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

Margaret R. Minsky
Atari Cambridge Research
Cambridge, Massachusetts

Author's present address:
Media Laboratory
Massachusetts Institute of Technology
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.

CR Categories and Subject Descriptors: I.3.6 [Computer Graphics] Methodology and Techniques - interaction techniques; H.1.2 [Models and Principles]: User/Machine Systems - human information processing; D.2.2 [Software Engineering] Tools and Techniques - user interfaces; I.3.1 [Computer Graphics] Hardware Architecture - input devices

General Terms: Design, Experimentation, Languages

Additional Key Words and Phrases: gesture, touch-sensitive screen, visual programming, computers and education, paint programs

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 [5] - to gestural and dynamic visual representations of programming-like activities.

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 *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 [2] 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 (Fig. 1).

Figure 1: Fingerpaint

If the user presses harder, he gets a bigger blob of paint (Fig. 2).

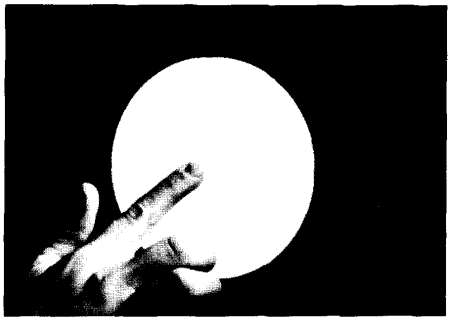

Figure 1: Fingerpaint with Varying Pressure

The user can choose from several paint colors, and can also paint with simulated spray paint (Fig. 3).

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 by Radia Perlman with special terminals (called Button Box and Slot Machine) built for preliterate children [6,7]. The Slot Machine is a plexiglass bar with

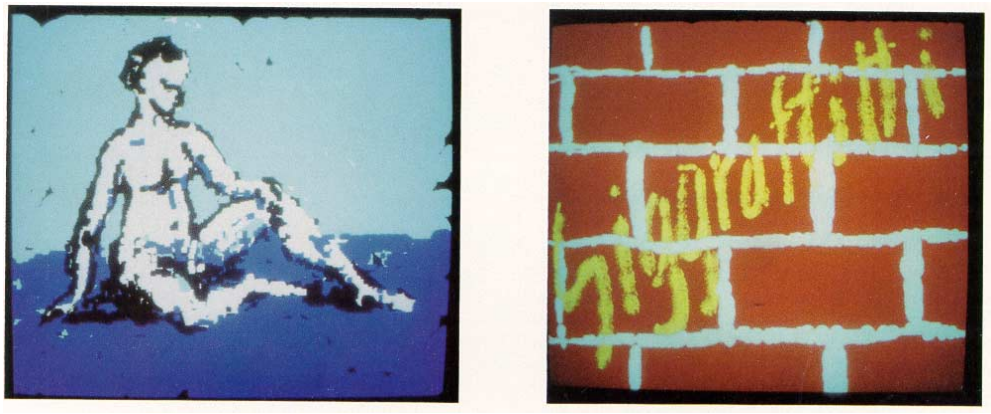

Figure 3: Fingerpaintings

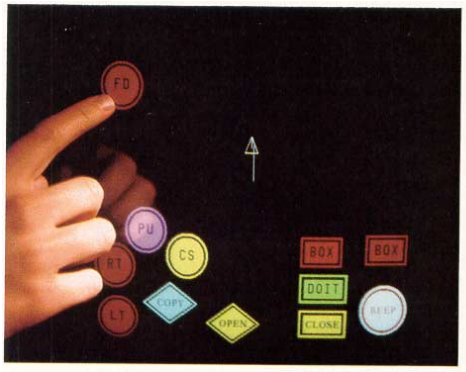

Figure 4: Forward

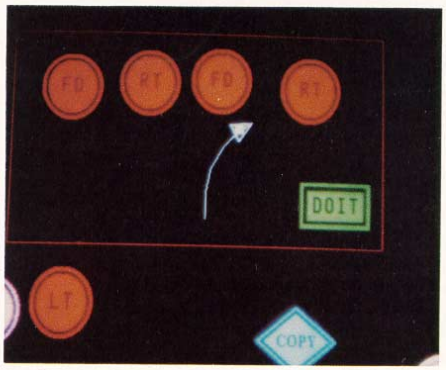

Figure 5: Arranging Buttons

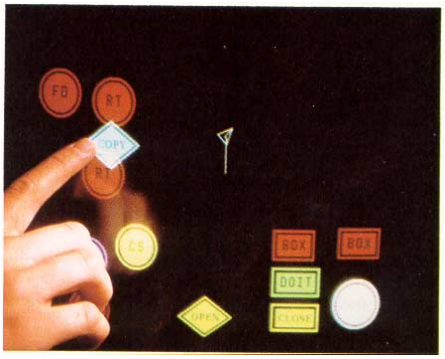

Figure 6: A Button being Copied

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 Button Box is even more flexible than the original specially constructed hardware devices; since it is software it can be modified and reconfigured. The current implementation makes use of some force and gesture information. It can be viewed as work in progress toward 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 [1]. 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 (Fig. 4).

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 (Fig. 5).

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 (Fig. 6).

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 (Fig. 7).

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 (Fig. 8).

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 (Fig. 9).

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].

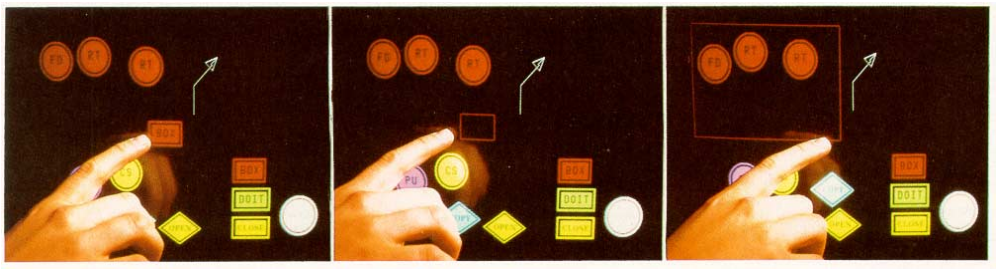

Figure 7: Making a Box and Using it to Group Buttons

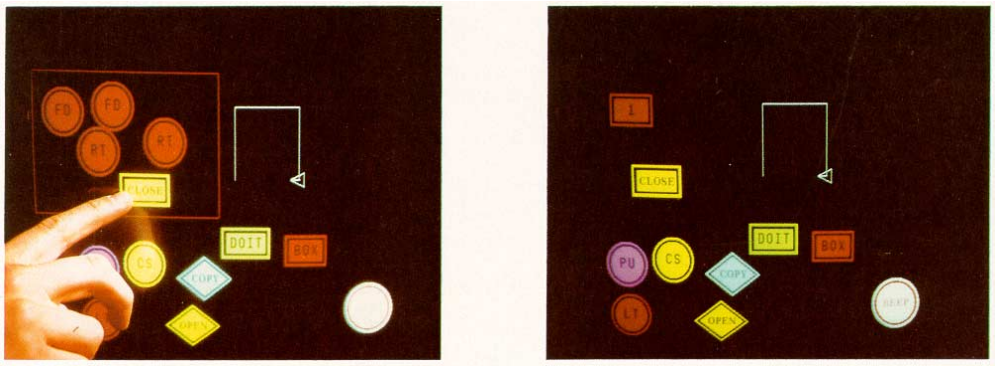

Figure 9: Creating a New Button by Closing a Box

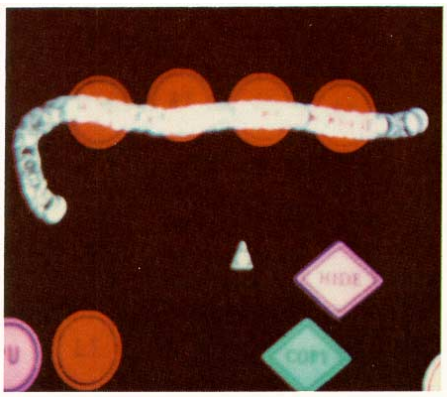

Figure 8: Using Magic Paint to Group Buttons

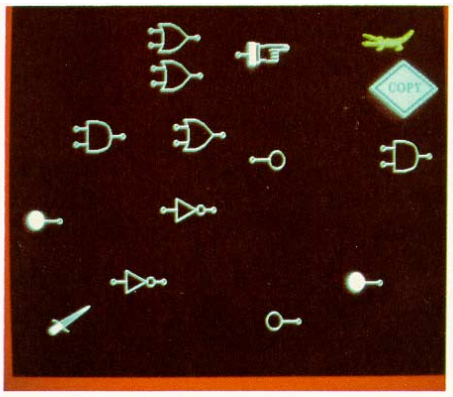

Figure 10: Logic objects:
Gates, HI input, clock input (blurred), output light

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.