GRAPHICAL INPUT THROUGH MACHINE RECOGNITION OF SKETCHES

Christopher F. Herot
Architecture Machine Group, Department of Architecture
Massachusetts Institute of Technology, Cambridge, Massachusetts

A family of programs has been developed to allow graphical input through continuous digitizing. Drawing data, sampled at a high and constant rate, is compressed and mapped into lines and splines, in two and three dimensions. This is achieved by inferring a particular user's intentions from measures of speed and pressure.

Recent experiments have shown that even the most basic inference making cannot rely solely upon knowledge of the user's drawing style, but needs additional knowledge of the subject being drawn, the protocols of its domain, and the stage of development of the user's design. This requirement implies a higher level of machine intelligence than currently exists. An alternate approach is to increase the user's involvement in the recognition process.

Contrary to previous efforts to move from sketch to mechanical drawing without human intervention, this paper reports on an interactive system for graphical input in which the user overtly partakes in training the machine and massaging the data at all levels of interpretation. The initial routines for data compression employ parallel functions for extracting such features as bentness, straightness, and endness. These are planned for implementation in microprocessors.

Results offer a system for rapid (and enjoyable) graphical input with real-time interpretation, the beginnings of an intelligent tablet.

1. INTRODUCTION

There are many areas of human endeavor which could benefit from the use of computer aids if there existed an effective means of communicating about these tasks with a machine. While the field of computer graphics arose to fill that need, it has too often added a new level of complexity. In computer-aided design, for instance, the process of "digitizing" is sufficiently cumbersome to delay its application until a relatively complete design has been produced by the human designer. The result is usually more akin to computer-aided evaluation or manipulation than to computer-aided design. The research described here is motivated by the desire to involve the computer in the early stages of the design process, where the feedback generated by the machine can be most useful. The medium chosen is free-hand sketching, as done with pencil and paper, as could be done at a data tablet. A machine is postulated to be looking on while the user is sketching. It could make inferences not only about the meaning of the sketch but also about the user's attitudes toward, and uncertainties about, his design.

This approach offers its own unique set of problems and solutions, since the data available to the machine are at once plentiful and incomplete. While the ultimate implementation assumes a near-human intelligence on the part of the machine, far off in time, there are many interesting things to be learned along the way.

Our previous experiments[1, 2, 3, 4] have been directed toward the creation of a "passive" input system which would make fairly complete inferences about a sketch while requiring a minimum of intervention from the user. The programs we used in these projects involved many levels of interpretation. From the 100 pen positions sampled each second, the machine would have to find lines, curves, and corners, building a description of a two-dimensional object. This description could be interpreted further, possibly as a three-dimensional description.

The following sections depict three experiments in computer processing of sketches. HUNCH, described in the next section, was directed toward answering the following question: Does there exist a syntax of sketching, something which could be processed independently of the embedding semantics? Could a machine make useful interpretations of a sketch without employing a knowledge of the subject domain? The mixed results of that experiment led to the investigation described in section 3, a rather ambitious effort to make use of architectural knowledge in recognizing a sketch. Finally, section 4 reports on current work which places more emphasis on user involvement in the input process.

The work reported herein has been sponsored by the Office of Naval Research, grant number N00014-67-A-0204-0074 and by the National Science Foundation, Division of Computer Research, grant number DCR74-20974-A01.

2. THE HUNCH SYSTEM

HUNCH is a name given to a set of FORTRAN programs which were designed to process freehand sketches drawn with a data tablet or light pen. Each program performs a different level of interpretation, storing its output in a file where it can be used as input by the other programs. Facilities exist to display and manipulate various stages of interpretation. The relationships among the programs are illustrated below.

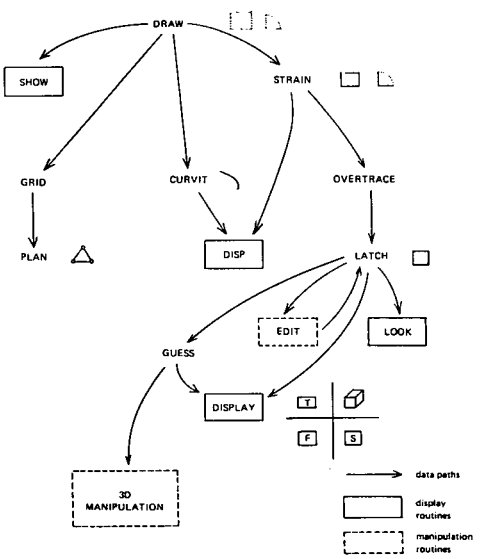

Fig. 1 - The HUNCH System

The system currently runs under the MAGIC[5] operating system on a 64K byte Interdata minicomputer. The MAGIC system allows a program to use any graphical input or output device in a device-independent manner, including a refresh display with light pen, a storage tube, a raster scan display, a large digitizer-plotter, and several varieties of data tablets.

Raw Data

The user of the system typically sits down at a table and draws with a ballpoint pen on a piece of paper taped to the surface of a data tablet. Considerable effort was expended to provide an environment as close to sketching as possible. One of the tablets provided measures three feet by

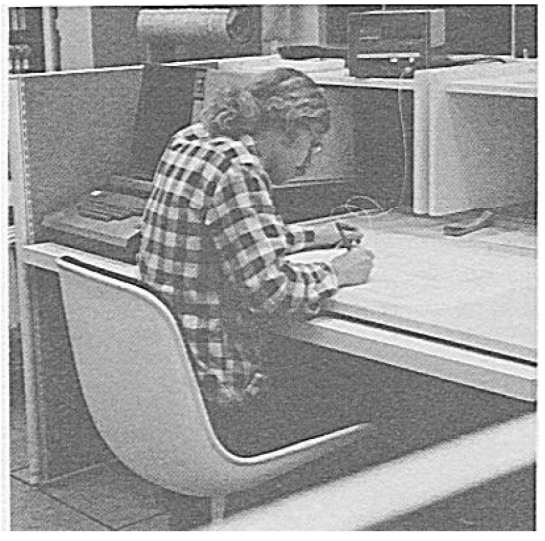

Fig. 2 – Large Data Tablet

four feet so as to allow large drawings. A special pencil was constructed so that the user could work with the more customary graphite rather than a ballpoint pen.[6] The pencil was designed with the additional intention of

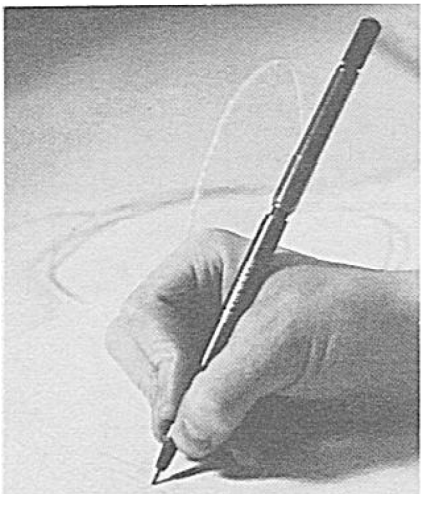

Fig. 3 – Pencil

making the record on paper correspond to the record in the machine. This is achieved by an eraser on the pencil which has its own sense coil, so that erasures on the paper can be recorded by the machine, in some sense, such as negative line.

In the current configuration the computer samples the position of the pen at constant intervals, variable under program control from 16 to 200 points per second. The data (100 points per inch in X and Y and 3 levels of Z) is stored in a disk file or on magnetic tape. Since the sample rate is constant, it is possible to determine the speed as well as position at any point in the sketch.

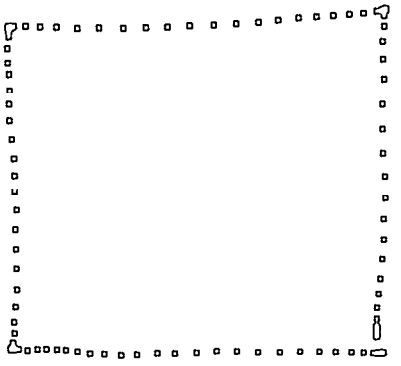

Fig. 4 – Raw Data

Finding Curves, Corners, end Lines

The original HUNCH system was conceived around a program called STRAIT, which found corners in a sketch as a function of speed, compressing the 100 points per second from the tablet into a much smaller list of endpoints of lines. The central assumption was that speed could be interpreted as a measure of intent-a quickly drawn line was intended less literally than a slowly drawn one. As a happy coincidence, it also turned out that as a result of the finite mass of the pen and hand, the speed necessarily decreased at corners. All that was necessary to find corners was to look for minima in the speed function. (Notice the bunching of raw data points at the corners in the previous figure.)

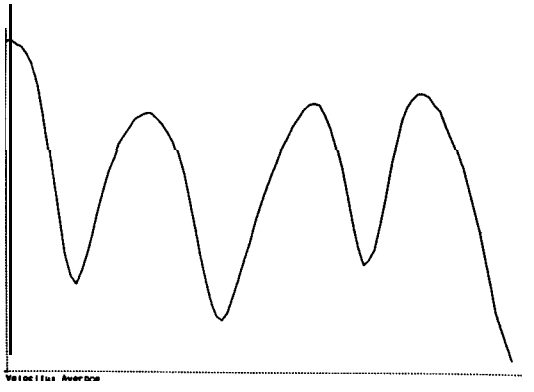

Fig. 5 – Speed Function for Figure 4

Curves were considered to be a special case of corners. With the assistance **of Dr. Richard Riesenfeld we decided to fit them *to*** B-splines[7]. **When** the curvature of a corner was too gradual, or the speed indicated that it was drawn too carefully, the output of the straightening program was flagged to cause subsequent invocation of the curve-fitting program. **CURVIT** would make one or more passes over the raw data at places pointed to by the output of STRAIT, producing its own output file.

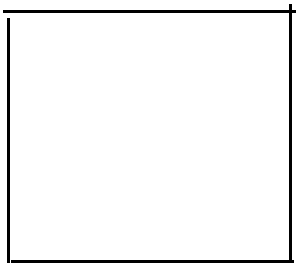

Fig. 6 -Output of STRAIN for Figure 4

When demonstrating STRAIT and CURVIT to visitors, it was somewhat disconcerting to find that it did not always make the same interpretation as the human observers. What was still more alarming, however, was that the program worked better for some people's sketches than for others. It appeared that the programmer had embedded a particular model of human sketching behavior that fit some users more closely than others. The question that arose was whether this model could be changed simply by varying parameters, and if so, how the correct values of those parameters could be determined.

Latching

A cursory examination of the above figure will reveal that it contains per force five endpoints. It would be convenient for programs using this data, such as editors or "square-finders" if there were only four endpoints. The first version of STRAIT used a simple latching algorithm which joined together any pair of endpoints falling within a fixed radius of each other. This simple "rote latching" often produced bizarre results where changes of scale were involved, as illustrated below.

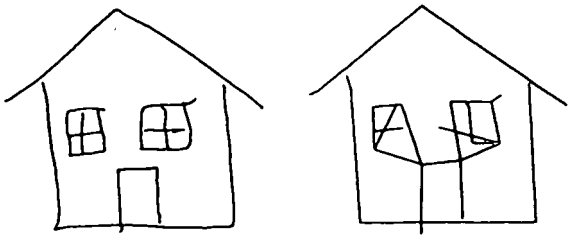

Fig. 7 - Rote Latching

With the realization that the latching problem was more complex than originally expected, STRAIT was rewritten without latching, hence STRAIN.'

Obviously some intelligent means of computing the latching radius is needed. In the next experiment, the latching radius is computed as a function of speed, under the assumption that the speed provided by STRAIN for each line is also a measure of the user's certainty in positioning that line's endpoint.

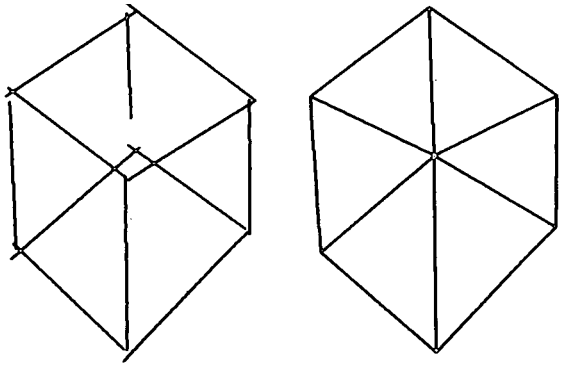

Fig. 8 - Latched Cube

As can be seen from the above, speed is not always well correlated with intent. In part this is because the speed figure used is the average for the line and does not accurately reflect conditions at the endpoints. But there is a more serious difficulty at work here-one which illustrates the interdependence of the different levels of interpretation. The program erroneously latched the front and back corners of the cube, since their two-dimensional projections happened to coincide.

*Another commonly used solution which was rejected was the use of a rectangular grid. In this method, the user can vary the resolution of the system, so that points digitized near grid points are adjusted to fall on these points. The grid requires that the user watch the display rather than the paper, or draw on graph paper, demands incompatible with sketching.

The only sure solution to this problem is to compare candidates for latching in three space instead of the (often ambiguous) two-dimensional projection. Paradoxically our current GUESS program requires latched data as input in order to determine the z-coordinates.

The solution will require that some latching decisions be made, perhaps on such syntactical considerations as the number of bodies or the sequence of construction, with the provision to modify decisions if they prove untenable in the later stages of interpretation. Several approaches to this problem will be discussed in subsequent sections of this paper.

Overtracing

Although not handled at all by the original HUNCH system, overtracings present many of the same problems as latching. Once again the process is one of reducing the quantity of data by making inferences about the user's

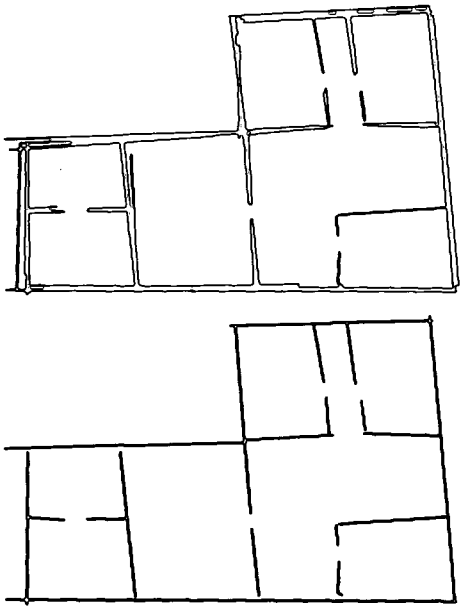

Fig. 9 - Overtracing

intentions, turning several lines into one line. The program must distinguish between two carefully drawn parallel lines and one wide, overtraced line. The amount and style of overtracing may serve as a measure of a particular user's attitude toward the design, with heavy overtracing indicating emphasis or reinforcement of selected areas.

Other Levels of Inference

In addition to the processes already mentioned, there are many other possible directions of interpretation. One of them involves inferring the third dimension of axonometric or perspective drawings. The currently implemented program makes use of some simple rules of projective geometry to map two-dimensional networks of lines into three-dimensional data structures.

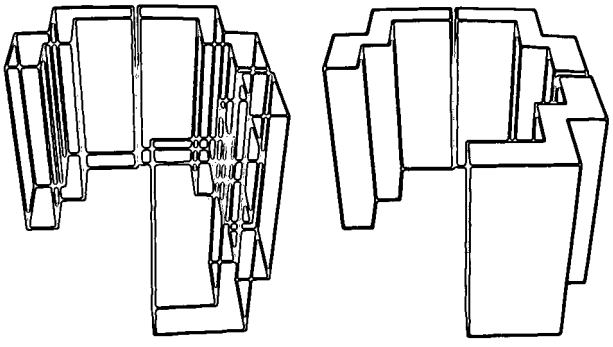

Fig. 10 - Three Dimensional Description

While easily confused by lines drawn in other than the three primary axes, the algorithm has enabled experimentation with manipulation of three-dimensional structures and cast some light on the interdependencies of latching and finding the third dimension.

Another type of interpretation is illustrated by a program which finds rooms and their connections in a floor plan. This program maps the raw data onto a two dimensional grid. Any point having a line pass through it is set to one. By mapping cells into an array at a suitable scale, any degree of detail may be chosen. The room-finding algorithm simply looks for areas of zeros completely surrounded by ones. Needless to say, the program works well for simple floor plans that conform to the programmer's original expectations. It works less well with odd-shaped rooms and doorways which

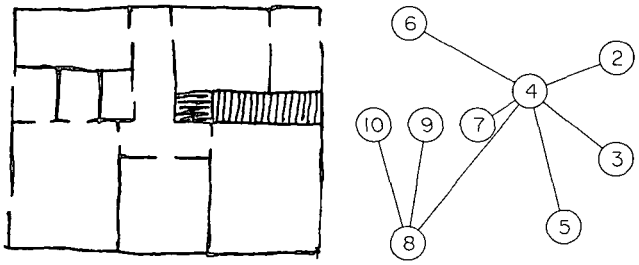

Fig. 11 - Room Finding

might not be doorways, as in the upper right room in the above figure. In this case, the choice of the "correct" interpretation is not so easy. In fact, it probably varies not only with different users, but also with different contexts. In architectural applications, the concept of "room" is usually less important than the ways in which the space is perceived and used-a level of inference perhaps above or perhaps equivalent to that of finding rooms.

A human observer resolves these ambiguities through the application of personal knowledge and years of learning how to look at pictures. Each line takes on meaning only in the context of the drawing and its subject matter. Before a machine can effectively communicate with a human user about a sketch it must possess a similar body of knowledge and experience.

3. USE OF CONTEXT

As suggested by the preceding descriptions, sketch recognition requires more than a simple hierarchical interpretation of the data. Although the programs described are usually run in sequence, the interpretations to be made are not independent. The bottom-up approach is hindered by the lack of any contextual information. In contrast to human vision, for example, where knowledge of what to expect in a sketch is involved in seeing things as detailed as individual lines, the HUNCH system operates solely on the basis of syntactic evidence. Its success can be attributed to its use of data not available to the human observer, such as speed and sequence.

The answer to the question posed at the beginning-is there a syntax of sketching independent of the semantics?-is still unresolved. While there is room for considerable improvement in the low-level inference-making routines, latchers that look for closure, overtracing routines that consider sequence, etc., it seems that a truly successful system would have to make use of context at the lowest levels. Such a context would have to include not only what the person is sketching but also the stage of completion of the sketch and the design, the idiosyncracies of the person, and his or her attitudes toward the design.

The knowledge of the sketching domain has to be structured in such a way that it can be used to direct the machine's analysis of the sketch, supplying goals to direct the low-level routines in their search for lines, curves, and corners and supplying enough information about what is plausible in the sketch to resolve ambiguities and fill in missing information. This knowledge also must be in a form that can be easily understood and modified by the machine so that a sketch-recognition system could be constructed which lends itself to diverse applications simply by changing the knowledge base.

A scheme employing these principles was explored in [9] and is synopsized in the following paragraphs.

The act of recognizing a sketch thus involves drawing a sketch for the machine and specifying the "context" to avoid the more difficult problem of "context recognition". This context specification might be as specific as "suburban homes" or "machine screws," or it might be as general as "built form" or "tools." The context description is structured as a network containing general case descriptions, for example:

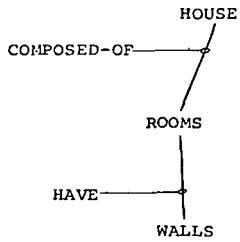

Fig. 12 - General Case Description

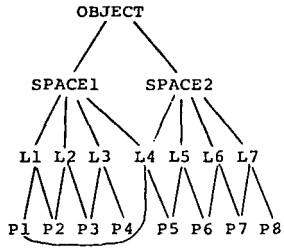

Fig. 13 - Context-Free Structure

This structure is matched against the context-free data structure generated by the low-level HUNCH routines to generate a composite structure where all of the syntactic entities of the sketch are assigned a meaning. The match-

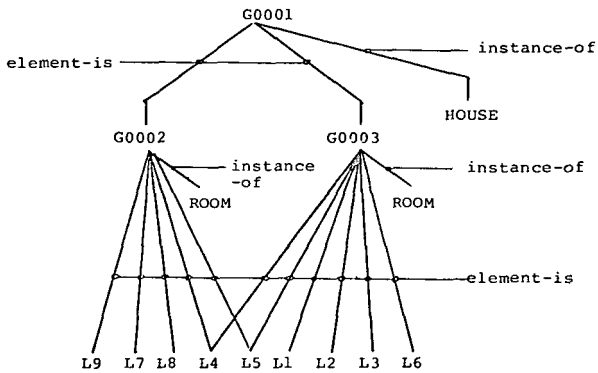

Fig. 14 - Instantiated Structure

ing operation takes place in a top-down fashion. That is, the machine attempts to find an instance of the top-level general case node in the specific data structure. In the example structures, a search would be performed for a room. Since there is nothing labeled "ROOM" in the data structure, the program searches for components of a room, specifically walls, which find their match in the lines of the data structure.

The process is, of course, more complicated than a simple recursive depth-first matcher, since the general case description will not match exactly, but only in part. If a required line does not exist in the data base, it may be necessary to reinvoke the line finder, which may have thrown away a line the first time. A yet more difficult problem is posed by erroneous or premature matches: the system is only as good as the matching machinery. Either some method must be provided for withdrawing incorrect matches, necessitating a control structure employing backup, or the matching process must be made more breadth-oriented, requiring a sophisticated pattern matcher which can look around and size up the situation before making a match. A preliminary system using the first approach was implemented in the CONNIVER[10] language, an artificial intelligence (AI) language which allows construction of complex control structures and provides facilities for maintaining and searching a data base. Unfortunately, the CONNIVER pro-

gram, while small in capability, consumed enormous amounts of computer time and memory. Any attempt to expand the capabilities of this modest program led directly into the same (vast) problems encountered by AI researchers working on machine vision or natural language. It appeared that the knowledge-based approach would require a rather large, permanent detour away from the initial problem of providing a graphical input facility.

4. TOWARD AN INTERACTIVE SYSTEM

The most promising approach seems to be a much more interactive system, one involving the user to make decisions of which the machine is not capable, but still affording the unobtrusive input method of sketching. The program must take into consideration the interdependence of different levels of interpretation. The bottom-up flow of information operates as in the HUNCH system, except that the data base is more integrated than the disparate disk files of HUNCH. The top-down flow, which is one of the strong points of the context-based system, comes from the corrections and manipulations of the user, as well as from higher-level programs. These capabilities require a much more integrated and interactive facility than is provided by HUNCH. Such a system forms the basis of our current work and is described in this section.

The new system consists of a data base and a set of programs to manipulate it. The data base is stored as PL/1-based structures, capable of being paged from a disk. Each time a new level of interpretation is encountered, a corresponding level is added to the data base. Pointers are maintained to show relationships between elements at different levels. For example, when

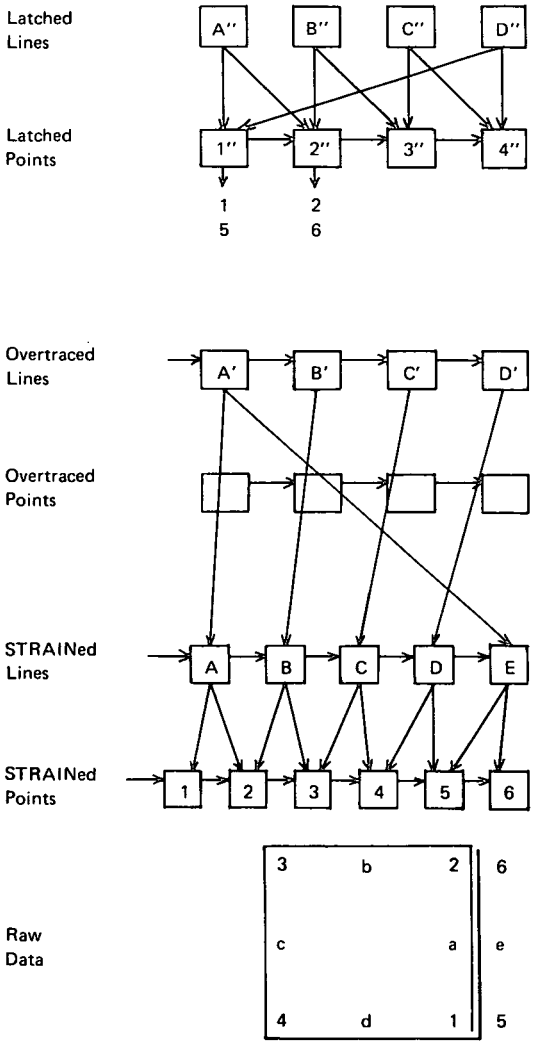

Fig. 15 - Data Base Organization

two points are latched together, pointers are stored at the latched level to show which two STRAIN points make up the new single latched point.

As in the HUNCH system, there are three kinds of programs which use the data base:

Inference programs are improved versions of STRAIN, LATCH, OVERTRACE, and GUESS. Each one employs tests of reasonableness to its output and can be controlled by means of a well-defined set of parameters.

Display programs allow displaying any (or all) levels of the data base on selected output devices.

Manipulation programs have been combined into a graphical editor which permits the user to modify the data base directly.

Finding Lines and Curves

If the new sketching system is to be truly interactive, STRAIN must operate in real time, finding lines and curves on the fly, while the user is drawing, instead of after the fact. The program must lend itself to "tuning" to the hand of the individual user.

Accordingly, the new program is based on a set of functions, each of which measures some characteristic of the sketch over an inverval of variable but specified size and position. Two of the most useful functions are speed and "bentness," which are shown below for a square composed of 95 sample points.

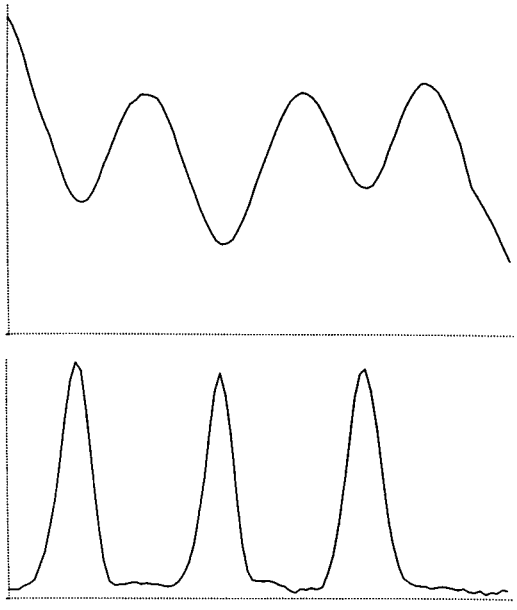

Fig. 16 - Speed and Bentness

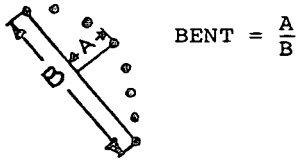

Fig. 17 - Bentness

Speed measures the length of the interval divided by the number of points in that interval. Bentness measures the maximum of the distances from the points to the chord connecting the endpoints of the interval. By varying the number of points in an interval, the number of peaks in the output can be controlled. A large interval filters out peaks resulting from local, high-frequency variations, while a small interval will show up every tiny variation in the sketch. This interval length is one parameter which can be varied to get the closest fit between the machine's interpretation and the user's intentions. For example, if the program were told that the user has drawn a

square, it could vary the size of the interval until the two functions produced exactly five peaks (remember that one corner, per force unlatched, is both a beginning point and an endpoint).

Given these two functions, they can be combined so as to indicate intended lines and curves. The contributions of the different functions are determined by a set of coefficients which can be varied for individual users. Bentness is used to find the actual corners in a sketch, while speed is used to indicate whether they are really intended as corners. These functions will eventually be computed by microprocessors, so that the main program will see a tablet which produces not only X, Y and Z but also speed, bentness, corners, and curves. The raw data will still be saved, along with the function values, so that other programs can examine the data.

The line/curve finder will be tunable both explicitly and implicitly. Explicit tuning is accomplished through a command which enables the user to modify the program's parameters to produce the most satisfactory result. It is envisioned that implicit tuning will be performed by means of the editor. If the user inserts points, creating corners in the data base, the system can change its parameters to generate more points. If the user deletes points, that serves as an indication that the corner-finding program is generating too many corners.

153 Words About Control Structure

Unlike HUNCH, where the user explicitly invoked programs such as STRAIN and LATCH, the new sketching system runs its inference-making procedures in the background, so that they will not cause interruptions in the sketching and thinking processes of the user. Placing the pen on the tablet immediately starts the line/curve finder, adding the new lines and curves to the data base. Whenever the user is not drawing, the higher-level programs, such as OVERTRACE and LATCH, are run. These programs look for any changes or additions to the data base, such as would result from drawing, editing, or running other inference making programs. A program which modifies the input level of another program causes that program to be run. Conversely, any program may decide that its input data level is unsatisfactory and request the program which created that level to regenerate it, perhaps using new parameters.

Latching Revisited

In view of the new data base and control structure, it may be useful to examine the metamorphosis of the latcher from a stand-alone program to part of an integrated system. Latching is an interesting problem from many viewpoints. While its ultimate solution requires a knowledge-based system, there are many clues available to the program which are independent of the subject matter being drawn. These include closure, changes in size or angle, and the number of lines latched to a point.

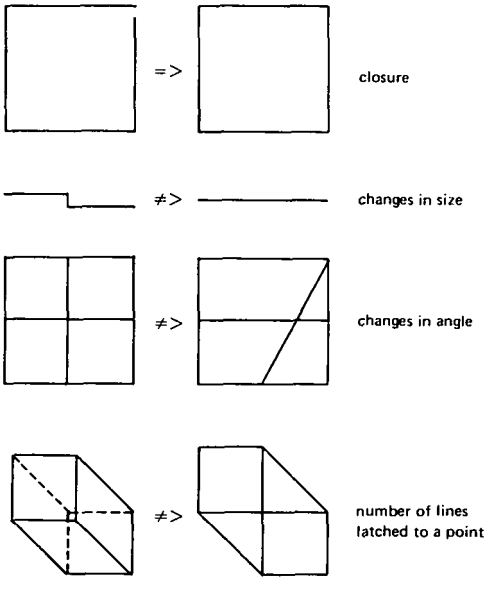

Fig. 18 - Latching Criteria

The latcher can attach a certainty factor to each of its local decisions, based on parameters such as speed and line length. These decisions can then be examined more globally in the light of the criteria mentioned above. Since those criteria tend to bias the interpretation in certain directions, their use will have to be controlled by a profile of the user created from a history of corrections he makes to the output of the latcher. On a more local level, these corrections can be used to control parameters such as the latching radius. By allowing profiles to be generated on an individual person/subject basis, some of the benefits of subject-area-based systems can be acquired.

Another problem in latching is the determination of the scale at which the designer was working. This problem is exemplified by the drawing in Figure 7 where the windows were drawn at a smaller scale than the house. The program, assuming one scale, used two large a latching radius on the windows, latching too many points together. Scale will have to be determined by use of various clues, such as the "busy-ness" of a part of a sketch, as measured by the density of lines drawn nearby in soace and in time.

The sequence of line input can be another useful clue since, for example, the two faces of a cube will probably form two separate clusters if divided by sequence. Once again this is a user-dependent property.

CONCLUSION

Progress in sketch recognition has in a sense come full circle from an insistence on machine recognition with no demands on the user, through knowledge-based systems, and back to a more modest interactive approach. It is our intention to involve the user in the way most meaningful to his design, through editing the machine's interpretation of his drawing to bring it into agreement with his own intentions. Our experiments have shown sketching to be a viable medium through which a person can communicate to a machine. The next question to be answered is. What balance can be struck between an intelligent but unwieldy system, and a tiresome but practical one?

REFERENCES

[1] Negroponte, Nicholas, and James Taggart, "HUNCH-An Experiment in Sketch Recognition," in *Computer Graphics,* edited by W. Giloi, Berlin, 1971.

[2] Negroponte, Nicholas, "Recent Advances in Sketch Recognition," *Proceedings of the AFIPS,* New York, 1973.

[3] Taggart, James, "Sketching, an Informal Dialogue between Designer and Computer," in *Computer Aids to Design and Architecture,* edited by Nicholas Negroponte, Petrocelli/Charter, New York 1975.

[4] Negroponte, Nicholas, "A Computational Paradigm for Personalized Searching," in

[5] "MAGIC Reference Manual," Architecture Machine Group, MIT, Cambridge, 1976.

[6] McIntosh, John F., "The Michigan Pencil," unpublished paper, Architectural Research Laboratory, University of Michigan, Ann Arbor, September 1975.

[7] Riesenfeld, Richard, "Applications of B-spline Approximation to Geometric Problems of Computer-Aided Design," Ph.D. thesis, University of Utah, Salt Lake City, 1973.

[8] Herot, Christopher, "PLAN," in *Machine Recognition and Inference Making in Computer Aids to Architecture,* proposal to the National Science Foundation, Architecture Machine Group, MIT, 1973.

[9] Herot, Christopher, *Using Context in Sketch Recognition,* Master's thesis, M.I.T., Cambridge, Massachusetts, 1974.

[10] Sussman, Gerald, and Drew McDermott, *The Conniver Reference Manual,* M.I.T. Artificial Intelligence Laboratory, M.I.T., Cambridge, Massachusetts, 1973.