The
Computer
Museum

One Iron Way
Marlboro
Massachusetts
01752

16 January 1984

Cynthia Solomon
Atari Research Laboratory
5 Cambridge Center
Cambridge
Massachusetts

Dear Cynthia

Thank you very much for a fascinating visit last week. I do hope that some form of your work can be incorporated in the exhibit on the computer image - certainly the Logo button box would qualify.

I shall follow up the contacts you and Margaret suggested, but would be most grateful if you could mention this project to your friends and colleagues in California. Two outline proposals for the gallery are enclosed. Do get in touch if you have any ideas or suggestions of further people to contact.

I hope this reaches you before you depart. Have a good trip!

Yours sincerely

Oliver Strimpel
Curator

PS: The address here is Computer Museum, Museum Wharf, 300 Congress Street, Boston MA 02210, tel 426-7190 ext 308.

Gestural Input to Computers through Visual Recognition of Body Silhouettes

Edward F. Hardebeck
Atari Cambridge Research Laboratory
Cambridge, Massachusetts, 02142

Abstract

This paper describes work in progress on using body positions and gestures to communicate with computers. A system has been built that recognizes digitized television images of body silhouettes.

This work is part of the Gesture Project at the Atari Cambridge Research Laboratory ([Minsky 84]). The focus of this project is on the use of gesture and body movement as an interface to a computer system.

Introduction

A major trend in the evolution of computer systems is the increasing ease of interaction with the computer. Nevertheless the communications interface between the user and the computer is still one of the worst bottlenecks in any computer system. The process of translating from the user's conception of what needs doing to what the machine needs to be told in order to do it is one of the most time consuming and error prone parts of computer use.

Many researchers are working to improve this situation by making the machine more responsive to the wishes of the user. At Atari Corporate Research we are exploring the use of gesture, both hand and body movement, to manipulate and control objects in computational environments.

Our goal is to make gestural systems which people can easily learn and use because of the naturalness of body movement and interacting with the physical world.

One kind of gestural input mechanism we are investigating is the use of visual recognition of the position of the body or certain body parts.

A fair amount of work has been done in the area of tracking body position. However, most of these systems require the attachment of some sort of device or marker to the body. We consider the use of any devices attached to the body to be too restrictive for the sort of free interaction we have in mind.

Some popular hardware for body tracking include the electrogoniometer, a cumbersome device which is strapped to the subject and provides measurements of joint angles using potentiometers; the ROPAMS (Remote Object Position Attitude Measurement System) from Polhemus Navigation Science, Inc., which uses polarized magnetic fields to sense the position of a small antenna; and the Selspot system which optically senses infrared Light Emitting Diodes placed on the subject. There are several other similiar optical systems in use.

These devices have frequently been used to gather medical data, and analyze the performance of athletes. There are only a few systems that use body position for the control of computational events. One of these is the "Put-That-There" system of the MIT Architecture Machine Group (ArchMach) [Bolt 80] which uses the Polhemus system to sense the position of a single pointing finger. Another system under construction at ArchMach [Ginsberg 83] uses hardware similiar to Selspot to sense body position. The data are then to be used to drive a graphical representation of a human figure, known as a "graphical marionette" [Maxwell 83]. The intended application of this system is in the generation of realistic figure animation.

We have built a system which uses an inexpensive television camera and video digitizer to look at a person in silhouette. A recognition program finds the positions of the various parts of the body.

The Hardware

A general description of the system is shown in Figure 1.

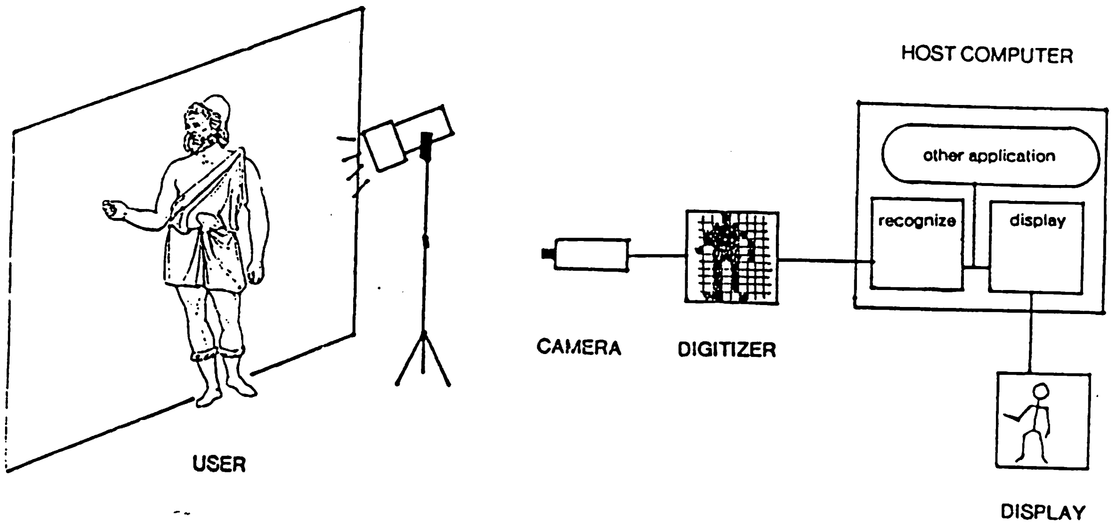


Figure 1: *Overview of the System*

The user stands before a light colored wall which is lit from the side by stage lights. A television camera receives the image and passes it to a digitizer. The digitizer produces a binary image, all points under a threshold are black, those above are white, thus a silhouette of the person is passed to the computer. This application does not require a high resolution digitizer. Currently we are using a Computer Station Dithertizer II (Computer Stations, Inc., St. Louise, Mo.) and an Apple II microcomputer for digitizing and transmitting to the host computer. This digitizer has a resolution of 280 horizontal by 192 vertical points. The image is then parsed by a recognition program on the host computer (a Symbolics Lisp Machine) and a representation of the body is built, consisting of a data structure which contains the position of the body (values of various joint angles and so forth).

This representation can then be used to control computational processes. The representation is usually displayed by a graphics routine to give feedback to the user or to produce animation. Various display formats could be used, from the strictly anthropomorphic to ones where body position controls arbitrary graphical events. In the latter case a training period might be necessary during which the user would learn to control this abstract "body".

We have implemented two display routines. The first displays a stick figure in a position corresponding to that of the recognized figure. The second displays a flowering plant in which the positions of leaves correspond to that of the arms and legs and the flower to the head. This display could be used to produce "anthropomorphic animation" similiar to the "Waltz of the Flowers" sequence in Disney's *Fantasia* where flowers dance like people.

The Recognition Program

General visual recognition of scenes and objects by computer is an unsolved problem. Most practical vision systems rely on some form of top down constraints, such as knowledge of what sorts of objects are to be found in the scene. This program also uses top down information. It "knows" that the scene contains a human figure, all it has to do is determine the location of the various parts of the body. (If there is not a human figure in the scene the program will do its best to hallucinate one.)

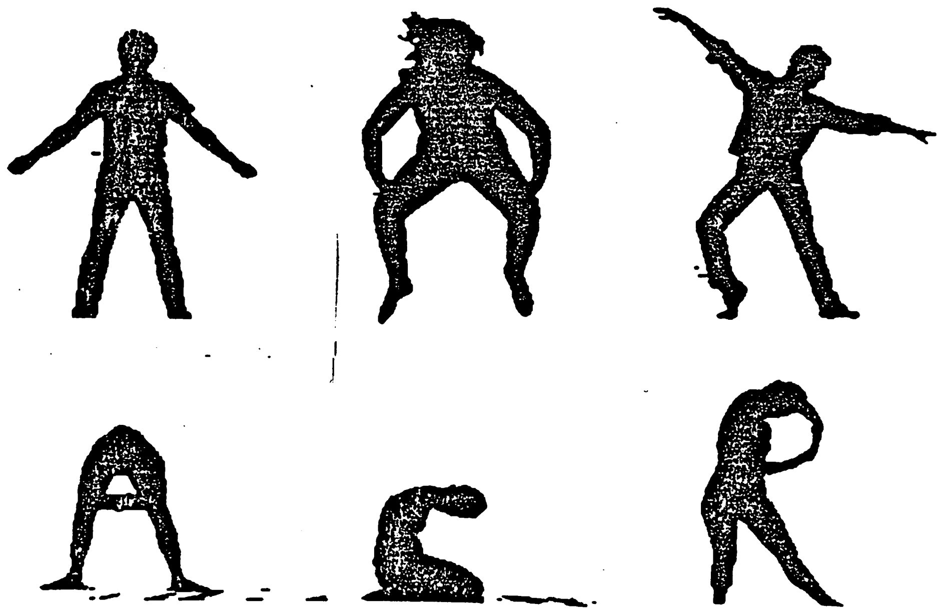


Figure 2: *Digitized Images of Human Figures*

The recognizer receives a low resolution binary image of a person (Figure 2), and possibly some history information of previous positions. The use of a silhouette simplifies the recognition process, since there is no need to separate the subject from the background. The process of finding edges and occluding contours is bypased. Further advances in the study of early visual processing and the speed of current methods ([Marr 82] [Hildreth 80]), would allow the program to use normally lit subjects rather than silhouettes

The major drawback of using silhouettes is that a body part which coincides with another part is occluded. This could be mitigated by using two cameras, although this increases the recognition time by requiring the analysis of two images. History information is also useful in locating a non-visible body part. For instance, information about where it was last seen and in which direction it was moving could be extrapolated to track a body part until it reappears.

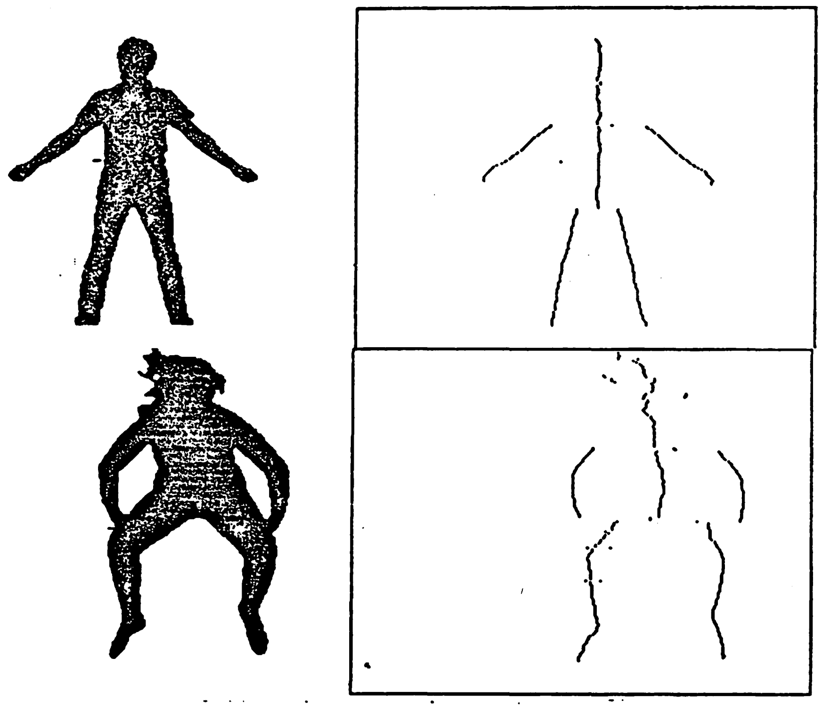


Figure 3: *Axes of the Body*

The first stage of recognition uses some low level routines to separate the image into manageable parts. The silhouette is represented as an array of ones and zeros. This array is scanned horizontally and for each sequence of ones the midpoint and width are collected. These sequences correspond to cross sections of the body parts. For nearly vertical parts these midpoints lie along a "skeleton" of the figure (Figure 3). For more horizontal parts a vertical scanning phase is neccessary; the resulting points are merged into the representation later. The midpoints thus found are close to the axes of limbs and other body parts. The true axes of these parts could be found by again scanning the image at each midpoint in the direction normal to the axis the point. As described in [Nevatia 73] this process will converge and eventually give the correct central axis of the parts of the figure. In practice vertical and horizontal scanning seem to give a close enough approximation for recognition.

Next, the midpoints are grouped into segments of points which are closely connected

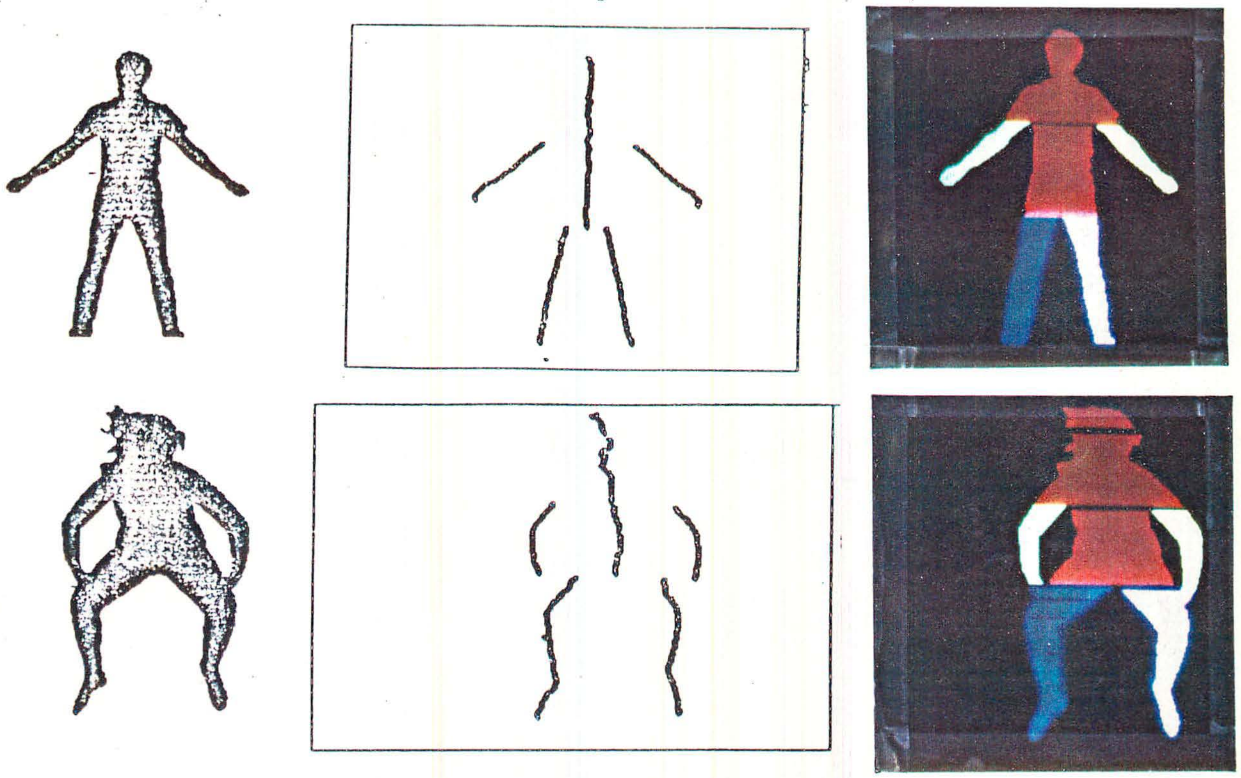


Figure 4: *Segments of the Body*

(Figure 4). These segments tend to correspond to body parts. Very small segments are thrown away. These are usually caused by noise, surface irregularities of clothes or hair, or scanning of parts in the direction parallel to their major axis. A line-finding algorithm is then run on the points in each segment, further decomposing it into straight pieces (Figure 5). Each of these pieces may correspond to an individual bone or a group of bones such as the spine or a limb in extension.

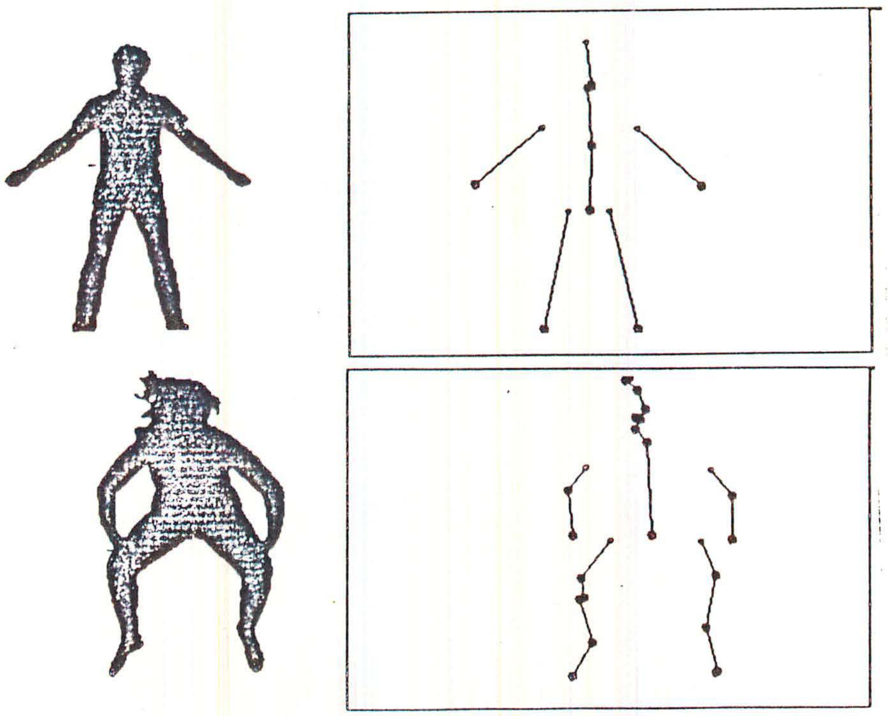


Figure 5: *Lines in the Body*

Once the body has been segmented, the task is to establish the correspondence between individual segments and their features and the parts of a human body. This is the job of the "body parser", a program analagous to a grammatical parser. It examines the segments and their relationships in the light of knowledge about the structure of the human body.

As of this writing we have implemented a simple body parser and have sketched out a "smarter" one. The program "knows" certain facts of basic anatomy such as "people generally have two legs", "the knee joint is generally near the middle of the leg", or "the thigh bone's connected to the hip bone". In our simple parser the latter fact is represented something like this code fragment (in Lisp):

```
(define possible-spine
        (find-closest-part (proximal-end possible-leg)))
```

Of course the assumption that such a part is a spine must be checked using other knowledge. It could, for instance, be a hand placed on the hip.

A more powerful recognizer can be built, based on heuristic programming methods. A heuristic is a piece of knowledge similiar to a "rule of thumb". It may be applicable under certain conditions in a particular domain. (For more information on heuristics see Chapter 4 of [Davis,Lenat 82]). Our piece of knowledge can be represented as an "anatomical heuristic" as follows:

```
(def-anatomical-heuristic
   the-thigh-bones-connected-to-the-spine        ;by the pelvis
   (possible-thigh possible-spine)
 (when (quite-close (proximal-end possible-thigh)
                    (base possible-spine))
   (believe possible-thigh 'thigh 1.)
   (believe possible-spine 'spine 1.)))
```

This heuristic checks the relationship between a piece which may be the thigh and a piece which may be the spine. If they have the right spatial relationship it asserts that the possible thigh may actually be a thigh (giving a numerical value for how much it "believes" it). There can be many such small heuristics all contributing their various beliefs about the anatomical pieces. The pieces can then be sorted according to which body part they are most likely to be.

The recognizer produces a labelling of the segments of the body (Figure 6). Each segment is then further parsed into joint positions (knees, elbows) and parts of limbs. The result of the recognition phase is a data structure which is an internal model of the position of the body. It contains numerical values for the state of each part and joint in the body. Currently the structure consists of a list of the 2-dimensional coordinates of each joint and distal end.

This data structure is then given to other programs which are controlled by the

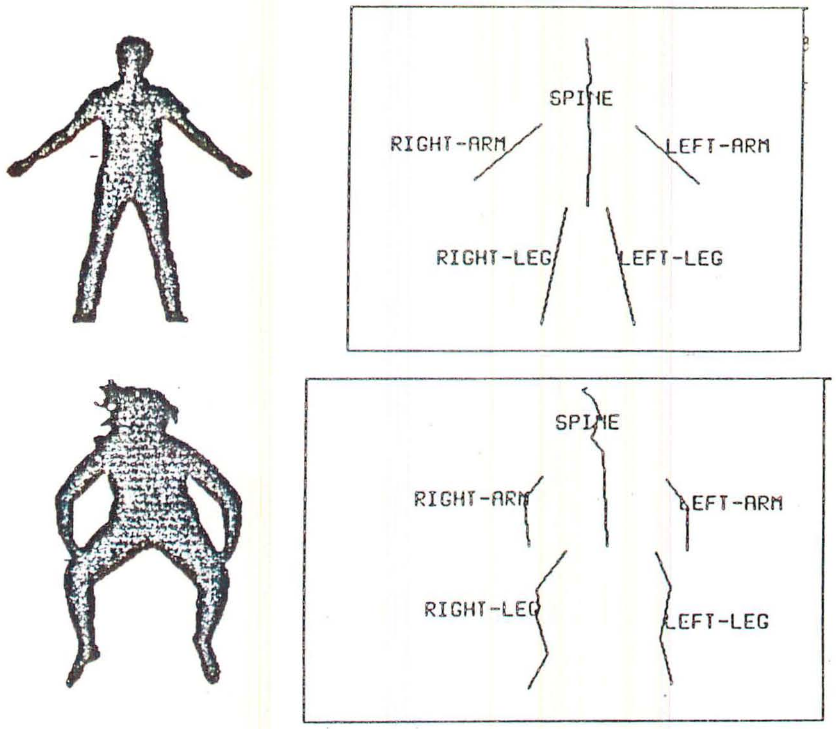


Figure 6: *Parsing of the Segments of a Body*

motions of the user. Currently it is passed to a display routine similiar to Maxwell's "graphical marionette" [Maxwell 83]. A simple routine draws a stick figure with line segments whose endpoints lie at each of the coordinates. A more interesting routine creates a flower (Figure 7) by drawing waving leaves (using cubic splines) that correspond to the limbs, a stem that corresponds to the spine, and a flower at the head position.

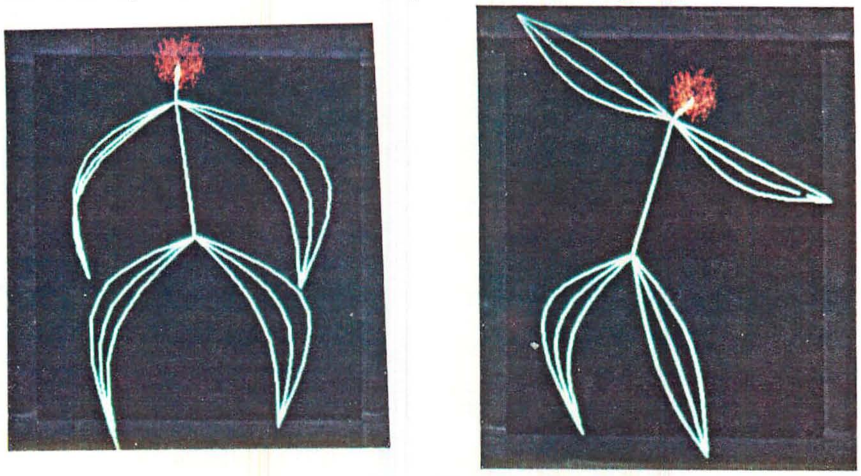


Figure 7: *The Flower Display*

Conclusions and Future Plans

We have built a system which can recognize several body parts from silhouettes of people in upright positions with limbs clearly visible. We plan to implement a better recognition program which will recognize less constrained figures.

Although experimental, the current system has reasonable performance with one processing cycle taking on the order of 10 seconds. We feel that through several improvements we may be able to achieve real time performance. These include the use of a faster computer (which we have available); the use of a digitizer which is on the data bus of the computer rather than communicating through an interface; and the optimization of our algorithms. One of the most promising optimizations is to pipeline the initial scanning for axis points. This simple but time consuming task could be performed by a small microcomputer or by hardware in the digitizer.

We have written two display programs which can take a list of joint positions and draw a figure on the display screen. We plan to build more display types. Some possible displays are:

- Animals both real and fantastic·
- Displays which exaggerate or otherwise distort shape or motions
- Displays in which the computer improvises along with your movements

We plan to explore possible applications of gestural interfaces. Some examples of such applications are animation, choreography, teleoperators, and whole body video games.

We would like to be able to recognize sequences of positions as gestures such as waving, beckoning, pushing, throwing, even "reeling and writhing and fainting in coils". Such gestures could be used to manipulate and control objects or creatures in artificial worlds.

Acknowledgements

I would like to thank those members of Atari Corporate Research who supported and encouraged this work, especially Cynthia Solomon and James Davis, other gesticulators and AI hackers Margaret Minsky, Dan Huttenlocher, Danny Hillis, Ken Haase, and Gary Drescher; all of whom read drafts of this paper and provided many helpful comments. Also thanks to those who contributed their bodies to science, Michael Grandfield, Margaret Minsky, and Jeannine Mosely.

References

[Bolt 80] Bolt, R. A., "'Put-That-There': Voice and Gesture at the Graphics Interface", Proc. Siggraph '80, *Computer Graphics* 14, no. 3, July 1980.

[Davis, Lenat 82] Davis, Randall, and Douglas B. Lenat, *Knowledge-Based Systems in Artificial Intelligence*, McGraw-Hill, New York, 1982.

[Ginsberg 83] Ginsberg, Carol M., "Human Body Motion as Input to an Animated Graphical Display", Masters Thesis, Department of Electrical Engineering and Computer Science, Massachusettes Institute of

Technology, 1983.

[Hildreth 80] Hildreth, Ellen C., "Implementation of a Theory of Edge Detection", AI-TR-579, Artificial Intelligence Laboratory, Massacusettes Institute of Technology, 1980.

[Marr 82] Marr, David, *Vision*, W. H. Freeman, San Francisco, 1982

[Minsky 84] Minsky, Margaret, "Manipulating Simulated Objects with Real-world Gestures using a Force and Position Sensitive Screen", Paper submitted to Siggraph '84.

[Maxwell 83] Maxwell, Delle R.,"Graphical Marionette: A Modern Day Pinocchio", Masters Thesis, Architecture Machine Group, Massachusettes Institute of Technology, 1983

[Nevatia 73] Nevatia, Ramakant and Thomas O. Binford, "Structured Descriptions of Complex Objects", *Proceedings of the Third International Joint Conference on Artificial Intelligence*, 1973.

Manipulating Simulated Objects with Real-world Gestures using a Force and Position Sensitive Screen

Margaret R. Minsky
Atari Cambridge Research Laboratory
Cambridge, Massachusetts

Abstract

A flexible interface to computing environments can be provided by gestural input. We describe a prototype system that recognizes some types of single-finger gestures and uses these gestures to manipulate displayed objects. An experimental gesture input device yields information about single finger gestures in terms of position, pressure, and shear forces on a screen. The gestures are classified by a "gesture parser" and used to control actions in a fingerpainting program, an interactive computing system designed for young children, and an interactive digital logic simulation.

Manipulating Simulated Objects with Real-world Gestures using a Force and Position Sensitive Screen

1. Introduction

We want to create worlds within the computer that can be manipulated in a concrete natural way using gesture as the mode of interaction. The effect is intended to have a quality of "telepresence" in the sense that, to the user, the distinction between real and simulated physical objects displayed on a screen can be blurred by letting the user touch, poke, and move the objects around with finger motions.

One goal of this research is to make a natural general purpose interface which feels physical. Another goal is to extend some ideas from the Logo pedagogical culture - where young children learn to program and control computing environments [Papert 80] - to gestural and dynamic visual representations of programming-like activites.

How could we introduce programming ideas to very young children? They already know how to accomplish goals by using motions and gestures. So we speculate, it would be easier for them to learn new things if we can give them the effect of literally *handling* somewhat abstract objects in our displayed worlds. For this we need to find simple languages of gesture that can be learned mostly by exploration, and to find visual representations that can be manipulated and programmed by these "gesture languages".

The "Put-That-There" project at the MIT Architecture Machine Group [Bolt 80] has some goals and techniques in common with this research. We also share some goals with the "visual programming" research community.

We wanted multiple sources of gesture information including position and configuration of the hand, velocity, and acceleration to experiment with hand gestures. Our first step was to build an experimental input device by mounting a transparent touch-sensitive screen in a force-sensing frame. This yields information about single finger gestures in terms of position, pressure and shear forces on the screen. Thus our system can measure the position of a touch, and the direction and intensity of the force being applied.

Sections 2, 3 and 4 of this paper describe environments that we have built that are controlled through this kind of gesture input, and our gesture classification. Section 5 decribes the hardware and signal processing we use to recognize these gestures. Section 6 discusses the future directions of this work.

2. Fingerpaint: A First Gesture Environment

To explore the issues involved in this kind of gestural input, we first built a fingerpaint program. The program tracks the motion of a finger (or stylus) on the screen and paints wherever the finger moves. This application makes essential use of the finger's pressure as well as its location. It also uses the shear-force information to smooth the interpretation of the gesture information.

The user's finger squooshes a blob of paint onto the screen.

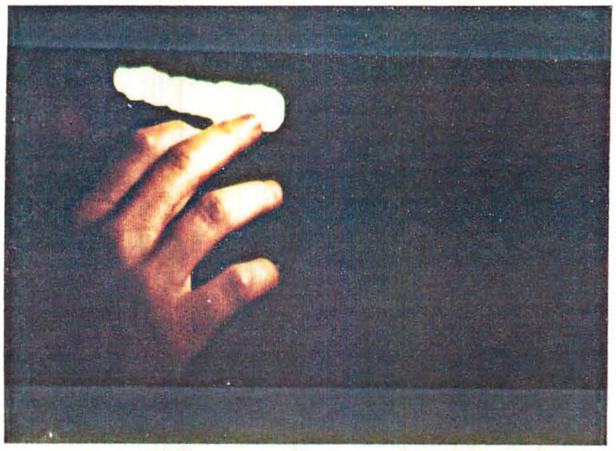


Figure: Fingerpaint

If the user presses harder, he gets a bigger blob of paint.

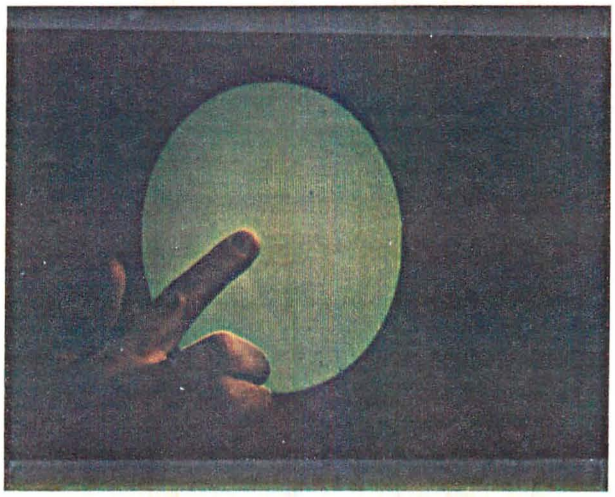


Figure: Fingerpaint with Varying Pressure

The user can choose from several paint colors, and can also paint with simulated spray paint.

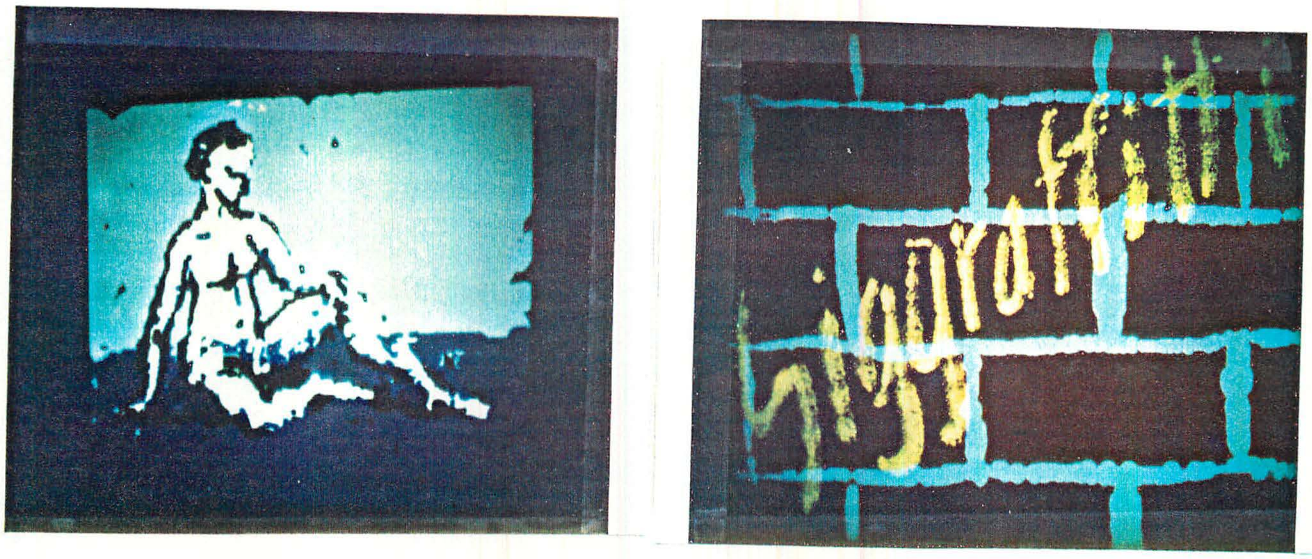


Figure: Fingerpainting

In one version of this program, brush "pictures" can be picked up and stamped in other places on the screen.

Directions for a Gesture Paint Program

We would like to improve fingerpaint in the direction of making a painting system that allows more artistic control and remains sensually satisfying. At the same time we want to avoid making the system too complex for young beginners. We plan to implement a "blend" gesture, a set of paint pots out of which to choose colors with the fingers, and some brushstrokes which depend on the force contour of the painting gesture.

The idea of magnification proportional to pressure used in the paint program suggests use of pressure to scale objects in other environments.

3. Parsing Gestures for Manipulating Simulated Objects

The paint program follows the finger and implicitly interprets gestures to spread paint on the screen. For applications in which discrete, previously defined objects are to be manipulated using gestures, we need more complex gesture recognition. We want the user to be able to indicate, by gestures, different actions to perform on objects. The process of recognizing these gestures can be though of as parsing the gestures of a "gesture language".

Our gesture parser recognizes the initiation of a new gesture (just touching the screen after lifting off), then dynamically assigns to it a gesture type. It can recognize three gesture types: the "selection" gesture, the "move" gesture that consists of motion along an approximate line, and the "path" gesture that moves along a path with inflections. We are planning to introduce recognition of a gesture that selects an area of the screen. These gesture types, along with details of their state (particular trajectory, nearest object, pressure, pressure-time contour, shear direction, and so forth) are used by the system to respond to the user's motions.

4. Soft Implementations of some Existing Visually Oriented Systems

To support our experimentation, we built a fairly general system to display the 2-D objects that are manipulated by gestures.

The following sections describe environments built from these components (gesture parser and 2-D object system), and some anecdotal findings.

4.1 Button Box

The gesture system Button Box was inspired by some experiments done by Radia Perlman around 1976 with special terminals (called Button Box and Slot Machine) built for preliterate children [Perlman 74,76]. Perlman's Slot Machine is a plexiglass bar with slots to put cards in. Each card represents a program command, for example, a Logo turtle command. A child writes a program by putting the cards in the slots in the order they want, then pushing a big red button at the end of the bar. Each card in sequence is selected by the progam counter (a light bulb lights up at the selected card) and that card's command is run. This provides a concrete model of computation and procedure. With various kinds of jump and iteration cards, kids use this physical equipment to learn about control structures and debugging.

The gesture system version is more flexible than the original, specially constructed Slot Machine, because it can be modified and reconfigured by software. The current implementation makes use of some force and gesture configuration information. It can be viewed as "work in progress" towards making models of computation that are particularly suited to having their pieces picked up, tapped upon, tossed about, and smudged by finger gestures.

Pictures of buttons that control various actions appear on the screen. In our example domain, the buttons are commands to a Logo-style turtle [Abelson 81]. For example, one button is the **FD** (**FORWARD**) command, another is the **RT** (**RIGHT TURN**) command. If the user taps a button rather hard (think of hitting or pressing a mechanical button), the button "does its thing". Whatever action the button represents happens. If the **FD** button is tapped, the display turtle moves forward.

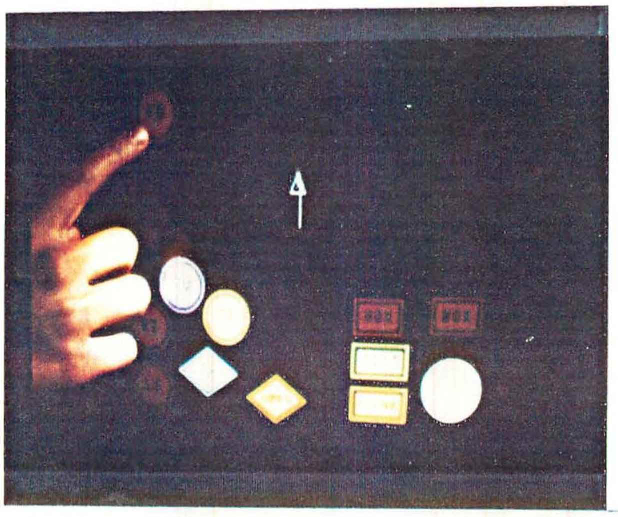


Figure: Forward

If the user selects a button by applying fairly constant pressure to it for a longer time than a "tap" gesture, the gesture is interpreted as a desire to move the selected button on the screen. The button follows the finger until the finger lifts off the screen, and the button remains in its new position.

This allows the user to organize the buttons in any way that makes sense to him, for example, the user may place buttons in a line in the order in which they should be tapped to make the turtle draw something.

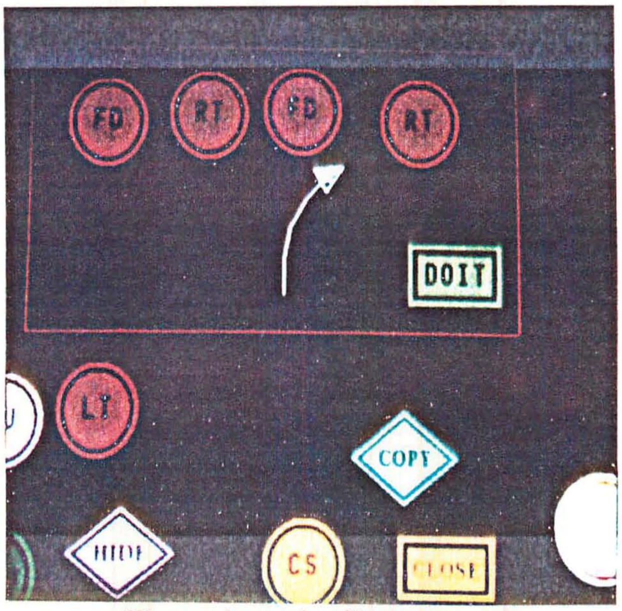


Figure: Arranging Buttons

Some of these buttons control rather concrete actions such as moving the turtle or producing a beep sound. Other buttons represent more abstract concepts, for example, the PU/PD button represents the state of the turtle's drawing pen. When the PU/PD button is tapped it changes the state of the turtle's pen, and it also changes its own label.

There are also buttons which operate on the other buttons. The **COPY** button can be moved to overlap any other button, and then tapped to produce a copy of the overlapped button.

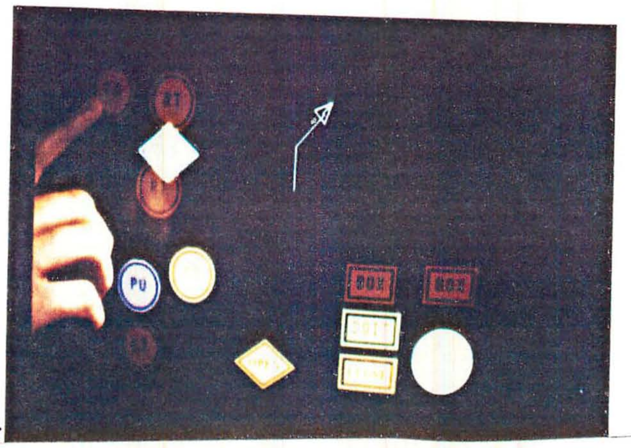


Figure: A Button being Copied

Some concepts in programming are available in the button box world. The environment lends itself to thinking about the visual organization of actions. In our anecdotal studies of non-programmers using the button box, most of our subjects produced a library of copies of turtle commands and arranged them systematically on the screen. They then chose from the library the buttons that allowed them to control the turtle in a desired way and arranged them at some favored spot on the screen.

There are mechanisms for explicitly creating simple procedures. At this time, only unconditionally ordered sequences of action represented by sequences of button pushes are available; we are working on representations of conditionals and variables. The user can specify a sequence of buttons to be grouped into a procedure.

We have experimented with two ways of gathering buttons into procedures: boxes and magic paint.

The first method uses boxes. The **BOX** button, when tapped, turns into a box. The box is stretchy and its corners can be moved, so it can be expanded to any size, and placed around a group of buttons (or the user can move buttons into the box). There are buttons which, when tapped, make the system "tap" every button in the box in sequence.

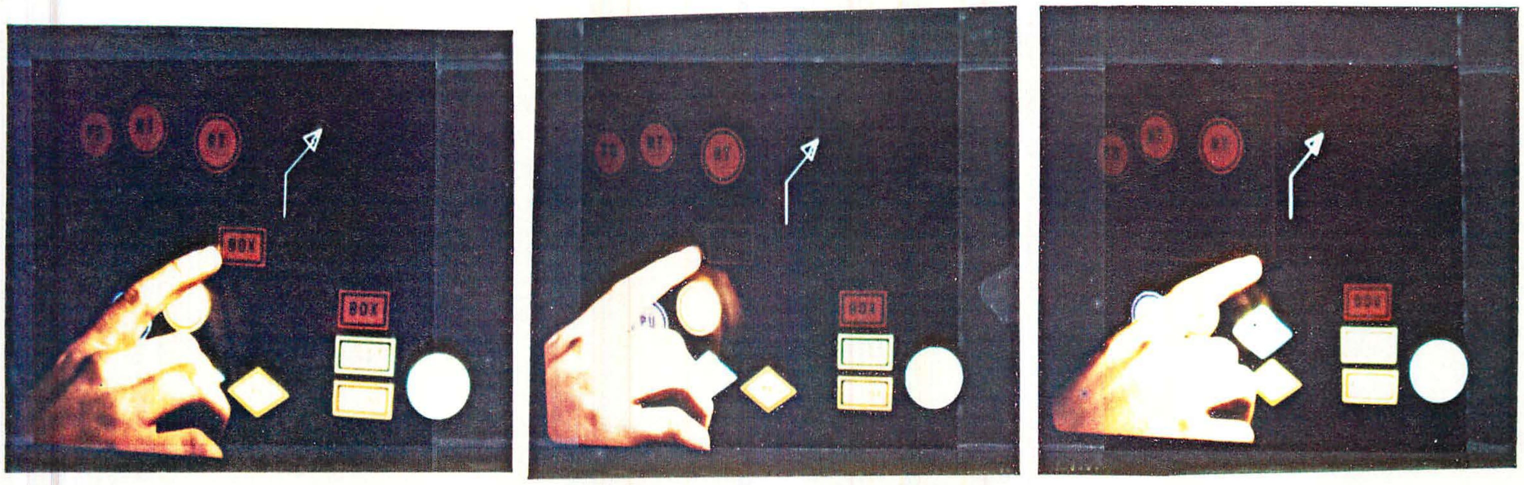


Figure: Making a Box and Using it to Group Buttons

The second method uses "magic paint". Magic paint is a genie button. As the user moves it, it paints. The user uses it to paint over a sequence of buttons. The path created shows the sequence in which buttons should be pushed. When the end of the paint path is tapped, the system "goes along" the path, tapping each button in sequence.

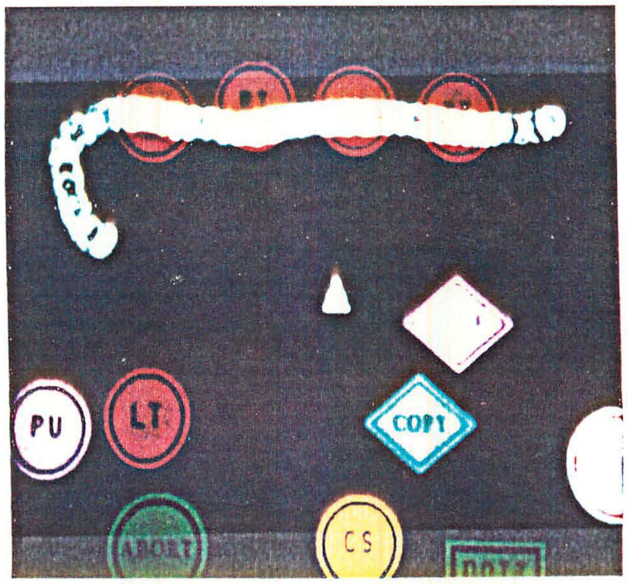


Figure: Using Magic Paint to Group Buttons

The user can group buttons with either method and have the system "push its own buttons". The user can also tell the system to create a new button from this grouping. The `CLOSE` button closes up a box and makes a new button. The new button becomes part of the button box world with a status equal to any other button. The new button appears on the screen and can be moved and can be tapped like any other. Thus it becomes a subroutine.

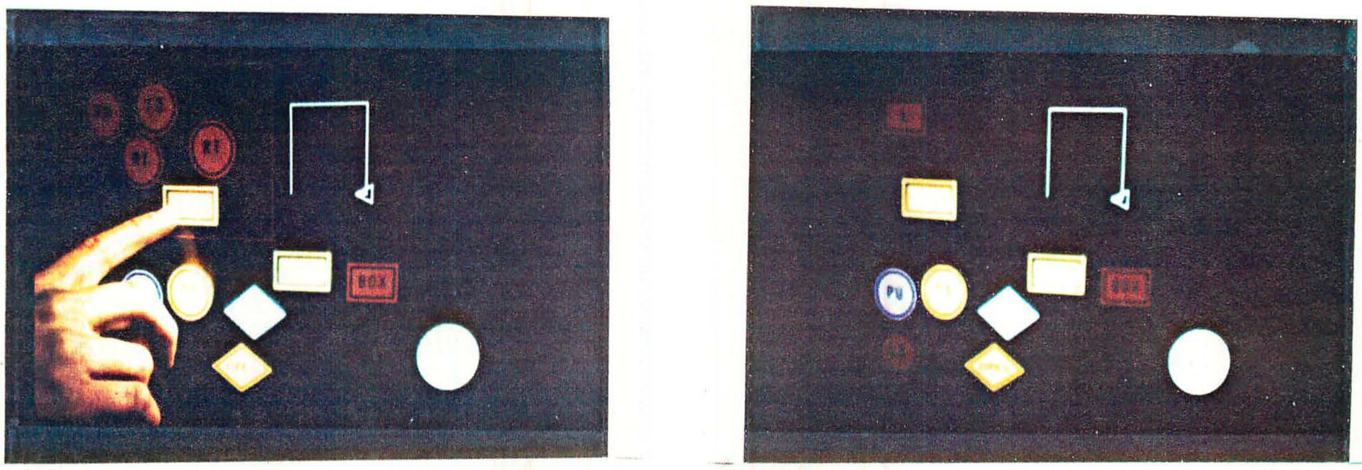


Figure: Creating a New Button by Closing a Box

Most of our experiments so far have used the box metaphor. We plan to develop gesture semantics for magic paint, which seems more promising because the paint path makes the order of button pushes more explicit than the box grouping. It feels more "gestural" to program by drawing a swooping path.

4.2 Logic Design - Rocky's Boots

We have applied the same set of gestures to make a smooth interface to another environment. A graphic logic-design system based on Warren Robinett's program, Rocky's Boots [Robinett 82].

Gates, wires, logic inputs, and outputs are the objects in this environment. The user moves them around in the same way as buttons. They are always "doing their thing". They connect to each other when an input is brought close to an output. The user can cut them apart by making a gesture while holding a knife.

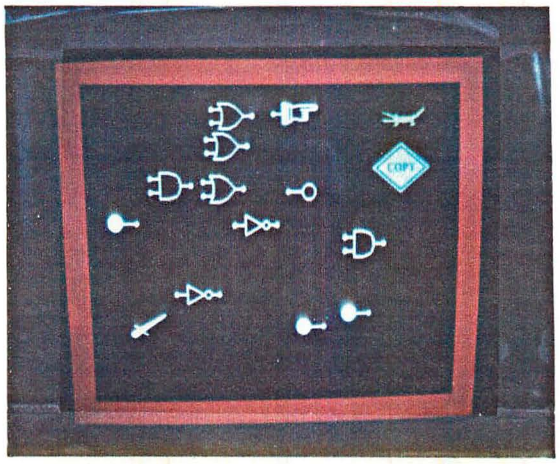


Figure: Logic objects: Gates, HI input, clock input(blurred), output light

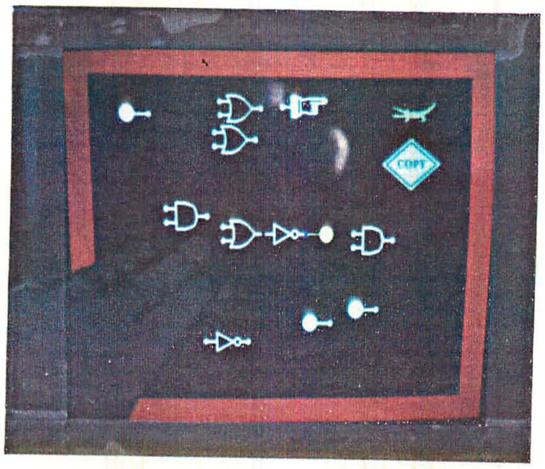


Figure: A Circuit with a Connection being Put In

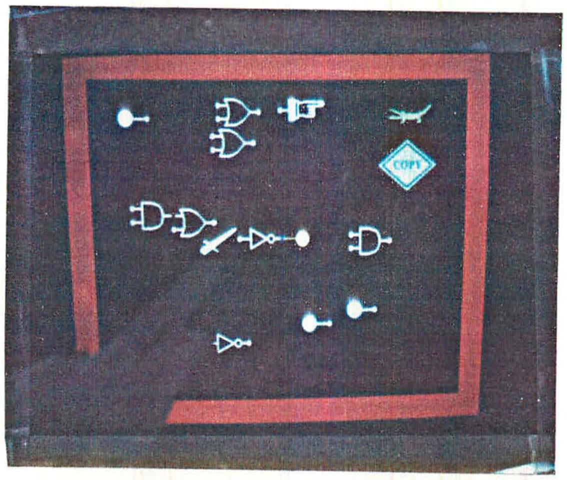


Figure: The Knife being used to Remove a Component

Since these logic components are objects, like buttons, we can operate on them the same ways we can with buttons (e.g. copying with the COPY button).

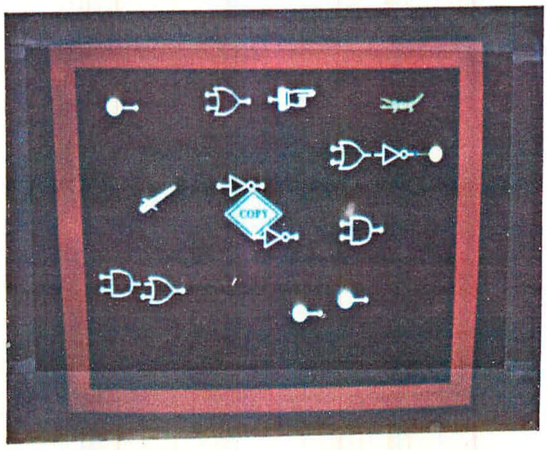


Figure: Copying a Logic Object

There are actions we plan to implement in the logic world, by creating buttons to perform new actions on the objects or in some cases by recognizing new gestures. For example, we could get rid of the need for the knife object by recognizing a "cutting" gesture. We haven't yet defined mechanisms for creating new logic elements from combinations of existing ones but these are the kinds of extensions that can be made with the current gesture repertoire.

4.3 Rooms

All of our gesture-controlled environments: button box programming, interactive logic simulation, and a rudimentary Colorforms^{tm}-like environment, have been combined in an information environment we call Rooms.

Rooms is an extension of visual representation for adventure game maps and other visual information designed by Warren Robinett [Robinett 79] [Robinett 82]. There are rooms which contain objects. The rooms connect to each other through doorways. In our implementation, each room takes up the whole screen, doors are at the edge of the screen. As the user's finger moves through a doorway, the adjacent room appears on the screen, filled with whatever objects it contains. The user can drag any object (button, logic gate, etc) through a doorway.

Our environment starts with a room containing the button box environment, a room containing the logic objects, a room containing colorforms, and a miscellaneous room with a few miscellaneous buttons in it.

Colorforms is a more free-form environment consisting of colorform-like shapes that can be moved around and stamped with finger gestures to create pictures. Our shapes are a face, eyes, mouth, etc which can be grabbed and moved around.

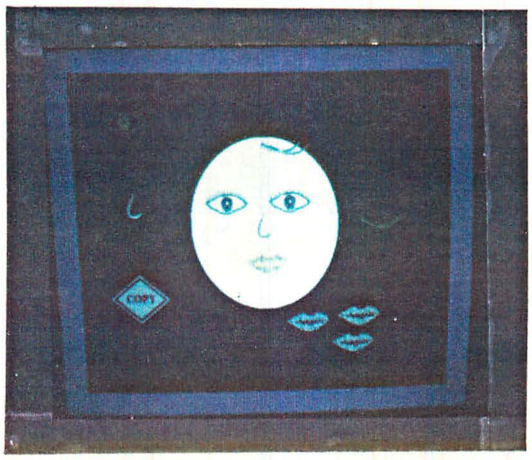


Figure: Colorforms Room

4.4 Combining Environments

One of the tenets of developing good gesture interfaces is that the objects being manipulated by gestures should act like possible physical objects in their reactions to the user's gestures. When the objects are brought near each other they should interact in plausible ways. We have seen that the **COPY** button can copy logic elements. We introduced logic elements that can act on buttons. A special output, the **HAND**, taps a button when it is clocked.

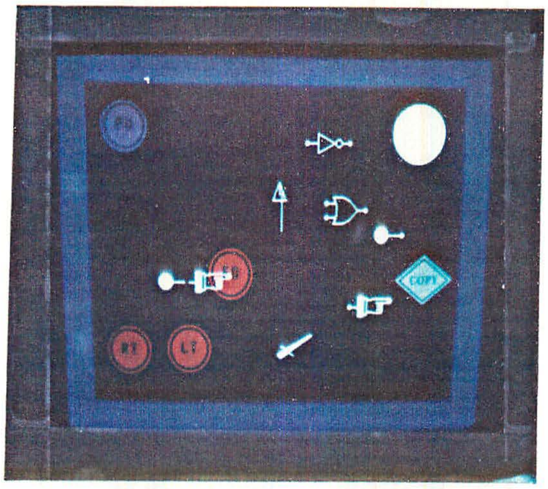


Figure: HAND Tapping the FD Button

The circuit in the figure is an example of parallel processing invented by a user who was experimenting with the interface between the Button World and the Logic World.

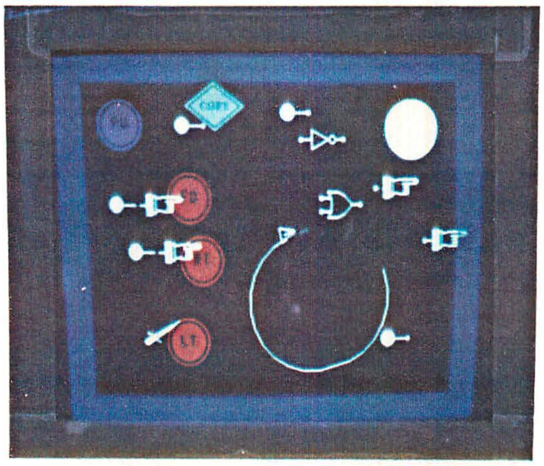


Figure: A "Program" that Draws a Circle

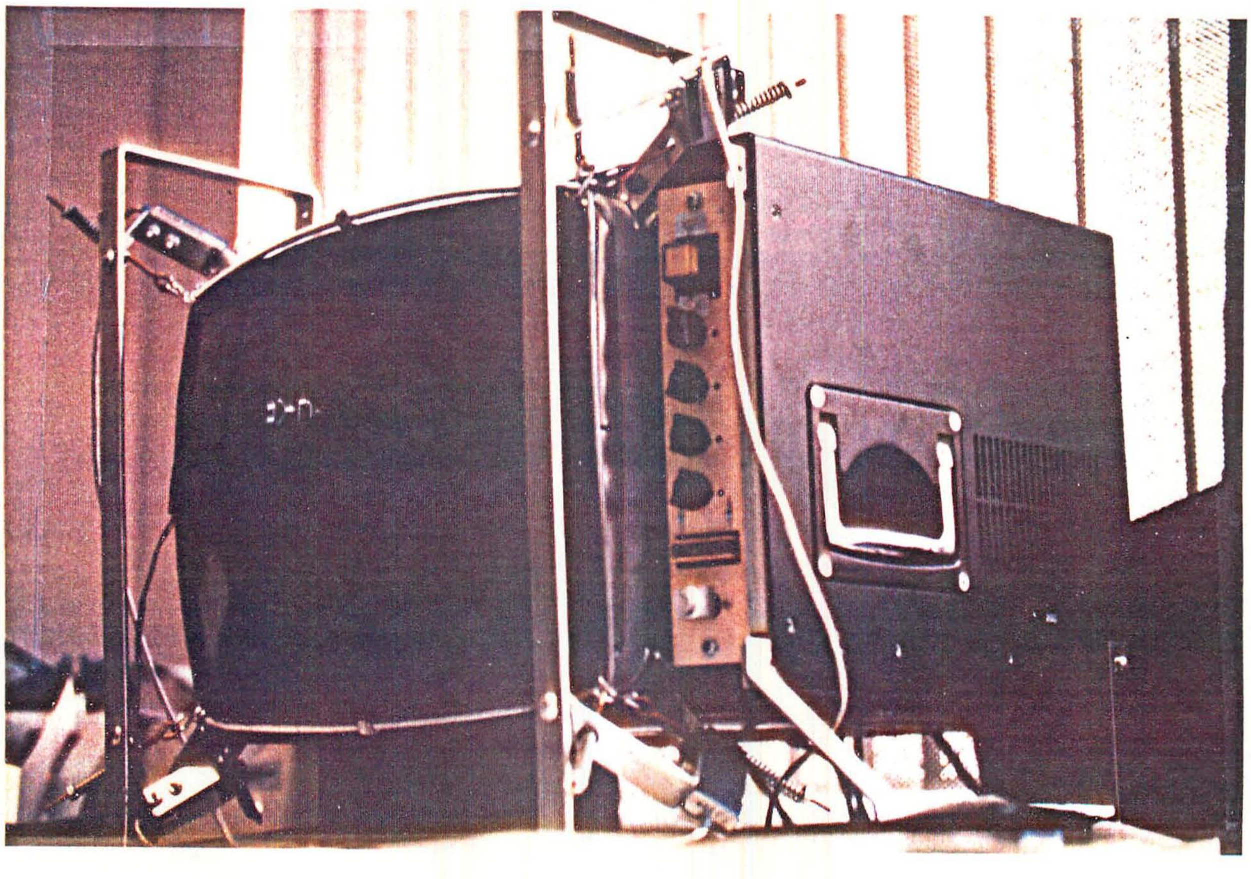
Amazing! Or is it? Nobody is amazed when a real object, like a teapot, can be stacked on top of another kind of real object, for example, a table. However, programs can hardly ever fit together meaningfully, much less smoothly. The object nature of these programming-like environments, and their necessary analogy to physical objects deriving from the gesture interface, has allowed this smooth combination to happen.

Reflecting on our box and magic paint programming metaphors, we can see that in these worlds, *procedures* are *things* which embody processes in their behavior. Here, the processes are represented by the paths of the user's gestures as he constructs a configuration of objects on the screen.

5. Gesture System Hardware and Software

Hardware Configuration

We mounted a commercially available transparent, resistive-film, touch-sensitive screen on the face of a color display monitor. The touch screen is supported by four force-measuring strain gauges ("load-cells") at the four corners of the screens. The mechanical arrangement is shown,



We note that a touch screen that used a different arrangement to obtain some force information from strain gauges was built in the 1970's at the MIT Architecture Machine Group.

The load cells are mounted so that they supply useful force information through a 0-10 lb. range for finger gestures, and are protected from damage due to overloading. To make the user feel that they are actually "touching" objects on the screen, the surface touched by the user must be as close as possible to the monitor face to prevent parallax problems. In this arrangement, the position sensitive panel floats about 1/8" above the display surface.

This is a block diagram of mechanical connections and information flow in our system.

```
 _________       __________________
| LISPM  |******>|  COLOR MONITOR  |
---------        ------------------
      /*\          |       |
       *         __________________
       ********|  POSITION PANEL  |  <- - -
           *     ------------------   Finger,
           *       |       |          Stylus
           *     __________________
           ****|  FORCE SENSORS   |
                 ------------------
```

Figure: Block Diagram

Our gesture system software is written in LISP and runs on a LISP Machine. Raw position and load information is processed by the gesture software into recognized motions or gestures. The effects of the gestures on one of our gesture-interfaced environments is computed, and the LISP Machine color system creates the display seen through the transparent gesture sensing screen.

Signal Processing

A variety of smoothing and calibration strategies are necessary.

The position screen reports a finger position with a nominal resolution of 4K x 4K. The force sensor hardware reports 12 bits of force data from each of the four load cells.

To get the display and touch screens into good registration, the system performs an initial calibration when it is turned on. This includes finding a linear scaling factor for the x and y position components, and finding x and y offsets. The x and y force offsets are also calculated.

During operation, an asynchronous process interprets position inputs and four force sensor inputs in terms of finger gestures. This process looks for the initiation of a touch, waits until it has computed a trustworthy, filtered initial position, and then signals that a new gesture has begun.

This process tracks the trajectory of the finger during a gesture. The tracking of touched positions is aided by a three-point median smoothing algorithm applied to the position data and by using continuity constraints derived from the force data.

The gesture process uses local continuity of the shear force magnitude and direction to ignore sudden position changes. Most position readings that are associated with these sudden changes must be ignored, since finger trajectories on the screen are mechanically constrained (over short time scales) to have shear forces parallel to the direction of motion, and smooth rotation of shear at inflection points.

Another asynchronous process recalibrates the force sensors every few seconds.

The load cells are arranged at the corners of the screen:

```
1--------------2
|              |
|              |
|              |
|              |
|              |
3--------------4
```

If the loads (corrected for zero force offset) on the load cells are **S1**, **S2**, **S3**, and **S4**, the z-force (pressure) is:

```
PRESSURE = S1 + S2 + S3 + S4
```

The components in the plane of the screen are:

```
X-FORCE (horizontal)
        = (S1 + S3) - (S2 + S4)
                 - zero calibration offset

Y-FORCE (vertical)
        = (S1 + S2) - (S3 + S4)
                 - zero calibration offset
```

We assume that the screen is flat, the load cells are at the corners, the load cells' data is not noisy, and the touch position from the position sensing screen (after smoothing) is correct.

There are no corrections for nonlinearity over the screen. There is noticeable nonlinearity, but it does not seem to affect gesture recognition and tracking. Thus the user has to "get the feel" of the screen, since tracking in the corners feels slightly different than in the center. We plan to correct for this. It may become more important if the system is trying to recognize more complex gestures than we have worked with so far.

We have gained an advantage in this setup from multiple sources of gesture information, e.g. the use of local continuity of forces to help track positions. We could theoretically derive position from the force sensors, but the touch-sensitive screen gives us higher position resolution and perhaps more reliability. We do not cross-check because this system worked well enough for our purposes.

6. Future Directions

Directions for Gesture Programming

There are several force controlled gestures with which we have experimented briefly and on which we plan to do more work.

An example is "flicking". The user can send an object to another part of the screen by flicking it with the finger, as in tiddlywinks. Shear is used to determine the direction of motion and the force determines the initial velocity of the object, which slows down by "friction".

We intend to create more application worlds to put into the Rooms. For example, we plan extensions to the Button Box, a gesture controlled kit for making treasure maps, and a button environment in which the buttons represent musical notes and phrases.

We would like to study classes of gestures that are useful in systems intended for expert use, in other words, systems where the gestures may be useful once learned but are not as easy or obvious as the ones in our current repertoire.

We see the need to do more careful motion studies, and to record more data about people's ability to use the gestures we recognize. There is already extensive literature in related areas, for example [Loomis 83]. We have several ideas for gestures we would like to recognize that make more use of force and force-time contours to express analog quantities. We also plan to build a multiple finger touch version of our current hardware to allow gestures that use more of the hand.

We also intend to explore vision based gesture input, which allows the user much more freedom. There is ongoing research in this laboratory on real-time visual interpretation of whole body gesture as input to a motion and animation display [Hardebeck 84]. We may also explore recognition of hand gestures through vision of hand silhouettes.

Directions for Information Layout

This research has prompted the beginining of a project in this laboratory to develop schemes for using gestural input in "Information Spaces". A prototype of an information organization system has been created that displays representations of file systems and large programs. There are objects displayed in these representations that are analogous to our logic gates and buttons. This system uses a modified mouse and recognizes more "iconic" gestures than the systems described in this paper.

Acknowledgements

I would like to thank Danny Hillis for providing the initial leadership in starting this project, and for continuing ideas, inventions, and support. Ed Hardebeck has done a large amount of the design and most of the implementation of the systems described in this paper. Others who worked on this project are Dan Huttenlocher, Gregor Kiczales, Warren Robinett, and Fred Thornburgh. David Wallace and David Chapman partipated in early design of the gesture environments.

Thanks to Cynthia Solomon in her role as Director of the Atari Cambridge Research Lab, and for reading many drafts of this paper. Many thanks for help with the paper also go to Ed Hardebeck, Danny Hillis, Dan Huttenlocher, and Marvin Minsky.

References

[Abelson 81] Abelson, Harold, and Andrea DiSessa, *Turtle Geometry*, MIT Press, Cambridge, 1981.

[Bolt 80] Bolt, R. A., "'Put-That-There': Voice and Gesture at the Graphics Interface", Proc. Siggraph '80, *Computer Graphics* **14**, no. 3, July 1980.

[Hardebeck 84] Hardebeck, Edward F.,"Gestural Inpput to Computers through Visual Recognition of Body Silhouettes", Submitted to Siggraph '84, 1984.

[Loomis 83] Loomis, Jeffrey, Poizner, Howard, Bellugi, Ursula, Blakemore, Alynn, and John Hollerbach, "Computer Graphic Modeling of American Sign Language", Proc. Siggraph '83, *Computer Graphics* **17**, no. 3, July 1983.

[Papert 80] Papert, S. A., *Mindstorms*, Basic Books, New York, 1980.

[Perlman 74] "Tortis: Toddler's Own Recursive Turtle Interpreter System", Logo Memo 9, Logo Laboratory, Massachusetts Institute of Technology, Cambridge, July 1974.

[Perlman 76] "How to Use the Slot Machine", Logo Working Paper 37, Logo Laboratory, Massachusetts Institute of Technology, Cambridge, January 1976.

[Robinett 79] Atari VCS Adventure, Software product of Atari, Inc., Sunnyvale, CA, 1979.

[Robinett 82] Rocky's Boots, Software product of The Learning Co., Portola Valley, CA, 1982.

News from Harper & Row
Electronic and Technical Publishing

10 East 53d Street, New York, New York 10022
Cable: Harpsam Telephone: (212) 207-7627

For immediate release
Contact: Jessica Solon

Coming in November!

LOGO works

GREAT PROGRAMS in your ATARI LOGO

edited
by Cynthia Solomon, Margaret Minsky, and Brian Harvey

preface by Marvin Minsky

The power of Logo, the most popular children's programming language, is unleashed as never before in this exciting new book.

Logo to date has been used mostly for educational purposes; yet Logo is in fact a very sophisticated and powerful tool. In this remarkable book the authors provide many innovative Logo applications. Developed by the foremost Atari Logo experts, LOGO WORKS is the first book that shows how beautifully Logo works for a vast range of programming applications.

LOGO WORKS features more than thirty programs for Atari Logo, including:

* Wordplay games: like Madlibs, Wordscram, Hangman, Argue, Animal Game, and more;

* Story programs: Cartoon, Jack and Jill, and Rocket Blast;

* Games: Boxgame, Pacgame, Dungeon, Alien, and others;

* Turtle Geometry: including Turtle Race, Four Corner Turtles, Polycirc, and animated line drawings;

* Music programs: for Melodies, Ear Training, Sound Effects and Naming Notes;

(more)

* Other programming ideas: such as making an operation for adding numbers, a small database manager, Mergesort, and lines and mirrors;

* Special features of Atari Logo: including Turtle Graphics, shapes, demons, and Turtle Collisions.

LOGO WORKS includes an entertaining preface by Marvin Minsky, one of the premier researchers in artificial intelligence--and a great Logo enthusiast.

This book represents a major advance for the millions of Logo users--even those who don't have Atari Logo. It provides exciting applications, full descriptions of their development, and complete program listings. Fully indexed, LOGO WORKS is the most complete, instructive, and inspiring book on Atari Logo for everyone.

About the Authors

Cynthia Solomon, Margaret Minsky, and Brian Harvey are acknowledged Logo experts. The programs in LOGO WORKS were developed while Ms. Solomon, Ms. Minsky and Mr. Harvey were working for Atari Incorporated. Ms. Solomon was previously Director of the Atari Cambridge Research Laboratory, where Ms. Minsky was a Research Scientist. Mr. Harvey was the founding Director of the Computer Dept. at the Lincoln-Sudbury Regional High School in Massachusetts, an innovative center for computer education in which Logo is the primary teaching language. He is currently a Ph.D Candidate in Science and Mathematic Education at the University of California, Berkeley.

Approximately 300 pages, illustrated, indexed.

ISBN 0-06-669020-X

$14.95

WORKS LOGO WORKS LOGO WORKS LOGO WORKS

1026852281