRLOOP

```

```

0000      STORE   EQU    0
0002      STORE2  EQU    2

0006      INCREM  EQU    6

```

```

* CALL PTCALC(STATUS,SCRATCH.ARR,SC.LENGTH,[N
*          STATUS=>C J-CK
*          =>-1 STORAGE NOT ALLOCA
*          => 1 RAN OUT OF SPACE
*          SCRATCH.ARR=>33*6 BYTE ARRAY
*          SC.LENGTH=>SIZE OF ARRAY IN H

```

```

*
*          IF NO NUMBER, INITIALIZE & STRAIT
*          IF NUMBER NEGATIVE, ONTINUE
*          IF NUMBER NON-NEGATIVE,STRAIGHTEN
*          (0=1)

```

```

0000R 4300      B      PTCALC
0024R 0024R

0004R      SAVER   DS      32
0022R      SAV15  EQU     *-2
0024R 0000      PTCALC  STP   0,SAVER
0004R 0004R

0028R 48AF      LH      10,0(15)
0000R 0000R

002CR C500      CLHI    B0,4
0004R 0004R

0030R 4280      BL      DONE
0130R 0130R

0034R C500      CLHI    B0,8
0008R 0008R

0038R 4380      BNL     OK
0044R 0044R

003CR C800      LHI     B0,-1
FFFFR FFFF

0040R 4300      B      PERROR
0124R 0124R

0044R E110      OK      SVC   1,REW
0160R 0160R

0048R 48AF      LH      B0,4(15)

```

0004				
004CR	26A6		AIS	10, INCREM
004ER	40A0		STH	D0, OFFSET
	0000F			
0052R	48D0		LH	13, DSKTAP
	0000F			
0056R	4330		BZ	NOEN
	006ER			
005AR	48DF		LH	13, 6 (15)
	0006			
005ER	4AAD		AH	D0, 0 (13)
	0000			
0062R	4AAD		AH	D0, 0 (13)
	0000			
0066R	CBA0		SHI	10, INCREM+INCREM
	000C			
006AR	40A0		STH	10, ENDN
	0000F			
006ER	41F0	NOEN	BAL	15, CONSTR
	0000F			
0072R	0004		LC	4
0074R	015ER		DC	SIZE
0076R	40B0		STH	14, PTMAX
	0000F			
007AR	48A0		LH	D0, MARK
	0000F			*****PARKED
007ER	40A0		STH	D0, DISKST
	0000F			
		*READ IN THE FIRST BLOCK OF DATA		
0082R	4850		LH	5, DSKTAP
	0054R			
0086R	4210		BM	TAPEI
	0092R			
008AR	4220		BP	PASS
	00DCR			
008ER	4300		B	NODAT
	00EER			
0092R	C8A0	TAPEI	LHI	10, -1
	FFFF			
0096R	40A0		STH	10, DISKST
	0080R			
009AR	48F0		LH	15, SAVER+3J.
	0022R			
009ER	48AF		LH	10, 0 (15)
	0000			
00A2R	C5A0		CLHI	10, 10
	000A			
00A6R	4380		BNL	COUNT
	00B0R			
00AAR	07AA		XHR	10, 10
00ACR	4300		B	INIT
	00C4R			

00B0R 48DF	COUNT	LH	13,8(15)
0008			
00B4R 48AD		LH	10,0(13)
0000			
00B8R 4210		BM	PASS
00DCR			
00BCR C8BA		LHI	11,1(10)
0001			
00C0R 40BD		STH	11,0(13)
0000			
00C4R 41F0	INIT	BAL	15,INITAP
0000F			
00C8R C5A0	CHECK	CLHI	10,2
0002			
00CCR 4280		BL	PASS
00ECR			
00D0R 41F0		BAL	15,SKIP
0000F			
00D4R CBA0		SHI	10,1
0001			
00D8R 4300		B	CHECK
00C8R			
00DCR 48D0	PASS	LH	13,OFFSET
0050R			
00E0R 48E0		LH	14,ENDN
006CR			
00E4R 26E5		AIS	14,INCREM-1
00E6R 41F0		BAL	15,GETDAT
0000F			
00EAR 4200		NOP	ICNE
0130R			
	*SET R9 FOR FIRST LINE END		
00EER 4890	NODAT	LH	9,OFFSET
00DER			
00F2R C8A0		LHI	10,X'FOOD'
F000			
00F6R 45A9		CLH	10,STORE(9)
0000			
00FAR 4330		BE	GO
00D06R			
00FER 48A0		LH	10,DSKTAP
0084R			
0102R 4210		BM	PASS
00ECR			
0106R 2796	GO	SIS	9,INCREM
0108R 40A9		STH	10,STORE(9)
0000			
010CR C8B0		LHI	14,GOTIT
0114R			
0110R E180		SVC	8,UNAME
0000F			
0114R 07AA	GOTIT	XHR	10,DO

```

0116R 40A9          STH  10,STORE2(9)
      0002
011AR 40A0          STH  10,ESTORE
      0000F
      *GO STRAIGHTEN
011ER 4100          BAL  1,TANDRV
      0000F
0122R 07AA          XHR  10,10
0124R 48B1  PERROR LH  15,SAV15
      0022R
0128R 48BF          LH  11,2(15)
      0002
012CR 40AB          STH  10,10(11)
      0000
0130R 48B0  DONE   LH  15,LSKTAP
      0100R
0134R 4310          BNM  EXIT
      013CR
0138R 41B0          BAL  15,SKIP
      00D2R
013CR  C8B0  EXIT   LHI  15,X'0E'
      000D
0140R 48B0          LH  14,ICDEV
      0000F
0144R 9DED          SSR  14,13
0146R 4290          BTC  9,*-2
      0144R
014AR 9AEF          WDR  14,15
014CR DB00          LM   0,SAVER
      0004R
0150R 4AFF          AH   15,0(15)
      0000
0154R 030F          BR   15
      0156R C8A0  PRL0OP LHI  10,1
      0001
015AR 4300          B    PERROR
      0124R
015ER 000C  SIZE    LC   12
0160R C002  REW     DC   X'C002',0
      0000
0164R          END
NO ERRORS

```

```

* BSTORE 011CR
  CHECK  00C8F
* CONSTR 0070R
  COUNT  00B0F
* DISKST 0098R
  DONE   0130F
* DSKTAP 0132R
* ENDN   00E2F
  EXIT   013CR

```

* GETDAT	00E8R
GO	0106R
GOTIT	0114R
INCREM	0006
INIT	00C4R
* INITAP	00C6R
* IODEV	0142R
* MARK	007CF
NODAT	00EER
NOEN	006ER
* OFFSET	00F0R
OK	0044R
PASS	00DCR
* PERROR	0124R
* PRLOOP	0156R
* PTCALC	0024R
* PTMAX	0078R
REW	0160R
SAV15	0022R
SAVER	0004R
SIZE	015ER
* SKIP	013AR
STORE	0000
STORE2	0002
* TANDRV	0120R
TAPEI	0092R
* UNAME	0112R

*TANDRV,CSTART,ENDSEG,CFIX,AND INTERS USE 'CO
 * TO STORE POINTERS TO THE BEGINNING AND END
 * SEGMENTS. WHEN FILLED, THE ARRAY 'CORNER'
 * CORNER(0) POINTER TO 0ST POINT ON 1ST L
 * CORNER(2) PTR TO LAST PT ON SEGMENT BEF
 * CORNER(4) SLOPE OF THE 1ST SEGMENT(DY/
 * (WHICH MOVES DECIMAL PLACE 6 BITS
 * CORNER(6) POINTER TO 1ST POINT AFTER TH
 * 1ST POINT OF 2ND SEGMENT
 * CORNER(8) POINTER TO LAST POINT ON 2ND
 * CORNER(A) SLOPE OF 2ND SEGMENT
 * CORNER(C) POINTER TO START OF 3RD SEGME
 * THIS SERIES CAN EITHER BE CUT OFF AT CORNER
 * IF THE PEN IS LIFTED

```

0000R      EXTRN SAVEC,EOB,PUSHER,GETSQ
0000R      EXTRN CSTART,ENDSEG
0000R      EXTRN RATER,EXTRMR,SEX,SEY
0000R      EXTRN DISKST,GETDAT
0000R      EXTRN INTERV,RTESTC,RMAX2,PMAX2
0000R      EXTRN TANGNT,TRNTST,TANDIF
0000R      ENTRY TANDRV,RITER,CNEXT,NOEND
0000R      ENTRY TANDRI
0000R      EXTRN DATP,CORRECT,ELEN
0000R      EXTRN OFFSET,DSKIAP
  
```

```

0000      STORE EQU 0
0002      STORE2 EQU 2
  
```

```

0008      PTSIZE EQU 8
0006      INCREM EQU 6
  
```

```

0000R 4000      TANDRV  STH  0,RETRN
0000R 0130R
0004R 0B55      SHR  5,5      SET UP POINTER
0006R 4050      STH  5,CORNCT  FOR 'CORNER' LI
0000R 0000F
0000R 4050      STH  5,IATP
0000R 0000F
0000R 4D00      BAL  0,GETSQ
0000R 0000F
0012R 0B77      TANDRI  SHR  7,7      ENTRY FOR BEGIN
0014R 4070      STH  7,RMAX2  INIT FASTEST DW
0000R 0000F
0018R 4070      STH  7,PMAX2  INIT HARDEST PR
  
```

0000F				
001CR 4820		LH	2,CORNCT	SET 'CORNER' PO
0008R				
0020R 4819	CNEXT	LH	1,STORE(9)	AM I AT AN ENDP
0000				
0024R C910		CHI	1,X'F000'	
F000				
0028R 4220		BP	FLAGD	
0030R				
002CR 0777		XHR	7,7	
002ER 4070		STH	7,SQFLAG	
0088R				
0032R 0889		LHR	8,9	
0034R 4B80		SH	8,CFFSET	
0000F				
0038R 48D0		LH	13,DISKST	
0000F				
003CR CDD0		SLHL	13,7	
0007				
0040R 0A8D		AHR	8,13	SET UP BACK POI
0042R C410		NHI	1,X'C000'	TO BLOCK ON DIS
0000				
0046R 0618		OHR	1,8	INDICATE ENDP0I
0048R 4012		STH	1,IATP(2)	AND SAVE
000CR				
004CR 2696	PBUMP	AIS	9,INCREM	
004ER 4590		CLH	9,ENDN	
0000F				
0052R 4280		BL	LOW	
007AR				
0056R 4330		BE	LOW	
007AR				
005AR 48D0		LH	13,DSKTAP	
0000F				
005ER 4330		BZ	ENDFL	
015ER				
0062R 48D0		LH	13,OFFSET	
0036R				
0066R 48E0		LH	14,ENDN	
0050R				
006AR CAE0		AHI	14,INCREM-1	
0005				
006ER 41F0		BAL	15,GETDAT	
0000F				
0072R 4200		NOP	ENDFL	
015ER				
0076R 4890		LH	9,OFFSET	
0064R				
007AR 2671	LOW	AIS	7,1	INDICATE A STAR
007CR 4300		B	CNEXT	LOOP AGAIN
0020R				
0080R 0877	FLAGED	LHR	7,7	WHEN NOT AT END

0082R 4330		BZ	NEND	LOOK FOR START
00D8R				
0086R C870		LHI	7,0	
0000				
0088R	SQFLAG	EQL	*-2	
008AR 4220		BP	PBUMP	
004CR				
008ER C870		LHI	7,X'8000'	
8000				
0092R 4570		CLH	7,SEX	
0000F				
0096R 4330		BE	NOSQ	
00C0R				
009AR 4879		LH	7,STORE(0)	
0000				
009ER 4570		CLH	7,SEX	
0094R				
00A2R 4230		BNE	NOSQ	
00C0R				
00A6R 4879		LH	7,STORE2(9)	
0002				
00AAR 4570		CLH	7,SEY	
0000F				
00AER 4230		BNE	NOSQ	
00C0R				
00B2R 4100		BAL	1,GETSQ	
0010R				
00B6R 2471		LIS	7,1	
00B8R 4070		STB	7,SQFLAG	
0088R				
00BCR 4300		B	PEUMP	
004CR				
00C0R 4D00	NOSQ	BAL	10,SAVEC	
0000F				
00C4R C870	STEND	LHI	10,X'800'	
0800				
00C8R 40A0		STB	10,RTESTC	FORCE IMMEDIATE
0000F				
00CCR 0B55		SHR	5,5	
00CER 4100		BAL	1,NOEND	GET 1ST PT, CHE
00FOR				
00D2R 4100		BAL	1,TANGNT	GET ARCTAN FOR
0000F				
00D6R 2652		AIS	5,2	
00D8R 4D00	NEND	BAL	10,NOEND	GET NEXT PT, CH
00FOR				
00DCR 4D00		BAL	1,TANGNT	GET NEXT ARCTAN
00D4R				
00E0R 4D00		BAL	1,TRATST	CHECK FOR CHANG
0000F				
00E4R 4510		CLH	1,TANDIF	IS IT GREATER T
0000F				

00E8R 4220	BP	CSTART	YES: CORNER HAS
0000F			
00ECR 4300	B	NEND	ELSE RETURN FOR
0018R			

*

*THIS ROUTINE CHECKS FOR END OF SEGMENT FLAG
 * DATA WITHIN AN INTERVAL OF CURRENT POINTER
 * IF IT FINDS ONE, EXIT IS TO ENDSEG

00FOR 4000	NOEND	STH	0,EXIT	
015CR				
00F4R 41F0		BAL	15,PUSHER	RETURNS WITH PR
0000F				
00F8R 41F0		BAL	15,RATER	
0000F				
00FCR 0889		LHR	8,9	
00FER 08A9		LHR	D0,9	
0100R 4AA0		AH	10,INTERV	GET POINTER TO
0000F				
0104R 4818	LOOP	LH	1,STORE(8)	LOOK TO SEE IF
0000				
0108R C910		CHI	1,X'F000'	
F000				
010CR 4320		BNP	ENDSEG	
0000F				
0110R 4980		CH	8,ENDN	AM I OUT OF DAT
0068R				
0114R 4320		BNP	ENUF	NO , GO ON
013CR				
0118R 0889		LHR	11,9	YES,
011AR 41B0		BAL	15,ECE	SAVE OLD POINT
0000F				
011ER 48D0		LH	13,OFFSET	
0078R				
0122R 48B0		LH	14,ENDN	
0112R				
0126R CAE0		AHI	14,INCREM-1	
0005				
012AR 41B0		BAL	15,GETDAT	
0070R				
012ER 4200		NOP	0	
0000				
0130R	REIERN	EQL	*-2	
0132R C889		LHI	8,-INCREM(9)	
FFFA				
0136R 0898		LHR	9,8	RESET REGISTERS
0138R C8A8		LHI	D0,INTERV(8)	
0102R				
013CR 0598	ENLF	CLHR	9,8	

013ER 4330	BE	NEXT1	
0148R			
0142R 00818	LFR	13,8	
0144R 41500	BAL	15,EXTRMR	LOOK FOR SMALL
0000F			
0148R CA800	NEXT1	AHI	8,INCREM
0006			GO FOR NEW POIN
014CR 05A8	CLHR	10,8	IF PAST END
014ER 4380	ENL	LOOP	OF NEXT INTERVA
0104R			
0152R 4819	LH	1,STORE(9)	YOU'RE OK
0000			
0156R 4839	LH	3,STORE2(9)	
0002			
015AR 4300	B	*	
015AR			
015CR	EXIT	EQ	*-2
015ER 4800	ENDFL	LH	10,RETERN
0130R			
0162R 0300	BR	0	
0164R	END		

NO ERRORS

* CNEXT	0020R
* CORNCT	001ER
* CSTART	000EAF
* DATP	004AR
* DISKST	003AR
* DSKTAP	005CR
ENDFL	015ER
* ENDN	0124R
* ENDSEG	010ER
ENUF	013CR
* EOB	011CR
EXIT	015CR
* EXTRMR	0146R
FLAGED	0080R
* GETDAT	012CR
* GETSQ	00B4R
INCREM	0006
* INTERV	013AR
LOOP	0104R
LOW	007AR
NEND	000D8R
NEXT1	0148R
* NOEND	00F0R
NOSQ	00C0R
* OFFSET	0120R
PBUMP	004CR

* PMAX2	001AR
PTSIZE	0008
* PUSHER	00F6R
* RATEK	00FAF
* RETERN	0130R
* RMAX2	0016R
* RTESTC	00CAR
* SAVEC	00C2R
* SEX	00A0R
* SEY	00ACR
SQFLAG	0088R
STEND	00C4R
STORE	0000
STORE2	0002
* TANDIF	00E6R
* TANDRI	0012R
* TANDRV	0000R
* TANGNT	00DER
* TRNTST	00E2R

0000R		ENTRY GETSQ,SEX,SEY	
0007	WRK2	EQU	7
0000R 0777	GETSQ	XHR	WRK2,WRK2
0002R 4070		STH	WRK2,DATA
0068R			
0006R E110		SVC	1,RLSQ
0060R			
000AR 4870		LH	WRK2,STAT
0062R			
0000R 4310		BNM	NOEND
001R			
0012R C870	NO DATA	LHI	WRK2,X'8000'
8000			
0016R 4070		STH	WRK2,SEX
006AR			
001AR 4310		B	EXIT
005ER			
001R 4870	NOEND	LH	WRK2,DATA
0068R			
0022R 4330		BZ	NO DATA
0012R			
0026R 2772		SIS	WRK2,2
0028R 4330		BZ	SQF
0042R			
002CR E110		SVC	1,RDSQ
0060R			
0030R 4870		LH	WRK2,DATA
0068R			
0034R 2771	RDLOOP	SIS	WRK2,1
0036R 4210		BNM	GETSQ
0000R			
003AR E110		SVC	1,RLSQ
0060R			
003ER 4310		B	RDLOOP
0034R			
0042R E110	SQF	SVC	1,RLSQ
0060R			
0046R E110		SVC	1,RLSQ
0060R			
004AR 4870		LH	WRK2,DATA
0068R			
004ER 4070		STH	WRK2,SEX
006AR			
0052R E110		SVC	1,RLSQ
0060R			
0056R 4870		LH	WRK2,DATA
0068R			
005AR 4070		STH	WRK2,SEY
006CR			
005ER 0300	EXIT	BR	00

0060R	4002	RDSQ	DC	A'4002'
0062R	0000	STAT	DC	0
0064R	0068R		LC	DATA
0066R	0069R		DC	DATA+1
0068R	0000	DATA	LC	0
006AR	0000	SEX	DC	0
006CR	0000	SEY	LC	0
006ER			END	

NO ERRORS

	DATA	0068R
	EXIT	005ER
*	GETSQ	0000R
	NODATA	0012R
	NOEND	001ER
	RDLOOP	0034R
	RDSQ	0060R
*	SEX	006AR
*	SEY	006CR
	SQF	0042R
	STAT	0062R
	WRK2	0007

0000R		ENTRY	PUSHER, PMAX1, PMAX2, RPFIC
0000R		EXTRN	INTERV
0004	STORE4	EQU	
0006	INCREM	EQU	6
0000R	D0B0	PUSHER	STM 14, SAV2
	002ER		
0004R	08E9	LHR	14, 9
0006R	08F9	LHR	15, 9
0008R	4AB0	AH	15, INTERV
	0000F		
000CR	48E	LOOP	LH 10, STORE4(14)
	0004		
0010R	CC0	SRHL	10, 2
	0002		
0014R	4510	CLH	10, PMAX2
	004CR		
0018R	4280	BL	HOP1
	0020R		
001CR	4000	STH	10, PMAX2
	004CR		
0020R	26E6	HOP1	AIS 14, INCREM
0022R	05FE	CLHR	15, 14
0024R	4380	BNL	LOOP
	000CR		
0028R	D1B0	LM	14, SAV2
	002ER		
002CR	030F	BR	15
002ER	SAV2	DS	4
0032R	D0B0	RPFIC	STM 14, SAV2
	002ER		
0036R	C8B0	LHI	15, 16
	0010		
0038R		SIXTEN	EQU *-2
003AR	0BF0	SHR	15, 0
003CR	0CEB	MHR	14, 11
003ER	4DE0	DH	14, SIXTEN
	0038R		
0042R	08BF	LHR	11, 15
0044R	D1B0	LM	14, SAV2
	002ER		
0048R	030F	BR	15
004AR	0000	PMAX1	DC 11
004CR	0000	PMAX2	DC 11
004ER		END	

NO ERRORS

HOP1 0020R

INCREM 0006
* INTERV 000AF
LOOP 000CR
* FMAX1 0004AF
* PMAX2 004CR
* FUSHER 00000R
* RPFIG 0032R
SAV2 0002ER
SIXTEN 0038R
STORE4 00004

0000R	ENTRY CSTART, LINE, TANFAC, SLOPER, SLOPE
0000R	ENTRY SAVEC, THCLD
0000R	EXTRN DAIP
0000R	EXTRN NOEND, TRNTST, TANDIF, TANGNT
0000R	EXTRN TANDRI, TAN, FCORN
0000R	EXTRN CORNCT, CORNER, INTERV
0000R	EXTRN DISKST, RTESTC
0000R	EXTRN INTERH, RMAX1, RMAX2
0000R	EXTRN OFFSET

0000	STORE	EQU	0
0002	STORE2	EQU	2

*REGISTERS:

*	0--BAL
*	1--WORKING
*	2--CORNCT
*	3--
*	4--
*	9--STORE POINTER
*	11--

0006	INCREM	EQU	6
0000R 2796	CSTART	SIS	9, INCREM
0002R 2752		SIS	5, 2
0004R 4820		LH	2, CORNCT
0000F			
0008R 2622		AIS	2, 2
000AR 4100		BAL	0, SAVEC
0007ER			
000ER 4810		LH	1, DISKST
0000F			
0012R CD10		SLHL	1, 7
0007			
0016R 0809		LHR	0, 9
0018R 4B00		SH	0, OFFSET
0000F			
001CR 0AD0		AHR	1, 0
001ER 4012		STH	1, DAIP(2)
0000F			
0022R 4020		STH	, CORNCT
0006R			
0026R 4100		BAL	0, LINE
000ACR			
002AR 4100	NCEND1	BAL	0, NOEND
0000F			
002ER 4100		BAL	0, TANGNT
0000F			
0032R 4100		BAL	0, TRNTST
0000F			
0036R 0811		LHR	1, 1
0038R 4330		BZ	NCEND1

003CR	002AR 4510 0000F	CLH	1,TANDIF
0040R	4380 002AR	BNL	NCEND1
0044R	2796	SIS	9,INCREM
0046R	4820 0024R	LH	2,CORNCT
004AR	2622	AIS	2,2
004CR	4100	BAL	0,SAVEC
007ER	4800	LH	0,DISKST
0050R	0010R		
0054R	CD00 0007	SLHL	0,7
0058R	0889	LHR	11,9
005AR	4B80 001AR	SH	11,OFFSET
005ER	0A0B	AHR	10,11
0060R	4002	STH	0,DAIP(2)
0064R	4020 0048R	STH	,CORNCT
0068R	2622	AIS	2,2
006AR	C520 0008	CLHI	2,8
006CR			
006ER	4230	VIII EQU	*-2
0072R	0000F	BNE	FCORN
0076R	48B0 0000F	LH	11,RMAX2
007AR	4080 0000F	STH	11,RMAX1
007AR	4300 0000F	B	TANDRI
		**	
		*	
007ER	4010 00A4R	SAVEC	STH 1,SAV1
0082R	40A0 00A8R	STH	10,SPVA
0086R	C8A0 013CR	LHI	10,THOLI
008AR	0AA2	AHR	10,2
008CR	0AA2	AHR	10,2
008ER	4819 0000	LH	1,STORE(9)
0092R	401A 0000	STH	1,STORE(10)
0096R	4819 0002	LH	1,STORE2(9)

009AR 401A		STH	1,STORE2(10)
00002			
009ER 40A2		STH	10,CORNER(2)
0000F			
00A2R C810		LHI	1,0
00000			
00A4R	SAV1	EQU	*-2
00A6R C8A0		LHI	10,0
00000			
00A8R	SAVA	EQU	*-2
00AAR 0300		BR	1
	*		
00ACR	LINE	EQU	*
00ACR CB50	OUT	SHI	5,2
0002			
00B0R 4815		LH	1,TAN(5)
0000F			
00B4R 0855		LHR	5,5
00B6R 4320		BNP	OKT
00C2R			
00BAR C510		CLHI	1,X'64AE'
64AE			
00BCR	TWOPI	EQU	*-2
00BER 4330		BE	OUT
00ACR			
00C2R 40D0	OKT	STH	1,TAN
00B2R			
00C6R 4820		LH	2,CORNCT
0066R			
00CAR 41D0		BAL	15,SLOPER
00D6R			
00CER 4020		STH	2,CORNCT
00C8R			
00D2R 2452		LIS	5,2
00D4R 0300		BR	1
	*		
	*		
00D6R 40F0	SLOPER	STH	15,H15
00F4R			
00DAR 4812		LH	1,CORNER(2)
00A0R			
00DER 4831		LH	3,STORE(1)
0000			
00E2R 4851		LH	5,STORE2(1)
0002			
00E6R 2722		SIS	2,2
00E8R 41F1		BAL	15,SLOPE1
00F6R			
00ECR 2624		AIS	2,4
00EER 4052		STH	5,CORNER(2)
00DCR			
00F2R 4300		B	*

00F4R	00F2R	H15	EQU	*-2
		*		
		*		
00F6R	4812	SLCPE1	LH	1,CORNER(2)
	00FOR			
00F8R	4B51		SE	5,STORE2(1)
	0002			
00FER	4B31		SE	3,STORE(1)
	0000			
0102R	4230		BNZ	FINITE
	010CR			
0106R	2431		LIS	3,1
0108R	4C40		MH	4,VIII
	006CR			
010CR	4C40	FINITE	MH	4,TANFAC
	013AR			
0110R	0874	LOOP1	LHR	7,4
0112R	0885		LHR	8,5
0114R	0D43		DHR	4,3
0116R	0585		CLFR	8,5
0118R	4230		BNE	OK
	0138R			
011CR	0574		CLHR	7,4
011ER	4230		BNE	OK
	0138R			
0122R	0844		LHR	4,4
0124R	4330		BZ	OK
	0138R			
0128R	4210		BZ	MINUS
	0134R			
012CR	C850		LHI	5,X'7FFF'
	7FFF			
0130R	4300		B	OK
	0138R			
0134R	C850	MINUS	LHI	5,X'8000'
	8000			
0138R	030F	OK	BR	15
		*		
013AR	0040	TANFAC	DC	X'40'
013CR		THOLD	IS	32
015CR			END	

NO ERRORS

* CORNCT 00D0R
 * CORNER 00F8R
 * CSTART 0000R
 * DATP 0062R
 * DISKST 0052R
 * FCORN 0070R

FINITE	010CR
H15	00F4F
INCREM	0006
**INTERH	0000
**INTERV	0000
* LINE	00ACF
LOOP1	0110R
MINUS	0134F
NCEND1	002AR
* NOEND	002CF
* OFFSET	005CR
OK	0138F
OKT	00C2R
OUT	00ACF
* RMAX1	0078R
* RMAX2	0074F
**RTESTC	0000
SAV1	00A4F
SAVA	00A8R
* SAVEC	007EF
* SLOPE1	00F6R
* SLOPER	00D6F
STORE	0000
STORE2	0002
* TAN	00C4R
* TANDIF	003EF
* TANDRI	007CR
* TANFAC	013AF
* TANGNT	0030R
* THOLD	013CF
* TRNTST	0034R
TWOPI	00BCF
VIII	006CR

```

0000R      EXTRN STABS,ATANUR
0000R      ENTRY TAN,DIST,INTERH
0000R      ENTRY TANGNT,TRNTST,IN ERV,TANDIF

```

```

0006      INCREM EQU 6

```

```

0000      STORE EQU 11
0002      STORE2 EQU 2

```

```

*
*
* TANGNT:  READS POINTS FROM STORE, FINDS DEL
*          SEGMENTS, FINDS ARCTANGN1 OF ANGLE
*          AND STORES IT IN A LIST OF ARCTANGENTS
*          R1 HAS STORE(9)
*          R3 HAS STORE2(9)
*          R9 HAS INDEX FROM POINTS
*          R5 HAS TAN INDEX
*          R4 WORKING
*          R15 BAL ATANUR,STABS
*          R0 RETURN ADDRESS
*
*

```

```

0000R 4A90      TANGNT AH 9,INTERV
0106R
0004R 4B39      SH 3,STORE2(9) DELTA Y
0002
000BR 4B19      SH 1,STORE(9) DELTA X
0000
000CR 4230      BNZ FINITE
0028R
0010R 2411      LIS 1,1
0012R 0833      LHR 3,3
0014R 4230      BNZ ENLARG
0024R
0018R C810      NIL LHI 1,X*64AE
64AE
001AR
001CR 4015      TWOPI EQU *-2
00C2R          STH 1,TAN(5)
0020R 4300      B ENIT
0054R
0024R CF30      ENLARG SLHA 3,4
0004
0028R 41E0      FINITE BAL 15,DIST
00A8R
002CR 40E0      STH 1,DELTAX
010AR
0030R 4030      STH 3,DELTAY
010CR
0034R 41E0      BAL 15,ATANUR
0000F

```

0038R	010CR	DC	A(DELTA Y)
003AR	0D0AR	DC	A(DELTA X)
003CR	010ER	DC	A(ANGLE)
003ER	4810	LH	1,ANGLE
	010ER		
0042R	4830	LH	3,ANGLE+2
	0110R		
0046R	CF10	SLHA	1,12
	000C		
004AR	CE30	SRHA	3,3
	0003		
004ER	0A13	AHR	1,3
0050R	4015	STH	1,TAN(5)
	00C2R		
0054R	C450	ENDT	NHI 5,X'3F'
	003F		
0058R	4015	STH	1,TAN(5)
	00C2R		
005CR	4B90	SH	9,INTERV
	0D06R		
0060R	2696	AIS	9,INCREM
0062R	0300	BR	"

*TRNTST: LOOKS AT A LIST OF ARCTANGENTS TO S
 * A TURN OCCURS SOMEWHERE WITHIN THEM
 *

*USES REGS 0(RETURN),1,2,5(WHICH HAS VALUE OF
 * TAN CHECKED),15(BAL)
 *

0064R	0845	TRNTST	LHR 4,5
0066R	4815		LH 1,TAN(5)
	00C2R		
006AR	4910		CH 1,TWOPI
	001AR		
006ER	4380		BNL ZERO
	00A2R		
0072R	2742	PILOOP	SIS 4,2
0074R	42D0		BM ZERO
	00A2R		
0078R	4814		LH 1,TAN(4)
	00C2R		
007CR	45D0		CLH 1,TWOPI
	001AR		
0080R	4330		EE PILOOP
	0072R		
0084R	4B15		SH 1,TAN(5)
	00C2R		
0088R	41B0		BAL 15,STABS
	0000F		
008CR	0821		LHR 2,1
008ER	4B10		SH 1,TWOPI
	001AR		

0092R	41B		BAL	15,STABS
	008AR			
0096R	0521		CLHR	2,1
0098R	4380		BNL	END
	009ER			
009CR	0812		LHR	1,2
009ER	2652	END	AIS	5,2
00A0R	0300		BR	

00A2R	0B11	ZERO	SHR	1,1
00A4R	4300		B	END
	009ER			

★

		★		
00A8R	0873	DIST	LHR	7,3
00AAR	08D1		LHR	13,1
00ACR	0CCD		MHR	12,13
00AER	0C67		MHR	6,7
00B0R	0AD7		AHR	13,7
00B2R	0EC6		ACHR	12,6
00B4R	08CC		LHR	12,12
00B6R	023F		BNZR	15
00B8R	C5D0	DISTFL	CLHI	13,X'10'
	0010			
00BCR	4280		BL	NIL
	0018R			
00C0R	030F		BR	15

★

★

00C2R		TAN	DS	66
0104R		TANDIF	IS	2
0106R		INTERV	DS	2
0108R		INTERH	IS	2
010AR		DELTAX	DS	2
010CR		DELTAY	IS	2
010ER		ANGLE	DS	2
0110R		ANGLE2	IS	2
0112R			END	

NO ERRORS

	ANGLE	010ER
	ANGLE2	0110R
★	ATANUR	0036R
	DELTAX	010AR
	DELTAY	010CR
★	DIST	00A8R
	DISTFL	00B8R
	END	009ER

ENDT	0054R
ENLARG	0024R
FINITE	0028R
INCREM	0006
* INTERH	0108R
* INTERV	0106R
NIL	0018R
FILOOP	0072R
* STABS	0094R
STORE	0000
STORE2	0002
* TAN	00C2R
* TANDIF	0104R
* TANGNT	0000R
* TRNTST	0064R
TWOPI	001AR
ZERO	00A2R

```

*RATER READJUSTS RATES EVERY 12 POINTS
*THIS VERSION USES A GENERAL FUNCTION
*- INTERH = INTERV = INTERVAL BETWEEN SLOPE C
* DELTH1 = DELT11 = DX FOR END TO END MATCH
*- DELTH2 = DELT22 = DY FOR END TO END MATCH
* DELTH3 = DELTA3 = Y DIFF IN CHECK (CFIX)
*- DELTH4 = DELTA4 = DX FOR INTERSECTIONS
* DELTH5 = DELTA5 = DY FOR INTERSECTIONS
* TANDIH = TANDIF = ANGLE CHANGE PERMISSIBLE
*
*
* DELTA1-3: 1/32 IN. TO 3/8 IN.
*          12 COUNTS + 8(RATE)
* DELTA4-5: 1/16 IN. TO 3/8+ IN.
*          23 COUNTS + 8(RATE)
* INTERV = 4 * (18 - R)/2 (SKIP) TO 8 POINT
*- TANDIF = PI/90 + (PI/90 * R) 14 TO 34 DEGR
*
*

```

```

0005      INCREM EQU 5

0000R      EXTRN INTERV,RFFIG
0000R      EXTRN INTERH,TANLIF
0000R      ENTRY TANDIH,RATER,RMAX1,RMAX2,RTESTC
0000R      ENTRY T1,EIGHTH
0000R      EXTRN FUNCT,OFFSET

0000      STORE EQU 0
0002      STORE2 EQU 2

0000R 48A0    RATER LH 00,RTESTC      HAV ENOUGH POIN
010AR
0004R 4AA0    AH 00,INTERV      ELAPSED SINCE I
0000F
0008R 40A0    STH 00,RTESTC      REFIGURED THE P
010AR
000CR 45A0    CLH 00,RTEST      IF NO
010CR
0010R 028F    BLR 15      RETURN
0012R 4000    STH 0,ST0      ELSE SET UP FOR
000ER
0016R 4090    STH 9,ST3
0002R
001AR 40F0    STH 15,ST15
0006R
001ER C8A9    LHI 10,-INCREM-INCREM(9)
FFF4
0022R 0766    XHR 6,6
0024R 4060    STH 6,LENGTH
0110R
0028R C800    LHI 12,X*8000

```

	F000				
002CR	2796	LOWER	SIS	9, INCREM	LOOK AHEAD
002ER	459		CLH	9, OFFSET	
	0000F				
0032R	4280		BL	UP1	
	0046R				
0036R	49C9		CH	12, STORE(9)	
	0000				
003AR	4310		BNM	UP1	
	0046R				
003ER	05A9		CLFR	D0, 9	
0040R	4280		BL	LOWER	
	002CR				
0044R	2102		BTFS	D, 2	
0046R	2696	UP1	AIS	9, INCREM	
0048R	48B9	LOOP1	LH	11, STORE(9)	OTHERWISE
	0000				
004CR	48D9		LH	13, STORE2(9)	STEP THRU GETLE
	0002				
0050R	2696		AIS	9, INCREM	TO COME JP
0052R	41F0		BAL	15, GETL	WITH RATE
	00C8R				
0056R	C560		CLHI	6, 4	
	0004				
005AR	4280		EL	LOOP1	
	0048R				
005ER	48D0	NOW	LH	13, LENGTH	
	0110R				
0062R	C8B0		LHI	11, 16	
	0010				
0066R	0CCB		MHR	12, 11	
0068R	4DC0		DH	12, EIGHTH	
	00E8R				
006CR	0ACC		AHR	12, 12	
006ER	4500		CLH	12, EIGHTH	
	00E8R				
0072R	2382		BFFS	8, 2	
0074R	26D1		AIS	13, 1	
0076R	081D		LHR	13, 13	
0078R	2232		BFBS	3, 2	
007AR	0CA6		MHR	D0, 6	
007CR	0DAD		DHR	10, 13	
007ER	086B		LHR	6, 11	
0080R	C8B0	LONG	LHI	11, 16	
	0010				
0084R	C560		CLHI	6, 16	
	0010				
0088R	4320		BNP	RATEJP	
	0090R				
008CR	C860		LHI	6, 16	
	0010				
0090R	0BB6	RATEUP	SHR	11, 6	ALL HAS RATE

0092R 41F0		BAL	15,RPFIG	
0000F				
0096R 4580		CLH	11,RMAX2	
00114R				
009AR 4280		BL	PARAMS	
000A2R				
009ER 4080		STH	11,RMAX2	
00114R				
00A2R 4100	PARAMS	BAL	0,T1	
000F4R				
00A6R 40D0		STH	13,TANDIH	
00D0ER				
00AAR C8D0	SAV12	LHI	13,INCREM+INCREM	
0000C				
00AER 40D0		STH	13,INTERV	
00006R				
00B2R 40D0		STH	13,INTERH	
00000F				
00B6R 0B99		SHR	9,9	
00B8R 4090		STH	9,RTESTC	
010AR				
00BCR C800		LHI	10,0	
0000				
00BER	ST0	EQL	*-2	
00C0R C890		LHI	9,0	
0000				
00C2R	ST9	EQU	*-2	
00C4R 4300		B	*	
00C4R				
00C6R	ST15	EQL	*-2	
00C8R 2661	GETL	AIS	6,1	
00CAR 48A9		LH	10,STORE(9)	
0000				
00CER C9A0		CHI	10,X'E000'	
F000				
00D2R 4320		ENP	NOW	
005ER				
00D6R 0BEA		SHR	11,D0	
00D8R 4BD9		SH	13,STORE2(9)	
0002				
00DCR 0CAB		MHR	10,11	
00DER 0CCD		MHR	12,13	
00E0R 0ABD		AHR	11,13	
00E2R 4AB		AH	11,LENGTH	
0110R				
00E6R C5B0		CLHI	11,X'1300'	(X'45' C(UNITS)*)
1300				
00E8R	EIGHTH	EQL	*-2	

000EAR	4380		BNL	LONG
	0080R			
000EER	40B0		STH	11,LENGTH
	0110R			
000F2R	03CF		BR	15

00F4R	41F0	T1	BAL	15,FUNCT
	0000F			
00F8R	0000	AT1	DC	0
00FAR	0064		DC	000
00FCR	0000	BT1	DC	0
00FER	0064		DC	000
0100R	0300	CT1	DC	768
0102R	0064		DC	000
0104R	028A	DT1	DC	650
0106R	0000F		DC	A(TANDIF)
0108R	0300		BR	

010AR		RIESTC	DS	2
010CR	0030	RTEST	DC	48
010ER		TANDIH	DS	2
0110R		LENGTH	DS	2
0112R		RMAX1	DS	2
0114R		RMAX2	DS	2
0116R			END	

NO ERRORS

	AT1	00F8R
	BT1	00FCR
	CT1	0100R
	DT1	0104R
*	EIGHTH	00E8R
*	FUNC1	00F6R
	GETL	00C8R
	INCREM	0006
*	INTERH	00B4R
*	INTERV	00B0R
	LENGTH	0110R
	LONG	0080R
	LOOP1	0048R
	LOWER	002CF
	NOW	005ER
*	OFFSET	0030R
	PARAMS	00A2R
*	RATER	0000R
	RATEUP	0090R

* RMAX1	0112R
* RMAX2	0114R
* RPFIC	0094R
RTEST	010CR
* RTESTC	010AR
SAV12	000AA
ST0	00BER
ST15	00C6R
ST9	00C2R
STORE	0000
STORE2	0002
* T1	00F4R
* TANDIF	0106R
* TANDIH	010ER
UP1	0046R

0000R

ENTRY FUNCT

*FUNCTION---R11 = RATE

* F(RATE)=AA'(RATE)**3+B '(RATE)**2+CC'(RAT

* WHERE P', B', & C' HAVE NEGATIVE EXPONENT

0000R 08DB

FUNCT LHR 13,11

0002R 0CCB

MHR 12,11

0004R 0CCB

MHR 12,11

0006R 4CCF

MH 12,0(15)

0000

0008R 4DCF

DH 12,2(15)

0002

000ER 08AC

LHR 10,13

0010R 08DB

LHR 13,11

0012R 0CCB

MHR 12,11

0014R 4CCF

MH 12,4(15)

0004

0018R 4DCF

DH 12,6(15)

0006

001CR 0AAD

AHR 10,13

001ER 08CB

LHR 13,11

0020R 4CCF

MH 12,8(15)

0008

0024R 4DCF

DH 12,10(15)

000A

0028R 4ADF

AH 13,12(15)

000C

002CR 0ADA

AHR 13,10

002ER 48CF

LH 12,14(15)

000E

0032R 40LC

STH 13,0(12)

0000

0036R 43CF

B 16(15)

0010

003AR

END

NO ERRORS

* FUNCT 0000R

```

0000R      EXTRN PMAX1,PMAX2
0000R      EXTRN RMAX1,RMAX2
0000R      EXTRN CORNCT,INTERV,CORNER,LINE
0000R      EXTRN CFIX,LINER
0000R      EXTRN TANDRI,INTERH,DATP
0000R      EXTRN LISKST,CISA
0000R      ENTRY GOTBAC,LINER2,CENDER
0000R      ENTRY LINER2,HOLDEN
0000R      ENTRY ENDSEG,FCORN
0000R      EXTRN THOL1,BSTORE
0000R      EXTRN OFFSET

```

```

0000      STORE EQU 0
0002      STORE2 EQU 2
0004      STORE4 EQU 4
000C      STOREC EQU 12

0006      INCREM EQU 6

0000R 482C      ENDSEG LH 2,CORNCT
0000F
0004R 4080      STH 8,HOLD
000BCR
0008R 2686      AIS 8,INCREM
000AR 4898      LH 9,STORE(8)
0000
0000R C990      CHI 9,X'0000' IF TWO FLAGS, G
F000
00012R 4220      BP SKIP
00018R
00016R 0819      LHR 1,9 INTO REGISTER
00018R CB80      SKIP SHI 8,INCREM+INCREM
0000C
0001CR 0898      LHR 9,8 R9 POINTS 2 POI
0001ER C410      NHI 1,X'0000'
C000
00022R 4800      LH 0,LISKST
0000F
00026R CD00      SLHL 0,7
0007
0002AR 0869      LHR 6,9
0002CR 4B60      SH 6,OFFSET
0000F
00030R 0616      OHR 1,6
00032R 0610      OHR 1,0
00034R C520      CLHI 2,10
0000A
00038R 4330      BE GOT1
0005AR
0003CR C520      CLHI 2,4
0004

```

0040R 4330		BE	GOT1
005AR			
0044R 2622		AIS	2,2
0046R 4092		STH	9,CORNER(2)
0000F			
004AR 4012		STH	1,DATP(2)
0000F			
004ER 4020		STH	,CORNCT
00002R			
0052R 4100		BAL	1,LINE
0000F			
0056R 4300		B	GOT2
0062R			
005AR 2722	GOT1	SIS	2,2
005CR 4012		STH	1,IATP(2)
004CR			
0060R 2622		AIS	2,2
0062R C520	GOT2	CLHI	2,4
0004			
0066R 4330		BE	LINER2
00A2R			
006AR 4100	GOTTEN	BAL	0,CFSA
0000F			
006ER 4100		BAL	1,CFIX
0000F			
0072R 00A2R		DC	LINER2
0074R 4860	GOTEAC	LH	6,PMAX1
0000F			
0078R CD60		SLHL	6,4
0004			
007CR 4660		OF	6,RMAX1
0000F			
0080R 0B22		SHR	2,2
0082R 4100		BAL	1,LINER
0000F			
0086R 2626		AIS	2,6
0088R 4860		LH	6,PMAX2
0000F			
008CR CD60		SLHL	6,4
0004			
0090R 4660		OF	6,RMAX2
0000F			
0094R 4D00		BAI	1,LINER
0084R			
0098R 0B22		SHR	2,2
009AR 4020		STH	,CORNCT
0050R			
009ER 4300		B	ENDED
00EAR			
00A2R 4860	LINER2	LH	6,PMAX2
008AR			
00A6R CD60		SLHL	6,4

0004			
000AR 466		OH	6,RMAX2
0002R			
000AER 0B22		SHR	2,2
00B0R 4100		BAL	1,LINER
00096R			
00B4R 0B22		SHR	2,2
00B6R 4020		STH	2,CORNCT
009CR			

★

00BAR C890	ENDED	LHI	9,0
00000			
00BCR	HOLDEN	EQU	*-2
00BCR	HOLD	EQU	*-2
00BER 4020		STH	2,DATP
005ER			
00C2R 0711		XHR	1,1
00C4R 40B0		STH	1,ESTORE
0000F			
00C8R 48B0	ENDER	LH	1,INTER
0000F			
00CCR 40B0		STH	1,INTERV
0000F			
00D0R 4300		P	TANDEI
0000F			

★

★

00D4R 4D00	FCCRN	EAL	0,CFSA
006CR			
00D8R 4D00		BAL	0,CFIX
0070R			
00DCR 014CR		DC	CENDER
00DER 4860		LH	6,PMAX1
0076R			
00E2R CD60		SLHL	5,4
0004			
00E6R 4660		OH	6,RMAX1
007ER			
00EAR 4820		LH	2,RMAX2
00ACR			
00EER 4020		STH	2,RMAX1
00E8R			
00F2R 4820		LH	2,PMAX2
00A4R			
00F6R 4020		STH	2,PMAX1
00E0R			
00FAR 0B22		SHR	2,2
00FCR 4D00		BAL	0,LIN R

000E2R			
0100R 2626		AIS	2,6
0102R 0744		XHR	4,4
0104R 4832	SHLOOP	LH	3,DATP(2)
000C0R			
0108R 4034		STH	,DATP(4)
000D6R			
010CR 2642		AIS	4,2
010ER 2622		AIS	2,2
0110R C520		CLHI	2,14
0000E			
0114R 4280		BL	SHLOOP
000D4R			
0118R 2742		SIS	4,2
011AR 4040		STH	4,CORNCT
000B8R			
011ER C830		LHI	3,THOLD
0000F			
0122R D0B0		STM	14,SAV2
0152R			
0126R 0722		XHR	2,2
0128R D1E3	STLOOP	LM	14,X'C'(3)
0000C			
012CR D0E3		STM	14,0(3)
00000			
0130R 2634		AIS	3,4
0132R 2621		AIS	2,1
0134R C520		CLHI	2,4
00004			
0138R 4280		BL	STLOOP
0128R			
013CR 2454		LIS	5,4
013ER 242A		LIS	2,10
0140R 4832		LH	3,CORNER(2)
00048R			
0144R 4035		STH	,CORNER(5)
0142R			
0148R D1E0		LM	14,SAV2
0152R			
014CR 08B9	CENDER	LHR	11,9
014ER 4300		B	ENDER
000C8R			
	*		
0152R	SAV2	DS	
0156R		END	

NO ERRORS

* BSTORE 00C6R
 * CENDER 014CR
 * CFIX 00DAR
 * CFSA 00D6R
 * CORNCT 011CR

* CORNER	0146R
* DATF	0010AR
* DISKST	0024R
ENDEL	000BAR
ENDER	00C8R
* ENDSEG	00000R
* FCORN	00D4R
GOT1	0005AR
GOT2	0062R
* GOTBAC	00074R
GOTTEN	006AR
HOLD	000BCR
* HOLDEN	00BCR
INCREM	00006
* INTERH	00CAR
* INTERV	00CEB
* LINE	0054R
* LINER	000FEF
* LINER2	00A2R
* OFFSET	0002EB
* PMAX1	00F8R
* PMAX2	00F4R
* RMAX1	00F0R
* RMAX2	000ECB
SAV2	0152R
SHLOOP	0104R
SKIP	0018R
STLOOP	0128R
STORE	0000
STORE2	00002
STORE4	0004
STOREC	0000C
* TANDRI	00D2R
* THOLD	0120R

0000R		ENTRY	CFSA,MINIM	
0000R		EXTRN	ATANUR,CORNER,DATP,BSTORE	
0000R		EXTRN	STABS,INDEX,STCREX,STOREY	
0000R		EXTRN	KIT,CFRTRN,PTMAX,GPOINT	
0000	STORE	EQU	0	
00002	STORE2	EQU	2	
3258	PI	EQU	X'3258'	
64AE	TWOPI	EQU	X'64AE'	
0008	CORNR	EQU	8	
0000R 4000	CFSA	STH	BACK+2	
0015CR				
0004R 2604		AIS	0,4	
0006R 4000		STH	CFRTRN	IN CFIX
0000F				
000AR C880		LHI	CORNR,CORNER	
0000F				
000ER 4828		LH	2,2(CORNR)	
0002				
0012R 4832		LH	3,STORE(2)	
0000				
0016R 4842		LH	4,STORE2(2)	
0002				
001AR 4828		LH	2,2(CORNR)	
0000				
001ER 4800		LH	1,PTMAX	
0000F				
0022R 4811		LH	1,6(1)	
0006				
0026R 0512		CLER	1,2	
0028R 4280		BL	GOON1	
0058R				
002CR 4020		STH	2,ANGLE	
0190R				
0030R 41F0		BAL	15,GPOINT	
0000F				
0034R 000A		DC	10	
0036R 0020R		DC	PTMAX	
0038R 0190R		DC	ANGLE	
003AR 0188R		DC	CNE	
003CR 018CR		DC	DELTAX	
003ER 41F0		BAL	15,GPOINT	
0032R				
0042R 000A		DC	10	
0044R 0036R		DC	PTMAX	
0046R 0190R		DC	ANGLE	
0048R 018AR		DC	TWO	
004AR 018ER		DC	DELTAY	
004CR 4B30		SH	DELTAX	

0050R	018CR 4B4'	SH	4,DEITAY	
0054R	018ER 4300	B	GOTAN	
0058R	0050R 4B32	GOON1 SH	3,STORE(2)	
005CR	0000 4B42	SE	4,STORE2(2)	
0060R	0002 4D00	GOTAN BAL	1,ATNR	
0064R	0162R 0815	LHR	1,5	
0066R	4828	LH	2,6(CORNR)	
006AR	0006 4832	LH	3,STORE(2)	
006ER	0000 4842	LH	4,STORE2(2)	
0072R	0002 4828	LH	2,8(CORNR)	
0076R	0008 4B32	SH	3,STORE(2)	
007AR	0000 4B42	SH	4,STORE2(2)	
007ER	0002 4100	BAL	1,ATNR	
0082R	0162R 0B15	SHR	1,5	
0084R	0B15 41B0	BAL	15,STABS	
0088R	0000F C5D0	CLHI	1,PI	
008CR	3258 4280	BL	COMPAR	
0090R	0098R C850	LHI	5,TWOPI	
0094R	64AE 0B51	SHR	5,1	
0096R	0815	LHR	1,5	
0098R	C5D0 0600	COMPAR CLHI	1,X'000'	^^20 DEGREES
009AR	009CR 4280	MINIM EQU	*-2	
00A8R	00A8R C810	BL	GOON	
00A4R	FFFF 4300	LHI	1,-1	
00A8R	00AAR 0711	B	GOON+2	
00AAR	00AAR 4010	GOON XHR	1,1	
00AER	0158R 4010	STH	1,SFLAG+2	
00B2R	010ER 4810	STH	1,NO+X+2	
		LH	1,ESTORE	

0000F				
00B6R 4330		BZ	NOEX	
010CR				
0008	DAT	EQU	8	
00BAR C880		LHI	IAT, IAT	
0000F				
00BER 4838		LH	3,0(IAT)	
0000				
00C2R C430		NHI	3,X'3FFF'	
3FFF				
00C6R 4828		LH	2,2(IAT)	
0002				
00CAR C420		NHI	2,X'3FFF'	
3FFF				
00CER 0A32		AHR	3,2	
00D0R CC30		SRHL	3,1	GET 'AVERAGE' I
0001				
00D4R 4848		LH	4,6(DAT)	
0006				
00D8R C440		NHI	4,X'3FFF'	
3FFF				
00DCR 4828		LH	2,8(DAT)	
0008				
00E0R C420		NHI	2,X'3FFF'	
3FFF				
00E4R 0A42		AHR	4,2	
00E6R CC40		SRHL	4,1	GET 'AVERAGE' I
0001				
00EAR 0722		XHR	2,2	
00ECR 4532	CLOOP	CLH	3,INDEX(2)	
0000F				
00FOR 4280		BL	HNEXT	
00FCR				
00F4R 4020		STH	1,NOEX+2	
000ER				
00F8R 4300		B	NEXT	
0104R				
00FCR 4542	HNEXT	CLH	4,INDEX(2)	
000ER				
0100R 4380		BNL	FOUND	
0124R				
0104R 2622	NEXT	AIS	2,2	
0106R 0521		CLHR	2,1	
0108R 4280		BL	CLOOP	
000ECR				
010CR C820	NOEX	LHI	2,0	
0000				
0110R 2412		LIS	1,2	
0112R 0A21		AHR	2,1	
0114R 4841		LH	4,CORNER(1)	
000CR				
0118R 4874		LH	7,STORE(4)	

011CR	0000 4834 0032		LH	3,STORE2(4)
0120R	4300 012CR		B	SHIFT
0124R	4872 0000F	FOUND	LH	7,STOREX(2)
0128R	4832 0000F		LH	3,STOREY(2)
012CR	0744	SHIFT	XHR	4,4
012ER	2452		LIS	5,2
0130R	4862 0126R	CLOOP1	LH	6,STOREX(2)
0134R	4064 0132R		STH	6,STOREX(4)
0138R	4862 012AR		LH	6,STOREY(2)
013CR	4064 013AR		STH	6,STOREY(4)
0140R	4862 00FER		LH	6,INDEX(2)
0144R	4064 0142R		STH	6,INDEX(4)
0148R	0A45		AHR	4,5
014AR	0A25		AHR	2,5
014CR	0521		CLHR	2,1
014ER	4280		BL	CLOOP1
0152R	0130R 4040 00B4R		STH	4,BSTORE
0156R	C840 0000	SFLAG	LHI	4,0
015AR	4210 015AR	BACK	BM	*
015ER	4300 0000F		B	KIT
0162R	4030 018CR	ATNR	STH	,DELTAX
0166R	4040 018ER		STH	4,DELTAY
016AR	41F0 0000F		BAL	15,ATANOR
016ER	018CR		DC	DELTAX
0170R	018ER		DC	DELTAY
0172R	0190R		DC	ANGLE
0174R	4850 0190R		LH	5,ANGLE
0178R	4860 0192R		LH	6,ANGLE+2
017CR	CF50		SLHA	5,12

0130R	000C			
	CE60		SRHA	5,3
	0003			
0184R	0A56		AHR	5,6
0186R	0300		BR	11
0188R	0001	ONE	DC	1
018AR	0012	TWO	DC	2
018CR	0000	DELTA	IC	0
018ER	0000	DELTA	DC	1
0190R	0000	ANGLE	IC	00,00
	0000			
0194R			END	

NO ERRORS

	ANGLE	0190R
*	ATANUR	016CF
	ATNR	0162R
	BACK	015AF
*	BSTORE	0154R
*	CFRTRN	0008R
*	CFSA	0000R
	CLOOP	00ECF
	CLOOP1	0130R
	COMPAR	0098R
*	CORNER	0116R
	CORNR	00008
	DAT	0008
*	DATP	00BCF
	DELTA	018CR
	DELTA	018ER
	FOUND	0124R
	GOON	000A8R
	GOON1	0058R
	GOTAN	00060R
*	GPOINT	0040R
	HNEXT	000FCF
*	INDEX	0146R
*	MINIM	009AF
	NEXT	0104R
	NOEX	010CF
	ONE	0188R
	PI	3258
*	PTMAX	0044R
	SFLAG	0156R
	SHIFT	012CR
*	STABS	0086R
	STORE	0000
	STORE2	0002
*	STOREX	0136R
*	STOREY	013ER
	TWO	018AR

TWOPI 64AE
* XIT 0160F

```

*ENTRY BAL DC,CFIX
*  THIS ROUTINE TAKES 2 LINE SEGMENTS
*  IN 'CORNER' & CALCULATES THEIR INTERSEC
*  THE INTERSECTION IS STORED HIGH IN
*  STORE AND THE POINTERS TO THE INTERSECT
*  IN CORNER ARE ALTERED TO IT
*
*R0--BAL; B & D FROM BD
*R1--WRKG
*R2--WRKG,PTR TO CORNER IN SLOPED
*R3--B,D FROM BD; Y FROM BY,DY
*R4--WRKG
*R5--WRKG
*R6--WRKG; PTR TO HOLD IN SLOPED
*R7--WRKG
*R12-BAL SETTER
*R14-BAL SLOPED
*R15-BAL SLOPE1,STABS,BD,BY,DY

```

```

*  Y1=AX1+B
*  Y2=CX2+D
*  AT INTERSECTION:
*  Y1=Y2
*  X1=X2
*  THEREFORE: AX+B=CX+D AT INTERS
*  AX-CX=D-B
*  X(A-C)=D-B
*  X(INTER)= (D-B)/(A-C)
*  Y(INTER)=A*X(INTER)+B=
*  C*X(INTER)+D

```

```

0000R      ENTRY CFIX,DELTA3,CORTRN,XIT
0000R      EXTRN CORNCT,SLOPER
0000R      EXTRN INOUTS
0000R      EXTRN DATP,THOLD
0000R      EXTRN CORNER,STABS
0000R      EXTRN ATANUR

```

```

0000      STORE EQU 0
0002      STORE2 EQU 2
000E      STOREE EQU 14
000C      STOREC EQU 12

```

```

0000R 4000      CFIX      STH 0,RETRN
0238R
0004R 2442      LIS 4,2      SET PTR TO GET
0006R 41F      BAL 15,BD      SET B
0008R
000AR 4000      STH 0,B

```

030ER		
000ER 2448	LIS 4,8	LET PTR TO GET
0010R 41F	BA 15,BD	GET D
0008ER		
0014R 4000	STH 0,D	
00310R		
0018R C900	CHI 0,X'9000'	IF D<900 I
9000		
001CR 4210	BM DXY	LINE2 IS VERTIC
000C6R		
0020R C900	CHI 0,X'7000'	IF D>700 I
7000		
0024R 4220	BP DXY	LINE2 IS VERTIC
000C6R		
0028R 4810	LH 1,B	
0030ER		
002CR C910	CHI 1,X'9000'	IF B<900 I
9000		
0030R 4210	BM DXY	LINE1 IS VERTIC
00164R		
0034R C910	CHI 1,X'7000'	IF B>700 I
7000		
0038R 4220	BP DXY	LINE1 IS VERTIC
00164R		
003CR 0B10	SHR 1,0	R1=B-D
003ER 4240	BTC 4,BOT	IF SUB. FAILS,S
000AR		
0042R 4010	MH 1,TANFAC	ADD IN ROUNDOFF
0031CR		
0046R 244A	LIS 4,10	
0048R 4854	LH 5,CORNER(4)	IS HAS C*TANFAC
0000F		
004CR 2746	SIS 4,6	
004ER 4B54	SH 5,CORNER(4)	R5=((A)*TANFAC
004AR		
0052R 0820	LHR 2,0	
0054R 0831	LHR 3,1	
0056R 0D05	LHR 0,5	IF HAS X AT COR
		= (D-B)*TANFAC/((A-C)*TANFA
0058R 0531	CLHR 3,1	
005AR 4230	BNE CHECK	CHECK FOR DIV F
0019AR		
005ER 0520	CLHR 2,0	
0060R 4230	BNE CHECK	IF NONE GO TO C
0019AR		
0064R 0811	LHR 1,1	
0066R 4330	BZ CHECK	
0019AR		
006AR C550	CLHI 5,1	
00001		
006ER 4330	BE CHECK	
0019AR		

0072R 4810	LH	1,E	OTHERWISE ONE L
030ER			
0076R 41B0	BAL	15,STABS	DECIDE WHICH
0000F			
007AR 0801	LHR	0,1	
007CR 4810	LH	1,D	
0310R			
0080R 41F0	BAL	15,STABS	AND GO TO APPRO
0078R			
0084R 0510	CLHR	1,0	
0086R 43B0	BNL	EXY	
00C6R			
008AR 4300	B	EXY	
0164R			

* *****

*

*ENTRY BAL 15,BD
 * THIS ROUTINE EXTRACTS B FROM THE
 * EQUATION $Y=AX+B$
 * $B=Y-AX$
 *

008ER 4814	BD	LH	1,CORNER(4)	R4=PTR TO 2ND L
0050R				
0092R 4831		LH	3,STORE(1)	R3=X
0000				
0096R 2642	AIS	4,2		R4 HAS PTR TO
0098R 41C0	BAL	12,SETTER		IF RETURNED
00A4R				

* R3 HAS AX
 * ELSE X INTERCEPT NEAR INFIN
 *

009CR 4801	LH	0,STORE2(1)	R0 HAS Y
0002			
00A0R 0B03	SHR	0,3	R0=B
00A2R 030F	BR	15	

*

*ENTRY BAL 12,SETTER
 * R3 HAS X ON ENTRY
 * R4=PTR TO SLOPE OF LINE (A)
 * $R2/3 = AX$ ON EXIT
 *

00A4R 4C24	SETTER	MH	2,CORNER(4)
0090R			
00A8R 0852	LHR	5,2	
00AAR 0863	LHR	6,3	
00ACE 4230	BNZ	HOP	
00B4R			

00B0R 0822	LHR	2,2	
00B2R 033C	BFCR	3,12	
00B4R 4D20	HOP DH	,TANFAC	REMOVE ROUND OFF
031CR			
00B8R 0536	CLHR	3,6	
00BAR 023C	BTCR	3,12	CHECK FOR DIVID
00BCR 0525	CLHR	2,5	IF NONE RETURN
00BER 023C	BTCR	3,12	
00C0R C800	LHI	0,X'8000'	ELSE LOAD RO=B=
8000			
00C4R 03)F	BR	15	RETURN TO CALLE

★

★ *****

★ ENTRY F (B)XY
★ BXY & DXY ARE IDENTICAL EXCEPT
★ YOU USE BXY WHEN LINE2 IS VERTICAL
★ USE DXY WHEN LINE1 IS VERTICAL.

*WITH ONE LINE VERTICAL, X ALONG THAT
*LINE WILL NOT CHANGE MUCH. IF YOU
*TAKE AN X NEAR THE INTERSECTION, IT CAN
*BE SAFELY ASSUMED TO BE THE X AT THE
*INTERSECTION. SUBSTITUTING THIS X IN
*THE EQUATION FOR THE OTHER LINE GIVES
*Y(INTER)

★

00C6R 4810	BXY	LH	1,B	
030ER				
00CAR C910		CHI	1,X'9000'	SEE IF BOTH LIN
9000				
00CER 4210		BM	BOTH	
00DAR				
00D2R C910		CHI	1,X'7000'	
7000				
00D6R 4320		BNP	NOT	IF NOT GO TO N
014AR				
00DAR 48F0	BOTH	LH	15,RETRN	IF FROM INTERS
0238R				
00DER 480F		LH	0,0(15)	
0000				
00E2R C500		CLHI	0,INOUTS	
0000F				
00E6R 0330		BER		RETURN
00E8R 2442		LIS	4,2	
00EAR 2438		LIS	3,8	ELSE YOU HAVE A
00ECR 0824		LHR	2,4	

000EER 4813	LOOP	LH	1,CORNER(3)	FORGET IT &
00A6R				
000F2R 4014		STH	1,CORNER(4)	GO LOOK FOR NEX
00F0R				
000F6R 4813		LH	1,IA1P(3)	
0000F				
000FAR 4014		STH	1,IA1P(4)	
00F8R				
000FER 0A32		AHR	3,2	
0100R 0A42		AHR	4,2	
0102R C54		CLHI	4,8	
000B				
0106R 4280		BL	LOOP	
00EER				
010AR 4840		LH	4,CORNCT	
0000F				
010ER 2746		SIS	4,6	
0110R 4040		STH	4,CORNCT	
0b0CR				
0114R 41F		BAL	15,SLOPER	
0000F				
0118R C830		LHI	3,THOLD	
0000F				
011CR 2634		AIS	3,4	
011ER 4843		LH	4,STOREC(3)	
000C				
0122R 4043		STH	4,STORE(3)	
0000				
0126R 4843		LH	4,STOREE(3)	
000E				
012AR 4043		STH	4,STORE2(3)	
0002				
012ER 2638		AIS	3,8	
0130R 4843		LH	4,STOREC(3)	
000C				
0134R 4043		STH	4,STORE(3)	
0000				
0138R 4843		LH	4,STOREE(3)	
000E				
013CR 4043		STH	1,STORE2(3)	
0002				
0140R 48F0		LH	15,RETRN	
0238R				
0144R 480F		LH	0,0(15)	
0000				
0148R 0300		BR		
014AR 2446	NOT	LIS	4,6	GET END OF LINE
014CR 4824		LH	2,CORNER(4)	NEAREST INTERSE
00F4R				
0150R 4812		LH	1,STORE(2)	GET ITS X COORD
0000				
0154R 41F0		BAL	15,BY	CALCULATE Y

0158R	017ER 4030 030AR	STH	3,Y1	
015CR	4030 030CR	STH	3,Y2	STORE IT FOR L1
0160R	4300 01AAR	B	CHEKR	SKIP CALCULATIO

0164R	2442	DXY	LIS	4,2
0166R	4824		LH	2,CORNER(4)
014ER				
016AR	4812		LH	1,STORE(2)
	0000			
016ER	41F0		BAL	15,DY
018CR				
0172R	4030		STH	3,Y1
030AR				
0176R	4030		STH	3,Y2
030CR				
017AR	4300		B	CHEKR
01AAR				

*
 *ENTRY BAL 15,(B)Y
 * BY & DY ARE IDENTICAL; THEY ARE
 * USED TO CALCULATE Y WHEN X IS
 * KNOWN. BY IS USED FOR LINE1; DY
 * FOR LINE2.
 *ON ENTRY, R1 HAS X-COORD AT THE
 * INTERSECTION
 *ON RETURN, R3 HAS Y
 *

017ER	0831	BY	LHR	3,1	
0180R	2444		LIS	4,4	
0182R	41C0		BAL	12,SETTER	GET AX
00A4R					
0186R	4A30		AH	3,B	=AX+
030ER					
018AR	030F		BR	15	

018CR	0831	DY	LHR	3,1
018ER	244A		LIS	4,10
0190R	410		BAL	12,SETTER
00A4R				
0194R	4A3		AH	3,I
0310R				
0198R	030F		BR	15

★ ★★★★★★★★★★

```

★
019AR 41F01 CHECK BAL 15,BY GET Y AT INTERS
017ER
019ER 40301 STH 3,Y1 FOR LINE1
030AR
01A2R 41F01 BAL 15,DY GET INTERSECTIO
018CR
01A6R 40301 STH 3,Y2 FOR LINE2
030CR

```

★ YOU'VE NOW GOT TWO POSSIBLE INTERSECTIO
★ X,Y1 & X,Y2. YOU WOULD LIKE TO FIND
★ FIND THE ONE WHICH IS CLOSEST
★ TO THE INTENDED CORNER
★ FIND THE LINE ALONG WHICH THE DIFFERENCE
★ IN LINE SLOPE

★★

★ IS GREATER (BETWEEN X,Y1 & X,Y2) THEN FIND
★ THE X,Y COORD WHICH GIVES THE LINE THE SLOP
★ CLOSER TO THE ONE ORIGINALLY CALCULATED
★

```

01AAR 40101 CHEKR STH 1,H1
0218R
01AER 0B66 SHR 6,6
01BOR 0B22 SHR 2,2
01B2R 41F01 BAL 14,SLOPED GET DIF BETWEEN
023ER
★ FOR LINES X,Y1-CORNER(0
★ X,Y2-CORNER(00)
01B6R 40101 STH 1,DEL1 STORE THE DIFFE
031AR
01BAR 2426 LIS 2,6
01BCR 2464 LIS 6,4
01BER 41F01 BAL 14,SLOPED SAME FOR CORNER
023ER
01C2R 45101 CLH 1,DEL1 FIND WHICH LINE
031AR
01C6R 4380 BNL SCND
01E8R
01CAR 0B66 SHR 6,6
01CCR 2422 LIS 2,2
01CER 4812 LH 1,CORNER(2)
0168R

```

01D2R 0722		XHR	2,2	
01D4R 4300		B	RET1	
01DER				
01D8R 2426	SCND	LIS	2,6	
01DAR 4812		LH	1,CORNER(2)	
01DOR				
01DER 4851	RET1	LH	5,STORE2(1)	
0002				
01E2R 4831		LH	3,STORE(1)	
0000				
01E6R 41B0		BAL	15,DIFFER	
0288R				
01EAR 4816		LH	1,HOLD(6)	
0312R				
01EER 0B15		SHR	1,5	GET DIF BETW CA
	*			SLOPE FOR X,Y1 & SLOPE OF
	*			LINE SEGMENT NEAREST
01FOR 41D0		BAL	13,PITEST	
026AR				
01F4R 4010		STH	1,DELL	
031AR				
01F8R 4816		LH	1,HOLD1(6)	
0314R				
01FCR 0B15		SHR	1,5	AME FOR X,Y2
01FER 41D0		EAL	13,PITEST	
026AR				
0202R 45B0		CLH	1,DELL	FIND WHICH IS L
031AR				
0206R 42B0		BL	SEC	
0212R				
020AR 4830		LH	3,Y1	LOAD R3 WITH NE
030AR				
020ER 43B0		B	RET2	
0216R				
0212R 4830	SEC	LH	3,Y2	
030ER				
0216R C8B0	RET2	LHI	7,0	
0000				
0218R	H1	EQL	*-2	
021AR 2416	XIT	LIS	1,6	
021CR 4841		LH	4,CORNER(1)	GET POINTER TO
01DCR				
0220R 4074		STH	7,STORE(4)	
0000				
0224R 4034		STF	3,STORE2(4)	
0002				
0228R 2714		SIS	1,4	
022AR 4841		LH	4,CORNER(1)	
021ER				
022ER 4074		STH	7,STORE(4)	
0000				
0232R 4034		STH	3,STORE2(4)	

0236R	0002		
	C800	LHI	10,10
	0000		
0238R		RETRN	EQU *-2
0238R		CFRTRN	EQU *-2
023AR	2602		AIS 10,2
023CR	0300		BR 7

*
 *ENTRY BAL 14,SLOPED
 * R15--BAL
 * R14--RETURN ADDR
 * R2--CORNER PTR
 * R1--STABS(SLOPE DIF) ON RETURN
 * R6--PTR TC HOLE
 * R3,R5--NRKC

023ER	4850	SLOPED	LH	5,Y1	
	030AR				
0242R	4830		LH	3,H1	41 HAS X COORD
	0218R				
0246R	41F0		BAL	15,DIFFER	
	0288R				
024AR	4056		STH	5,HOLD(6)	R5=SLOPE (X,Y1-
	0312R				
024ER	4830		LH	3,H1	
	0218R				
0252R	4850		LH	5,Y2	
	030CR				
0256R	41F0		BAL	15,DIFFER	
	0288R				
025AR	4056		STH	5,HOLD(6)	R5=SLOPE (X,Y2-
	0314R				
025ER	4816		LH	1,HOLD(6)	
	0312R				
0262R	0B15		SHR	1,5	
0264R	41D0		EAL	13,PITEST	
	026AR				
0268R	030E	EXIT	BR	14	
026AR	41F0	PITEST	BAL	15,STABS	
	0082R				
026ER	C510		CLHI	1,X'3257'	
	3257				
0270R		PI	EQU	*-2	
0272R	028D		BTCR	8,13	
0274R	4230		BNE	NOTBAC	
	027CR				

0278R	0B11	SHR	1,1
027AR	030D	BR	13
027CR	C820	NOTBAC LHI	2,X'64AE'
	64AE		
027ER		TWOPI EQU	*-2
0280R	0B12	SHR	1,2
0282R	41B0	BAL	15,STABS
	026CR		
0286R	030D	BR	13
0288R	40B0	DIFFER STH	15,HCLDF
	0306R		
028CR	4812	LH	1,CORNER(2)
	022CR		
0290R	0875	LHR	7,5
0292R	0843	LHR	4,3
0294R	4B51	SH	5,STORE2(1)
	0002		
0298R	4B31	SH	3,STORE(1)
	0000		
029CR	2622	AIS	2,2
029ER	4812	LH	1,CORNER(2)
	028ER		
02A2R	2722	SIS	2,2
02A4R	4B71	SH	7,STORE2(1)
	0002		
02A8R	4B41	SH	4,STORE(1)
	0000		
02ACR	0817	LHR	1,7
02AER	41F	BAL	15,STABS
	0284R		
02B2R	4010	STH	1,TEMPH
	0216R		
02B6R	0814	LHR	1,4
02B8R	41B0	BAL	15,STABS
	02B0R		
02BCR	61B0	AHM	1,TEMPH
	02D6R		
02C0R	0815	LHR	1,5
02C2R	41F	BAL	15,STABS
	02EAR		
02C6R	4010	STH	1,TEMPH1
	02D2R		
02CAR	0813	LHR	1,3
02CCR	41B0	BAL	15,STABS
	02C4R		
02D0R	CA00	PHI	1,0
	0000		
02D2R		TEMPH1 EQU	*-2
02D4R	C510	CLHI	1,0
	0000		
02D6R		TEMPH EQU	*-2

02D8R	4380		BNL	OK
	02E0R			
02DCR	0857		LHR	5,7
02DER	0834		LHR	3,4
02E0R	4030	OK	STH	5, DELTAX
	0320R			
02E4R	4050		STH	5, DELTAY
	031ER			
02E8R	41F0		BAL	15, ATANUR
	0000F			
02ECR	031ER		DC	A(DELTAY), A(DELTAX), A(ANGLE)
	0320R			
	0322R			
02F2R	4850		LH	5, ANGLE
	0322R			
02F6R	4830		LH	3, ANGLE2
	0324R			
02FAR	CF50		SLHA	5, 12
	000C			
02FER	CE30		SRHA	3, 3
	0003			
0302R	0A53		AHR	5, 3
0304R	4300		B	*
	0304R			
0306R		HOLDF	EQU	*-2

0308R		DELTA3	IS	2
030AR		Y1	DS	2
030CR		Y2	IS	2
030ER		B	DS	2
0310R		D	IS	2
0312R		HOLD	DS	2
0314R		HOLD1	IS	2
0316R		HOLD2	DS	2
0318R		HOLD3	IS	2
031AR		DEL1	DS	2
031CR	0040	TANFAC	IC	X' 40'
031ER		DELTAY	DS	2
0320R		DELTAX	IS	2
0322R		ANGLE	DS	2
0324R		ANGLE2	IS	2
0326R			END	

	ANGLE	0322R
	ANGLE2	0324R
*	ATANUR	02EAR
	B	030ER
	BD	0008ER
	BOTH	000AR
	EXY	0006R
	BY	017ER
*	CFIX	0000F

* CFRTN	0238R
CHECK	019AR
CHEK	01AAR
* CORNCT	0112R
* CORNER	02A0R
D	0310R
* DATP	00FCR
DEL1	031AR
* DELTA3	0308R
DELTAX	0320R
DELTAY	031ER
DIFFER	0288R
EXY	0164R
DY	018CR
EXIT	0268R
H1	0218R
HOLD	0312R
HOLD1	0314R
HOLD2	0316R
HOLD3	0318R
HOLDF	0306R
HOP	00B4R
* INOUTS	00E4R
LOOP	00EER
NOT	014AR
NOTBAC	027CR
OK	02E0R
PI	0270R
PITEST	026AR
RET1	01DER
RET2	0216R
RETRN	0238R
SCND	01D8R
SEC	0212R
SETTER	00A4R
SLOPED	023ER
* SLOPER	0116R
* STABS	02CER
STORE	0000
STORE2	0002
STOREC	0000C
STOREE	000E
TANFAC	031CR
TEMPH	02D6R
TEMPH1	02D2R
* THOLD	011AR
TWOPI	027ER
* XIT	021AR
Y1	030AR
Y2	030CR

```

0000R      ENTRY DELT11,DELT22
0000R      ENTRY LINER,DELT11,DELTA2
0000R      EXTRN CORNER,GPOINT
0000R      EXTRN PRLOOP,WNWONE
0000R      EXTRN LAID,PPFIDF,PAID
0000R      EXTRN DATP
0000R      ENTRY FLIN1,FLINE2
0000R      EXTRN PTMAX

```

```

0000      STORE EQU 0
0006      STORE6 EQU 6

```

```

*REGS: 0 RETURN,BAL
*      1
*      2 CONTAINS CORNCT
*      3
*      4
*      5
*      6 CONTAINS RMAX
*      7
*      8
*      9
*     10 PTR TO LINE END FROM PLOOP
*     12 MULTIPLY,DIVIDE--
*     13 DATA CHANGES
*     15 BAL

```

```

0000R 4000      LINER  STH  0,RETRN
0000R 00C0R
0004R 4020      STH  2,HOLD
0000R 00E8R
0008R 4090      STH  9,HOLD2
0000R 00BCR
000CR 4060      STH  5,HRATE
0000R 0148R
0010R 4810      LH    1,DELT11
0000R 0140R
0014R 4010      STH  1,DELT11
0000R 013CR
0018R 4810      LH    1,DELT22
0000R 0142R
001CR 4010      STH  1,DELTA2
0000R 013ER
0020R 4812      LH    1,DATP(2)
0000R 0000F
0024R 4210      BM    GETPT
0000R 0030R
0028R 4810      LH    1,HOLDPT
0000R 015ER
002CR 4012      STH  1,CORNER(2)
0000R 0000F
0030R 41F0      GETPT  BAL  15,PL00P2

```

0034R	00CAR 40A0 0146R	STH	10,FLINE2
0038R	2622	ARE	2,2
003AR	41F0 00CAR	BAL	15,PL00P2
003ER	40A0 015ER	STH	1, HOLDPT
0042R	40A0 0144R	STH	10,FLINE
0046R	41F0 0000F	BAL	15,LADD
004AR	000A	DC	10
004CR	0000F	IC	PTMAX
004ER	0144R	DC	FLINE
0050R	0146R	IC	FLINE+2
0052R	014AR	DC	SIZE
0054R	08EE	LHR	14,14
0056R	4230 0000F	BNZ	PRLOOP
005AR	41F0 0000F	BAL	15,PPFLDF
005ER	000E	DC	14
0060R	004CR	IC	PTMAX
0062R	0144R	DC	FLINE
0064R	0146R	IC	FLINE+2
0066R	015CR	DC	FIELD1
0068R	015AR	DC	TWC
006AR	0148R	DC	HRATE
006CR	08EE	LHR	14,14
006ER	4230 0058R	BNZ	PRLOOP
0072R	41F0 0000F	BAL	15,GPOINT
0076R	000A	DC	10
0078R	0060R	IC	PTMAX
007AR	0144R	DC	FLINE
007CR	0158R	IC	CNE
007ER	00C2R	DC	K1
0080R	41F0 0074R	BAL	15,GPOINT
0084R	000A	IC	10
0086R	0078R	DC	PTMAX
0088R	0144R	DC	FLINE
008AR	015AR	DC	TWO
008CR	00C4R	IC	Y1
008ER	41F0 0082R	BAL	15,GPOINT
0092R	000A	DC	10
0094R	0086R	IC	PTMAX
0096R	0146R	DC	FLINE+2
0098R	0158R	IC	CNE

009AR	00C6R		EC	X2
009CR	41F		BAL	15, GPOINT
	0090R			
00A0R	001A		DC	10
00A2R	0094R		EC	PTMAX
00A4R	0146R		DC	FLINE+2
00A6R	015AR		EC	TWO
00A8R	00C8R		DC	Y2
00AAR	C89		LHI	9, X1
	00C2R			
00AER	C80		LHI	12, X2
	00C6R			
00B2R	4D0		BAL	10, WNWONE
	0000F			
00B6R	C820	RETRY2	LHI	2, 0
	0000			
00B8R		HOLD	EQU	*-2
00BAR	C890		LHI	9, HOLD2
	00BCR			
00BCR		HOLD2	EQU	*-2
00BER	430		B	*
	00BER			
00COR		RETRN	EQU	*-2
		*		
00C2R		X1	ES	2
00C4R		Y1	DS	2
00C6R		X2	ES	2
00C8R		Y2	DS	2
		*		
00CAR	D000	PL00P2	STM	12, SAV4
	0134R			
00CER	4060		STH	6, SAV6+2
	0130R			
00D2R	48A2		LH	10, CORNER(2)
	002ER			
00D6R	4860		LH	6, PTMAX
	00A2R			
00DAR	45A6		CLH	10, 6(6)
	0006			
00DER	4280		BL	KNOWN
	012AR			
00E2R	4330		BE	KNOWN
	012AR			
00E6R	40A0		STH	10, X
	00F8R			
00EAR	26A2		AIS	10, 2
00ECR	40A0		STH	10, Y
	00FAR			
00FOR	41F0		BAL	15, PADD
	0000F			
00F4R	0008		DC	8

00F6R	00D8R		DC	PTMAX
00F8R	0000	X	IC	
00FAR	0000	Y	DC	
00FCR	008AE		LHR	10,14
00FER	40E0		STH	14,VPN
	0150R			
0102R	41F0		BAL	15,LADD
	0048R			
0106R	000A		DC	10
0108R	00F6R		DC	PTMAX
010AR	0150R		DC	VPN
010CR	014ER		DC	ZERO
010ER	0156R		DC	LSIZE
0110R	4812		LH	13,EATP(2)
	0022R			
0114R	40D0		STH	13,PLACE
	0152R			
0118R	41B0		BAL	15,PPFLD
	005CR			
011CR	000E		IC	14
011ER	0108R		DC	PTMAX
0120R	0190R		DC	VPN
0122R	014ER		DC	ZERO
0124R	0154R		DC	FIELD2
0126R	014CR		DC	FOUR
0128R	0152R		DC	PLACE
012AR	D1C0	KNOWN	LM	12,SAV4
	0134R			
012ER	C860	SAV6	LHI	6,0
	0000			
0132R	030F		BR	15

*

0134R	SAV4	DS	8
013CR	DELTA1	DS	2
013ER	DELTA2	IS	2
0140R	DELT11	DS	2
0142R	DELT22	IS	2
0144R	FLINE	DS	2
0146R	FLINE2	IS	2
0148R	0000	HRATE	DC 0
014AR	0000C	SIZE	EC 12
014CR	00 14	FOUR	DC 4
014ER	0000	ZERO	IC 10
0150R	0000	VPN	DC 1
0152R	0000	PLACE	EC 1
0154R	0009	FIELD2	DC 9
0156R	0000C	LSIZE	EC 12
0158R	0001	ONE	DC 1
015AR	0002	TWO	EC 2
015CR	0007	FIELD1	DC 7
015ER	0000	HOLDPT	EC 11

0160R

END

NO ERRORS

* CORNER 00D4R
* DATF 0112R
* DELT11 0140R
* DELT22 0142R
* DELTA1 013CR
* DELTA2 013ER
FIELD1 015CR
FIELD2 0154R
* FLINE 0144R
* FLINE2 0146R
FOUR 014CR
GETPT 0030R
* GPOINT 009ER
HOLD 00B8R
HOLD2 00BCR
HOLDPT 015ER
HRATE 0148R
KNOWN 012AR
* LADD 0104R
* LINER 0000R
LSIZE 0156R
ONE 0158R
* PADD 00F2R
PLACE 0152R
PLOOP2 00CAR
* PPFLDF 011AR
* PRLOOP 0070R
* PTMAX 011ER
RETRN 00COR
RETRY2 00B6R
SAV4 0134R
SAV6 012ER
SIZE 014AR
STORE 0000
STORE6 0006
TWO 015AR
VPN 0150R
* WNWONE 00B4R
X 00F8R
X1 00C2R
X2 00C6R
Y 00FA
Y1 00C4R
Y2 00C8R
ZERO 014ER

004CR 4280		BL	NEXT	PT IS EXTREME I
00EAR				
0050R 4570		CLH	7,GREAT1	4B6>+DIFF>0192
013ER				
0054R 4280		BL	EXTREM	
0070R				
0058R 4300		B	NEXT	ELSE LOOK AT NE
00EAR				
005CR 4570	LOWLM	CLH	7,SMALL	PT IS EXTREME I
013CR				
0060R 4380		BNL	NEXT	FE6E>-DIFF>FB40
00EAR				
0064R 4570		CLH	7,SMALL1	
0140R				
0068R 4380		BNL	EXTREM	
0070R				
006CR 4300		B	NEXT	ELSE LOOK AT NE
00EAR				
0070R 48B0	EXTREM	LH	11,BSTORE	
0182R				
0074R CB80		SHI	11,2	
0002				
0078R 4310		BNM	DTEST	
0084R				
007CR CAB0		AHI	11,2	
0002				
0080R 4300		B	GOES	
00E6R				
0084R CAD0	DTEST	AHI	13,INCREM	
0006				
0088R 48FD		LH	15,STORE(13)	
002AR				
008CR 487D		LH	7,STORE2(13)	
0038R				
0090R 4BFB		SH	15,STOREX(11)	
0154R				
0094R 4B7B		SH	7,STOREY(11)	
0166R				
0098R 0CEF		MHR	14,15	
009AR 0C67		MHR	6,7	
009CR CBD0		SHI	13,INCREM	
0006				
00A0R CAB0		AHI	11,2	
0002				
00A4R 0AF7		AHR	15,7	
00A6R 0EE6		ACHR	14,6	
00A8R 08EE		LHR	14,14	
00AAR 4230		BNZ	GOES	
00B6R				
00AER 45D0		CLH	15,MIN	SEE IF EXTREMES
0138R				
00B2R 4280		EL	NEXT	

000EAR			
000B6R 4870	GOES	LH	7,DISKST
0000F			
000BAR CD70		SLHL	7,7
0007			
000BER 08FD		LHR	15,13
000C0R 4BF0		SH	15,OFFSET
0000F			
000C4R 067F		OHR	7,15
000C6R 407B		STH	7,INDEX(11)
0142R			
000CAR 48FD		LH	15,STORE(13)
0008AR			
000CER 40FE		STH	15,STOREX(11)
0154R			
000D2R 48FD		LH	15,STORE2(13)
0008ER			
000D6R 40FB		STH	15,STOREY(11)
0166R			
000DAR CAB0		AHI	11,2
0002			
000DER 40B0		STH	11,BSTORE
0182R			
000E2R 4090		STH	5,SLOP3
0186R			
000E6R 4300		B	INCR
00F2R			
000EAR 48F0	NEXT	LH	15,SLOP2
0184R			
000EER 40B0		STH	15,SLOP3
0186R			
000F2R 4090	INCR	STH	5,SLOP2
0184R			
000F6R	BACK	EQU	*
000F6R C850	SAV5	LHI	5,0
00000			
000FAR C850	SAV6	LHI	6,0
00000			
000FER C870	SAV7	LHI	7,0
00000			
0102R C830	SAV11	LHI	11,0
00000			
0106R C8D0	SAV13	LHI	13,0
00000			
010AR C8E0	SAV14	LHI	14,0
00000			
010ER 4300	SAV15	B	*
010ER			

0006	INCREM	EQU	6	
0000R		EXTRN	STORE, STORE2, ATANUR	
0000R		ENTRY	TFG	
0000R		ENTRY	EXTRMR	
0000R		ENTRY	BSTORE, SLOP2, SLOP3, GREAT, GREAT1	
0000R		ENTRY	SMALL, SMALL1, TAN1, STOREX, STOREY	
0000R		ENTRY	AIN, INDEX	
0000R		EXTRN	STORE, STORE2	
0000R		EXTRN	OFFSET, DISKST	
*				
* EXTRMR EXPECTS PIR TO CURRENT POINT IN R13				
0000R	4050	EXTRMR	STH	5, SAV5+2
	00F8R			
0004R	4060		STH	6, SAV6+2
	00FCR			
0008R	4070		STH	7, SAV7+2
	0100R			
000CR	40B0		STH	11, SAV11+2
	0104R			
0010R	40D0		STH	13, SAV13+2
	0108R			
0014R	40E0		STH	14, SAV14+2
	010CR			
0018R	40F0		STH	15, SAV15+2
	0110R			
001CR	485D		LH	5, STORE(13) R5=X2
	0000F			
0020R	486D		LH	6, STORE2(13) R6=Y2
	0000F			
0024R	CBD0	GO	SHI	13, INCREM
	0006			
0028R	487D		LH	7, STORE(13)
	001ER			
002CR	C570		CLHI	7, X*8000
	8000			
0030R	4330		BE	GO
	0024R			
0034R	0B57		SHR	5, 7 X2-X1
0036R	4B6D		SH	6, STORE2(13) Y2-Y1
	0022R			
003AR	41E1		BAL	14, TAN1 CALL ARCTAN SUB
	0112R			
003ER	4870		LH	7, SLOP3
	0186R			
0042R	0B75		SHR	7, 5 BECAUSE RE=0 IN
0044R	4210		BM	LOWLM -90<1ST SLOP<+9
	005CR			
0048R	4570		CLF	7, GREAT NOT STORED AS 1
	013AR			

0112R	4050	TAN1	STH	5,DX	
	017AR				
0116R	4060		STH	5,DY	
	0178R				
011AR	41F0		BAL	15,ATANUR	CALL FLOATING P
	0000F				
011ER	0178R		DC	A(DY),A(DX),3(RESULT)	
	017AR				
	017CR				
0124R	4890		LH	5,RESULT	TO EXPRESS FLOA
	017CR				
0128R	CF90		SLFA	5,8	IN ONE HALF WOR
	0008				
012CR	4860		LH	6,FRACTN	
	017ER				
0130R	CE60		SRHA	6,7	SO THAT THE 3RD
	0007				
0134R	0A56		AHR	5,6	AS 1ST SIGNIFIC
0136R	030E		BR	14	
0138R	0211	MIN	DC	X'211'	
013AR	0192	GREAT	DC	X'0192'	PI/2 OR 90 DEGR
013CR	FE6E	SMALL	DC	X'FE6E'	-PI/2 OR -90 DE
013ER	04B6	GREAT1	DC	X'04B6'	3PI/2 OR 270 DE
0140R	FB40	SMALL1	DC	X'FB40'	-3PI/2 OR -270
0142R		INDEX	DS	18	
0154R		STCREX	IS	18	
0166R		STOREY	DS	18	
0178R		DY	IS	2	
017AR		DX	DS	2	
017CR		RESULT	IS	2	
017ER		FRACTN	DS	2	
0180R		TAG	IS	2	
0182R		BSTORE	DS	2	
0184R		SLOP2	IS	2	
0186R		SLOP3	DS	2	
0188R			END		

NO ERRORS

*	ATANUR	011CR
	BACK	00F6R
*	BSTORE	0182R
*	DISKST	00B8F
	DTEST	0084R
	IX	017AF
	DY	0178R
	EXTREM	0070R
*	EXTRMR	0000R
	FRACTN	017ER
	GO	0024R
	GOES	00B6F
*	GREAT	013AR
*	GREAT1	013ER

INCR	00F2F
INCREM	0006
* INDEX	0142F
LOWLM	005CR
* MIN	0138F
NEXT	00EAR
* OFFSET	00C2F
RESULT	017CR
SAV11	0102F
SAV13	0106R
SAV14	010AF
SAV15	010ER
SAV5	00F6F
SAV6	00FAR
SAV7	00FEF
* SLOP2	0184R
* SLOP3	0186F
* SMALL	013CR
* SMALL1	0140F
* STORE	00CCR
* STORE2	00D4F
* STOREX	0154R
* STOREY	0166F
* TAG	0180R
* TAN1	0112F
