

22: MAN-MACHINE SYSTEMS

HUNCH
AN EXPERIMENT IN SKETCH RECOGNITION

The Architecture Machine Group
Department of Architecture
Massachusetts Institute of Technology

Nicholas Negroponte
Leon B. Groisser
James Taggart

This paper highlights the operations and problems of a set of computer programs called HUNCH. The specific goal of HUNCH is to be able to recognize and make inferences about a user's sloppy, incomplete and equivocal drawings without that user having to be more explicit or categorical than he would be with an on-looking human colleague. Steps toward this goal are being made within a particular context -- architecture -- that furnishes a "knowledge base" or "assumption base" from which programs can procure (rather than develop) those heuristics necessary to handle two dimensional and three dimensional ambiguities.

While HUNCH is indeed an exercise in "computer graphics", we believe that a great many of its operating principles are characteristic of all kinds of future systems that will be expected to cope with vague information and to deal with nebulous ideas. For example, 1) we not only expect the program to make errors, but also to benefit from errors (ie: to improve its recognition ability and context handling ability); 2) we expect the program to support an evergrowing acquaintance with the user (ie: a knowledge of his idiosyncracies, habits, whims); 3) we anticipate that the program will have many channels of access to the real world (to the extent of recognizing gestures and smiles); 4) we insist that the program pursue a multiplicity of candidate "solutions", sorted and selected by "criticizer" programs.

Since 1967, the authors¹ have argued that computer-aided design is an issue of artificial intelligence, that machines have to understand the meaning of decisions in order to be partners in making them. We have advocated the exploration of artificial intelligence and have condemned computerized solution generation which does not consider it. One severe consequence of this position is that there remains very little to work on both because of the state of the art of computer sciences in general, and because intelligence per se cannot be tackled in parts.

Sketch recognition has provided us an interim domain. While it is certainly difficult to shift between wanting to deal with "partially formed ideas" and wanting a program to find corners (which is what most of this paper is about), it is a topic that has inherent uncertainties and ambiguities at all levels. It is representative of a class of problems that can be tackled with modesty, but in the ultimate case, require an intelligence.

Acknowledgements

Precursory work commenced in 1966 in a project called URBAN5 jointly sponsored by IBM and MIT. Actual work, however, started only in early 1970 with the Bachelor of Architecture thesis of Mr. James Taggart. This work was conducted within the Architecture Machine Group at MIT and was supported by the Ford Foundation. The more concentrated effort began in the summer of 1970 with a grant from the Graham Foundation for Advanced Study in the Fine Arts and major support from MIT's Project MAC (Advanced Research Projects Administration contract N0014-70-A-001).

At present HUNCH is being developed as an integrated package to act as a front end for any system for "Computer Aids to Participatory Architecture",² sponsored by the National Science Foundation.

Along with these many generous sponsors, the authors would like to thank Dr. Gordon Pask who, in spring 1971, provided radical inputs, which guided the project into new directions.

Introduction

We view the problem of sketch recognition as the step by step resolution of the mismatch between the user's intentions (of which he himself may not be aware) and his graphical articulations. In a design context, the convergence to a match between the meaning and the graphical statement of that meaning is complicated

by continually changing intentions that result from the user's viewing his own graphic statements. In some sense, the sketch acts as a form of memory for the user while he loops, so to speak, into the real (physical) world to gain a better understanding of his problem. Consequently, the nature of his drawing (ie: the wobbliness of the lines, the collections of overtracings, the darkness of the lines) has important meanings for the most part overlooked in computer graphics. We have already proposed that: "A straight line 'sketch' on a cathode ray tube could trigger an aura of completeness injurious to the designer as well as antagonistic to the design."³

HUNCH tries to take into account some usually overlooked (or not taken seriously) graphic subtleties such as: speed of drawing, pressure upon the pen, and crookedness of lines. At present, this is for the purpose of making those transformations necessary to pass from a drawing that is meaningful to the user (figure 1) to an interpretation (figure 2) which is both manageable by the machine and appropriate for first approximations, guesses and extrapolations. Later, we expect to employ graphical innuendoes for the purpose of having the machine recognize attributes which have to do with the user's attitude toward his own project, his confidence in a solution and the like.

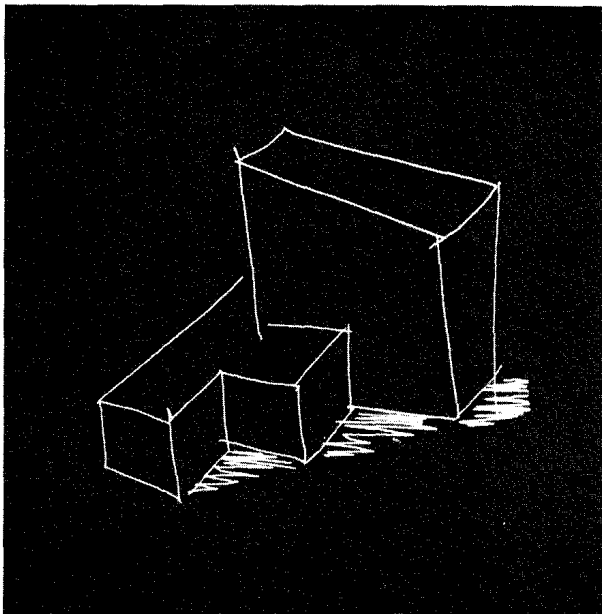


Figure 1.

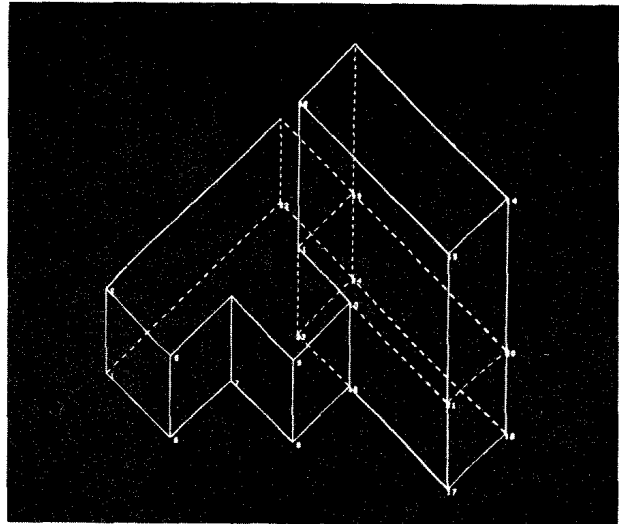


Figure 2.
We believe that HUNCH will have general applicability as a front end to computer systems that require the graphical input of ideas which are not well formed. It should have particular importance for architectural applications inasmuch as present day computer applications are always hampered by the means of input. The architect interested in computer-aided design techniques must either 1) do a design away from the machine, and at some level of completion initiate the usually clumsy procedures necessary to make it readable by the machine; or 2) must stymie his own design behavior and subject himself to drawing techniques which are inappropriate for "creative" thought. Neither is suitable.

Within the next two years, HUNCH is expected to handle true sketches (figure 3), drawings marked by inaccuracies, missing information and even coffee stains. We propose to do this by: 1) providing the user with a broad range of amenable hardware interfaces with the machine; 2) developing programs that get to know the particular user better and better; 3) creating an inference-making system that capitalizes upon a history of encounters with many users as well as with the particular user.

The following pages should be viewed as an interim report.

Configuration

We start with a description of the configuration of our hardware because we believe that each device has unique operational characteristics built into it.

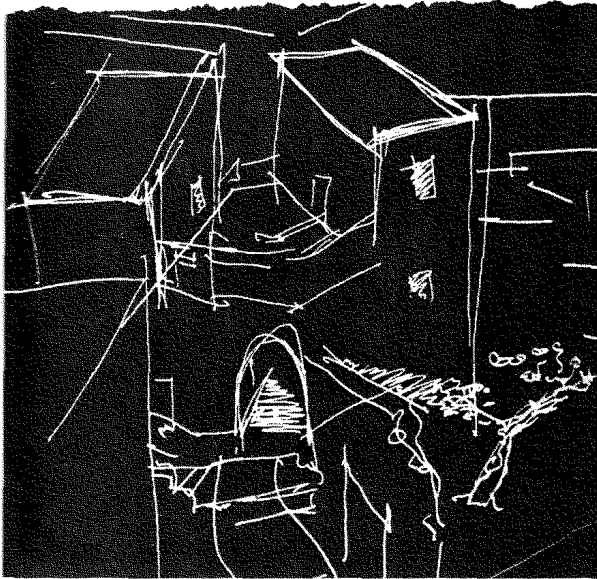
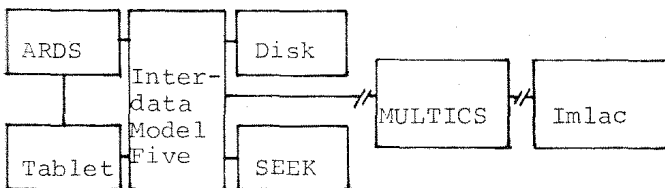


Figure 3
We depend very heavily upon these inherent features for the purpose of recognition as well as of intensive, congenial interaction. The hardware described is a subset of what we call "The Architecture Machine" (figure 4).

The diagram illustrates the subset of The Architecture Machine used for HUNCH as well as the interconnections among the seven operating elements: a Sylvania Tablet, an Advanced Remote Display Station (ARDS), an Interdata model 5 minicomputer, a DisEstor



disk, the MULTICS time-sharing system, an IMLAC display unit and a three dimensional positioning device called SEEK. The reader should note that this configuration is highly redundant and, while it employs a plurality of devices, it is marked by a very low cost, approximately \$40,000 purchase price.

Sylvania Tablet

The Sylvania Tablet, with its stylus, is the primary input medium of the system (figure 5). The tablet operates in a manner that is crucial to HUNCH. It issues to the minicomputer a constant number of X and Y (and even Z) coordinates per second (in our case 200 hundred per second). This constant rate acts as a form of a clock.

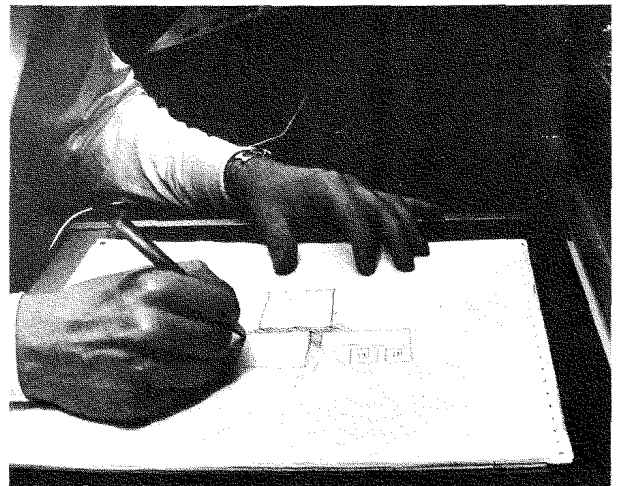


Figure 5

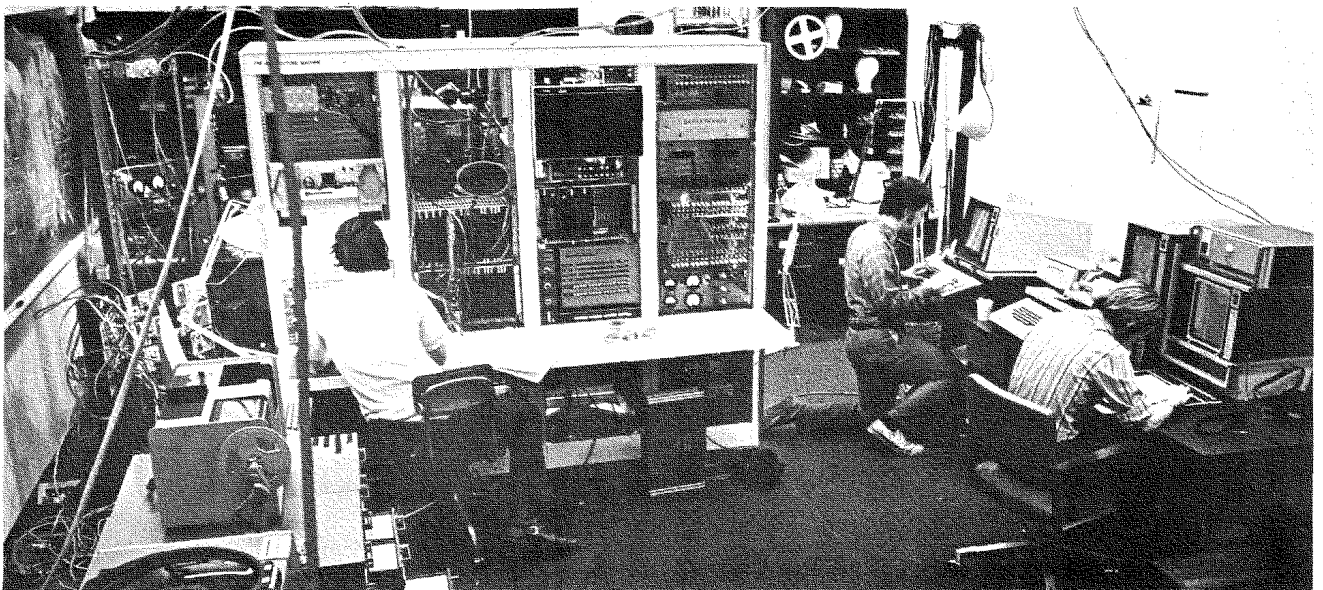


Figure 4

It means that lines drawn slowly will have digitized points closely spaced, whereas lines drawn rapidly will have them more dispersed. As a result, we have a built-in record of the speed at which each part of a drawing is created. We subsequently employ this parameter as a major criterion for determining the user's graphical intentions: "did he mean this to be ... a straight line, a square, a corner, etc?". In effect, we are correlating the user's speed of drawing with his purposefulness.

We also associate intentions with the user's pressure upon the stylus. This feature has been added to the Sylvania Tablet by Wade Shaw as part of his Electrical Engineering Thesis, Textural Input and Definition for Graphic Display. A pressure sensing gage placed within the shaft of the pen measures how hard the user is pressing down. When connected with the focus control of the display, this feature simulates pushing harder on a pencil to get a blacker line. We believe that this is particularly important to sketching because it is quite common to over-trace lines, and thus without erasing, to make the most up-to-date lines emphatically dark.

It is important to note that both speed and pressure provide inputs which would not really be available to an onlooking human, especially if he were looking at the sketch after the fact. Some argue that it is more appropriate or meaningful to artificial intelligence to use only those cues available to humans. We don't.

The tablet has two further important features: First, it has a transparent surface which permits it to be used vertically in front of the display with the minor inconvenience of a half inch parallax (figure 6).

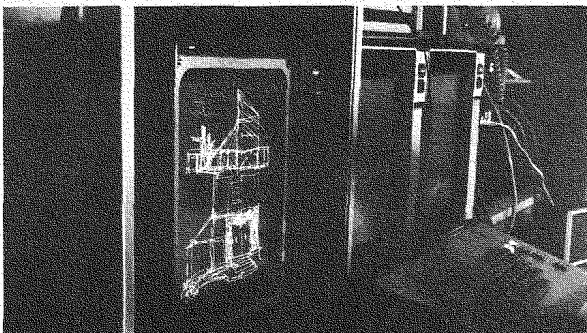


Figure 6

This provides an ability for drawing without the computations necessary for pen tracking (which is necessary for light pens). Second, it has a limited three dimensional capability because it can record three levels of Z (where the range of each level is variable with a screw driver). At this time we make only limited use of this last feature (for example, we determine the probable beginnings and ends of pictures and opportune moments to write information on the disk).

Advanced Remote Display Station
ARDS was the first storage tube system developed (figure 7). The advantage of a storage tube is that it maintains the image on the face of the scope without "refreshing" it and thus without the associated computations necessary to continually redraw the image. (Hence its convenience to time-sharing.) Its drawback is its resulting inability to locally erase, that is to remove a part of the picture without erasing the whole and redrawing it.



Figure 7

This does not pose a problem in our context; consider whether you can locally erase with a felt tip pen, or how often you erase pencil lines. Furthermore, a storage tube allows an unlimited number of vectors to be displayed (in contrast to the more expensive "dynamic displays" that allow only a few seconds of sketching before flicker sets in because so many points must be continually drawn).

Storage tubes have the ability to "write thru". This mode of operation can be initiated under computer control. It affects a low beam voltage which produces

vectors that are not stored. Therefore, with a minicomputer at hand you can partially circumvent the local erase problem and maintain a certain number of vectors dynamically -- we employ this for editing and moving about elements of the picture. (It should be noted that this feature varies dramatically with the brand of storage display station. The ARDS is the least appropriate in that it steps vectors, rather than sweeping them, and hence cannot display lines as fast as other brands.) A drawback of write thru mode is that storage tubes are inherently dim, write thru is even dimmer, and consequently when using it the user must operate in a partially darkened room.

Interdata

The Interdata Model 5 computer is typical of any good minicomputer; it is very fast and provides very cheap computing power. The processor and memory is similar to that employed for HUNCH and can be purchased for \$10,000. Since the Interdata is faster than both the Tablet and the ARDS, it can perform background operations, in effect, dealing with the sketch in parallel with the user creating it. Its drawback (not inherent in the computer) is its comparatively small memory. Given 200 points per second (4800 bits per second), a 16K memory is filled in only 16 seconds of sketching; hence an auxiliary memory is necessary.

DisEstor

The DisEstor is a removable disk pack system, with single surface disks (512K bytes per side) and fixed heads, one per track. The fixed heads eliminate "seek time" making the disk extremely fast. This feature permits the disk to accept the initial sketch as if the tablet were connected directly to core memory. With the disk, the user can draw continuously for over an hour. However, since nothing is stored when the pen is lifted off the tablet, this means about four or five hours of natural sketching.

MULTICS

MULTICS is a time-sharing system developed by MIT's Project MAC on a GE645 computer. It has powerful file-sharing abilities and caters heavily to systems programming.

It may be argued that time-sharing will not be cost effective when, in ten or twenty years, minicomputers sell for \$200. Its future role may well be that of communications and of sharing large data bases, and then only by machines

(ie minicomputers) not people. Nevertheless, presently it is particularly useful because it makes available high level languages and their software support: these usually are not available on minicomputers. It is for this reason that we use MULTICS. The list processing and recursive features of the MULTICS version of PL/1 are necessary for mapping a perspective sketch from two dimensions.

We also employ the time-sharing system as a communications medium between the ARDS Interdata display and the IMLAC.

IMLAC

An IMLAC is a display unit which has a self-contained processor memory and display processor that can support up to inches of vectors (depending upon available memory and other options). It has recently been added to the HUNCH configuration in order to experiment with those dynamic features usually available only on very expensive display units. At present we are programming it to present the results of the mapping into three dimensions performed on MULTICS. This includes rotations, translations, etc., and a limited ability to change proportions by moving planes while keeping the topology constant.

SEEK

SEEK is a homemade, three dimensional positioning device that can build physical models using predescribed blocks of wood (figure 8).

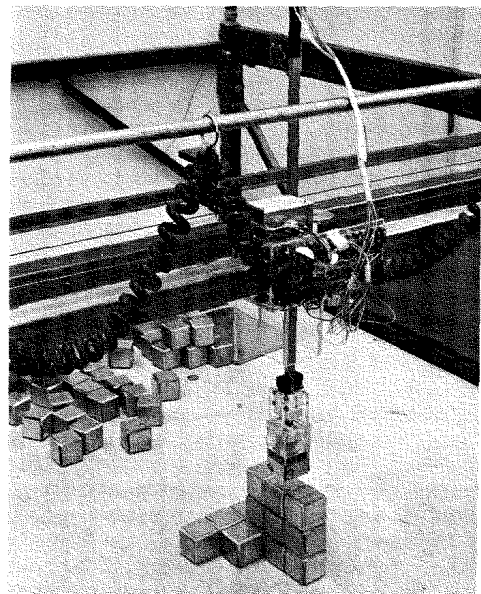


Figure 8.

It receives its instructions from the Interdata which in turn has received from MULTICS the X,Y, and Z coordinates of the centroids of the blocks necessary to best approximate the sketched image. In some sense it is a punchline to the demonstration program we employ to demonstrate the present workings of HUNCH; it is not an integral part of sketch recognition. However, consider the spectacle: you sketch a perspective of an assemblage of housing units; behind you a machine is building them. You change your sketch; the physical model is changed.

Operations

In this section we take some liberty in mingling the present and future tense with respect to what is and what is about to be operational. A great deal of HUNCH is presently working and demonstrable at the lowest levels of recognition; very little is developed at the upper end. The operations are carried out in a somewhat sequential manner and are presented following that order. However, it is important to recognize that the initial data and the subsequent transformations are saved on disk. This permits interactive re-examination of the data in order to build evidence that "he meant this" or "he meant that". This contradicts the sequential nature of the following presentation and the present demonstration program. Being able to look back, to collect more evidence, and to alter previous (now proved erroneous) decisions are all crucial to sketch recognition.

Initiation

HUNCH is being developed to be trained (we use the word advisedly) by a particular user. This training will occur both implicitly and explicitly through reinforcing an evergrowing model of the user and his method of work (particularly his method of drawing). In order to start the program, some initialization is necessary when the user and HUNCH first meet. No two people draw in the same manner and HUNCH would flounder without some means of correlating the pressure and speed of a given user with his graphical intentions. In the early days we saw this dramatically demonstrated when the initiation procedures did not exist and the program was tuned to the hand of its chief programmer, Mr. James Taggart. If another member of the Architecture Machine Group would try to demonstrate HUNCH, the result would be embarrassing.

Therefore, a simple scenario precedes the first sitting. "Draw me a square," says HUNCH. The user draws a square and HUNCH proceeds to vary its own parameters of speed and pressure until it manages to compose a square out of the stream (let's say 600 hundred or so) of x's and y's. Then it asks in turn for a fast square and slow square. Eventually enough parameters will have been set to enable HUNCH to proceed with the different levels of recognition and data compression.

Diagrams

After initialization, the first operation which must be performed on the stream of data (ie: sketch) is the recognition necessary to separate diagrammatic elements from projections (either orthographic or perspective). In projections, the vectors delineating a figure represent the intersection of planes or delimiting contours of physical bodies; in diagrams, they do not. Letters of the alphabet, arrows and symbolic notations are examples of diagrammatic elements. Plans, sections and perspectives are typical projections. At present we handle only one kind of diagram -- the squiggle. We anticipate handling many others including a character recognition ability.

"Squiggles" must be distinguished from intentional wavy lines. This can be done with uncanny accuracy simply by analyzing the speed and the accelerations at the corners. Once recognized as an authentic squiggle, we consider it to be only one of two things: 1) shading; 2) a rubbing out gesture.

It is important to recognize shading, not only to avoid processing it "projectively", but also to have it as future evidence for guesses about solidity and orientation (figures 9 and 10). The direction of the sun, the parallelism of its rays, and the interpositioning necessary for shadows are simple concepts, easily included in programs (though we would prefer that the programs could develop the concepts themselves). Such an analysis affords excellent evidences for locating the ground plane or perceiving depth.

The rubbing out gesture is particularly important for editing. This technique of rubbing out using squiggles, is found as far back as 1969 in the Grail Language⁴ developed by the Rand Corporation. It is distinguished from shading by con-

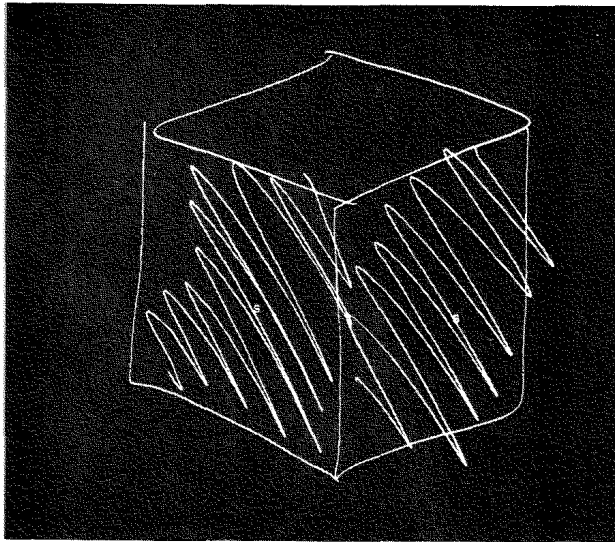


Figure 9.

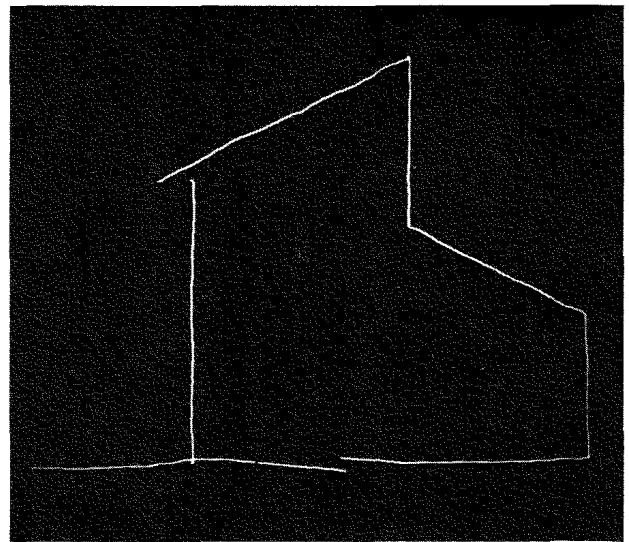


Figure 11.

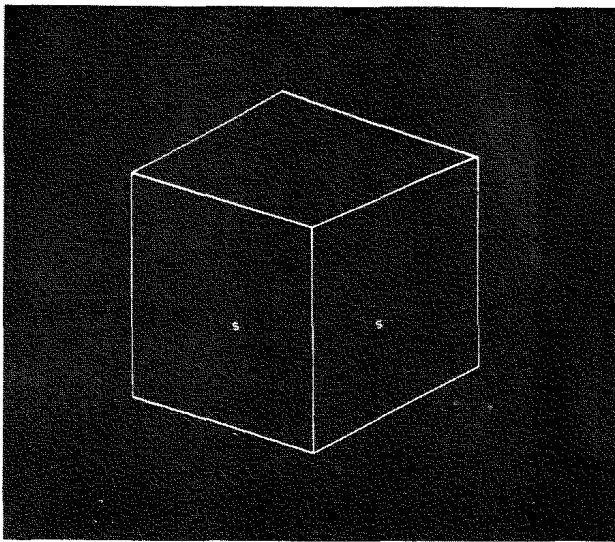


Figure 10.
considering when (in the sequence of constructing or editing an image) it is drawn, and by the location of its centroid (ie: is it on top of a line or within a surface?)

Data Compression

Data compression is a major element in the presently operating version of HUNCH. Figure 11 represents 4128 core locations as raw data; figure 12, the same drawing, represents 128 locations after data compression. This operation is important for subsequent inference making and for transmitting the image to the time-sharing system at 1200 bits per

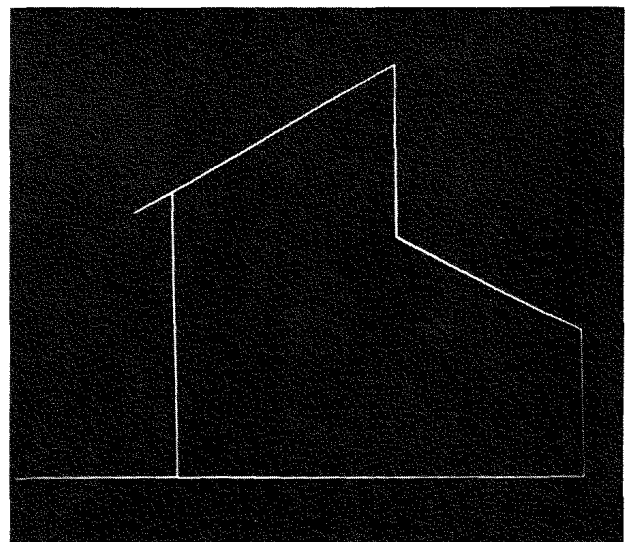


Figure 12.
second. The adjacent image would take half a minute to transmit unstraightened, and less than a second after data compression.

Line straightening is not a trivial operation. The problem is that decisions on the straightness of a line cannot be made by looking only at a given point and its immediate neighbors (let's say within an eighth inch radius). Local nicks and bumps caused by a shaking of the wrist, a bump on the paper, or simple noise in the tablet could indicate corners that were indeed unintended. Instead, it is necessary to connect, for example, every

tenth point and look for violent changes in angle (arctangent) among the resulting line segments.

This procedure is greatly assisted by considering the speed of drawing; a hastily drawn square with rounded corners (figure 13) will be assumed to be a closed straight line figure, whereas one drawn slowly, of identical shape, will be interpreted as a figure meant to be rounded.

In fact, it is interesting to note that a stream of incoming graphical data can be examined solely for peaks and troughs in speed and that if the points of slowest velocity were interconnected, the resulting image has about 80% fit with the intended image!

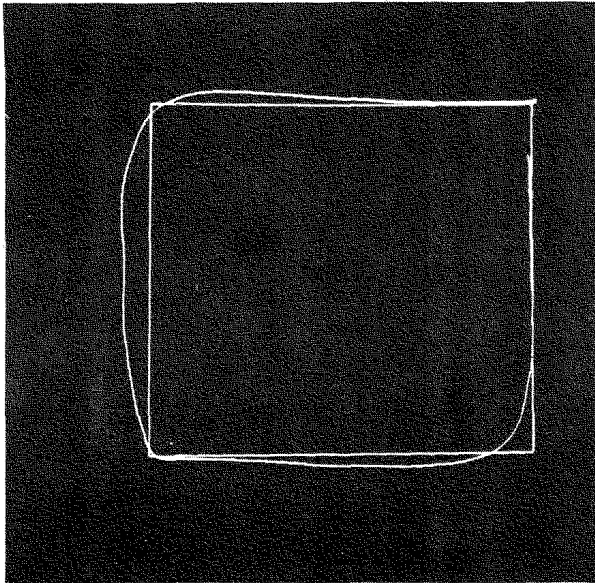


Figure 13.

Curve Recognition

We have ignored curves for a long time. They are not indiginous to architecture; we don't believe that sophisticated computer graphics techniques will lead to a proliferation of Gaudiesque architecture (in fact we might want to discourage it). However, in any demonstration we have ever conducted, the visitor/user inevitably draws cruves. Thus we are willing to recognize them, but in no sense to try to "fit" them.

Fortuitously, as a direct result of the above data compression procedures, curve recognition is relatively simple. The method we employ is to slightly vary those parameters that control the importance of speed. If the user intended

to draw an actual curve (versus a wobbly straight line) the straightened interpretation will be dramatically different when different speed factors are employed. If a user intended to draw an oval, the straightened version is almost unpredictable: we can be assured, however, that each straightened version will be markedly different from the others. Figure 14 shows a curve straightened employing three different weightings for the speed parameter. Figure 15 shows a curve that has been recognized and subsequently left unstraightened. In effect, what we are capitalizing upon is the results furnished by a procedure given a task it was not meant to handle.

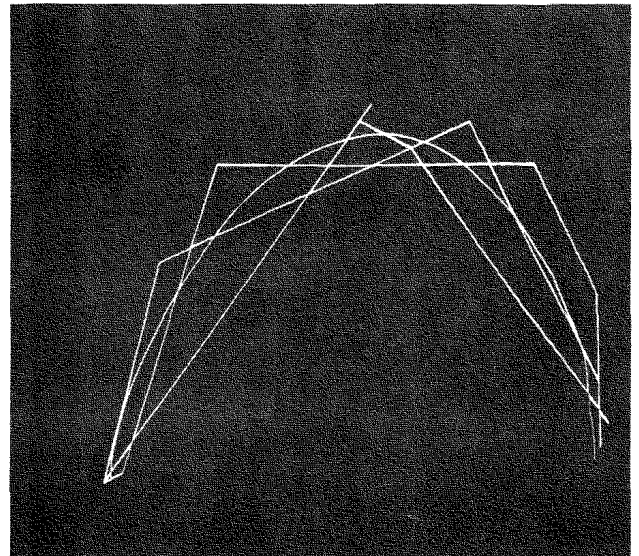


Figure 14.

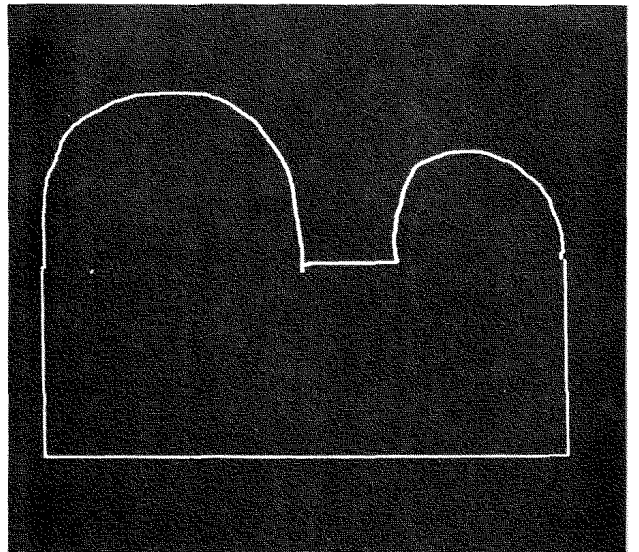


Figure 15.

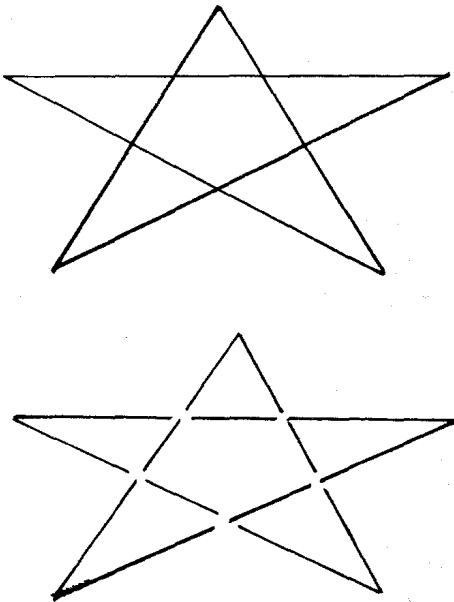
Latching

Our first approach was that the latching of lines should be purely a function of velocity. Two lines that almost meet, but that were drawn slowly (where slow and fast are in the vernacular of the particular user) were not to be connected.

However, this approach turned out to be wrong when users composed small, intricate drawings. The mullions of a window, for example, would inevitably latch onto the corner of the sill, offering absurd interpretations of the sketch. The error resulted from the philosophical oversight of not looking for all the candidates (for latching, in this case) and choosing the most appropriate (or none at all). There are obvious and important determinants for latching beyond simply speed: is it a connected body? is the shape closing upon itself? what does this do to horizontality and verticality? and so on. The result is a much more timid "latcher", but a much more correct one. (More farreaching consequences are discussed in Section IV.)

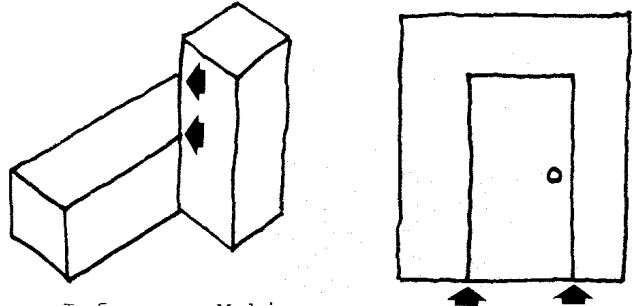
Intersections

After a sequence of lines have been straightened (ie: compressed), the lines are represented only by their end points (See Appendix). HUNCH then locates intersections, and breaks the intersecting lines into line segments. Thus a five pointed star results in fifteen line segments with ten end points.



The problem that we suspect may arise will be when we want to recognize a star or "starness" which is certainly intrinsically more than a five sided "thing".

Nevertheless, in some contexts, intersections are crucial, especially when a line segment meets another line at other than an end point. This occurrence results in a so-called T joint. This is also a problem because an almost meeting T or an overshoot T is made to end exactly at the line (as a function of velocity and pressure). It is interesting to note, however, that non-perpendicular T's are often unlatched by later "mapping" operations since they are good evidence that one body or plane lies behind the other.



Inference Making

In the jargon of Sketchpad⁵ and Sketchpad III⁶ making inferences is "constraint resolution"; in our case, it is handled implicitly. We consider, for the time being, only four types of inferences that the machine makes without the user's explicit request: 1) horizontal/vertical, 2) parallel/perpendicular, 3) continuities, 4) overtracing.

When almost horizontal and almost vertical lines are made plumb, the picture takes on a dramatically different character. This is particularly true inasmuch as the preceeding routines, straightening and latching, frequently operate at the expense of intended horizontality and verticality as dramatized in figures 16, 17 and 18. The operation is especially important in the context of architecture, where verticality and horizontality have structural meaning. In areas of industrial or city design, this operation would have less propriety.

Parallelism and perpendicularity, on the other hand, are more context dependent. Inferring that two almost parallel or perpendicular lines were meant to be parallel or perpendicular is more difficult. This is because the positional relationship between two lines must be considered. In the case of parallelism,

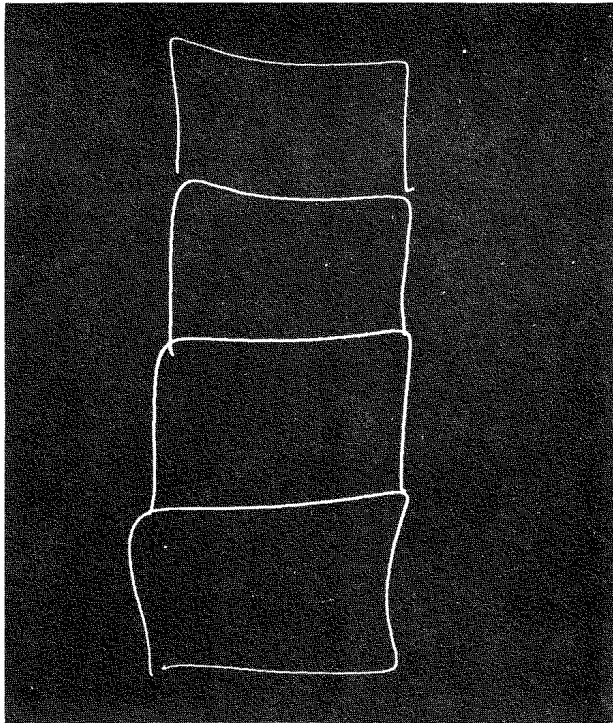


Figure 16

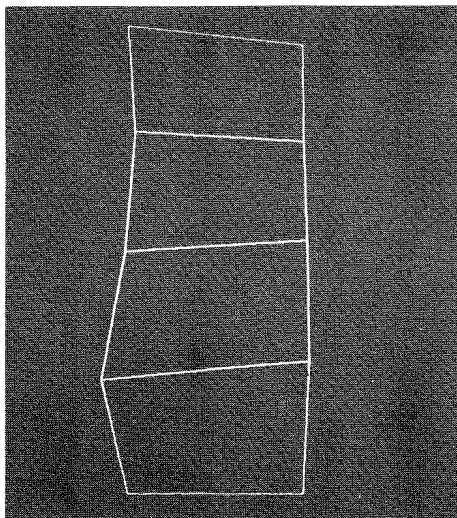


Figure 17 for example, the paradox exists that distant lines that happen to be nearly parallel are less likely to be intended to be parallel (especially if they are short and many other lines intervene); however, it is more difficult to draw lines parallel when they are far apart, no matter how carefully it is done, (again, especially when they are short). The problem is only manageable by the fact that over-enthusiastic orthogonalizing and parallelizing does not produce drastic errors (as does a nontimid latcher).

Continuities are once again rather architectural in nature. They are heavily reinforced by the sequence of the drawing and have very little

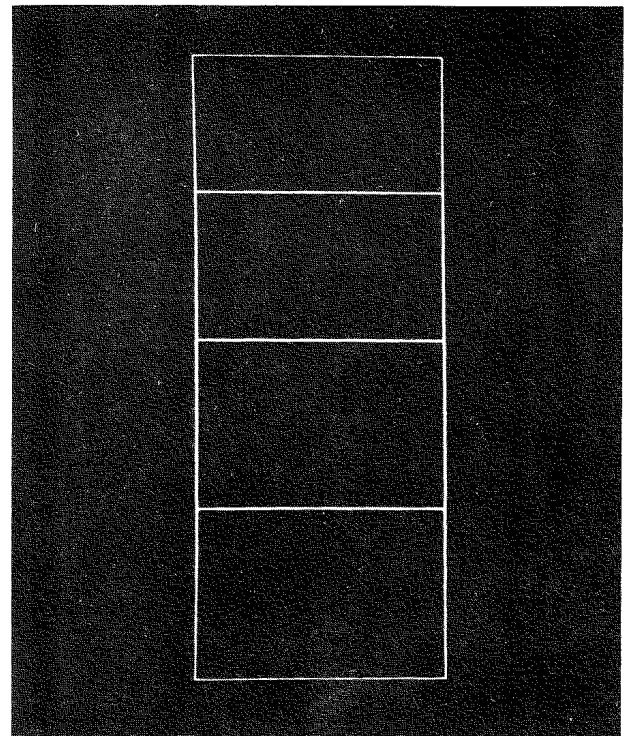
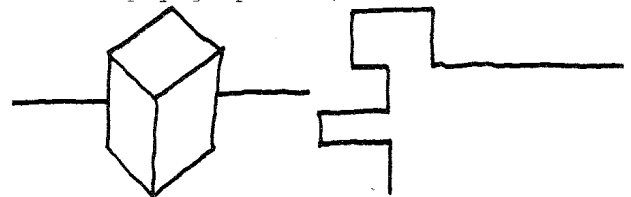


Figure 18.
to do with speed.

Inferring continuities is a good example of where it is necessary to go back to the original raw data because intended continuities are often broken by straightening, horizontalizing and the like. The computation problem is messy as it demands extending each line segment within the bounds of the picture and then making good guesses at intentions of co-lineality. This operation can be greatly reinforced by the knowing of the user and his methods of work (architectural, in this case, rather than simply graphical).



Overtracing, on the other hand, is straightforward because it is very much a function of velocity, sequence and pressure. Two lines drawn side by side, one after the other, very slowly, are probably intended to be two closely spaced parallel lines. Two lines which are drawn rapidly in succession and are almost coincident are probably meant to be a single line, emphasized or corrected. Lines drawn in succession, but one over the other with greater force can be viewed as a form of implicit editing (a good guess would be that the overtracing implied erasure).

Editing

There are two forms of editing: altering your drawing and altering the machine's interpretation of your drawing. Some people might argue that the latter results only from the ineptitude of the machine and a good "sketch recognizer" would not need that sort of editing.

On the contrary, we believe that the editor, over the next two years, will evolve into one of the most important parts of the system, not to facilitate correcting a stupid machine, but to include some features of learning. The editor should ask better and better questions. It should analyse erroneous interpretations as a means of altering its model of the user and of reinforcing its propensity to be a surrogate hand/pencil.

At present, the editor is a modest facility for graphical changes of either kind. It permits adding and subtracting lines and corners, moving segments of the picture (figure 19, time lapse photograph) and explicitly evoking any of the routines included in inference making. It exercises write thru mode, which, in the case of ARDS, can support four or five vectors of modest length. (This limitation is not true for other brands of storage tubes.)

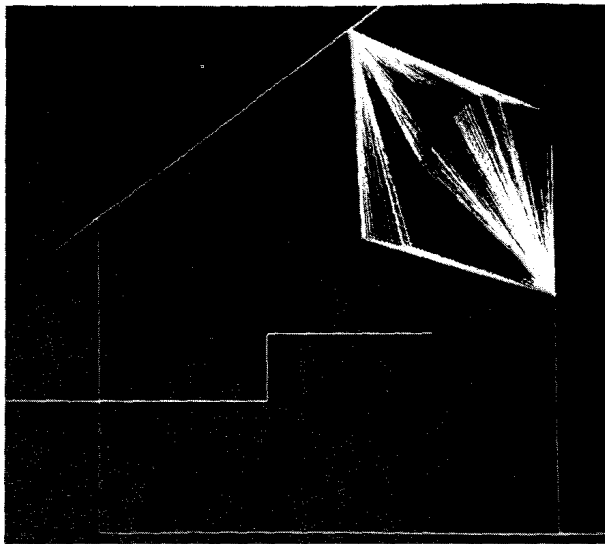


Figure 19.

Communications

All the operations discussed above take place in the Interdata, using only local computing power; they are programmed in Interdata machine language, a 360-like language. Most of the remaining operations require higher level languages and are performed on a remote time-sharing system, MULTICS.

Communications operations are utilities that make the Interdata look like a 2741 typewriter terminal (135 bits per second) or like a stand alone ARDS (1200 bits per second), depending upon which port is available (it prefers the latter). In one case, the compressed data is transmitted in EBCDIC, in the other USASCII: the user and the other operations are unaware of which is being used. At present, the only requirement for the user is that he dial up MULTICS by phone (shortly this will be replaced with an automatic dial up system that will work under computer control).

Projection Recognizer

At MULTICS, the operations no longer take place in "real time", but are at the mercy of the time of day, the number of other users and the load on the system. All programming at MULTICS is done in a MULTICS version of PL/I.

The projection recognizer is fundamentally a branching procedure that seeks to recognize what kind of drawing it is receiving. (Note that this entire problem could be easily circumvented by simply requiring the minor inconvenience of a label, but in keeping with our concept of wanting the user to be no more categorical than with an onlooking human, we feel that the following recognition routines are necessary.)

The first task is to determine whether it is a perspective or an orthographic projection. This branching procedure is relatively simple because it can be handled by supporting or refuting the candidacy of its being a perspective. You can look for a horizon line, points of infinite intersection, or try to establish a picture plane or viewing position. The more complicated the image, the easier it is to gather convincing evidence and make a failsafe decision.

If it is an orthographic projection, the question is whether it is a vertical section, a horizontal section (plan) or an axonometric. The axonometric is the easiest to recognize and, if found, is subsequently treated like a perspective, with the onserver at infinity. Plans and sections are the most difficult to distinguish. They depend heavily upon our knowledge of the real world and associated implications. For example, floors are usually horizontal and walls vertical, stairs

are a give-away and roofs slope for a purpose. There are, nonetheless, cases where even the most experienced architect would have difficulty deciding whether he was looking at a plan, or a section. We do not expect our program to handle these cases. We expect it to ask, and like the editor, to improve.

If the transmitted image is indeed an orthographic projection, the next question is: what is its scale? As in the problem of distinguishing plans from sections, a knowledge of the world must be imbedded in the program in the form of heuristics, rules of thumb. While there are give-aways like figures and trees, most drawings are given scale by a knowledge of the problem at hand or by association with, for example, a winding street seen in plan or a facade in section.

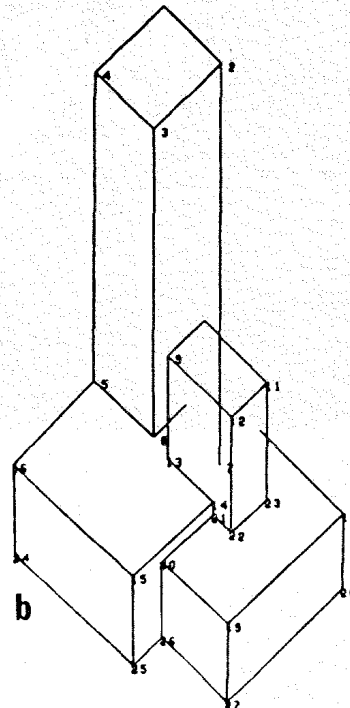
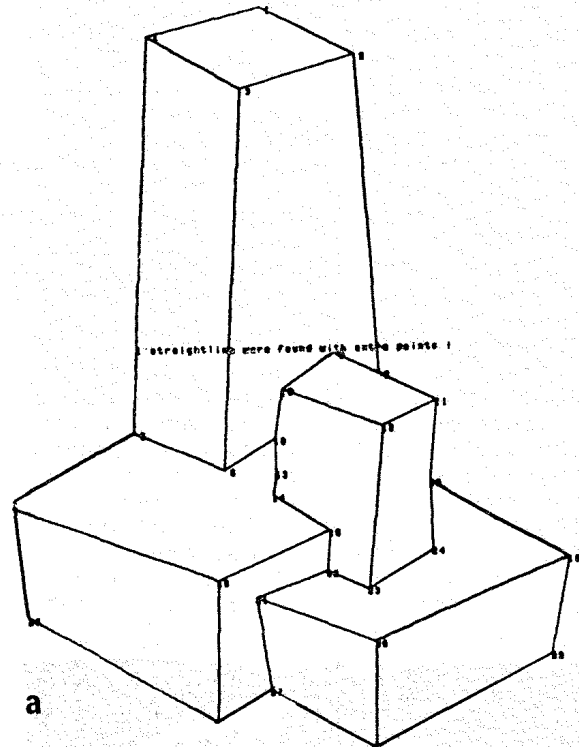
If the picture has been made out to be a perspective the program proceeds directly to a process of mapping into three dimensions.

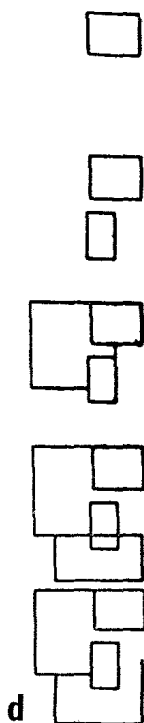
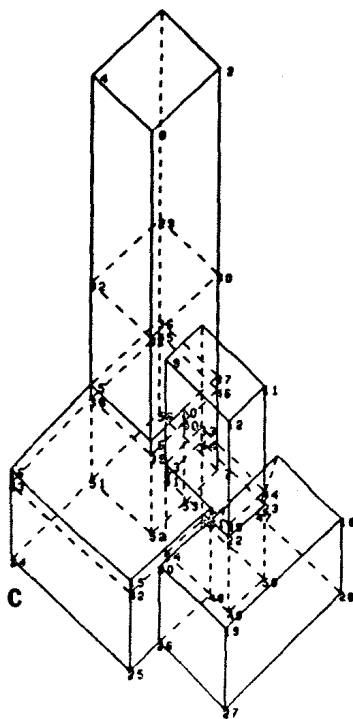
Mapping Perspectives

The strategy for mapping into three dimensions results from the authors' previous experiences with machine vision⁷, and the indiginous character of perspective drawing at the level of a sketch. The problem is to map a two dimensional image into three dimensions, and to handle all the ambiguities that exist for even the human observer. The solution is not deterministic in any sense; guesses and assumptions are necessary. The reader familiar with problems of machine vision should recognize a major difference between a sketched perspective and an image acquired by a device like a vidiosector or a fly spot scanner: that is, in the sketched image, we inherently know the sequence of lines.

The fundamental and well documented program of Guzman⁸ develops evidence for connecting bodies (the initial scene being composed of separate, assumed unrelated regions). Our efforts of mapping into three dimensions postulates the opposite. That is, we assume the scene (ie: the perspective) is one single body in one single plane (which is, of course, impossible). Heuristics are then used to break them apart and separate the regions in terms of their depth in the scene. T joints, diagonals and parallelograms are the main sources of evidence for positioning planes. At present, we assume a flat ground plane.

The following four illustrations (photostat reversals of photographs from the ARDS) describe the four stages of mapping a rather complex scene into three dimensions.





The description was transmitted to MULTICS from the Interdata (a), normalized (b), mapped (c), and described in terms of plans (d). You will notice that all the hidden lines are filled in, including adding points that represent cuts through the different levels (not inherent

in a description of the physical body.) This results in a mishmash of unreadable numbers and dotted lines that are being displayed only for the purposes of debugging. (The observant reader will notice a small error at the bottommost level.)

Mapping Sections

In the case of orthographic projections like plans and sections, mapping into three dimensions is fundamentally a problem of joining; for example connecting a section with a plan, several plans, or several sections. At this writing work has not been started on this problem. We expect this work to be particularly important because when doodling plans and sections, one often describes geometric impossibilities. At the same time, there is no minimum of orthographic views which can unambiguously describe any scene. The result is that: a) many views may be necessary, b) modelling in three dimensions may mean carrying several alternatives, c) heuristics will be necessary to limit the number of these alternatives.

Massaging

Our interest in massaging a three dimensional body results partly from the presence of the Imlac display unit capable of handling this task. It is important because the mapping routines are often performed at the expense of the intended proportions. Therefore, we employ the Imlac as a display device for rotations, translations and the like. It has a singular means of input (to be a light pen, but presently a keyboard) for changing the proportions, but not the topology of a given figure. The Imlac communicates directly with MULTICS, not the Interdata. It manipulates only the three dimensional model (figure 20).

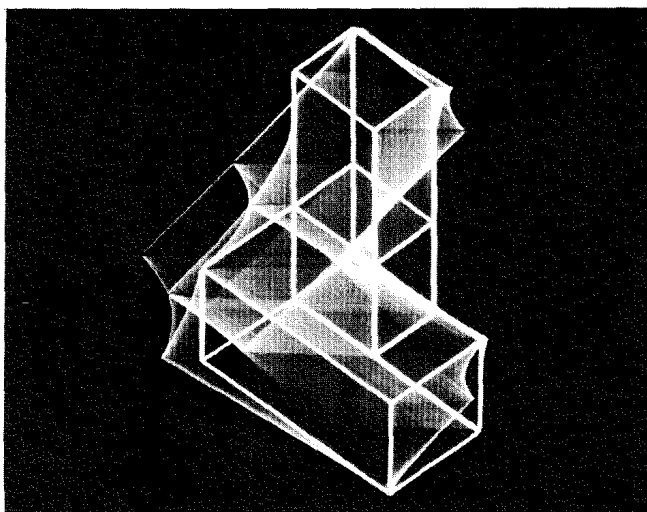


Figure 20.

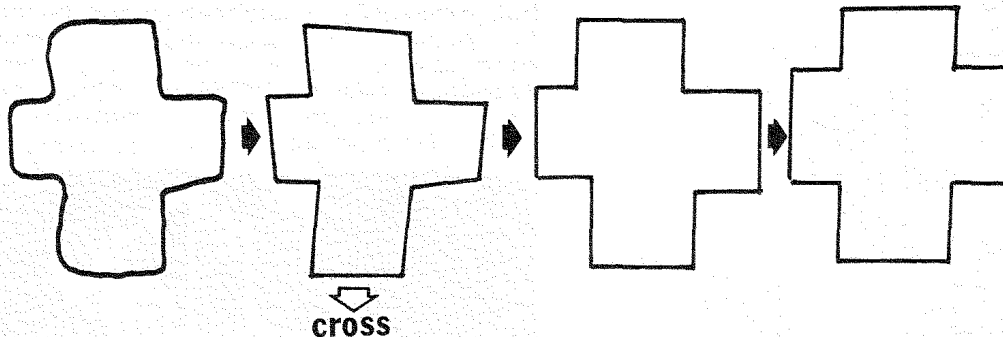
Building in Three Dimensions

Following a successful mapping into three dimensions and perhaps some massaging, routines are evoked at the MULTICS end that compute the best approximation of the figure(s) which can be built within SEEK's repertoire of two inch cubes. The X,Y and Z locations of the centroids are sent back to the Interdata which drives the apparatus to build it (see figure 8).

Shape Recognition

At this time, with graphical transformations in some sense under control, we foresee our major work being in the area of shape recognition. While we are describing the aspect of recognition at the end of Operations, it may result in being an initial step of recognition, one providing evidence to other operations.

For example, the following diagram depicts the present sequence of graphical treatments applied to twelve lines. The fact that the four wings have (in our present scheme of things) no way of becoming the same size, though that was what was probably meant, might suggest that we simply add an inference making feature that looks for the repetition of line lengths (which we could do but which might lead to looking for repetition of shapes, volumes, line spacing and so on ad nauseam).



We suspect that a more appropriate approach would be to look, in this case, for "crossness". The very concept of "cross" furnishes many of the graphical inferences that until now have been handled in some sense brutally.

Comments

Suggestor/Criticizer

So far we have presented operations that perform tasks, through either heuristics or rules, with some confidence that they will eventually arrive at a

single, best of all possible, answer or guess. However, while we have been willing to make erroneous judgements (ie: have programs make errors), look back to original data, try to recover, and even draw benefits from the error, we have not so far viewed the problem as having a multiplicity of "reasonable" interpretations. To meet this deficiency, we propose that two computing entities are necessary in all operations: those that recommend "reasonable candidates", and those that review them. This may mean carrying several interpretations a long way through the process of recognition before they are disqualified.

One major advantage of this bipartition is that routines can be allowed more aggressive interpretations, even wild guesses.

The implementation of this notion of suggestor/criticizer is not a dramatic departure from our present operational model. The present routines will be turned into the suggestors and simply made less timid. Criticizers, meanwhile are new programs that must include both the heuristics for selections and the arrangement of the storage of candidates. We envision this collecting and sorting of interpretations as a tree that gets bigger in the middle and diminishes toward the end, transmitting only a few

candidates (conceivably just one) to MULTICS. It is unclear to us at this time, how much suggesting and criticizing goes on between the local and the satellite computers.

Modelling the User

Gordon Pask⁹ has proposed that we must deal with three levels of models. The first level deals with HUNCH's model of the user ranging between his manner of drawing, (speed, pressure, etc.) and his attitudes toward architecture. This

model is for the purpose of faster and more complete understanding. When a user does what HUNCH predicts he will do (for example by recognizing his propensity for repeating elements), it can shortcircuit a great deal of inference making, or at least, reinforce it. The success of this model can be easily measured as a function of the closeness of fit between the anticipated event and the actual intention. In no sense should such a model be failsafe or infallible. That is the wrong attitude toward the problem. While at the onset such a model would appear passive and thus not exhibit inept behavior, as soon as it became active it would make many errors and be expected to improve from them.

The next more complicated level of modelling is HUNCH's model of the designer's model of it. This is important for inference making because one tends to leave implicit only those issues which one assumes that the other party understands. In effect, this model grows out of a prosperity of matches between the inferred information and the intended information. If HUNCH infers a square and you meant a square, the procedures that led to the assumption are reinforced as a function of the kind of drawing and its context (the most difficult part).

The last level of models may appear overly circuitous. It is HUNCH's model of the user's model of its model of him. We suspect that this level will be particularly important when we approach a more intelligent recognition system, because it is the convergence of this model with the first level (HUNCH's model of the user) that leads to an acquaintance into a partnership.

At present, only modest implementations exist. Beyond the purely graphical inferences from simple methods of the user, we are working on a measure of intentions at a more global scale. "Rapidly formed lines appear to be the designer's context or background, and they represent elements about which he either knows a great deal or almost nothing at all!! When lines are formed at lower velocities, the interaction between the designer's concept and the machine's model of the concept appears to be most revealing. Crookedness appears, and there are uncertain gestures and highly reworked lines: this all may represent (perhaps semantic) dispositions toward a design such as being "concerned about", "sure of", "puzzled by" and so forth.¹⁰

Notes and References

1. HUNCH is being developed by the Architecture Machine Group, established in 1967 by Professors Leon B. Groisser and Nicholas Negroponte of the Department of Architecture, M.I.T. Mr. James Taggart has been the protagonist-programmer, assisted by Miss Cindy O'Connell and Mr. Michael Miller.
2. Computer Aided Participatory Architecture is a proposal to the National Science Foundation that has been reprinted. It is available through The Architecture Machine Group, Room 9-520, MIT, Cambridge, Massachusetts, (\$5.00).
3. Negroponte, N. The Architecture Machine, M.I.T. Press, 1970; Cambridge, Massachusetts, p.19.
4. T.O. Ellis, J.F. Heafner and W.L. Sibley, The Grail Project: an Experiment in Man-Machine Communications, The Rand Corporation, Memorandum RM-5999-ARPA, September, 1969.
5. I.E. Sutherland, "Sketchpad: A Man-Machine Graphical Communications System", AFIPS Proceedings, 1963.
6. T.E. Johnson, "Sketchpad III: A Computer Program for Drawing in Three Dimensions", AFIPS Proceedings, 1963.
7. Machine Vision of Models of the Physical Environment, The Architecture Machine Group, NSF Proposal 1969 (reprints no longer available).
8. Guzman, A., "Decomposition of a Visual Scene into Three-Dimensional Bodies", Automatic Interpretation and Classification of Images, Grasselli (Ed.) pp. 243-276, Academic Press, New York, 1969.
9. Through private communication.
10. N. Negroponte and L.B. Groisser, "The Semantics of Architecture Machine", Architectural Forum (US), October 1970 and Architectural Design (GB), September 1969.