ware moved from the ungainly and delicate world of vacuum tubes and paper tape to the reliable and efficient world of transistors and magnetic storage. The 1950s saw the development of key technical underpinnings for widespread computing: cheap and reliable transistors available in large quantities, rotating magnetic drum and disk storage, magnetic core memory, and beginning work in semiconductor packaging and miniaturization, particularly for missiles. In telecommunications, American Telephone and Telegraph (AT&T) introduced nationwide dialing and the first electronic switching systems at the end of the decade. A fledgling commercial computer industry emerged, led by International Business Machines (IBM) (which built its electronic computer capability internally) and Remington Rand (later Sperry Rand), which purchased Eckert-Mauchly Computer Corporation in 1950 and Engineering Research Associates in 1952. Other important participants included Bendix, Burroughs, General Electric (GE), Honeywell, Philco, Raytheon, and Radio Corporation of America (RCA).

In computing, the technical cutting edge, however, was usually pushed forward in government facilities, at government-funded research centers, or at private contractors doing government work. Government funding accounted for roughly three-quarters of the total computer field. A survey performed by the Army Ballistics Research Laboratory in 1957, 1959, and 1961 lists every electronic stored-program computer in use in the country (the very possibility of compiling such a list says a great deal about the community of computing at the time). The surveys reveal the large proportion of machines in use for government purposes, either by federal contractors or in government facilities.

The Government's Early Role

(From pp. 87-88): Before 1960, government—as a funder and as a customer—dominated electronic computing. Federal support had no broad, coherent approach, however, arising somewhat ad hoc in individual federal agencies. The period was one of experimentation, both with the technology itself and with diverse mechanisms for federal support. From the panoply of solutions, distinct successes and failures can be discerned, from both scientific and economic points of view. After 1960, computing was more prominently recognized as an issue for federal policy. The National Science Foundation and the National Academy of Sciences issued surveys and reports on the field.

If government was the main driver for computing research and development (R&D) during this period, the main driver for government was the defense needs of the Cold War. Events such as the explosion of a Soviet atomic bomb in 1949 and the Korean War in the 1950s heightened

Copyright © National Academy of Sciences. All rights reserved.

international tensions and called for critical defense applications, especially command-and-control and weapons design. It is worth noting, however, that such forces did not exert a strong influence on telecommunications, an area in which most R&D was performed within AT&T for civilian purposes. Long-distance transmission remained analog, although digital systems were in development at AT&T's Bell Laboratories. Still, the newly emergent field of semiconductors was largely supported by defense in its early years. During the 1950s, the Department of Defense (DOD) supported about 25 percent of transistor research at Bell Laboratories.

However much the Cold War generated computer funding, during the 1950s dollars and scale remained relatively small compared to other fields, such as aerospace applications, missile programs, and the Navy's Polaris program (although many of these programs had significant computing components, especially for operations research and advanced management techniques). By 1950, government investment in computing amounted to $15 million to $20 million per year.

All of the major computer companies during the 1950s had significant components of their R&D supported by government contracts of some type. At IBM, for example, federal contracts supported more than half of the R&D and about 35 percent of R&D as late as 1963 (only in the late 1960s did this proportion of support trail off significantly, although absolute amounts still increased). The federal government supported projects and ideas the private sector would not fund, either for national security, to build up human capital, or to explore the capabilities of a complex, expensive technology whose long-term impact and use was uncertain. Many federally supported projects put in place prototype hardware on which researchers could do exploratory work.

Establishment of Organizations

(From pp. 88-95): The successful development projects of World War II, particularly radar and the atomic bomb, left policymakers asking how to maintain the technological momentum in peacetime. Numerous new government organizations arose, attempting to sustain the creative atmosphere of the famous wartime research projects and to enhance national leadership in science and technology. Despite Vannevar Bush's efforts to establish a new national research foundation to support research in the nation's universities, political difficulties prevented the bill from passing until 1950, and the National Science Foundation (NSF) did not become a significant player in computing until later in that decade. During the 15 years immediately after World War II, research in computing and communications was supported by mission agencies of the federal government, such as DOD, the Department of Energy (DOE), and NASA. In

Copyright © National Academy of Sciences. All rights reserved.

Innovation in Information Technology
http://www.nap.edu/catalog/10795.html

retrospect, it seems that the nation was experimenting with different models for supporting this intriguing new technology that required a subtle mix of scientific and engineering skill.

Military Research Offices

Continuity in basic science was provided primarily by the Office of Naval Research (ONR), created in 1946 explicitly to perpetuate the contributions scientists made to military problems during World War II. In computing, the agency took a variety of approaches simultaneously. First, it supported basic intellectual and mathematical work, particularly in numerical analysis. These projects proved instrumental in establishing a sound mathematical basis for computer design and computer processing. Second, ONR supported intellectual infrastructure in the infant field of computing, sponsoring conferences and publications for information dissemination. Members of ONR participated in founding the Association for Computing Machinery in 1947.

ONR's third approach to computing was to sponsor machine design and construction. It ordered a computer for missile testing through the National Bureau of Standards from Raytheon, which became known as the Raydac machine, installed in 1952. ONR supported Whirlwind, MIT's first digital computer and progenitor of real-time command-and-control systems. John von Neumann built a machine with support from ONR and other agencies at Princeton's Institute for Advanced Study, known as the IAS computer. The project produced significant advances in computer architecture, and the design was widely copied by both government and industrial organizations.

Other military services created offices on a model similar to that of ONR. The Air Force Office of Scientific Research was established in 1950 to manage U.S. Air Force R&D activities. Similarly, the U.S. Army established the Army Research Office to manage and promote Army programs in science and technology.

National Bureau of Standards

Arising out of its role as arbiter of weights and measures, the National Bureau of Standards (NBS) had long had its own laboratories and technical expertise and had long served as a technical advisor to other government agencies. In the immediate postwar years, NBS sought to expand its advisory role and help U.S. industry develop wartime technology for commercial purposes. NBS, through its National Applied Mathematics Laboratory, acted as a kind of expert agent for other government agencies, selecting suppliers and overseeing construction and delivery of

Copyright © National Academy of Sciences. All rights reserved.

new computers. For example, NBS contracted for the three initial Univac machines—the first commercial, electronic, digital, stored-program computers—one for the Census Bureau and two for the Air Materiel Command.

NBS also got into the business of building machines. When the Univac order was plagued by technical delays, NBS built its own computer inhouse. The Standards Eastern Automatic Computer (SEAC) was built for the Air Force and dedicated in 1950, the first operational, electronic, stored-program computer in this country. NBS built a similar machine, the Standards Western Automatic Computer (SWAC) for the Navy on the West Coast. Numerous problems were run on SEAC, and the computer also served as a central facility for diffusing expertise in programming to other government agencies. Despite this significant hardware, however, NBS's bid to be a government center for computing expertise ended in the mid-1950s. Caught up in postwar debates over science policy and a controversy over battery additives, NBS research funding was radically reduced, and NBS lost its momentum in the field of computing.

Atomic Energy Commission

Nuclear weapons design and research have from the beginning provided impetus to advances in large-scale computation. The first atomic bombs were designed only with desktop calculators and punched-card equipment, but continued work on nuclear weapons provided some of the earliest applications for the new electronic machines as they evolved. The first computation job run on the ENIAC in 1945 was an early calculation for the hydrogen bomb project "Super." In the late 1940s, the Los Alamos National Laboratory built its own computer, MANIAC, based on von Neumann's design for the Institute for Advanced Study computer at Princeton, and the Atomic Energy Commission (AEC) funded similar machines at Argonne National Laboratory and Oak Ridge National Laboratory.

In addition to building their own computers, the AEC laboratories were significant customers for supercomputers. The demand created by AEC laboratories for computing power provided companies with an incentive to design more powerful computers with new designs. In the early 1950s, IBM built its 701, the Defense Calculator, partly with the assurance that Los Alamos and Livermore would each buy at least one. In 1955, the AEC laboratory at Livermore, California, commissioned Remington Rand to design and build the Livermore Automatic Research Computer (LARC), the first supercomputer. The mere specification for LARC advanced the state of the art, as the bidding competition required the use of transistors instead of vacuum tubes. IBM developed improved

Copyright © National Academy of Sciences. All rights reserved.

ferrite-core memories and supercomputer designs with funding from the National Security Agency, and designed and built the Stretch supercomputer for the Los Alamos Scientific Laboratory, beginning it in 1956 and installing it in 1961. Seven more Stretch supercomputers were built. Half of the Stretch supercomputers sold were used for nuclear weapon research and design.

The AEC continued to specify and buy newer and faster supercomputers, including the Control Data 6600, the STAR 100, and the Cray 1 (although developed without AEC funds), practically ensuring a market for continued advancements. AEC and DOE laboratories also developed much of the software used in high-performance computing including operating systems, numerical analysis software, and matrix evaluation routines. In addition to stimulating R&D in industry, the AEC laboratories also developed a large talent pool on which the computer industry and academia could draw. In fact, the head of IBM's Applied Science Department, Cuthbert Hurd, came directly to IBM in 1949 from the AEC's Oak Ridge National Laboratory. Physicists worked on national security problems with government support providing demand, specifications, and technical input, as well as dollars, for industry to make significant advances in computing technology.

Private Organizations

Not all the new organizations created by the government to support computing were public. A number of new private organizations also sprang up with innovative new charters and government encouragement that held prospects of initial funding support. In 1956, at the request of the Air Force, the Massachusetts Institute of Technology (MIT) created Project Lincoln, now known as the Lincoln Laboratory, with a broad charter to study problems in air defense to protect the nation from nuclear attack. The Lincoln Laboratory then oversaw the construction of the Semi-Automatic Ground Environment (SAGE) air-defense system. In 1946, the Air Force and Douglas Aircraft created a joint venture, Project RAND, to study intercontinental warfare. In the following year RAND separated from Douglas and became the independent, nonprofit RAND Corporation.

RAND worked only for the Air Force until 1956, when it began to diversify to other defense and defense-related contractors, such as the Advanced Research Projects Agency and the Atomic Energy Commission, and provided, for a time, what one researcher called "in some sense the world's largest installation for scientific computing [in 1950]." RAND specialized in developing computer systems, such as the Johnniac, based on the IAS computer, which made RAND the logical source for the pro-

Copyright © National Academy of Sciences. All rights reserved.

gramming on SAGE. While working on SAGE, RAND trained hundreds of programmers, eventually leading to the spin-off of RAND's Systems Development Division and Systems Training Program into the Systems Development Corporation. Computers made a major impact on the systems analysis and game theoretic approaches that RAND and other similar think tanks used in attempts to model nuclear and conventional warfighting strategies.

Engineering Research Associates (ERA) represented yet another form of government support: the private contractor growing out of a single government agency. With ERA, the Navy effectively privatized its wartime cryptography organization and was able to maintain civilian expertise through the radical postwar demobilization. ERA was founded in St. Paul, Minnesota, in January 1946 by two engineers who had done cryptography for the Navy and their business partners. The Navy moved its Naval Computing Machine Laboratory from Dayton to St. Paul, and ERA essentially became the laboratory. ERA did some research, but it primarily worked on task-oriented, cost-plus contracts. As one participant recalled, "It was not a university atmosphere. It was 'Build stuff. Make it work. How do you package it? How do you fix it? How do you document it?'" ERA built a community of engineering skill, which became the foundation of the Minnesota computer industry. In 1951, for example, the company hired Seymour Cray for his first job out of the University of Minnesota.

As noted earlier, the RAND Corporation had contracted in 1955 to write much of the software for SAGE owing to its earlier experience in air defense and its large pool of programmers. By 1956, the Systems Training Program of the RAND Corporation, the division assigned to SAGE, was larger than the rest of the corporation combined, and it spun off into the nonprofit Systems Development Corporation (SDC). SDC played a significant role in computer training. As described by one of the participants, "Part of SDC's nonprofit role was to be a university for programmers. Hence our policy in those days was not to oppose the recruiting of our personnel and not to match higher salary offers with an SDC raise." By 1963, SDC had trained more than 10,000 employees in the field of computer systems. Of those, 6,000 had moved to other businesses across the country.

Observations

(From pp. 95-96): In retrospect, the 1950s appear to have been a period of institutional and technological experimentation. This diversity of approaches, while it brought the field and the industry from virtually nothing to a tentative stability, was open to criticisms of waste, duplica-

Copyright © National Academy of Sciences. All rights reserved.

tion of effort, and ineffectiveness caused by rivalries among organizations and their funding sources. The field was also driven largely by the needs of government agencies, with relatively little input from computer-oriented scientists at the highest levels. Criticism remained muted during the decade when the military imperatives of the Cold War seemed to dominate all others, but one event late in the decade opened the entire system of federal research support to scrutiny: the launch of Sputnik in 1957. Attacks mounted that the system of R&D needed to be changed, and they came not only from the press and the politicians but also from scientists themselves.

1960-1970: Supporting a Continuing Revolution

(From p. 96): Several significant events occurred to mark a transition from the infancy of information technology to a period of diffusion and growth. Most important of these was the launching of Sputnik in 1957, which sent convulsions through the U.S. science and engineering world and redoubled efforts to develop new technology. President Eisenhower elevated scientists and engineers to the highest levels of policy making. Thus was inaugurated what some have called the golden age of U.S. research policy. Government support for information technology took off in the 1960s and assumed its modern form. The Kennedy administration brought a spirit of technocratic reform to the Pentagon and the introduction of systems analysis and computer-based management to all aspects of running the military. Many of the visions that set the research agendas for the following 15 years (and whose influence remains today) were set in the early years of the decade.

Maturing of a Commercial Industry

(From pp. 96-97): Perhaps most important, the early 1960s can be defined as the time when the commercial computer industry became significant on its own, independent of government funding and procurement. Computerized reservation systems began to proliferate, particularly the IBM/American Airlines SABRE system, based in part on prior experience with military command-and-control systems (such as SAGE). The introduction of the IBM System/360 in 1964 solidified computer applications in business, and the industry itself, as significant components of the economy.

This newly vital industry, dominated by "Snow White" (IBM) and the "Seven Dwarfs" (Burroughs, Control Data, GE, Honeywell, NCR, RCA, and Sperry Rand), came to have several effects on government-supported R&D. First, and most obvious, some companies (mostly IBM) became

Copyright © National Academy of Sciences. All rights reserved.

Innovation in Information Technology
http://www.nap.edu/catalog/10795.html

large enough to conduct their own in-house research. IBM's Thomas J. Watson Research Center was dedicated in 1961. Its director, Emanuel Piore, was recruited from ONR, and he emphasized basic research. Such laboratories not only expanded the pool of researchers in computing and communications but also supplied a source of applied research that allowed or, conversely, pushed federal support to focus increasingly on the longest-term, riskiest ideas and on problems unique to government. Second, the industry became a growing employer of computer professionals, providing impetus to educational programs at universities and making computer science and engineering increasingly attractive career paths to talented young people.

These years saw turning points in telecommunications as well. In 1962, AT&T launched the first active communications satellite, Telstar, which transmitted the first satellite-relay telephone call and the first live transatlantic television signal. That same year, a less-noticed but equally significant event occurred when AT&T installed the first commercial digital-transmission system. Twenty-four digital speech channels were time multiplexed onto a repeatered digital transmission line operating at 1.5 megabits per second. In 1963, the first Stored Program Control electronic switching system was placed into service, inaugurating the use of digital computer technology for mainstream switching.

The 1960s also saw the emergence of the field called computer science, and several important university departments were founded during the decade, at Stanford and Carnegie Mellon in 1965 and at MIT in 1968. Hardware platforms had stabilized enough to support a community of researchers who attacked a common set of problems. New languages proliferated, often initiated by government and buoyed by the needs of commercial industry. The Navy had sponsored Grace Hopper and others during the 1950s to develop automatic programming techniques that became the first compilers. John Backus and a group at IBM developed FORTRAN, which was distributed to IBM users in 1957. A team led by John McCarthy at MIT (with government support) began implementing LISP in 1958, and the language became widely used, particularly for artificial intelligence programming, in the early 1960s. In 1959, the Pentagon began convening a group of computer experts from government, academia, and industry to define common business languages for computers. The group published a specification in 1959, and by 1960 RCA and Remington Rand Univac had produced the first COBOL compilers. By the beginning of the 1960s, a number of computer languages, standard across numerous hardware platforms, were beginning to define programming as a task, as a profession, and as a challenging and legitimate subject of intellectual inquiry.

Copyright © National Academy of Sciences. All rights reserved.

The Changing Federal Role

(From pp. 98-107): The forces driving government support changed during the 1960s. The Cold War remained a paramount concern, but to it were added the difficult conflict in Vietnam, the Great Society programs, and the Apollo program, inaugurated by President Kennedy's 1961 challenge. New political goals, new technologies, and new missions provoked changes in the federal agency population. Among these, two agencies became particularly important in computing: the new Advanced Research Projects Agency and the National Science Foundation.

The Advanced Research Projects Agency

The founding of the Advanced Research Projects Agency (ARPA) in 1958, a direct outgrowth of the Sputnik scare, had immeasurable impact on computing and communications. ARPA, specifically charged with preventing technological surprises like Sputnik, began conducting long-range, high-risk research. It was originally conceived as the DOD's own space agency, reporting directly to the Secretary of Defense in order to avoid interservice rivalry. Space, like computing, did not seem to fit into the existing military service structure. ARPA's independent status not only insulated it from established service interests but also tended to foster radical ideas and keep the agency tuned to basic research questions: when the agency-supported work became too much like systems development, it ran the risk of treading on the territory of a specific service.

ARPA's status as the DOD space agency did not last long. Soon after NASA's creation in 1958, ARPA retained essentially no role as a space agency. ARPA instead focused its energies on ballistic missile defense, nuclear test detection, propellants, and materials. It also established a critical organizational infrastructure and management style: a small, high-quality managerial staff, supported by scientists and engineers on rotation from industry and academia, successfully employing existing DOD laboratories and contracting procedures (rather than creating its own research facilities) to build solid programs in new, complex fields. ARPA also emerged as an agency extremely sensitive to the personality and vision of its director.

ARPA's decline as a space agency raised questions about its role and character. A new director, Jack Ruina, answered the questions in no uncertain terms by cementing the agency's reputation as an elite, scientifically respected institution devoted to basic, long-term research projects. Ruina, ARPA's first scientist-director, took office at the same time as Kennedy and McNamara in 1961, and brought a similar spirit to the

Copyright © National Academy of Sciences. All rights reserved.

agency. Ruina decentralized management at ARPA and began the tradition of relying heavily on independent office directors and program managers to run research programs. Ruina also valued scientific and technical merit above immediate relevance to the military. Ruina believed both of these characteristics—independence and intellectual quality—were critical to attracting the best people, both to ARPA as an organization and to ARPA-sponsored research. Interestingly, ARPA's managerial success did not rely on innovative managerial techniques per se (such as the computerized project scheduling typical of the Navy's Polaris project) but rather on the creative use of existing mechanisms such as "no-year money," unsolicited proposals, sole-source procurement, and multiyear forward funding.

ARPA and Information Technology. From the point of view of computing, the most important event at ARPA in the early 1960s, indeed in all of ARPA's history, was the establishment of the Information Processing Techniques Office, IPTO, in 1962. The impetus for this move came from several directions, including Kennedy's call a year earlier for improvements in command-and-control systems to make them "more flexible, more selective, more deliberate, better protected, and under ultimate civilian authority at all times." Computing as applied to command and control was the ideal ARPA program—it had no clearly established service affinity; it was "a new area with relatively little established service interest and entailed far less constraint on ARPA's freedom of action," than more familiar technologies. Ruina established IPTO to be devoted not to command and control but to the more fundamental problems in computing that would, eventually, contribute solutions.

Consistent with his philosophy of strong, independent, and scientific office managers, Ruina appointed J.C.R. Licklider to head IPTO. The Harvard-trained psychologist came to ARPA in October 1962, primarily to run its Command and Control Group. Licklider split that group into two discipline-oriented offices: Behavioral Sciences Office and IPTO. Licklider had had extensive exposure to the computer research of the time and had clearly defined his own vision of "man-computer symbiosis," which he had published in a landmark paper of 1960 by the same name. He saw human-computer interaction as the key, not only to command and control, but also to bringing together the then-disparate techniques of electronic computing to form a unified science of computers as tools for augmenting human thought and creativity. Licklider formed IPTO in this image, working largely independently of any direction from Ruina, who spent the majority of his time on higher-profile and higher-funded missile defense issues. Licklider's timing was opportune: the 1950s had produced a stable technology of digital computer hardware, and the big systems

Copyright © National Academy of Sciences. All rights reserved.

projects had shown that programming these machines was a difficult but interesting problem in its own right. Now the pertinent questions concerned how to use "this tremendous power . . . for other than purely numerical scientific calculations." Licklider not only brought this vision to IPTO itself, but he also promoted it with missionary zeal to the research community at large. Licklider's and IPTO's success derived in large part from their skills at "selling the vision" in addition to "buying the research."

Another remarkable feature of IPTO, particularly during the 1960s, was its ability to maintain the coherent vision over a long period of time; the office director was able to handpick his successor. Licklider chose Ivan Sutherland, a dynamic young researcher he had encountered as a graduate student at MIT and the Lincoln Laboratory, to succeed him in 1964. Sutherland carried on Licklider's basic ideas and made his own impact by emphasizing computer graphics. Sutherland's own successor, Robert Taylor, came in 1966 from a job as a program officer at NASA and recalled, "I became heartily subscribed to the Licklider vision of interactive computing." While at IPTO, Taylor emphasized networking. The last IPTO director of the 1960s, Lawrence Roberts, came, like Sutherland, from MIT and Lincoln Laboratory, where he had worked on the early transistorized computers and had conducted ARPA research in both graphics and communications.

During the 1960s, ARPA and IPTO had more effect on the science and technology of computing than any other single government agency, sometimes raising concern that the research agenda for computing was being directed by military needs. IPTO's sheer size, $15 million in 1965, dwarfed other agencies such as ONR. Still, it is important to note, ONR and ARPA worked closely together; ONR would often let small contracts to researchers and serve as a talent agent for ARPA, which would then fund promising projects at larger scale. ARPA combined the best features of existing military research support with a new, lean administrative structure and innovative management style to fund high-risk projects consistently. The agency had the freedom to administer large block grants as well as multiple-year contracts, allowing it the luxury of a long-term vision to foster technologies, disciplines, and institutions. Further, the national defense motivation allowed IPTO to concentrate its resources on centers of scientific and engineering excellence (such as MIT, Carnegie Mellon University, and Stanford University) without regard for geographical distribution questions with which NSF had to be concerned. Such an approach helped to create university-based research groups with the critical mass and stability of funding needed to create significant advances in particular technical areas. But although it trained generations of young researchers in those areas, ARPA's funding style did little to help them pursue the

Copyright © National Academy of Sciences. All rights reserved.

Innovation in Information Technology
http://www.nap.edu/catalog/10795.html

same lines of work at other universities. As an indirect and possibly unintended consequence, the research approaches and tools and the generic technologies developed under ARPA's patronage were disseminated more rapidly and widely, and so came to be applied in new nonmilitary contexts by the young M.S. and Ph.D. graduates who had been trained in that environment but could not expect to make their research careers within it.

ARPA's Management Style. To evaluate research proposals, IPTO did not employ the peer-review process like NSF, but rather relied on internal reviews and the discretion of program managers as did ONR. These program managers, working under office managers such as Licklider, Sutherland, Taylor, and Roberts, came to have enormous influence over their areas of responsibility and became familiar with the entire field both personally and intellectually. They had the freedom and the resources to shape multiple R&D contracts into a larger vision and to stimulate new areas of inquiry. The education, recruiting, and responsibilities of these program managers thus became a critical parameter in the character and success of ARPA programs. ARPA frequently chose people who had training and research experience in the fields they would fund, and thus who had insight and opinions on where those fields should go.

To have such effects, the program managers were given enough funds to let a large enough number of contracts and to shape a coherent research program, with minimal responsibilities for managing staffs. Program budgets usually required only two levels of approval above the program manager: the director of IPTO and the director of ARPA. One IPTO member described what he called "the joy of ARPA. . . . You know, if a program manager has a good idea, he has got two people to convince that that is a good idea before the guy goes to work. He has got the director of his office and the director of ARPA, and that is it. It is such a short chain of command."

Part of ARPA's philosophy involved aiming at radical change rather than incremental improvement. As Robert Taylor put it, for example, incremental innovation would be taken care of by the services and their contractors, but, ARPA's aim was "an order of magnitude difference." ARPA identified good ideas and magnified them. This strategy often necessitated funding large, group-oriented projects and institutions rather than individuals. Taylor recalled, "I don't remember a single case where we ever funded a single individual's work. . . . The individual researcher who is just looking for support for his own individual work could [potentially] find many homes to support that work. So we tended not to fund those, because we felt that they were already pretty well covered. Instead, we funded larger groups—teams." NSF's peer-review process worked

Copyright © National Academy of Sciences. All rights reserved.

Innovation in Information Technology
http://www.nap.edu/catalog/10795.html

well for individual projects, but was not likely to support large, team-oriented research projects. Nor did it, at this point in history, support entire institutions and research centers, like the Laboratory for Computer Science at MIT. IPTO's style meshed with its emphasis on human-machine interaction, which it saw as fundamentally a systems problem and hence fundamentally team oriented. In Taylor's view, the university reward structure was much more oriented toward individual projects, so "systems research is most difficult to fund and manage in a university." This philosophy was apparent in ARPA's support of Project MAC, an MIT-led effort on time-shared computing. . . .

ARPA, with its clearly defined mission to support DOD technology, could also afford to be elitist in a way that NSF, with a broader charter to support the country's scientific research, could not. "ARPA had no commitment, for example, to take geography into consideration when it funded work." Another important feature of ARPA's multiyear contracts was their stability, which proved critical for graduate students who could rely on funding to get them through their Ph.D. program. ARPA also paid particular attention to building communities of researchers and disseminating the results of its research, even beyond traditional publications. IPTO would hold annual meetings for its contract researchers at which results would be presented and debated. These meetings proved effective not only at advancing the research itself but also at providing valuable feedback for the program managers and helping to forge relationships between researchers in related areas. Similar conferences were convened for graduate students only, thus building a longer-term community of researchers. ARPA also put significant effort into getting the results of its research programs commercialized so that DOD could benefit from the development and expansion of a commercial industry for information technology. ARPA sponsored conferences that brought together researchers and managers from academia and industry on topics such as time-sharing, for example.

Much has been made of ARPA's management style, but it would be a mistake to conclude that management per se provided the keys to the agency's successes in computing. The key point about the style, in fact, was its light touch. Red tape was kept to a minimum, and project proposals were turned around quickly, frequently into multiple-year contracts. Typical DOD research contracts involved close monitoring and careful adherence to requirements and specifications. ARPA avoided this approach by hiring technically educated program managers who had continuing research interests in the fields they were managing. This reality counters the myth that government bureaucrats heavy-handedly selected R&D problems and managed the grants and contracts. Especially during the 1960s and 1970s, program managers and office directors were not

Copyright © National Academy of Sciences. All rights reserved.

bureaucrats but were usually academics on a 2-year tour of duty. They saw ARPA as a pulpit from which to preach their visions, with money to help them realize those visions. The entire system displayed something of a self-organizing, self-managing nature. As Ivan Sutherland recalled, "Good research comes from the researchers themselves rather than from the outside."

National Science Foundation

While ARPA was focusing on large projects and systems, the National Science Foundation played a large role in legitimizing basic computer science research as an academic discipline and in funding individual researchers at a wide range of institutions. Its programs in computing have evolved considerably since its founding in 1950, but have tended to balance support for research, education, and computing infrastructure. Although early programs tended to focus on the use of computing in other academic disciplines, NSF subsequently emerged as the leading federal funder of basic research in computer science.

NSF was formed before computing became a clearly defined research area, and it established divisions for chemistry, physics, and biology, but not computing. NSF did provide support for computing in its early years, but this support derived more from a desire to promote computer-related activities in other disciplines than to expand computer science as a discipline, and as such was weighted toward support for computing infrastructure. For example, NSF poured millions of dollars into university computing centers so that researchers in other disciplines, such as physics and chemistry, could have access to computing power. NSF noted that little computing power was available to researchers at American universities who were not involved in defense-related research and that "many scientists feel strongly that further progress in their field will be seriously affected by lack of access to the techniques and facilities of electronic computation." As a result, NSF began supporting computing centers at universities in 1956 and, in 1959, allocated a budget specifically for computer equipment purchases. Recognizing that computing technology was expensive, became obsolete rapidly, and entailed significant costs for ongoing support, NSF decided that it would, in effect, pay for American campuses to enter the computer age. In 1962, it established its first office devoted to computing, the program for Computers and Computing Science within the Mathematical Sciences Division. By 1970, the Institutional Computing Services (or Facilities) program had obligated $66 million to university computing centers across the country. NSF intended that use of the new facilities would result in trained personnel to fulfill increasing needs for computer proficiency in industry, government, and academia.

Copyright © National Academy of Sciences. All rights reserved.

NSF provided some funding for computer-related research in its early years. Originally, such funding came out of the mathematics division in the 1950s and grew out of an interest in numerical analysis. By 1955, NSF began to fund basic research in computer science theory with its first grants for the research of recursion theory and one grant to develop an analytical computer program under the Mathematical Sciences Program. Although these projects constituted less than 10 percent of the mathematics budget, they resulted in significant research.

In 1967, NSF united all the facets of its computing support into a single office, the Office of Computing Activities (OCA). The new office incorporated elements from the directorates of mathematics and engineering and from the Facilities program, unifying NSF's research and infrastructure efforts in computing. It also incorporated an educational element that was intended to help meet the radically increasing demand for instruction in computer science. The OCA was headed by Milton Rose, the former head of the Mathematical Sciences Section, and reported directly to the director of NSF.

Originally, the OCA's main focus was improving university computing services. In 1967, $11.3 million of the office's $12.8 million total budget went toward institutional support. Because not all universities were large enough to support their own computing centers but would benefit from access to computing time at other universities, the OCA also began to support regional networks linking many universities together. In 1968, the OCA spent $5.3 million, or 18.6 percent of its budget, to provide links between computers in the same geographic region. In the 1970s, the computer center projects were canceled, however, in favor of shifting emphasis toward education and research.

Beginning in 1968, through the Education and Training program, the OCA began funding the inauguration of university-level computer science programs. NSF funded several conferences and studies to develop computer science curricula. The Education and Training program obligated $12.3 million between 1968 and 1970 for training, curricula development, and support of computer-assisted instruction.

Although the majority of the OCA's funding was spent on infrastructure and education, the office also supported a broad range of basic computer science research programs. These included compiler and language development, theoretical computer science, computation theory, numerical analysis, and algorithms. The Computer Systems Design program concentrated on computer architecture and systems analysis. Other programs focused on topics in artificial intelligence, including pattern recognition and automatic theory proving.

Copyright © National Academy of Sciences. All rights reserved.

1970-1990: Retrenching and International Competition

(From p. 107): Despite previous successes, the 1970s opened with computing at a critical but fragile point. Although produced by a large and established industry, commercial computers remained the expensive, relatively esoteric tools of large corporations, research institutions, and government. Computing had not yet made its way to the common user, much less the man in the street. This movement would begin in the mid-1970s with the introduction of the microprocessor and then unfold in the 1980s with even greater drama and force. If the era before 1960 was one of experimentation and the 1960s one of consolidation and diffusion in computing, the two decades between 1970 and 1990 were characterized by explosive growth. Still, this course of events was far from clear in the early 1970s.

Accomplishing Federal Missions

(From pp. 141-142): In addition to supporting industrial innovation and the economic benefits that it brings, federal support for computing research has enabled government agencies to accomplish their missions. Investments in computing research by the Department of Energy (DOE), the National Aeronautics and Space Administration (NASA), and the National Institutes of Health (NIH), as well as the Department of Defense (DOD), are ultimately based on agency needs. Many of the missions these agencies must fulfill depend on computing technologies. DOD, for example, has maintained a policy of achieving military superiority over potential adversaries not through numerical superiority (i.e., having more soldiers) but through better technology. Computing has become a central part of information gathering, management, and analysis for commanders and soldiers alike.

Similarly, DOE and its predecessors would have been unable to support their mission of designing nuclear weapons without the simulation capabilities of large supercomputers. Such computers have retained their value to DOE as its mission has shifted toward stewardship of the nuclear stockpile in an era of restricted nuclear testing. Its Accelerated Strategic Computing Initiative builds on DOE's earlier success by attempting to support development of simulation technologies needed to assess nuclear weapons, analyze their performance, predict their safety and reliability, and certify their functionality without testing them. In addition, NASA could not have accomplished its space exploration or its Earth observation and monitoring missions without reliable computers for controlling spacecraft and managing data. New computing capabilities, including the World Wide Web, have enabled the National Library of Medicine to expand access to medical information and have provided tools for researchers who are sequencing the human genome.

Copyright © National Academy of Sciences. All rights reserved.

EVOLVING THE HIGH PERFORMANCE COMPUTING AND COMMUNICATIONS INITIATIVE TO SUPPORT THE NATION'S INFORMATION INFRASTRUCTURE (1995)

CITATION: Computer Science and Telecommunications Board (CSTB), National Research Council. 1995. *Evolving the High Performance Computing and Communications Initiative to Support the Nation's Information Infrastructure.* National Academy Press, Washington, D.C.

Continued Federal Investment Is Necessary to Sustain Our Lead

(From pp. 23-25): What must be done to sustain the innovation and growth needed for enhancing the information infrastructure and maintaining U.S. leadership in information technology? Rapid and continuing change in the technology, a 10- to 15-year cycle from idea to commercial success, and successive waves of new companies are characteristics of the information industry that point to the need for a stable source of expertise and some room for a long-term approach. Three observations seem pertinent.

1. *Industrial R&D cannot replace government investment in basic research.* Very few companies are able to invest for a payoff that is 10 years away. Moreover, many advances are broad in their applicability and complex enough to take several engineering iterations to get right, and so the key insights become "public" and a single company cannot recoup the research investment. Public investment in research that creates a reservoir of new ideas and trained people is repaid many times over by jobs and taxes in the information industry, more innovation and productivity in other industries, and improvements in the daily lives of citizens. This investment is essential to maintain U.S. international competitiveness. . . .

Because of the long time scales involved in research, the full effect of decreasing investment in research may not be evident for a decade, but by then, it may be too late to reverse an erosion of research capability. Thus, even though many private-sector organizations that have weighed in on one or more policy areas relating to the enhancement of information infrastructure typically argue for a minimal government role in commercialization, they tend to support a continuing federal presence in relevant basic research.

2. *It is hard to predict which new ideas and approaches will succeed.* Over the years, federal support of computing and communications research in universities has helped make possible an environment for exploration and experimentation, leading to a broad range of diverse ideas from which the marketplace ultimately has selected winners and losers. . . . [I]t is

Copyright © National Academy of Sciences. All rights reserved.

Innovation in Information Technology
http://www.nap.edu/catalog/10795.html

difficult to know in advance the outcome or final value of a particular line of inquiry. But the history of development in computing and communications suggests that innovation arises from a diversity of ideas and some freedom to take a long-range view. It is notoriously difficult to place a specific value on the generation of knowledge and experience, but such benefits are much broader than sales of specific systems.

3. *Research and development in information technology can make good use of equipment that is 10 years in advance of current "commodity" practice.* When it is first used for research, such a piece of equipment is often a supercomputer. By the time that research makes its way to commercial use, computers of equal power are no longer expensive or rare. . . .

The large-scale systems problems presented both by massive parallelism and by massive information infrastructure are additional distinguishing characteristics of information systems R&D, because they imply a need for scale in the research effort itself. In principle, collaborative efforts might help to overcome the problem of attaining critical mass and scale, yet history suggests that there are relatively few collaborations in basic research within any industry, and purely industrial (and increasingly industry-university or industry-government) collaborations tend to disseminate results more slowly than university-based research.

The government-supported research program . . . is small compared to industrial R&D . . . but it constitutes a significant portion of the research component, and it is a critical factor because it supports the exploratory work that is difficult for industry to afford, allows the pursuit of ideas that may lead to success in unexpected ways, and nourishes the industry of the future, creating jobs and benefits for ourselves and our children. The industrial R&D investment, though larger in dollars, is different in nature: it focuses on the near term—increasingly so, as noted earlier—and is thus vulnerable to major opportunity costs. The increasing tendency to focus on the near term is affecting the body of the nation's overall R&D. Despite economic studies showing that the United States leads the world in reaping benefits from basic research, pressures in all sectors appear to be promoting a shift in universities toward near-term efforts, resulting in a decline in basic research even as a share of university research. Thus, a general reduction in support for basic research appears to be taking place.

It is critical to understand that there are dramatic new opportunities that still can be developed by fundamental research in information technology—opportunities on which the nation must capitalize. These include high-performance systems and applications for science and engineering; high-confidence systems for applications such as health care, law enforcement, and finance; building blocks for global-scale information utilities (e.g., electronic payment); interactive environments for applica-

Copyright © National Academy of Sciences. All rights reserved.

tions ranging from telemedicine to entertainment; improved user interfaces to allow the creation and use of ever more sophisticated applications by ever broader cross sections of the population; and the creation of the human capital on which the next generation's information industries will be based. Fundamental research in computing and communications is the key to unlocking the potential of these new applications.

How much federal research support is proper for the foreseeable future and to what aspects of information technology should it be devoted? Answering this question is part of a larger process of considering how to reorient overall federal spending on R&D from a context dominated by national security to one driven more by other economic and social goals. It is harder to achieve the kind of consensus needed to sustain federal research programs associated with these goals than it was under the national security aegis. Nevertheless, the fundamental rationale for federal programs remains:

> That R&D can enhance the nation's economic welfare is not, by itself, sufficient reason to justify a prominent role for the federal government in financing it. Economists have developed a further rationale for government subsidies. Their consensus is that most of the benefits of innovation accrue not to innovators but to consumers through products that are better or less expensive, or both. Because the benefits of technological progress are broadly shared, innovators lack the financial incentive to improve technologies as much as is socially desirable. Therefore, the government can improve the performance of the economy by adopting policies that facilitate and increase investments in research. [Linda R. Cohen and Roger G. Noll. 1994. "Privatizing Public Research," *Scientific American* 271(3): 73]

Copyright © National Academy of Sciences. All rights reserved.

What Is CSTB?

As a part of the National Research Council, the Computer Science and Telecommunications Board (CSTB) was established in 1986 to provide independent advice to the federal government on technical and public policy issues relating to computing and communications. Composed of leaders from industry and academia, CSTB conducts studies of critical national issues and makes recommendations to government, industry, and academia. CSTB also provides a neutral meeting ground for consideration of complex issues where resolution and action may be premature. It convenes discussions that bring together principals from the public and private sectors, assuring consideration of key perspectives. The majority of CSTB's work is requested by federal agencies and Congress, consistent with its National Academies context.

A pioneer in framing and analyzing Internet policy issues, CSTB is unique in its comprehensive scope and effective, interdisciplinary appraisal of technical, economic, social, and policy issues. Beginning with early work in computer and communications security, cyber-assurance and information systems trustworthiness have been a cross-cutting theme in CSTB's work. CSTB has produced several reports known as classics in the field, and it continues to address these topics as they grow in importance.

To do its work, CSTB draws on some of the best minds in the country and from around the world, inviting experts to participate in its projects as a public service. Studies are conducted by balanced committees without direct financial interests in the topics they are addressing. Those

Copyright © National Academy of Sciences. All rights reserved.

committees meet, confer electronically, and build analyses through their deliberations. Additional expertise is tapped in a rigorous process of review and critique, further enhancing the quality of CSTB reports. By engaging groups of principals, CSTB gets the facts and insights critical to assessing key issues.

The mission of CSTB is to

- *Respond to requests* from the government, nonprofit organizations, and private industry for advice on computer and telecommunications issues and from the government for advice on computer and telecommunications systems planning, utilization, and modernization;
- *Monitor and promote the health of the fields* of computer science and telecommunications, with attention to issues of human resources, information infrastructure, and societal impacts;
- *Initiate and conduct studies* involving computer science, technology, and telecommunications as critical resources; and
- *Foster interaction* among the disciplines underlying computing and telecommunications technologies and other fields, at large and within the National Academies.

CSTB projects address a diverse range of topics affected by the evolution of information technology. Recently completed reports include *Beyond Productivity: Information Technology, Innovation, and Creativity; Cybersecurity Today and Tomorrow: Pay Now or Pay Later; Youth, Pornography, and the Internet; Broadband: Bringing Home the Bits; The Digital Dilemma: Intellectual Property in the Information Age; IDs—Not That Easy: Questions About Nationwide Identity Systems; The Internet Under Crisis Conditions: Learning from September 11*; and *IT Roadmap to a Geospatial Future.* For further information about CSTB reports and active projects, see <http://cstb.org>.

Copyright © National Academy of Sciences. All rights reserved.

Issues and Techniques in Touch-Sensitive Tablet Input

William Buxton
Ralph Hill
Peter Rowley

Computer Systems Research Institute
University of Toronto
Toronto, Ontario
Canada M5S 1A4

(416) 978-6320

Abstract

Touch-sensitive tablets and their use in human-computer interaction are discussed. It is shown that such devices have some important properties that differentiate them from other input devices (such as mice and joysticks). The analysis serves two purposes: (1) it sheds light on touch tablets, and (2) it demonstrates how other devices might be approached. Three specific distinctions between touch tablets and one button mice are drawn. These concern the signaling of events, multiple point sensing and the use of templates. These distinctions are reinforced, and possible uses of touch tablets are illustrated, in an example application. Potential enhancements to touch tablets and other input devices are discussed, as are some inherent problems. The paper concludes with recommendations for future work.

CR Categories and Subject Descriptors: I.3.1 [**Computer Graphics**]: Hardware Architecture: Input Devices. I.3.6 [**Computer Graphics**]: Methodology and Techniques: Device Independence, Ergonomics, Interaction Techniques.

General Terms: Design, Human Factors.

Additional Keywords and Phrases: touch sensitive input devices.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1985 ACM 0-89791-166-0/85/007/0215 $00.75

1. Introduction

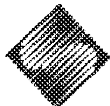
Increasingly, research in human-computer interaction is focusing on problems of input [Foley, Wallace & Chan 1984; Buxton 1983; Buxton 1985]. Much of this attention is directed towards input technologies. The ubiquitous Sholes keyboard is being replaced and/or complemented by alternative technologies. For example, a major focus of the marketing strategy for two recent personal computers, the Apple Macintosh and Hewlett-Packard 150, has been on the input devices that they employ (the mouse and touch-screen, respectively).

Now that the range of available devices is expanding, how does one select the best technology for a particular application? And once a technology is chosen, how can it be used most effectively? These questions are important, for as Buxton [1983] has argued, the ways in which the user *physically* interacts with an input device have a marked effect on the type of user interface that can be effectively supported.

In the general sense, the objective of this paper is to help in the selection process and assist in effective use of a specific class of devices. Our approach is to investigate a specific class of devices: touch-sensitive tablets. We will identify touch tablets, enumerate their important properties, and compare them to a more common input device, the mouse. We then go on to give examples of transactions where touch tablets can be used effectively. There are two intended benefits for this approach. First, the reader will acquire an understanding of touch tablet issues. Second, the reader will have a concrete example of how the technology can be investigated, and can utilize the approach as a model for investigating other classes of devices.

2. Touch-Sensitive Tablets

A touch-sensitive tablet (touch tablet for short) is a flat surface, usually mounted horizontally or nearly horizontally, that can sense the location of a finger pressing on it. That is, it is a tablet that can sense that it is being touched, and where it is being



touched. Touch tablets can vary greatly in size, from a few inches on a side to several feet on a side. The most critical requirement is that the user is not required point with some manually held device such as a stylus or puck.

What we have described in the previous paragraph is a *simple* touch tablet. Only one point of contact is sensed, and then only in a binary, touch/no touch, mode. One way to extend the potential of a simple touch tablet is to sense the degree, or pressure, of contact. Another is to sense multiple points of contact. In this case, the location (and possibly pressure) of several points of contact would be reported. Most tablets currently on the market are of the "simple" variety. However, Lee, Buxton and Smith [1985], and Nakatani [private communication] have developed prototypes of multi-touch, multi-pressure sensing tablets.

We wish to stress that we will restrict our discussion of touch technologies to touch tablets, which can and should be used in ways that are different from touch screens. Readers interested in touch-screen technology are referred to Herot & Weinsapfel [1978], Nakatani & Rohrlich [1983] and Minsky [1984]. We acknowledge that a flat touch screen mounted horizontally is a touch tablet as defined above. This is not a contradiction, as a touch screen has exactly the properties of touch tablets we describe below, as long as there is no attempt to mount a display below (or behind) it or to make it the center of the user's visual focus.

Some sources of touch tablets are listed in Appendix A.

3. Properties of Touch-Sensitive Tablets

Asking "Which input device is best?" is much like asking "How long should a piece of string be?" The answer to both is: it depends on what you want to use it for. With input devices, however, we are limited in our understanding of the relationship between device properties and the demands of a specific application. We will investigate touch tablets from the perspective of improving our understanding of this relationship. Our claim is that other technologies warrant similar, or even more detailed, investigation.

Touch tablets have a number of properties that distinguish them from other devices:

- They have no mechanical intermediate device (such as stylus or puck). Hence they are useful in hostile environments (e.g., classrooms, public access terminals) where such intermediate devices can get lost, stolen, or damaged.
- Having no puck to slide or get bumped, the tracking symbol "stays put" once placed, thus making them well suited for pointing tasks in environments subject to vibration or motion (e.g., factories, cockpits).
- They present no mechanical or kinesthetic restrictions on our ability to indicate more than one point at a time. That is, we can use two hands or more than one finger simultaneously on a single tablet. (Remember, we can manually control at most two mice at a time: one in each hand. Given that we have ten fingers, it is conceivable that we may wish to indicate more than two points simultaneously. An example of such an application appears below).
- Unlike joysticks and trackballs, they have a very low profile and can be integrated into other equipment such as desks and low-profile keyboards (e.g., the Key Tronic Touch Pad, see Appendix A). This has potential benefits in portable systems, and, according to the Keystroke model of Card, Newell and Moran [1980], reduces homing time from the keyboard to the pointing device.
- They can be molded into one-piece constructions thus eliminating cracks and grooves where dirt can collect. This makes them well suited for very clean environments (eg. hospitals) or very dirty ones (eg., factories).
- Their simple construction, with no moving parts, leads to reliable and long-lived operation, making them suitable for environments where they will be subjected to intense use or where reliability is critical.

They do, of course, have some inherent disadvantages, which will be discussed at the close of the paper.

In the next section we will make three important distinctions between touch tablets and mice. These are:

- Mice and touch tablets vary in the number and types of events that they can transmit. The difference is especially pronounced when comparing to simple touch tablets.
- Touch tablets can be made that can sense multiple points of contact. There is no analogous property for mice.
- The surface of a tablet can be partitioned into regions representing a collection of independent "virtual" devices. This is analogous to the partitioning of a screen into "windows" or virtual displays. Mice, and other devices that transmit "relative change" information, do not lend themselves to this mode of interaction without consuming display real estate for visual feedback. With conventional tablets and touch tablets, graphical, physical or virtual templates can be placed over the input device to delimit regions. This allows valuable screen real estate to be preserved. Physical templates, when combined with touch sensing, permit the operator to sense the regions without diverting the eyes from the primary display during visually demanding tasks.

After these properties are discussed, a simple finger painting program is used to illustrate them in the context of a concrete example. We wish to stress that we do not pretend that the program represents a viable paint program or an optimal interface. It is simply a vehicle to illustrate a variety of transactions in an easily understandable context.

Finally, we discuss improvements that must be made to current touch tablet technology, many of which we have demonstrated in prototype form. Also, we suggest potential improvements to other devices, motivated by our experience with touch technology.

4. Three Distinctions Between Touch Tablets and Mice[1]

The distinctions we make in this section have to do with suitability of devices for certain tasks or use in certain configurations. We are only interested in showing that there are some uses for which touch tablets are not suitable, but other devices are, and vice versa. We make no quantitative claims or comparisons regarding performance.

Signaling

Consider a rubber-band line drawing task with a one button mouse. The user would first position the tracking symbol at the desired starting point of the line by moving the mouse with the button released. The button would then be depressed, to signal the start of the line, and the user would manipulate the line by moving the mouse until the desired length and orientation was achieved. The completion of the line could then be signaled by releasing the button.[2]

Figure 1 is a state diagram that represents this interface. Notice that the button press and release are used to signal the beginning and end of the rubber-band drawing task. Also note that in states 1 and 2 both motion and signaling (by pressing or releasing the button, as appropriate) are possible.

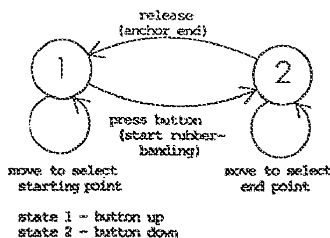


Figure 1. State diagram for rubber-banding with a one-button mouse.

Now consider a simple touch tablet. It can be used to position the tracking symbol at the starting point of the line, but it cannot generate the signal needed to initiate rubber-banding. Figure 2 is a state diagram representation of the capabilities of a simple touch tablet. In state 0, there is no contact with the tablet.[3] In this state only one action is possible: the user may touch the tablet. This causes a change to state 1. In state 1, the user is pressing on the tablet, and as a consequence position reports are sent to the host. There is no way to signal a change to some other state, other than to release (assuming the exclusion of temporal or spatial cues, which tend to be clumsy and difficult to learn). This returns the system to state 0. This signal could not be used to initiate rubber-banding, as it could also mean that the user is pausing to think, or wishes to initiate some other activity.

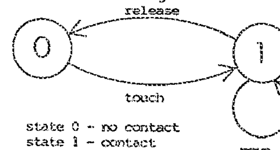


Figure 2. Diagram for showing states of simple touch-tablet.

This inability to signal while pointing is a severe limitation with current touch tablets, that is, tablets that do not report pressure in addition to location. (It is also a property of trackballs, and joysticks without "fire" buttons). It renders them unsuitable for use in many common interaction techniques for which mice are well adapted (e.g., selecting and dragging objects into position, rubber-band line drawing, and pop-up menu selection); techniques that are especially characteristic of interfaces based on *Direct Manipulation* [Shneiderman 1983].

One solution to the problem is to use a separate function button on the keyboard. However, this usually means two-handed input where one could do, or, awkward co-ordination in controlling the button and pointing device with a single hand. An alternative solution when using a touch tablet is to provide some level of pressure sensing. For example, if the tablet could report two levels of contact pressure (i.e., hard and soft), then the transition from soft to hard pressure, and vice versa, could be used for signaling. In effect, pressing hard is equivalent to pressing the button on the mouse. The state diagram showing the rubber-band line drawing task with this form of touch tablet is shown in Figure 3.[4]

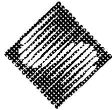
As an aside, using this pressure sensing scheme would permit us to select options from a menu, or

[1] Although we are comparing touch tablets to one button mice throughout this section, most of the comments apply equally to tablets with one-button pucks or (with some caveats) tablets with styli.

[2] This assumes that the interface is designed so that the button is held down during drawing. Alternatively, the button can be released during drawing, and pressed again, to signal the completion of the line.

[3] We use state 0 to represent a state in which no location, information is transmitted. There no analogous state for mice, and hence no state 0 in the diagrams for mice. With conventional tablets, this corresponds to "out of range" state.
At this point the alert reader will wonder about difficulty in distinguishing between hard and soft pressure, and friction (especially when pressing hard). Taking the last first, hard is a relative term. In practice friction need not be a problem (see Inherent Problems, below).

[4] One would conjecture that in the absence of button clicks or other feedback, pressure would be difficult to regulate accurately. We have found two levels of pressure to be easily distinguished, but this is a ripe area for research. For example, Stu Card [private communication] has suggested that the threshold between soft and hard should be reduced (become "softer") while hard pressure is being maintained. This suggestion, and others, warrant formal experimentation.



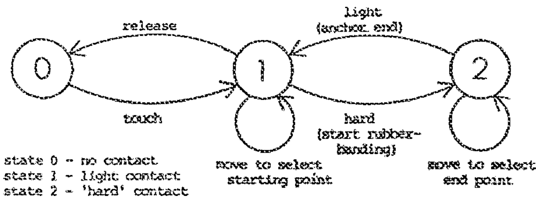


Figure 3. State diagram for rubber-banding with pressure sensing touch tablet.

activate light buttons by positioning the tracking symbol over the item and "pushing". This is consistent with the gesture used with a mouse, and the model of "pushing" buttons. With current simple touch tablets, one does just the opposite: position over the item and then lift off, or "pull" the button.

From the perspective of the signals sent to the host computer, this touch tablet is capable of duplicating the behaviour of a one-button mouse. This is not to say that these devices are equivalent or interchangeable. *They are not. They are physically and* kinesthetically very different, and should be used in ways that make use of the unique properties of each. Furthermore, such a touch tablet can generate one pair of signals that the one-button mouse cannot – specifically, press and release (transition to and from state 0 in the above diagrams). These signals (which are also available with many conventional tablets) are very useful in implementing certain types of transactions, such as those based on character recognition.

An obvious extension of the pressure sensing concept is to allow continuous pressure sensing. That is, pressure sensing where some large number of different levels of pressure may be reported. This extends the capability of the touch tablet beyond that of a traditional one button mouse. An example of the use of this feature is presented below.

Multiple Position Sensing

With a traditional mouse or tablet, only one position can be reported per device. One can imagine using two mice or possibly two transducers on a tablet, but this increases costs, and two is the practical limit on the number of mice or tablets that can be operated by a single user (without using feet). However, while we have only two hands, we have ten fingers. As playing the piano illustrates, there are some contexts where we might want to use several, or even all of them, at once.

Touch tablets need not restrict us in this regard. Given a large enough surface of the appropriate technology, one could use all fingers of both hands simultaneously, thus providing ten separate units of input. Clearly, this is well beyond the demands of many applications and the capacity of many people, however, there are exceptions. Examples include chording on buttons or switches, operating a set of slide potentiometers, and simple key roll-over when touch typing. One example (using a set of slide potentiometers) will be illustrated below.

Multiple Virtual Devices and Templates

The power of modern graphics displays has been enhanced by partitioning one physical display into a number of virtual displays. To support this, display window managers have been developed. We claim (see Brown, Buxton and Murtagh [1985]) that similar benefits can be gained by developing an input window manager that permits a single physical input device to be partitioned into a number of virtual input devices. Furthermore, we claim that multi-touch tablets are well suited to supporting this approach.

Figure 4a shows a thick cardboard sheet that has holes cut in specific places. When it is placed over a touch tablet as shown in Figure 4b, the user is restricted to touching only certain parts of the tablet. More importantly, the user can *feel* the parts that are touchable, and their shape. Each of the "touchable" regions represents a separate virtual device. The distinction between this template and traditional tablet mounted menus (such as seen in many CAD systems) is important.

Traditionally, the options have been:

a) Save display real estate by mounting the menu on the tablet surface. The cost of this option is eye diversion from the display to the tablet, the inability to "touch type", and time consuming menu changes.

b) Avoid eye diversion by placing the menus on the display. This also make it easier to change menus, but still does not allow "touch typing", and consumes display space.

Touch tablets allow a new option:

c) Save display space and avoid eye diversion by using templates that can be felt, and hence, allow "touch typing" on a variety of virtual input devices. The cost of this option is time consuming menu (template) changes.

It must be remembered that for each of these options, there is an application for which it is best. We have contributed a new option, which makes possible new interfaces. The new possibilities include more elaborate virtual devices because the improved kinesthetic feedback allows the user to concentrate on providing input, instead of staying in the assigned region. We will also show (below) that its main cost (time consuming menu changes) can be reduced in some applications by eliminating the templates.

5. Examples of Transactions Where Touch Tablets Can Be Used Effectively

In order to reinforce the distinctions discussed in the previous section, and to demonstrate the use of touch tablets, we will now work through some examples based on a toy paint system. We wish to stress again that we make no claims about the quality of the example as a paint system. A paint system is a common and easily understood application, and thus, we have chosen to use it simply as a vehicle for discussing interaction techniques that use touch tablets.

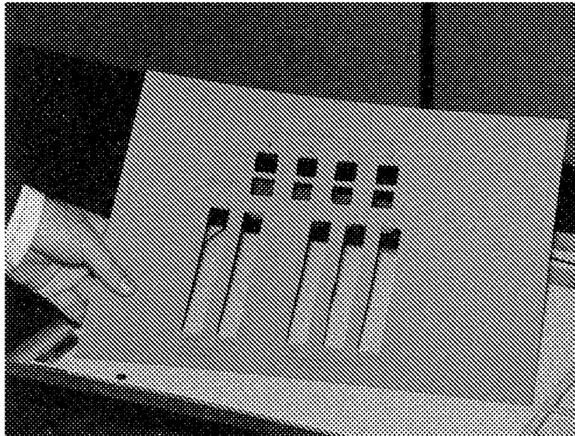


Figure 4a. Sample template.

Figure 5. Main display for paint program.

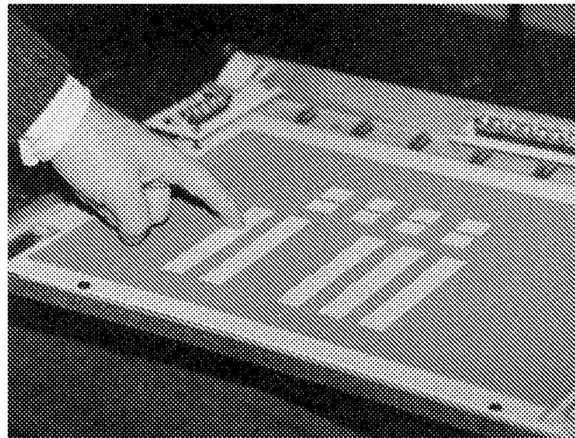


Figure 4b. Sample template in use.

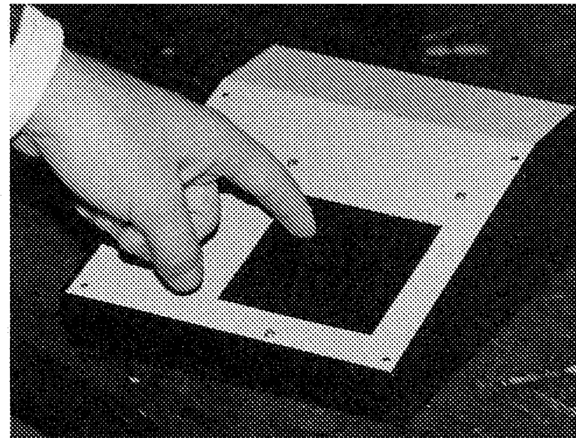


Figure 6. Touch tablet used in demonstrations.

The example paint program allows the creation of simple finger paintings. The layout of the main display for the program is shown in Figure 5. On the left is a large drawing area where the user can draw simple free-hand figures. On the right is a set of menu items. When the lowest item is selected, the user enters a colour mixing mode. In switching to this mode, the user is presented with a different display that is discussed below. The remaining menu items are "paint pots". They are used to select the colour that the user will be painting with.

In each of the following versions of the program, the input requirements are slightly different. In all cases an 8 cm x 8 cm touch tablet is used (Figure 6), but the pressure sensing requirements vary. These are noted in each demonstration.

5.1. Painting Without Pressure Sensing

This version of the paint program illustrates the limitation of having no pressure sensing. Consider the paint program described above, where the only input device is a touch tablet without pressure sensing. Menu selections could be made by pressing down somewhere in the menu area, moving the tracking symbol to the desired menu item and then selecting by releasing. To paint, the user would simply press down in the drawing area and move (see Figure 7 for a representation of the signals used for painting with this program).

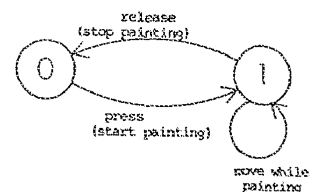


Figure 7. State diagram for drawing portion of simple paint program.

There are several problems with this program. The most obvious is in trying to do detailed drawings. The user does not know where the paint will appear until it appears. This is likely to be too late. Some form of feedback, that shows the user where the brush is, without painting, is needed. Unfortunately, this cannot be done with this input device, as it is not possible to signal the change from tracking to painting and vice versa.

The simplest solution to this problem is to use a button (e.g., a function key on the keyboard) to signal state changes. The problem with this solution is the need to use two hands on two different devices to do one task. This is awkward and requires practice to develop the co-ordination needed to make small rapid strokes in the painting. It is also inefficient in its use of two hands where one could (and normally should) do.

Alternatively, approaches using multiple taps or timing cues for signalling could be tried, however, we have found that these invariably lead to other problems. It is better to find a direct solution using the properties of the device itself.

5.2. Painting with Two Levels of Pressure

This version of the program uses a tablet that reports two levels of contact pressure to provide a satisfactory solution to the signaling problem. A low pressure level (a light touch by the user) is used for general tracking. A heavier touch is used to make menu selections, or to enable painting (see Figure 8 for the tablet states used to control painting with this program). The two levels of contact pressure allow us to make a simple but practical one finger paint program.

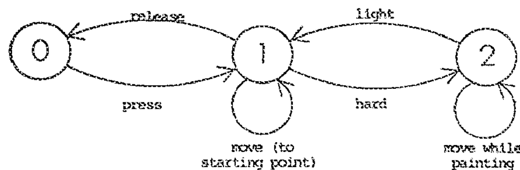


Figure 8. State diagram for painting portion of simple paint program using pressure sensing touch tablet.

This version is very much like using the one button mouse on the Apple Macintosh with MacPaint [Williams, 1984]. Thus, a simple touch tablet is not very useful, but one that reports two levels of pressure is similar in power (but not feel or applicability) to a one button mouse.[5]

5.3. Painting with Continuous Pressure Sensing

In the previous demonstrations, we have only implemented interaction techniques that are common using existing technology. We now introduce a technique that provides functionality beyond that obtainable using most conventional input technologies.

[5] Also, there is the problem of friction, to be discussed below under "Inherent Problems".

In this technique, we utilize a tablet capable of sensing a continuous range of touch pressure. With this additional signal, the user can control both the width of the paint trail and its path, using only one finger. The new signal, pressure, is used to control width. This is a technique that cannot be used with any mouse that we are aware of, and to our knowledge, is available on only one conventional tablet (the GTCO Digipad with pressure pen [GTCO 1982]).

We have found that using current pressure sensing tablets, the user can accurately supply two to three bits of pressure information, after about 15 minutes practice. This is sufficient for simple doodling and many other applications, but improved pressure resolution is required for high quality painting.

5.4. "Windows" on the Tablet: Colour Selection

We now demonstrate how the surface of the touch tablet can be *dynamically* partitioned into "windows" onto virtual input devices. We use the same basic techniques as discussed under templates (above), but show how to use them without templates. We do this in the context of a colour selection module for our paint program. This module introduces a new display, shown in Figure 9.

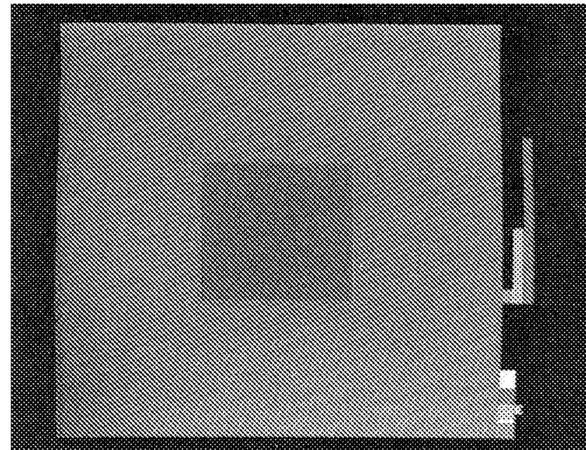


Figure 9. Colour mixing display.

In this display, the large left side consists of a colour patch surrounded by a neutral grey border. This is the patch of colour the user is working on. The right side of the display contains three bar graphs with two light buttons underneath. The primary function of the bar graphs is to provide feedback, representing relative proportions of red, green and blue in the colour patch. Along with the light buttons below, they also serve to remind the user of the current layout of the touch tablet.

In this module, the touch tablet is used as a "virtual operating console". Its layout is shown (to scale) in Figure 10. There are 3 valuators (corresponding to the bar graphs on the screen) used to control

colour, and two buttons: one, on the right, to bring up a pop-up menu used to select the colour to be modified, and another, on the left, to exit.

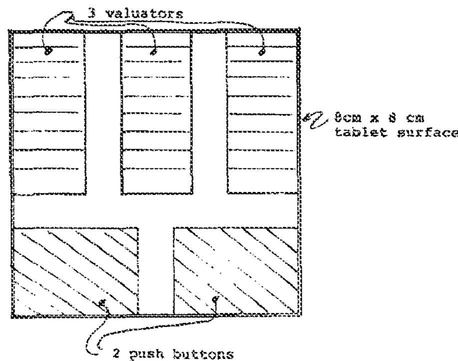


Figure 10. Layout of virtual devices on touch tablet.

The single most important point to be made in this example is that a single *physical* device is being used to implement 5 *virtual* devices (3 valuators and 2 buttons). This is analogous to the use of a display window system, in its goals, and its implementation.

The second main point is that there is nothing on the tablet to delimit the regions. This differs from the use of physical templates as previously discussed, and shows how, in the absence of the need for a physical template, we can instantly change the "windows" on the tablet, without sacrificing the ability to touch type.

We have found that when the tablet surface is small, and the partioning of the surfaces is not too complex, the users very quickly (typically in one or two minutes) learn the positions of the virtual devices relative to the edges of the tablet. More importantly, they can use the virtual devices, practically error free, without diverting attention from the display. (We have repeatedly observed this behaviour in the use of an application that uses a 10 cm square tablet that is divided into 3 sliders with a single button across the top).

Because no template is needed, there is no need for the user to pause to change a template when entering the colour mixing module. Also, at no point is the user's attention diverted from the display. These advantages cannot be achieved with any other device we know of, without consuming display real estate.

The colour of the colour patch is manipulated by *dragging* the red, green and blue values up and down with the valuators on the touch tablet. The valuators are implemented in relative mode (i.e., they are sensitive to changes in position, not absolute position), and are manipulated like one dimensional mice. For example, to make the patch more red, the user presses near the left side of the tablet, about half way to the top, and slides the finger up (see Figure 11). For larger changes, the device can be repeatedly stroked (much like stroking a mouse). Feedback is provided by changing the level in the bar graph on the screen and the colour of the patch.

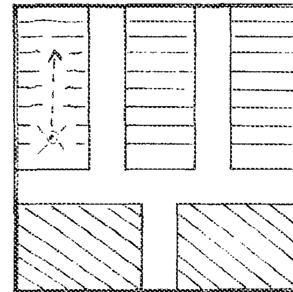


Figure 11. Increasing red content, by pressing on red valuator and sliding up.

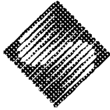
Using a mouse, the above interaction could be approximated by placing the tracking symbol over the bars of colour, and dragging them up or down. However, if the bars are narrow, this takes acuity and concentration that distracts attention from the primary task — monitoring the colour of the patch. Furthermore, note that the touch tablet implementation does not need the bars to be displayed at all, they are only a convenience to the user. There are interfaces where, in the interests of maximizing available display area, there will be no items on the display analogous to these bars. That is, there would be nothing on the display to support an interaction technique that allows values to be manipulated by a mouse.

Finally, we can take the example one step further by introducing the use of a touch tablet that can sense multiple points of contact (e.g., [Lee, et al. 1985]). With this technology, all three colour values could be changed at the same time (for example, fading to black by drawing all three sliders down together with three fingers of one hand). This simultaneous adjustment of colours could *not* be supported by a mouse, nor any single commercially available input device we know of. Controlling several valuators with one hand is common in many operating consoles, for example: studio light control, audio mixers, and throttles for multi-engine vehicles (e.g., aircraft and boats). Hence, this example demonstrates a cost effective method for providing functionality that is currently unavailable (or available only at great cost, in the form of a custom fabricated console), but has wide applicability.

5.5. Summary of Examples

Through these simple examples, we have demonstrated several things:

- The ability to sense at least two levels of pressure is a virtual necessity for touch tablets, as without it, auxiliary devices must be used for signaling, and "direct manipulation" interfaces cannot be effectively supported.
- The extension to continuous pressure sensing opens up new possibilities in human-computer interaction.



- Touch tablets are superior to mice and tablets when many simple devices are to be simulated. This is because: (a) there is no need for a mechanical intermediary between the fingers and the tablet surface, (b) they allow the use of templates (including the edges of the tablet, which is a trivial but useful template), and (c) there is no need for positional feedback that would consume valuable display space.
- The ability to sense multiple points of contact radically changes the way in which users may interact with the system. The concept of multiple points of contact does not exist for, nor is it applicable to, current commercially available mice and tablets.

6. Inherent Problems with Touch Tablets

A problem with touch tablets that is annoying in the long term is friction between the user's finger and the tablet surface. This can be a particularly severe problem if a pressure sensitive tablet is used, and the user must make long motions at high pressure. This problem can be alleviated by careful selection of materials and care in the fabrication and calibration of the tablet.[6] Also, the user interface can be designed to avoid extended periods of high pressure.

Perhaps the most difficult problem is providing good feedback to the user when using touch tablets. For example, if a set of push-on/push-off buttons are being simulated, the traditional forms of feedback (illuminated buttons or different button heights) cannot be used. Also, buttons and other controls implemented on touch tablets lack the kinesthetic feel associated with real switches and knobs. As a result, users must be more attentive to visual and audio feedback, and interface designers must be freer in providing this feedback. (As an example of how this might be encouraged, the input "window manager" could automatically provide audible clicks as feedback for button presses).

7. Potential Enhancements to Touch Tablets (and other devices)

The first problem that one notices when using touch tablets is "jitter" when the finger is removed from the tablet. That is, the last few locations reported by the tablet, before it senses loss of contact, tend to be very unreliable.

This problem can be eliminated by modifying the firmware of the touch tablet controller so that it keeps a short FIFO queue of the samples that have most recently be sent to the host. When the user releases pressure, the oldest sample is re-transmitted, and the queue is emptied. The length of the queue depends on the properties of the touch tablet (e.g., sensitivity, sampling rate). We have found that determining a suitable value requires only a few minutes of experimentation.

A related problem with most current tablet controllers (not just touch tablets) is that they do not inform the host computer when the user has ceased pressing on the tablet (or moved the puck out of range). This information is essential to the development of certain types of interfaces. (As already mentioned, this signal is not available from mice). Currently, one is reduced to deducing this event by timing the interval between samples sent by the tablet. Since the tablet controller can easily determine when pressure is removed (and must if it is to apply a de-jittering algorithm as above), it should share this information with the host.

Clearly, pressure sensing is an area open to development. Two pressure sensitive tablets have been developed at the University of Toronto [Sasaki, et al. 1981; Lee, et al. 1985]. One has been used to develop several experimental interfaces and was found to be a very powerful tool. They have recently become available from Elographics and Big Briar (see Appendix A). Pressure sensing is not only for touch tablets. Mice, tablet pucks and styli could all benefit by augmenting switches with strain gauges, or other pressure sensing instruments. GTCO, for example, manufactures a stylus with a pressure sensing tip [GTCO 1982], and this, like our pressure sensing touch tablets, has proven very useful.

8. Conclusions

We have shown that there are environments for which some devices are better adapted than others. In particular, touch tablets have advantages in many hostile environments. For this reason, we suggest that there are environments and applications where touch tablets may be the most appropriate input technology.

This being the case, we have enumerated three major distinctions between touch tablets and one button mice (although similar distinctions exist for multi-button mice and conventional tablets). These assist in identifying environments and applications where touch tablets would be most appropriate. These distinctions concern:

- limitation in the ability to signal events,
- suitability for multiple point sensing, and
- the applicability of tactile templates.

These distinctions have been reinforced, and some suggestions on how touch tablets may be used have been given, by discussing a simple user interface. From this example, and the discussion of the distinctions, we have identified some enhancements that can be made to touch tablets and other input devices. The most important of these are pressure sensing and the ability to sense multiple points of contact.

We hope that this paper motivates interface designers to consider the use of touch tablets and shows some ways to use them effectively. Also, we hope it encourages designers and manufacturers of input devices to develop and market input devices with the enhancements that we have discussed.

[6] As a bad example, one commercial "touch" tablet requires so much pressure for reliable sensing that the finger cannot be smoothly dragged across the surface. Instead, a wooden or plastic stylus must be used, thus loosing many of the advantages of touch sensing.

The challenge for the future is to develop touch tablets that sense continuous pressure at multiple points of contact and incorporate them in practical interfaces. We believe that we have shown that this is worthwhile and have shown some practical ways to use touch tablets. However, interface designers must still do a great deal of work to determine where a mouse is better than a touch tablet and vice versa.

Finally, we have illustrated, by example, an approach to the study of input devices, summarized by the credo: "Know the interactions a device is intended to participate in, and the strengths and weaknesses of the device." This approach stresses that there is no such thing as a "good input device," only good interaction task/device combinations.

9. Acknowledgements

The support of this research by the Natural Sciences and Engineering Research Council of Canada is gratefully acknowledged. We are indebted to Kevin Murtagh and Ed Brown for their work on virtual input devices and windowing on input. Also, we are indebted to Elographics Corporation for having supplied us with the hardware on which some of the underlying studies are based.

We would like to thank the referees who provided many useful comments that have helped us with the presentation.

10. References

Brown, E., Buxton, W., Murtagh, K. 1985 — **Windows on Tablets as a Means of Achieving Virtual Input Devices.** Submitted for publication.

Buxton, W. 1983 — **Lexical and Pragmatic Considerations of Input Structures.** *Computer Graphics* 17.1. Presented at the SIGGRAPH Workshop on Graphical Input Techniques, Seattle, Washington, June 1982.

Buxton, W. 1985 — **There is More to Interaction Than Meets the Eye: Some Issues in Manual Input.** (in) Norman, D.A. and Draper, S.W. (Eds.), *User Centered System Design: New Perspectives on Human-Computer Interaction.* Hillsdale, N.J.: Lawrence Erlbaum and Associates. Publication expected late 1985.

Buxton, W., Fiume, E., Hill, R., Lee, A., Woo, C. 1983 — **Continuous Hand-Gesture Driven Input.** *Proceedings Graphics Interface '83*: pp. 191-195. May 9-13, 1983, Edmonton, Alberta.

Card, S.K., Moran, T.P., Newell, A. Jul 1980 — **The Keystroke-Level Model for User Performance Time with Interactive Systems.** *Communications of the ACM* 23.7: pp. 396-409.

Foley, J.D., Wallace, V.L., Chan, P. Nov 1984 — **The Human Factors of Computer Graphics Interaction Techniques.** *IEEE Computer Graphics and Applications* 4.11: pp. 13-48.

GTCO 1982 — **DIGI-PAD 5 User's Manual.** GTCO Corporation, 1055 First Street, Rockville, MD 20850.

Herot, C.F., Weinzapfel, G. Aug 1978 — **One-Point Touch Input of Vector Information for Computer Displays.** *Computer Graphics* 12.3: pp. 210-216. SIGGRAPH'78 Conference Proceedings, August 23-25, 1978, Atlanta, Georgia.

Lee, S., Buxton, W., Smith, K.C. 1985 — **A Multi-Touch Three Dimensional Touch-Sensitive Tablet.** *Human Factors in Computer Systems*: pp. 21-25. (CHI'85 Conference Proceedings, April 14-18, 1985, San Fransisco).

Minsky, M.R. Jul 1984 — **Manipulating Simulated Objects with Real-world Gestures using a Force and Position Sensitive Screen.** *Computer Graphics* 18.3: pp. 195-203. (SIGGRAPH'84 Conference Proceedings, July 23-27, 1984, Minneapolis, Minnesota).

Nakatani, L.H., Rohrlich, J.A. Dec 1983 — **Soft Machines: A Philosophy of User-Computer Interface Design.** *Human Factors in Computing Systems*: pp. 19-23. (CHI'83 Conference Procedings, December 12-15, 1983, Boston).

Sasaki, L., Fedorkow, G., Buxton, W., Retterath, C., Smith, K.C. 1981 — **A Touch Sensitive Input Device.** *Proceedings of the 5th International Conference on Computer Music.* North Texas State University, Denton Texas, November 1981.

Shneiderman, B. Aug 1983 — **Direct Manipulation: A Step Beyond Programming Languages.** *Computer* 16.8: pp. 57-69.

Williams, G. Feb 1984 — **The Apple Macintosh Computer.** *Byte* 9.2: pp. 30-54.

Appendix A: Touch Tablet Sources

Big Briar: 3 by 3 inch continuous pressure sensing touch tablet

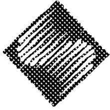
Big Briar, Inc.
Leicester, NC
28748

Chalk Board Inc.: "Power Pad", large touch table for micro-computers

Chalk Board Inc.
3772 Pleasantdale Rd.,
Atlanta, GA 30340

Elographics: various sizes of touch tablets, including pressure sensing

Elographics, Inc.
105 Randolph Toad
Oak Ridge, Tennessee
37830
(615)-482-4100



Key Tronic: Keyboard with touch pad.

Keytronic
P.O. Box 14687
Spokane, WA 99214
(509)-928-8000

KoalaPad Technologies: Approx. 5 by 7 inch touch tablet for micro-computers

Koala Technologies
3100 Patrick Henry Drive
Santa Clara, California
95050

Spiral Systems: Trazor Touch Panel, 3 by 3 inch touch tablet

Spiral System Instruments, Inc.
4853 Cordell Avenue, Suite A-10
Bethesda, Maryland
20814

TASA: 4 by 4 inch touch tablet (relative sensing only)

Touch Activated Switch Arrays Inc.
1270 Lawrence Stn. Road, Suite G
Sunnyvale, California
94089

Lexical and Pragmatic Considerations of Input Structures

William Buxton
Computer Systems Research Group
University of Toronto
Toronto, Ontario
Canada M5S 1A4

Introduction

Increased access to computer-based tools has made only too clear the deficiencies in our ability to produce effective user interfaces [1]. Many of our current problems are rooted in our lack of sufficiently powerful theories and methodologies. User interface design remains more of a creative art than a hard science.

Following an age-old technique, the point of departure for much recent work has been to attempt to impose some structure on the problem domain. Perhaps the most significant difference between this work and earlier efforts is the weight placed on considerations falling outside the scope of conventional computer science. The traditional problem-reduction paradigm is being replaced by a holistic approach which views the problem as an integration of issues from computer science, electrical engineering, industrial design, cognitive psychology, psychophysics, linguistics, and kinesthetics.

In the main body of this paper, we examine some of the taxonomies which have been proposed and illustrate how they can serve as useful structures for relating studies in user interface problems. In so doing, we attempt to augment the power of these structures by developing their ability to take into account the effect of gestural and positional factors on the overall effect of the user interface.

Two Taxonomies

One structure for viewing the problem domain of the user interface is provided by Foley and Van Dam [12]. They describe the space in terms of the following four layers:

- conceptual
- semantic
- syntactic
- lexical

The *conceptual* level incorporates the main concepts of the system as seen by the user. Therefore, Foley and Van Dam see it as being equivalent to the *user model.* The *semantic* level incorporates the functionality of the system: what can be expressed. The *syntactic* level defines the grammatical structure of the tokens used to articulate a semantic concept. Finally, the *lexical* component defines the structure of these tokens.

One of the benefits of such a taxonomy is that it can serve as the basis for systems analysis in the design process. It also helps us categorize various user interface studies so as to avoid "apples and bananas" type of comparisons. For example, the studies of Ledgard, Whiteside, Singer and Seymour [16] and Barnard, Hammond, Morton and Long [3] both address issues at the syntactic level. They can, therefore, be compared (which is quite interesting since they give highly contradictory results[1]). On the other hand, by recognizing the "keystroke" model of Card, Moran and Newell [7] as addressing the lexical level, we have a good way of understanding its limitations and comparing it to related studies (such as Embley, Lan, Leinbaugh and Nagy, [8]), or relating it to studies which address different levels (such as the two studies in syntax mentioned above).

While the taxonomy presented by Foley and Van Dam has proven to be a useful tool, our opinion is that it has one major shortcoming. That is, the grain of the lexical level is too coarse to permit the full benefit of the model to be derived. As defined, the authors lump together issues as diverse as:

- how tokens are spelt (for example "add" *vs* "append" *vs* "a" *vs* some graphical icon)

[1] Barnard *et al* invalidate Ledgard *et al's* main thesis that the syntax of natural language is necessarily the best suited for command languages. They demonstrate cases where fixed-field format is less prone to user error than the direct object — indirect object syntax of natural language. A major problem of the paper of Ledgard *et al* is that they did not test many of the interesting cases and then drew conclusions that went beyond what their results supported.

- where items are placed spatially on the display (both in terms of the layout and number of windows, and the layout of data within those windows)
- where devices are placed in the work station
- the type of physical gesture (as determined by the transducer employed) used to articulate a token (pointing with a joystick *vs* a lightpen *vs* a tablet *vs* a mouse, for example)

These issues are sufficiently different to warrant separate treatment. Grouping them under a single heading has the danger of generating confusion comparable to that which could result if no difference was made between the semantic and syntactic levels. Therefore, taking our cue from work in language understanding research in the AI community, we chose to subdivide Foley and Van Dam's lexical level into the following two components:

- lexical: issues having to do with spelling of tokens (*i.e.*, the ordering of lexemes and the nature of the alphabet used — symbolic or iconic, for example).
- pragmatic: issues of gesture, space and devices.

To illustrate the distinction, in the Keystroke model the number of key pushes would be a function of the *lexical* structure while the homing time and pointing time would be a function of *pragmatics*.

Factoring out these two levels helps us focus on the fact that the issues affecting each are different, as is their influence on the overall effect of the user interface. This is illustrated in examples which are presented later in this paper.

It should be pointed out that our isolation of what we have called pragmatic issues is not especially original. We see a similar view in the Command Language Grammar of Moran [18], which is the second main taxonomy which we present. Moran represents the domain of the user interface in terms of three components, each of which is sub-divided into two levels. These are as follows:

- Conceptual Component
  - task level
  - semantic level
- Communication Component
  - syntactic level
  - interaction level
- Physical Component
  - spatial level
  - device level

The *task level* encompasses the set of tasks which the user brings to the system and for which it is intended to serve as a tool. The *semantic level* lays out the conceptual entities of the system and the conceptual operations upon them. As with the Foley and Van Dam model, the *syntactic level* then incorporates the structure of the language within which the semantic level is embedded. The *interaction level* relates the user's physical actions to the conventions of the interactions in the dialogue. The *spatial level* then encompasses issues related to how information is laid out on the display, while the *device level* covers issues such as what types of devices are used and their properties (for example, the effect on user performance if the locator used is a mouse *vs* an isometric joystick *vs* step-keys). (A representative discussion of such issues can be found in Card, English and Burr, [5].)

One subtle but important emphasis in Moran's paper is on the point that it is the effect of the user interface *as a whole* (that is, all levels combined) which constitutes the user's model. The other main difference of his taxonomy, when compared to that of Foley and Van Dam, is his emphasis on the importance of the physical component. A shortcoming, however, lies in the absence of a slot which encapsulates the lexical level as we have defined it above. Like the lexical level (as defined by Foley and Van Dam), the interaction level of Moran appears a little too broad in scope when compared to the other levels in the taxonomy.

Pragmatics

In examining the two studies discussed above, one quickly recognizes that the effect of the pragmatic level on the user interface, and therefore on the user model, is given very little attention. Moran, for example, points out that the physical component exists and that it is important, but does not discuss it further. Foley and Van Dam bury these issues within the lexical level. Our main thesis is that since the primary level of contact with an interactive system is at the level of pragmatics, this level *has one of the strongest effects on the user's perception of the system*. Consequently, the models which we adopt in order to specify, design, implement, compare and evaluate interactive systems *must* be sufficiently rich to capture and communicate the system's properties at this level. This is clearly not the case with most models, and this should be cause for concern. To illustrate this, let us examine a few case studies which relate the effect of pragmatics to:

- pencil-and-paper tests of query languages
- ease of use with respect to action language grammars
- device independence

Pencil-and-Paper Tests

As an aid to the design of effective data base query languages, Reisner [19] has proposed the use of pencil-and-paper tests. Subjects were taught a query language in a class-room environment and then tested as to their

ability to formulate and understand queries. Different control groups were taught different languages. By comparing the test results of the different groups, Reisner drew conclusions as to the relative "goodness" of structure and ease of learning of the different languages. She then made the argument that the technique could be used to find weaknesses in new languages before they are implemented, thereby shortening their development cycle.

While the paper makes some important points, it has a serious defect in that it does not point out the limitations of the technique. The approach does tell us something about the cognitive burden involved in the learning of a query language. But it does not tell us everything. In particular, the technique is totally incapable of taking into account the effect that the means and medium of doing something has on our ability to remember how to do it. To paraphrase McLuhan, the medium does affect the message.

Issues of syntax are not independent of pragmatics, but pencil-and-paper tests cannot take such dependencies into account. For example, consider the role of "muscle memory" in recalling how to perform various tasks. The strength of its influence can be seen in my ability to type quite effectively, even though I am incapable of telling you where the various characters are on my QWERTY keyboard, or in my ability to open a lock whose combination I cannot recite. Yet, this effect will never show up in a pencil-and-paper test. Another example is seen in the technique's inability to take into account the contribution that appropriate feedback and help mechanisms can provide in developing mnemonics and other memory and learning aids.

We are not trying to claim that such pencil-and-paper tests are not of use (although Barnard *et al*, [3], point out some important dangers in using such techniques). We are simply trying to illustrate some of their limitations, and demonstrate that lack of adequate emphasis on pragmatics can result in readers (and authors) drawing false or misleading conclusions from their work. Furthermore, we conjecture that if pragmatics were isolated as a separate level in a taxonomy such as that of Foley and Van Dam, they would be less likely to be ignored.

Complexity and Chunking

In another study, Reisner [20] makes an important contribution by showing how the analysis of the grammar of the "action language" of an interactive system can provide valuable metrics for predicting the ease of use and proneness to error of that system. Thus, an important tool for system design, analysis and comparison is introduced.

The basis of the technique is that the complexity of the grammar is a good metric for the cognitive burden of learning and using the system. Grammar complexity is measured in terms of number of productions and production length. There is a problem, however, which limits our ability to reap the full benefits of the technique. This has to do with the technique's current inability to take into account what we call *chunking*. By this we mean the phenomenon where two or more actions fuse together into a single gesture (in a manner analogous to the formation of a compound word in language). In many cases, the cognitive burden of the resulting aggregate may be the equivalent of a single token. In terms of formal language theory, a non-terminal *when effected by an appropriate compound gesture* may carry the cognitive burden of a single terminal.

Such chunking may be either sequential, parallel or both. Sequentially, it should be recognized that some actions have different degrees of *closure* than others. For example, take two events, each of which is to be triggered by the change of state of a switch. If a foot-switch similar to the high/low beam switch in some cars is used, the down action of a down/up gesture triggers each event. The point to note is that there is no kinesthetic connection between the gesture that triggers one event and that which triggers the other. Each action is complete in itself and, as with driving a car, the operator is free to initiate other actions before changing the state of the switch again.

On the other hand, the same binary function could be controlled by a foot pedal which functions like the sustain pedal of a piano. In this case, one state change occurs on depression, a second on release. Here, the point to recognize is that the second action is a direct consequent of its predecessor. The syntax is implicit, and the cognitive burden of remembering what to do after the first action is minimal.

There are many cases where this type of kinesthetic connectivity can be bound to a sequence of tokens which are logically connected. One example given by Buxton [4] is in selecting an item from a graphics menu and "dragging" it into position in a work space. A button-down action (while pointing at an item) "picks it up." For as long as the button is depressed, the item tracks the motion of the pointing device. When the button is released, the item is anchored in its current position. Hence, the interface is designed to force the user to follow proper syntax: select then position. There is no possibility for syntactic error, and cognitive resources are not consumed in trying to remember "what do I do next?". Thus, by recognizing and exploiting such cases, interfaces can be constructed which are "natural" and easy to learn.

There is a similar type of chunking which can take place when two or more gestures are articulated at one time. Again we can take an example from driving a car, where in changing gears the actions on the clutch, accelerator and gear-shift reinforce one another and are coordinated into a single gesture. Choosing appropriate gestures for such coordinated actions can accelerate their bonding into what the user thinks of as a single act,

thereby freeing up cognitive resources to be applied to more important tasks. What we are arguing here is that by matching appropriate gestures with tasks, we can help render complex skills routine and gain benefits similar to those seen at different level in Card, Moran and Newell [6].

In summary, there are three main points which we wish to make with this example:

- there is an important interplay between the syntactic-lexical levels and the pragmatic level
- that this interplay can be exploited to reduce the cognitive burden of learning and using a system
- that this cannot be accomplished without a better understanding of pragmatic issues such as chunking and closure.

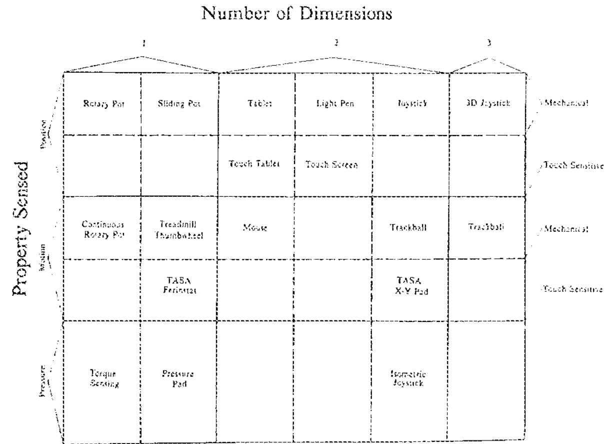
Pragmatics and Device Independence

We began by declaring the importance of being able to incorporate pragmatic issues into the models which we use to specify, design, compare and evaluate systems. The examples which followed then illustrated some of the reasons for this belief. When we view the CORE proposal [13, 14] from this perspective, however, we see several problems. The basis of how the CORE system approaches input is to deal with user actions in terms of abstractions, or logical devices (such as "locators" and "valuators"). The intention is to facilitate software portability. If all "locators," for example, utilized a common protocol, then user A (who only had a mouse) could easily implement software developed by B (who only had a tablet). From the application programmer's perspective, this is a valuable feature. However, for the purposes of specifying systems from the user's point of view, these abstractions are of very limited benefit. As Baecker [2] has pointed out, the effectiveness of a particular user interface is often due to the use of a *particular* device, and that effectiveness will be lost if that device were replaced by some other of the same logical class. For example, we have a system [10] whose interface depends on the simultaneous manipulation of four joysticks. Now in spite of tablets and joysticks both being "locator" devices, it is clear that they are not interchangeable in this situation. We cannot simultaneously manipulate four tablets. Thus, for the full potential of device independence to be realized, such pragmatic considerations must be incorporated into our overall specification model so that appropriate equivalencies can be determined in a methodological way. (That is, in specifying a generic device, we must also include the required pragmatic attributes. But to do so, we must develop a taxonomy of such attributes, just as we have developed a taxonomy of virtual devices.)

A Taxonomy of Devices

In view of the preceding discussion, we have attempted to develop a taxonomy which helps isolate relevant characteristics of input devices. The tableau shown in Figure 1 summarizes this effort in a two dimensional representation. The remainder of this section presents the details and motivation for this tableau's organization.

Figure 1. Tableau of Continuous Input Devices



To begin with, the tableau deals only with continuous hand-controlled devices. (Pedals, for example, are not included for simplicity's sake.) Therefore the first (but implicit) questions in our structure are:

- continuous *vs* discrete?
- agent of control (hand, foot, voice, ...)?

The table is divided into a matrix whose rows and columns delimit

- what is being sensed (position, motion or pressure), and
- the number of dimensions being sensed (1, 2 or 3),

respectively. These primary partitions of the matrix are delimited by solid lines. Hence, both the rotary and sliding potentiometer fall into the box associated with one-dimensional position-sensitive devices (top left-hand corner).

Note that the primary rows and columns of the matrix are sub-divided, as indicated by the dotted lines. The sub-columns exist to isolate devices whose control motion is roughly similar. These groupings can be seen in examining the two-dimensional devices. Here the tableau implies that tablets and mice utilize similar types of hand control and that this control is different from that shared in using a light-pen or touch-screen. Furthermore, it is shown that joysticks and trackballs share a common control motion which is, in turn, different than the other sub-classes of two-dimensional devices.

The rows for *position* and *motion* sensing devices are subdivided in order to differentiate between transducers which sense potential *via* mechanical *vs* touch-sensitive means. Thus, we see that the light-pen and touch-screen are closely related, except that the light-pen employs a mechanical transducer. Similarly, we see that trackball and TASA touch-pad[2] provide comparable signals from comparable gestures (the 4" by 4" dimensions of the TASA device compare to a 3 1/2" diameter trackball).

The tableau is useful for many purposes by virtue of the structure which it imposes on the domain of input devices. First, it helps in finding appropriate equivalences. This is important in terms of dealing with some of the problems which arose in our discussion of device independence. For example, we saw a case where four tablets would not be suitable for replacing four joysticks. By using the tableau, we see that four trackballs will probably do.

The tableau makes it easy to relate different devices in terms of metaphor. For example, a tablet is to a mouse what a joystick is to a trackball. Furthermore, if the taxonomy defined by the tableau can suggest new transducers in a manner analogous to the periodic table of Mendeleev predicting new elements, then we can have more confidence in its underlying premises. We make this claim for the tableau and cite the "torque sensing" one-dimensional pressure-sensitive transducer as an example. To our knowledge, no such device exists commercially. Nevertheless it is a potentially useful device, an approximation of which has been demonstrated by Herot and Weinzaphel [15].

Finally, the tableau is useful in helping quantify the generality of various physical devices. In cases where the work station is limited to one or two input devices, then it is often in the user's interest to choose the least constraining devices. For this reason, many people claim that tablets are the preferred device since they can emulate many of the other transducers (as is demonstrated by Evans, Tanner and Wein, [9]). The tableau is useful in determining the degree of this generality by "filling in" the squares which can be adequately covered by the tablet.

Before leaving the topic of the tableau, it is worth commenting on why a primary criterion for grouping devices was whether they were sensitive to position, motion or pressure. The reason is that what is sensed has a *very* strong effect on the nature of the dialogues that the system can support with any degree of fluency. As an example, let us compare how the user interface of an instrumentation console can be affected by the choice of whether motion or position sensitive transducers are used. For such consoles, one design philosophy follows the traditional model that for every function there should be a device. One of the rationales behind this approach is to avoid the use of "modes" which result when a single device must serve for more than one function. Another philosophy takes the point of view that the number of devices required in a console need only be in the order of the control bandwidth of the human operator. Here, the rationale is that careful design can minimize the "mode" problem, and that the resulting simple consoles are more cost-effective and less prone to breakdown (since they have fewer devices).

One consequence of the second philosophy is that the same transducer must be made to control different functions, or parameters, at different times. This context switching introduces something known as the *nulling problem*. The point which we are going to make is that this problem can be completely avoided if the transducer in question is motion rather than position sensitive. Let us see why.

Imagine that you have a sliding potentiometer which controls parameter A. Both the potentiometer and the parameter are at their minimum values. You then raise A to its maximum value by pushing up the position of the potentiometer's handle. You now want to change the value of parameter B. Before you can do so using the same potentiometer, the handle of the potentiometer must be repositioned to a position corresponding to the current value of parameter B. The necessity of having to perform this normalizing function is the nulling problem.

Contrast the difficulty of performing the above interaction using a position-sensitive device with the ease of doing so using one which senses motion. If a thumb-wheel or a treadmill-like device was used, the moment that the transducer is connected to the parameter it can be used to "push" the value up or "pull" it down. Furthermore, the same transducer can be used to simultaneously change the value of a group of parameters, all of whose instantaneous values are different.

Horizontal vs Vertical Strata

The above example brings up one important point: the different levels of the taxonomies of Foley and Van Dam or of Moran are not orthogonal. By describing the user interface in terms of a horizontal structure, it is very easy to fall into the trap of believing that the effect of modifications at one level will be isolated. This is clearly not true as the above example demonstrated: the choice of transducer type had a strong effect on syntax.

The example is not isolated. In fact, just as strong an argument could be made for adopting a model based on a vertical structure as the horizontal ones which we have discussed. Models based on interaction techniques such as those described in Martin [17] and Foley, Wallace and Chan [11] are examples. With them, the primary gestalt is the transaction, or interaction. The user model is described in terms of the set and style of

[2] The TASA X-Y 360 is a 4" by 4" touch sensitive device which gives 60 units of delta modulation in 4 inches of travel. The device is available from TASA, 2346 Walsh Ave., Santa Clara CA. 95051.

the interactions which take place over time. Syntactic, lexical and pragmatic questions become sub-issues.

Neither the horizontal or vertical view is "correct." The point is that *both* must be kept in mind during the design process. A major challenge is to adapt our models so that this is done in a well structured way. That we still have problems in doing so can be seen in Moran's taxonomy. Much of the difficulty in understanding the model is due to problems in his approach in integrating vertically oriented concepts (the interaction level) into an otherwise horizontal structure.

In spite of such difficulties, both views must be considered. This is an important cautionary bell to ring given the current trend towards delegating personal responsibilities according to horizontal stratification. The design of a system's data-base, for example, has a very strong effect on the semantics of the interactions that can be supported. If the computing environment is selected by one person, the data-base managed by another, the semantics or functional capability by another, and the "user interface" by yet another, there is an inherent danger that the decisions of one will adversely affect another. This is not to say that such an organizational structure cannot work. It is just imperative that we be aware of the pitfalls so that they can be avoided. Decisions made at all levels affect one another and *all* decisions potentially have an effect on the user model.

Summary and Conclusions

Two taxonomies for describing the problem domain of the user interface were described. In the discussion it was pointed out that the outer levels of the strata, those concerning lexical, spatial, and physical issues were neglected. The notion of pragmatics was introduced in order to facilitate focusing attention on these issues. Several examples were then examined which illustrated why this was important. In so doing, it was seen that the power of various existing models could be extended if we had a better understanding of pragmatic issues. As a step towards such an understanding, a taxonomy of hand controlled continuous input devices was introduced. It was seen that this taxonomy made some contribution towards addressing problems which arose in the case studies. It was also seen, however, that issues at this outer level of devices had a potentially strong effect on the other levels of the system. Hence, the danger of over-concentration on horizontal stratification was pointed out.

The work reported has made some contribution towards an understanding of the effect of issues which we have called pragmatics. It is, however, a very small step. While there is a great deal of work still to be done right at the device level, perhaps the biggest challenge is to develop a better understanding of the interplay among the different levels in the strata of a system. When we have developed a methodology which allows us to determine the gesture that best suits the expression of a particular concept, then we will be able to build the user interfaces which today are only a dream.

Acknowledgements

The ideas presented in this paper have developed over a period of time and owe much to discussions with our students and colleagues. In particular, a great debt is owed to Ron Baecker who was responsible for helping formulate many of the ideas presented. In addition, we would like to acknowledge the contribution of Alain Fournier, Russel Kirsch, Eugene Fiume and Ralph Hill in the intellectual development of the paper, and the help of Monica Delange in the preparation of the manuscript. Finally, we gratefully acknowledge the financial support of the National Sciences and Engineering Research Council of Canada.

References

1. Baecker, R. Human-computer interactive systems: a state-of-the-art review. In P. Kolers, E. Wrolftad & H. Bouma, Eds., *Processing of Visible Language II*, New York: Plenum, (1980), 423–444.
2. Baecker, R. Towards an effective characterization of graphical interaction. In R. A. Guedj, P. Ten Hagen, F. Hopgood, H. Tucker & D. Duce, Eds., *Methodology of Interaction*, Amsterdam: North-Holland, (1980), 127–148.
3. Barnard, P., Hammond, N., Morton, J., and Long, J. Consistency and compatibility in human-computer dialogue. *International Journal of Man-Machine Studies 15*, (1981), 87–134.
4. Buxton, W. An informal study of selection-positioning tasks. *Proceedings of Graphics Interface '82*, Toronto, (1982), 323–328.
5. Card, S., English, W., and Burr, B. Evaluation of mouse, rate-controlled isometric joystick, step keys, and text keys for text selection on a CRT. *Ergonomics 8*, (1978), 601–613.
6. Card, S., Moran, T., and Newell, A. Computer text editing: an information-processing analysis of a routine cognitive skill. *Cognitive Psychology 12*, (1980),32–74.
7. Card, S., Moran, T., and Newell, A. The keystroke-level model for user performance time with interactive systems. *Communications of the ACM 23*, 7 (1980), 396–410.
8. Embley, D., Lan, M., Leinbaugh, D., and Nagy, G. A procedure for predicting program editor performance from the user's point of view. *International Journal of Man-Machine Studies 10*, (1978), 639–650.

9. Evans, K., Tanner, P., and Wein, M. Tablet-based valuators that provide one, two, or three degrees of freedom. *Computer Graphics 15*, 3 (1981), 91–97.

10. Fedorkow, G., Buxton, W., and Smith, K. C. A computer controlled sound distribution system for the performance of electroacoustic music. *Computer Music Journal 2*, 3 (1978), 33–42.

11. Foley, J., Wallace, V., and Chan, P. The human factors of interaction techniques. Technical Report GWU-IIST-81-03, Washington: The George Washington University, Institute for Information Science and Technology, (1981).

12. Foley, J. and Van Dam, A. *Fundamentals of Interactive Computer Graphics.* Reading, MA: Addison Wesley, (1982).

13. GSPC. Status Report of the Graphics Standards Planning Committee. *Computer Graphics 11*, (1977).

14. GSPC. Status Report of the Graphics Standards Committee. *Computer Graphics 13*, 3 (1979).

15. Herot, C. and Weinzaphel, G. One-point touch input of vector information for computer displays. *Computer Graphics 12*, 3 (1978), 210–216.

16. Ledgard, H., Whiteside, J., Singer, A., and Seymour, W. The natural language of interactive systems. *Communications of the ACM 23*, 10 (1980), 556–563.

17. Martin, J. *Design of Man-Computer Dialogues.* Engelwood Cliffs, NJ: Prentice-Hall, (1973).

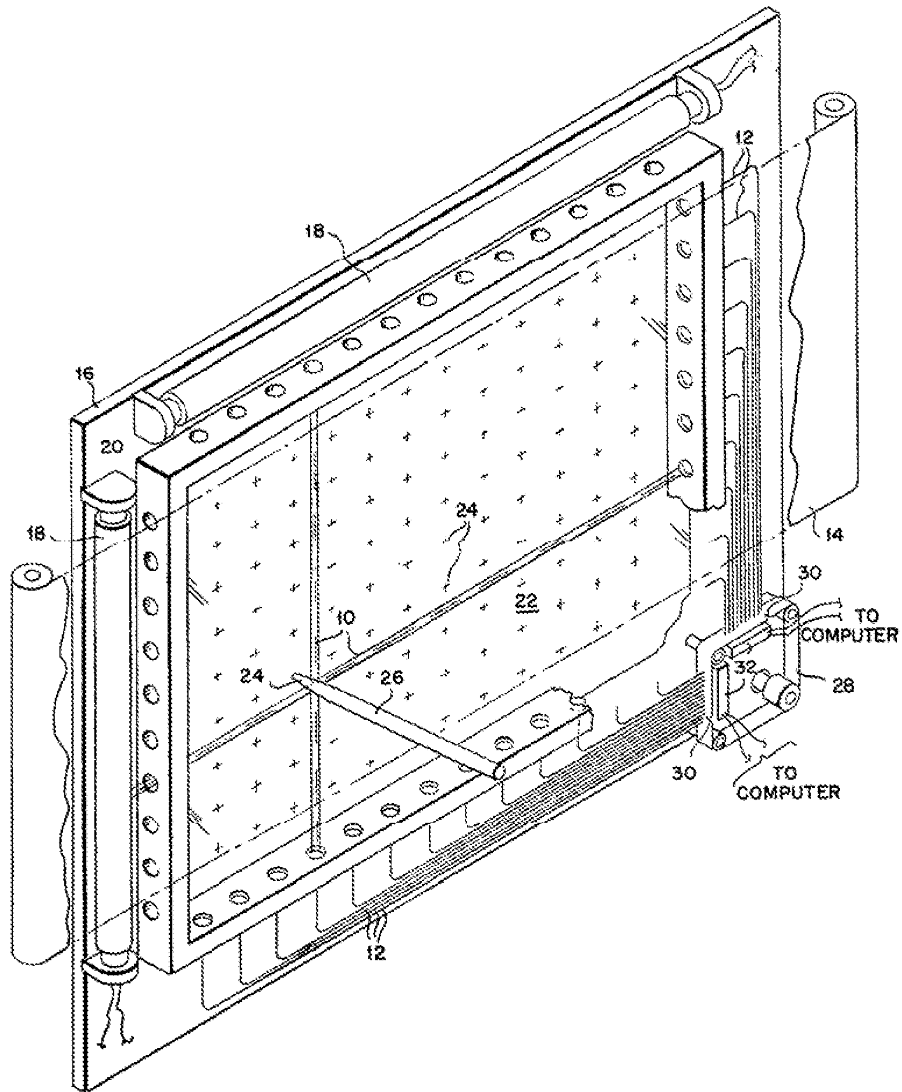
18. Moran, T. The command language grammar: a representation for the user interface of interactive computer systems. *International Journal of Man-Machine Studies 15*, (1981), 3–50.

19. Reisner, P. Use of psychological experimentation as an aid to development of a query language. *IEEE Transactions on Software Engineering 3*, 3 (1977), 218–229.

20. Reisner, P. Formal grammar and human factors design of an interactive graphics system. *IEEE Transactions on Software Engineering 7*, 2 (1981), 229–240.

Light Beam Matrix Input Terminal

This display and computer input device consists of a rectangular matrix of light beams 10 and associated photosensitive devices 12 overlaying document 14. Mount 16 contains a pair of light sources 18 at right angles to each other. Beams 10 are formed by holes in frame 20 and image on optical fibers 12 opposite sources 18. Thus, a light beam matrix is formed. The frame assembly is spaced slightly above document 14 by thin, clear screen 22 having response holes 24 at each intersection of beams 10. When probe 26 or the finger is placed in a hole of screen 22, intersecting beams are interrupted. Fibers 12 are merged to moving belt 28 having light detectors 32 at its underside. Fibers 12 are so arranged that slots 30 scan them serially. Document 14 can be one of a plurality on a roll.



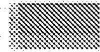
2

Multi-Touch Systems that I Have Known and Loved

Bill Buxton
Microsoft Research
Original: Jan. 12, 2007
Version: March 21st, 2011

Keywords / Search Terms

Multi-touch, multitouch, input, interaction, touch screen, touch tablet, multi-finger input, multi-hand input, bi-manual input, two-handed input, multi-person input, interactive surfaces, soft machine, hand gesture, gesture recognition .

This page is also available in Belorussian, thanks to the translation by Martha Ruszkowski.

Preamble

Since the announcements of the *iPhone* and Microsoft*'s Surface* (both in 2007), an especially large number of people have asked me about multi-touch. The reason is largely because they know that I have been involved in the topic for a number of years. The problem is, I can't take the time to give a detailed reply to each question. So I have done the next best thing (I hope). That is, start compiling my would-be answer in this document. The assumption is that ultimately it is less work to give one reasonable answer than many unsatisfactory ones.

Multi-touch technologies have a long history. To put it in perspective, my group at the University of Toronto was working on multi-touchin 1984 (Lee, Buxton & Smith, 1985), the same year that the first Macintosh computer was released, and we were not the first. Furthermore, during the development of the iPhone, Apple was very much aware of the history of multi-touch, dating at least back to 1982, and the use of the pinch gesture, dating back to 1983. This is clearly demonstrated by the bibliography of the PhD thesis of Wayne Westerman, co-founder of FingerWorks, a company that Apple acquired early in 2005, and now an Apple employee

> Westerman, Wayne (1999). *Hand Tracking,Finger Identification, and Chordic Manipulation on a Multi-Touch Surface.* U of Delaware PhD Dissertation: http://www.ee.udel.edu/~westerma/main.pdf

In making this statement about their awareness of past work, I am not criticizing Westerman, the iPhone, or Apple. It is simply good practice and good scholarship to know the literature and do one's homework when embarking on a new product. What I *am* pointing out, however, is that "new" technologies - like multi-touch - do not grow out of a vacuum. While marketing tends to like the "great invention" story, real innovation rarely works that way. In short, the evolution of multi-touch is a text-book example of what I call "the long-nose of innovation."

So, to shed some light on the back story of this particular technology, I offer this brief and incomplete summary of some of the landmark examples that I have been involved with, known about and/or encountered over the years. As I said, it is incomplete and a work in progress (so if you come back a second time, chances are there will be more and better information). I apologize to those that I have missed. I have erred on the side of timeliness *vs* thoroughness. Other work can be found in the references to the papers that I do include.

Note: for those note used to searching the HCI literature, the primary portal where you can search for and download the relevant literature, including a great deal relating to this topic (including the citations in the Westerman thesis), is the ACM Digital Library: http://portal.acm.org/dl.cfm. One other relevant source of interest, should you be interested in an example of the kind of work that has been done studying gestures in interaction, see the thesis by Hummels:

> http://id-dock.com/pages/overig/caro/publ_caro.htm

While not the only source on the topic by any means, it is a good example to help gauge what might be considered new or obvious.

Please do not be shy in terms of sending me photos, updates, etc. I will do my best to integrate them.

For more background on input, see also the incomplete draft manuscript for my book on input tools, theories and techniques:

> http://www.billbuxton.com/inputManuscript.html

For more background on input devices, including touch screens and tablets, see my directory at:

- http://www.billbuxton.com/InputSources.html

I hope this helps.

Some Dogma

There is a lot of confusion around touch technologies, and despite a 25 year history, very little information or experience with multi-touch interaction. I have three comments to set up what is to follow:

1. Remember that it took 30 years between when the mouse was invented by Engelbart and English in 1965 to when it became ubiquitous, on the release of Windows 95. Yes, it was released commercially on the Xerox Star and PERQ workstations in 1982, and I used my first one in 1972 at the National Research Council of Canada. But statistically, that doesn't matter. It took 30 years to hit the tipping point. So, by that measure, multi-touch technologies have 5 years to go before they fall behind.
2. Keep in mind one of my primary axioms: *Everything is best for something and worst for something else.* The trick is knowing what is what, for what, when, for whom, where, and most importantly, why. Those who try the replace the mouse play a fool's game. The mouse is great for many things. Just not everything. The challenge with new input is to find devices that work together, simultaneously with the mouse (such as in the other hand), or things that are strong where the mouse is weak, thereby complimenting it.
3. To significantly improve a product by a given amount, it probably takes about two more orders of magnitude of cost, time and effort to improve the display as to get the same amount of improvement on input. Why? Because we are ocular centric, and displays are therefore much more mature. Input is still primitive, and wide open for improvement. So it is a good thing that you are looking at this stuff. What took you so long?

Some Framing

I don't have time to write a treatise, tutorial or history. What I can do is warn you about a few traps that seem to cloud a lot of thinking and discussion around this stuff. The approach that I will take is to draw some distinctions that I see as meaningful and relevant. These are largely in the form of contrasts:

- **Touch-tablets *vs* Touch screens:** In some ways these are two extremes of a continuum. If, for example, you have paper graphics on your tablet, is that a display (albeit more-or-less static) or not? What if the "display" on the touch tablet is a tactile display rather than visual? There are similarities, but there are real differences between touch-sensitive display surfaces, *vs* touch pads or tablets. It is a difference of *directness.* If you touch exactly where the thing you are interacting with is, let's call it a touch screen or touch display. If your hand is touching a surface that is not overlaid on the screen, let's call it a touch tablet or touch pad.

- **Discrete *vs* Continuous:** The nature of interaction with multi-touch input is highly dependent on the nature of discrete *vs* continuous actions supported. Many conventional touch-screen interfaces are based discrete items such as pushing so-called "light buttons", for example. An example of a multi-touch interface using such discrete actions would be using a soft graphical QWERTY keyboard, where one finger holds the shift key and another pushes the key for the upper-case character that one wants to enter. An example of two fingers doing a coordinated continuous action would be where they are stretching the diagonally opposed corners of a rectangle, for example. Between the two is a continuous/discrete situation, such as where one emulates a mouse, for example, using one finger for indicating continuous position, and other fingers, when in contact, indicate mouse button pushes, for example.

- **Degrees of Freedom:** The richness of interaction is highly related to the richness/numbers of degrees of freedom (DOF), and in particular, continuous degrees of freedom, supported by the technology. The conventional GUI is largely based on moving around a single 2D cursor, using a mouse, for example. This

results in 2DOF. If I am sensing the location of two fingers, I have 4DOF, and so on. When used appropriately, these technologies offer the potential to begin to capture the type of richness of input that we encounter in the everyday world, and do so in a manner that exploits the everyday skills that we have acquired living in it. This point is tightly related to the previous one.

- **Size matters:** Size largely determines what muscle groups are used, how many fingers/hands can be active on the surface, and what types of gestures are suited for the device.

- **Orientation Matters - Horizontal *vs* Vertical:** Large touch surfaces have traditionally had problems because they could only sense one point of contact. So, if you rest your hand on the surface, as well as the finger that you want to point with, you confuse the poor thing. This tends not to occur with vertically mounted surfaces. Hence large electronic whiteboards frequently use single touch sensing technologies without a problem.

- **There is more to touch-sensing than contact and position:** Historically, most touch sensitive devices only report that the surface has been touched, and where. This is true for both single and multi touch devices. However, there are other aspects of touch that have been exploited in some systems, and have the potential to enrich the user experience:

  1. **Degree of touch / pressure sensitivity:** A touch surfaces that that can independently and continuously sense the degree of contact for each toouch point has a far higher potential for rich interaction. Note that I use "degree of contact" rather than pressure since frequently/usually, what passes for pressure is actually a side effect – as you push harder, your finger tip spreads wider over the point of contact, and what is actually sensed is amount/area of contact, not pressure, *per se*. Either is richer than just binary touch/no touch, but there are even subtle differences in the affordances of pressure *vs* degree.

  2. **Angle of approach:** A few systems have demonstrated the ability to sense the angle that the finger relative to the screen surface. See, for example, McAvinney's *Sensor Frame*, below. In effect, this lgives the finger the capability to function more-or-less as a virtual joystick at the point of contact, for example. It also lets the finger specify a vector that can be projected into the virtual 3D space behind the screen from the point of contact - something that could be relevant in games or 3D applications.

  3. **Force vectors:** Unlike a mouse, once in contact with the screen, the user can exploit the friction between the finger and the screen in order to apply various force vectors. For example, without moving the finger, one can apply a force along any vector parallel to the screen surface, including a rotational one. These techniques were described as early as 1978, as shown below, by Herot, C. & Weinzapfel, G. (1978). Manipulating Simulated Objects with Real-World Gestures Using a Force and Position Sensitive Screen, *Computer Graphics*, 18(3), 195-203.].

  Such historical examples are important reminders that it is human capability, not technology, that should be front and centre in our considerations. While making such capabilities accessible at reasonable costs may be a challenge, it is worth remembering further that the same thing was also said about multi-touch. Furthermore, note that multi-touch dates from about the same time as these other touch innovations.

- **Size matters II:** The ability of to sense the size of the area being touched can be as important as the size of the touch surface. See the Synaptics example, below, where the device can sense the difference between the touch of a finger (small) *vs* that of the cheek (large area), so that, for example, you can answer the phone by

holding it to the cheek.

- **Single-finger *vs* multi-finger:** Although multi-touch has been known since at least 1982, the vast majority of touch surfaces deployed are single touch. If you can only manipulate one point, regardless of with a mouse, touch screen, joystick, trackball, etc., you are restricted to the gestural vocabulary of a fruit fly. We were given multiple limbs for a reason. It is nice to be able to take advantage of them.

- **Multi-point *vs* multi-touch:** It is really important in thinking about the kinds of gestures and interactive techniques used if it is peculiar to the technology or not. Many, if not most, of the so-called "multi-touch" techniques that I have seen, are actually "multi-point". Think of it this way: you don't think of yourself of using a different technique in operating your laptop just because you are using the track pad on your laptop (a single-touch device) instead of your mouse. Double clicking, dragging, or working pull-down menus, for example, are the same interaction technique, independent of whether a touch pad, trackball, mouse, joystick or touch screen are used.

- **Multi-hand *vs* multi-finger:** For much of this space, the control can not only come from different fingers or different devices, but different hands working on the same or different devices. A lot of this depends on the scale of the input device. Here is my analogy to explain this, again referring back to the traditional GUI. I can point at an icon with my mouse, click down, drag it, then release the button to drop it. Or, I can point with my mouse, and use a foot pedal to do the clicking. It is the same dragging technique, even though it is split over two limbs and two devices. So a lot of the history here comes from a tradition that goes far beyond just multi-touch.

- **Multi-person *vs* multi-touch:** If two points are being sensed, for example, it makes a huge difference if they are two fingers of the same hand from one user *vs* one finger from the right hand of each of two different users. With most multi-touch techniques, you do *not* want two cursors, for example (despite that being one of the first thing people seem to do). But with two people working on the same surface, this may be exactly what you *do* want. And, insofar as multi-touch technologies are concerned, it may be valuable to be able to sense which person that touch comes from, such as can be done by the *Diamond Touch* system from MERL (see below).

- **Points *vs* Gesture:** Much of the early relevant work, such as Krueger (see below) has to do with sensing the pose (and its dynamics) of the hand, for example, as well as position. That means it goes way beyond the task of sensing multiple points.

- **Stylus and/or finger:** Some people speak as if one must make a choice between stylus *vs* finger. It certainly is the case that many stylus systems will not work with a finger, but many touch sensors work with a stylus or finger. It need not be an either or question (although that might be the correct decision – it depends on the context and design). But any user of the Palm Pilot knows that there is the potential to use either. Each has its own strengths and weaknesses. Just keep this in mind: if the finger was the ultimate device, why didn't Picasso and Rembrandt restrict themselves to finger painting? On the other hand, if you want to sense the temperature of water, your finger is a better tool than your pencil.

- **Hands and fingers *vs* Objects:** The stylus is just one object that might be used in multi-point interaction. Some multi-point / multi-touch systems can not only sense various different objects on them, but what object it is, where it is, and what its orientation is. See Andy Wilson's work, below, for example. And, the objects,

stylus or otherwise, may or may not be used in conjunction and simultaneously with fingers.

- **Different *vs* The Same:** When is something the same, different or obvious? In one way, the answer depends on if you are a user, programmer, scientist or lawyer. From the perspective of the user interface literature, I can make three points that would be known and assumed by anyone skilled in the art:

    1. *Device-Independent Graphics:* This states that the same technique implemented with an alternative input device is still the same technique. For example, you can work your GUI with a stylus, touch screen, mouse, joystick, touchpad, or trackball, and one would still consider techniques such as double-clicking, dragging, dialogue boxes as being "the same" technique;

    2. *The Interchange of devices is not neutral from the perspective of the user:* While the skill of using a GUI with a mouse transfers to using a touchpad, and the user will consider the interface as using the same techniques, nevertheless, the various devices have their own idiomatic strengths and weaknesses. So, while the user will consider the techniques the "same", their performance (speed, accuracy, comfort, preference, etc.) will be different from device to device. Hence, the interactive experience is not the same from device to device, despite using the same techniques. Consequently, it is the norm for users and researchers alike to swap one device for another to control a particular technique.

Some Attributes

As I stated above, my general rule is that everything is best for something and worst for something else. The more diverse the population is, the places and contexts where they interact, and the nature of the information that they are passing back in forth in those interactions, the more there is room for technologies tailored to the idiosyncrasies of those tasks.

The potential problem with this, is that it can lead to us having to carry around a collection of devices, each with a distinct purpose, and consequently, a distinct style of interaction. This has the potential of getting out of hand and our becoming overwhelmed by a proliferation of gadgets – gadgets that are on their own are simple and effective, but collectively do little to reduce the complexity of functioning in the world. Yet, traditionally our better tools have followed this approach. Just think of the different knives in your kitchen, or screwdrivers in your workshop. Yes there are a great number of them, but they are the "right ones", leading to an interesting variation on an old theme, namely, "more is less", i.e., more (of the right) technology results is less (not more) complexity. But there are no guarantees here.

What touch screen based "soft machines" offer is the opposite alternative, "less is more". Less, but more generally applicable technology results in less overall complexity. Hence, there is the prospect of the multi-touch soft machine becoming a kind of chameleon that provides a single device that can transform itself into whatever interface that is appropriate for the specific task at hand. The risk here is a kind of "jack of all trades, master of nothing" compromise.

One path offered by touch-screen driven appliances is this: instead of making a device with different buttons and dials mounted on it, soft machines just draw a picture of the devices, and let you interact with them. So, ideally, you get far more flexibility out of a single device. Sometimes, this can be really good. It can be especially good if, like physical devices, you can touch or operate more than one button, or virtual device at a time. For an example of where using more than one button or device at a time is important in the physical world, just think of having to type

without being able to push the SHIFT key at the same time as the character that you want to appear in upper case. There are a number of cases where this can be of use in touch interfaces.

Likewise, multi-touch greatly expands the types of gestures that we can use in interaction. We can go beyond simple pointing, button pushing and dragging that has dominated our interaction with computers in the past. The best way that I can relate this to the everyday world is to have you imagine eating Chinese food with only one chopstick, trying to pinch someone with only one fingertip, or giving someone a hug with – again – the tip of one finger or a mouse. In terms of pointing devices like mice and joysticks are concerned, we do everything by manipulating just one point around the screen – something that gives us the gestural vocabulary of a fruit fly. One suspects that we can not only do better, but as users, deserve better. Multi-touch is one approach to accomplishing this – but by no means the only one, or even the best. (How can it be, when I keep saying, everything is best for something, but worst for something else).

There is no Free Lunch.

- **Feelings**: The adaptability of touch screens in general, and multi-touch screens especially comes at a price. Besides the potential accumulation of complexity in a single device, the main source of the downside stems from the fact that you are interacting with a picture of the ideal device, rather than the ideal device itself. While this may still enable certain skills from the specialized physical device transfer to operating the virtual one, it is simply not the same. Anyone who has typed on a graphical QWERTY keyboard knows this.

  User interfaces are about look *and* feel. The following is a graphic illustration of how this generally should be written when discussing most touch-screen based systems:

  # Look and Feel

  Kind of ironic, given that they are "touch" screens. So let's look at some of the consequences in our next points.

- **If you are blind you are simply out of luck. p.s., we are all blind at times** - such as when lights are out, or our eyes are occupied elsewhere – such as on the road). On their own, soft touch screen interfaces are nearly all "eyes on". You cannot "touch type", so to speak, while your eyes are occupied elsewhere (one exception is so-called "heads-up" touch entry using single stroke gestures such as Graffiti that are location independent). With an all touch-screen interface you generally cannot start, stop, or pause your MP3 player, for example, by reaching into your pocket/purse/briefcase. Likewise, unless you augment the touch screen with speech recognition for all functions, you risk a serious accident trying to operate it while driving. On the other hand, MP3 players and mobile phones mechanical keys can to a certain degree be operated eyes free – the extreme case being some 12-17 year old kids who can text without looking!
- **Handhelds that rely on touch screens for input virtually all require two hands to operate:** one to hold the device and the other to operate it. Thus, operating them generally requires both eyes *and* both hands.
- **Your finger is not transparent:** The smaller the touch screen the more the finger(s) obscure what is being pointed at. Fingers do not shrink in the same way that chips and displays do. That is one reason a stylus is sometimes of value: it is a proxy for the finger that is very skinny, and therefore does not obscure the screen.

- **There is a reason we don't rely on finger painting:** Even on large surfaces, writing or drawing with the finger is generally not as effective as it is with a brush or stylus. On small format devices it is virtually useless to try and take notes or make drawings using a finger rather than a stylus. If one supports good digital ink and an appropriate stylus and design, one can take notes about as fluently as one can with paper. Note taking/scribble functions are notably absent from virtually all finger-only touch devices.
- **Sunshine:** We have all suffered trying to read the colour LCD display on our MP3 player, mobile phone and digital camera when we are outside in the sun. At least with these devices, there are mechanical controls for some functions. For example, even if you can't see what is on the screen, you can still point the camera in the appropriate direction and push the shutter button. With interfaces that rely exclusively on touch screens, this is not the case. Unless the device has an outstanding reflective display, the device risks being unusable in bright sunlight.

Does this property make touch-devices a bad thing? No, not at all. It just means that they are distinct devices with their own set of strengths and weaknesses. The ability to completely reconfigure the interface on the fly (so-called "soft interfaces") has been long known, respected and exploited. But there is no free lunch and no general panacea. As I have said, everything is best for something and worst for something else. Understanding and weighing the relative implications on use of such properties is necessary in order to make an informed decision. The problem is that most people, especially consumers (but including too many designers) do not have enough experience to understand many of these issues. This is an area where we could all use some additional work. Hopefully some of what I have written here will help.

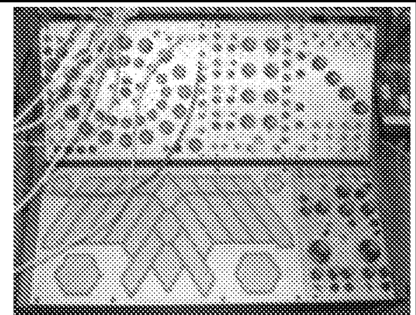
An Incomplete Roughly Annotated Chronology of Multi-Touch and Related Work

In the beginning .... Typing & N-Key Rollover (IBM and others).

Photo Credit

- While it may seem a long way from multi-touch screens, the story of multi-touch starts with keyboards.
- Yes they are mechanical devices, "hard" rather than "soft" machines. But they do involve multi-touch of a sort.
- First, most obviously, we see sequences, such as the SHIFT, Control, Fn or ALT keys in combination with others. These are cases where we *want* multi-touch.
- Second, there are the cases of unintentional, but inevitable, multiple simultaneous key presses which we want to make proper sense of, the so-called question of n-key rollover (where you push the next key before releasing the previous one).

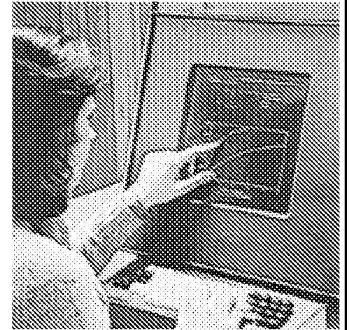
Electroacoustic Music: The Early Days of Electronic Touch Sensors (Hugh LeCaine , Don Buchla & Bob Moog).
http://www.hughlecaine.com/en/instruments.html.



- The history of touch-sensitive control devices pre-dates the age of the PC
- A number of early synthesizer and electronic music instrument makers used touch-sensitive capacitance-sensors to control the sound and music being made.
- These were touch pads, rather than touch screens
- The tradition of innovating on touch controls for musical purposes continued/continues, and was the original basis for the University of Toronto multitouch surface, as well as the CMU Sensor Frame.

1972: PLATO IV Touch Screen Terminal (Computer-based Education Research Laboratory, University of Illinois, Urbana-Champain)
http://en.wikipedia.org/wiki/Plato_computer

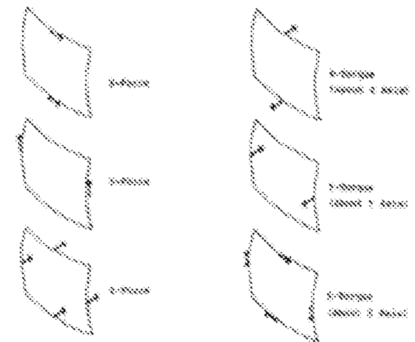
- Touch screens started to be developed in the second half of the 1960s.
- Early work was done at the IBM, the University of Illinois, and Ottawa Canada.



By 1971 a number of different techniques had been disclosed

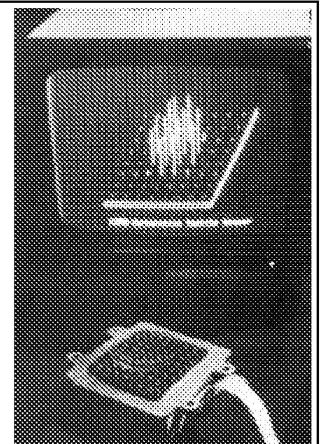
- All were single-touch and none were pressure-sensitive
- One of the first to be generally known was the terminal for the PLATO IV computer assisted education system, deployed in 1972.
- As well as its use of touch, it was remarkable for its use of real-time random-access audio playback, and the invention of the flat panel plasma display.
- the touch technology used was a precursor to the infrared technology still available today from CarrollTouch.
- The initial implementation had a 16 x 16 array of touch-sensitive locations

1978: One-Point Touch Input of Vector Information (Chris Herot & Guy Weinzapfel, Architecture Machine Group, MIT).



- The screen demonstrated by Herot & Weinzapfel could sense 8 different signals from a single touch point: position in X & Y, force in X, Y, & Z (i.e., sheer in X & Y & Pressure in Z), and torque in X, Y & Z.
- While we celebrate how clever we are to have multi-touch sensors, it is nice to have this reminder that there are many other dimensions of touch screens that can be exploited in order to provide rich interaction
- See: Herot, C. & Weinzapfel, G. (1978). One-Point Touch Input of Vector Information from Computer Displays, *Computer Graphics*, 12(3), 210-216.
- For a video demo, see: http://www.youtube.com/watch?v=vMkYfd0sOLM
- For similar work, see also: Minsky, M. (1984). Manipulating Simulated Objects with Real-World Gestures Using a Force and Position Sensitive Screen, *Computer Graphics*, 18(3), 195-203.

1981: Tactile Array Sensor for Robotics (Jack Rebman, Lord Corporation).

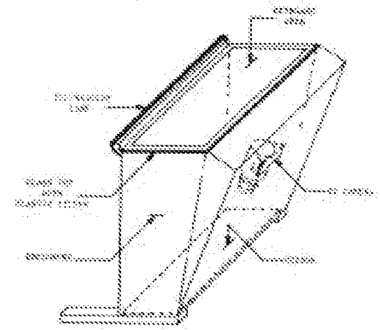


- A multi-touch sensor designed for robotics to enable sensing of shape, orientation, etc.
- Consisted of an 8 x 8 array of sensors in a 4" x 4" square pad
- Usage described in: Wolfeld, Jeffrey A. (1981). *Real Time Control of a Robot Tactile Sensor*. MSc Thesis. Philadelphia: Moore School of Electrical Engineering.
- The figure to the right shows a computer display of the tactile impression of placing a round object on the tactile sensor, shown in the foreground. Groover, M.P., Weiss, M., Nagel, R.N. & Odrey, N. (1986). *Industrial Robots.* New York: McGraw-Hill, p.152.)
- A US patent (4,521,685) was issued for this work to Rebman in 1985.

1982: Flexible Machine Interface (Nimish Mehta , University of Toronto).

- The first multi-touch system that I am aware of designed for human input to a computer system.
- Consisted of a frosted-glass panel whose local optical properties were such that

when viewed behind with a camera a black spot whose size depended on finger pressure appeared on an otherwise white background. This with simple image processing allowed multi touch input picture drawing, etc. At the time we discussed the notion of a projector for defining the context both for the camera and the human viewer.



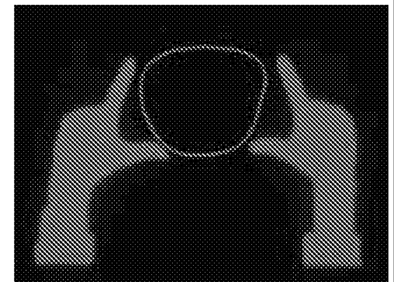
- Mehta, Nimish (1982), *A Flexible Machine Interface,* M.A.Sc. Thesis, Department of Electrical Engineering, University of Toronto supervised by Professor K.C. Smith.

1983: Soft Machines (Bell Labs, Murray Hill)

- This is the first paper that I am aware of in the user interface literature that attempts to provide a comprehensive discussion the properties of touch-screen based user interfaces, what they call "soft machines".
- While not about multi-touch specifically, this paper outlined many of the attributes that make this class of system attractive for certain contexts and applications.
- Nakatani, L. H. & Rohrlich, John A. (1983). Soft Machines: A Philosophy of User-Computer Interface Design. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'83)*, 12-15.

1983: Video Place / Video Desk (Myron Krueger)

- A vision based system that tracked the hands and enabled multiple fingers, hands, and people to interact using a rich set of gestures.
- Implemented in a number of configurations, including table and wall.
- Didn't sense touch, per se, so largely relied on dwell time to trigger events intended by the pose.
- On the other hand, in the horizontal desktop configuration, it inherently *was* touch based, from the user's perspective.
- Essentially "wrote the book" in terms of unencumbered (i.e., no gloves, mice, styli, etc.) rich gestural interaction.
- Work that was more than a decade ahead of its time and hugely influential, yet not as acknowledged as it should be.
- His use of many of the hand gestures that are now starting to emerge can be clearly seen in the following 1988 video, including using the pinch gesture to scale and translate objects: http://youtube.com/watch?v=dmmxVA5xhuo
- There are many other videos that demonstrate this system. Anyone in the field should view them, as well as read his books:
- Krueger, Myron, W. (1983). *Artificial Reality*. Reading, MA:Addison-Wesley.
- Krueger, Myron, W. (1991). *Artificial Reality II*. Reading, MA: Addison-Wesley.
- Krueger, Myron, W., Gionfriddo, Thomas., &Hinrichsen, Katrin (1985). VIDEOPLACE - An Artificial Reality, *Proceedings of the ACM Conference on Human Factors in Computing Systems* (CHI'85), 35 - 40.



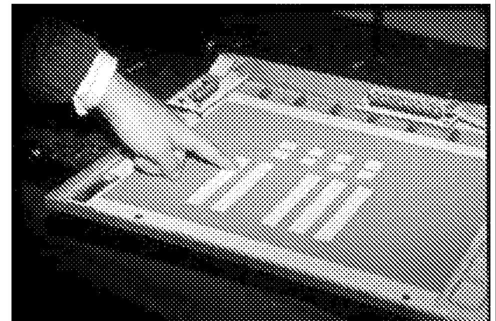
Myron's work had a staggeringly rich repertoire of gestures, muti-finger, multi-hand and multi-person interaction.

1984: Multi-Touch Screen (Bob Boie, Bell Labs, Murray Hill NJ)

- A multi-touch touch *screen*, not tablet.
- The first muti-touch screen that I am aware of.
- Used a transparent capacitive array of touch sensors overlaid on a CRT. Could manipulate graphical objects with fingers with excellent response time

- Developed by Bob Boie, but was shown to me by Lloyd Nakatani (see above), who invited me to visit Bell Labs to see it after he saw the presentation of our work at SIGCHI in 1985
- Since Boie's technology was transparent and faster than ours, when I saw it, my view was that they were ahead of us, so we stopped working on hardware (expecting that we would get access to theirs), and focus on the software and the interaction side, which was our strength. Our assumption (false, as it turned out) was that the Boie technology would become available to us in the near future.
- Around 1990 I took a group from Xerox to see this technology it since I felt that it would be appropriate for the user interface of our large document processors. This did not work out.
- There was other multi-touch work at Bell Labs around the time of Boie's. See the 1984 work by Leonard Kasday, (US Patent 4484179), which used optical techniques

1985: Multi-Touch Tablet (Input Research Group, University of Toronto): http://www.billbuxton.com/papers.html#anchor1439918



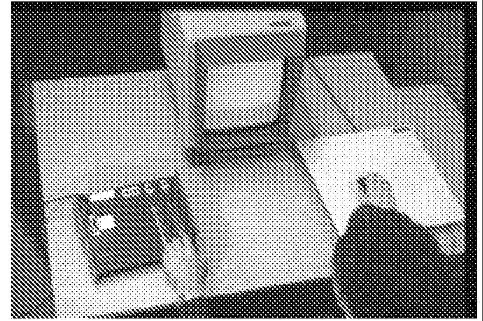
- Developed a touch tablet capable of sensing an arbitrary number of simultaneous touch inputs, reporting both location and degree of touch for each.
- To put things in historical perspective, this work was done in 1984, the same year the first Macintosh computer was introduced.
- Used capacitance, rather than optical sensing so was thinner and much simpler than camera-based systems.
- A Multi-Touch Three Dimensional Touch-Sensitive Tablet (1985). Video at: http://www.billbuxton.com/buxtonIRGVideos.html
- Issues and techniques in touch-sensitive tablet input.(1985). Video at: http://www.billbuxton.com/buxtonIRGVideos.html

1985: Sensor Frame (Carnegie Mellon University)

- This is work done by Paul McAvinney at Carengie-Mellon University
- The device used optical sensors in the corners of the frame to detect fingers.
- At the time that this was done, miniature cameras were essentially unavailable. Hence, the device used DRAM IC's with glass (as opposed to opaque) covers for imaging.
- It could sense up to three fingers at a time fairly reliably (but due to optical technique used, there was potential for misreadings due to shadows.
- In a later prototype variation built with NASA funding, the Sensor Cube, the device could also could detect the angle that the finger came in to the screen.
  - McAvinney, P. (1986). *The Sensor Frame - A Gesture-Based Device for the Manipulation of Graphic Objects*. Carnegie-Mellon University.
  - McAvinney, P. (1990). Telltale Gstures: 3D applications need 3D input. Byte Magazine, 15(7), 237-240.
  - http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19940003261_1994003261.pdf

1986: Bi-Manual Input (University of Toronto)

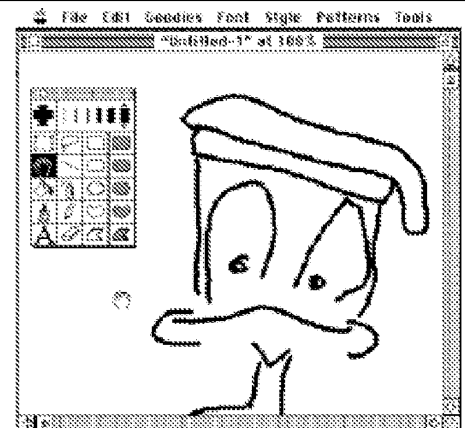
- In 1985 we did a study, published the following year, which examined the benefits of two different compound bi-manual tasks that involved continuous control with each hand
- The first was a positioning/scaling task. That is, one had to move a shape to a particular location on the screen with one hand, while adjusting its size to match a particular target with the other.
-



The second was a selection/navigation task. That is, one had to navigate to a particular location in a document that was currently off-screen, with one hand, then select it with the other.

- Since bi-manual continuous control was still not easy to do (the ADB had not yet been released - see below), we emulated the Macintosh with another computer, a PERQ.
- The results demonstrated that such continuous bi-manual control was both easy for users, and resulted in significant improvements in performance and learning.
- See Buxton, W. & Myers, B. (1986). A study in two-handed input. *Proceedings of CHI '86,* 321-326.[video]
- Despite this capability being technologically and economically viable since 1986 (with the advent of the ADB - see below - and later USB), there are still no mainstream systems that take advantage of this basic capability. Too bad.
- This is an example of techniques developed for multi-device and multi-hand that can easily transfer to multi-touch devices.

1986: Apple Desktop Bus (ADB) and the Trackball Scroller Init (Apple Computer / University of Toronto)



- The Macintosh II and Macintosh SE were released with the Apple Desktop Bus. This can be thought of as an early version of the USB.
- It supported plug-and-play, and also enabled multiple input devices (keyboards, trackballs, joysticks, mice, etc.) to be plugged into the same computer simultaneously.
- The only downside was that if you plugged in two pointing devices, by default, the software did not distinguish them. They both did the same thing, and if a mouse and a trackball were operate at the same time (which they could be) a kind of tug-of-war resulted for the tracking symbol on the screen.
- My group at the University of Toronto wanted to take advantage of this multi-device capability and contacted friends at Apple's Advanced Technology Group for help.
- Due to the efforts of Gina Venolia and Michael Chen , they produced a simple "init" that could be dropped into the systems folder called the *trackballscroller init.*
- It enabled the mouse, for example, to be designated the pointing device, and a trackball, for example, to control scrolling independently in X and Y. See, for example, Buxton, W. (1990). The Natural Language of Interaction: A Perspective on Non-Verbal Dialogues. In Laurel, B. (Ed.). *The Art of Human-Computer Interface Design,* Reading, MA: Addison-Wesley. 405-416.
- They also provided another init that enabled us to grab the signals from the second device and use it to control a range of other functions. See fr example, Kabbash, P., Buxton, W.& Sellen, A. (1994). Two-Handed Input in a

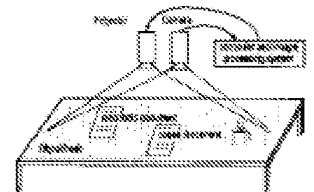
Compound Task. *Proceedings of CHI '94*, 417-423.

- In short, with this technology, we were able to deliver the benefits demonstrated by Buxton & Myers (see above) on standard hardware, without changes to the operating system, and largely, with out changes even to the applications.
- This is the closest that we came, without actually getting there, of supporting multi-point input - such as all of the two-point stretching, etc. that is getting so much attention now, 20 years later. It was technologically and economically viable then.
- To our disappointment, Apple never took advantage of this - one of their most interesting - innovations.

1991: Bidirectional Displays (Bill Buxton & Colleagues , Xerox PARC)

- First discussions about the feasibility of making an LCD display that was also an input device, i.e., where pixels were input as well as output devices. Led to two initiatives. (Think of the paper-cup and string "walkie-talkies" that we all made as kids: the cups were bidirectional and functioned simultaneously as both a speaker and a microphone.)
- Took the high res 2D a-Si scanner technology used in our scanners and adding layers to make them displays. The bi-directional motivation got lost in the process, but the result was the dpix display (http://www.dpix.com/about.html);
- The Liveboard project. The rear projection Liveboard was initially conceived as a quick prototype of a large flat panel version that used a tiled array of bi-directional dpix displays.

1991: Digital Desk(Pierre Wellner, Rank Xerox EuroPARC, Cambridge)



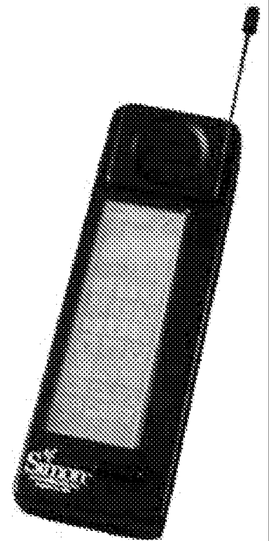
- A classic paper in the literature on augmented reality.
- Wellner, P. (1991). The Digital Desk Calculator: Tactile manipulation on a desktop display. *Proceedings of the Fourth Annual Symposium on User Interface Software and Technology (UIST '91)*, 27-33.
- An early front projection tablet top system that used optical and acoustic techniques to sense both hands/fingers as well as certain objects, in particular, paper-based controls and data.
- Clearly demonstrated multi-touch concepts such as two finger scaling and translation of graphical objects, using either a pinching gesture or a finger from each hand, among other things.
- For example, see segment starting at 6:30 in the following 1991 video demo: http://video.google.com/videoplay?docid=5772530828816089246

1992: Flip Keyboard(Bill Buxton, Xerox PARC): www.billbuxton.com

- A multi-touch pad integrated into the bottom of a keyboard. You flip the keyboard to gain access to the multi-touch pad for rich gestural control of applications.
- Combined keyboard / touch tablet input device (1994). Click here for video (from 2002 in conjunction with Tactex Controls).

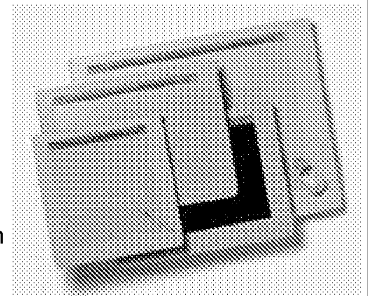
Sound Synthesizer Audio Mixer
Graphics on multi-touch surface defining controls for various virtual devices.

1992: Simon (IBM & Bell South)

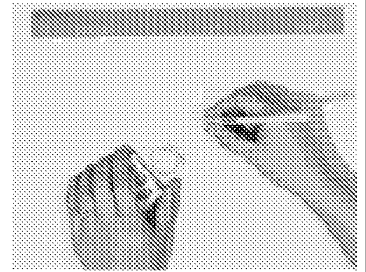


- IBM and Bell South release what was arguably the world's first smart phone, the *Simon.*
- What is of historical interest is that the Simon, like the iPhone, relied on a touch-screen driven "soft machine" user interface.
- While only a single-touch device, the Simon foreshadows a number of aspects of what we are seeing in some of the touch-driven mobile devices that we see today.
- Sidebar: my two working Simons are among the most prized pieces in my collection of input devices.

1992: Wacom (Japan)



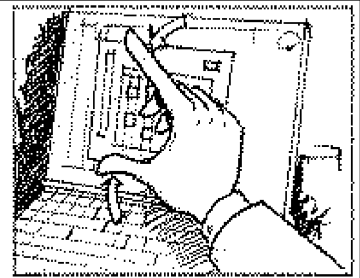
- In 1992 Wacom introduced their UD series of digitizing tablets. These were special in that they had mutli-device / multi-point sensing capability. They could sense the position of the stylus and tip pressure, as well as simultaneously sense the position of a mouse-like puck. This enabled bimanual input.
- Working with Wacom, my lab at the University of Toronto developed a number of ways to exploit this technology to far beyond just the stylus and puck. See the work on Graspable/Tangible interfaces, below.
- Their next two generations of tablets, the Intuos 1 (1998) and Intuos 2 (2001) series extended the multi-point capability. It enabled the sensing of the location of the stylus in x and y, plus tilt in x and tilt in y (making the stylus a location-sensitive joystick, in effect), tip pressure, and value from a side-mounted dial on their airbrush



stylus. As well, one could simultaneously sense the position and rotation of the puck, as well as the rotation of a wheel on its side. In total, one was able to have control of 10 degrees of freedom using two hands.

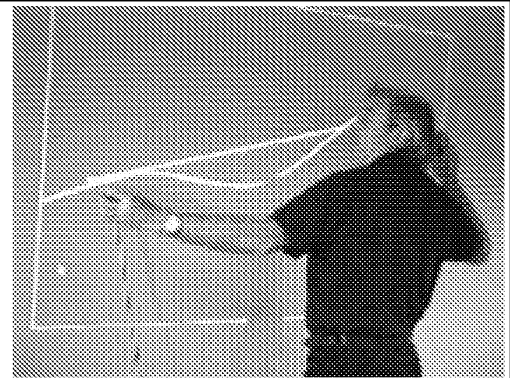
- While this may seem extravagant and hard to control, that all depended on how it was used. For example, all of these signals, coupled with bimanual input, are needed to implement any digital airbrush worthy of the name. With these technologies we were able to do just that with my group at Alias|Wavefront, again, with the cooperation of Wacom.
- See also: Leganchuk, A., Zhai, S.& Buxton, W. (1998).Manual and Cognitive Benefits of Two-Handed Input: An Experimental Study.*Transactions on Human-Computer Interaction,* 5(4), 326-359.

1992: Starfire (Bruce Tognazinni, SUN Microsystems)



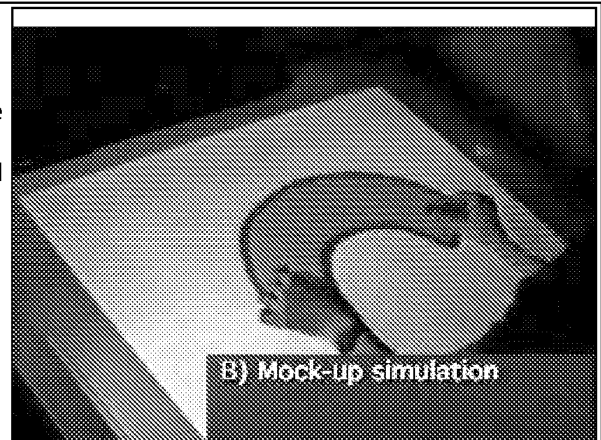
- Bruce Tognazinni produced an future envisionment film, Starfire, that included a number of multi-hand, multi-finger interactions, including pinching, etc.

1994-2002: Bimanual Research (Alias|Wavefront, Toronto)



- Developed a number of innovative techniques for multi-point / multi-handed input for rich manipulation of graphics and other visually represented objects.
- Only some are mentioned specifically on this page.
- There are a number of videos can be seen which illustrate these techniques, along with others: http://www.billbuxton.com/buxtonAliasVideos.html
- Also see papers on two-handed input to see examples of multi-point manipulation of objects at: http://www.billbuxton.com/papers.html#anchor1442822

1995: Graspable/Tangible Interfaces (Input Research Group, University of Toronto)



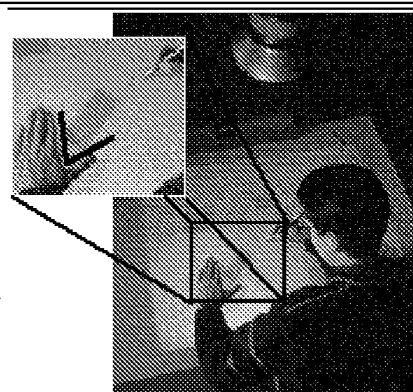
- Demonstrated concept and later implementation of sensing the identity, location and even rotation of multiple physical devices on a digital desk-top display and using them to control graphical objects.
- By means of the resulting article and associated thesis introduced the notion of what has come to be known as “graspable” or “tangible” computing.
- Fitzmaurice, G.W., Ishii, H. & Buxton, W. (1995). Bricks: Laying the foundations for graspable user interfaces. *Proceedings of the ACMSIGCHI Conference on Human Factors in Computing Systems (CHI'95),* 442–449.

1995: DSI Datotech (Vancouver BC)

- In 1995 this company made a touch tablet, the *HandGear*, capable of multipoint sensing. They also developed a software package, *Gesture Recognition Technology (GRT)*, for recognizing hand gestures captured with the tablet.
- The company went out of business around 2002

1995/97: Active Desk (Input Research Group / Ontario Telepresence Project,University of Toronto)

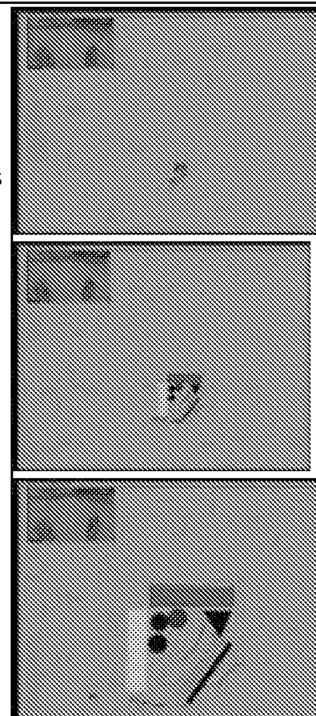
- Around 1992 we made a drafting table size desk that had a rear-projection data display, where the rear projection screen/table top was a translucent stylus controlled digital graphics tablet (Scriptel). The stylus was operated with the dominant hand. Prior to 1995 we mounted a camera bove the table top. It tracked the position of the non-dominant hand on the tablet surface, as well as the pose (open angle) between the thumb and index finger. The non-dominant hand could grasp and manipulate objects based on what it was over and opening and closing the grip on the virtual object. This vision work was done by a student, Yuyan Liu.
- Buxton,W. (1997). Living in Augmented Reality: Ubiquitous Media and Reactive Environments. In K. Finn, A. Sellen & S. Wilber (Eds.). *Video Mediated Communication. Hillsdale, N.J.: Erlbaum, 363-384. An earlier version of this chapter also appears in Proceedings of Imagina '95, 215-229.*

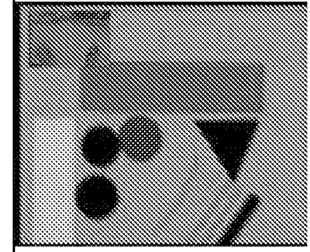


Simultaneous bimanual and multi-finger interaction on large interactive display surface

1997: T3 (Alias|Wavefront, Toronto)

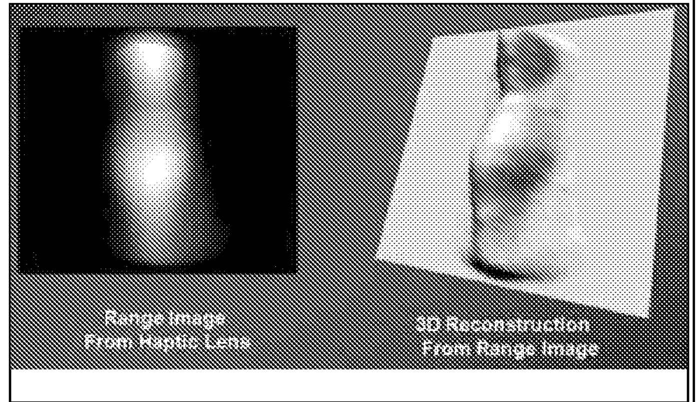
- T3 was a bimanual tablet-based system that utilized a number of techniques that work equally well on multi-touch devices, and have been used thus.
- These include, but are not restricted to grabbing the drawing surface itself from two points and scaling its size (i.e., zooming in/out) by moving the hands apart or towards each other (respectively). Likewise the same could be done with individual graphical objects that lay on the background. (Note, this was simply a multi-point implementation of a concept seen in Ivan Sutherland's Sketchpad system.)
- Likewise, one could grab the background or an object and rotate it using two points, thereby controlling both the pivot point and degree of the rotation simultaneously. Ditto for translating (moving) the object or page.
- Of interest is that one could combine these primitives, such as translate and scale, simultaneously (ideas foreshadowed by Fitzmaurice's graspable interface work – above).
- Kurtenbach, G., Fitzmaurice, G., Baudel, T. & Buxton, W. (1997). The design and evaluation of a GUI paradigm based on tabets, two-hands, and transparency. *Proceedings of the 1997 ACM Conference on Human Factors in Computing Systems, CHI '97,* 35-42. [Video].





1997: The Haptic Lens (Mike Sinclair, Georgia Tech / Microsoft Research)

- The Haptic Lens, a multi-touch sensor that had the feel of clay, in that it deformed the harder you pushed, and resumed it basic form when released. A novel and very interesting approach to this class of device.
- Sinclair, Mike (1997). The Haptic Lens. *ACM SIGGRAPH 97 Visual Proceedings: The art and interdisciplinary programs of SIGGRAPH '97,* Page: 179

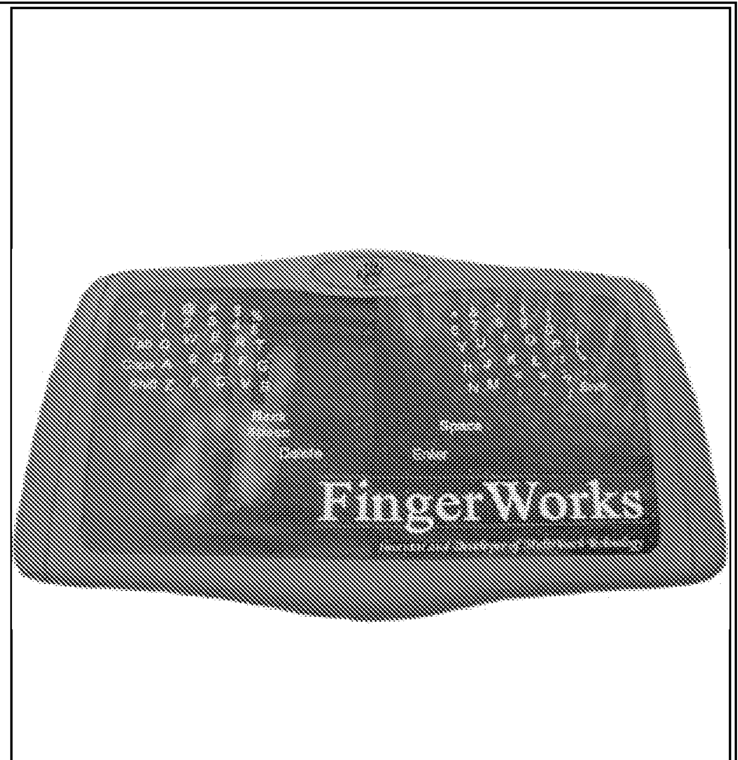


1998: Tactex Controls (Victoria BC) http://www.tactex.com/

- Kinotex controller developed in 1998 and shipped in Music Touch Controller, the MTC Express in 2000.
- See video at: http://www.billbuxton.com/flip_keyboard_s.mov

~1998: Fingerworks (Newark, Delaware).

- Made a range of touch tablets with multi-touch sensing capabilities, including the *iGesture Pad*. They supported a fairly rich library of multi-point / multi-finger gestures.
- Founded by two University of Delaware academics, John Elias and Wayne Westerman
- Product largely based on Westerman's thesis: Westerman, Wayne (1999). *Hand Tracking,Finger Identification, and Chordic Manipulation on a Multi-Touch Surface.* U of Delaware PhD Dissertation: http://www.ee.udel.edu/~westerma/main.pdf
- Note that Westerman's work was solidly built on the above work. His thesis cites Matha's 1982 work which introduced multi-touch, as well as Krueger's work, which introduced - among other things - the pinch gesture. Of the 172 publications cited, 34 (20%) are authored or co-authored by me an/or my students.
- The company was acquired in early 2005 by Apple Computer.
- Elias and Westerman moved to Apple.
- Fingerworks ceased operations as an independent



company.
- However, it left a lot of fans, and documentation, including tutorials and manuals are still downloadable from: http://www.fingerworks.com/downloads.html

1999: Portfolio Wall (Alias|Wavefront,Toronto On, Canada)

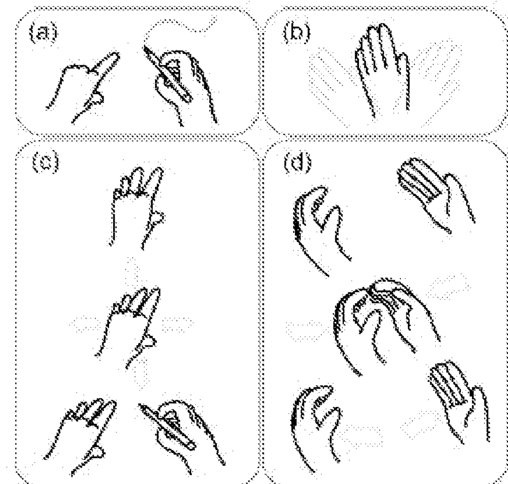
- A product that was a digital cork-board on which images could be presented as a group or individually. Allowed images to be sorted, annotated, and presented in sequence.
- Due to available sensor technology, did not us multi-touch; however, its interface was entirely based on finger touch gestures that went well beyond what typical touch screen interfaces were doing at the time, and which are only now starting to appear on some touch-based mobile devices.
- For example, to advance to the next slide in a sequence, one flicked to the right. To go back to the previous image, one flicked left.
- The gestures were much richer than just left-right flicks. One could instigate different behaviours, depending on which direction you moved your finger.
- In this system, there were eight options, corresponding to the 8 main points of the compass. For example, a downward gesture over a video meant "stop". A gesture up to the right enabled annotation. Down to the right launched the application associated with the image. etc.
- They were self-revealing, could be done eyes free, and leveraged previous work on “marking menus.”
- See a number of demos at: http://www.billbuxton.com/buxtonAliasVideos.html

Touch to open/close image
Flick right = next
Flick left = previous

Portfolio Wall (1999)

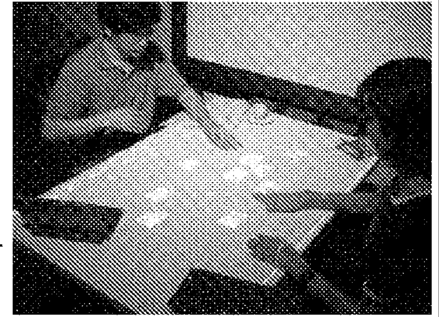
2001: Diamond Touch (Mitsubishi Research Labs, Cambridge MA)
http://www.merl.com/

- example capable of distinguishing which person's fingers/hands are which, as well as location and pressure
- various gestures and rich gestures.
- http://www.diamondspace.merl.com/



2002: Jun Rekimoto Sony Computer Science Laboratories (Tokyo)
http://www.csl.sony.co.jp/person/rekimoto/smartskin/

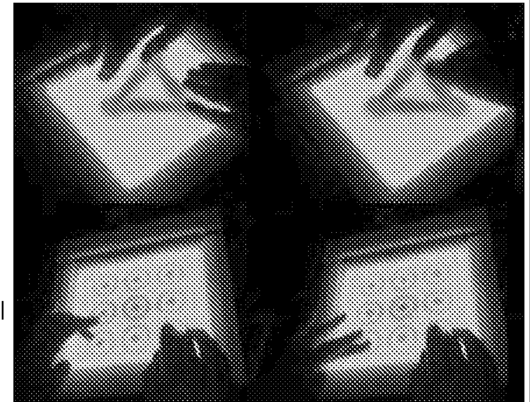
- *SmartSkin*: an architecture for making interactive surfaces that are sensitive to human hand and finger gestures. This sensor recognizes multiple hand positions and their shapes as well as calculates the distances between the hands and the surface by using capacitive sensing and a mesh-shaped antenna. In contrast to camera-based gesture recognition systems, all sensing elements can be integrated within the surface, and this method does not suffer from lighting and



occlusion problems.

- SmartSkin: An Infrastructure for Freehand Manipulation on Interactive Surfaces. *Proceedings of ACM SIGCHI*.
- Kentaro Fukuchi and Jun Rekimoto, Interaction Techniques for SmartSkin, ACM UIST2002 demonstration, 2002.
- SmartSkin demo at Entertainment Computing 2003 (ZDNet Japan)
- Video demos available at website, above.

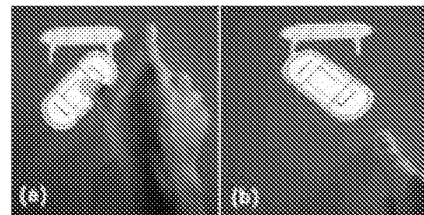
2002: Andrew Fentem (UK) http://www.andrewfentem.com/



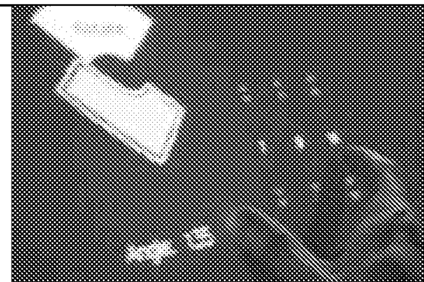
- States that he has been working on multi-touch for music and general applications since 2002
- However, appears not to have published any technical information or details on this work in the technical or scientific literature.
- Hence, the work from this period is not generally known, and - given the absence of publications - has not been cited.
- Therefore it has had little impact on the larger evolution of the field.
- This is one example where I am citing work that I have *not* known and loved for the simple reason that it took place below the radar of normal scientific and technical exchange.
- I am sure that there are several similar instances of this. Hence I include this as an example representing the general case.

2003: University of Toronto (Toronto)

- paper outlining a number of techniques for multi-finger, multi-hand, and multi-user on a single interactive touch display surface.
- Many simpler and previously used techniques are omitted since they were known and obvious.
- Mike Wu, Mike & Balakrishnan, Ravin (2003). Multi-Finger and Whole Hand Gestural Interaction Techniques for Multi-User Tabletop Displays. *CHI Letters*

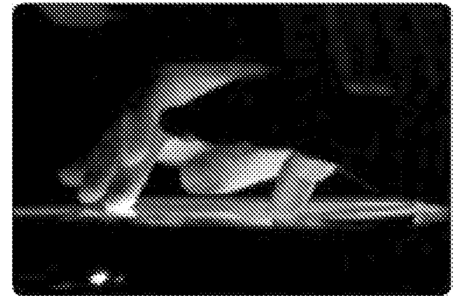


Freeform rotation. (a) Two fingers are used to rotate an object. (b) Though the pivot finger is lifted, the second finger can continue the rotation.



This parameter adjustment widget allows two-fingered manipulation.

2003: Jazz Mutant (Bordeaux France) http://www.jazzmutant.com/
Stantum: http://stantum.com/

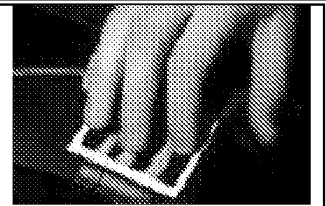


- Make one of the first transparent multi-touch, one that became – to the best of my knowledge – the first to be offered in a commercial product.
- The product for which the technology was used was the *Lemur,* a music controller with a true multi-touch screen interface.
- An early version of the Lemur was first shown in public in LA in August of 2004.
- Jazz Mutant is the company that sells the music product, while Stantum is the sibling company set up to sell the underlying multi-touch technology to other

2004: TouchLight (Andy Wilson, Microsoft Research): http://research.microsoft.com/~awilson/

- TouchLight (2004). A touch screen display system employing a rear projection display and digital image processing that transforms an otherwise normal sheet of acrylic plastic into a high bandwidth input/output surface suitable for gesture-based interaction. Video demonstration on website.
- Capable of sensing multiple fingers and hands, of one or more users.
- Since the acrylic sheet is transparent, the cameras behind have the potential to be used to scan and display paper documents that are held up against the screen .

2005: Blaskó and Steven Feiner (Columbia University):
http://www1.cs.columbia.edu/~gblasko/



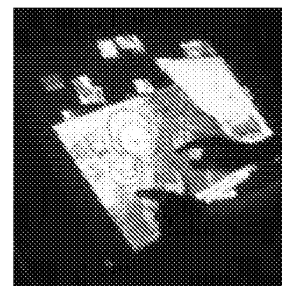
- Using pressure to access virtual devices accessible below top layer devices
- Gábor Blaskó and Steven Feiner (2004). Single-Handed Interaction Techniques for Multiple Pressure-Sensitive Strips,
  Proc. ACM Conference on Human Factors in Computing Systems (CHI 2004) Extended Abstracts, 1461-1464

2005: PlayAnywhere (Andy Wilson, Microsoft Research):
http://research.microsoft.com/~awilson/

- PlayAnywhere (2005). Video on website
- Contribution: sensing and identifying of objects as well as touch.
- A front-projected computer vision-based interactive table system.
- Addresses installation, calibration, and portability issues that are typical of most vision-based table systems.
- Uses an improved shadow-based touch detection algorithm for sensing both fingers and hands, as well as objects.
- Object can be identified and tracked using a fast, simple visual bar code scheme. Hence, in addition to manual mult-touch, the desk supports interaction using various physical objects, thereby also supporting graspable/tangible style interfaces.
- It can also sense particular objects, such as a piece of paper or a mobile phone, and deliver appropriate and desired functionality depending on which..

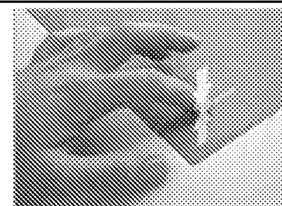
2005: Jeff Han (NYU): http://www.cs.nyu.edu/~jhan/

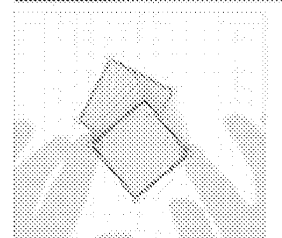
2006: (Perceptive Pixel: http://www.perceptivepixel.com/)



- Very elegant implementation of a number of techniques and applications on a table format rear projection surface.
- Multi-Touch Sensing through Frustrated Total Internal Reflection (2005). Video on website.
- Formed Peceptive Pixel in 2006 in order to further develop the technology in the private sector
- See the more recent videos at the Perceptive Pixel site: http://www.perceptivepixel.com/

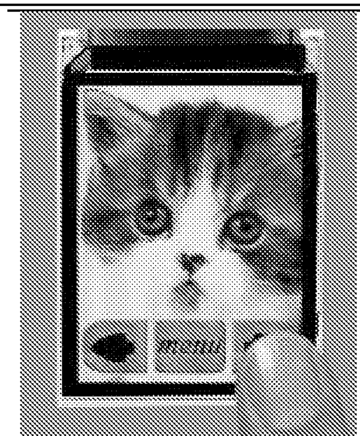
2005: Tactiva (Palo Alto) http://www.tactiva.com/





- Have announced and shown video demos of a product called the TactaPad.
- It uses optics to capture hand shadows and superimpose on computer screen, providing a kind of immersive experience, that echoes back to Krueger (see above)
- Is multi-hand and multi-touch
- Is tactile touch tablet, i.e., the tablet surface feels different depending on what virtual object/control you are touching

2005: Toshiba Matsusita Display Technology (Tokyo)



- Announce and demonstrate LCD display with "Finger Shadow Sensing Input" capability
- One of the first examples of what I referred to above in the 1991 Xerox PARC discussions. It will not be the last.
- The significance is that there is no separate touch sensing transducer. Just as there are RGB pixels that can produce light at any location on the screen, so can pixels detect shadows at any location on the screen, thereby enabling multi-touch in a way that is hard for any separate touch technology to match in performance or, eventually, in price.
- http://www3.toshiba.co.jp/tm_dsp/press/2005/05-09-29.htm

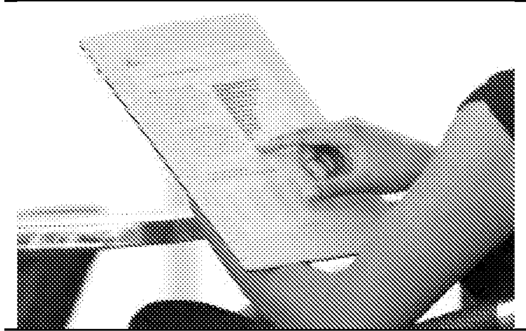
2005: Tomer Moscovich & collaborators (Brown University)

- a number of papers on web site: http://www.cs.brown.edu/people/tm/
- T. Moscovich, T. Igarashi, J. Rekimoto, K. Fukuchi, J. F. Hughes. "A Multi-finger Interface for Performance Animation of Deformable Drawings." Demonstration at *UIST 2005 Symposium on User Interface Software and Technology*, Seattle, WA, October 2005. (video)

2006: Benko & collaborators (Columbia University & Microsoft Research)

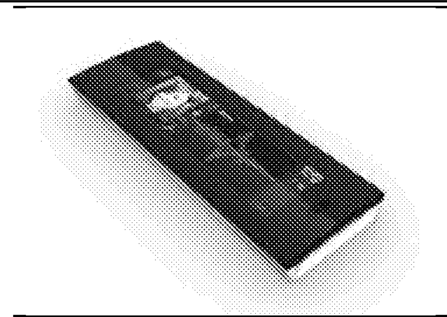
- Some techniques for precise pointing and selection on muti-touch screens
- Benko, H., Wilson, A. D., and Baudisch, P. (2006). Precise Selection Techniques for Multi-Touch Screens. *Proc. ACM CHI 2006 (CHI'06: Human Factors in Computing Systems,* 1263–1272
- video

2006: Plastic Logic (Cambridge UK)



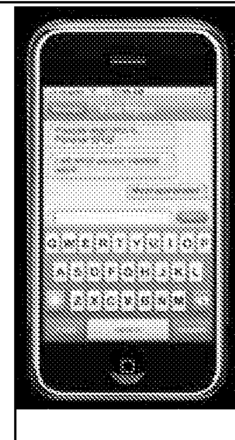
- A flexible e-ink display mounted over a multi-point touch pad, thereby creating an interactive multi-touch display.
- Was an early prototype of their ill-fated QUE e-reader

2006: Synaptics & Pilotfish (San Jose) http://www.synaptics.com



- Jointly developed Onyx, a soft multi-touch mobile phone concept using transparent Synaptics touch sensor. Can sense difference of size of contact. Hence, the difference between finger (small) and cheek (large), so you can answer the phone just by holding to cheek, for example.
- http://www.synaptics.com/onyx/

2007: Apple iPhone http://www.apple.com/iphone/technology/



- Like the 1992 Simon (see above), a mobile phone with a soft touch-based interface.
- Outstanding industrial design and very smooth interaction.
- Employed multi-touch capability to a limited degree
- Uses it, for example, to support the "pinching" technique introduced by Krueger, i.e., using the thumb and index finger of one hand to zoom in or out of a map or photo.
- Works especially well with web pages in the browser
- Uses Alias Portfolio Wall type gestures to flick forward and backward through a sequence of images.
- Did not initially enable use of multi-touch to hold shift key with one finger in order to type an upper case character with another with the soft virtual keyboard. This did not get implemented until about a year after its release.

2007: Microsoft Surface Computing http://www.surface.com

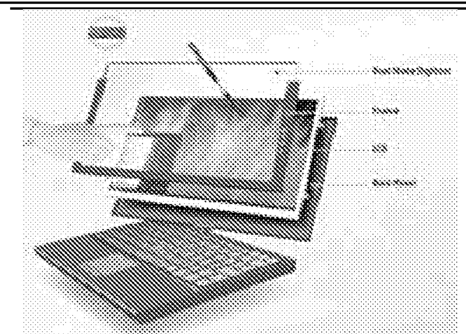
- Interactive table surface
- Capable of sensing multiple fingers and hands
- Capable of identifying various objects and their position on the surface
- Commercial manifestation of internal research begun in 2001 by Andy Wilson (see above) and Steve Bathiche
- Image is displayed by rear-projection and input is captured opticaly via cameras
- A key indication of this technology making the transition from research, development and demo to mainstream commercial applications.
- See also ThinSight and Surface 2.0

2007: ThinSight, (Microsoft Research Cambridge,UK)
http://www.billbuxton.com/UISTthinSight.pdf

- Thin profile multi-touch technology that can be used with LCD displays.
- Hence, can be accommodated by laptops, for example
- Optical technology, therefore capable of sensing both fingers and objects
- Therefore, can accommodate both touch and tangible styles of interaction
- Research undertaken and published by Microsoft Research
- see also Surface 2.0

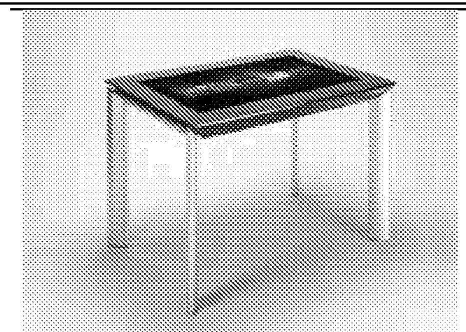
2008: N-trig http://www.n-trig.com/

- Commercially multi-touch sensor
- Can sense finger and stylus simultaneously
- unlike most touch sensors that support a stylus, this incorporates specialized stylus sensor
- result is much higher quality digital ink from stylus
- Incorporated into some recent Tablet-PCs
- Technology scales to larger formats, such as table-top size



2011: Surface 2.0 (Microsoft & Samsung) http://www.microsoft.com/surface/

- 4" thick version of Surface
- Rear projection and projectors replaced by augmented LCD technology
- builds on research such as ThinSight
- result is more that just a multi-touch surface
- since pixels have integrated optical sensors, the whole display is also an imager
- hence, device can "see" what is placed on it, including shapes, bar-codes, text, drawings, etc. - and yes - fingers



One-Point Touch Input of Vector Information for Computer Displays

Christopher F. Herot*
Guy Weinzapfel
Architecture Machine Group
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

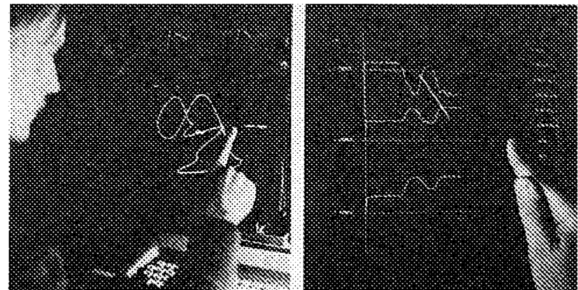
The finger as a graphical stylus enjoys a coefficient of friction with glass sufficient to provide input of direction and torque as well as position from a single point. This report describes a pressure-sensitive digitizer (PSD) capable of accepting these force inputs, and discusses a set of five simple input applications used to assess the capabilities of this device. These applications include techniques for specifying vectors, and pushing, pulling, dispersing and reorienting objects with a single touch. Experience gained from these applications demonstrates that touch and pressure sensing open a rich channel for immediate and multi-dimensional interaction.

Key Words: Touch Input, Pressure Sensing, Force Input, Tactile Input, Kinesthetic Input, Pressure Sensitive Digitizer, Touch Sensitive Digitizer.

1.0 INTRODUCTION

It is a central thesis of the Architecture Machine Group, that work places as opposed to work stations, are a necessary ingredient for the amplification of creativity. (1) Work places are defined as having a multiplicity of interactive media which encourage a high degree of motor involvement - tactile participation. By austere comparison, work stations are characterized by the all-too-prevalent black and white CRT with its keyboard and occasional light pen or other stylus. The need for multimedia is based on the assertion that, regardless of task, information relating to creative performance is best perceived through a variety of senses including at the least sight, sound, and touch. While several of our current projects explore the integration of multiple media (2,3,4), this paper reports on one effort to develop a channel of tactile input.

Recently, interest has grown around a class of instruments known as touch sensitive digitizers (TSDs). Using a variety of technologies (5), these devices are capable of determining the X, Y position of a finger's touch without resorting to an intermediate physical stylus.



The excitement generated by TSDs derives directly from their ability to provide a more natural input path to the computer. The umbilical cord attached to the conventional stylus is removed; in fact the entire notion of a physical stylus is voided. Also, dislocations caused by separate input and presentation surfaces can be circumvented by superimposing transparent TSDs directly over display surfaces.

While the potentials for more natural, coincident and even multi-finger input are obvious and are being developed by other programs(6), little exploration has been undertaken in the area of multi-dimensional input - the sensing of pressure as well as location parameters(7,8,9). Yet this domain offers a rich potential for man-machine interaction. The work described in the following pages was designed to explore that potential - to test the ability of the human finger to input variable pressure and direction from a single touch.

The work reported herein was conducted between July 1, 1977, and October 31, 1977, under Army Research Institute Grant number DAHC19-77-G-0014, Nicholas Negroponte, principal investigator.

* Mr. Herot's current address: Computer Corporation of America, 575 Technology Square, Cambridge, Massachusetts.

1.1 OUR LABORATORY'S TSD.

In April, 1976, the Architecture Machine Group acquired a TSD from Instronics, Ltd. of Ontario, Canada. This device consists of a sheet of clear glass with piezoelectric transducers mounted on two adjacent edges. The glass is doubly curved to match the face of a display tube(10). The transducers are used to induce acoustic waves in the surface of the glass. These waves are reflected back to their source by fingers touched to the glass surface. The location of the touch is determined by ranging those echos(11).

It was hoped that the TSD would enable users to sweep their fingers over the display surface, thus drawing, even "fingerpainting," with the computer. It was found, however, that in order to insure proper input readings, users had to press the TSD with a force that generated friction between finger and glass sufficient to prevent smooth, sweeping gestures. As a result, the device seemed better suited to pointing than to drawing or painting.

This reality, however, opened the possibility of using the finger-glass friction to unique advantage. Namely, the TSD could be mounted on the display with strain gauges such that forces induced by the finger could be used to input pressures both normal to and parallel with the input surface. In this way, the device could become a pressure-(as well as touch-) sensitive digitizer - a TSD/PSD.

Such a configuration was implemented (as described in Section 3.0) and provided the basis for a four month research program designed to evaluate the characteristics of pressure sensitive input. The following section discusses the methods used to conduct that evaluation.

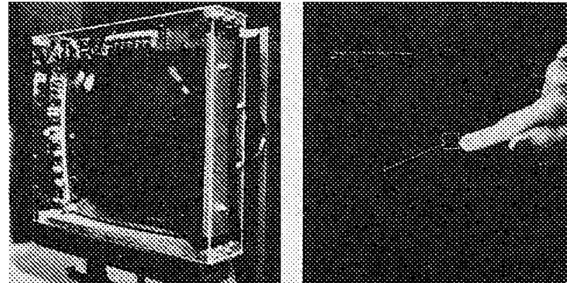
2.0 APPLICATIONS.

Five input routines were developed to assess the input characteristics of the PSD. These included:

1. Force Cursor,
2. Vector History,
3. Pushing/Pulling,
4. Dispersion, and
5. Rotation

In addition, an attempt was made to utilize the X and Y torques to determine the position of the finger, so as to eliminate the need for a TSD altogether.

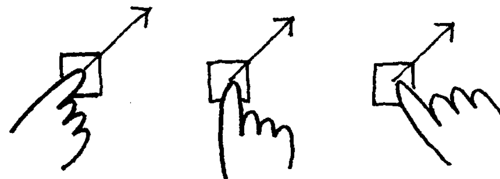
Due to the short duration of the project, evaluation of the device was limited to informal use of the five input routines by a diverse user population, consisting of the laboratory staff and the many visitors which the laboratory attracts from computer science, the arts and various industries. No attempt was made to quantify improvements in throughput, productivity, or task enjoyment resulting from use of the device. However, all users agreed that improvements were indicated in each of these areas.



2.1 FORCE CURSOR.

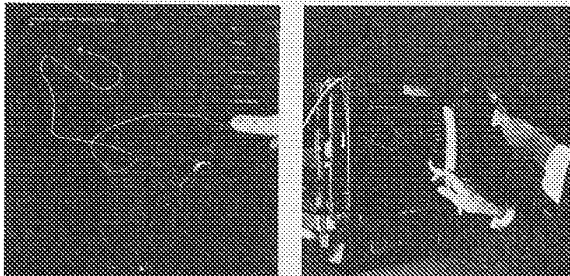
The initial routine provides the pressure sensing equivalent of a conventional cursor - that is, a graphic feedback mechanism which shows the user what is being input. The routine does this by displaying a vector, or arrow, whose origin coincides with the touch point, whose head lies in the direction of the force being exerted by the finger, and whose length is proportional to that force. At the same time, the z force (pressure normal to the face of the screen) is reported as a square, whose size is proportional to that force.

Use of the force cursor has produced some surprising results. While its function is obvious to all who observe it, many people experience initial difficulty making it behave as they expect. Most notably, novice users have difficlty making the vector point in the directions they desire. This difficulty derives not from the equipment, but from the fact that people do not always press in the direction which their finger appears to indicate. Typically, this problem is encountered with the user's first vector. The novice will press the surface of the device, causing an arrow to appear in proper alignment with the finger; but as the finger is rotated, the direction of the vector often fails to follow. Close observation has revealed that this results from the fact that the user actually maintains pressure in the original direction though the finger changes orientation.



Fortunately, the learning curve with this routine is quite steep. This most certainly has to do with the fact that the device takes advantage of the user's existing eye-hand coordination skills. Following some initial difficulty, most users are able to control the direction of their vectors with less than a minute's practice. In fact, many users, realizing that the orientation of their fingers is irrelevant to the direction of the vector, are able to manipulate the cursor from a single, natural hand position.

Beyond this initial training problem, there was a more chronic difficulty: placing the tip of the arrow with acceptable accuracy. This was expecially true as greater extensions (and hence larger forces) were attempted. This problem is similar to that of using a long pointer at a blackboard; the vector bobbed and wobbled at its greatest extension. To counteract this drawback, a damping effect was added to the cursor routine to filter out minor pressure fluctuatons. This filter proved a sufficient solution, as users are now able to point at specific targets (e.g., the menu labels) from origins well across the screen. The force cursor demonstrates that the PSD can be used for reasonably accurate inputs of direction and magnitude.



2.2 VECTOR HISTORY.

The second routine was designed to evaluate the potential for guiding a cursor from a stationary input position. In this case, the cursor scribed a path as it moved under control of the finger's pressure. This routine underwent two implementations. In the first, the speed of the cursor was constant; only its direction was controlled through the PSD. A later implementation allowed the speed to be controlled as well.

Most users, having trained with the force vector, encountered little difficulty in directing the mobile cursor. For example, many people were able to write their names on their first attempt.

Surprisingly, though, the variable-speed version was more difficult to use. This results from the fact that as the cursor deviates from an intended path, most people's reaction is to press harder on the input surface. Since this does not necessarily change the cursor's direction but does increase its speed, "errors" are exaggerated.

Nonetheless, the process of controlling a mobile cursor from a single point on the screen appears to be an engaging and successful use of the device. Real world applications (such as navigating about a map display) can easily be imagined for its use.

2.3 PUSHING/PULLING.

To explore the PSD's potential for moving objects other than a cursor, a routine was implemented which allows users to move objects about the screen. This routine differs from the previous capability, as a specific object is indicated simultaneously with the input of force. Here, the user points to an object and gives it a push. The touch is used to identify the object, the pressure to impart a direction and speed. Thereafter, the user does not need to track the object with the finger but can direct its movement from a static position. Use has demonstrated this routine to be a viable means for directing the movement of selected objects, as users are able to reposition objects with considerable ease.

It was hoped that this routine might also provide users with a sense for the relative "weights" of displayed objects. To test this potential, the routine was elaborated to incorporate parameters for differentially weighted objects. That is, the routine would cause a "lighter" object to move in response to a lighter touch than that required for a "heavier" object.

However, the routine failed to provide the desired perception of weighted objects. This failure was attributed to the absence of an essential mode of feedback from the input device - namely, the tactile/kinesthetic sensation of the object's physical displacement. When a person pushes an object in the natural environment, the weight of the object is reported not only by the pressure returned to the finger, but the movement which is both seen by the eyes and felt by the finger. In short, several feedback channels coalesce to impart a coherent perception of the object's physical properties(12). In the case of the pressure-sensitive device, there is a conflict between the kinesthetic response of the real object (the glass surface), which the finger reports as stationary, and the virtual object which the eyes

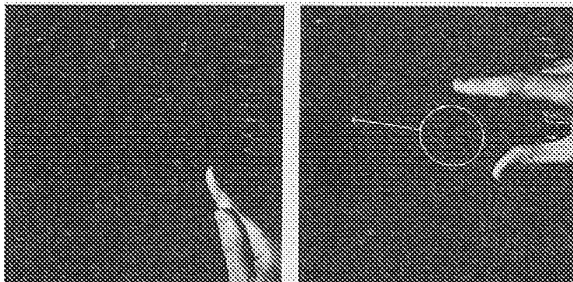
report as moving. This conflict is sufficient to impair the appraisal of the object's weight reported by the finger-sensed pressure. This is not to say that users gained no perception of weightings, for it was clear to all users that some objects moved more easily than others. But no one was able to say that one object was "twice as heavy" as another.

Nonetheless, it should be emphasized that the limited perceptions of weight did not impair the user's ability to manipulate the objects. Most users were equally comfortable using either routine to relocate objects.

2.4 DISPERSION.

In perhaps the most engaging of all the PSD applications, a graphic "shooting gallery" was devised to test the device's ability to accommodate inputs which disperse numbers of elements in various directions. This routine causes small, BB-like circles to emanate from the user's finger tip as it is pressed on the screen's surface. The number, speed, and direction of the BB's is controlled by the pressure of the user's finger. A procession of moving targets (in fact, small ducks) is played across the top of the screen to test the accuracy of the users "shots."

Interestingly enough, even users who had experienced some difficulties with the previous routines adapted to the requirements of this application quite rapidly. In fact, some "hunters" advanced to the point where selected ducks could be felled with single shots. This calls for very accurate control indeed.



2.5 ROTATION.

The fifth routine was designed to evaluate the PSD's ability to measure torque inputs and to use those measurements to advantage in interaction. For this purpose, a simple knob is displayed on the screen with an arrow indicating its angular position. It was hoped that torque about the z axis could be measured with sufficient sensitivity that even minute twists of a single finger could be used to turn the displayed knob. However, when the device was tuned to a level sensitive enough to measure these subtle inputs, the user's intentions were overshadowed by vibrations in the room and in the equipment itself. Once the sensitivity of the z torque pickup was lowered, it became possible for users to turn the knob with two fingers. In fact, the position of the knob can be adjusted to within 5 degrees of rotation with little difficulty. Further tuning of the algorithm and the hardware might permit even greater accuracy.

Though a single, rather sizeable knob was used for this application, the success achieved opens numerous additional possibilities. For example, specific machine parts in a complex display could be identified and reoriented via simple, direct manipulation, thus obviating the need for multiple commands for object selection and action specification.

2.6 POSITION DETECTION.

In addition to the more elaborate capabilities described above, it was hoped that a means of detecting the position of a surface touch could be accomplished directly by the PSD without using the echo ranging of the Instronics device. The algorithm used for this measurement divided the X and Y torques by the Z force. The results of this function were normalized for the direction of forces parallel to the input surface and amplified to produce the location of the finger. This approach produced a calculated touch point with a resolution equal to that of the TSD, but the locus of the point was influenced by the force and direction of the touch. The source of this influence was never adaquately understood, and no solution was conceived in the course of the study (13).

2.7 EXTENSIBILITY.

It should be noted that the applications described above were selected because they could all be accomplished in the time available. It was clear from the onset how each capability should work, and the amount of programming required for each was quite limited. In short, the routines were appropriately matched to the four month duration of the research.

It is not difficult, however, to conceive of more elaborate uses for a pressure sensitive device. For example, a three dimensional dynamic modeling system could use the PSD for tactile manipulation of machine parts, building volumes, and the like. It is easy to imagine turning a machine part by twisting its representation on the screen, or rotating a building display by pushing on a corner. In short, the potentials for tactile involvement and physical feedback from such a device were only hinted at by this brief exploratory work.

3.0 PRINCIPLES OF OPERATION.

The PSD employs eight strain gauges, two each secured to mounting rings centered on the four sides of the TSD. Of the two gauges secured to each ring, one measures force perpendicular to the glass and the other measures shear parallel to the glass. These eight measurements are then used to derive the three force and three torque outputs which are used by the routines described in the previous section.

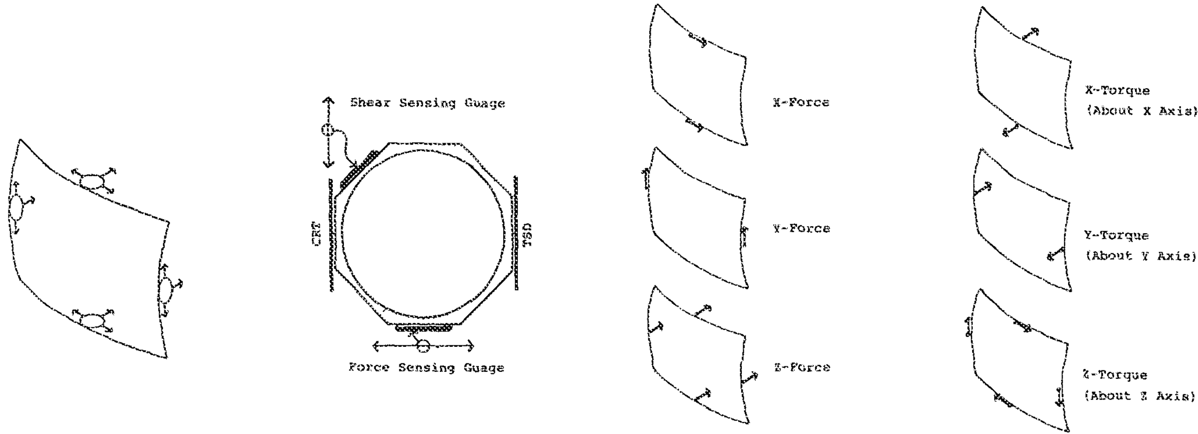
3.1 MOUNTING AND STRAIN GAUGES.

The TSD is secured to the CRT by means of four specially machined, octagonal, aluminum rings. All forces exerted on the TSD are transmitted to these rings, thus causing deformations which in turn flex the strain gauges secured to them. The two gauges are cemented to each ring as shown in the adjacent figure. Their placement insures that the forces which they sense are orthogonal to one another.

3.2 ELECTRICAL DESCRIPTION.

The PSD utilized nine BLH (SPB3-35-500) semiconductor strain gauges. Semiconductor gauges were selected because of their sensitivity to miniscule strains. However, as semiconductor devices, they are also very sensitive to changes in temperature. Accordingly, a ninth gauge mounted such that no strain could be exerted upon it is employed to provide a reference output to which all other gauges can be compared. The gauge outputs, which vary between plus and minus 10 millivolts peak to peak, are each connected to preamplifiers which impart a gain of 50; the resultant "raw" output is .5 volts peak to peak.

The "raw" voltages from the strain gauge preamplifiers are combined by sum and difference networks to produce outputs which correspond to X force, Y force, X moment, Y moment, and Z moment. The sums of opposite torque gauges are used to provide the torques about each axis.



It happens that the thickness, and hence flexibility of these rings is critical to the sensitivity of the gauge's measurements. Unfortunately, the rings machined for this implementation were designed to accommodate very subtle pressures; the fact that the TSD necessitates high finger pressures was not taken into account in their design. Nor was the vibration from nearby machinery foreseen as a problem. As a result, development of the five input routines was somewhat hampered by vibration and pressures which exceeded the output range of the gauges and related circuitry. Were the equipment to be rebuilt, heavier rings would greatly improve its performance. Alternatively, load cells, rather than strain gauges might be used. Load cells measure pressure without deformation. However, these devices are significantly more expensive than the strain gauges used for this implementation.

The six force and torque outputs are converted to digital signals by a Burr-Brown SDM853 data acquisition system (DAS). The inputs to the DAS are limited to 6.2 volts to prevent overloading the A/D converters. The DAS produces a 12 bit output for each of the 6 analogue inputs.

3.3 DIGITAL INTERFACE.

The outputs from the DAS are stored in a buffer, allowing the DAS to assemble the next sample while waiting for the computer to read the current values.

The computer interface allows program selection of either byte or halfword mode. In byte mode, only 8 most significant bits of each force and torque are used, allowing fast and easy access to the data. In halfword mode, the programing is a bit more complicated, but all of the data bits

are available. Due to the influence of vibration on low order bits, the device was operated primarily in byte mode for the experiments described here.

3.4 SYSTEMS SOFTWARE.

The PSD is equipped to interrupt the computer when data is available. However, since the PSD is always used in conjunction with the TSD, the interrupt circuitry of that device was used. When the TSD detects the finger touch, the program reads the position from the TSD and the forces and torques from the PSD. Since hysteresis of the strain rings and uncompensated temperatre drift often cause the untouched PSD to produce non-zero readings, it is additonally important that the TSD interrupt be used. When the software detects that the device is not being touched, it reads the values of the forces/torques so as to use them as zero references the next time the PSD is touched.

Drift due to temperature changes generated problems for the initial input routines. This was overcome by adding software to sample the force and torque readings when the TSD was not being touched. The latest readings, then, were used as offsets for subsequent inputs. However, this software compensation was made at the expense of the system's overall response range: the offsets biased the device unpredictably. A zeroing circuit was designed to correct for temperature drift in hardware. This circuit was not installed due to the short duration of the study and the anticipated cost associated with its installation.

4.0 CONCLUSIONS.

Development of the PSD and related input routines was undertaken in order to determine answers to several questions. First, we wished to know if it was technically feasible to measure finger pressures on a sheet of glass and to decompose those pressures into their X, Y, Z force and torque components. That question has been answered in the affirmative.

Second, the work was conducted to determine if force and torque inputs could be applied with sufficient accuracy and control to be useful for man-machine communication. All of the input routines indicate that accuracy presents no serious problem, especially where continuous, real-time, graphic feedback is provided (as in the Force Cursor and Rotation routines). Vector History indicates that flexible, easily controlled interaction is possible as well. However, this routine also shows that force input is more suited to the modulation of velocities than for the control of accelerations.

Third, the input routines were used to determine if a pressure-sensitive device could convey more natural perceptions of virtual objects. While limited success was achieved in conveying the differential weights of objects, the quality of such perceptions is only marginally improved by the use of the PSD. It would be misleading to rely upon the device as a mechanism for providing passive force feedback.

Finally, the PSD and its routines were developed to explore any unforeseen benefits which might accrue from the implementation of such a device. Here, two definite advantages can be identified. First, the PSD/TSD combination affords engaging and facile interaction which attracts and maintains the participation of all who witness its use. Second, the device has proven innately simple to use. By capitalizing on natural skills, the PSD enables users to take advantage of virtually all its capabilities within minutes. At a recent open house it was astounding to see four-and five-year-old children pointing at words with the vector, turning the knob about and shooting ducks with obvious glee.

Of course, the PSD's ultimate advantage is its ability to collapse activities which otherwise require several disjoint commands into single, natural, tactile actions.

ACKNOWLEDGMENTS.

The work has been very much a group project, initially launched by experiments with touch sensitive display (TSD) devices, conducted by Richard Bolt and under ARPA contract number MDA-903-76-C-0261, April 1, 1976, to September 30, 1976. During that period, William Donelson, a graduate student, speculated that translational and Z-forces could be sensed by strain gauges. William Kelley, assisted by Robert Hoffman, John Soltes, and Harry Boadwee, constructed the hardware outlined in Section 3.0. Peter Clay, building upon earlier TSD software by Rimas Ignaitis, implemented the input routines outlined in Section 2.0. Finally, Michael Naimark made the film which is to be shown at SIGGRAPH 1978. This long story attributes the conjoint efforts of our laboratory, as encouraged by Dr. Frank Moses of the Army Research Institute.

FOOTNOTES/REFERENCES

1. Nicholas Negroponte. On being creative with computer aided design. In Information Processing 77, B. Gilchrist, Editor, IFIP, North-Holland Publishing Co., New York, 1977.

2. Dr. Richard A. Bolt. Spatial data - management - interim report. Architecture Machine Group, M. I. T. Cambridge, Massachusetts, November, 1977.

3. William Donelson. Spatial management of information. SIGGRAPH '78, Proceedings of the Fifth Annual Conference on Computer Graphics and Interactive Techniques. Atlanta, 1978.

4. Guy Weinzapfel. Mapping by yourself. Proceedings of the conference Interactive Techniques in Computer Aided Design, Bologna, Italy, 1978.

5. A discussion of various TSD technologies is provided by Dr. Richard A. Bolt. Touch sensitive displays. Architecture Machine Group, M.I.T. Cambridge, Massachusetts, September, 1976.

6. TSD work is continuing under the aegis of ONR Contract Number N00014-75-C-0460 with Elographics Corporation furnishing a transparent sensing medium laminated to a Tektronix 650 display.

7. Several "kinesthetic" systems, that is systems incorporating force feedback, are summarized by Frederick P. Brooks, Jr. The computer "scientist" as toolsmith - studies in interactive graphics. In Information Processing 77, B. Gilchrist, Editor, IFIP, North-Holland Publishing Co., New York, 1977. However, none of the systems summarized in this paper correlates physical and grahical feedback at a common locus.

8. P.J. Kilpatrick. The use of a kinesthetic supplement in an interactive graphics system. Ph.D. dissertation, University of North Carolina, Chapel Hill, N.C., 1976.

9. A.M. Noll. Man-machine tactile communication. Ph.D. dissertation, Polytechnic Institute of Brooklyn, Brooklyn, N.Y., 1971.

10. The display used for this project is an IMLAC PDS-1 dynamic CRT. This dislay may be driven by any of the laboratory's several Interdata minicomputers (models 70, 85 or 7/32). The operating system (MAGIC) and the display software are both of Architecture Machine Group design.

11. A more complete technical description of the TSD is provided in the reference cited in (5) above.

12. A general discussion of task performance related to diminished and augmented feedback is provided in Paul M. Fitts and Michael I. Posner. Human Performance. Brooks/Cole Publishing Co., Belmont, California, 1967.

13. Subsequent to this study, it was learned that Robert Anderson and Ivan Sutherland, working at Rand Corporation had explored a pressure-sensing device to locate the x,y position of a touch. Though the Rand device used a considerably different mounting configuration, the calculated touch point migrated as the touch pressure was varied. To minimize this problem, positions were calculated using a low pressure threshold.

Department of Computer & Information Science

Technical Reports (CIS)

University of Pennsylvania *Year* 1981

Real Time Control of a Robot Tacticle Sensor

Jeffrey A. Wolfeld
University of Pennsylvania

This paper is posted at ScholarlyCommons.
http://repository.upenn.edu/cis_reports/678

UNIVERSITY OF PENNSYLVANIA
THE MOORE SCHOOL OF ELECTRICAL ENGINEERING
SCHOOL OF ENGINEERING AND APPLIED SCIENCE

REAL TIME CONTROL OF A ROBOT TACTILE SENSOR

Jeffrey A. Wolfeld

Philadelphia, Pennsylvania

August, 1981

A thesis presented to the Faculty of Engineering and Applied Science in partial fulfillment of the requirements for the degree of Master of Science in Engineering for graduate work in Computer and Information Science.

____
Ruzena Bajcsy

____
Aravind K. Joshi

The work reported here was supported in part by NSF grant number MCS-78-07466.

REAL TIME CONTROL OF A ROBOT TACTILE SENSOR

Jeffrey A. Wolfeld

Philadelphia, Pennsylvania

August 1981

Abstract

The goal of the Experimental Sensory Processor project is to build a system which employs both visual and tactile senses, and then explore their interaction in a robotic environment. Here we describe the software involved in the low level control of the tactile branch of this system, and present results of some simple experiments performed with a prototype tactile sensor.

Acknowledgments

I would like to thank the following people:

Jim Korein, my office mate, with whom I had many fascinating discussions between 9:00 and 5:00 on weekdays;

Gerry Radack, who occasionally dragged me away from my terminal in order to play music;

Clayton Dane, who helped keep my feet on the ground;

Jeff Shrager, without whom I might never have gotten past the Abstract;

Taylor Adair, who kept the computer running when it really wanted to crash;

Ira Winston, who served as the local oracle;

Jack Rebman of the Lord Corporation, without whom this thesis would have been entirely speculation;

David Brown, who got me into this mess in the first place;

and my advisor, Ruzena Bajcsy, mother to us all.

Table of Contents PAGE

1. Introduction 2

1.1 Motivation. 2

1.2 Project Overview. 4

2. Proposed Microprocessor Software 9

2.1 Processors. 10
2.1.1 Tactile Sensing Processor 11
2.1.2 Motor Control Processor 13

2.2 Cross-Sectional Scan Command. 17

3. The Implemented Software 22

3.1 Environmental Details 22

3.2 Command Format and Interpretation 23

3.3 Motor Control. 26

3.4 Tactile Data Acquisition 33

4. Experiments and Results 35

4.1 Calibration 35

4.2 Static Tactile Image Analysis 37
4.2.1 Single Image. 37
4.2.2 Spatial Resolution. 39
4.2.3 Multiple Images. 40
4.2.4 Large Objects 41
4.2.5 Small Angle Measurement 42

4.3 Dynamic Texture Analysis 44

4.4 Conclusions 48

5. Further Work 50

Copyright 1981 by Jeffrey Wolfeld

Chapter 1: Introduction

1.1 Motivation

Artificial Intelligence researchers have worked extensively with vision systems in an attempt to give computers, and eventually robots, a sense of sight. A great deal of this research has been directed toward overcoming certain basic inadequacies in our current technology. For example, imperfect light sensors dictate that noise must be eliminated or tolerated. Insufficient spatial resolution requires routines which will interpolate below the pixel level.

One of the most important problems is that a camera produces a two-dimensional image of a three-dimensional scene. This invalidates an assumption which one would like to rely upon -- that two adjacent points in the image are adjacent in the scene. Therefore, substantial effort has been devoted to reproducing 3-D data from one or several visual images. Tactile sensors can be used to aid the process.

An imaging tactile sensor, by its very nature, does not have the problem. Since it produces a two-dimensional image

of a two-dimensional scene, it does not provide as much information, but it yields useful information clearly, without the need for complicated heuristics.

We can take this one step further. Suppose a tactile sensor is mounted on some kind of computer controlled 3-D positioning device. Then, by moving the sensor to different points on a target object, the computer can actually obtain 3-D data directly, and much more selectively. If this information is used to supplement and augment visual data, a great deal of processing may be avoided.

One can come up with many other uses for varying kinds of tactile sensors. Briot [BRIOT-79] demonstrated that tactile sensors mounted on the fingers of a robot hand can be used to determine the position, orientation, and perhaps even the identity of an object which it has grasped. He also showed that a grid of pressure sensitive sites on a table can tell a robot the location, orientation, and again, the identity of a part. It should be possible with multi-valued pressure sensors, as opposed to binary sensors, to determine the mass of the object. When the angle is small, a tactile sensor can be used to compute the angle between it and the object being grasped, possibly with a view toward improving the grip. Also, if the device is sensitive enough, it can be an invaluable aid to a robot attempting to grasp a fragile object without breaking it. Finally, a tactile sensor makes it possible to incorporate the properties of surface texture

and resilience into the object recognition process.

1.2 Project Overview

The design and development of the tactile system has proceeded with two different sensors in mind. Unfortunately, there are so many disparities between the two that we had difficulty keeping the system general enough to handle both. Let this serve as a demonstration of the variety of characteristics that must be considered for a given application.

The first sensor is about five inches long, with an octagonal cross section about 3/4 inches in diameter. Each of the eight rectangular faces is connected to a tapered piece, which is in turn connected to a common tip piece. There are a total of 133 sensitive sites -- 16 on each main face, one on each alternate taper, and one on the tip. Because of the the vague resemblance, we will refer to this sensor as the Finger.

The second sensor, the Pad, is a flat rubber square about two and one half inches on a side. An 8 x 8 grid of conical protrusions identify the 64 pressure sensitive sites. The pad is mounted on a square metal piece, about three and one half inches on a side, which is in turn connected to another similar piece by four metal posts. These posts have strain gauges on them which measure the force parallel to the object's surface.

Initially, we only considered the finger. Because of its shape and organization, the sensor is best suited to applications involving probing and tracing. This includes testing for resiliency, examining surface texture, and tracing cross-sections of an object. In our view, texture would be thought of as a kind of microscopic contour, while the cross-section tracings would yield a macroscopic contour. Taken together, we would be able to acquire an extremely detailed description of very selective parts of the object in question.

Unfortunately, this rather vague idea has not been developed. We have instead dealt with the two descriptions independently with the assumption that they can both be incorporated into a general object recognition system.

For his Master's Thesis, David Brown [BROWN-80] developed a three-dimensional positioning device for the finger. Basically, it is a square horizontal metal frame mounted on four legs. Moving forward and backward on this is a second, vertical square frame. A vertical track rides left to right on that, and a rod moves up and down in the track. The finger would be mounted with its tip downward at the bottom of the rod.

Thus, we have three degrees of freedom -- the X, Y and Z axes -- each positioned by a stepper motor driving a lead screw. This gives us the capability of examining, from the top, any object or objects placed on a table below the

horizontal frame, in a total working volume of about 18 cubic inches. Since the degrees of freedom are strictly positional, as opposed to rotational, we are not capable of reaching under an overhanging lip, or sideways below a covering section. This places certain restrictions on the kind of object we can examine. If we think of the horizontal axes as X and Y, then the object must be describable as a strict function of those two variables. Needless to say, this is not a robot arm, but we felt it would suffice, temporarily at least, for our research.

The positioning device and tactile sensor are directly controlled by a pair of Z80 microprocessors, which are in turn under the command of a PDP-11/60 minicomputer. Of the Z80's, one (the Motor Control Processor, MCP) is responsible for driving and positioning the stepper motors, and the other (the Tactile Sensing Processor, TSP) is dedicated to tactile data acquisition and compression. The MCP and TSP communicate with each other via a 14-bit wide parallel data path. The PDP-11/60 issues high level commands, and receives positional information, through a serial connection to the MCP. Finally, tactile data is passed to the 11/60 through a DMA link from the TSP.

One of the aforementioned high level commands would request the microprocessors to trace the cross-section of an object in any arbitrary plane in space, passing the sequence of 3-D coordinates back to the host computer. A great deal

of thought went into the implementation of this command, and it is, to some extent, responsible for the architecture described above. The procedure will be described in detail in a later section. It is a good example of how tactile sensory feedback can be used in a real time, closed loop fashion.

The finger was designed and fabricated at L.A.A.S., the major robotics establishment of the French government. Because of a severe lack of communication, many of the finger's details were not known to us when the software was being designed. This had a positive affect in that we were forced to be as general as possible. However, due to a number of unexpected delays, we still do not have the finger in our possession.

We arranged to borrow the pad sensor from Lord Corporation in Erie, Penna.* They traditionally deal with blending rubbers and bonding rubber to metal. This sensor, still in the prototype stage, is an attempt to expand their business.

At any rate, we had the pad sensor in our possession for three very long days. In preparation for that ordeal, we planned a number of different experiments. The Lord people were very helpful in this, and they provided us with the appropriate wooden test objects.

---

* Lord has since moved to Cary, South Carolina.

The characteristics of the pad sensor are very different than those of the finger. In particular, there is only one sensitive face. This makes the pad much less suited to contour tracing. We therefore decided to concentrate on some of the other aspects of tactile sensing -- dynamic texture analysis, static pattern recognition, and measurement of small angles between the object and sensor surfaces.

The ensuing sections will describe in detail the work performed.

Chapter 2: The Proposed Microprocessor Software

In anticipation of the arrival of the finger, a great deal of software was planned. Then, when the delays became apparent, work on those aspects not directly applicable to the pad sensor screeched to a halt. As a result, some of the design described here has not yet been implemented. In a later section we will discuss in detail exactly what the existing software does.

One of the important features of the Experimental Sensor Processor is its delegation of low level tasks to other processors. This helps to diminish the computational load on the host pdp-11/60. The tactile branch, in keeping with this principle, would have a set of commands which could be invoked by the host to perform various I/O and timing intensive operations, or functions involving real time feedback. Following are some of the commands that were considered:

1. Reset the machine.
2. Move to absolute coordinates (x, y, z), stop on collision with an object. This can be used as a "find something in this direction" command.

3. Scan Cross-section -- Trace the contour of an object in an arbitrary plane in 3-space. Returns to the host a list of step vectors describing the finger's path.

4. Local Texture -- Trace around a small circle on the surface of an object and produce a description of the texture. This could be in terms of degree of roughness, degree of compliance, or something as crude as a list of pressure values for each point in the path.

5. Search (in an as yet unspecified manner) for either a concave or a convex edge. It is assumed that the finger is already in contact with a surface.

6. Follow the contour of a concave or convex edge. Passes a list of step vectors to the host describing the finger's path.

The first command, Reset, is trivial. It simply involves the reinitialization of variables. The move command, due to its fundamental nature, has been implemented for use with the pad sensor. The cross-sectional scan command has received a great deal of attention, but has not been completely implemented because of its incompatibility with a single-face sensor. The final three commands, Local Texture, Find Edge, and Follow Edge, have to date received very little serious consideration. They are quite tentative, and may never be implemented.

2.1 Processors

As described in other sections of this thesis, the tactile branch consists of two microprocessors, the Tactile Sensing Processor (TSP), and the Motor Control Processor (MCP). A different program runs in the firmware of each

processor. Both are entirely interrupt driven using the Z-80 vectored interrupt system. From the host computer's point of view, the TSP provides data for texture analysis, and the MCP provides data for contour analysis.

2.1.1 Tactile Sensing Processor

The TSP program consists of a single loop in which each of the sensors is interrogated for its 8-bit pressure value. Each value is thrown into one of three categories with respect to a low and a high threshold. the category indicates whether the sensor is not touching anything, is in contact with an object, or is pressing the object too hard.* The sensors are then grouped by finger face, and a face status is computed for each face using the following rules:

If any sensor is over range, the face is over range;
If all sensors are below range, the face is below range;
Otherwise, the face is within range.

If there were any face status changes since the last pass, the Motor Control Processor is informed.

It is worth noticing that this condensation algorithm is independent of the particular organization of the finger. The number of faces, the faces' orientations, and even the

---

* We hope that the sensors have enough compliance of their own so we can arrange the thresholds successfully. We would like to guarantee that for any movement toward an object, there is at least one position in which the leading sensor is "in contact" before it exceeds the upper threshold.

mapping of sensor number to face number are stored in tabular form, and may be altered according to the parameters of a different sensor. It will be obvious later that the more faces we have, the easier it is to keep in contact with an object. In the ideal case, we would like a hemispherical finger with many sensors, each on its own face. Such an organization can be accommodated just as well as the current finger.

In addition to providing this condensed status information for the sister processor, the TSP must send some data to the host, for the texture analysis. How much data does the host need? If we send it all we can -- 133 8-bit bytes per step, 125 steps per second -- we would need the equivalent of 20 9600 baud serial communication lines to handle the load! The bottleneck is removed by using a Direct Memory Access (DMA) interface. But even so, we cannot expect the PDP-11/60 to analyze data arriving at such an incredible rate, and still be able to keep up with the other sensory branches, and perform the higher level recognition tasks at the same time. It simply does not have the computational power.

The answer, of course, is to filter or condense the data before sending it. We have several possibilities in mind. First, a sensor is only considered valid if its pressure value is "within range". This filter is always in effect. Other possibilities include averaging sensor

readings over time and only reporting after a fixed number of steps, or combining somehow the readings from all sensors on each face which is "within range" to produce a single face pressure value. A final possibility is to arrive at some kind of measure of roughness for the surface under consideration, and only pass that number back to the host computer. This decision has not been made.

2.1.2 Motor Control Processor

The Motor Control Processor's basic job is to control and coordinate the three stepping motors which position the finger. When it is necessary that the host computer know the path that the finger follows during the execution of a command, the MCP provides it.

Steps are taken in a synchronous fashion. That is, if the step rate is set to 125 steps per second (the default case), the processor is interrupted every eight milliseconds to determine which motors are to be stepped, and in which direction.

So, after each interval, the MCP may pulse any combination of the three motors, and each can be in one of two directions. This leads to 26 possible directions in which a single step can move (ignoring the case where no step is taken at all). We represent this direction as a

6-bit "step vector", organized as follows:

bit 5	4	3	2	1	0
Z direction	Z step	Y direction	Y step	X direction	X step

Since this fits easily in an 8-bit byte, it is very convenient now for the MCP to give a path to the host computer. It simply sends a one-byte step vector over the serial line for each step taken. The host collects the sequence of step vectors in a buffer, and the exact path can be reconstructed very quickly at any time.

There are, of course, situations in which it is necessary to give an absolute coordinate. For example, when the absolute move command is aborted due to collision with an object, it is necessary to inform the host what the new position is. A mechanism is provided for this, too.

Notice that the MCP returns (effectively) a sequence of points. It does not try to fit them to curves, surface patches, generalized cylinders, etc. This is left to the host computer. It is unreasonable to expect an 8-bit microprocessor which lacks even a multiply instruction to do these in real time.

When moving from one position to another in 3-space, it is desirable to do so in a straight line. This requires varying the speeds of the individual motors so that they all arrive at their destinations simultaneously. The following

example shows how we would like to arrange the steps in a sample situation.

	A steps	B desired time between steps
X	17	9.88 milliseconds
Y	21	8.00 milliseconds
Z	5	33.6 milliseconds

The values in column B were arrived at by dividing the column A values into the greatest column A value, and multiplying the result by 8 millisecs. (8 millisecs is the speed at which we would like the fastest motor to operate).

This is a lot of work for an 8-bit microprocessor to perform. Also, if the precision of these calculations is not great enough, it becomes virtually impossible to predict exactly where the finger will be at any given point in time.

Fortunately, the synchronous stepping scheme makes matters much simpler. The overall line of motion is a line in 3-space. This is described and stored in terms of three direction components. There are also two accumulating counters, one for the mid direction, and one for the min direction. (The mid direction is the dimension which has the second-largest number of steps to take. Min direction is defined similarly.) Both are preset to zero.

After each 8-millisecond interval, a step vector is created, and the motors are stepped accordingly. The max direction is always stepped. For each of the other two

directions, the accumulating counter is incremented by the corresponding direction component value, and the result is taken modulo the max direction component. If an overflow occured, a step is taken.

Applying the algorithm to the above example results in the following sequence of steps.

Step	X	Y	Z	Step	X	Y	Z
1		*		11		*	
2	*	*		12	*	*	
3	*	*		13	*	*	*
4	*	*		14	*	*	
5	*	*	*	15	*	*	
6		*		16		*	
7	*	*		17	*	*	*
8	*	*		18	*	*	
9	*	*	*	19	*	*	
10	*	*		20	*	*	
				21	*	*	*

When a step is taken, two corollary actions occur. First, if the MCP is providing path information, the step vector is sent to the host. Second, a termination test is made. For the absolute move command, termination occurs when the finger reaches its destination.

This command also terminates if the Tactile Sensing Processor indicates that the finger has come in contact with an object. Primarily, this is to protect the finger from damage. However, it also makes it possible for the host to say, "look in this direction for an object." In that sense, this command can be used as an object finder.

2.2 Cross-Sectional Scan Command

This command is invoked by the host to trace the contour of an object's cross-section in any arbitrary plane in 3-space.* The arguments include the coefficients a, b and c in the equation of the plane $ax + by + cz = 0$, and a pair of special 3-D points which define the search volume. The finger must already be touching an object, and the plane is assumed to pass through the finger's current position.

Consider a conical object and a slicing plane parallel to the x-y plane. The MCP will drive the finger in the plane such that it remains in contact with the surface of the cone. All the while, it passes its path back to the host. Later, the host will analyze the path, and discover that it describes a circle.

The search volume is included to limit the finger's range of motion. Suppose, for example, the host wanted to construct a 3-D bicubic surface patch. It could do this by requesting four cross-sectional scans using vertical planes whose y-z projection is a rectangle. Then it could fit curves to each of the four point sequences, and perhaps fit a patch to these four curves.

---

* My terms will be very confusing unless I define them at the outset. "Plane" generally refers to the arbitrary cross-sectional plane given by the host. "Surface" is the (possibly curved) surface of the object. "Face" refers to one of the faces of the finger on which sensors are mounted. "Search volume" means the physical volume in which the finger is allowed to move.

Unless we provide some mechanism for limiting the search space, there is no way to prevent the finger from doing a complete scan of the object's cross-section, when only a small portion of that scan is needed.

The search volume is a rectangular parallelepiped with diagonally opposed corners defined by two arbitrary points in 3-space. The arbitrary points are chosen by the host computer and passed to the MCP as arguments to this command. Very often, the points may contain special coordinate values of 0 or 'max'. These may be used to effectively leave one or more dimensions completely unconstrained.

In the surface patch example, we would like to constrain the x and y position to the projection of the four slicing planes onto the x-y plane. The z position should not be constrained at all. Thus, the two arbitrary points might be (X1, Y1,0) and (X2, Y2, max).*

The scan will terminate when the finger either exceeds one of the bounds, or returns to its initial position. This second termination condition is useful if the host is interested in producing a contour map of the object. It could do this by requesting a series of scans, using cross-section planes parallel to the x-y plane, but at varying z values. In this case we would like the finger to

---

* In addition to this constraint, there is an implicit maximum search volume given by the dimensions of the device.

completely circumscribe the object, continuing until it returns to its starting point.

A problem which has not yet been mentioned is that of keeping in contact with the surface of an object. It turns out that in most situations, this is relatively simple. The method requires three kinds of information.

As described earlier, the finger has a number of distinct faces. The present structure of the positioning device does not allow for rotation or re-orientation of any kind. Hence, except for possible translation, these faces are fixed. Their equations, as well as those of the planes perpendicular to them, are predefined as constants in the MCP program.

Second, we have the equation of the cross-sectioning plane. All motion of the finger is to be restricted to that plane. By intersecting this plane with either the plane of a face of the plane perpendicular to a face, we can calculate a line of motion. This can then be fed to the absolute move routine to effect the movement.

Finally, there is the data from the Tactile Sensing Processor. This indicates whether each face is below range, within range, or above range. Typically, there will be only one face which is within range. This is labelled the "active face," because it is the one which is in contact with the surface. There are exceptions, and we will see

shortly how we can account for them.

The objective in keeping in contact with a surface is to keep the active face within range. Recalling that by definition of the command, the active face is initially within range, we have the following cases:

(1) Active face is within range;
(2) Active face is below range;
(3) Active face is above range; and
(4) A second face comes within or above range.

In case (1), the finger is in contact with the surface. Our best estimate of the shape of the object at this point is a plane parallel to the active face. Calculate the line of motion (if it has not been calculated already) as the intersection between the active face and the cross-sectioning plane. Send the current position to the host, and take a step.

In cases (2) and (3), the finger either has lost contact, or is pressing the surface too hard. Calculate a line of motion as the intersection between the cross-sectioning plane and the plane perpendicular to the active face. Then take a step along it away from or toward the finger's center, respectively. Do not send this step vector to the host, because it is not part of the surface contour.

Case (4) could result from several different situations. Take the scenario in which the finger hit a

concave corner. In this case, the appropriate action is to make the new face the active face, and then act according to its status.

Another scenario in which case (4) could occur involves reaching either a convex corner, or a point at which the surface curves away from the currently active face. Again, the appropriate action is to declare the new face as the active face, and act according to its status.

There are a number of other situations in which a second face could come within or above range. The appropriate action is not always the same as above. In fact, one could imagine situations in which a third and perhaps a fourth face must be considered. Though these cases have not yet been adequately resolved, we do not expect them to be overly troublesome.

Chapter 3: The Implemented Software

We noted earlier that although the software was designed for the finger, it was eventually implemented for the pad sensor. The most notable difference between design and implementation was the fact that in the end, we only used one microprocessor. All those commands which required multiple face sensing -- trace contour, follow edge, etc. -- were eliminated because the pad sensor in fact has only one face. It happened that these commands coincided with the ones which required real time feedback. Therefore, the requirements of the tactile data acquisition software became almost trivial, and could be handled easily and much more simply by the Motor Control Processor.

3.1 Environmental Details

The microprocessor software is written in Z80 assembly language. It resides on the PDP-11/60, which runs under the RSX-11M operating system. We use a primitive Z80 assembler, written in C, which produces Intel hex-format object code. This we download to the microprocessor via the 1200 baud serial line which connects the two systems. As it turned

out, 1200 baud was as fast as the 11/60 could reliably receive and store data.

The microprocessor system is made up of a California Computer Systems S-100 bus and mainframe, 8K of RAM, and a Cromemco Single Card Computer (SCC) with 1K RAM and room for 8K of PROM, 1K of which is taken up by a modified form of Cromemco's power-on monitor. The SCC has five timers, three parallel ports (input/output), and a serial port. Since the A/D converter built into the pad sensor produced CMOS output levels, we decided to temporarily add our own converter, a Cromemco D+7A board.

In the following sections we give a complete description of the software as it currently stands.

3.2 Command Format and Interpretation

The command language was to be a permanent part of the software. It would be used initially by a human user to control the pad sensor's movement and data acquisition. Eventually, however, it would become the Experimental Sensory Processor's way of driving its tactile branch.

Thus we had three goals in mind. First, the command language should be versatile. It should be able to handle the commands described in the previous chapter as well as the simple placement and data acquisition commands we needed for the pad sensor experiments. Second, it should be

concise enough, and easy enough to interpret, to be used for interprocessor communication. Finally, it had to be legible, so that the user could issue commands from his keyboard.

We settled on a syntax with mnemonic, single character commands, optionally preceded by an ascii-coded positive or negative integer which defaults to +1 if omitted, and optionally followed by any special arguments required by the command. The preceding integer is decoded by the parser. It generally refers to the multiplicity, though its interpretation is up to the individual command routines. The trailing arguments are parsed and interpreted completely by the individual command routines.

Commands may be strung together to form a command sequence. Execution will not begin until a carriage return is received. The sequence is, of course, stored in a buffer until execution is complete. A key advantage to this is that it makes loops possible. In the syntax, a subsequence may be grouped by parentheses, which in turn may optionally be preceded by a multiplicity M. The entire subsequence will be repeated M times. Subsequences may be nested to any reasonable depth.

There is one more rather important feature. While the command sequence is incomplete, the Motor Control Processor completely disables interrupts. Since the motors are driven by periodic timer interrupts, all movement must stop.

Similarly, characters coming from the serial line during command execution are ignored. This generally does not matter, because execution will have terminated before a new command sequence arrives. However, should it become necessary for the host computer (or user) to abort execution, it (he) may send an ESCape character. This causes a non-local subroutine return to the command sequence input routine, which immediately disables interrupts.

The following is a list of the commands currently available.

H	Home -- return to inner, upper left corner, and reset the current position to (0,0,0).
nX	Move n steps in the X direction (n may be positive or negative, and defaults to +1 if omitted).
nY	Move n steps in the Y direction.
nZ	Move n steps in the Z direction.
@x,y,z	Move to absolute position (x,y,z).
n(	Begin nest.
)	End nest.
=	Return current position as x,y,z coordinates, ascii-coded decimal values separated by commas.
Q	Quit the program -- return to power-on monitor.
1S	Take a snapshot of the sensor, store data in memory, increment frame count.
-1S	Take as many snapshots as possible until the completion of the current motor step.
0S	Clear the frame memory.
G	Send the contents of the frame memory to the host, beginning with the frame count. All data is in ascii-coded hexadecimal. Then clear the frame memory.
space	Null operation.

These commands are obviously very simple. However, they can be very powerful when grouped together. For example, the sequence

```
@100,100,100 50( 3( 20X 20Z S -20Z) 20Y -60X) G
```

takes 150 snapshots, in a 50 by 3 grid, beginning at (100,100,100), then sends all the collected data to the host computer. Since optical limit switches prevent the motors from moving past the ends of travel, one could find the maximum limits in all directions by issuing

@10000,10000,10000 =

(the actual range is roughly 1200 steps per axis). This would move the sensor to the corner opposite the home position and report the actual coordinates.

This list will eventually be enhanced to include the commands described in the previous chapter. We expect to be able to continue to denote each command with one mnemonic character.

3.3 Motor Control

It is not surprising that the most complicated task performed by the Motor Control Processor is, in fact, motor control. The complexity arises for two reasons. First, it is intended to be a permanent part of the MCP software, and is therefore very general in design. Second and most important, the step service routines effectively and completely insulate the higher level command execution processes from the hardware.

At the top level, an individual command routine uses

the step services in the following fashion:

```
Set the direction components in LINE
Call SCFILL to fill the step control table
Do until termination-condition:
    Call STEP to initiate a step when ready
    Call NEWPOS to update current position
    Call NEXTPO to prepare the next step
End
```

Note that it does not concern itself with timing in any way, nor does it have to take into account the physical limits of the device. The STEP routine guarantees a minimum pulse width (maximum step rate), and even modifies the step request if such an action would drive a motor past its end of travel.

Also note that the routine must actively request that a step be taken. If, for some reason, the evaluation of the termination condition is very time consuming, the motors will simply run slower. This has another advantage. Should the program be damaged by an unusually high incidence of cosmic rays, the motors will not go out of control. They will simply stop, because nothing is calling the STEP routine.

Before we take a closer look at these routines, we must discuss the data structures involved. The first one that was mentioned is LINE. It takes three numbers to define the direction of a line in 3-space: delta-x, delta-y, and delta-z. These are the line's direction components. Simply put, when we take delta-x steps in the x direction, we must

also take delta-y steps in the y direction, and delta-z steps in the z direction. Within the MCP, these values are stored and manipulated as unrestricted 16-bit integers. However, should it later become necessary to compare line directions, these may have to be restricted to relatively prime integers. LINE is a three word array which defines the desired path to the step routines.*

A commonly accepted canonical form for these values is a list of direction cosines. This requires that the values be real numbers, and that the sum of their squares equal unity. Fortunately, we have not found this form necessary.

The second data structure is the Step Control Table (SCTAB). This 15-byte table is basic to the operation of the step service routines. Following is a layout of its contents.

```
SCTAB+ 0: (byte)  Next port image
       1: (byte)  Port image skeleton (direction bits)
       2: (word)  Max direction component
       4: (word)  Mid direction component
       6: (word)  Min direction component
       8: (word)  Mid accumulating counter
      10: (word)  Min accumulating counter
      12: (byte)  Max direction's motor pulse and power bits
      13: (byte)  Mid direction's motor pulse and power bits
      14: (byte)  Min direction's motor pulse and power bits
```

Let us digress a moment before we explain SCTAB. Instructions are passed to the stepper motors via an 8-bit

---

* The Z80, of course, does not really have any distinct concept of a "word." However, being an old PDP-11 man, I always have and always will refer to a 2-byte quantity as a word.

output port, which looks like this:

bit 7	6	5	4	3	2	1	0
Z dir	Z step	Y-Z power	Y dir	Y step	X power	X dir	X step

The three direction bits indicate which direction the corresponding motor is to move. One implies the negative direction, zero implies the positive. The step bits, when pulsed, cause their corresponding motors to take a step in the indicated direction. Due to a low-pass filter which is applied to these bits for noise immunization purposes, there is a minimum pulse width. The MCP uses a separate timer for this, as will be described later.

Finally, the power bits, when on, cause drive power to be applied to the corresponding motors. For now, the reader need only understand that a motor must have power in order to operate.

Now we should be able to make sense out of the Step Control Table. The first item, the "next port image" is exactly that -- the 8-bit quantity that is to be sent by the STEP subroutine to the motor drive output port at the next opportunity. It is very important to note that this value is, in general, calculated concurrently with the previous step, by a call to NEXTPO.

The second item, the "port image skeleton," contains the three direction bits. These bits are applied with every

step. The SCFILL routine sets them according to the signs of the three direction components in LINE, and they do not change again until a new line is chosen.

The next three items, the Max, Mid and Min direction components, are actually the magnitudes of the numbers that appeared in the LINE array, but in sorted order. These are used in conjunction with the Mid and Min accumulating counters to determine which motors to step at the next timing interval.

Finally, the mapping from the sorted order to the x-y-z order is given by the last three items. Each of these bytes has exactly two bits set, corresponding to the appropriate motor's step and power bits.

The NEXTPO routine first decides which motors are to be stepped, and then adds together the corresponding mapping bytes, along with the direction bits from the skeleton. The resulting value is the next motor port image.

Let us now return to the high level control loop given at the beginning of this section. First of all, note that the values passed in the LINE array indicate a direction only. They do not completely describe a line segment in 3-space. It is assumed that the line of motion will begin at the current position, and the control loop is responsible for knowing when to stop.

Once the LINE table is set, SCFILL is called to fill

the Step Control Table. All values are calculated independent of the previous contents. The NEXTPO routine is then called automatically to use the new table to compute the first port image and place it in the zeroth location.

Since a step is never taken unless specifically requested by the control loop, it is perfectly reasonable to completely change direction at any time by simply changing LINE and calling SCFILL, before calling STEP again. One need not be concerned with the timing considerations.

Within the control loop itself, the first action is a call to the STEP routine. This routine waits, if necessary, for the previous step to complete. Then it calls CHECK to check the optical end-of-travel limit switches and, if necessary, modify the candidate port image. Finally, the routine outputs the image to the motor port and returns to the calling control loop.

Internally, one of the five on-board timers is also set to cause an interrupt after a time equal to half the minimum step pulse width has elapsed. The routine which handles that interrupt will clear the motor step bits and set the timer to interrupt again after another equal interval. At that point, an entire step has completed. The STEP routine, if it is waiting, is allowed to proceed with another step. In this way, something like an open ended square wave is generated on the motor pulse bits.

This brings us to the other subroutine calls in the main control loop. During the timing delays, the CPU is free to do quite a substantial amount of processing. Recall that the STEP routine has the power to modify the candidate port image. This modified image is returned to the control loop, where it is passed again to the NEWPOS routine. NEWPOS, based on the direction and step bits which were actually sent, updates the current coordinate counters.

The calculation of the next port image is then accomplished by a call to NEXTPO, which proceeds as follows.

1. Begin with the motor port skeleton, which defines the direction bits.
2. Add in the Max direction's pulse and power bits. That motor is to move at the maximum rate, and will therefore always take a step.
3. Add the Mid direction component to the Mid accumulating counter, and take the result modulo the Max direction component. If there was an overflow, we want to step the Mid motor. Add in its pulse and power bits.
4. Repeat step 3 for the Min direction.

The resulting value is placed in the first byte of the Step Control Table. An example of this algorithm in operation was given in chapter 2.

There is one final item to discuss. Conceptually, a stepper motor has a series of magnetic coils arranged in a circle around an iron core. As steps are taken, each coil in succession is energized, drawing the core around the circle. During normal operation, a given coil is only

energized for a brief period before its successor takes over. However, when the motor is standing still, one coil is energized continuously for a long period of time. It can generate quite a bit of heat -- enough, perhaps, to burn itself out.*

To solve this problem we implemented the following scheme. Every time a motor is stepped, its power is automatically turned on. At the same time, its corresponding usage counter is reset to some constant. Periodically, another of the on-board timers interrupts the processor to decrement all the usage counters. When any one reaches zero, the corresponding power bit is turned off.

The effect of this is to power down any motor that has not been stepped in the last two seconds. The action is so completely transparent to the higher level control software that we refer to it as the "burnout protection demon."

3.4 Tactile Data Acquisition

Due to its temporary status, the tactile data acquisition is perhaps the least important part of the software. As soon as the finger arrives, these routines will be removed from the Motor Control Processor and rewritten completely for the Tactile Sensing Processor,

---

* I don't know whether motors would actually burn out, but when I found I could fry eggs on them, I did not want to take chances.

according to the plans given in chapter 2. Therefore, as might be expected, the current code is far from general. It is entirely driven by the S and G commands described earlier. Nothing happens asynchronously.

The entire unused portion of the MCP's memory board is used as a buffer for tactile data. Upon MCP initialization, the frame count is reset to zero. Then, each time a snapshot is requested, the data record is placed in the next position in the buffer, and the frame count is incremented. When the readout is requested (via the G command), the program simply types it all out, one line per record, beginning with a line consisting solely of the frame count. The information is transmitted in ascii coded hexadecimal, as an optimization of both transmission time and coding time.

Chapter 4: Experiments and Results

In this chapter we will discuss the experiments which were actually performed using the pad sensor. We will consider the methods, the goals, the problems, and the results. When possible and appropriate, we will refer to figures which illustrate the results.

4.1 Calibration

The pad sensor consists of an 8 x 8 array of sensitive sites whose analog output values are fed into an analog multiplexer, and finally into an analog to digital converter. All this circuitry is part of the sensing device. Unfortunately, since the A/D converter emits CMOS voltage levels, and our parallel ports use TTL inputs, we had to bypass the internal A/D and use our own. This resolved the incompatibility, but gave vent to another problem. The pressure signals coming out of the multiplexer ranged roughly from +2.0 to +2.5 volts, and our A/D converter expected a range of -2.5 to +2.5. As a result, the digital pressure readings never went below about 235, out of a maximum 255.

In other words, the fact that we can exhibit only a

little over four bits of precision is not a reflection on the device, but on the interface. With the right interface, we would estimate upwards of six bits of valid data.

Each of the 64 pressure sensitive sites puts out a slightly different range of voltage levels. They therefore required individual calibration. The most straightforward way of doing this is to press the sensor down hard on a flat surface, take a snapshot, release the sensor entirely, and take another snapshot. This yields a matrix of minimum and maximum pressure values, to which all subsequent data would be scaled in a linear transformation.

Of course, nothing is ever so simple. Each pressure sensitive site requires roughly 1.3 pounds of pressure to completely depress it. Multiplying that by 64 sites, we find that we need over 80 pounds of pressure to acquire the maximum readings. Our Z-axis motor is not capable of this.

The solution was to depress each site individually, and then combine the data into a single matrix of maximum pressure values. Fortunately, the Motor Control Processor's command language was flexible and powerful enough to do this painlessly in one command sequence, with two loops for X and Y positioning.

Once the minima and maxima were obtained, it was a simple matter to map all input data into a uniform range of 0 - 255. It is worth mentioning here that throughout the

entire testing period, these ranges never changed more than one unit. In addition, we never had any problem with spurious data being generated where there was no contact. Those points always mapped to zero. We were quite impressed with the robustness of the pad sensor.

4.2 Static Tactile Image Analysis

4.2.1 Single Image

The obvious first step in analyzing tactile images is to lay the sensor down on a known object, take a snapshot, and see whether it is recognizable. This we did, and the results are depicted in fig. 1.

In fig. 1f we used a one inch square, set off-center, but oriented orthogonally with the sensor's grid axes. There is no question as to the identity of that object. A simple threshold operation would clearly distinguish it from the background.

Fig. 1e and fig. 1d show the same square rotated counterclockwise 30 degrees and 45 degrees, respectively. Fig. 1c shows an equilateral triangle, point downward, and fig. 1b depicts the same triangle rotated clockwise about 75 degrees. Notice how some pixels are much lighter than others in the images with non-orthogonal edges. This phenomenon arises when the object covers less than half the area of a site. Since the site is conical in shape, the

edge must be pressing on the wall of the cone. It cannot depress the cone as far as it could if it were pressing on the apex.

In theory, it should be possible in some cases to determine exactly how much of the cone is actually covered by the object. However, we must assume the following: 1) that the object surface, particularly the edge in question, is smooth, 2) that the object surface is in a plane parallel to that of the pad sensor, 3) that the individual sites on the sensor are in fact conical, with bases that meet the bases of their neighbors, and 4) that we know how to calculate the actual depression as a function of output pressure value.

Unfortunately, neither of the last two assumptions are valid in our case. The cones are actually cut off before they reach the apex,* and we do not have the data to perform the depression calculation.

Finally, fig. 1a shows a one inch diameter circle. Notice that it appears to be identical to the square in fig. 1c. This is a question of resolution. Clearly, if the spatial resolution were doubled or quadrupled, the distinction would be obvious.

---

* My office-mate tells me that the technical term for this shape is "frustum."

4.2.2 Spatial Resolution

How do variations in sensor resolution effect the image? The simplest way to tackle this question is to vary the size of the features on the test objects. We used a set of disks with raised concentric circles projecting from them in relief. The variations consisted of two amplitudes and three frequencies, totalling six disks.

Fig. 2 shows the images obtained. As might be expected, those disks in which the spacing between the circles approach the spacing between the sensitive sites (figs. 2a and 2d) are clear. As the frequency increases, the shape becomes less obvious, until it is completely unintelligible at the highest frequency.

The effect of amplitude is also fairly predictable. At low amplitude, the circles are wider, and therefore more sites are in contact with the surface. This can be seen most clearly (again) in figs. 2a and 2d. Also, the inner circle is more distinct in fig. 2e than in fig. 2b. This is because at the lower amplitude, the depth of a trough is considerably less than the height of a conical site, and therefore some trough sites come in contact with the surface.

Theoretically, it should be possible to compare pressure values and determine where the troughs and crests occur. However, here we run into the limitation in our 3-D

positioning device which we alluded to in the Calibration section. The Z-axis motor, which supplies the normal force, is a bit too weak for this pad sensor. Each sensitive site requires a certain amount of force to depress it, and the motor must be able to exert the sum of these forces in order to obtain a reliable reading. Therefore, as more sites contact the surface, each one receives less pressure. Furthermore, if the surface is not uniform, neither are the reductions in pressure.

4.2.3 Multiple Images

How can we improve the spatial resolution with the equipment available to us? One simple way to double the number of data points on each dimension is to take a reading at each of the four corners of a small square, whose sides are half the length of the distance between sites. This we did, using the same six disks, and the results are visible in fig. 3.

The images are slightly clearer, but not as much as we had hoped. Again, the disappointment is indirectly caused by the deficient Z-axis motor. When taking a snapshot, we try to depress the sensitive cones as much as possible, since we are not capable of depressing any of them completely. To do this, we simply instruct the Motor Control Processor to lower the Z-axis motor until it won't go any further.

This works quite well in general. However, consider the following hypothetical case. Suppose the test object is a single sine wave and the sensor is a single cone. First, we lower the cone onto the crest of the wave as far as it will go, and take a snapshot. Then we move the cone to the trough and repeat the operation. The two images look identical! In both cases, the cone was depressed as far as it would go, and it is in fact the cone depression which determines the image. This, we believe, is the root of the multiple image problem.*

The solution, of course, is to strengthen the Z-axis motor. Then, instead of simply lowering the sensor until it stops, we would lower it to a consistent Z-coordinate. The resulting set of images would be much clearer.

4.2.4 Large Objects

Can we examine objects which are much larger than the sensor? For this experiment we used a flat surface about 12 inches long and three inches wide -- slightly wider than the sensor pad itself. A set of eight grooves were cut into this surface in order to form a pattern of diverging lines (see fig. 4a). By taking a series of snapshots at successive lengthwise positions, we should be able to reconstruct the entire image, in spite of the fact that it

---

* Or, "Aye, there's the rub!"

is much longer than the sensor.

The Motor Control Processor's command language again made this a simple task. We took fifty images, stepping about five millimeters between each. The reconstruction, shown in fig. 4b, was accomplished by superimposing the images in the appropriate positions relative to each other. As before, when the distance between features approaches the distance between sensitive sites, the pattern becomes clearer.

Can we use our multiple image trick to improve the resolution? We repeated the same procedure, except that this time we took three snapshots, four millimeters apart widthwise, for each of the fifty steps lengthwise. The reconstruction, fig. 4c, shows the angled edges much more clearly at lower frequencies than does fig. 4b. At higher frequencies, however, both reconstructions are equally unintelligible. Once again, we blame the failure on the Z-axis motor, and our method of maximizing pressure.

4.2.5 Small Angle Measurement

When a robot hand grasps an object, does it have a good grip? Very often, a "good grip" is one in which the flat surfaces of the object are wholly in contact with the flat faces of the fingers. The question can then be answered very simply by measuring, for each finger, the angle between these two planes.

This experiment proved to be extremely successful. Using the one inch square as our test object, we took four snapshots. In the first image we layed the pad sensor flat on the square, as usual, giving us a zero degree standard. For the three subsequent images, we lowered the left end of the table by 1.0, 1.25, and 1.5 inches respectively, producing angles of 3.3, 4.1, and 4.9 degrees.

The results are shown in table 1. For each image we arrived at a single number describing the slant. The number was calculated simply by averaging all the pressure differences between horizontally adjacent sites. In theory the ratio of the third slant value to the second should be 1.25,* and the fourth to the the second should be 1.5. This was not the case.

However, the first image, whose slant should have been zero, did exhibit a small slant value. If we take this as an error, we can produce a correction factor by dividing it by the slant value for the second frame. When that percentage is subtracted from each of the two ratios arrived at earlier, we get remarkable results. The corrected ratios differ from the expected values by less than two percent!

---

* Proof is obvious from the geometry, as long as we assume a linear relationship between depression distance and output value.

4.3 Dynamic Texture Analysis

We believe that until tactile sensors can be fabricated with extremely fine resolution, information about the texture of a surface would best be obtained by moving the sensor along the surface, and examining the changes in pressure readings, as opposed to the pressure readings themselves.

Toward this end, we tried several times to make the positioning device drag the pad sensor along different surfaces, but failed each time. The sensitive cones, because they were designed to grasp an object without allowing it to slip, were made out of high friction rubber. This, of course, directly hindered the experiment. The stepper motors were not powerful enough to pull the sensor and still maintain enough contact pressure to yield a significant reading.

In the end we performed a singularly unscientific experiment. We dismounted the pad sensor from the positioning device and dragged it by hand along a flat wooden surface, taking 100 snapshots over a period of about five seconds. This may not have been so bad, except that we neglected to measure the exact distance traversed, or anything that could directly or indirectly give us the velocity.

The analysis is interesting, though quite inconclusive.

The sensor is made up of an 8 by 8 grid of sensitive cones. Let us define a column as the series of cones lined up in the X-direction, and a row as the cones lined up in the Y-direction. Given that the sensor was dragged in the positive X-direction, we contend that there should be some aspect of the data which is consistent down a column, but different across a row. Furthermore, there should be a small but constant time delay between the features exhibited by one site and those exhibited by the next site down the column.

The motivation for this hypothesis is as follows. Picture a textured surface as a terrain of bumps and ridges. As the sensor grid passes over this terrain, the cones across a row will collect entirely unrelated data. However, those down a column will encounter the exact same bumps and ridges that were encountered by their predecessors, but a little bit later. Thus we have eight instances of eight-fold redundant data. We should be able to find some consistency somewhere.

Initially, we plotted the raw pressure data from each of the 64 cones as a function of time. Fig. 5 is a reproduction of this, with each plot placed in the same grid position as the corresponding cone. We expect to be able to look down a column and see some consistency that does not occur across a row. Unfortunately, no such consistencies were immediately obvious.

The next step was to try to home in on the changes in pressure, as opposed to the pressures themselves. However, a simple pairwise difference derivative (see fig. 6) was no more enlightening than the raw data.

Well, what about the Fourier transform? Surely the frequency domain is closer to our goal than the time domain. Unfortunately, applying this transform meant giving up our time delay information, which we needed for comparing curves.

What we really needed was some smooth measure of frequency as a function of time. A colleague* suggested the following procedure. First, take the pairwise difference derivative. Then, pass a window along the time axis. For each point in time, count the number of zero crossings in the window, and divide by the width of the window. A window n units wide would have a maximum of n zero crossings, and thus the ratio would be unity. No crossings would produce a ratio of zero. Note that the operator is valid, and produces the same range of values, independent of the window size. The only difference is in the precision.

We used a window with an odd number of points, so it could be symmetric about the point under consideration. If the distance to one margin or the other was smaller than half the window size, the window was shrunk accordingly, so

---

* Thank you, Gerry Radack.

that symmetry was maintained. We tried various window sizes in order to obtain the smoothest curve possible without losing too many features. The optimal size was about 25 units (out of 100), shown in fig. 7a. A 15 unit window is shown in fig. 7b for comparison.

There are (finally) some definitely visible similarities among the resultant curves of fig. 7a. Examine, for example, the troughs in rows 6, 7 and 8 of column 1. Notice how similar they are, and how a small, constant time delay occurs between each curve and its successor. The same phenomenon is visible in rows 1, 3, 5 and 6 of the third column, and in rows 1 and 3 of column 7.

As one looks up and down a column, there seems to be some kind of topological similarity. This is exactly what we want to find. However, identifying it mathematically is no simple task. The obvious operator to apply would be the cross correlation. This compares two graphs and produces a number describing the closeness of the match, then shifts one graph relative to the other and repeats the calculation. One correlation value is generated for each possible shift. The resulting curve shows not only how well the two graphs match, but at what time delay value the match is optimal.

Unfortunately, the results were very disappointing. No matter which pair of graphs we compared, the cross correlation never went substantially higher than zero, and the best match always occured at zero shift. Needless to

say, at least one more level of processing is called for.

4.4 Conclusions

First, it is clear that an 8 by 8 grid of pressure sensitive sites is generally not enough for pattern recognition of single static images. In most real applications, either the objects will be larger than the pad, or the features will be below the pad's resolution.

With reasonably good positioning equipment, the resolution can be significantly improved, and the size of the area under consideration considerably increased, by taking multiple images. However, this is often too time consuming, and therefore infeasible.

The straightforward solution is to increase the spatial resolution, the number of sites, or both. We have shown that when feature dimensions are comparable to resolution, shape recognition can be quite simple. This has also been demonstrated by Hillis [HILLIS-81], using a sensor recently developed at the MIT A.I. Laboratory, and of course by Briot [BRIOT-79], who used an array of binary sensors. One typical application for this might be the table sensor which was described in the introduction.

A more novel approach might be to build multijointed fingers for the robot gripper, such as the three fingered hand developed by Ken Salisbury [SALISBURY-81] at the

Stanford A.I. Laboratory. This would enable the robot to manipulate the object while transporting it, in such a way that it becomes not only feasible, but a matter of course to take multiple tactile images.

In the experiment concerning measurement of small angles, we obtained impressive results. The computed values were even more accurate than we had hoped. From this we conclude that a tactile sensor with properties similar to those of the pad sensor is eminently suited to applications involving small angle measurement, such as grip improvement.

As far as texture analysis is concerned, we believe our approach is a good one. Visually, it is apparent that we are on the right track. However, the experiment must be repeated in a much more controlled fashion, and different surfaces must be examined and compared. Then, we hope we will eventually be able to manipulate the data in such a way that we can use it to identify the surface.

Chapter 5: Further Work

As was mentioned earlier, the pad sensor was in our possession for only a short time, by no means long enough for exhaustive experimentation. In fact, many of the more interesting ideas occured to us after the sensor was returned, when we began to analyze the data.

It should be possible to calculate the coefficient of friction between various surfaces and the rubber face of the sensor. First, one must know the force as a function of digital output for each sensitive site, as well as for the strain gauges on the metal posts. Then, one would drag the sensor along the surface in question, and take force measurements. The normal force N is simply the sum of the forces on all the sites, and the frictional force F is derived from the horizontal forces given by the strain gauges. By plugging these numbers into the equation $F = uN$ one can calculate u, the coefficient of friction. This might be usable as a distinguishing characteristic between surfaces.

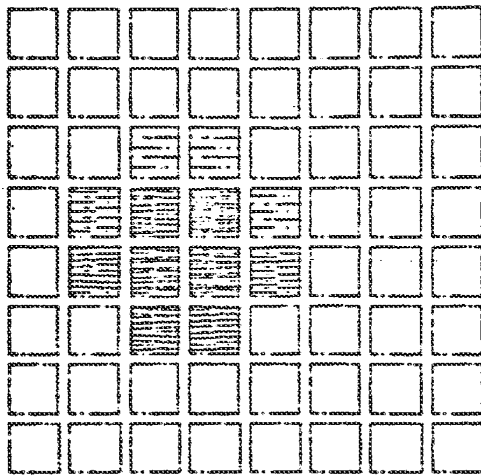
It might also be useful to measure granularity. This could be done simply by placing the sensor onto the surface

and counting the number of sensitive sites which exhibit significant pressure. Of course, the grains in the test surfaces must be comparable in size to the resolution of the sensor.

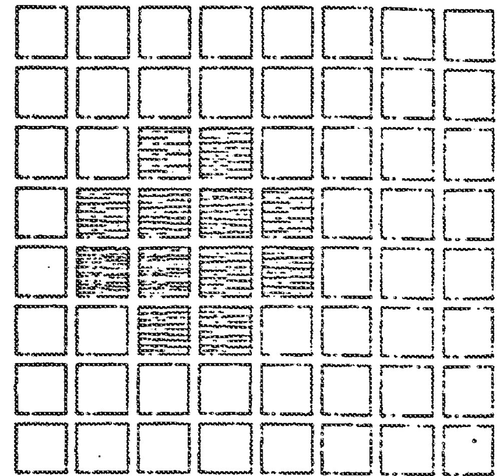
Certainly the dynamic texture analysis tests should be repeated and extended. Once that data has been hashed out, it should be possible to identify surfaces based on pressure response to friction.

Finally, there are two aspects of tactile sensing which we have not experimented with because they are better suited to the finger than the pad sensor. First, the finger should be capable of poking a surface and comparing predicted pressure with actual pressure in order to measure of surface resilience. Second, there is the whole question of tracing cross sections and producing, essentially, a 3-D description of the contour of an object.

Thus we have shape based on both static images and contour descriptions, granularity, coefficient of friction, and surface resilience and texture. These features, when they are better understood, should be incorporated as distinguishing characteristics into the Experimental Sensory Processor.

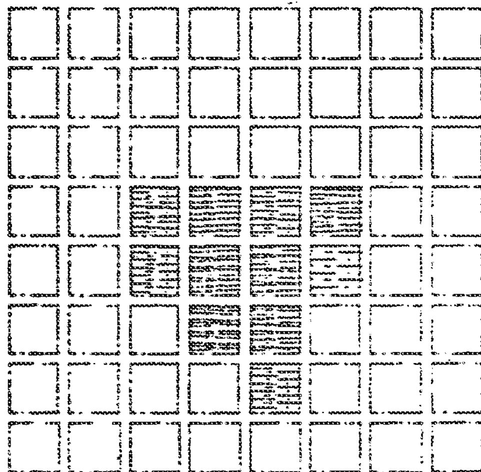


a) 1" diameter circle



d) 1" square rotated 45°

b) 1.5" triangle



e) 1" square rotated 30°

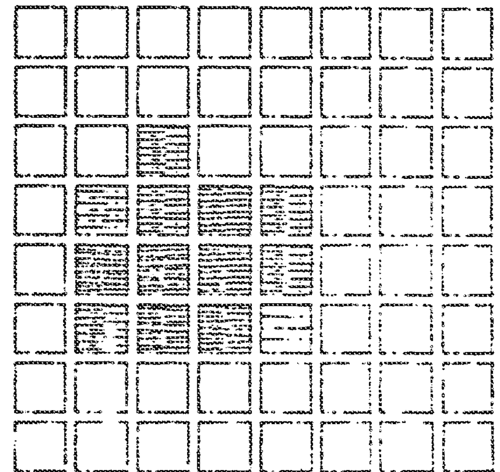
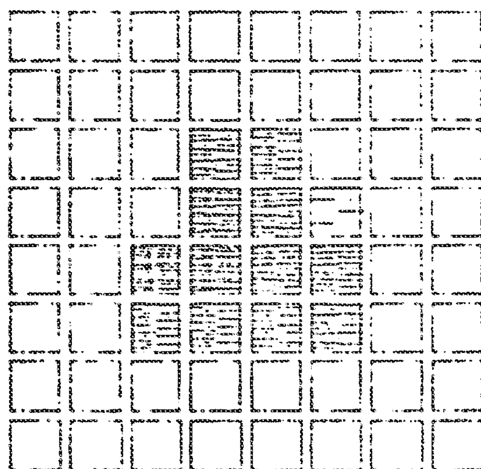
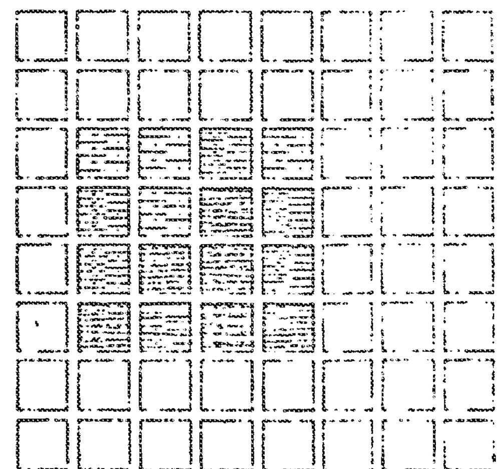


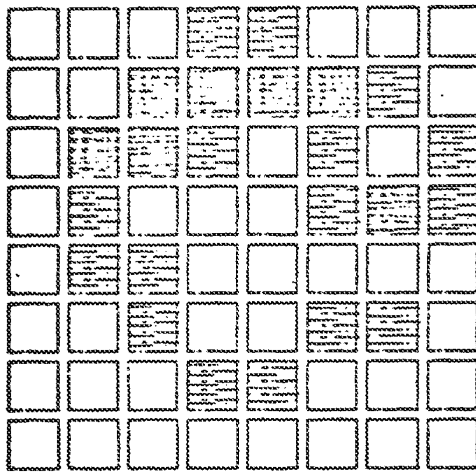
Fig. 1. Single Image Shape Recognition

c) 1.5" triangle rotated 75°

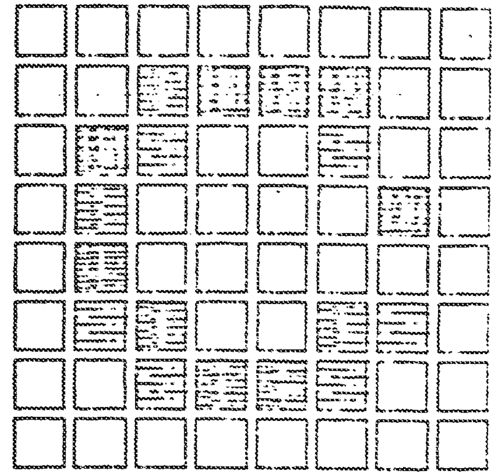


f) 1" square



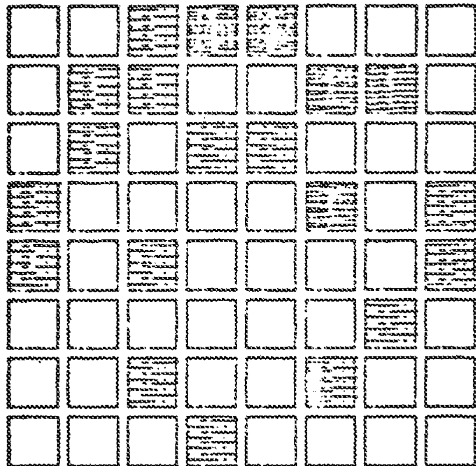


a) One circle, low amplitude

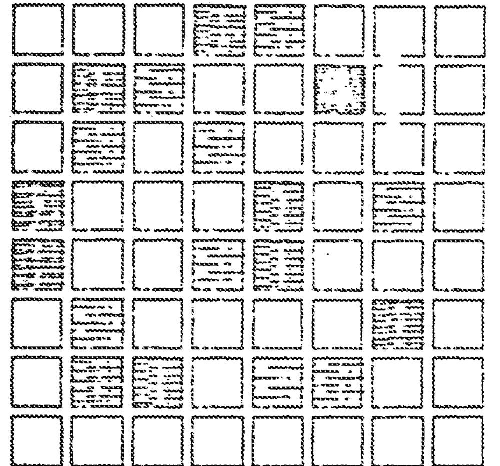


d) One circle, high amplitude

Fig. 2. Single Image Recognition Varying Frequency and Amplitude

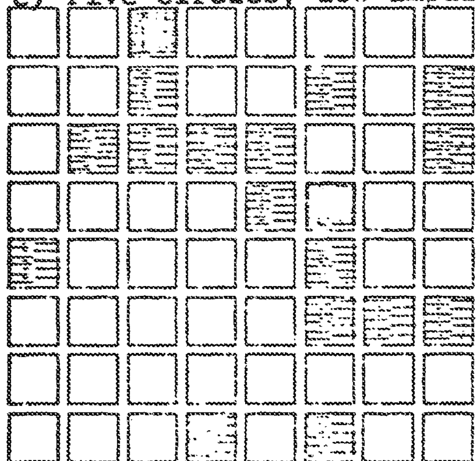


b) Two circles, low amplitude

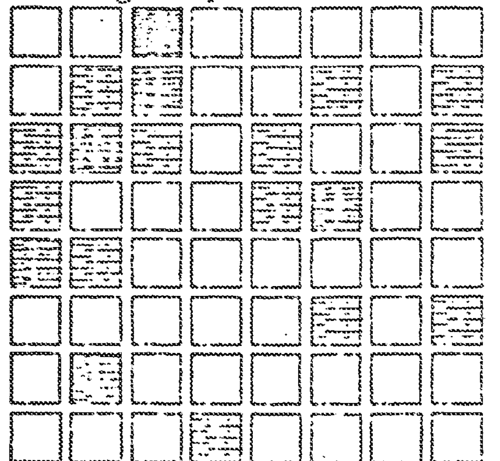


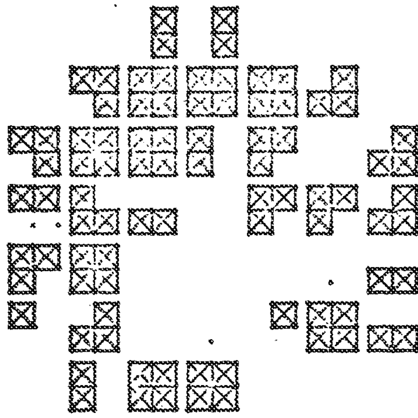
e) Two circles, high amplitude

c) Five circles, low amplitude

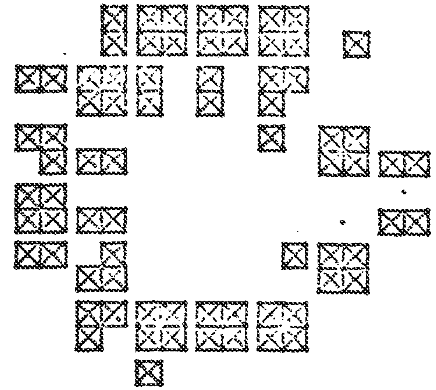


f) Five circles, high amplitude



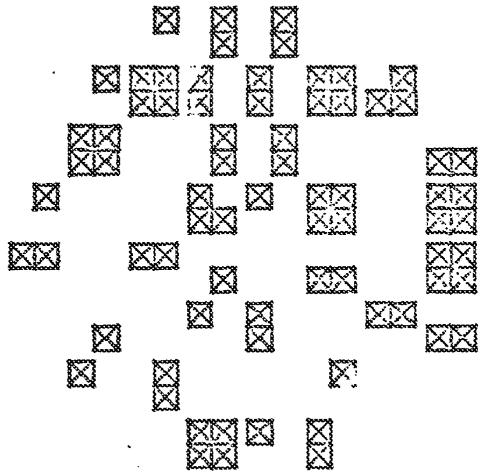


a) One circle, low amplitude

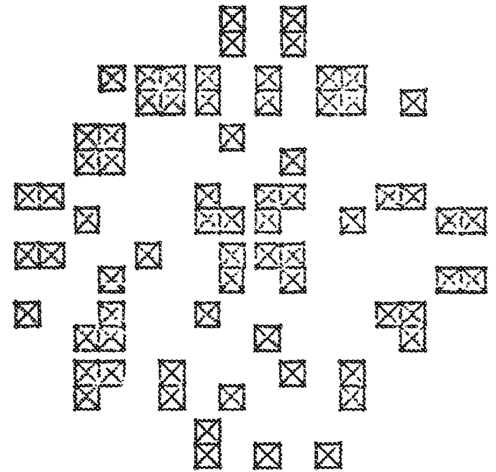


d) One circle, high amplitude

Fig. 3. Multiple Image Recognition
Varying Frequency and Amplitude

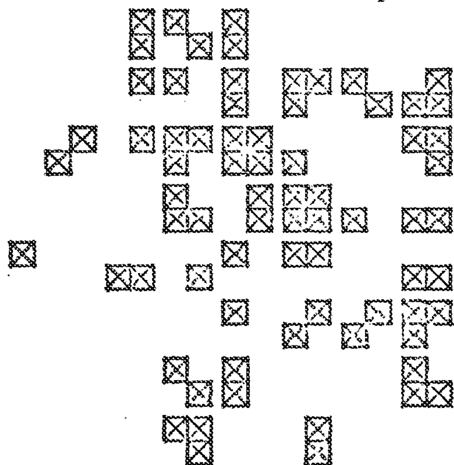


b) Two circles, low amplitude

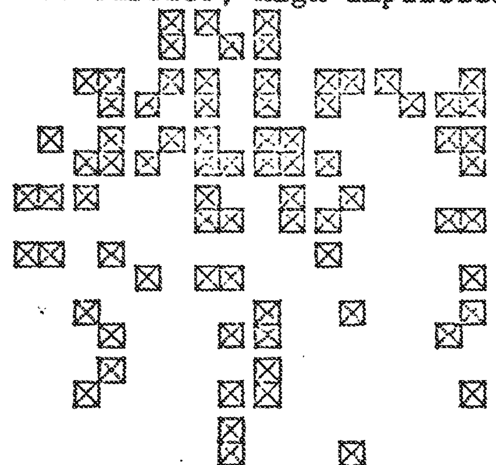


e) Two circles, high amplitude

c) Five circles, low amplitude



f) Five circles, high amplitude



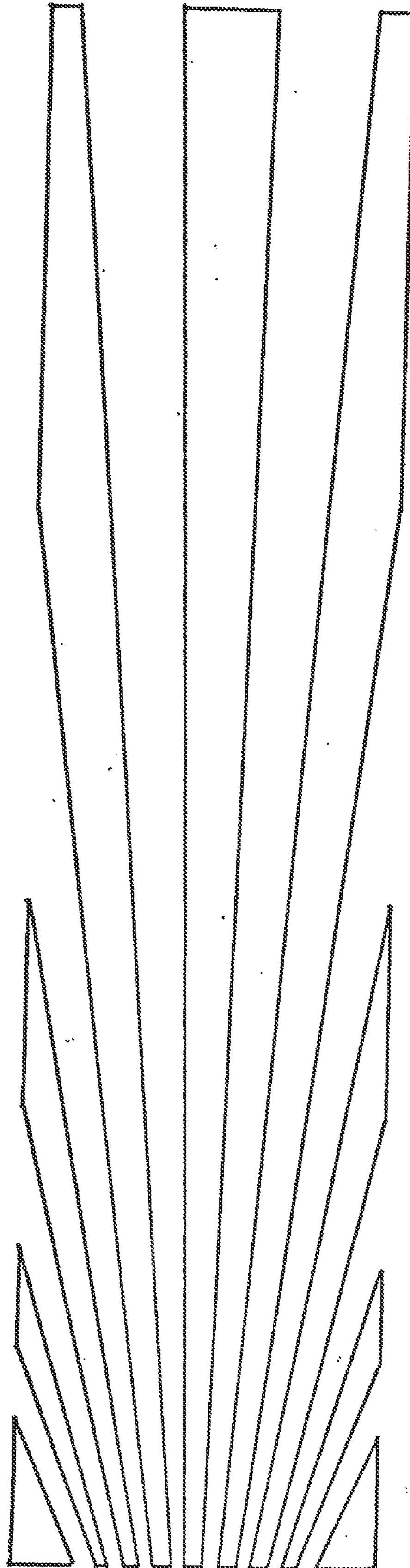


Fig. 4a.
Drawing of the
Large Test Object

Fig. 4b
Using 50 successive frames

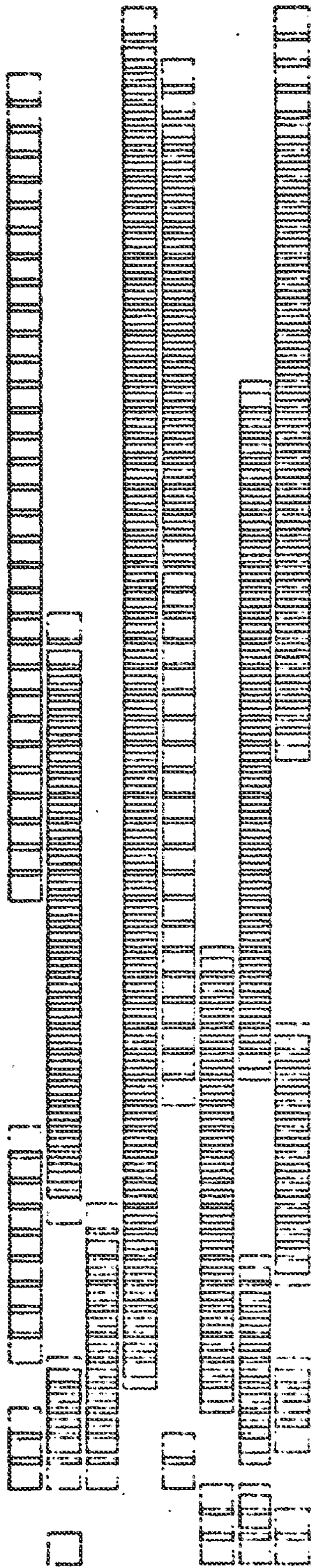


Fig. 4c
Using 3 x 50 matrix of frames

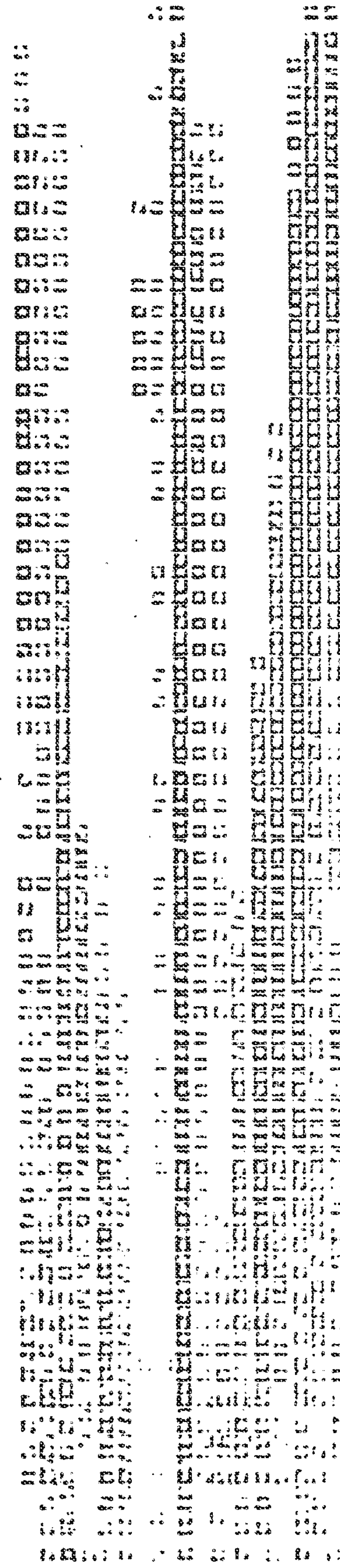


TABLE 1 -- Measurement of Small Angles

Table Slant	Data			Horiz. Difference		Avg. Diff.	Ratio*
0"	45	64	80	19	16	12.625	
	42	48	64	6	16		
	48	60	75	12	15		
	34	56	51	22	-5		
1"	15	64	160	49	96	78	1.00
	28	80	192	52	112		
	16	75	195	59	120		
	17	85	153	68	68		
1.25"		48	160		112	114	1.23
		48	192		144		
		60	180		120		
		56	136		80		
1.5"		48	160		112	142	1.53
		48	240		192		
		60	225		165		
		71	170		99		

* Ratio is calculated as the vertical average divided by the vertical average at 1" slant, multiplied by one minus the ratio of the 1" slant to the 0" slant. The closer this value is to the table slant, the better the results. As the reader can see, the results are exceedingly good.

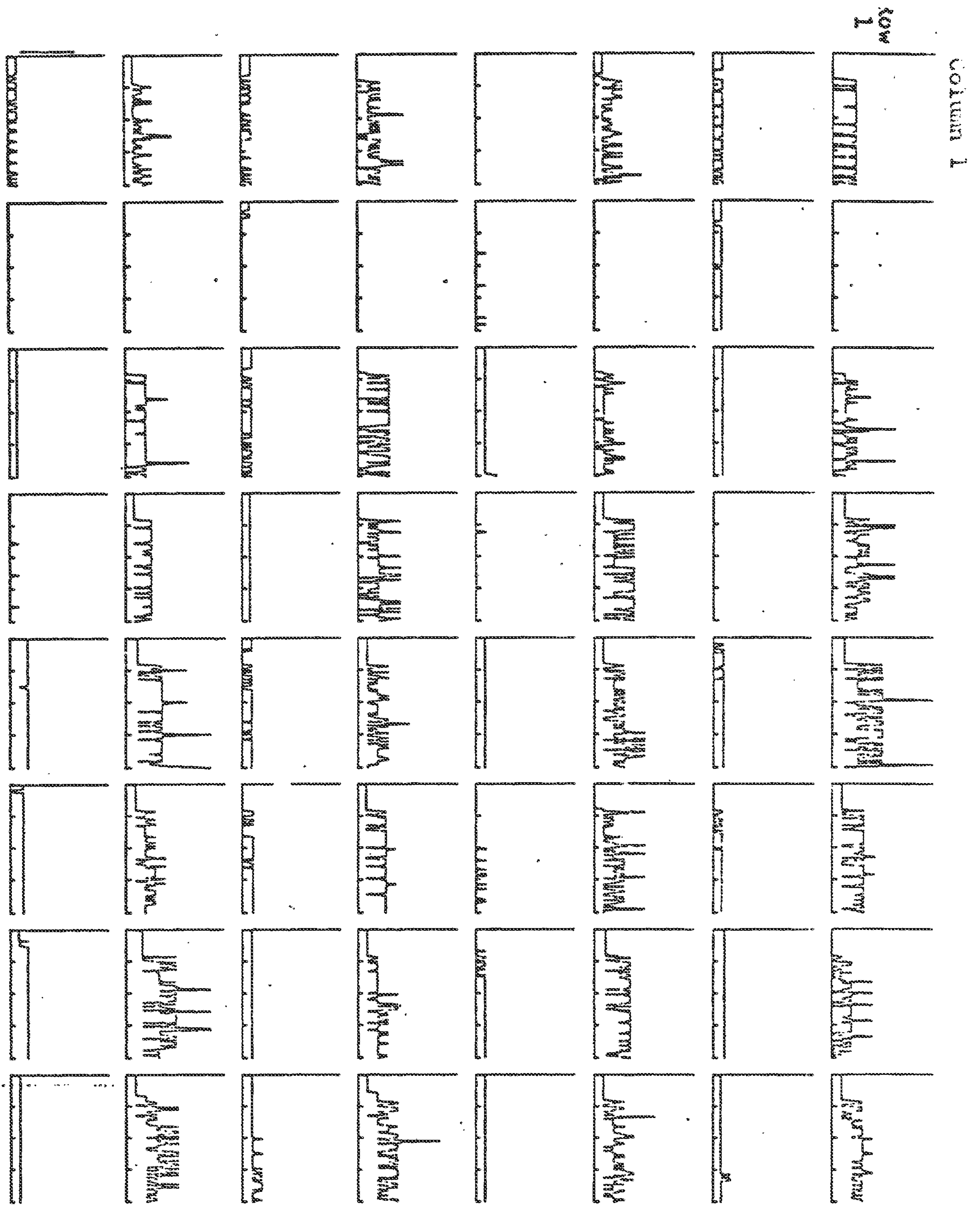


Fig. 5. Raw pressure data as a function of time. Each plot is in the same grid position as the corresponding cone. Sensor was dragged toward +X, with Row 1 in the lead.

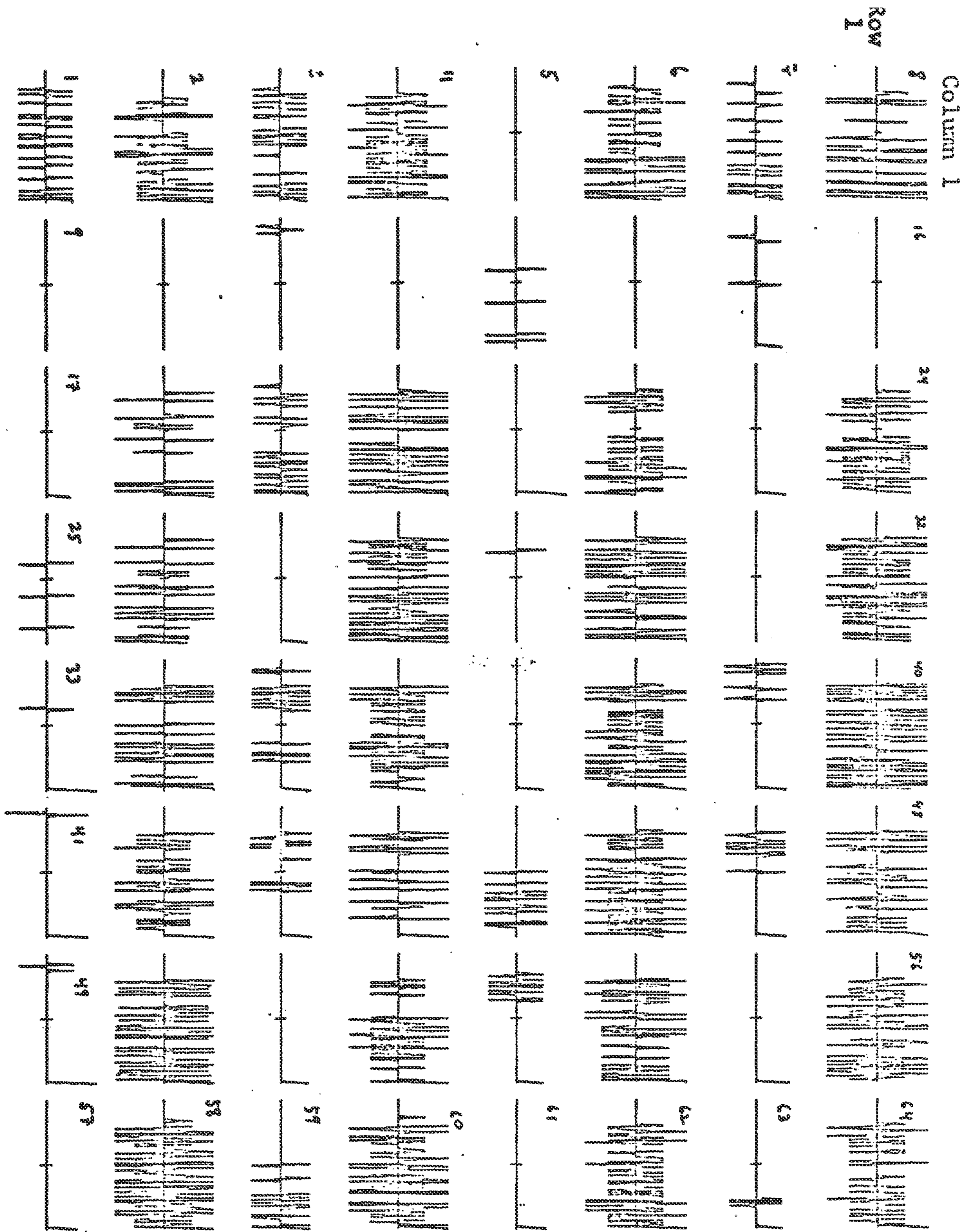


Fig. 6. Pairwise Difference Derivative

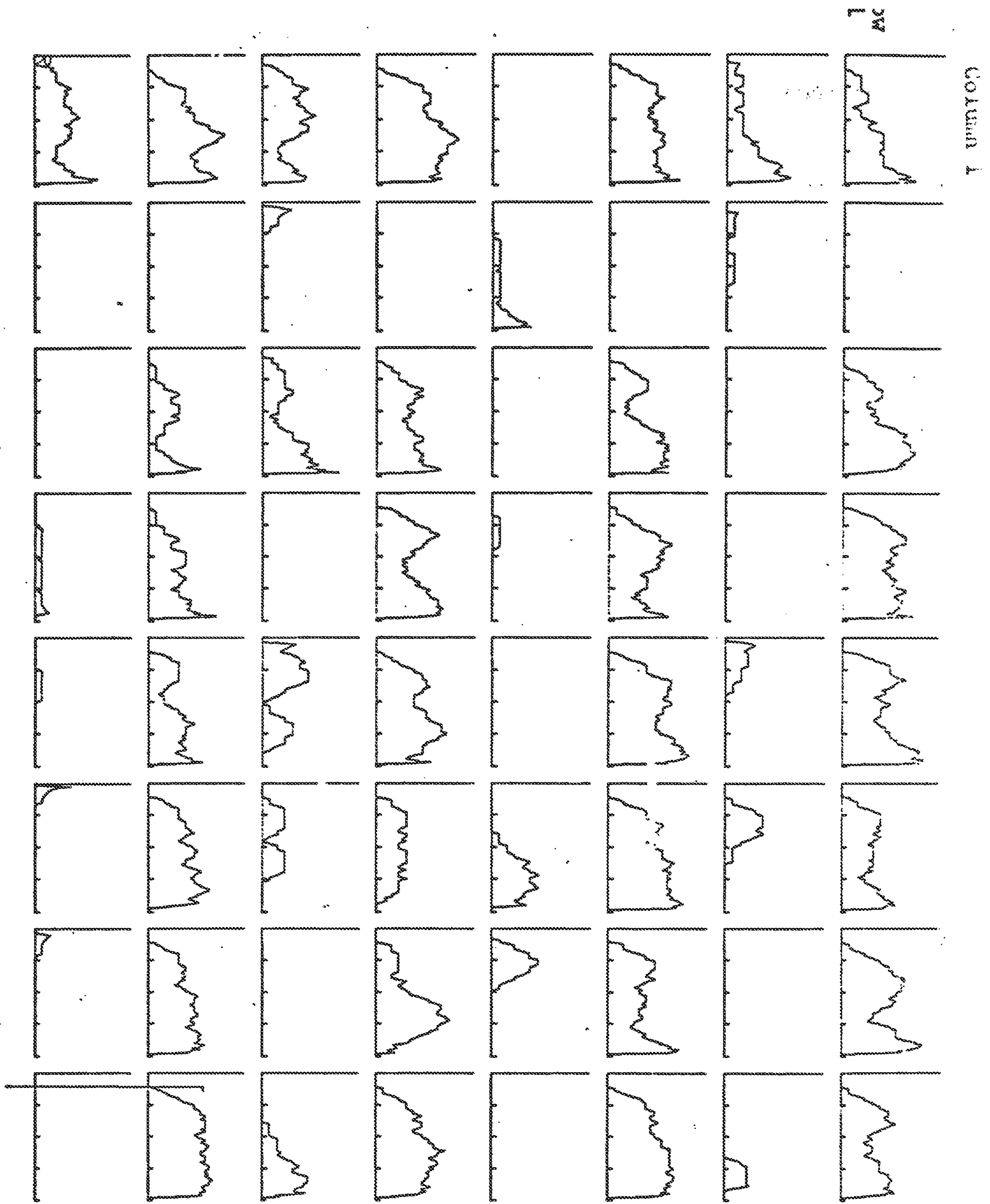


Fig. 7a. Frequency as a Function of Time
Window = 25 Units

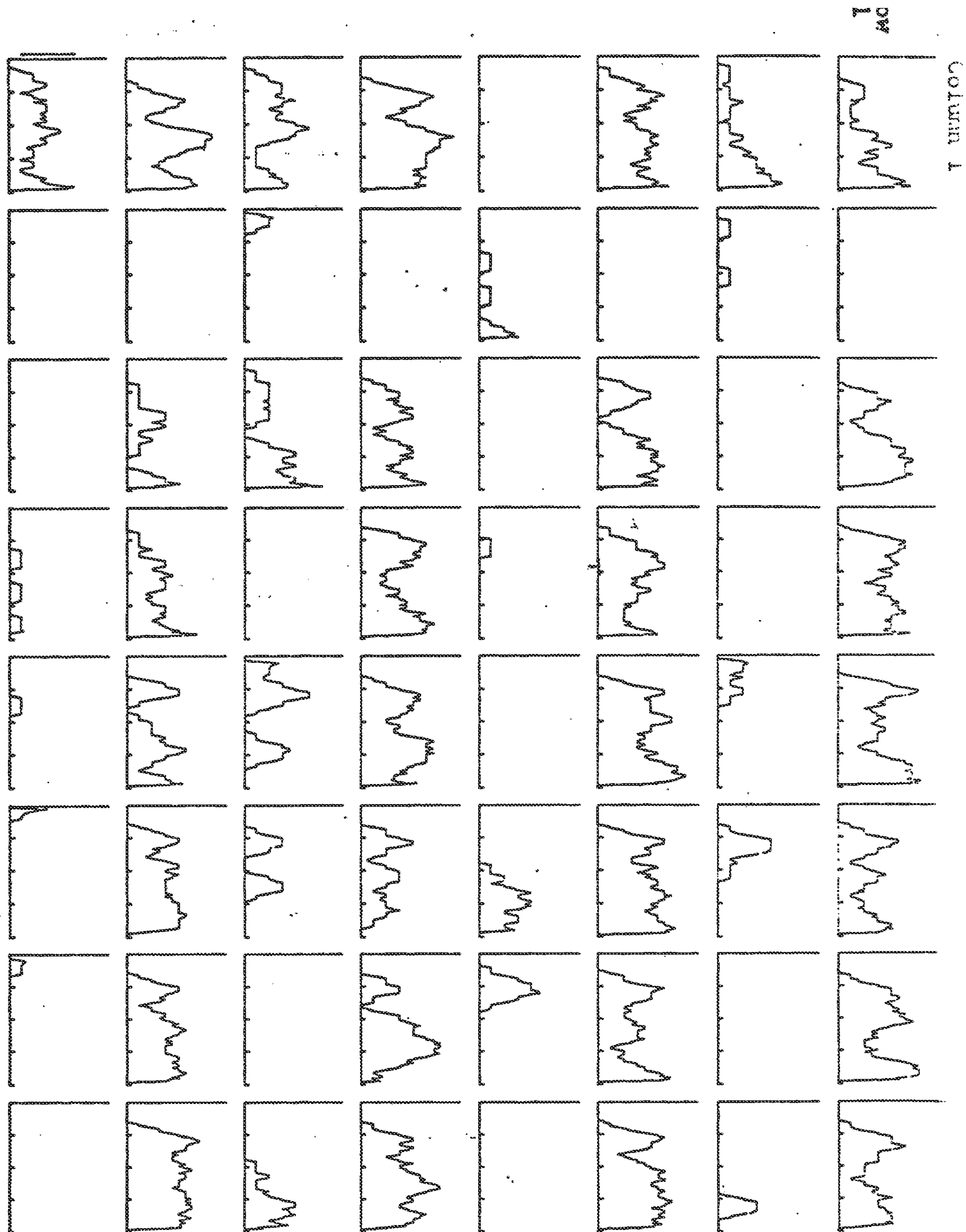


Fig. 7b. Frequency as a Function of Time
Window = 15 Units

60a

References

[BOYKIN-80] Boykin, W. H., and Diaz, Gary., "The Application of Robotic Sensors -- a Survey and Assessment," ASME Century 2 Conference, August 12-15, 1980.

[BRIOT-79] Briot, Maurice, "Utilization of an 'Artificial Skin' Sensor for the Identification of Solid Objects," Proc. of 9th International Symposium on Industrial Robots, Washington, D.C., March 12-15, 1979.

[BROWN-80] Brown, David J., "Computer Architecture for Object Recognition and Sensing," Master's Thesis, Department of Computer and Information Science, University of Pennsylvania, December, 1980.

[DANE-81] Dane, Clayton, Forthcoming PhD. Dissertation, Department of Computer and Information Science, University of Pennsylvania, 1981.

[HILL-73] Hill, John W., and Sword, Antony J., "Touch Sensors and Control," in Remotely Manned Systems -- Exploration and Operation in Space, ed. by Ewald Heer, California Institute of Technology Press, Pasadena, California, 1973.

[HILLIS-81] Hillis, William Daniel, "Active Touch Sensing," Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Memo 629, April, 1981.

[IVANCEVIC-74] Ivancevic, Nebojsa S., "Stereometric Pattern Recognition by Artificial Touch," Pattern Recognition, Vol. 6 pp. 77-83, 1974.

[KINOSHITA-75] Kinoshita, Gen-ichiro, "A Pattern Classification by Dynamic Tactile Sense Info. Processing," Pattern Recognition, Vol. 7 pp. 243-251, 1975.

[NITZAN-80] Nitzan, David, "Assessment of Robotic Sensors," Workshop on the Research Needed to Advance the State of Knowledge in Robotics, April 15-17 1980.

[OKADA-77] Okada, T., and Tsuchiya, S., "Object Recognition by Grasping," Pattern Recognition, Vol. 9 pp. 111-119, 1977.

[PURBRICK-81] Purbrick, John A., "A Force Transducer Employing Conductive Silicone Rubber," Proc. 1st International Conference on Robot Vision and Sensory Controls, Stratford-upon-Avon, UK., IFS (Publications) Ltd., April 1-3, 1981.

[SALISBURY-81] Salisbury, Ken, Stanford Artificial Intelligence Laboratory, Personal Communication, May, 1981, and, Proc. of 1981 Joint Automatic Control Conference, Charlotsville, Virginia, June, 1981.

REAPING THE BENEFITS OF MODERN USABILITY EVALUATION: THE SIMON STORY

James R. Lewis
IBM Human Factors Group
P. O. Box 1328
Boca Raton, FL 33429-1328
Tel: +1 (407) 443-1066
Fax: +1 (407) 443-2778
E-mail: JIMLEWIS@VNET.IBM.COM

product usability; usability evaluation methods; personal communicators

Simon (TM-BellSouth Corp.) is a commercially available personal communicator (PC), combining features of a PDA (personal digital assistant) with a full suite of communications features. This paper describes the involvement of human factors engineering in the development of Simon, and summarizes the various approaches to usability evaluation employed during its development. Simon has received a considerable amount of praise from the industry and won several industry awards, with recognition both for its innovative engineering and its usability.

INTRODUCTION

The Simon is a cellular telephone, designed with a 36 x 115 mm touch screen (CGA resolution) replacing the standard telephone key area. Research in the usability of cellular telephones (Tsoi, 1993) has shown that many of the problems people have using cellular telephones are the result of inflexible control labeling and limited feedback. Replacing the standard key/display area with a touch screen allowed the Simon developers to create a simpler user interface for cellular telephone tasks. It also allowed the development of a suite of applications in addition to the cellular telephone, including an appointment calendar, an address book, a to-do list, a world clock, a note pad, a sketch pad, sending and receiving electronic mail, sending and receiving faxes, reception of pages, file management, a calculator, access to system settings, and security.

My first contact with the Simon development group came as a request to answer an apparently simple question: How small can a touch screen button be, and still be usable? Fortunately, I had just completed a literature review covering the results of human factors studies of touch screens from 1980 to 1992 (Lewis, 1992), so I was able to convey to Simon development that the answer to this simple question was actually somewhat complex and depended on the touch selection strategy (Sears and Schneiderman, 1989). From this start, I spent the next two years as a part of the Simon team, conducting studies and providing usability guidance. The approaches to usability engineering and assessment applied during Simon development illustrate the broad spectrum of modern usability methods, and the resulting product demonstrates the effectiveness of these modern methods. The descriptions appear in rough order of occurrence, but the activities overlapped considerably.

APPROACHES TO USABILITY ENGINEERING AND ASSESSMENT IN THE DEVELOPMENT OF SIMON

Focus Groups

After preliminary design work, an independent agency conducted several focus groups with different types of cellular telephone and computer users to help define the appropriate goals for the product.

Daily Design Meetings

Before writing any significant amount of code, the software team (including a human factors engineer and graphic designer) worked out more specific details about how to achieve the design goals. These meetings lasted for several hours every morning over a period of several months. After each meeting, the individual designers worked on their assignments, which typically involved detailed functional and task analyses. During the meetings, the designers presented their analyses and the rest of the team proposed scenarios for testing the task flows. Determination of problems with task flows in these meetings led to additional refinement of task analyses, which led to refinement of design concepts.

Literature Reviews

Literature reviews of human factors studies of touch screens (Lewis, 1992) and cellular telephone usability provided early, valuable guidance to Simon development. It is often tempting to skip the tedium inherent in a literature review, but keep in mind that it would be foolish to spend three months in the laboratory to obtain information available with an investment of three hours in a library.

Expert Evaluations of Competitive Products

Using an approach similar to Nielsen's (1992) heuristic evaluations, I conducted several expert evaluations of competitive products, both defining the sequence of steps required to perform key tasks and making note of probable problem areas. These evaluations revealed opportunities for improved design in such diverse areas as battery installation and removal, display contrast adjustment, key definition as a function of mode, setting calendar alarms, effective setting and removal of repeating meetings, and clear procedures for setting passwords and locking units.

Development of Test Scenarios

Considering the focus groups, daily design meetings, and expert evaluations of competitive products, the team developed an initial set of 38 test scenarios. By the end of iterative testing, there were 54 scenarios. As suggested by Lewis, Henry, and Mack (1990), some scenarios focused on tasks within a single application, while others evaluated work that crossed application boundaries. We used the scenarios for both gathering competitive performance and satisfaction benchmarks and for iterative problem discovery studies with development-level versions of Simon.

Competitive Usability Benchmarking

One application of the test scenarios was the determination of competitive usability benchmarks for both user performance (scenario completion times and success rates) and satisfaction. We used the After-Scenario Questionnaire (ASQ) to assess user satisfaction following each scenario, and the Post-Study System Usability Questionnaire (PSSUQ) to assess more global usability satisfaction following the completion of all scenarios (Lewis, 1995a). Figure 1 shows the PSSUQ benchmarks established during the competitive usability benchmarking. We collected data from three products regarded as the most likely competitors of Simon. Analysis of the problems discovered during these evaluations provided additional opportunities for improved design in Simon.

Iterative Usability Studies

We conducted three fairly extensive problem discovery studies at different stages during Simon development (early 1992 prototype, first design with reasonably comprehensive function, and the design immediately preceding the final design). Our philosophy for these studies was that measurement of scenario performance and preference variables were important, but that problem discovery was more important. As long as you have competitive benchmarks, scenario measurements give you an idea about where you are relative to your competition, but provide no real guidance about what to do when your product fails to measure up. Analysis of usability problems, on the other hand, provides strong guidance for product redesign. We used the methods described in Lewis (1994b) to determine appropriate sample sizes for these studies. As a consequence of this process of iterative problem identification and design improvement, each iteration showed significant improvement in both user performance and satisfaction. Figure 1 shows the PSSUQ scale ratings for the final iteration (showing means and 95% confidence intervals), with the competitive PSSUQ benchmarks for reference. (A lower PSSUQ score is better than a higher one.) As Figure 1 shows, Simon significantly exceeded its benchmarks for all PSSUQ scales.

Icon Assessment

Most icons that appear on Simon include a descriptive label. There are four icons, however, that appear on every Simon screen. Because these icons appear on every screen, we had a design goal to provide small icons that did not require labels (conserving valuable screen space). We assessed these icons using a battery of icon assessment methods including a matching and confidence task, icon production task, and a semantic differential (Lewis, 1988; Lin, 1992). The outcome of the study indicated a problem with recognition of the icon representing access to the non-phone office tools, and led to re-representation of the function with a focus on its access to a mobile office.

Language Guidelines and Automated Readability Measures

An often neglected area of usability design and evaluation is that of language. Even modern, otherwise usable, systems often contain complicated terms for which there are much more common names. On-line messages and other documentation contain numerous sentences in the passive voice that it would be easy to recast in active voice. These considerations might seem trivial, except that psycholinguistic research has shown that (1) frequency of occurrence of a word in a language significantly affects the speed of human lexical access (Forster, 1990) and (2) it is harder to extract meaning from a passive sentence relative to its active counterpart (Bailey, 1989). To promote clarity and consistency in terminology, I provided the Simon developers with a set of language guidelines, and iteratively reviewed messages and documentation against the guidelines. Our source book for determining the best word to use when considering several synonyms was The Living Word Vocabulary (Dale and O'Rourke, 1981). I also selected random text samples from competitors' documents and developed competitive readability benchmarks for text cloudiness (a measure based on the number of specifically identified abstract words and passivized verbs in a passage divided by the number of words in the passage). At the end of Simon development, measurements taken from a random sample of texts from Simon's documentation showed that the Simon texts had a significantly lower (lower is better) text cloudiness than any of its competitors. Furthermore, using data collected during competitive usability benchmarking and iterative usability studies, Simon had a significantly better PSSUQ Information Quality rating (Lewis, 1995b) than any of its competitors.

Statistical Modeling

Because Simon had a relatively small display area, it was necessary to provide some simple statistical modeling for the size of calendar entries (Lewis, 1993a) and name lengths (Lewis, 1993b) to provide guidance to the calendar and address book developers. The calendar entry research indicated that: (1) managers use computer calendars more than non-managers; (2) managers have more entries per day than

calendar-using nonmanagers; and (3) for user-generated entries, the 95th percentile for the number of characters in an entry was 253. The name length research showed that the mean name length in the United States was about 14 characters, and that a touch-screen button that could show 20 characters would show a person's complete name 99.2% of the time (in the United States).

Designed Experiments

On occasion, it was necessary to conduct designed experiments to answer questions that arose during development. One such experiment (Lewis, 1994a) explored different screen designs for setting dates and times. Although such settings seem straightforward, users have conflicting direction stereotypes that appear to preclude the use of arrows alone for setting times and dates. Two other experiments (Lewis, Allard, and Hudson, 1994; Lewis, 1995a) evaluated different aspects of Simon's predictive keyboard. A predictive keyboard is an on-screen keyboard that contains fewer buttons than a standard keyboard, and uses linguistic probabilities to predict which letters a user will most likely want to type next. These most-likely letters appear in the keyboard's buttons. Lewis, Allard, and Hudson (1994) studied the effects of different word populations, number of displayed letters, and number of trigraph tables on the likelihood that the desired next letter would appear on the predictive keyboard. Lewis (1995a) studied input rates and user preference for the three Simon data input methods (tapping on a small on-screen standard keyboard, tapping on the predictive keyboard, and handwriting on the sketch pad). The results showed that the most effective and preferred input method was tapping on the standard keyboard. In conducting these experiments, the experimental designs described in Lewis (1993c) were quite useful.

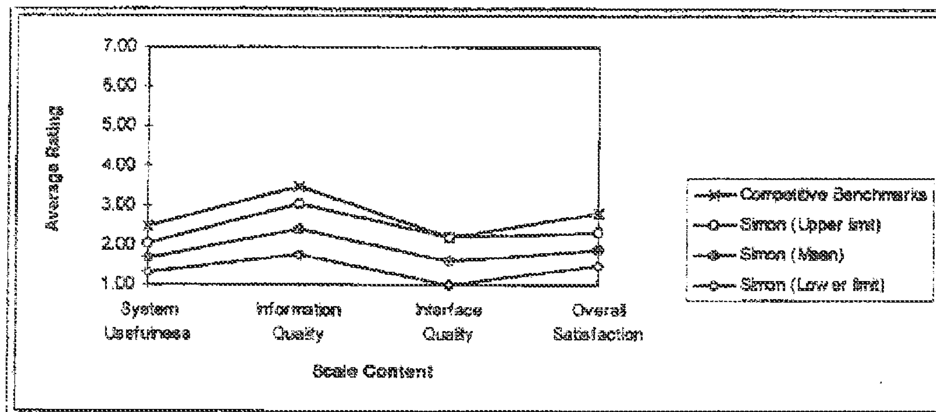


Figure 1. PSSUQ scale scores for Simon and competition

INDUSTRY RECOGNITION

One indication of the success of Simon's design is that it won the Best of Show award at Comdex '93, won an Award of Distinction in the 1994 BYTE awards (BYTE, January 1995), and was a Grand Award winner in the 7th Annual Best of What's New awards (Popular Science, December, 1994). The following quotations from reviews of Simon in trade journals also reflect the success of the usability effort.

"It looks and feels like a product you already know how to use, rather than a new religion you must immerse yourself in." (O'Malley, 1994)

"I hope that Simon is the first in a long series of personal communications tools, but even as a first generation product, Simon is a joy to use." (Nelson, 1995)

"Simon is not the first personal communicator product I've demoed, but it is by far the most comprehensive, well-designed, and easiest to use." (Carter-Lome, 1994)

DISCUSSION

This paper has described the broad range of usability evaluation methods applied to the development of Simon. The industry recognition for Simon stands as evidence for the success of the application of modern usability evaluation methods in this case. The breadth of methods also suggests that professional usability practitioners need to be fluent with a wide array of usability techniques because different development situations demand the application of different usability methods. Some of these methods come from traditional experimental psychology (statistical modeling, designed experiments, literature reviews), and others are more recent techniques (heuristic evaluations, competitive usability benchmarking, scenario-based usability problem discovery studies). All of these techniques have potential application in product development, and deserve a place in the toolbox of the professional usability practitioner.

ACKNOWLEDGMENTS

Successful product development is a team effort, and a human factors engineer is one member of a team. I gratefully acknowledge the contributions of Simon management, software development and hardware development to the final usability of Simon. Also, the efforts of Suvit Nopachai, who served with us as a graduate intern in human factors during Simon development, were invaluable.

REFERENCES

BAILEY, R. W. (1989). Human performance engineering: Using human factors/ergonomics to achieve computer system usability. (Prentice Hall, Englewood Cliffs, NJ).

BYTE. (January 1995). 1994 BYTE awards. BYTE, vol. 20, 49-60.

CARTER-LOME, M. (1994). A Simon for our times. CM: Cellular Marketing, 6.

DALE, E., and O'ROURKE, J. (1981). The living word vocabulary. (World Book, Chicago).

FORSTER, K. (1990). Lexical processing. In Osherson, D. N., and Lasnik, H. (Eds.), Language (pp. 95-131). (MIT Press, Cambridge, MA).

LEWIS, J. R. (1988). A review of symbol test methodologies (Tech. Report 54.475). (IBM Corp., Boca Raton, FL).

LEWIS, J. R. (1992). Literature review of touch-screen research from 1980 to 1992 (Tech. Report 54.694). (IBM Corp., Boca Raton, FL).

LEWIS, J. R. (1993a). Calendar entry statistics for computer calendar users (Tech. Report 54.754). (IBM Corp., Boca Raton, FL)

LEWIS, J. R. (1993b). Name length statistics for touch-screen buttons (Tech. Report 54.810). (IBM Corp., Boca Raton, FL).

LEWIS, J. R. (1993c). Pairs of Latin squares that produce digram-balanced Greco-Latin designs: A BASIC program. Behavior Research Methods, Instruments, and Computers, vol. 25, 414-415.

LEWIS, J. R. (1994a). Direction stereotypes for setting dates and times (Tech. Report 54.867). (IBM Corp., Boca Raton, FL).

LEWIS, J. R. (1994b). Sample sizes for usability studies: Additional considerations. Human Factors, vol. 36, 368-378.

LEWIS, J. R. (1995a). IBM computer usability satisfaction questionnaires: Psychometric evaluation and instructions for use. International Journal of Human-Computer Interaction, vol. 7, 57-78.

LEWIS, J. R. (1995b). Input rates and user preference for three small-screen input methods: Standard keyboard, predictive keyboard and handwriting (Tech. Report 54.889). (IBM Corp., Boca Raton, FL).

LEWIS, J. R., ALLARD, D. J., and HUDSON, H. D. (1994). Predictive keyboard design study: Effects of different word populations, number of displayed letters, and number of trigraph tables (Tech. Report 54.846). (IBM Corp., Boca Raton, FL).

LEWIS, J. R., HENRY, S. C., and MACK, R. L. (1990). Integrated office software benchmarks: A case study. In Human-Computer Interaction -- INTERACT '90 (pp. 337-343). (Elsevier, London, England).

LIN, R. (1992). An application of the semantic differential to icon design. In Proceedings of the Human Factors Society 36th Annual Meeting (pp. 336-340). (Human Factors Society, Santa Monica, CA).

NELSON, M. W. (March/April 1995). The Simon personal communicator. PDA Developer, vol. 3.2, 13-16.

NIELSEN, J. (1992). Finding usability problems through heuristic evaluation. In Proceedings of the Association for Computing Machinery CHI '92 Conference (pp. 373-380). (ACM, Monterey, CA).

O'MALLEY, C. (December, 1994). Simonizing the PDA. BYTE, 145-147.

POPULAR SCIENCE. (December 1994). Best of what's new: The year's 100 greatest achievements in science & technology. Popular Science, 50-76.

SEARS, A., and SCHNEIDERMAN, B. (1989). High precision touchscreens: Design strategies and comparisons with a mouse (Tech. Report CS-TR-2268). (University of Maryland, College Park, MD).

TSOI, K. C. (April 1993). User interface issues for cellular phones. Cellular Business, vol. 10, 32-43.

Soft Machines: A Philosophy of User-Computer Interface Design

Lloyd H. Nakatani
Bell Laboratories, Murray Hill, New Jersey 07974

John A. Rohrlich
Bell Laboratories, Whippany, New Jersey 07981

ABSTRACT

Machines and computer systems differ in many characteristics that have important consequences for the user. Machines are special-purpose, have forms suggestive of their functions, are operated with controls in obvious one-to-one correspondence with their actions, and the consequences of the actions on visible objects are immediately and readily apparent. By contrast, computer systems are general-purpose, have inscrutable form, are operated symbolically via a keyboard with no obvious correspondence between keys and actions, and typically operate on invisible objects with consequences that are not immediately or readily apparent. The characteristics possessed by machines, but typically absent in computer systems, aid learning, use and transfer among machines. But "hard," physical machines have limitations: they are inflexible, and their complexity can overwhelm us. We have built in our laboratory "soft machine" interfaces for computer systems to capitalize on the good characteristics of machines and overcome their limitations. A soft machine is implemented using the synergistic combination of real-time computer graphics to display "soft controls," and a touch screen to make soft controls operable like conventional hard controls.

INTRODUCTION

The juxtaposition of the terms "soft" and "machine" connotes the essence of a philosophy for the design of user-computer interfaces to interactive computer systems. "Machine" connotes an interface which is machine-like in appearance and operation. Such interfaces, we believe, can make computer systems as obvious, easy and efficient to use as well-designed conventional machines. "Soft" connotes a machine realized through computer generated images of controls on a high resolution color display with a touch-sensitive screen for actuating the controls. This software realization gives us the flexibility and power to overcome the limitations of conventional machines.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1983 ACM 0-89791-121-0/83/012/0019 $00.75

From our experience in building prototypes of soft machines in our laboratory, we have become aware of principles underlying the design and use of machines. We hope here to make some of these principles explicit, and to indicate how soft machines based on these principles can lead to better user-computer interfaces. We conclude with thoughts on how a collection of soft machines might be organized.

MACHINES AND COMPUTERS

We are struck by how easy most conventional machines—stoves, tape recorders and calculators—are to use, and how troublesome computer systems are to use by comparison. Machines and computers seem to contrast most sharply on the following aspects of their use[1]:

- *Learning* — We can learn how to use many machines by "playing around" and seeing what happens; learning is usually casual and easy. By contrast, we learn computer systems by reading instruction manuals and seeking help; learning is deliberate and often effortful. The recent flowering of human factors is reflective of this fact.
- *Transfer* — Having mastered a machine, say a copier, we can usually switch to another copier in a matter of minutes. Transfer between machines is generally so easy that we take it for granted and are surprised when it is hard. Having mastered a computer system, say a text editor, we find it relatively hard to learn another text editor. Transfer between computer systems can be so troublesome that ads for word processing personnel specify brands of equipment.
- *Efficiency* — Machines have specialized controls optimized for efficient operation; multi-purpose computers have unspecialized keyboards. We are

1. What follows are broad generalizations. Exceptions and counterexamples can be found, but we feel that the generalizations capture important differences between machines and computer systems which help explain why specialized computer systems are usually more machine-like in design and operation than general-purpose computer systems, and why microprocessor-based consumer products retain their machine-like character.

seeing, however, that as computer systems become more specialized, they acquire specialized, machine-like controls optimized for the functions they perform. For example, dedicated word processors have special function keys, and home computers used for games have joysticks or trackballs that are superior for pointing (Card, English & Burr, 1978; Albert, 1982). And, we observe that dumb machines that acquire microprocessor brains continue to be operated like machines rather than computers. These trends suggest that typical user interfaces to computer systems represent a step backward in interface design compared to the control panel of machines.

For ease of learning, transfer of knowledge and efficiency of operation well-designed hard machines seem better than computers. True, any single machine is not asked to do the wide variety of tasks that we perform with computers, but the superior usability of machines for their intended task is attributable to some intrinsic characteristics of machines that can exploited even for computers intended for multiple functions. What are these intrinsic characteristics?

HARD MACHINES AND HARD CONTROLS

By "hard" machines and controls, we mean conventional machines such as stoves, radios and copiers operated with knobs, switches, keys, pushbuttons and other familiar controls. Hard machines have many characteristics that make for ease of learning, efficiency of operation and ease of transfer, but they are ultimately limited by their "hardness."

Modularity

The modularity of hard machines, most typically mechanical machines, is a natural consequence of design constrained by size, complexity and cost. Modularity is obvious in the kitchen where different machines perform different functions: a stove for cooking, a mixer for mixing, and so on. Modularity keeps complexity within manageable limits, and also provides a "big picture" for organizing the bits of knowledge in learning and using a machine.

Form Follows Function

In machine design, form follows function; in its use, insight follows form. Form encompasses the overall shape of the machine, the control panel, and the individual controls with their labels and markings. Scrutiny of the form leads to conjectures about what the machine does and how it is operated. The conjectures are tested by operating the controls and observing what happens. By "playing around," we discover the what and how of the machine.

One-to-One Mapping of Controls and Operations

The success of "playing around" depends critically on the mapping between the controls on a machine's panel and operations or actions that the machine performs. Ideally, the mapping is one-to-one; that is, corresponding to each machine operation is a control which causes the operation to happen. Then if a machine has N controls, we know that the machine is capable of N and only N operations. This limits the possible conjectures to a reasonable number, and testing each conjecture is a trivial matter of actuating a control and observing what happens. Contrast this with the case where two controls have to be actuated in sequence to get each machine operation to happen. Now there are N×N possible things the machine could do, and N×N possible control sequences. We are unlikely to discover such a machine by playing around because the possibilities are too numerous and testing too tedious. Tape recorders turn this fact into a safety feature by requiring two controls to be actuated in sequence to make a recording. The improbability of discovering the proper sequence makes accidental erasure unlikely by a naive user.

Manual Operation

Machines are operated manually rather than symbolically. *Manual operations* conform to a universal language based on physical laws that govern the interactions between physical objects. Knowledge of this language enables us to cope with novel situations and tasks, usually without instruction or training. For example, if we want to toast bread, from the nature of the bread, toaster and the toasting process, it should be discoverable—if not immediately, then eventually after some trial-and-error—a procedure that will accomplish the task. By contrast, *symbolic operations* require languages which by definition are human inventions. The existence of English and Chinese, FORTRAN and Pascal, and different command languages makes clear that there is no universal language for symbolic operations. The multitude of languages and their arbitrariness is bound to render us illiterate and helpless when faced with a computer that speaks a language we do not know. Suppose, for example, that the toaster was operated by an unknown command language. We are unlikely to discover by trial-and-error how to operate such a toaster.

Immediate Feedback

It goes almost without saying that being able to observe immediately the consequences of our actions is important for evaluating whether our conjectures were correct or not, and for stimulating further conjectures.

The Language of Controls

Over centuries of machine design, a subtle language of controls (Chapanis, 1972) has evolved that we learn from our experience with machines. Designers of machines can use this language to tell us what the machine does and how to use it. Of course, the existence of this language does not guarantee good design, but we believe that a design which does not speak this language is likely to be bad. Some of the important messages in this language follow:

- *Presence* — The presence and absence of controls tell us what the machine can and cannot do. For example, the presence of controls labeled "LIGHTER" and "DARKER" on a copier tell us that we can make copies lighter or darker than the original.
- *Labels* — Good labels, whether text or symbols, tell us what the controls do and thereby what the machine as a whole can do.

- *Type* — The type of control suggests the nature of the thing controlled. For example, a toggle switch controls something with only two states, and a knob controls something that varies continuously.
- *Clustering* — Distinct clusters of controls often correspond to the distinct subfunctions of a machine. A copier may have, for example, a cluster of controls to specify the number of copies, and another to start and stop copying.
- *Arrangement* — The proper arrangement of controls can make labels superfluous. In a car, for example, a rectangular arrangement of power window switches on the center console makes obvious without labels the correspondence between switches and windows.
- *Movement* — Controls operate according to well-established conventions. For example, we flip a light switch up to turn the lights on, and turn the volume knob clockwise to make the music louder.
- *Status* — The settings of the controls can tell us what the machine is doing and what state it is in. On a toaster, for example, the position of the lever tells us that bread is toasting.
- *Graphics* — Graphic cues such as a frame around a group of controls or lines connecting controls can indicate the relationship between controls. On a control panel for a model train layout, for example, a line connects switches controlling points on a common section of track.

Limitations of Hard Controls

The physical and mechanical properties of hard controls make them nice to use. They can be felt and operated without looking, their distinctive movements provide kinesthetic feedback, and their sounds confirm their actuation. Unfortunately, the "hardness" of hard controls is also the source of many limitations.

- *Inflexibility* — The inflexibility of hard controls is the root of other limitations. Hard controls can't appear or disappear, move around, or change their appearance. Inflexible hard controls make for inflexible machines. We are now in an awkward situation where the functionality of machines is easily changed by software, but the inflexibility of hard controls severely limits the changes that can be accommodated without changing the hardware or compromising the operability of the machine. For example, if a keyboard does not provide special cursor positioning keys, we have to make do with controls intended for other uses; most likely, cursor positioning will be more awkward as a consequence.
- *Management of complexity* — Some machines are already too complex for many people, and the use of microprocessors which allow the easy addition of "bells and whistles" will lead to more complexity. The complex electronic calculator compared to the simple mechanical adding machine is an example of this trend. With hard controls it is difficult to keep the complexity from overwhelming us because the progressive disclosure of controls is difficult to achieve. Some machines, television sets for example, hide infrequently used controls behind panels to simplify their appearance and use. The problem of too many controls is aggravated by the compactness made possible by microelectronics. There may be no room on compact machines for controls which are large enough and spaced widely enough to be easily operable. Digital watches indicate the problem. The inflexibility of hard controls limit the complexity that can be easily managed with machines to far below their potential promised by microelectronics.

SOFT MACHINES AND SOFT CONTROLS

Definition and Antecedents

A soft machine can have practically all the advantages of hard machines without the disadvantages that accrue from hardware implementations. A soft machine is implemented by software which simulates hard machines in two important respects. First, a soft machine is made to *look* like a hard machine by graphics software that generates images of controls such as keys, pushbutton switches, and slide potentiometers on the screen of a color video display. The screen serves as a tabula rasa upon which computer systems are visually represented as soft machines through images of their control panels. Second, a soft machine is made to *operate* like a hard machine by covering the display screen with a touch-sensitive position sensor, or touch screen for short. The touch screen enables us to touch and operate the controls in the display as if they were physical controls. And we are not limited to pointing. We can, for example, drag our finger to activate "slide" switches, and forthcoming force-discriminating touch screens will make possible soft controls regulated by pressure. This mode of direct operation of controls by touch rather than through some intermediary pointing device such as a light pen or mouse gives soft machine users a sense of immediacy they would otherwise not have.

The basic ideas underlying soft machines were first articulated by Ken Knowlton (1977) who explored how the inflexibility of hard controls could be overcome partially by optically superimposing computer-generated labels on hard keys. Keys were made to disappear visually and logically by eliminating labels and voiding their operations. Computer graphics and color were used to indicate the clusters of related keys and their proper sequencing.

The first commercial realization of a soft machine to our knowledge is the XEROX 5700 Electronic Printing System (Schuyten, 1980). All the controls for the 5700 appear on a black-and-white video display with a touch screen for operating the controls.

More recently, Schmandt (1981) described a soft machine for editing speech recordings. Like us, Schmandt used color graphics and a touch screen to implement his soft machine. Mirrer (1982) developed in our laboratory a similar but more elaborate soft machine for making hybrid speech documents consisting of a speech recording and associated text outline. We have also developed soft machines for displaying colored speech spectrograms, and

for spreadsheet analysis.

A Calculated Example

An example should make clearer how a soft machine retains the attributes of hard machines that lead to ease in learning and transfer while taking advantage of the power and flexibility of computers to manage complexity.

Our example will be a calculator. The forerunner of the calculator is the adding machine, a hard machine with one purpose and a form suggestive of that purpose. A simple adding machine has few keys, and a complex one has many more. There is a one-to-one correspondence between keys and functions. The close resemblance of an electronic calculator to the adding machine enables us to use a calculator for simple calculations with a bit of exploration and without reading a manual.

In design, today's complex, multifunction calculator is no more than a shrunken adding machine with extra capabilities. It offers some aids to help us manage complexity, but its appearance, except for more labels, reveals little about its added capabilities. Its operation is obscured by keys with multiple labels and mode-dependent actions that require many-to-one mapping of controls onto functions, and by invisible stacks and memory registers that hide their contents.

A calculator implemented as a soft machine makes obvious much more of its functionality and current state while maintaining a simple appearance. The "soft calculator" appears on the screen as a simple four function calculator augmented with keys to access the more complex functions, memory registers and on-line instructions. The placement of the extra keys off to one side and their labels hint at their purpose. Touching one of these keys labeled "STATISTICAL FUNCTIONS" causes it to light up and another group of keys to appear. These new keys enable us to do statistical calculations easily. Touching the "STATISTICAL FUNCTIONS" key again causes it to go dim and the evoked keys to disappear. We can achieve the ideal of a one-to-one mapping between keys and functions regardless of the number of functions the calculator may have because there is ample room on the screen, and keys can disappear when no longer needed. Additional displays are created on demand to store and show intermediate results and useful constants. Such numbers are entered into further calculations simply by touching the corresponding displays. Touching a key labeled "MEMORY" evokes keys to store, recall and accumulate numbers in memory registers with corresponding displays showing their contents.

The calculator is troublesome to represent as a computer system using other interface designs. A calculator operated with a command language could not be learned without consulting a manual. A menu interface would be extremely tedious. Rapid entry of numbers would be difficult by selecting soft keys with a mouse in see-and-point interfaces like those of the XEROX Star (Smith, Irby, Kimball & Verplank, 1982) and Apple Lisa[1] (Ehardt, 1983) professional workstations. Of course, such interfaces will be ideal in other situations and applications. We hope that this calculator example shows clearly and convincingly that a soft machine interface is qualitatively different from command languages, menus and see-and-point interfaces, and that there are circumstances where a soft machine offers obvious advantages.

Operability with Flexibility

A soft machine, properly designed, preserves the essential properties of hard machines that make them easy to use: the global properties—modularity, revealing form, a one-to-one mapping between controls and operations, etc.—and the local properties—presence, labels, type, etc.—that are the language of controls.

A soft machine, furthermore, is flexible. Graphics software enables soft controls to appear and disappear on demand, to move about the screen, and to change appearance so that the form of the machine acquires a dynamic character indicative of the ever changing state of the soft machine. This flexibility gives the designer of soft machines the power to manage the complexity of computer systems to keep us from being overwhelmed. A complex soft machine can be composed of many simpler soft machines, each serving one of the subfunctions of the whole machine. Then to accomplish the overall function, we need deal with only one simple machine at a time. This strategy for managing complexity is essentially identical to the notion of progressive disclosure that characterizes the XEROX[2] Star interface (Smith et al., 1982). This layered approach also overcomes the problem of overcrowding of controls on complex hard machines. Since only those controls relevant to a subfunction need be present at any given time, the limited space on the display screen can be shared among many controls. Hence the space available for controls on a soft machine is practically limitless.

Primitive Operations: Sow's Ears and Silk Purses

A well-designed machine, hard or soft, is comprised of primitive operations which are comprehensible and complete. By comprehensible, we mean that the nature of the operations themselves and how they should be combined and sequenced to accomplish some larger task are easily understood, learned and remembered. By complete, we mean that the operations are sufficient to do all the tasks we demand of the machine. Soft machines represent a way to organize, present and actuate the primitive operations, but leave unanswered an important question in machine design: How do we determine a good set of primitive operations, and rules for combining and sequencing the operations? A sow's ear of a design will not yield a silk purse of a machine—hard or soft. Soft machines are no panacea for bad design, but they do give the designer the flexibility and power to make a good design even better.

1. Apple is a registered trademark, and Lisa a trademark, of Apple Computer.
2. XEROX is a registered trademark of the Xerox Corporation.

ORGANIZING A COLLECTION OF SOFT MACHINES

Work on any substantial project will entail working with a collection of soft machines. We want the collection organized so that we have easy access to all the machines needed for the project with no unneeded machines cluttering our work environment. We propose that our work environment be organized into parallel three-level structures of *tools* (a soft machine is an instance of a tool) and *data* (e.g., documents, spreadsheets and databases.)

For tools, the three levels are tool bin, workshop and workbench. The *tool bin* is the entire collection of tools available on a particular computer. The *workshop* is a work environment specialized for a particular type of work or task such as document preparation or programming, and containing all and only those tools needed to accomplish the task. The tools in the workshop are simply copies to those found in the tool bin. The *workbench* is analogous to a work surface or counter in the workshop where the actual work is done. On the workbench are tools needed just for the current task. These tools are temporary copies which are "put away" when the work is done. These three levels correspond naturally to a houseware store, kitchen, and kitchen counter.

For data, the three levels are file, folder and paper as in the Star and Lisa systems. As in our traditional office environment, *files* contain relatively inactive data, *folders* contain data for an active project, and *papers* represent the aspect of the project that is being actively worked on. Files reside in some independent space, but folders reside in workshops, and papers on workbenches. The analogy to the traditional office environment is clear.

Our houses have evolved special work environments such as the kitchen, bathroom and woodshop to make activities more efficient and to eliminate unwanted interference between activities. We believe that computers should be organized for similar reasons into specialized work environments with both the tools and data needed for particular tasks conveniently and simultaneously on hand.

REFERENCES

1. A. F. Albert, "The Effect of Graphic Input Devices on Performance in a Cursor Positioning Task," *Proceedingsof the Human Factors Society*, 1982, pp. 54-58.
2. S. K. Card, W. K. English and B. J. Burr, "Evaluation of Mouse, Rate-Controlled Isometric Joystick, Step Keys, and Text Keys for Text Selection on a CRT," *Ergonomics*, 21, 1978, pp. 601-613.
3. A. Chapanis, "Design of Controls," in H. P. Van Cott and R. G. Kinkade, *Human Engineering Guide to Equipment Design*, McGraw-Hill, 1972, pp. 346-379.
4. J. L. Ehardt, "Apple's Lisa: A Personal Office System," *The Seybold Report on Office Systems*, 6, January 1983.
5. K. C. Knowlton, "Computer Displays Optically Superimposed on Input Devices," *Bell System Technical Journal*, 56, March 1977, pp. 367-383.
6. B. J. Mirrer, "An Interactive, Graphical, Touch-Oriented Speech Editor," MIT Master's Thesis, 1982.
7. C. Schmandt, "The Intelligent Ear—A Graphical Interface to Digital Audio," *Proceedings of the 1981 International Conference on Cybernetics and Society*, 1981, pp. 393-397.
8. P. J. Schuyten, "Xerox Introduces 'Intelligent Copier,'" *The New York Times*, Thursday, September 25, 1980, p. D4.
9. D. C. Smith, C. Irby, R. Kimball and B. Verplank, "Designing the Star User Interface," *Byte*, April 1982, pp. 242-282.

The Automatic Recognition of Gestures

Dean Harris Rubine

December, 1991

CMU–CS–91–202

Submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Computer Science at Carnegie Mellon University.

Thesis Committee:

Roger B. Dannenberg, Advisor
Dario Giuse
Brad Myers
William A. S. Buxton, University of Toronto

Copyright © 1991 Dean Harris Rubine

Abstract

Gesture-based interfaces, in which the user specifies commands by simple freehand drawings, offer an alternative to traditional keyboard, menu, and direct manipulation interfaces. The ability to specify objects, an operation, and additional parameters with a single intuitive gesture makes gesture-based systems appealing to both novice and experienced users.

Unfortunately, the difficulty in building gesture-based systems has prevented such systems from being adequately explored. This dissertation presents work that attempts to alleviate two of the major difficulties: the construction of gesture classifiers and the integration of gestures into direct-manipulation interfaces. Three example gesture-based applications were built to demonstrate this work.

Gesture-based systems require classifiers to distinguish between the possible gestures a user may enter. In the past, classifiers have often been hand-coded for each new application, making them difficult to build, change, and maintain. This dissertation applies elementary statistical pattern recognition techniques to produce gesture classifiers that are trained by example, greatly simplifying their creation and maintenance. Both single-path gestures (drawn with a mouse or stylus) and multiple-path gestures (consisting of the simultaneous paths of multiple fingers) may be classified. On a 1 MIPS workstation, a 30-class single-path recognizer takes 175 milliseconds to train (once the examples have been entered), and classification takes 9 milliseconds, typically achieving 97% accuracy. A method for classifying a gesture as soon as it is unambiguous is also presented.

This dissertation also describes GRANDMA, a toolkit for building gesture-based applications based on Smalltalk's Model/View/Controller paradigm. Using GRANDMA, one associates sets of gesture classes with individual views or entire view classes. A gesture class can be specified at runtime by entering a few examples of the class, typically 15. The semantics of a gesture class can be specified at runtime via a simple programming interface. Besides allowing for easy experimentation with gesture-based interfaces, GRANDMA sports a novel input architecture, capable of supporting multiple input devices and multi-threaded dialogues. The notion of virtual tools and semantic feedback are shown to arise naturally from GRANDMA's approach.

ii

Acknowledgments

First and foremost, I wish to express my enormous gratitude to my advisor, Roger Dannenberg. Roger was always there when I needed him, never failing to come up with a fresh idea. In retrospect, I should have availed myself more than I did. In his own work, Roger always addresses fundamental problems, and his solutions are always simple and elegant. I try to follow Roger's example in my own work, usually falling far short. Roger, thank you for your insight and your example. Sorry for taking so long.

I was incredibly lucky that Brad Myers showed up at CMU while I was working on this research. His seminar on user interface software gave me the knowledge and breadth I needed to approach the problem of software architectures for gesture-based systems. Furthermore, his extensive comments on drafts of this document improved it immensely. Much of the merit in this work is due to him. Thank you, Brad. I am also grateful to Bill Buxton and Dario Giuse, both of whom provided valuable criticism and excellent suggestions during the course of this work.

It was Paul McAvinney's influence that led me to my thesis topic; had I never met him, mine would have been a dissertation on compiler technology. Paul is an inexhaustible source of ideas, and this thesis is really the *second* idea of Paul's that I've spent multiple years pursuing. Exploring Paul's ideas could easily be the life's work of hundreds of researchers. Thanks, Paul, you madman you.

My wife Ruth Sample deserves much of the credit for the existence of this dissertation. She supported me immeasurably, fed me and clothed me, made me laugh, motivated me to finish, and lovingly tolerated me the whole time. Honey, I love you. Thanks for everything.

I could not have done it with the love and support of my parents, Shirley and Stanley, my brother Scott, my uncle Donald, and my grandma Bertha. For years they encouraged me to be a doctor, and they were not the least bit dismayed when they found out the kind of doctor I wanted to be. They hardly even balked when "just another year" turned out to be six. Thanks, folks, you're the best. I love you all very much.

My friends Dale Amon, Josh Bloch, Blaine Burks, Paul Crumley, Ken Goldberg, Klaus Gross, Gary Keim, Charlie Krueger, Kenny Nail, Eric Nyberg, Barak Pearlmutter, Todd Rockoff, Tom Neuendorffer, Marie-Helene Serra, Ellen Siegal, Kathy Swedlow, Paul Vranesevic, Peter Velikonja, and Brad White all helped me in innumerable ways, from technical assistance to making life worth living. Peter and Klaus deserve special thanks for all the time and aid they've given me over the years. Also, Mark Maimone and John Howard provided valuable criticism which helped me prepare for my oral examination. I am grateful to you all.

I wish to also thank my dog Dismal, who was present at my feet during much of the design, implementation, and writing efforts, and who concurs on all opinions. Dismal, however, strongly objects to this dissertation's focus on *human* gesture.

I also wish to acknowledge the excellent environment that CMU Computer Science provides; none of this work would have been possible without their support. In particular, I'd like to thank Nico Habermann and the faculty for supporting my work for so long, and my dear friends Sharon Burks, Sylvia Berry, Edith Colmer, and Cathy Copetas.

Contents

1 Introduction **1**
1.1 An Example Gesture-based Application 2
1.1.1 GDP from the user's perspective 2
1.1.2 Using GRANDMA to Design GDP's Gestures 4
1.2 Glossary 7
1.3 Summary of Contributions 8
1.4 Motivation for Gestures 9
1.5 Primitive Interactions 12
1.6 The Anatomy of a Gesture 12
1.6.1 Gestural motion 12
1.6.2 Gestural meaning 13
1.7 Gesture-based systems 14
1.7.1 The four states of interaction 15
1.8 A Comparison with Handwriting Systems 16
1.9 Motivation for this Research 17
1.10 Criteria for Gesture-based Systems 18
1.10.1 Meaningful gestures must be specifiable 18
1.10.2 Accurate recognition 18
1.10.3 Evaluation of accuracy 19
1.10.4 Efficient recognition 19
1.10.5 On-line/real-time recognition 19
1.10.6 General quantitative application interface 19
1.10.7 Immediate feedback 20
1.10.8 Context restrictions 20
1.10.9 Efficient training 20
1.10.10 Good handling of misclassifications 20
1.10.11 Device independence 20
1.10.12 Device utilization 21
1.11 Outline 21
1.12 What Is Not Covered 22

2 Related Work 25

2.1 Input Devices . . . 25
2.2 Example Gesture-based Systems . . . 28
2.3 Approaches for Gesture Classification . . . 34
2.3.1 Alternatives for Representers . . . 35
2.3.2 Alternatives for Deciders . . . 37
2.4 Direct Manipulation Architectures . . . 41
2.4.1 Object-oriented Toolkits . . . 43

3 Statistical Single-Path Gesture Recognition 47

3.1 Overview . . . 47
3.2 Single-path Gestures . . . 48
3.3 Features . . . 49
3.4 Gesture Classification . . . 53
3.5 Classifier Training . . . 55
3.5.1 Deriving the linear classifier . . . 55
3.5.2 Estimating the parameters . . . 58
3.6 Rejection . . . 59
3.7 Discussion . . . 61
3.7.1 The features . . . 62
3.7.2 Training considerations . . . 63
3.7.3 The covariance matrix . . . 63
3.8 Conclusion . . . 65

4 Eager Recognition 67

4.1 Introduction . . . 67
4.2 An Overview of the Algorithm . . . 68
4.3 Incomplete Subgestures . . . 69
4.4 A First Attempt . . . 71
4.5 Constructing the Recognizer . . . 72
4.6 Discussion . . . 76
4.7 Conclusion . . . 78

5 Multi-Path Gesture Recognition 79

5.1 Path Tracking . . . 79
5.2 Path Sorting . . . 81
5.3 Multi-path Recognition . . . 83
5.4 Training a Multi-path Classifier . . . 85
5.4.1 Creating the statistical classifiers . . . 85
5.4.2 Creating the decision tree . . . 86
5.5 Path Features and Global Features . . . 86
5.6 A Further Improvement . . . 87
5.7 An Alternate Approach: Path Clustering . . . 88

5.7.1 Global features without path sorting 88
5.7.2 Multi-path recognition using one single-path classifier 88
5.7.3 Clustering 89
5.7.4 Creating the decision tree 92
5.8 Discussion 93
5.9 Conclusion 94

6 An Architecture for Direct Manipulation **95**
6.1 Motivation 95
6.2 Architectural Overview 96
6.2.1 An example: pressing a switch 96
6.2.2 Tools 98
6.3 Objective-C Notation 99
6.4 The Two Hierarchies 101
6.5 Models 101
6.6 Views 103
6.7 Event Handlers 105
6.7.1 Events 105
6.7.2 Raising an Event 106
6.7.3 Active Event Handlers 107
6.7.4 The View Database 109
6.7.5 The Passive Event Handler Search Continues 110
6.7.6 Passive Event Handlers 111
6.7.7 Semantic Feedback 113
6.7.8 Generic Event Handlers 115
6.7.9 The Drag Handler 117
6.8 Summary of GRANDMA 120

7 Gesture Recognizers in GRANDMA **125**
7.1 A Note on Terms 125
7.2 Gestures in MVC systems 126
7.2.1 Gestures and the View Class Hierarchy 126
7.2.2 Gestures and the View Tree 127
7.3 The GRANDMA Gesture Subsystem 128
7.4 Gesture Event Handlers 130
7.5 Gesture Classification and Training 139
7.5.1 Class `Gesture` 139
7.5.2 Class `GestureClass` 140
7.5.3 Class `GestureSemClass` 141
7.5.4 Class `Classifier` 142
7.6 Manipulating Gesture Event Handlers at Runtime 145
7.7 Gesture Semantics 148
7.7.1 Gesture Semantics Code 148

7.7.2 The User Interface . . . 150
7.7.3 Interpreter Implementation . . . 156
7.8 Conclusion . . . 162

8 Applications 163
8.1 GDP . . . 163
8.1.1 GDP's gestural interface . . . 164
8.1.2 GDP Implementation . . . 164
8.1.3 Models . . . 166
8.1.4 `Views` . . . 166
8.1.5 Event Handlers . . . 167
8.1.6 Gestures in GDP . . . 168
8.2 GSCORE . . . 170
8.2.1 A brief description of the interface . . . 170
8.2.2 Design and implementation . . . 173
8.3 MDP . . . 181
8.3.1 Internals . . . 181
8.3.2 MDP gestures and their semantics . . . 188
8.3.3 Discussion . . . 193
8.4 Conclusion . . . 194

9 Evaluation 195
9.1 Basic single-path recognition . . . 195
9.1.1 Recognition Rate . . . 195
9.1.2 Rejection parameters . . . 201
9.1.3 Coverage . . . 205
9.1.4 Varying orientation and size . . . 205
9.1.5 Interuser variability . . . 208
9.1.6 Recognition Speed . . . 213
9.1.7 Training Time . . . 216
9.2 Eager recognition . . . 218
9.3 Multi-finger recognition . . . 221
9.4 GRANDMA . . . 222
9.4.1 The author's experience with GRANDMA . . . 222
9.4.2 A user uses GSCORE and GRANDMA . . . 223

10 Conclusion and Future Directions 225
10.1 Contributions . . . 225
10.1.1 New interactions techniques . . . 225
10.1.2 Recognition Technology . . . 226
10.1.3 Integrating gestures into interfaces . . . 227
10.1.4 Input in Object-Oriented User Interface Toolkits . . . 228
10.2 Future Directions . . . 228

10.3 Final Remarks 233

A Code for Single-Stroke Gesture Recognition and Training 235

A.1 Feature Calculation 235
A.2 Deriving and Using the Linear Classifier 243
A.3 Undefined functions 255

List of Figures

1.1 Proofreader's Gesture (from Buxton [15]) 1
1.2 GDP, a gesture-based drawing program 2
1.3 GDP's View class hierarchy and associated gestures 4
1.4 Manipulating gesture handlers at runtime 5
1.5 Adding examples of the delete gesture 5
1.6 Macintosh Finder, MacDraw, and MacWrite (from Apple [2]) 10

2.1 The Sensor Frame 27
2.2 The DataGlove, Dexterous Hand Master, and PowerGlove (from Eglowstein [32]) 27
2.3 Proofreading symbols (from Coleman [25]) 28
2.4 Note gestures (from Buxton [21]) 29
2.5 Button Box (from Minksy [86]) 30
2.6 A gesture-based spreadsheet (from Rhyne and Wolf [109]) 30
2.7 Recognizing flowchart symbols 31
2.8 Sign language recognition (from Tamura [128]) 32
2.9 Copying a group of objects in GEdit (from Kurtenbach and Buxton [75]) 33
2.10 GloveTalk (from Fels and Hinton [34]) 33
2.11 Basic PenPoint gestures (from Carr [24]) 34
2.12 Shaw's Picture Description Language 39

3.1 Some example gestures 48
3.2 Feature calculation 51
3.3 Feature vector computation 54
3.4 Two different gestures with identical feature vectors 62
3.5 A potentially troublesome gesture set 64

4.1 Eager recognition overview 68
4.2 Incomplete and complete subgestures of U and D 70
4.3 A first attempt at determining the ambiguity of subgestures 71
4.4 Step 1: Computing complete and incomplete sets 73
4.5 Step 2: Moving accidentally complete subgestures 75
4.6 Accidentally complete subgestures have been moved 76
4.7 Step 3: Building the AUC 76

4.8 Step 4: Tweaking the classifier 77
4.9 Classification of subgestures of U and D 77

5.1 Some multi-path gestures 80
5.2 Inconsistencies in path sorting 82
5.3 Classifying multi-path gestures 84
5.4 Path Clusters 91

6.1 GRANDMA's Architecture 97
6.2 The Event Hierarchy 106

7.1 GRANDMA's gesture subsystem 129
7.2 Passive Event Handler Lists 146
7.3 A Gesture Event Handler 147
7.4 Window of examples of a gesture class 147
7.5 The interpreter window for editing gesture semantics 152
7.6 An empty message and a selector browser 153
7.7 Attributes to use in gesture semantics 154

8.1 GDP gestures 165
8.2 GDP's class hierarchy 165
8.3 GSCORE's cursor menu 170
8.4 GSCORE's palette menu 171
8.5 GSCORE gestures 172
8.6 A GSCORE session 174
8.7 GSCORE's class hierarchy 175
8.8 An example MDP session 182
8.9 MDP internal structure 184
8.10 MDP gestures 189

9.1 GSCORE gesture classes used for evaluation 196
9.2 Recognition rate vs. number of classes 197
9.3 Recognition rate vs. training set size 197
9.4 Misclassified GSCORE gestures 199
9.5 A looped corner 200
9.6 Rejection parameters 202
9.7 Counting correct and incorrect rejections 203
9.8 Correctly classified gestures with $d^2 \geq 90$ 204
9.9 Correctly classified gestures with $\tilde{P} \leq .95$ 204
9.10 Recognition rates for various gesture sets 206
9.11 Classes used to study variable size and orientation 207
9.12 Recognition rate for set containing classes that vary 208
9.13 Mistakes in the variable class test 209
9.14 Testing program (user's gesture not shown) 209

9.15 PV's misclassified gestures (author's set) 211
9.16 PV's gesture set 212
9.17 The performance of the eager recognizer on easily understood data 219
9.18 The performance of the eager recognizer on GDP gestures 220
9.19 PV's task 223
9.20 PV's result 223

List of Tables

9.1 Speed of various computers used for testing 214
9.2 Speed of feature calculation 214
9.3 Speed of Classification 215
9.4 Speed of classifier training 217

to Grandma Bertha

Chapter 1

Introduction

People naturally use hand motions to communicate with other people. This dissertation explores the use of human gestures to communicate with computers.

Random House [122] defines "gesture" as "the movement of the body, head, arms, hands, or face that is expressive of an idea, opinion, emotion, etc." This is a rather general definition, which characterizes well what is generally thought of as gesture. It might eventually be possible through computer vision for machines to interpret gestures, as defined above, in real time. Currently such an approach is well beyond the state of the art in computer science.

Because of this, the term "gesture" usually has a restricted connotation when used in the context of human-computer interaction. There, gesture refers to hand markings, entered with a stylus or mouse, which function to indicate scope and commands [109]. Buxton [14] gives a fine example, reproduced here as figure 1.1. In this dissertation, such gestures are referred to as *single-path* gestures.

Recently, input devices able to track the paths of multiple fingers have come into use. The Sensor Frame [84] and the DataGlove [32, 130] are two examples. The human-computer interaction community has naturally extended the use of the term "gesture" to refer to hand motions used to indicate commands and scope, entered via such multiple finger input devices. These are referred to here as *multi-path* gestures.

Rather than defining gesture more precisely at this point, the following section describes an

Ideally, we want a one-to-one mapping between concepts and gestures. User interfaces should be designed with a clear objective of the mental model we are trying to establish. Phrasing can reinforce the chunks or structure of the model.

Figure 1.1: Proofreader's Gesture (from Buxton [15])

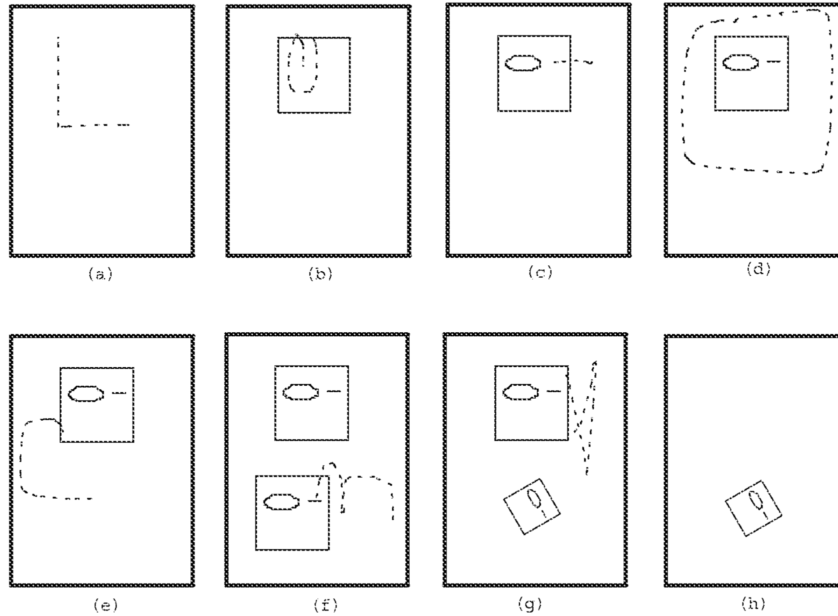


Figure 1.2: GDP, a gesture-based drawing program

example application with a gestural interface. A more technical definition of gesture will be presented in section 1.6.

1.1 An Example Gesture-based Application

GRANDMA is a toolkit used to create gesture-based systems. It was built by the author and is described in detail in the pages that follow. GRANDMA was used to create GDP, a gesture-based drawing editor loosely based on DP [42]. GDP provides for the creation and manipulation of lines, rectangles, ellipses, and text. In this section, GDP is used as an example gesture-based system. GDP's operation is presented first, followed by a description of how GRANDMA was used to create GDP's gestural interface.

1.1.1 GDP from the user's perspective

GDP's operation from a user's point of view will now be described. (GDP's design and implementation is presented in detail in Section 8.1.) The intent is to give the reader a concrete example of a gesture-based system before embarking on a general discussion of such systems. Furthermore, the description of GDP serves to illustrates many of GRANDMA's capabilities. A new interaction technique, which combines gesture and direct manipulation in a single interaction, is also introduced in the description.

Figure 1.2 shows some snapshots of GDP in action. When first started, GDP presents the user with a blank window. Panel (a) shows the rectangle gesture being entered. This gesture is drawn like an "L."[1] The user begins the gesture by positioning the mouse cursor and pressing a mouse button. The user then draws the gesture by moving the mouse.

The gesture is shown on the screen as is being entered. This technique is called *inking* [109], and provides valuable feedback to the user. In the figure, inking is shown with dotted lines so that the gesture may be distinguished from the objects in the drawing. In GDP, the inking is done with solid lines, and disappears as soon as the gesture has been recognized.

The end of the rectangle gesture is indicated in one of two ways. If the user simply releases the mouse button immediately after drawing "L" a rectangle is created, one corner of which is at the start of the gesture (where the button was first pressed), with the opposite corner at the end of the gesture (where the button was released). Another way to end the gesture is to stop moving the mouse for a given amount of time (0.2 seconds works well), while still pressing the mouse button. In this case, a rectangle is created with one corner at the start of the gesture, and the opposite corner at the current mouse location. As long as the button is held, that corner is dragged by the mouse, enabling the size and shape of the rectangle to be determined interactively.

Panel (b) of figure 1.2 shows the rectangle that has been created and the ellipse gesture. This gesture creates an ellipse with its center at the start of the gesture. A point on the ellipse tracks the mouse after the gesture has been recognized; this gives the user interactive control over the size and eccentricity of the ellipse.

Panel (c) shows the created ellipse, and a line gesture. Similar to the rectangle and the ellipse, the start of the gesture determines one endpoint of the newly created line, and the mouse position after the gesture has been recognized determines the other endpoint, allowing the line to be rubberbanded.

Panel (d) shows all three shapes being encircled by a pack gesture. This gesture packs (groups) all the objects which it encloses into a single composite object, which can then be manipulated as a unit. Panel (e) shows a copy gesture being made; the composite object is copied and the copy is dragged by the mouse.

Panel (f) shows the rotate-and-scale gesture. The object is made to rotate around the starting point of the gesture; a point on the object is dragged by the mouse, allowing the user to interactively determine the size and orientation of the object.

Panel (g) shows the delete gesture, essentially an "X" drawn with a single stroke. The object at the gesture start is deleted, as shown in panel (h).

This brief description of GDP illustrates a number of features of gesture-based systems. Perhaps the most striking feature is that each gesture corresponds to a high-level operation. The class of the gesture determines the operation; attributes of the gesture determine its scope (the operands) and any additional parameters. For example, the delete gesture specifies the object to be deleted, the pack gesture specifies the objects to be combined, and the line gesture specifies the endpoints of the line.

[1] It is often convenient to describe single-path gestures as if they were handwritten letters. This is not meant to imply that gesture-based systems can only recognize alphabetic symbols, or even that they usually recognize alphabetic symbols. The many ways in which gesture-based systems are distinct from handwriting-recognition systems will be enumerated in section 1.8.

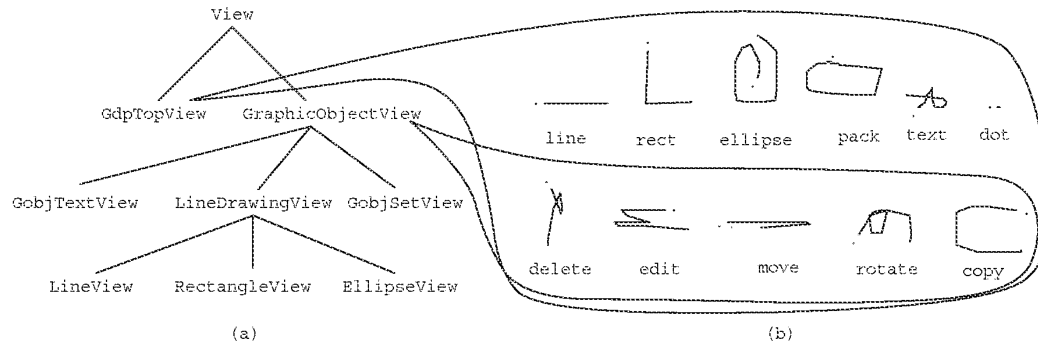


Figure 1.3: GDP's `View` class hierarchy and associated gestures
A period indicates the first point of each gesture.

It is possible to control more than positional parameters with gestural attributes. For example, one version of GDP uses the length (in pixels) of the line gesture to control the thickness of the new line.

Note how gesturing and direct manipulation are combined in a new two-phase interaction technique. The first phase, the collection of the gesture, ends when the user stops moving the mouse while holding the button. At that time, the gesture is recognized and a number of parameters to the application command are determined. After recognition, a manipulation phase is entered during which the user can control additional parameters interactively.

In addition to its gestural interface, GDP provides a more traditional click-and-drag interface. This is mainly used to compare the two styles of interface, and is further discussed in Section 8.1. The gestural interface is grafted on top of the click-and-drag interface, as will be explained next.

1.1.2 Using GRANDMA to Design GDP's Gestures

In the current work, the *gesture designer* creates a gestural interface to an application out of an existing click-and-drag interface to the application. Both the click-and-drag interface and the application are built using the object-oriented toolkit GRANDMA. The gesture designer only modifies the way input is handled, leaving the output mechanisms untouched.

A system built using GRANDMA utilizes the object-oriented programming paradigm to represent windows and the graphics objects displayed in windows. For example, figure 1.3a shows GDP's `View` class hierarchy.[2] This hierarchy shows the relationship of the classes concerned with output. The task of the gesture designer is to determine which of these classes are to have associated gestures, and for each such view class, to design a set of gestures that intuitively expresses the allowable operations on the view. Figure 1.2b shows the sets of gestures associated with GDP's `GraphicObjectView` and `GdpTopView` classes. The `GraphicObjectView` collectively

[2]For expositional purposes, the hierarchy shown is a simplified version of the actual hierarchy. Some of the details that follow have also been simplified. Section 8.1 tells the truth in gory detail.

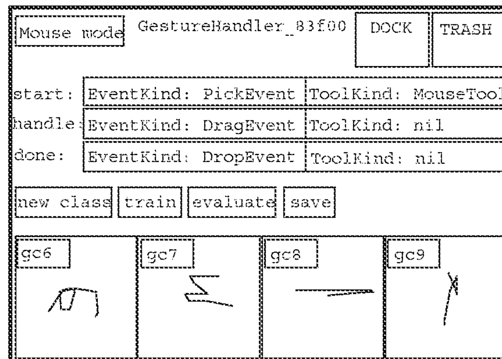


Figure 1.4: Manipulating gesture handlers at runtime

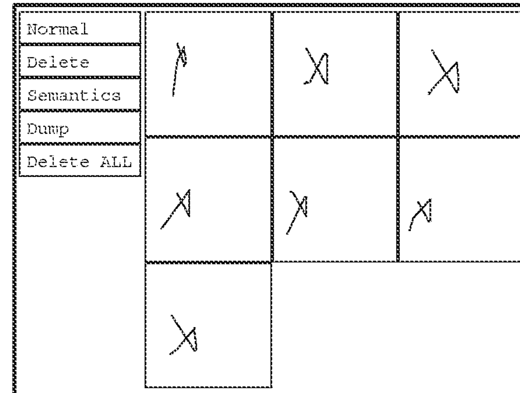


Figure 1.5: Adding examples of the delete gesture

refers to the line, rectangle, and ellipse shapes, while `GdpTopView` represents the window in which GDP runs.

GRANDMA is a Model/View/Controller-like system [70]. In GRANDMA, an input event handler (a "controller" in MVC terms) may be associated with a view class, and thus shared between all instances of the class (including instances of subclasses). This adds flexibility while eliminating a major overhead of Smalltalk MVC, where one or more controller objects are associated with each view object that expects input.

The gesture designer adds gestures to GDP's initial click-and-drag interface at runtime. The first step is to create a new gesture handler and associate it the `GraphicObjectView` class, easily done using GRANDMA. Figure 1.4 shows the gesture handler window after a number of gestures have been created (using the "new class" button), and figure 1.5 shows the window in which examples of the delete gesture have been entered. Fifteen examples of each gesture class typically suffice. If a gesture is to vary in size and/or orientation, the examples should reflect that.

Clicking on the "Semantics" button brings up a window that the designer uses to specify the semantics of each gesture in the handler's set. The window is a structured editing and browsing interface to a simple Objective-C [28] interpreter, and the designer enters three expressions: `recog`, evaluated when the gesture is first recognized; `manip`, evaluated on subsequent mouse points; and `done`, evaluated when the mouse button is released. In this case, the delete semantics simply change the mouse cursor to a delete cursor, providing feedback to the user, and then delete the view at which the gesture was aimed. The expressions entered are[3]

[3]Objective C syntax is used throughout. `[view delete]` sends the `delete` message to the object referred to by the variable `view`. `[handler mousetool:DeleteCursor]` sends the `mousetool:` message to the object referred to by the variable `handler` passing the value of the variable `DeleteCursor` as an argument. See Section 6.3 for more information on Objective C notation.

```
recog = [_Seq :[handler mousetool:DeleteCursor]
              :[view delete]];
manip = nil;
done = nil;
```

The designer may now immediately try out the delete gesture, as in figure 1.2g.

The designer repeats the process to create a gesture handler for the set of gestures associated with class GdpTopView, the view that refers to the window in which GDP runs. This handler deals with the gestures that create graphic objects, the pack gesture (which creates a set out of the enclosed graphic objects), the dot gesture (which repeats the last command), and the gestures also handled by GraphicObjectView's gesture handler (which when made at a GdpTopView change the cursor without operating directly on a graphic object).

The attributes of the gesture are directly available for use in the gesture semantics. For example, the semantics of the line gesture are:

```
recog = [Seq :[handler mousetool:LineCursor]
             :[[view createLine]
                     setEndpoint:0 x:<startX> y:<startY>]];
manip = [recog setEndpoint:1 x:<currentX> y:<currentY>];
done = nil;
```

The semantic expressions execute in a rich environment in which, for example, view is bound to the view at which the gesture was directed (in this case a GdpTopView) and handler is bound to the current gesture handler. Note that Seq executes its arguments sequentially, returning the value of the last, in this case the newly created line. This is bound to recog for later use in the manip expression.

The example shows how the gesture attributes, shown in angle brackets, are useful in the semantic expressions. The attributes <startX> and <startY>, the coordinates of the first point in the gesture, are used to determine one endpoint of the line, while <currentX> and <currentY>, the mouse coordinates, determine the other endpoint.

Many other gesture attributes are useful in semantics. The line semantics could be augmented to control the thickness of the line from the maximum speed or total path length of the gesture. The rectangle semantics could use the initial angle of the rectangle gesture to determine the orientation of the rectangle. The attribute <enclosed> is especially noteworthy: it contains a list of views enclosed by the gesture and is used, for example, by the pack gesture (figure 1.2d). When convenient, the semantics can simulate input to the click-and-drag interface, rather than communicating directly with application objects or their views, as shown above.

When the first point of a gesture is over more than one gesture-handling view, the union of the set of gestures recognized by each handler is used, with priority given to the foremost views. For example, any gesture made at a GDP GraphicObjectView is necessarily made over the GdpTopView. A delete gesture made at a graphic object would be handled by the GraphicObjectView while a line gesture at the same place would be handled by the GdpTopView. Set union also occurs when gestures are (conceptually) inherited via the view class hierarchy. For example, the gesture designer might create a new gesture handler for the GobjSetView class containing an unpack gesture. The set of gestures recognized by

GobjSetViews would then consist of the unpack gesture as well as the five gestures handled by GraphicObjectView.

1.2 Glossary

This section defines and clarifies some terms that will be used throughout the dissertation. It may safely be skipped and referred back to as needed. Some of the terms (click, drag) have their common usage in the human-computer interaction community, while others (pick, move, drop) are given technical definitions solely for use here.

class In this dissertation, "class" is used in two ways. "Gesture class" refers to a set of gestures all of which are intended to be treated the same, for example, the class of delete gestures. (In this dissertation, the names of gesture classes will be shown in sans serif typeface.) The job of a gesture recognizer is, given an example gesture, to determine its class (see also "gesture"). "Class" is also used in the object-oriented sense, referring to the type (loosely speaking) of a software object. It should be clear from context which of these meanings is intended.

click A click consists of positioning the mouse cursor and then pressing and releasing a mouse button, with no intervening mouse motion. In the Macintosh, a click is generally used to select an object on the screen.

click-and-drag A click-and-drag interface is a direct-manipulation interface in which objects on the screen are operated upon using mouse clicks, drags, and sometimes double-clicks.

direct manipulation A direct-manipulation interface is one in which the user manipulates a graphic representation of the underlying data by pointing at and/or moving them with an appropriate device, such as a mouse with buttons.

double-click A double-click is two clicks in rapid succession.

drag A drag consists of locating the mouse cursor and pressing the mouse button, moving the mouse cursor while holding the mouse button, and then releasing the mouse button. Drag interactions are used in click-and-drag interfaces to, for example, move objects around on the screen.

drop The final part of a drag (or click) interaction in which the mouse button is released.

eager recognition A kind of gesture recognition in which gestures are often recognized without the end of the gesture having to be explicitly signaled. Ideally, an eager recognizer will recognize a gesture as soon as enough of it has been seen to determine its class unambiguously.

gesture Essentially a freehand drawing used to indicate a command and all its parameters. Depending on context, the term maybe used to refer to an example gesture or a class of gestures, *e.g.* "a delete gesture" means an example gesture belonging to the class of delete gestures. Usually "gesture" refers to the part of the interaction up until the input is recognized as one

of a number of possible gesture classes, but sometimes the entire interaction (which includes a manipulation phase after recognition) is referred to as a gesture.

move The component of drag interaction during which the mouse is moved while a mouse button is held down. It is the presence of a move that distinguishes a click from a drag.

multi-path A multi-path gesture is one made with an input device that allows more than one position to be indicated simultaneously (multiple pointers). One may make multi-path gestures with a Sensor Frame, a multiple-finger touch pad, or a DataGlove, to name a few such devices.

off-line Considering an algorithm to be a sequence of operations, an off-line algorithm is one which examines subsequent operations before producing output for the current operation.

on-line An on-line algorithm is one in which the output of an operation is produced before any subsequent operations are read.

pick The initial part of a drag (or click) interaction consisting of positioning the mouse cursor at the desired location and pressing a mouse button.

press refers to the pressing of a mouse button.

real-time A real-time algorithm is an on-line algorithm in which each operation is processed in time bounded by a constant.

release refers to the releasing of a mouse button.

segment A segment is an approximately linear portion of a stroke. For example, the letter "L" is two segments, one vertical and one horizontal.

single-path A single-path gesture is one drawn by an input device, such as a mouse or stylus, capable of specifying only a single point over time. A single-path gesture may consist of multiple strokes (like the character "X").

single-stroke A single-stroke gesture is a single-path gesture that is one stroke. Thus drawing "L" is a single-stroke gesture, while "X" is not. In this dissertation the only single-path gestures considered are single-stroke gestures.

stroke A stroke is an unbroken curve made by a single movement of a pen, stylus, mouse, or other instrument. Generally, strokes begin and end with explicit user actions (*e.g.* , pen down/pen up, mouse button down/mouse button up).

1.3 Summary of Contributions

This dissertation makes contributions in four areas: new interaction techniques, new algorithms for gesture recognition, a new way of integrating gestures into user interfaces, and a new architecture for input in object-oriented toolkits.

The first new interaction technique is the two-phase combination of single-stroke gesture collection followed by direct manipulation, mentioned previously. In the GDP example discussed above, the boundary between the two phases is an interval of motionlessness. Eager recognition, the second new interaction technique, eliminates this interval by recognizing the single-stroke gesture and entering the manipulation phase as soon as enough of the gesture has been seen to do so unambiguously, making the entire interaction very smooth. A third new interaction technique is the two-phase interaction applied to multi-path gestures: after a multi-path gesture has been recognized, individual paths (*i.e.* fingers, possibly including additional fingers not involved in making the recognized gesture) may be assigned to manipulate independent application parameters simultaneously.

The second contribution is a new trainable, single-stroke recognition algorithm tailored for recognizing gestures. The classification is based on meaningful features, which in addition to being useful for recognition are also suitable for passing to application routines. The particular set of features used has been shown to be suitable for many different gesture sets, and is easily extensible. When restricted to features that can be updated incrementally in constant time per input point, arbitrarily large gestures may be handled. The single-stroke recognition algorithm has been extended to do eager recognition (eager recognizers are automatically generated from example gestures), and also to multi-path gesture recognition.

Third, a new paradigm for creating gestural interfaces is also propounded. As seen in the example, starting from a click-and-drag implementation of an interface, gestures are associated with classes of views (display objects), with the set of gestures recognized at a particular screen location dynamically determined by the set of overlapping views at the location, and by inheritance up the class hierarchy of each such view. The classification and attributes of gestures map directly to application operations and parameters. The creation, deletion, and manipulation of gesture handlers, gesture classes, gesture examples, and gesture semantics all occur at runtime, enabling quick and easy experimentation with gestural interfaces.

Fourth, GRANDMA, as an object-oriented user interface toolkit, makes some contributions to the area of input handling. Event handler objects are associated with particular views or entire view classes. A single event handler may be shared between many different objects, eliminating a major overhead of MVC systems. Multiple event handlers may be associated with a single object, enabling the object to support multiple interaction techniques simultaneously, including the use of multiple input devices. Furthermore, a single mechanism handles both mouse tools (*e.g.* a delete cursor that deletes clicked-upon objects) and virtual tools (*e.g.* a delete icon that is dragged around and dropped upon objects to delete them). Additionally, GRANDMA provides support for semantic feedback, and enables the runtime creation and manipulation of event handlers.

1.4 Motivation for Gestures

At this point, the reader should have a good idea of the scope of the work to be presented in this dissertation. Stepping back, this section begins a general discussion of gestures by examining the motivation for using and studying gesture-based interfaces. Much of the discussion is based on that of Buxton [14].

Computers get faster, bitmapped displays produce ever increasing information rates, speech and

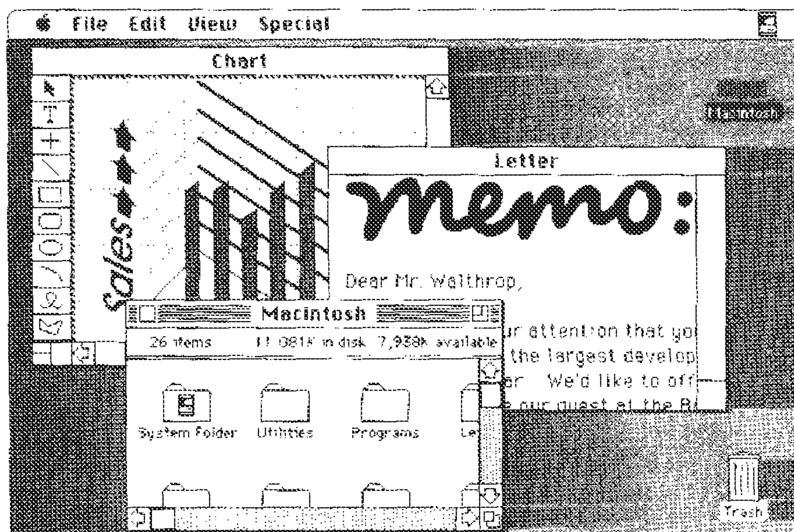


Figure 1.6: Macintosh Finder, MacDraw, and MacWrite (from Apple [2])

music can be generated in real-time, yet input just seems to plod along with little or no improvement. This is regrettable because, in Paul McAvinney's words [84], most of the useful information in the world resides in humans, not computers. Most people who interact with computers spend most of their time entering information [22]. Due to this input bottleneck, the total time to do many tasks would hardly improve even if computers became infinitely fast. Thus, improvements in input technology are a major factor in improving the productivity of computer users in general.

Of course, progress has been made. Input has progressed from batch data entry, to interactive line editors, to two-dimensional screen editors, to mouse-based systems with bitmapped displays. Pointing with a mouse has proved a useful interaction technique in many applications. "Click and drag" interfaces, where the user directly manipulates graphic objects on the screen with a mouse, are often very intuitive to use. Because of this, direct manipulation interfaces have become commonplace, despite being rather difficult to build.

Consider the Macintosh [2], generally regarded as having a good direct-manipulation interface. As shown in figure 1.6, the screen has on it a number of graphic objects, including file icons, folder icons, sliders, buttons, and pull-down menu names. Each one is generally a rectangular region, which may be clicked, sometimes double-clicked, and sometimes dragged. The Macintosh Finder, which may be used to access all Macintosh applications and documents, is almost entirely controlled via these three interaction techniques.[4]

The click and double-click interactions have a single object (or location) as parameter. The drag

[4]Obviously this discussion ignores keyboard entry of text and commands.

interaction has two parameters: an object or location where the mouse button is first pressed, and another object or location at the release point. Having only these three interaction techniques is one reason the Macintosh is simple to operate. There is, however, a cost: both the application and the user must express all operations in terms of these three interaction techniques.

An application that provides more than three operations on any given object (as many do) has several design alternatives. The first, exemplified by the Finder, relies heavily on *selection*. In the Finder, a click interaction selects an object, a double-click opens an object (the meaning of which depends upon the object's type), and a drag moves an object (the meaning of which is also object-type specific). Opening an object by a double-click is a means for invoking the most common operation on the object, *e.g.* opening a MacWrite document starts the MacWrite application on the document. Dragging is used for adjusting sliders (such as those which scroll windows), changing window size or position, moving files between folders, and selecting menu items.

All other operations are done in at least two steps: first the object to be operated upon is selected, and then the desired operation is chosen from a menu. For example, to print an object, one selects it (click) then chooses "Print" from the appropriate menu (drag); to move some text, one selects it (drag), chooses "Cut" (drag), selects an insertion point (click), and chooses "Paste" (drag). The cost of only having three interaction techniques is that some operations are necessarily performed via a sequence of interactions. The user must adjust her mental model so that she thinks in terms of the component operations.

An alternative to the selection-based click-and-drag approach is one based on modes. Consider MacDraw [2], a drawing program. The user is presented with a palette offering choices such as line, text, rectangles, circles, and so on. Clicking on the "line" icon puts the program into line-drawing mode. The next drag operation in the drawing window cause lines to be drawn. In MacDraw, after the drag operation the program reverts back to selection mode. DP, the program upon which GDP is based, is similar except that it remains in its current mode until it is explicitly changed. Mistakes occur when the user believes he is in one mode but is actually in another. The claim that direct manipulation interfaces derive their power from being modeless is not really true. Good direct manipulation interfaces simply make the modes very visible, which helps to alleviate the problems of modal interfaces.

By mandating the sole use of click, double-click, and drag interactions, the Macintosh interface paradigm necessarily causes conceptually primitive tasks to be divided into a sequence of primitive interactions. The intent of gestural interfaces is to avoid this division, by packing the basic interaction with all the parameters necessary to complete the entire transaction. Ideally, each primitive task in the user's model of the application is executed with a single gesture. Such interfaces would have less modeness than the current so-called modeless interfaces.

The Macintosh discussion in the previous section is somewhat oversimplified. Many applications allow variations on the basic interaction techniques; for example "shift-click" (holding the shift key while clicking the mouse) adds an object to the current set of selected objects. Other computer systems allow different mouse buttons to indicated different operations. There is a tradeoff between having a small number and a large number of (consistently applied) interaction techniques. The former results in a system whose primitive operations are easy to learn, perform, and recall, but a single natural chunk may be divided into a sequence of operations. In the latter case, the primitive

operations are harder to learn (because there are more of them), but each one can potentially implement an entire natural chunk.

The motivation for gestural interfaces may also apply to interfaces which combine modalities (*e.g.* speech and pointing). As with gestures, one potential benefit of multi-modal interfaces is that different modalities allow many parameters to be specified simultaneously, thus eliminating the need for modes. The "Put-That-There" system is one example [12].

1.5 Primitive Interactions

The discussion thus far has been vague as to what exactly may be considered a "primitive" interaction technique. The Macintosh has three: click, double-click, and drag. It is interesting to ask what criteria can be used for judging the "primitiveness" of proposed interaction techniques.

Buxton [14] suggests physical tension as a criterion. The user, starting from a relaxed state, begins a primitive interaction by tensing some muscles. The interaction is over when the user again relaxes those muscles. Buxton cites evidence that "such periods of tension are accompanied by a heightened state of attentiveness and improved performance." The three Macintosh interaction techniques all satisfy this concept of primitive interaction. (Presumably the user remains tense during a double-click because the time between clicks is short.)

Buxton likens the primitive interaction to a musical phrase. Each consists of a period of tension followed by a return to a state where a new phrase may be introduced. In human-computer interaction, such a phrase is used to accomplish a chunk of a task. The goal is to make each of these chunks a primitive task in the user's model of the application domain. This is what a gesture-based interface attempts to do.

1.6 The Anatomy of a Gesture

In this section a technical definition of gesture is developed, and the syntactic and semantic properties of gestures are then discussed. The dictionary definition of gesture, "expressive motion," has already been seen. How can the notion of gesture in a form suitable for sensing and processing by machine be captured?

1.6.1 Gestural motion

The motion aspect of gesture is formalized as follows: a gesture consists of the paths of multiple points over time. The points in question are (conceptually) affixed to the parts of the body which perform the gesture. For hand gestures, the points tracked might include the fingertips, knuckles, palm, and wrist of each hand. Over the course of a gesture, each point traces a path in space. Assuming enough points (attached to the body in appropriate places), these paths contain the essence of the gestural motion. A computer with appropriate hardware can rapidly sample positions along the paths, thus conveniently capturing the gesture.

The idea of gesture as the motion of multiple points over time is a generalization of pointing. Pointing may be considered the simplest gesture: it specifies a single position at an instance of

time. This is generalized to allow for the movement of the point over time, *i.e.* a path. A further generalization admits multiple paths, *i.e.* the movement of multiple points over time.[5]

Current gesture-sensing hardware limits both the number of points which may be tracked simultaneously and the dimensionality of the space in which the points travel. Gestures limited to the motion of a single point are referred to here as *single-path* gestures. Most previous gestural research has focused upon gestures made with a stylus and tablet, mouse, or single-finger touch pad. The gestures which may be made with such devices are two-dimensional, single-path gestures.

An additional feature of existing hardware is that the points are not tracked at all times. For example, a touch pad can only determine finger position when the finger is touching the pad. Thus, the path of the point will have a beginning (when the finger first makes contact) and an end (when the finger is lifted). This apparent limitation of certain gesture-sensing hardware may be used to delineate the start and possibly the end of each gesture, a necessary function in gesture-based systems. Mouse buttons may be used to similar effect.[6]

In all the work reported here, a gesture (including the manipulation phase after recognition) is always a primitive interaction. A gesture begins with the user going from a relaxed state to one of muscular tension, and ends when the user again relaxes. It is further assumed that the tension or relaxation of the user is directly indicated by some aspect of the sensing hardware. For mouse gestures, the user is considered in a state of tension if and only if a mouse button is pressed. Thus, in the current work a double-click is not considered a gesture. This is certainly a limitation, but one that could be removed, for example by having a minimum time that the button needs to be released before the user is considered to have relaxed. This added complication has not been explored here.

The space in which the points of the gesture move is typically physical space, and thus a path is represented by a set of points (x, y, z, t) consisting of three spatial Cartesian coordinates and time. However, there are devices which measure non-spatial gestural parameters; hence, gestures consisting of paths through a space where at least some of the coordinates are not lengths are possible. For example, some touch pads can sense force, and for this hardware a gesture path might consist of a set of points (x, y, f, t), f being the force measurement at time t.

The formalization of gesture as multiple paths is just one among many possible representations. It is a good representation because it coincides nicely with most of the existing gesture-sensing hardware, and it is a useful form for efficient processing. The multiple-snapshot representation, in which each snapshot gives the position of multiple points at a single instant, is another possibility, and in some sense may be considered the dual of multiple paths. Such a representation might be more suitable for gestural data derived from hardware (such as video cameras) which are not considered in this dissertation.

1.6.2 Gestural meaning

In addition to the physical aspect of a gesture, there is the content or meaning of the gesture to consider. Generally speaking, a gesture contains two kinds of information: *categorical* and

[5]A configuration of multiple points at a single instance of time may be termed *posture*. Posture recognition is commonly used with the DataGlove.

[6]Buxton [17] presents a model of the discrete signaling capabilities of various pointing devices and a list of the signaling requirements for common interaction techniques.

parametric. Consider the different motions between people meaning "come here" (beckoning gestures), "stop" (prohibiting gestures), and "keep going" (encouragement gestures). These are different categories, or *classes*, of gestures. Within each class, a gesture also can indicate parametric data. For example, a parameter of the beckoning gesture is the urgency of the request: "hurry up" or "take your time." In general, the category of the gesture must be determined before the parameters can be interpreted.

Parametric information itself comes in two forms. The first is the kind of information that can be culled at the time the gesture is classified. For example, the position, size and orientation of the gesture fall into this category. The second kind of parametric gestural information is manipulation information. After the gesture is recognized, the user can use this kind of parametric information to continuously communicate information. An example would be the directional information communicated by the gestures of a person helping a driver to back up a truck. An example from GDP (see Section 1.1) is the rubberbanding of a line after it is created, where the user continuously manipulates one endpoint.

The term "gesture" as used here does not exactly correspond to what is normally thought of as gesture. Many gestures cannot currently be processed by machine due to limitations of existing gesture-sensing hardware. Also, consider what might be referred to as "direct-manipulation gestures." A person turning a knob would not normally be considered to be gesturing. However, a similar motion used to manipulate the graphic image of a knob drawn on a computer display is considered to be a gesture. Actually, the difference here is more illusory than real: a person might make the knob-turning gesture at another person, in effect asking the latter to turn the knob. The intent here is simply to point out the very broad class of motions considered herein to be gesture.

While the notion of gesture developed here is very general (multiple paths), in practice, machine gestures have hitherto almost always been limited to finger and/or hand motions. Furthermore, the paths have largely been restricted to two dimensions. The concentration on two-dimensional hand gesturing is a result of the available gesture-sensing hardware. Of course, such hardware was built because it was believed that hand and fingers are capable of accurate and diverse gesturing, yet more amenable to practical detection than facial or other body motions. With the appearance of new input devices, three (or more) dimensional gesturing, as well as the use of parts of the body other than the hand, are becoming possible. Nonetheless, this dissertation concentrates largely on two-dimensional hand gestures, assuming that by viewing gesture simply as multiple paths, the work described may be applied to non-hand gestures, or generalized to apply to gestures in three or more dimensions.

1.7 Gesture-based systems

A gesture-based interface, as the term is used here, is one in which the user specifies commands by gesturing. Typically, gesturing consists of drawing or other freehand motions. Excluded from the class of gesture-based interfaces are those in which input is done solely via keyboard, menu, or click-and-drag interactions. In other words, while pointing is in some sense the most basic gesture, those interfaces in which pointing is the only form of gesture are not considered here to be gesture-based interfaces. A gesture-based system is a program (or set of programs) with which the user interacts via a gesture-based interface.

In all but the simplest gesture-based systems, the user may enter a gesture belonging to one of several different gesture categories or classes; the different classes refer to different commands to the system. An important component of gesture-based systems is the gesture *recognizer* or *classifier*, the module whose job is to classify the user's gesture as the first step toward inferring its meaning. This dissertation addresses the implementation of gesture recognizers, and their incorporation into gesture-based systems.

1.7.1 The four states of interaction

User interaction with the gesture-based systems considered in this dissertation may be described using the following four state model. The states--WAIT, COLLECT, MANIPULATE, EXECUTE--usually occur in sequence for each interaction.

- The WAIT state is the quiescent state of the system. The system is waiting for the user to initiate a gesture.
- The COLLECT state is entered when the user begins to gesture. While in this state, the system collects gestural data from the input hardware in anticipation of classifying the gesture. For most gesturing hardware, an explicit start action (such as pressing a mouse button) indicates the beginning of each gesture, and thus causes the system to enter this state.
- The MANIPULATE state is entered once the gesture is classified. This occurs in one of three ways:

  1. The end of the gesture is indicated explicitly, *e.g.* by releasing the mouse button;
  2. the end of the gesture is indicated implicitly, *e.g.* by a timeout which indicates the user has not moved the mouse for, say, 200 milliseconds; or
  3. the system initiates classification because it believes it has now seen enough of the gesture to classify it unambiguously (eager recognition).

  When the MANIPULATE state is entered, the system should provide feedback to the user as to the classification of the gesture and update any screen objects accordingly. While in this state, the user can further manipulate the screen objects with his motions.

- The EXECUTE state is entered when the user has completed his role in the interaction, and has indicated such (*e.g.* by releasing the mouse button). At this point the system performs any final actions as implied by the user's gesture. Ideally, this state lasts only a very short time, after which the display is updated to reflect the current state of the system, and the system reverts back to the WAIT state.

This model is sufficient to describe most current systems which use pointing devices. (For simplicity, keyboard input is ignored.) Depending on the system, the COLLECT or MANIPULATE state may be omitted from the cycle. A handwriting interface will usually omit the MANIPULATE state, classifying the collected characters and executing the resulting command. Conversely, a

direct-manipulation system will omit the COLLECT state (and the attendant classification). The GDP example described above has both COLLECT and MANIPULATE phases. The result is the new two-phase interaction technique mentioned earlier.

1.8 A Comparison with Handwriting Systems

In this section, the frequently asked question, "how do gesture-based systems differ from handwriting systems?" is addressed.

Handwriting systems may broadly be grouped into two classes: on-line and off-line. On-line handwriting recognition simply means characters are recognized as they are drawn. Usually, the characters are drawn with a stylus on a tablet, thus the recognition process takes as input a list of successive points or line segments. The problem is thus considerably different than off-line handwriting recognition, in which the characters are first drawn on paper, and then optically scanned and represented as two-dimensional rasters. Suen, Berthod, and Mori review the literature of both on-line and off-line handwriting systems [125], while Tappert, Suen, and Wakaha [129] give a recent review of on-line handwriting systems. The intention here is to contrast gesture-based systems with on-line handwriting recognition systems, as these are the most closely related.

Gesture-based systems have much in common with systems which employ on-line handwriting recognition for input. Both use freehand drawing as the primary means of user input, and both depend on recognizers to interpret that input. However, there are some important differences between the two classes of systems, differences that illustrate the merits of gesture-based systems:

- Gestures may be motions in two, three, or more dimensions, whereas handwriting systems are necessarily two-dimensional. Similarly, single-path and multiple-path gestures are both possible, whereas handwriting is always a single path.

- The alphabet used in a handwriting system is generally well-known and fixed, and users will generally have lifelong experience writing that alphabet. With gestures, it is less likely that users will have preconceptions or extensive experience.

- In addition to the command itself, a single gesture can specify parameters to the command. The proofreader's gesture (figure 1.1) discussed above, is an excellent example. Another example, also due to Buxton [21], and used in GSCORE (Section 8.2), is a musical score editor, in which a single stroke indicates the location, pitch, and duration of a note to be added to the score.

- As stated, a command and all its parameters may be specified with a single gesture. The physical relaxation of the user when she completes a gesture reinforces the conceptual completion of a command [14].

- Gestures of a given class may vary in both size and orientation. Typical handwriting recognizers expect the characters to be of a particular size and oriented in the usual manner (though successful systems will necessarily be able to cope with at least small variations in size and orientation). However, some gesture commands may use the size and orientation to specify

parameters; gesture recognizers must be able to recognize such gestures in whatever size and orientation they occur. Kim [67] discusses augmenting a handwriting recognition system so as to allow it to recognize some gestures independently of their size and orientation. Chapter 3 discusses the approach taken here toward the same end.

- Gestures can have a dynamic component. Handwriting systems usually view the input character as a static picture. In a gesture-based system, the same stroke may have different meanings if drawn left-to-right, right-to-left, quickly, or slowly. Gesture recognizers may use such directional and temporal information in the recognition process.

In summary, gestures may potentially deal in dimensions other than the two commonly used in handwriting, be drawn from unusual alphabets, specify entire commands, vary in size and orientation, and have a dynamic component. Thus, while ideas from on-line handwriting recognition algorithms may be used for gesture recognition, handwriting recognizers generally rely on assumptions that make them inadequate for gesture recognition. The ideal gesture recognition algorithm should be adaptable to new gestures, dimensions, additional features, and variations in size and orientation, and should produce parametric information in addition to a classification. Unfortunately, the price for this generality is the likelyhood that a gesture recognizer, when used for handwriting recognition, will be less accurate than a recognizer built and tuned specifically for handwriting recognition.

1.9 Motivation for this Research

In spite of the potential advantages of gesture-based systems, only a handful have been built. Examples include Button Box [86], editing using proofreader's symbols [25], the Char-rec note-input tool [21], and a spreadsheet application built at IBM [109]. These and other gesture-based systems are discussed in section 2.2. Gesture recognition in most existing systems has been done by writing code to recognize the particular set of gestures used by the system. This code is usually complicated, making the systems (and the set of gestures accepted) difficult to create, maintain, and modify. These difficulties are the reasons more gesture-based systems have not been built.

One goal of the present work is to eliminate hand-coding as the way to create gesture recognizers. Instead, gesture classes are specified by giving examples of gestures in the class. From these examples, recognizers are automatically constructed. If a particular gesture class is to be recognized in any size or orientation, its examples of the class should reflect that. Similarly, by making all of the examples of a given class the same size or orientation, the system learns that gestures in this class must appear in the same size or orientation as the examples. The first half of this dissertation is concerned with the automatic construction of gesture recognizers.

Even given gesture recognition, it is still difficult to build direct-manipulation systems which incorporate gestures. This is the motivation for the second half of this dissertation, which describes GRANDMA–Gesture Recognizers Automated in a Novel Direct Manipulation Architecture.

1.10 Criteria for Gesture-based Systems

The goal of this research was to produce tools which aid in the construction of gesture-based systems. The efficacy of the tools may be judged by how well the tools and resulting gesture-based systems satisfy the following criteria.

1.10.1 Meaningful gestures must be specifiable

A meaningful gesture may be rather complex, involving simultaneous motions of a number of points. These complex gestures must be easily specifiable. Two methods of specification are possible: specification by example, and specification by description. In the former, each application has a training session in which examples of the different gestures are submitted to the system. The result of the training is a representation for all gestures that the system must recognize, and this representation is used to drive the actual gesture recognizer that will run as part of the application. In the latter method of specification, a description of each gesture is written in a gesture description language, which is a formal language in which the "syntax" of each gesture is specified. For example, a set of gestures may be specified by a context-free grammar, in which the terminals represent primitive motions (e.g. "straight line segment") and gestures are non-terminals composed of terminals and other non-terminals.

All else being equal, the author considers specification by example to be superior to specification by description. In order to specify gestures by description, it will be necessary for the specifier to learn a description language. Conversely, in order to specify by example, the specifier need only be able to gesture. Given a system in which gestures are specified by example, the possibility arises for end users to train the system directly, either to replace the existing gestures with ones more to their liking, or to have the system improve its recognition accuracy by adapting to the particular idiosyncrasies of a given user's gestures.

One potential drawback of specification by example is the difficulty in specifying the allowable variations between gestures of a given class. In a description language, it can be made straightforward to declare that gestures of a given class may be of any size or of any orientation. The same information might be conveyed to a specify-by-example system by having multiple examples of a single class vary in size or orientation. The system would then have to *infer* that the size or orientation of a given gesture class was irrelevant to the classification of the gesture. Also, training classifiers may take longer, and recognition may be less accurate, when using examples as specifications, though this is by no means necessarily so. Similar issues arise in demonstrational interfaces [97].

1.10.2 Accurate recognition

An important characterization of a gesture recognition system will be the frequency with which gestures fail to be recognized or are recognized incorrectly. Obviously it is desirable that these numbers be made as small as possible. Questions pertaining to the amount of inaccuracy acceptable to people are difficult to answer objectively. There will likely be tradeoffs between the complexity of gestures, the number of different gestures to be disambiguated, the time needed for recognition, and the accuracy of recognition.

In speech recognition there is the problem that the accuracy of recognition decreases as the user population grows. However the analogous problem in gesture recognition is not as easy to gauge. Different people speak the same words differently due to inevitable differences in anatomy and upbringing. The way a person says a word is largely determined before she encounters a speech recognition system. By contrast, most people have few preconceptions of the way to gesture at a machine. People will most likely be able to adapt themselves to gesturing in ways the machine understands. The recognition system may similarly adapt to each user's gestures. It would be interesting, though outside the scope of this dissertation, to study the fraction of incorrectly recognized gestures as a function of a person's experience with the system.

1.10.3 Evaluation of accuracy

It should be possible for a gesture-based system to monitor its own performance with respect to accuracy of recognition. This is not necessarily easy, since in general it is impossible to know which gesture the user had intended to make. A good gesture-based system should incorporate some method by which the user can easily inform the system when a gesture has been classified incorrectly. Ideally, this method should be integrated with the undo or abort features of the systems. (Lerner [78] gives an alternative in which subsequent user actions are monitored to determine when the user is satisfied with the results of system heuristics.)

1.10.4 Efficient recognition

The goal of this work is to enable the construction of applications that use gestures as input, the idea being that gesture input will enhance human/computer interaction. Speed of recognition is very important–a slow system would be frustrating to use and hinder rather than enhance interaction

Speed is a very important factor in the success or failure of user interfaces in general. Baecker and Buxton [5] state that one of the chief determinants of user satisfaction with interactive computer systems is response time. Poor performance in a direct-manipulation system is particularly bad, as any noticeable delay destroys the feeling of directness. Rapid recognition is essential to the success of gesture as a medium for human-computer interaction, even if achieving it means sacrificing certain features or, perhaps, a limited amount of recognition accuracy.

1.10.5 On-line/real-time recognition

When possible, the recognition system should attempt to match partial inputs with possible gestures. It may also be desirable to inform the user as soon as possible when the input does not seem to match any possible gesture. An on-line/real-time matching algorithm has these desirable properties. The gesture recognition algorithms discussed in Chapters 3, 4, and 5 all do a small, bounded amount of work given each new input point, and are thus all on-line/real-time algorithms.

1.10.6 General quantitative application interface

An application must specify what happens when a gesture is recognized. This will often take the form of a callback to an application-specific routine. There is an opportunity here to relay the

parametric data contained in the gesture to the application. This includes the parametric data which can be derived when the gesture is first recognized, as well as any manipulation data which follows.

1.10.7 Immediate feedback

In certain applications, it is desirable that the application be informed immediately once a gesture is recognized but before it is completed. An example is the turning of a knob: once the system recognizes that the user is gesturing to turn a knob it can monitor the exact details of a gesture, relaying quantitative data to the application. The application can respond by immediately and continuously varying the parameter which the knob controls (for example the volume of a musical instrument).

1.10.8 Context restrictions

A gesture sensing system should be able, within a single application, to sense different sets of gestures in different contexts. An example of a context is a particular area of the display screen. Different areas could respond to different sets of gestures. The set of gestures to which the application responds should also be variable over time–the application program entering a new mode could potentially cause a different set of gestures to be sensed.

The idea of contexts is closely related to the idea of using gestures to manipulate graphic objects. Associated with each picture of an object on the screen will be an area of the screen within which gestures refer to the object. A good gesture recognition system should allow the application program to make this association explicit.

1.10.9 Efficient training

An ideal system would allow the user to experiment with different gesture classes, and also adapt to the user's gestures to improve recognition accuracy. It would be desirable if the system responded immediately to any changes in the gesture specifications; a system that took several hours to retrain itself would not be a good platform for experimentation.

1.10.10 Good handling of misclassifications

Misclassifications of gestures are a fact of life in gesture-based systems. A typical system might have a recognition rate of 95% or 99%. This means one out of twenty or one out of one hundred gestures will be misunderstood. A gesture-based system should be prepared to deal with the possibility of misclassification, typically by providing easy access to abort and undo facilities.

1.10.11 Device independence

Certain assumptions about the form of the input data are necessary if gesture systems are to be built. As previously stated, the assumption made here is that the input device will supply position as a function of time for each input "path" (or supply data from which it is convenient to calculate such positions). (A path may be thought of as a continuous curve drawn by a single finger.) This form of

data is supplied by the Sensor Frame, and (at least for the single finger case) a mouse and a clock can be made to supply similar data. The recognition systems should do their recognition based on the position versus time data; in this way other input devices may also benefit from this research.

1.10.12 Device utilization

Each particular brand of input hardware used for gesture sensing will have characteristics that other brands of hardware will not have. It would be unfortunate not to take advantage of all the special features of the hardware. For example, the Sensor Frame can compute finger angle and finger velocity.[7] While for device independence it may be desirable that the gesture matching not depend on the value of these inputs, there should be some facility for passing these parameters to the application specific code, if the application so desires. Baecker [4] states the case strongly: "Although portability is facilitated by device-independence, interactivity and usability are enhanced by *device dependence*."

1.11 Outline

The following chapter describes previous related work in gesture-based systems. This is divided into four sections: Section 2.1 discusses various hardware devices suitable for gestural input. Section 2.2 discusses existing gesture-based systems. Section 2.3 reviews the various approaches to pattern recognition in order to determine their potential for gesture recognition. Section 2.4 examines existing software systems and toolkits that are used to build direct-manipulation interfaces. Ideas from such systems will be generalized in order to incorporate gesture recognition into such systems.

Everything after Chapter 2 focuses on various aspects of the gesture-based interface creation tool built by the author. Such a tool makes it easy to 1) specify and create classifiers, and 2) associate gestures classes and their meanings with graphic objects. The former goal is addressed in Chapters 3, 4, and 5, the latter in 6 and 7.

The discussion of the implementation of gesture recognition begins in Chapter 3. Here the problem of classifying single-path, two-dimensional gestures is tackled. This chapter assumes that the start and end of the gesture are known, and uses statistical pattern recognition to derive efficient gesture classifiers. The training of such classifiers from example gestures is also covered.

Chapter 3 shows how to classify single-path gestures; Chapter 4 shows *when*. This chapter addresses the problem of recognizing gestures while they are being made, without any explicit indication of the end of the gesture. The approach taken is to define and construct another classifier. This classifier is intended solely to discriminate between ambiguous and unambiguous subgestures.

Chapter 5 extends the statistical approach to the recognition of multiple-path gestures. This is useful for utilizing devices that can sense the positions of multiple fingers simultaneously, in particular the Sensor Frame.

Chapter 6 presents the architecture of an object-oriented toolkit for the construction of direct-manipulation systems. Like many other systems, this architecture is based on the Model-View-

[7] This describes the Sensor Frame as originally envisioned. The hardware is capable of producing a few bits of finger velocity and angle information, although to date this has not been attempted.

Controller paradigm. Compared to previous toolkits, the input model is considerably generalized in preparation for the incorporation of gesture recognition into a direct-manipulation system. The notion of virtual tools, through which input may be generated by software objects in the same manner as by hardware input devices, is introduced. Semantic feedback will be shown to arise naturally from this approach.

Chapter 7 shows how gesture recognizers are incorporated into the direct-manipulation architecture presented in Chapter 6. A gesture handler may be associated with a particular view of an object on the screen, or at any level in the view hierarchy. In this manner, different objects will respond to different sets of gestures. The communication of parametric data from gesture handler to application is also examined.

Chapter 8 discusses three gesture-based systems built using these techniques: GDP, GSCORE, and MDP. The first two, GDP and GSCORE, use mouse gestures. GDP, as already mentioned, is the drawing editor based on DP. GSCORE is a musical score editor, based on Buxton's SSSP work [21]. MDP is also a drawing editor, but it operates using multi-path gestures made with a Sensor Frame. The design and implementation of each system is discussed, and the gestures for each shown.

Chapter 9 evaluates a number of aspects of this work. The particular recognition algorithms are tested for recognition accuracy. Measurements of the performance of the gesture classifiers used in the applications is presented. Then, an informal user study assessing the utility of gesture-based systems is discussed.

Finally, Chapter 10 concludes this dissertation. The contributions of this dissertation are discussed, as are the directions for future work.

1.12 What Is Not Covered

This dissertation attempts to cover many topics relevant to gesture-based systems, though by no means all of them. In particular, the issues involved in the ergonomics and suitability of gesture-based systems applied to various task domains have not been studied. It is the opinion of the author that such issues can only be studied after the tools have been made available which allow easy creation of and experimentation with such systems. The intent of the current work is to provide such tools. Future research is needed to determine how to use the tools to create the most usable gesture-based systems possible.

Of course, choices have had to be made in the implementation of such tools. By avoiding the problem of determining which kind of gesture-based systems are best, the work opens itself to charges of possibly "throwing the baby out with the bath-water." The claim is that the general system produced is capable of implementing systems comparable to many existing gesture-based systems; the example applications implemented (see Chapter 8) support this claim. Furthermore, the places where restrictive choices have been made (*e.g.* two-dimensional gestures) have been indicated, and extensible and scalable methods (*e.g.* linear discrimination) have been used wherever possible.

There are two major limitations of the current work. The first is that single-path multi-stroke gestures (*e.g.* handwritten characters) are not handled. Most existing gesture-based systems use single-path multi-stroke gestures. The second limitation is that the start of a gesture must be

explicitly indicated. This rules out (at least at first glance) using devices such as the DataGlove which lack buttons or other explicit signaling hardware. However, one result of the current work is that these apparent limitations give rise to certain advantages in gestural interfaces. For example, the limitations enforce Buxton's notion of tension and release mentioned above.

Gestural output, *i.e.* generating a gesture in response to a query, is also not covered. For an example of gestural output, ask the author why he has taken so long to complete this dissertation.

Chapter 2

Related Work

This chapter discusses previous work relevant to gesture recognition. This includes hardware devices suitable for gestural input, existing gesture-based systems, pattern recognition techniques, and software systems for building user interfaces.

Before delving into details, it is worth mentioning some general work that attempts to define gesture as a technique for interacting with computers. Morrel-Samuels [87] examines the distinction between gestural and lexical commands, and then further discusses problems and advantages of gestural commands. Wolf and Rhyne [140] integrate gesture into a general taxonomy of direct manipulation interactions. Rhyne and Wolf [109] discuss in general terms human-factors concerns of gestural interfaces, as well as hardware and software issues.

The use of gesture as an interaction technique is justified in a number of studies. Wolf [139] performed two experiments that showed gestural interfaces compare favorably to keyboard interfaces. Wolf [141] showed that many different people naturally use the same gestures in a text-editing context. Hauptmann [49] demonstrated a similar result for an image manipulation task, further showing that people prefer to combine gesture and speech rather than use either modality alone.

2.1 Input Devices

A number of input devices are suitable for providing input to a gesture recognizer. This section concentrates on those devices which provide the position of one or more points over time, or whose data is easily converted into that representation. The intention is to list the types of devices which can potentially be used for gesturing. The techniques developed in this dissertation can be applied, directly or with some generalization, to the devices mentioned.

A large variety of devices may be used as two-dimensional, single-path gesturing devices. Some graphical input devices, such as mice [33], tablets and styli, light pens, joysticks, trackballs, touch tablets, thumb-wheels, and single-finger touch screens [107, 124], have been in common use for years. Less common are foot controllers, knee controllers, eye trackers [12], and tongue-activated joysticks. Each may potentially be used for gestural input, though ergonomically some are better suited for gesturing than others. Baecker and Buxton [5], Buxton [14], and Buxton, Hill and Rowley [18] discuss the suitability of many of the above devices for various tasks. Buxton further points out

that two different joysticks, for example, may have very different properties that must be considered with respect to the task.

For gesturing, as with pointing, it is useful for a device to have some signaling capability in addition to the pointer. For example, a mouse usually has one or more buttons, the pressing of which can be used to indicate the start of a gesture. Similarly, tablets usually indicate when the stylus makes or breaks contact with the tablet (though with a tablet it is not possible to carefully position the screen cursor before contact). If a device does not have this signaling capacity, it will be necessary to simulate it somehow. Exactly how this is done can have a large impact on whether or not the device will be suitable for gesturing.

The 3SPACE Isotrack system, developed by Polhemus Navigation Sciences Division of McDonnel Douglas Electronics Company [32], is a device which measures the position and orientation of a stylus or a one-inch cube using magnetic fields. The Polhemus sensor, as it is often called, is a full six-degree-of-freedom sensor, returning x, y, and z rectangular coordinates, as well as azimuth, altitude, and roll angles. It is potentially useful for single path gesturing in three positional dimensions. By considering the angular dimensions, 4, 5, or 6 dimensional gestures may be entered. It is also possible to use one of the angular dimensions for signaling purposes.

Bell Laboratories has produced prototypes of a clear plate capable of detecting the position and pressure of many fingers [10, 99]. The position information is two-dimensional, and there is a third dimension as well: finger pressure. The author has seen the device reliably track 10 fingers simultaneously. The pressure detection may be used for signaling purposes, or as a third dimension for gesturing. The inventor of the multi-finger touch plate has invented another device, the Radio Drum [11], which can sense the position of multiple antennae in three dimensions. To date, the antennae have been embedded in the tips of drum sticks (thus the name), but it would also be possible to make a glove containing the antenna which would make the device more suitable for detecting hand gestures.

The Sensor Frame [84] is a frame mounted on a workstation screen (figure 2.1). It consists of a light source (which frames the screen) and four optical sensors (one in each corner). The Sensor Frame computes the two-dimensional positions of up to three fingertips in a plane parallel to, and slightly above the screen. The net result is similar to a multi-finger touch screen. The author has used the Sensor Frame to verify the multi-finger recognition algorithm described in Chapter 5. The Sensor Cube [85] is a device similar to the Sensor Frame but capable of sensing finger positions in three dimensions. It is currently under construction. The VideoHarp [112, 111] is a musical instrument based on the same sensing technology, and is designed to capture parametric gestural data.

The DataGlove [32, 130] is a glove worn on the hand able to produce the positions of multiple fingers as well as other points on the hand in three dimensions. By itself it can only output relative positions. However, in combination with the Polhemus sensor, absolute finger positions can be computed. Such a device can translate gestures as complex as American Sign Language [123] into a multi-path form suitable for processing. The DataGlove, the similar Dexterous Hand Master from Exos, and the Power Glove from Mattel, are shown in figure 2.2.

The DataGlove comes with hardware which may be trained to recognize certain static configurations of the glove. For example, the DataGlove hardware might be trained to recognize a fist,

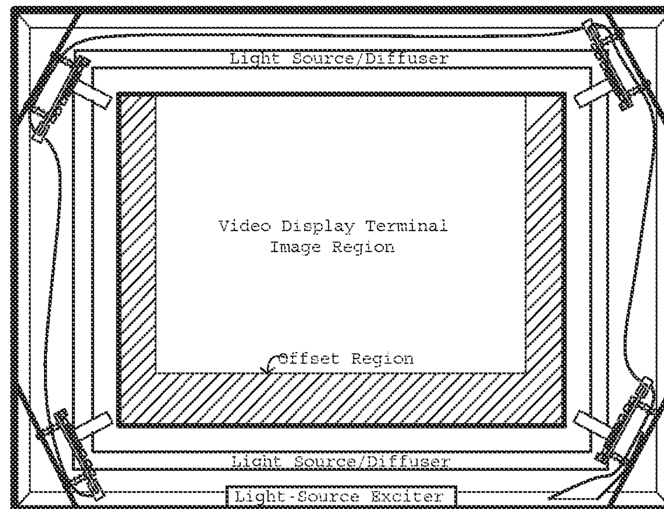


Figure 2.1: The Sensor Frame
The Sensor Frame is a frame mounted on a computer display consisting of a rectangular light source and four sensors, one in each corner. It is capable of detecting up to three fingers its field of view. (Drawing by Paul McAvinney.)

Figure 2.2: The DataGlove, Dexterous Hand Master, and PowerGlove (from Eglowstein [32])
The DataGlove, Dexterous Hand Master, and PowerGlove are three glove-like input devices capable of measuring the angles of various hand and finger joints.

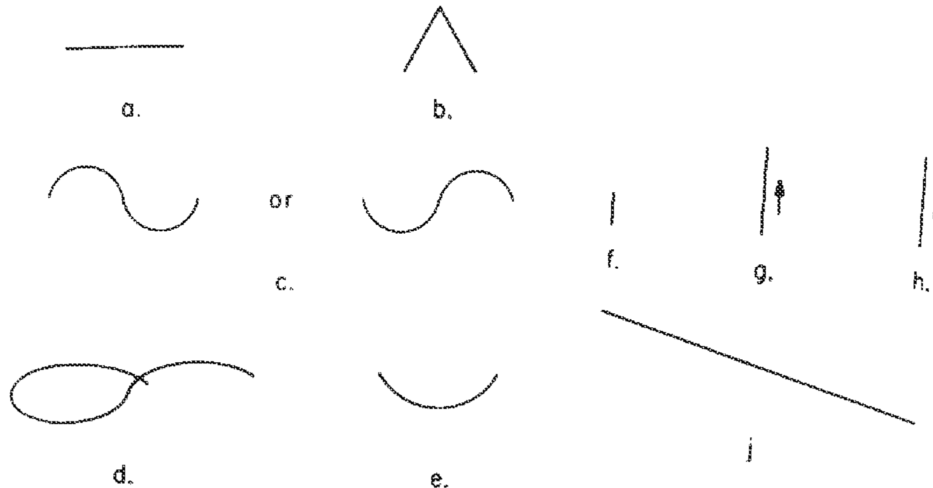


Figure 2.3: Proofreading symbols (from Coleman [25])
The operations intended by each are as follows: a) delete text (from a single line), b) insert text, c) swap text, d) move text, e) join (delete space), f) insert space, g) scroll up, h) scroll down, and i) delete multiple lines of text. Many of the marks convey additional parameters to the operation, e.g. the text to be moved or deleted.

signaling the host computer whenever a fist is made. These static hand positions are not considered to be gestures, since they do not involve motion. The glove hardware recognizes "posture" rather than gesture, the distinction being that posture is a static snapshot (a pose), while gesture involves motion over time. Nonetheless, it is a rather elegant way to add signaling capability to a device without buttons or switches.

The Videodesk [71, 72] is an input device based on a constrained form of video input. The Videodesk consists of a translucent tablecloth over a glass top. Under the desk is a light source, over the desk a video camera. The user's hands are placed over the desk. The tablecloth diffuses the light, the net effect being that the camera receives an image of the silhouette of the hands. Additional hardware is used to detect and track the user's fingertips.

Some researchers have investigated the attachment of point light sources to various points on the body or hand to get position information as a function of time. The output of a camera (or pair of cameras for three dimensional input) can be used as input to a gesture sensor.

2.2 Example Gesture-based Systems

This section describes a number of existing gesture-based systems that have been described in the literature. A system must both classify its gestural input and use information other than the class (*i.e.* parametric information) to be included in this survey. The order is roughly chronological.

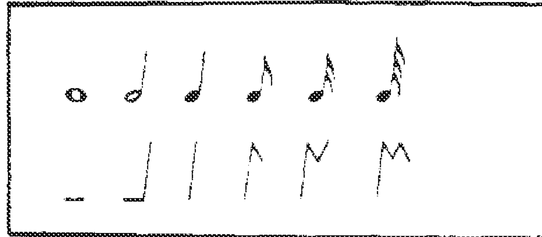


Figure 2.4: Note gestures (from Buxton [21])
A single gesture indicates note duration (from the shape of the stroke as shown) as well as pitch and starting time, both of which are determined from the position of the start of the gesture.

Coleman [25] has created a text editor which used hand-drawn proofreader's symbols to specify editing commands (figure 2.3). For example, a sideways "S" indicated that two sets of characters should be interchanged, the characters themselves being delimited by the two halves of the "S." The input device was a touch tablet, and the gesture classification was done by a hand-coded discrimination net (*i.e.* a loop-free flowchart).[1]

Buxton [21] has built a musical score editor with a small amount of gesture input using a mouse (figure 2.4). His system used simple gestures to indicate note durations and scoping operations. Buxton considered this system to be more a character recognition system than a gesture-based system, the characters being taken from an alphabet of musical symbols. Since information was derived not only from the classification of the characters, but their positions as well, the author considers this to be a gesture-based system in the true sense. Buxton's technique was later incorporated into Notewriter II, a commercial music scoring program. Lamb and Buckley [76] describe a gesture-based music editor usable by children.

Margaret Minsky [86] implemented a system called Button Box, which uses gestures for selection, movement, and path specification to provide a complete Logo programming environment (figure 2.5). Her input device was a clear plate mounted in front of a display. The device sensed the position and shear forces of a single finger touching the plate. Minksy proposed the use of multiple fingers for gesture input, but never experimented with an actual multiple-finger input device.

In Minsky's system, buttons for each Logo operation were displayed on the screen. Tapping a button caused it to execute; touching a button and dragging it caused it to be moved. The classification needed to distinguish between a touch and a tap was programmed by hand. There were buttons used for copying other buttons and for grouping sets of buttons together. A path could be drawn through a series of buttons--touching the end of a path caused its constituent buttons to execute sequentially.

VIDEOPLACE [72] is a system based on the Videodesk. As stated above, the silhouette of the user's hands are monitored. When a hand is placed in a pointing posture, the tip of the index finger

[1]Curiously, this research was done while Coleman was a graduate student at Carnegie Mellon. Coleman apparently never received a Ph.D. from CMU, and it would be twenty years before another CMU graduate student (me) would go near the topic of gesture recognition.

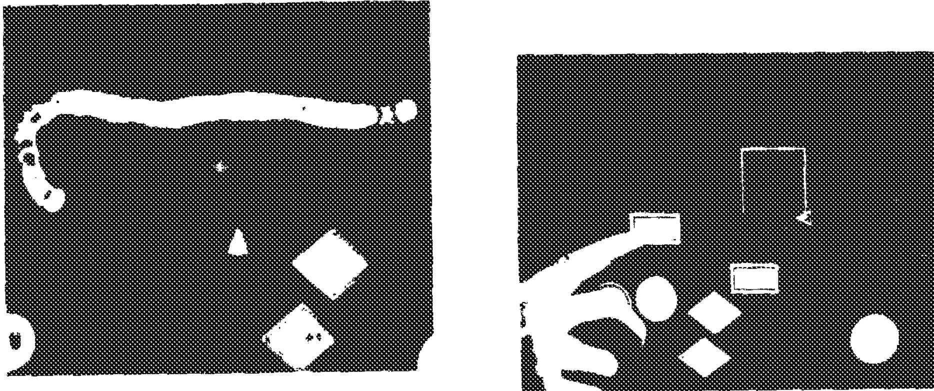


Figure 2.5: Button Box (from Minksy [86])

Tapping a displayed button causes it to execute its assigned function while touching a button and dragging it causes it to be moved.

	A	B	C	D	E
1		1Q86	1Q86		
2		Proj	Actual		
3		----------	----------		
4					
5	Noreast	$1,200	$1,152		96.00%
6	MidWest	$600	$541		90.17%
7	South	$850	$925		108.82%
8	SouthWest	$800	$781		97.63%
9	West	$1,000	$878		87.80%
10	Americas	$300	$221		73.67%
11	Europe	$500	$557		111.40%
12					
13					
14	TOTALS	$5,250	$4,855		92.48%
15					
16					
17					
18					

>G5

Figure 2.6: A gesture-based spreadsheet (from Rhyne and Wolf [109])

The Paper-Like Interface project produces systems which combine gesture and handwriting. The input shown here selects a group of cells and requests they be moved to the cell beginning at location "G5."

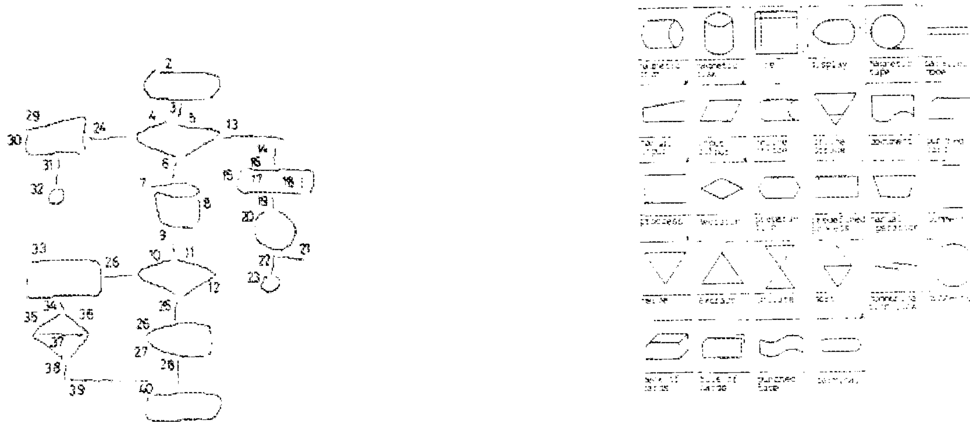


Figure 2.7: Recognizing flowchart symbols

Recognizing flowchart symbols (from Murase and Wakahara [89]). The system takes an entire freehand drawing of a flowchart (left) and recognizes the individual flowchart symbols (right), producing an internal representation of the flowchart (as nodes and edges) and a flowchart picture in which the freehand symbols are replaced by machine generated line-drawings drawn from the alphabet of symbols. This system shows a style of interface in which pattern recognition is used for something other than the detection of gestures or characters.

may be used for menu selection. After selection, the fingertips may be used to manipulated graphic objects, such as the controlling points of a spline curve.

A group at IBM doing research into gestural human-computer systems has produced a gesture-based spreadsheet application [109]. Somewhat similar to Coleman's editor, the user manipulates the spreadsheet by gesturing with a stylus on a tablet (figure 2.6). For example, deletion is done by drawing an "X" over a cell, selection by an "0", and moving selected cells by an arrow, the tip of which indicates the destination of the move. The application is interesting in that it combines handwriting recognition (isolated letters and numbers) with gesturing. For example, by using handwriting the user can enter numbers or text into a cell without using a keyboard. The portion of the recognizer which classifies letters, numbers, and gestures of a fixed size and orientation has (presumably) been trained by example using standard handwriting recognition techniques. However, the recognition of gestures which vary in size or orientation requires hand coding [67].

Murase and Wakahara [89] describe a system in which freehand-drawn flowcharts symbols are recognized by machine (figure 2.7). Tamura and Kawasaki [128] have a system which recognizes sign-language gestures from video input (figure 2.8).

HITS from MCC [55] and Artkit from the University of Arizona [52] are both systems that may be used to construct gesture-based interfaces. The author has seen a system built with HITS similar

Figure 2.8: Sign language recognition (from Tamura [128])
This system processes an image from a video camera in order to recognize a form of Japanese sign language.

to that of Murase; in it an entire control panel is drawn freehand, and then the freehand symbols are segmented, classified and replaced by icons. (Similar work is discussed by Martin, *et. al* [82], also from MCC.) Artkit has much in common with the GRANDMA system described in this dissertation, and will be mentioned again later (Sections 4.1 and 6.8). Artkit systems tend to be similar to those created using GRANDMA, in that gesture commands are executed as soon as they are entered.

Kurtenbach and Buxton [75] have implemented a drawing program based on single-stroke gestures (figure 2.9). They have used the program to study, among other things, issues of scope in gestural systems. To the present author, GEdit's most interesting attribute is the use of compound gestures, as shown in the figure. GEdit's gesture recognizer is hand-coded.

The Glove-talk system [34] uses a DataGlove to control a speech synthesizer (figure 2.10). Like Artkit and the work described in Chapter 4, Glove-talk performs eager recognition: a gesture is recognized and acted upon without its end being indicated explicitly. Weimer and Ganapathy [136] describe a system combining DataGlove gesture and speech recognition.

The use of the circling gesture as an alternative means of selection is considered in Jackson and Roske-Hofstrand [61]. In their system, the start of the circling gesture is detected automatically, *i.e.* the mouse buttons are not used. Circling is also used for selection in the JUNO system from Xerox Corporation [142].

A number of computer products offer a stylus and tablet as their sole or primary input device. These systems include GRID Systems Corp.'s GRIDPad [50], Active Book Company's new portable [43], Pencept Inc.'s computer [59], Scenario's DynaWriter, Toshiba's PenPC, Sony's Palmtop, Mometa's laptop, MicroSlate's Datalite, DFM System's TraveLite, Agilis Corp.'s system, and Go Corp.'s PenPoint system [81, 24]. While details of the interface of many of these systems are hard to find (many of these systems have not yet been released), the author suspects that many use gestures. For further reading, please see [16, 106, 31].

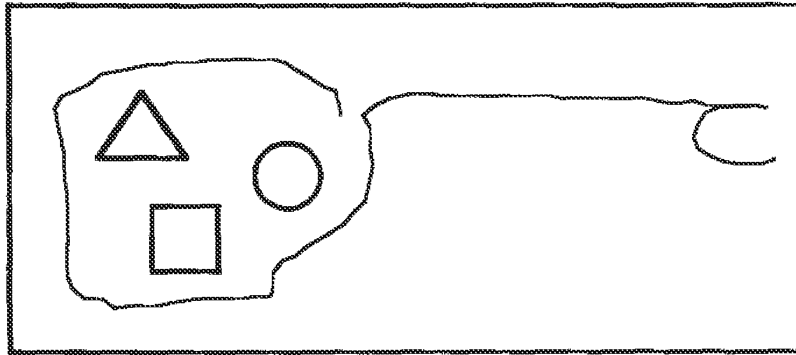


Figure 2.9: Copying a group of objects in GEdit (from Kurtenbach and Buxton [75])
Note the compound gesture: the initial closed curve does selection, and the final "C" indicates the data should be copied rather than moved.

root word	hand shape
come	
go	
I	
you	
short	

Figure 2.10: GloveTalk (from Fels and Hinton [34])
GloveTalk connects a DataGlove to a speech synthesizer through several neural networks. Gestures indicate root words (shown) and modifiers. Reversing the direction of the hand motion causes a word to be emitted from the synthesizer as well as indicating the start of the next gesture.

Tap	Select/Invoke
Press-hold	Initiate drag (move, wipe-through)
Tap-hold	Initiate drag (copy)
Flick (four directions)	Scroll/Browse
Cross out	Delete
Scratch out	Delete
Circle	Edit
Check	Options
Caret	Insert
Brackets	Select object, adjust selection
Pigtail (vertical)	Delete character
Down-right	Insert space

Figure 2.11: Basic PenPoint gestures (from Carr [24])

Recently, prototypes of Go Corporation's PenPoint system have been demonstrated. Each consists of a notebook-sized computer with a flat display. The sole input device is a stylus, which is used for gestures and handwriting on the display itself. Figure 2.11 shows the basic gestures recognized; depending on the context, additional gestures and handwriting can also be recognized. As can be seen, PenPoint gestures may consist of multiple strokes. Although it seems that trainable recognition algorithms are used internally, at the present time the user cannot add any new gestures to the existing set. The hardware is able to sense pen *proximity* (how near the stylus is to the tablet), which is used to help detect the end of multi-stroke gestures and characters. PenPoint applications include a drawing program, a word processor, and a form-based data entry system.

Many of the above systems combine gesture and direct manipulation in the same interface. GEdit, for example, appears to treat mouse input as gestural when begun on the background window, but drags objects when mouse input begins on the object. Almost none combine gesture and direct manipulation in the same interaction. One exception, PenPoint, uses the dot gesture (touching the stylus to the tablet and then not moving until recognition has been indicated) to drag graphic objects. Button Box does something similar for dragging objects. Artkit [52] uses eager recognition, more or less crediting the idea to me.

2.3 Approaches for Gesture Classification

Fu [40] states that "the problem of pattern recognition usually denotes a discrimination or classification of a set of processes or events." Clearly gesture recognition, in which the input is considered to be an event to be classified as one of a particular set of gestures, is a problem of pattern recognition.

In this dissertation, known techniques of pattern recognition are applied to the problem of sensing gestures.

The general pattern recognition problem consists of two subproblems: pattern representation and decision making [40]. This implies that the architecture of the general pattern recognition consists of two main parts. First, the *representer* takes the raw pattern as input and outputs the internal representation of the pattern. Then, the *decider* takes as input the output of the representer, and outputs a classification (and/or a description) of the pattern.

This section reviews the pattern recognition work relevant to gesture recognition. In particular, the on-line recognition of handwritten characters is discussed whenever possible, since that is the closest solved problem to gesture recognition. For a good overview of handwriting systems in general, see Suen *et. al.* [125] or Tappert *et. al.* [129].

The review is divided into two parts: alternatives for representers and alternatives for deciders. Each alternative is briefly explained, usually by reference to an existing system which uses the approach. The advantages and disadvantages of the alternative are then discussed, particularly as they apply to single-path gesture recognition.

2.3.1 Alternatives for Representers

The representer module takes the raw data from the input device and transforms it into a form suitable for classification by the decider. In the case of single-path gestures, as with on-line handprint, the raw data consists of a sequence of points. The representer outputs *features* of the input pattern.

Representers may be grouped in terms of the kinds of features which they output. The major kinds of features are: templates, global transformations, zones, and geometric features. While a single representer may combine different kinds of features, representers are discussed here as if each only outputs one kind of feature. This will make clearer the differences between the kinds of features. Also, in practice most representers do depend largely on a single kind of feature.

Templates.

Templates are the simplest features to compute: they are simply the input data in its raw form. For a path, a template would simply consist of the sequence of points which make up the path. Recognition systems based on templates require the decider to do the difficult work; namely, matching the template of the input pattern to stored example templates for each class.

Templates have the obvious advantage that the features are simple to compute. One disadvantage is that the size of the feature data grows with the size of the input, making the features unsuitable as input to certain kinds of deciders. Also, template features are very sensitive to changes in the size, location, or orientation of the input, complicating classifiers which attempt to allow for variations of these within a given class. Examples of template systems are mentioned in the discussion of template matching below.

Global Transformations.

Some of the problems of template features are addressed by global transformations of the input data. The transformations are often mathematically defined so as to be invariant under *e.g.* rotation, translation, or scaling of the input data. For example, the Fourier transform will result in features invariant with respect to rotation of the input pattern [46]. Global transformations generally output a fixed number of features, often smaller than the input data.

A set of fixed features allows for a greater variety in the choice of deciders, and obviously the invariance properties allow for variations within a class. Unfortunately, there is no way to "turn off" these invariances in order to disallow intra-class variation. Also, the global transformations generally take as input a two-dimensional raster, making the technique awkward to use for path data (it would have to first be transformed into raster data). Furthermore, the computation of the transformation may be expensive, and the resulting features do not usually have a useful parametric interpretations (in the sense of Section 1.6.2), requiring a separate pass over the data to gather parametric information.

Zones.

Zoning is a simple way of deriving features from a path. Space is divided into a number of zones, and an input path is transformed into the sequence of zones which the path traverses [57]. One variation on this scheme incorporates the direction each zone is entered into the encoding [101]. As with templates, the number of features are not fixed; thus only certain deciders may be used. The major advantage of zoning schemes are their simplicity and efficiency.

If the recognition is to be size invariant, zoning schemes generally require the input to be normalized ahead of time. Making a zoning scheme rotationally invariant is more difficult. Such normalizations make it impossible to compute zones incrementally as the input data is received. Also, small changes to a pattern might cause zones to be missed entirely, resulting in misclassification. And again, the features do not usually hold any useful parametric information.

Geometric Features.

Geometric features are the most commonly used in handwriting recognition [125]. Some geometric features of a path (such as its total length, total angle, number of times it crosses itself, *etc.*) represent global properties of the path. Local properties, such as the sequence of basic strokes, may also be represented.

It is possible to use combinations of geometric features, each invariant under some transformations of the input pattern but not others. For example, the initial angle of a path may be a feature, and all other features might be invariant with respect to rotation of the input. In this fashion, classifiers may potentially be created which allow different variations on a per-class basis.

Geometric features often carry useful parametric information, *e.g.* the total path length, a geometric feature, is potentially a useful parameter. Also, geometric features can be fed to deciders which expect a fixed number of features (if only global geometric features are used), or to deciders which expect a sequence of features (if local features are used).

Geometric features tend to be more complex to compute than the other types of features listed. With care, however, the computation can be made efficient and incremental. For all these reasons, the current work concentrates on the use of global geometric features for the single-path gesture recognition in this dissertation (see Chapter 3).

2.3.2 Alternatives for Deciders

Given a vector or sequence of features output by a representer, it is the job of the decider to determine the class of the input pattern with those features. Seven general methods for deciders may be enumerated: template-matching, dictionary lookup, a discrimination net, statistical matching, linguistic matching, connectionism, and *ad hoc*. Some of the methods are suitable to only one kind of representer, while others are more generally applicable.

Template-matching.

A template-matching decider compares a given input template to one or more prototypical templates of each expected class. Typically, the decider is based on a function which measures the similarity (or dissimilarity) between pairs of templates. The input is classified as being a member of the same class as the prototype to which it is most similar. Usually there is a similarity threshold, below which the input will be rejected as belonging to none of the possible classes.

The similarity metric may be computed as a correlation function between the input and the prototype [69]. Dynamic programming techniques may be used to efficiently warp the input in order to better match up points in the input template to those in the prototype [133, 60, 9].

Template systems have the advantage that the prototypes are simply example templates, making the system easy to train. In order to accommodate large variations, for example in the orientation of a given gesture, a number of different prototypes of various orientation must be specified. Unfortunately, a large number of prototypes can make the use of template matching prohibitively expensive, since the input pattern must be compared to every template.

Lipscomb [80] presents a variation on template matching used for recognizing gestures. In his scheme, each training example is considered at different resolutions, giving rise to multiple templates per example. (The algorithm is thus similar to multiscale algorithms used in image processing [138].) Lipscomb has applied the multiscale technique to stroke data by using an angle filter, in which different resolutions correspond to different thresholds applied to the angles in the gestures. To represent a gesture at a given resolution, points are discarded so that the remaining angles are all below the threshold. To classify an input gesture, first its highest resolution representation is (conceptually) compared to each template (at every resolution). Successively lower resolutions of the input are tried in turn, until an exact match is found. Multiple matches are decided in favor of the template whose resolution is closest to the current resolution of the input.

Dictionary lookup.

When the input features are a sequence of tokens taken from a small alphabet, lookup techniques may be used. This is often how zoning features are classified [101]. The advantage is efficient

recognition, since binary search (or similar algorithms) may be used to lookup patterns in the dictionary. Often some allowance is made for non-exact matches, since otherwise classification is sensitive to small changes in the input. Even with such allowances, dictionary systems are often brittle, due to the features employed (*e.g.* sequences of zones). Of course, a dictionary is initially created from example training input. It is also a simple matter to add new entries for rejected patterns; thus the dictionary system can adapt to a given user.

Discrimination nets.

A discrimination net (also called a decision tree) is basically a flowchart without loops. Each interior node contains a boolean condition on the features, and is connected to two other nodes (a "true" branch and a "false" branch). Each leaf node is labeled with a class name. A given feature set is classified by starting at the root note, evaluating each condition encountered and taking the appropriate branch, stopping and outputting the classification when a leaf node is reached.

Discrimination nets may be created by hand [25], or derived from example inputs [8]. They are more appropriate to classifying fixed-length feature vectors, rather than sequences of arbitrary length, and often result in accurate and efficient classifiers. However, discrimination nets trained by example tend to become unwieldy as the number of examples grows.

Statistical matching.

In statistical matching, the statistics of example feature vectors are used to derive classifiers. Typically, statistical matchers operate only on feature vectors, not sequences. Some typical statistics used are: average feature vector per class, per-class variances of the individual features, and per-class correlations within features. One method of statistical matching is to compute the distance of the input feature vector to the average feature vector of each class, choosing the class which is the closest. Another method uses the statistics to derive per-class discrimination functions over the features. Discrimination functions are like evaluation functions: each discrimination function is applied to the input feature vector, the class being determined by the largest result. Fisher [35] showed how to create discrimination functions which are simply linear combinations of the input features, and thus particularly efficient. Arakawa *et. al.*[3] used statistical classification of Fourier features for on-line handwriting recognition; Chapter 3 of the present work uses statistical classification of geometric features.

Some statistical classifiers, such as the Fisher classifier, make assumptions about the distributions of features within a class (such as multivariate normality); those tend to perform poorly when the assumptions are violated. Other classifiers [48] make no such assumptions, but instead attempt to estimate the form of the distribution from the training examples. Such classifiers tend to require many training examples before they function adequately. The former approach is adopted in the current work, with the feature set carefully chosen so as to not violate assumptions about the underlying distribution too drastically.

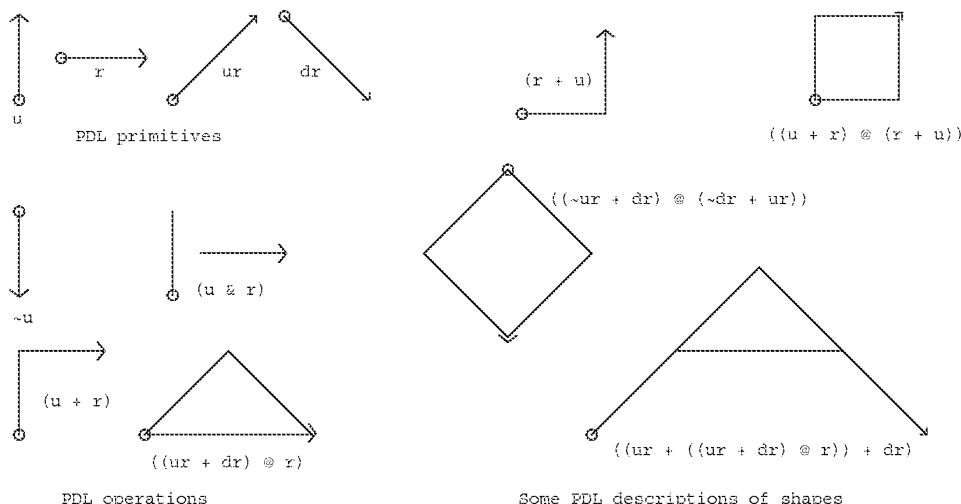


Figure 2.12: Shaw's Picture Description Language
PDL enables line drawings to be coded in string form, making it possible to apply textual parsing algorithms to the recognition of line drawings. The component line segments and combining operations are shown on the left; the right shows how the letter "A" can be described using these primitives.

Linguistic matching.

The linguistic approach attempts to apply automata and formal language theory to the problem of pattern recognition [37]. The representer outputs a sequence of tokens which is composed of a set of pattern primitives and composition operators representing the relation between the primitives. The decider has a grammar for each possible pattern class. It takes as input the sentence and attempts to parse it with respect to each pattern class grammar. Ideally, exactly one of the parses is successful and the pattern is classified thus. A useful side effect of the syntax analysis is the parse tree (or other parse trace) which reveals the internal structure of the pattern.

Linguistic recognizers may be classified based on the form of the representer output. If the output is a string then standard language recognition technology, such as regular expressions and context-free grammars, may be used to parse the input. An error-correcting parser may be used in order to robustly deal with errors in the input. Alternatively, the output of the representer may be a tree or graph, in which case the decider could use graph matching algorithms to do the parse.

The token sequence could come from a zoning representer, a representer based on local geometric properties, or from the output of a lower-level classifier. The latter is a hybrid approach–where, for example, statistical recognition is used to classify paths (or path segments), and linguistic recognition is used to classify based on the relationships between paths. This approach is similar to that taken by Fu in a number of applications [40, 39, 38].

Shaw's picture description language (PDL, see figure 2.12) has been used successfully to describe and classify line drawings [116, 40]. In another system, Stallings [120, 37] uses the composition

operators *left-of, above,* and *surrounds* to describe the relationships between strokes of Chinese characters.

A major problem with linguistic recognizers is the necessity of supplying a grammar for each pattern class. This usually represents considerably more effort than simply supplying examples for each class. While some research has been done on automatically deriving grammars from examples, this research appears not to be sufficiently advanced to be of use in a gesture recognition system. Also, linguistic systems are best for patterns with substantial internal structure, while gestures tend to be atomic (but not always [75]).

Connectionism.

Pattern recognition based on neural nets has received much research attention recently [65, 104, 132, 134]. A neural net is a configuration of simple processing elements, each of which is a super-simplified version of a neuron. A number of methods exist for training a neural network pattern recognizer from examples. Almost any of the different kinds of features listed above could serve as input to a neural net, though best results would likely be achieved with vectors of quantitative features. Also, some statistical discrimination functions may be implemented as simple neural networks.

Neural nets have been applied successfully to the recognition of line drawings [55, 82], characters [47], and DataGlove gestures [34]. Unfortunately, they tend to require a large amount of processing power, especially to train. It now appears likely that neural networks will, in the future, be a popular method for gesture recognition. The chief advantage is that neural nets, like template-based approaches, are able to take the raw sensor data as input. A neural network can learn to extract interesting features for use in classification. The disadvantage is that many labeled examples (often thousands) are needed in training.

The statistical classification method discussed in this dissertation may be considered a one-level neural network. It has the advantage over multilayer neural networks, in that it may be trained quickly using relatively few examples per class (typically 15). Rapid training time is important in a system that is used for prototyping gesture-based systems, since it allows the system designer to easily experiment with different sets of gestures for a given application.

Ad hoc methods.

If the set of patterns to be recognized is simple enough, a classifier may be programmed by hand. Indeed, this was the case in many of the gesture-based systems mentioned in Section 2.2. Even so, having to program a recognizer by hand can be difficult and makes the gesture set difficult to modify. The author believes that the difficulty of creating recognizers is one major reason why more gesture-based systems have not been built, and why there is a dearth of experiments which study the effect of varying the gestures in those systems which have been built. The major goal of this dissertation is to make the building of gesture-based systems easy by making recognizers specifiable by example, and incorporating them into an easy-to-use direct manipulation framework.

2.4 Direct Manipulation Architectures

A direct manipulation system is one in which the user manipulates graphical representations of objects in the task domain directly, usually with a mouse or other pointing device. In the words of Shneiderman [117],

> the central ideas seemed to be visibility of the object of interest; rapid, reversible, incremental actions; and replacement of complex command language syntax by direct manipulation of the object of interest–hence the term "direct manipulation."

As examples, he mentions display editors, Visicalc, video games, computer-aided design, and driving an automobile, among others.

For many application domains, the direct manipulation paradigm results in programs which are easy to learn and use. Of course there are tasks for which direct manipulation is not appropriate, due to the fact that the abstract nature of the task domain is not easily mapped onto concrete graphical objects [58]. For example, direct manipulation systems for the abstract task of programming have been rather difficult to design, though much progress has been made [98].

It is not intended here to debate the merits and drawbacks of direct manipulation systems. Instead, it is merely noted that direct manipulation has become an increasingly important and popular style of user interface. Furthermore, all existing gesture-based systems may be considered direct-manipulation systems. The reason is that graphical objects on the screen are natural targets of gesture commands, and updating those objects is an intuitive way of feeding back to the user the effect of his gesturing. In this section, existing approaches for constructing direct manipulation systems are reviewed. In Chapters 6 and 7 it is shown how some of these approaches may be extended to incorporate gestural input.

While direct manipulation systems are easy to use, they are among the most difficult kinds of interface to construct. Thus, there is a great interest in software tools for creating such interfaces. Myers [96] gives an excellent overview of the various tools which have been proposed for this purpose. Here, it is sufficient to divide user-interface software tools into three levels.

The lowest software level potentially seen by the direct manipulation system programmer is usually the *window manager*. Example window managers include X [113], News [127], Sun Windows [126], and Display Postscript [102]; see Myers [94] for an overview. For current purposes, it is sufficient to consider the window manager as providing a set of routines (*i.e.* a programming interface) for both output (textual and graphical) and input (keyboard and mouse or other device). Programming direct manipulation interfaces at the window manager level is a usually avoided, since a large amount of work will likely need to be redone for each application (*e.g.* menus will have to be implemented for each). Building from scratch this way will probably result in different and inconsistent interfaces for each application, making the total system difficult to recall and use.

The next software level is the *user interface toolkit*. Toolkits come in two forms: non-object-oriented and object-oriented. A toolkit provides a set of procedures or objects for constructing menus, scroll bars, and other standard interaction techniques. Most of the toolkits come totally disassembled, and it is up to the programmer to decide how to use the components. Some toolkits, notably MacApp [115] and GWUIMS [118], come partially assembled, making it easier for the programmer to customize the structure to fit the application. For this reason, some authors have

referred to these systems as User Interface Management Systems, though here they are grouped with the other toolkits.

A non-object-oriented toolkit is simply a set of procedures for creating and manipulating the interaction techniques. This saves the programmer the effort involved in programming these interaction techniques directly, and has the added benefit that all systems created using a single toolkit will look and act similarly. One problem with non-object-oriented toolkits is that they usually do not give much support for the programmer who wishes to create new interaction techniques. Such a programmer typically cannot reuse any existing code and thus finds himself bogged down with many low-level details of input and screen management.

Instead of procedures, object-oriented toolkits provide a class (an object type) for each of the standard interaction techniques. To use one of the interaction techniques in an interface, the programmer creates an instance of the appropriate class. By using the inheritance mechanism of the object-oriented programming language, the programmer can create new classes which behave like existing classes except for modifications specified by the programmer. This subclassing gives the programmer a method of customizing each interaction technique for the particular application. It also provides assistance to the programmer wishing to create new interaction techniques–he can almost always subclass an existing class, which is usually much easier than programming the new technique from scratch. One problem with object-oriented toolkits is their complexity; often the programmer needs to be familiar with a large part of the class hierarchy before he can understand the functionality of a single class.

User Interface Management Systems (UIMSs) form the software level above toolkits [96]. UIMSs are systems which provide a method for specifying some aspect of the user interface that is at a higher level than simply using the base programming language. For example, the RAPID/USE system [135] uses state transition diagrams to specify the structure of user input, the Syngraph system [64] uses context-free grammers similarly, and the Cousin system [51] uses a declarative language. Such systems encourage or enforce a strict separation between the user interface specification and the application code. While having modularity advantages, it is becoming increasingly apparent that such a separation may not be appropriate for direct manipulation interfaces [110].

UIMSs which employ *direct graphical specification* of interface components are becoming increasingly popular. In these systems, the UIMS is itself a direct manipulation system. The user interface designer thus uses direct manipulation to specify the components of the direct manipulation interface he himself desires to build. The NeXT Interface Builder [102] and the Andrew Development Environment Workbench (ADEW) [100] allow the placement and properties of existing interface components to be specified via direct manipulation. However, new interface components must be programmed in the object-oriented toolkit provided. In addition to the direct manipulation of existing interface components, Lapidary [93] and Peridot [90] enable new interface components to be created by direct graphical specification.

UIMSs are generally built on top of user interface toolkits. The UIMSs that support the construction of direct manipulation interfaces, such as the ones which use direct graphical specification, tend to be built upon object-oriented toolkits. Since object-oriented toolkits are currently the preferred vehicle for the creation of direct manipulation systems, this dissertation concentrates upon the problem of integrating gesture into such toolkits. In preparation for this, the architectures of

several existing object-oriented toolkits are now reviewed.

2.4.1 Object-oriented Toolkits

The object-oriented approach is often used for the construction of direct manipulation systems. Using object-oriented programming techniques, graphical objects on the screen can be made to correspond quite naturally with software objects internal to the system. The ways in which a graphic object can be manipulated correspond to the messages to which the corresponding software object responds. It is assumed that the reader of this dissertation is familiar with the concepts of object-oriented programming. Cox [27, 28], Stefik and Bobrow [121], Horn [56], Goldberg and Robson [44], and Schmucker [115] all present excellent overviews of the topic.

The Smalltalk-80 system [44] was the first object-oriented system that ran on a personal computer with a mouse and bitmapped display. From this system emerged the Model-View-Controller (MVC) paradigm for developing direct manipulation interfaces. Though MVC literature is only now beginning to appear in print [70, 63, 68], the MVC paradigm has directly influenced every object-oriented user interface architecture since its creation. For this reason, the review of object-oriented architectures for direct manipulation systems begins with a discussion of the use of the MVC paradigm in the Smalltalk-80 system.

The terms "model," "view," and "controller" refer to three different kinds of objects which play a role in the representation of single graphic object in a direct manipulation interface. A *model* is an object containing application specific data. Model objects encapsulate the data and computation of the task domain, and generally make no reference to the user interface.

A *view* object is responsible for displaying application data. Usually, a view is associated with a single model, and communicates with the model in order to acquire the application data that it will render on the screen. A single model may have multiple views, each potentially displaying different aspects of the model. Views implement the "look" of a user interface.

A *controller* object handles user interaction (*i.e.* input). Depending on the input, the controller may communicate directly with a model, a view, or both. A controller object is generally paired with a view object, where the controller handles input to a model and the view handles output. Internally, the controller and view objects typically contain pointers to each other and the associated model, and thus may directly send messages to each other and the model. Controllers implement the "feel" of a user interface.

When the application programmer codes a model object, for modularity purposes he does not generally include references to any particular view(s). The result is a separation between the application (the models) and the user interface (the views and controllers). There does however need to be some connection from a model to a view–otherwise how can the view be notified when the state of the model changes? This connection is accomplished in a modular fashion through the use of *dependencies*.

Dependencies work as follows: Any object may register itself as a dependent of any other object. Typically, a view object, when first created, registers as a dependent of a model object. Generally, there is a list of dependents associated with an object; in this way multiple views may be dependent on a single model. When an object that potentially has dependents changes its state, it sends itself the message `[self changed]`. Each dependent `d` of the object will then get sent the message `[d`

`update]`, informing it that an object upon which it is dependent has changed. Thus, dependencies allow a model to communicate to its views the fact that it has changed, without referring to the views explicitly.

Many views display rectangular regions on the screen. A view may have subviews, each of which typically results in an object displayed within the rectangular region of the parent view. The subviews may themselves have subviews, and so on recursively, giving rise to the *view hierarchy*. Typically, a subview's display is clipped so as to wholly appear within the rectangular region of its parent. A subview generally occludes part of its parent's view.

A common criticism of the MVC paradigm is that two objects (the view and controller) are needed to implement the user interface for a model where one would suffice. This, the argument goes, is not only inefficient, but also not modular. Why implement the look and feel separately when in practice they always go together?

The reply to this criticism states that it is useful (often or occasionally) to control look and feel separately [68]. Knolle discusses the usefulness of a single view having several interchangeable controllers; implementing different user abilities (*i.e.* beginning, intermediate, and advanced) with different controllers, and having the system adapt to the user's ability at runtime is one example. While Knolle's examples may not be very persuasive, there is an important application of separating views from controllers, namely, the ability to handle multiple input devices. Chapters 6 and 7 explore further the benefits accrued from the separation of views and controllers.

Nonetheless, there is a simplicity to be had by combining views and controllers into a single object, giving rise to object-oriented toolkits based on the Data-View (DV) paradigm. Though the terminology varies, MacApp [115, 114], the Andrew Toolkit [105], the NeWS Development environment [108], and InterViews [79] all use the DV paradigm. In this paradigm, data objects contain application specific data (and thus are identical to MVC models) while view objects combine the functionality of MVC view and controller objects. In DV systems, the look and feel of an object are very tightly coupled, and detailed assumptions about the input hardware (*e.g.* a three button mouse) get built into every view.

Object-oriented toolkits also vary in the method by which they determine which controller objects get informed of a particular input event, and also in the details of that communication. Typically, input events (such as mouse clicks) are passed down the view hierarchy, with a view querying its subviews (and so on recursively) to see if one of them wishes to handle the event before deciding to handle the event itself. Many variations on this scheme are possible.

Controllers may be written to have methods for messages such as `leftButtonDown`. This style, while convenient for the programmer, has the effect of wiring in details of the input hardware all throughout the system [115, 68]. The NeXT AppKit [102], passes input events to the controller object in a more general form. This is generalized even further in Chapters 6 and 7.

Controllers are a very general mechanism for handling input. Garnet [92], a modern MVC-based system, takes a different approach, called *interactors* [95, 91]. The key insight behind interactors is that there are only several different kinds of interactive behavior, and a (parameterizable) interactor can be built for each. The user-interface designer then needs only to choose the appropriate interactor for each interaction technique he creates.

Gestural input is not currently handled by the existing interactors. It would be interesting to see

if the interactor concept in Garnet is general enough to handle a gesture interactor. Unfortunately, the author was largely unaware of the Garnet project at the time he began the research described in Chapters 6 and 7. Had it been otherwise, a rather different method for incorporating gestures into direct manipulation systems than the one described here might have been created.

The Artkit system [52] has a considerably more general input mechanism than the MVC systems discussed thus far. Like the GRANDMA system discussed in this dissertation, Artkit integrates gesture into an object-oriented toolkit. Though developed simultaneously and independently, Artkit and GRANDMA have startlingly similar input architectures. The two systems will be compared in more detail in chapter 6.

Chapter 3

Statistical Single-Path Gesture Recognition

3.1 Overview

This chapter address the problem of recognizing single-path gestures. A single-path gesture is one that can be input with a single pointer, such as a mouse, stylus, or single-finger touch pad. It is further assumed that the start and end of the input gesture are clearly delineated. When gesturing with a mouse, the start of a gesture might be indicated by the pressing of a mouse button, and the end by the release of the button. Similarly, contact of the stylus with the tablet or of a finger with the touch screen could be used to delineate the endpoints of a gesture.

Baecker and Buxton[5] warn against using a mouse as a gestural input device for ergonomic reasons. For the research described in this chapter, the author has chosen to ignore that warning. The mouse was the only pointing device readily available when the work began. Furthermore, it was the only pointing device that is widely available--an important consideration as it allows others to utilize the present work. In addition, it is probably the case that any trainable recognizer that works well given mouse input could be made to work even better on devices more suitable for gesturing, such as a stylus and tablet.

The particular mouse used is labeled DEC Model VS10X-EA, revision A3. It has three buttons on top, and a metal trackball coming out of the bottom. Moving the mouse on a flat surface causes its trackball to roll. Inside the mouse, the trackball motion is mechanically divided into x and y components, and the mouse sends a pulse to the computer each time one of its components changes by a certain amount. The windowing software on the host implements mouse acceleration, meaning that the faster the mouse is moved a given distance, the farther the mouse cursor will travel on the screen. The metal mouseball was rolled on a Formica table, resulting it what might be termed a "hostile" system for studying gestural input.

All the work described in this chapter was developed on a Digital Equipment Corporation MicroVAX II.[1] The software was written in C [66] and runs on top of the MACH operating system

[1] MicroVAX is trademark of Digital Equipment Corporation.

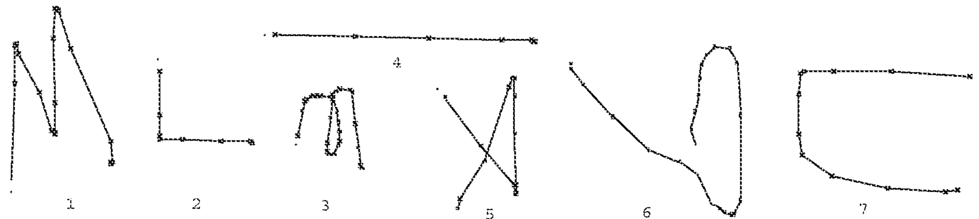


Figure 3.1: Some example gestures
The period indicates the start of the gesture. The actual mouse points that make up the gestures are indicated as well.

[131], which is UNIX[2] 4.3 BSD compatible. X10 [113] was the window system used, though there is a layer of software designed to make the code easy to port to other window systems.

3.2 Single-path Gestures

The gestures considered in this chapter consist of the two-dimensional path of a single point over time. Each gesture is represented as an array g of P time-stamped sample points:

$$g_p = (x_p, y_p, t_p) \qquad 0 \leq p < P.$$

The points are time stamped (the t_p) since the typical interface to many gestural input devices, particularly mice, does not deliver input points at regular intervals. In this dissertation, only two-dimensional gestures are considered, but the methods described may be generalized to the three-dimensional case.

When an input point is very close to the previous input point, it is ignored. This simple preprocessing of the input results in features that are much more reliable, since much of the jiggle, especially at the end of a gesture, is eliminated. The result is a large increase in recognition accuracy.

For the particular mouse used for the majority of this work, "very close" meant within three pixels. This threshold was empirically determined to produce an optimal recognition rate on a number of gesture sets.

Similar, but more complicated preprocessing was done by Leedham, *et. al.*, in their Pittman's shorthand recognition system[77]. The difference in preprocessing in Leedham's system and the current work stems largely from the difference in input devices (Leedham used an instrumented pen), indicating that preprocessing should be done on a per-input-device basis.

Figure 3.1 shows some example gestures used in the GDP drawing editor. The first point (g_0) in each gesture is indicated by a period. Each subsequent point (g_p) is connected by a line segment to the previous point (g_{p-1}). The time stamps are not shown in the figure.

The gesture recognition problem is stated as follows: There is a set of C gesture classes, numbered 0 through $C-1$. The classes may be specified by description, or, as is done in the present

[2] UNIX is a trademark of Bell Laboratories.

work, by example gestures for each class. Given an input gesture g, the problem is to determine the class to which g belongs (*i.e.* the class whose members are most like g). Some classifiers have a reject option: if g is sufficiently different so as not to belong to any of the gesture classes, it should be rejected.

3.3 Features

Statistical gesture recognition is done in two steps. First, a set of features is extracted from the input gesture. This set is represented as a feature vector, $\mathbf{f} = [f_1, \ldots, f_F]'$. (Here and throughout, the prime denotes vector transpose.) The feature vector is then classified as one of the possible gesture classes.

The set of features used was chosen according to the following criteria:

The number of features should be small. In the present scheme, the amount of time it takes to classify a gesture given the feature vector is proportional to the product of the size of the feature vector (*i.e.* the number of features) and the number of different gesture classes. Thus, for efficiency reasons, the number of features should be kept as small as possible while still being able to distinctly represent the different classes.

Each feature should be calculated efficiently. It is essential that the calculation of the feature vector itself not be too expensive: the amount of time to update the value of a feature when an input point g_p is received should be bounded by a constant. In particular, features that require all previous points to be examined for each new input point are disallowed. In this manner, very large gestures (those consisting of many points) are recognized as efficiently as smaller gestures.

In practice, this incremental calculation of features is often achieved by computing auxiliary features not used in classification. For example, if one feature is the average x value of the input points, an auxiliary feature consisting of the sum of the x values might be computed. This would require constant time (one addition) per input point. When the feature vector is needed (for classification) the average x value feature is computed in constant time by dividing the above sum by the number of input points.

Each feature should have a meaningful interpretation. Unlike simple handwriting systems, the gesture-based systems built here use the features not only for classification, but also for parametric information. For example, a drawing program might use the initial angle of a gesture to orient a newly created rectangle. While it is possible to extract such gestural attributes independent of classification, it is potentially less efficient to do so.

Meaningful features also provide useful information to the designer of a set of gesture classes for a particular application. By understanding the set of features, the designer has a better idea of what kind of gestures the system can and cannot distinguish; she is thus more likely to design gestures that can be classified accurately.

Individual features should have Gaussian-like distributions. The classifier described in this chapter is optimal when, among other things, within a given class each feature has a Gaussian distribution. This is because a class is essentially represented by its mean feature vector, and classification of an example takes place, to a first approximation, by determining the class whose mean feature vector is closest to the example's. Classification may suffer if a given feature in a given class has, for example, a bimodal distribution, whereby it tends toward one of two different values.

This requirement is satisfied when the feature is *stable*, meaning a small change in the input gesture results in a small change in the value of the feature. In general, this rules out features that are small integers, since presumably some small change in a gesture will cause a discrete unit step in the feature. When possible, features that depend on thresholds should also be avoided for similar reasons. Ideally, a feature is a real-valued continuous function of the input points.

Note that the input preprocessing is essentially a thresholding operation, and does have the effect that a seemingly small change in the gesture can cause big changes in the feature vector. However, eliminating this preprocessing would allow the noise inherent in the input device to seriously affect certain features. Thus, thresholding should not be ruled out per-se, but the tradeoffs must be considered. Another alternative is to use multiple thresholds to achieve a kind of multiscale representation of the input, thus avoiding problems inherent in using a single threshold [80].

The particular set of features used here evolved over the creation of two classifiers, the first being for a subset of GDP gestures, the second being a recognizer of upper-case letters, as handwritten by the author. In the current version of the recognition program, thirteen features are employed. Figure 3.2 depicts graphically the values used in the feature calculation.

The features are:

Cosine and sine of initial angle with respect to the X axis:

$$f_1 = \cos\alpha = (x_2 - x_0)/d$$
$$f_2 = \sin\alpha = (y_2 - y_0)/d$$

where $d = \sqrt{(x_2 - x_0)^2 + (y_2 - y_0)^2}$.

Length of the bounding box diagonal:

$$f_3 = \sqrt{(x_{max} - x_{min})^2 + (y_{max} - y_{min})^2}$$

where $x_{max}, x_{min}, y_{max}, y_{min}$ are the maximum and minimum values for x_p and y_p respectively.

Angle of the bounding box:

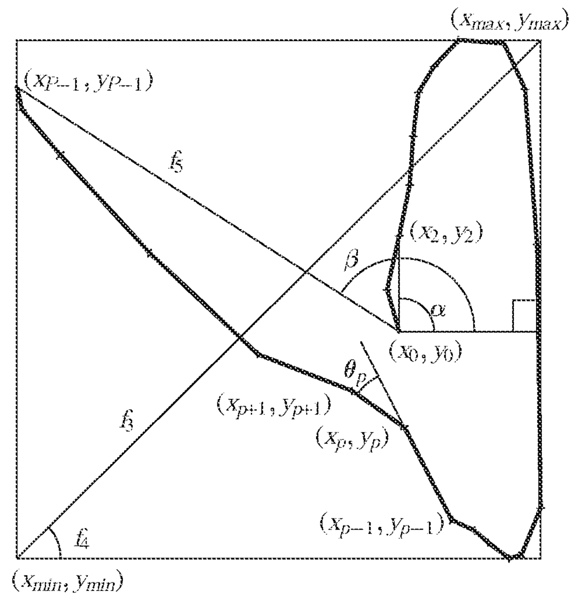


Figure 3.2: Feature calculation

Gesture 6 of figure 3.1 is shown with its relevant lengths and angles labeled with the intermediate variables used to compute features or the features themselves where possible.

$$f_4 = \arctan \frac{y_{max} - y_{min}}{x_{max} - x_{min}}$$

Distance between first and last point:

$$f_5 = \sqrt{(x_{P-1} - x_0)^2 + (y_{P-1} - y_0)^2}$$

Cosine and sine of angle between first and last point:

$$f_6 = \cos \beta = (x_{P-1} - x_0)/f_5$$
$$f_7 = \sin \beta = (y_{P-1} - y_0)/f_5$$

Total gesture length:

$$\text{Let } \Delta x_p = x_{p+1} - x_p$$
$$\Delta y_p = y_{p+1} - y_p$$

$$f_8 = \sum_{p=0}^{P-2} \sqrt{\Delta x_p^2 + \Delta y_p^2}$$

Total angle traversed (derived from the dot and cross product definitions[73]):

$$\theta_p = \arctan \frac{\Delta x_p \Delta y_{p-1} - \Delta x_{p-1} \Delta y_p}{\Delta x_p \Delta x_{p-1} + \Delta y_p \Delta y_{p-1}}$$

$$f_9 = \sum_{p=1}^{P-2} \theta_p$$

$$f_{10} = \sum_{p=1}^{P-2} |\theta_p|$$

$$f_{11} = \sum_{p=1}^{P-2} \theta_p^2$$

Maximum speed (squared):

$$\Delta t_p = t_{p+1} - t_p$$

$$f_{12} = \max_{p=0}^{P-2} \frac{\Delta x_p^2 + \Delta y_p^2}{\Delta t_p^2}$$

Path duration:

$$f_{13} = t_{P-1} - t_0$$

Features f_{12} and f_{13} allow the gesture recognition to be based on temporal factors; thus gestures have a dynamic component and are not simply static pictures.

Some features (f_1, f_2, f_6, and f_7) are sines or cosines of angles, while others (f_5, f_{10}, f_{11}, f_{12}) depend on angles directly and thus require inverse trigonometric functions to compute. A four-quadrant arctangent is needed to compute θ_p; the arctangent function must take the numerator and denominator as separate parameters, returning an angle between $-\pi$ and π. For efficient recognition, it would be desirable to use just a single feature to represent an angle, rather than both the sine and cosine. However, the recognition algorithm requires that each feature have approximately a Gaussian distribution; this poses a problem when a small change in a gesture causes a large change in angle measurement due to the discontinuity when near $\pm\pi$. This mattered for initial angle, and the angle between the start and end point of the gesture, so each of these angles is represented by its sine and cosine. The bounding box angle is always between 0 and $\pi/2$ so there was no discontinuity problem for it.

For features dependent on θ_i, the angle between three successive input points, the discontinuity only occurs when the gesture stroke turns back upon itself. In practice, likely due to the few gestures used which have such changes, the recognition process has not been significantly hampered by the potential discontinuity (but see Section 9.1.1). The feature f_9 is a measure of the total angle traversed; in a gesture consisting of two clockwise loops, this feature might have a value near 4π. If the gesture was a clockwise loop followed by a counterclockwise loop, f_9 would be close to zero. The feature f_{10} accumulates the absolute value of instantaneous angle; in both loop gestures, its value would be near 4π. The feature f_{11} is a measure of the "sharpness" of gesture.

Figure 3.3 shows the value of some features as a function of p, the input point, for gestures 1 and 2 of figure 3.1. Note in particular how the value for f_{11} (the sharpness) increases at the angles of the gesture. The feature values at the last (rightmost) input point are the ones that are used to classify the gesture. The intent of the graph is to show how the features change with each new input point.

All the features can be computed incrementally, with a constant amount of work being done for each new input point. By utilizing table lookup for the square root and inverse trig functions, the amount of computation per input point can be made quite small.

A number of features were tried and found not to be as good as the features used. For example, instead of the sharpness metric f_{11}, initially a count of the number of times θ_p exceeded a certain threshold was used. The idea was to count sharp angles. While this worked fairly well, the more continuous measure of sharpness was found to give much better results. In general, features that are discrete counts do not work as well as continuous features that attempt to quantify the same phenomena. The reason for this is probably that continuous features more closely satisfy the normality criterion. In other words, an error or deviation in a discrete count tends to be much more significant than an error or deviation in continuous metric.

Appendix A shows the C code for incrementally calculating the feature vector of a gesture.

3.4 Gesture Classification

Given the feature vector $\mathbf{x}$ computed for an input gesture g, the classification algorithm is quite simple and efficient. Associated with each gesture class is a linear evaluation function over the features. Gesture class c has weights $w_i^{\hat{c}}$ for $0 \leq i \leq F$, where F is the number of features, currently 13. (Per-class variables will be written using superscripts with hats to indicate the class. These are not and should not to be confused with exponentiation.) The evaluation functions are calculated as follows:

$$v^{\hat{c}} = w_0^{\hat{c}} + \sum_{i=1}^{F} w_i^{\hat{c}} x_i \qquad 0 \leq c < C \tag{3.1}$$

The value $v^{\hat{c}}$ is the evaluation of class c. The classifier simply determines the c for which $v^{\hat{c}}$ is a maximum; this c is the classification of the gesture g. The possibility of rejecting g is discussed in Section 3.6.

Practitioners of pattern recognition will recognize this classifier as the classic linear discriminator [35, 30, 62, 74]. With the correct choice of weights $w_i^{\hat{c}}$, the linear discriminator is known to be optimal when (1) within a class the feature vectors have a multivariate normal distribution, and (2)

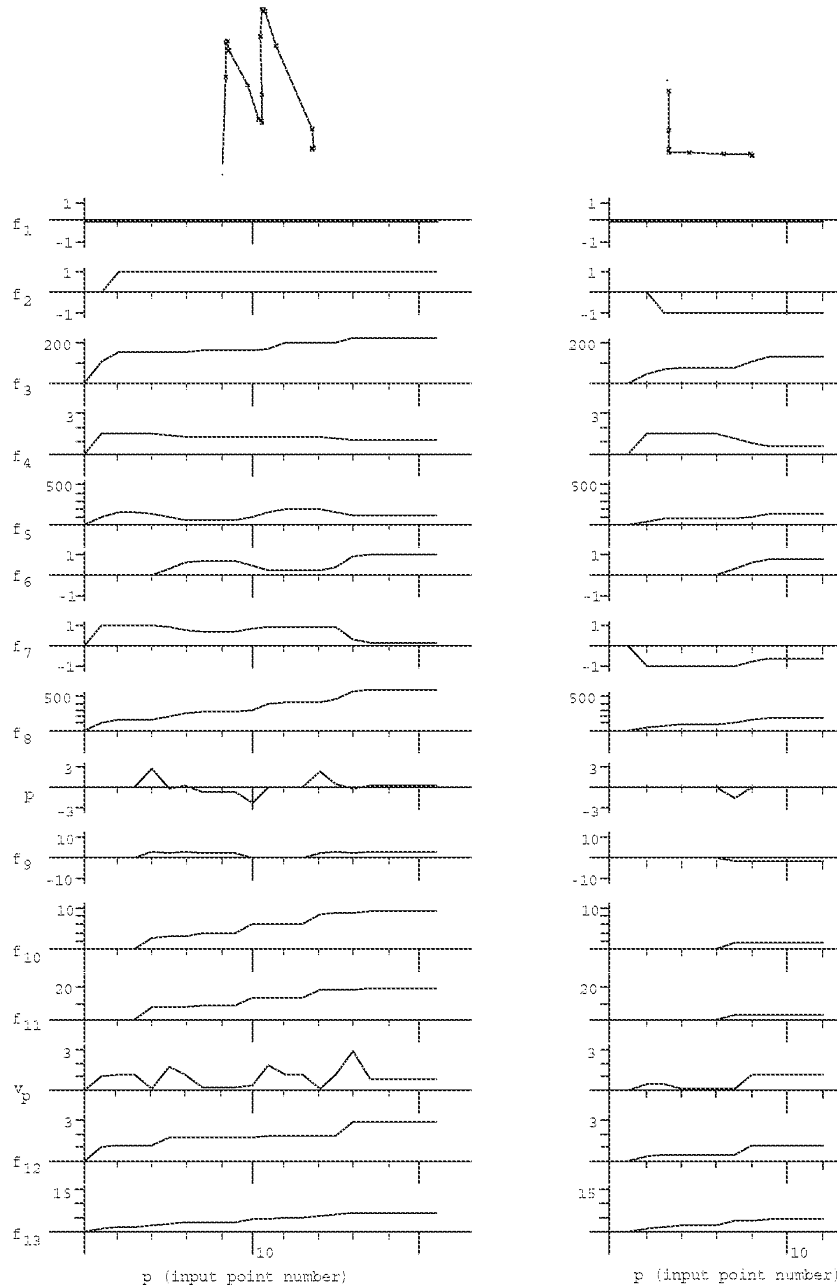


Figure 3.3: Feature vector computation

These graphs show how feature vectors change with each new input point. The left graphs refer to features of gesture 1 of 3.1 (an "M"), the right graphs to gesture 2 (an "L"). The final values of the features ($p = 21$ for gesture 1, $p = 12$ for gesture 2) are the ones used for classification. The instantaneous angle θ_p and velocity v_p have been included in the figure, although they are not part of the feature vector.

the per class feature covariance matrices are equal. (Exactly what this means is discussed in the next section. Other continuous distributions for which linear discriminant functions are optimal are investigated by Cooper [26].) These conditions do not hold for most sets of gesture classes given the feature set described; thus weights calculated assuming these conditions will not be optimal, even among linear classifiers (and even the optimal linear classifier can be outperformed by some non-linear classifiers if the above conditions are not satisfied). However, given the above set of features, linear discriminators computed as if the conditions are valid have been found to perform quite acceptably in practice.

3.5 Classifier Training

Once the decision has been made to use linear discriminators, the only problem that remains is the determination of the weights from example gestures of each class. This is known as the training problem.

Two methods for computing the weights were tried. The first was the multiclass perceptron training procedure described in Sklansky and Wassel[119]. The hope was that this method, which does not depend on the aforementioned conditions to choose weights, might perform better than methods that did. In this method, an initial guess of the weights was made, which are then used to classify the first example. For each class whose evaluation function scored higher than the correct class, each weight is reduced by an amount proportional to the corresponding feature of the example, while the correct class has its weights increased by the same amount. This is similar to back-propagation learning procedures in neural nets [34]. In this manner, all the examples are tried, multiple times if desired.

This method has the advantage of being simple, as well as needing very few example gestures to achieve reasonable results. However, the behavior of the classifier depends on the order in which the examples are presented for training, and good values for the initial weights and the constant of proportionality are difficult to determine in advance but have a large effect on the success and training efficiency of the method. The number of iterations of the examples is another variable whose optimum value is difficult to determine. Perhaps the most serious problem is that a single bad example might seriously corrupt the classifier.

Eventually, the perceptron training method was abandoned in favor of the plug-in estimation method. The plug-in estimation method usually performs approximately equally to the best perceptron-trained classifiers, and has none of the vagueness associated with perceptron training. In this method, the means of the features for each class are estimated from the example gestures, as is the common feature covariance matrix (all classes are assumed to have the same one). The estimates are then used to approximate the linear weights that would be optimal assuming the aforementioned conditions were true.

3.5.1 Deriving the linear classifier

The derivation of the plug-in classifier is given in detail in James [62]. James' explanation of the derivation is particularly good, though unfortunately the derivation itself is riddled with typos and

other errors. Krzanowski [74] gives a similar derivation (with no errors), as well as a good general description of multivariate analysis. The derivation is summarized here for convenience.

Consider the class of "L" gestures, drawn starting from the top-left. One example of this class is gesture 2 in figure 3.1. It is easy to generate many more examples of this class. Each one gives rise to a feature vector, considered to be a column vector of F real numbers $(f_0, \ldots, f_{F-1})'$.

Let f be the random vector (*i.e.* a vector of random variables) representing the feature vectors of a given class of gestures, say "L" gestures. Assume (for now) that f has a *multivariate normal* distribution. The multivariate normal distribution is a generalization to vectors of the normal distribution for a single variable. A single variable (univariate) normal distribution is specified by its mean value and variance. Analogously, a multivariable normal distribution is specified by its mean vector, $\overline{\mu}$, and covariance matrix, Σ. In a multivariate normal distribution, each vector element (feature) has a univariate normal distribution, and the mean vector is simply a vector consisting of the means of the individual features. The variance of the features form the diagonal of the covariance matrix; the off-diagonal elements represent correlations between features.

The univariate normal distribution has a density function which is the familiar bell-shaped curve. The analog in the two variable (bivariate) case is a three-dimensional bell shape. In this case, the lines of equal probability (cross sections of the bell) are concentric ellipses. The axes of the ellipses are parallel to the feature axes if and only if the variables are uncorrelated. By analogy, in the higher dimensional cases, the distribution has a hyper-bell shape, and the equiprobability hypersurfaces are ellipsoids.

A more in-depth discussion of the properties of the multivariate normal distribution would take us too far afield here. The reader unfamiliar with the subject is asked to rely on the analogy with the univariate case, or to refer to a good text, such as Krzanowski [74].

The multivariate normal probability density function is the multivariate analog to the bell-shaped curve. It is written here as a conditional probability density, *i.e.* the density of the probability of getting vector $\mathbf{x}$ given $\mathbf{x}$ comes from multivariate distribution L with F variables, mean $\overline{\mu}$, and covariance matrix Σ.

$$p(\mathbf{x} \mid L) = (2\pi)^{-F/2}|\Sigma|^{-1/2} e^{-\frac{1}{2}(\mathbf{x}-\overline{\mu})'\Sigma^{-1}(\mathbf{x}-\overline{\mu})} \tag{3.2}$$

Note that this expression involves both the determinant and the inverse of the covariance matrix. The interested reader should verify that it reduces to the standard bell-shaped curve in the univariate case ($F = 1, \Sigma = [\sigma^2]$).

In the univariate case, to determine the probability that the value of a random variable will lie within a given interval, simply integrate the probability density function over that interval. Analogously in the multivariate case, given an interval for each of the variables (*i.e.* a hypervolume) perform a multiple integral, integrating each variable over its interval to determine the probability a random vector is within the hypervolume.

All this is preparation of the derivation of the linear classifier. Assume an example feature vector $\mathbf{x}$ to be classified is given. Let $C^{\hat{c}}$ denote the event that a random feature vector $\mathbf{X}$ is in class c and $\mathbf{x}$, when used as an event, denote the event that the random feature vector $\mathbf{X}$ has value $\mathbf{x}$. We are interested in $P(C^{\hat{c}} \mid \mathbf{x})$, the probability that the particular feature vector $\mathbf{x}$ is in group $C^{\hat{c}}$. A reasonable classification rule is to assign $\mathbf{x}$ to the class i whose probability $P(C^{\hat{\imath}} \mid \mathbf{x})$ is greater than that of the other classes, *i.e.* $P(C^{\hat{\imath}} \mid \mathbf{x}) > P(C^{\hat{\jmath}} \mid \mathbf{x})$ for all $j \neq i$. This rule, which assigns the example

to the class with the highest conditional probability, is known as *Bayes' rule*.

The problem is thus to determine $P(C^{\hat{c}} \mid \mathbf{x})$ for all classes c. Bayes' theorem tells us

$$P(C^{\hat{c}} \mid \mathbf{x}) = \frac{P(\mathbf{x} \mid C^{\hat{c}})P(C^{\hat{c}})}{\sum_{\text{all } k} P(\mathbf{x} \mid C^{\hat{k}})P(C^{\hat{k}})} \tag{3.3}$$

Substituting, the assignment rule now becomes: assign $\mathbf{x}$ to class i if $P(\mathbf{x} \mid C^{\hat{\imath}})P(C^{\hat{\imath}}) > P(\mathbf{x} \mid C^{\hat{\jmath}})P(C^{\hat{\jmath}})$ for all $j \neq i$.

The terms of the form $P(C^{\hat{c}})$ are the *a priori* probabilities that a random example vector is in class c. In a gesture recognition system, these prior probabilities would depend on the frequency that each gesture command is likely to be used in an application. Lacking any better information, let us assume that all gestures are equally likely, resulting in the rule: assign $\mathbf{x}$ to class i if $P(\mathbf{x} \mid C^{\hat{\imath}}) > P(\mathbf{x} \mid C^{\hat{\jmath}})$ for all $j \neq i$.

A conditional probability of the form $P(\mathbf{x} \mid C^{\hat{c}})$ is known as the *likelihood* of $C^{\hat{c}}$ with respect to $\mathbf{x}$ [30]; assuming equal priors essentially replaces Bayes' rule with one that gives the maximum likelihood.

Assume now that each $C^{\hat{c}}$ is multivariate normal, with mean vector $\overline{\mu}^{\hat{c}}$, and covariance matrix $\Sigma_{\hat{c}}$. Substituting the multivariate normal density functions (equation 3.2) for the probabilities gives the assignment rule: assign $\mathbf{x}$ to class i if, for all $j \neq i$,

$$(2\pi)^{-F/2}|\Sigma_{\hat{\imath}}|^{-1/2}e^{-\frac{1}{2}(\mathbf{x}-\overline{\mu}^{\hat{\imath}})'\Sigma_{\hat{\imath}}^{-1}(\mathbf{x}-\overline{\mu}^{\hat{\imath}})} > (2\pi)^{-F/2}|\Sigma_{\hat{\jmath}}|^{-1/2}e^{-\frac{1}{2}(\mathbf{x}-\overline{\mu}^{\hat{\jmath}})'\Sigma_{\hat{\jmath}}^{-1}(\mathbf{x}-\overline{\mu}^{\hat{\jmath}})}$$

Taking the natural log of both sides, canceling, and multiplying through by -1 (thus reversing the inequality) gives the rule: assign $\mathbf{x}$ to class i if, for all $j \neq i$,

$$d^{\hat{\imath}}(\mathbf{x}) < d^{\hat{\jmath}}(\mathbf{x}), \text{ where } d^{\hat{c}}(\mathbf{x}) = \ln|\Sigma_{\hat{c}}| + (\mathbf{x} - \overline{\mu}^{\hat{c}})'\Sigma_{\hat{c}}^{-1}(\mathbf{x} - \overline{\mu}^{\hat{c}}) \tag{3.4}$$

$d^{\hat{c}}(\mathbf{x})$ is the discrimination function for class c applied to $\mathbf{x}$. This is *quadratic* discrimination, since $d^{\hat{c}}(\mathbf{x})$ is quadratic in elements of $\mathbf{x}$ (the features). The discriminant computation involves the weighted sum of the pairwise products of features, as well as terms linear in the features, and a constant term.

Making the further assumption that all the per-class covariances matrices are equal, *i.e.* $\Sigma_{\hat{\imath}} = \Sigma_{\hat{\jmath}} = \Sigma$, the assignment rule takes the form: assign $\mathbf{x}$ to class i if, for all $j \neq i$,

$$\ln|\Sigma| + (\mathbf{x} - \overline{\mu}^{\hat{\imath}})'\Sigma^{-1}(\mathbf{x} - \overline{\mu}^{\hat{\imath}}) < \ln|\Sigma| + (\mathbf{x} - \overline{\mu}^{\hat{\jmath}})'\Sigma^{-1}(\mathbf{x} - \overline{\mu}^{\hat{\jmath}}).$$

Distributing the subtractions and multiplying through by $-\frac{1}{2}$ gives the rule: assign $\mathbf{x}$ to class i if, for all $j \neq i$,

$$\nu^{\hat{\imath}}(\mathbf{x}) > \nu^{\hat{\jmath}}(\mathbf{x}), \text{ where } \nu^{\hat{c}}(\mathbf{x}) = (\overline{\mu}^{\hat{c}})'\Sigma^{-1}\mathbf{x} - \tfrac{1}{2}(\overline{\mu}^{\hat{c}})'\Sigma^{-1}\overline{\mu}^{\hat{c}} \tag{3.5}$$

Note that the discrimination functions $\nu^{\hat{c}}(\mathbf{x})$ are linear in the features (*i.e.* the elements of $\mathbf{x}$), the weights being $(\overline{\mu}^{\hat{c}})'\Sigma^{-1}$ and the constant term being $-\frac{1}{2}(\overline{\mu}^{\hat{\jmath}})'\Sigma^{-1}\overline{\mu}^{\hat{\jmath}}$.

Comparing equations 3.5 and 3.1 it is seen that to have the optimum classifier (given the assumptions) we take

$$w_j^{\hat{c}} = \sum_{i=1}^{F} \Sigma_{ij}^{-1} \overline{\mu}_i^{\hat{c}} \qquad 1 \leq j \leq F$$

and

$$w_0^{\hat{c}} = -\frac{1}{2} \sum_{i=1}^{F} w_i^{\hat{c}} \overline{\mu}_i^{\hat{c}}$$

for all classes c. It is not possible to know the $\overline{\mu}^{\hat{c}}$ and Σ; these must be estimated from the examples as described in the next section. The result will be that the $w_i^{\hat{c}}$ will be estimates of the optimal weights.

The possibility of a tie for the largest discriminant has thus far neglected. If $v^{\hat{i}}(\mathbf{x}) = v^{\hat{k}}(\mathbf{x}) > v^{\hat{j}}(\mathbf{x})$ for all $j \neq i$ and $j \neq k$, it is clear that the classifier may arbitrarily choose i or k as the class of $\mathbf{x}$. However, this is a prime case for rejecting the gesture $\mathbf{x}$ altogether, since it is ambiguous. This kind of rejection is generalized in Section 3.6.

3.5.2 Estimating the parameters

The linear classifier just derived is optimal (given all the assumptions) in the sense that it maximizes the probability of correct classification. However, the parameters needed to operate the classifier, namely the per-class mean vectors $\overline{\mu}^{\hat{c}}$ and the common covariance matrix Σ, are not known *a priori*. They must be estimated from the training examples. The simplest approach is to use the plug-in estimates for these statistics. Since the equations that follow actually need to be programmed, the matrix notation is discarded in favor of writing the sums out explicitly in terms of the components.

Let $f_{ei}^{\hat{c}}$ be the ith feature of the eth example of gesture class c, $0 \leq e < E^{\hat{c}}$, where $E^{\hat{c}}$ is the number of training examples of class c. The plug-in estimate of $\overline{\mu}^{\hat{c}}$, the mean feature vector per class, is denoted $\overline{f}_i^{\hat{c}}$. It is simply the average of the features in the class:

$$\overline{f}_i^{\hat{c}} = \frac{1}{E^{\hat{c}}} \sum_{e=0}^{E^{\hat{c}}-1} f_{ei}^{\hat{c}}$$

$s_{ij}^{\hat{c}}$ is the plug-in estimate of $\Sigma_{\hat{c}}$, the feature covariance matrix for class c:

$$s_{ij}^{\hat{c}} = \sum_{e=0}^{E^{\hat{c}}-1} (f_{ei}^{\hat{c}} - \overline{f}_i^{\hat{c}})(f_{ej}^{\hat{c}} - \overline{f}_j^{\hat{c}})$$

(For convenience in the next step, the usual $1/(E^{\hat{c}} - 1)$ factor has not been included in $s_{ij}^{\hat{c}}$.) The $s_{ij}^{\hat{c}}$ are averaged to give s_{ij}, an estimate of the common covariance matrix Σ.

$$s_{ij} = \frac{\sum_{c=0}^{C-1} s_{ij}^{\hat{c}}}{-C + \sum_{c=0}^{C-1} E^{\hat{c}}} \qquad (3.6)$$

The plug-in estimate of the common covariance matrix s_{ij} is then inverted, the result of which is denoted $(s^{-1})_{ij}$.

The $v^{\hat{c}}$ are estimates of the optimal evaluation functions $\nu^{\hat{c}}(\mathbf{x})$. The weights $w_j^{\hat{c}}$ are computed from the estimates as follows:

$$w_j^{\hat{c}} = \sum_{i=1}^{F} (s^{-1})_{ij} \overline{f}_i^{\hat{c}} \qquad 1 \leq j \leq F$$

and

$$w_0^{\hat{c}} = -\frac{1}{2} \sum_{i=1}^{F} w_i^{\hat{c}} \overline{f}_i^{\hat{c}}$$

As mentioned before, it is assumed that all gesture classes are equally likely to occur. The constant terms $w_0^{\hat{c}}$ may be adjusted if the *a priori* probabilities of each gesture class are known in advance, though the author has not found this to be necessary for good results. If the derivation of the classifier is carried out without assuming equal probabilities, the net result is, for each class, to add $\ln P(C^{\hat{c}})$ to $w_0^{\hat{c}}$. A similar correction may be made to the constant terms if differing per-class costs for misclassification must be taken into account [74].

Estimating the covariance matrix involves estimating its $F(F+1)/2$ elements. The matrix will be singular if, for example, less than approximately F examples are used in its computation. Or, a given feature may have zero variance in every class. In these cases, the classifier is underconstrained. Rather then give up (which seems an inappropriate response when underconstrained) an attempt is made to fix a singular covariance matrix. First, any zero diagonal element is replaced by a small positive number. If the matrix is still singular, then a search is made to eliminate unnecessary features.

The search starts with an empty set of features. At each iteration, a feature i is added to the set, and a covariance matrix based only on the features in the set is constructed (by taking the singular $F \times F$ covariance matrix and using only the rows and columns of those features in the set). If the constructed matrix is singular, feature i is removed from the set, otherwise i is kept. Each feature is tried in turn. The result is a covariance matrix (and its inverse) of dimensionality smaller than $F \times F$. The inverse covariance matrix is expanded to size $F \times F$ by adding rows and columns of zeros for each feature not used. The resulting matrix is used to compute the weights.

Appendix A shows C code for training classifiers and classifying feature vectors.

3.6 Rejection

Given an input gesture g the classification algorithm calculates the evaluation $v^{\hat{c}}$, for each class c. The class k whose evaluation $v_{\hat{k}}$ is larger than all other $v^{\hat{c}}$ is presumed to be the class of g. However, there are two cases that might cause us to doubt the correctness of the classifier. The gesture g may be *ambiguous*, in that it is similar to the gestures of more than one class. Also, g may be an *outlier*, different from any of the expected gesture classes.

It would be desirable to get an estimate of how sure the classifier is that the input gesture is unambiguously in class i. Intuitively, one might expect that if some $v^{\hat{m}}$, $m \neq i$, is close to $v^{\hat{i}}$, then

the classifier is unsure of its classification, since it almost picked m instead of i. This intuition is borne out in the expression for the probability that the feature vector $\mathbf{x}$ is in class $\hat{\imath}$. Again assuming normal features, equal covariances, and equal prior probabilities, substitute the multivariate normal density function (equation 3.2) into Bayes' Theorem (equation 3.3).

$$P(\hat{\imath} \mid \mathbf{x}) = \frac{e^{-\frac{1}{2}(\mathbf{x}-\overline{\mu}^{\hat{\imath}})'\Sigma^{-1}(\mathbf{x}-\overline{\mu}^{\hat{\imath}})}}{\sum_{j=0}^{C-1} e^{-\frac{1}{2}(x-\overline{\mu}^{\hat{j}})'\Sigma^{-1}(x-\overline{\mu}^{\hat{j}})}}$$

The common factor $(2\pi)^{-F/2}|\Sigma|^{-1/2}$ has been canceled from the numerator and denominator. We may further factor out and cancel $e^{-\frac{1}{2}\mathbf{x}'\Sigma^{-1}\mathbf{x}}$ and substitute equation 3.5, yielding

$$P(\hat{\imath} \mid \mathbf{x}) = \frac{e^{v^{\hat{\imath}}(\mathbf{x})}}{\sum_{j=0}^{C-1} e^{v^{\hat{j}}(\mathbf{x})}}$$

Substituting the estimates $v^{\hat{c}}$ for the $\nu^{\hat{c}}(\mathbf{x})$ and incorporating the numerator into the denominator yields an estimate for the probability that i is the correct class for $\mathbf{x}$:

$$\tilde{P}(\hat{\imath} \mid \mathbf{x}) = \frac{1}{\sum_{j=0}^{C-1} e^{(v^{\hat{j}}-v^{\hat{\imath}})}}$$

This value is computed after recognition and compared to a threshold T_P. If below the threshold, instead of accepting g as being in class i, g is rejected. The effect of varying T_P will be evaluated in Chapter 9. There is a tradeoff between wanting to reject as many ambiguous gestures as possible and not wanting to reject unambiguous gestures. Empirically, $T_P = 0.95$ has been found to be a reasonable value for a number of gesture sets (see Section 9.1.2).

The expression for $\tilde{P}(\hat{\imath} \mid \mathbf{x})$ bears out the intuition that if two or more classes evaluate to near the same result the gesture is ambiguous. In such cases the denominator will be significantly larger than unity. Note that the denominator is always at least unity due to the $j = i$ term in the sum. Also note that all the other terms the exponents $(v^{\hat{j}} - v^{\hat{\imath}})$ for $j \neq i$ will always be negative, because $\mathbf{x}$ has been classified as class i by virtue of the fact that $v^{\hat{\imath}} > v^{\hat{j}}$ for $j \neq i$.

$\tilde{P}(\hat{\imath} \mid \mathbf{x})$ may be computed efficiently by using table-lookup for the exponentiation. The table need not be very extensive, since any time $v^{\hat{j}} - v^{\hat{\imath}}$ is sufficiently negative (less than -6, say) the term is negligible. In practice this will be the case for almost all j.

A linear classifier will give no indication if g is an outlier. Indeed, most outliers will be considered unambiguous by the above measure of $\tilde{P}$. To test if g is an outlier, a separate metric is needed to compare g to the typical gesture of class k. An approximation to the Mahalanobis distance [74] works well for this purpose.

Given a gesture with feature vector $\mathbf{x}$, the Mahalanobis distance between $\mathbf{x}$ and class i is defined as

$$\delta^2 = (\mathbf{x} - \overline{\mu}^{\hat{\imath}})'\Sigma^{-1}(\mathbf{x} - \overline{\mu}^{\hat{\imath}})$$

Note that δ^2 is used in the exponent of the multivariate normal probability density function (equation 3.2). It plays the role that $((x-\mu)/\sigma)^2$ plays in the univariate normal distribution: the Mahalanobis distance δ^2 essentially measures the (square of the) number of standard deviations that $\mathbf{x}$ is away from the mean $\overline{\mu}^{\hat{i}}$.

If Σ^{-1} happens to be the identity matrix, the Mahalanobis distance is equivalent to the Euclidean distance. In general, the Mahalanobis distance normalizes the effects of different scales for the different features, since these presumably show up as different magnitudes for the variances s_{ii}, the diagonal elements of the common covariance matrix. The Mahalanobis distance also normalizes away the effect of correlations between pairs of features, the off-diagonal elements of the covariance matrix.

As always, it is only possible to approximate the Mahalanobis distance between a feature vector $\mathbf{x}$ and a class i. Substituting the plug-in estimators for the population statistics and writing out the matrix multiplications explicitly gives

$$d^2 = \sum_{j=1}^{F}\sum_{k=1}^{F} s_{jk}^{-1}(x_j - \overline{f}_j^{\hat{i}})(x_k - \overline{f}_k^{\hat{i}}).$$

In order to reject outliers, compute d^2, an approximation of the Mahalanobis distance from the feature vector $\mathbf{x}$ to its computed class $\hat{i}$. If the distance is greater than a certain threshold T_{d^2} the gesture is rejected. Section 9.1.2 evaluates various settings of T_{d^2}; here it is noted that setting $T_{d^2} = \frac{1}{2}F^2$ is a good compromise between accepting obvious outliers and rejecting reasonable gestures.

Now that the underlying mechanism of rejection has been explained, the question arises as to whether it is desirable to do rejections at all. The answer depends upon the application. In applications with easy to use undo and abort facilities, the reject option should probably be turned off completely. This is because in either failure mode (rejection or misclassification) the user will have to redo the gesture (probably about the same amount of work in both cases) and turning on rejection merely increases the number of gestures that will have to be redone.

In applications in which it is deemed desirable to do rejection, the question arises as to how the interface should behave when a gesture is rejected. The system may prompt the user with an error message, possibly listing the top possibilities for the class (judging from the discriminant functions) and asking the user to pick. Or, the system may choose to ignore the gesture and any subsequent input until the user indicates the end of the interaction. The proper response presumably depends on the application.

3.7 Discussion

One goal of the present research was to enable the implementor of a gesture-based system to produce gesture recognizers without the need to resort to hand-coding. The original plan was to try a number of pattern recognition techniques of increasing complexity until one powerful enough to recognize gestures was found. The author was pleasantly surprised when the first technique he tried, linear discrimination, produced accurate and efficient classifiers.

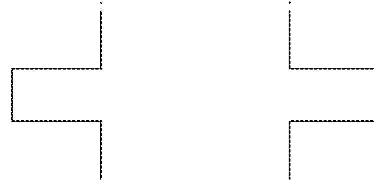


Figure 3.4: Two different gestures with identical feature vectors

The efficiency of linear recognition is a great asset: gestures are recognized virtually instantaneously, and the system scales well. The incremental feature calculation, with each new input point resulting in a bounded (and small) amount of computation, is also essential for efficiency, enabling the system to handle large gestures as efficiently as small ones.

3.7.1 The features

The particular feature set reported on here has worked fine discriminating between the gestures used in three sample applications: a simple drawing program, the uppercase letters in the alphabet, and a simple score editor. Tests using the gesture set of the score editor application are the most significant, since the recognizer was developed and tested on the other two. Chapter 9 studies the effect of training set size and number of classes on the performance of the recognizer. A classifier which recognizes thirty gestures classes had a recognition rate of 96.8% when trained with 100 examples per class, and a rate of 95.6% when trained with 10 examples per class. The misclassifications were largely beyond the control of the recognizer: there were problems using the mouse as a gesturing device and problems using a user process in a non-real-time system (UNIX) to collect the data.

It would be desirable to somehow show that the feature set was adequate for representing differences between all gestures likely to be encountered in practice. The measurements in Chapter 9 show good results on a number of different gestures sets, but are by no means a proof of the adequacy of the features. However, the mapping from gestures (sequences of points) to feature vectors is not one-to-one. In fact, it can easily be demonstrated that there are apparently different gestures that give rise to the same feature vector. Figure 3.4 shows one such pair of gestures. Since none of the features in the feature set depend on the order in which the angles in the gesture are encountered, and the two gestures are alike in every other respect, they have identical feature vectors. Obviously, any classifier based on the current feature set will find it impossible to distinguish between these gestures.

Of course, this particular deficiency of the feature set can be fixed by adding a feature that does depend on the order of the angles. Even then, it would be possible to generate two gestures which have the same angles in the same order, which differ, say, in the segment lengths between the angles, but nontheless give rise to the same feature vector. A new feature could then be added to handle this case, but it seems that there is still no way of being sure that there do not exist two different gestures giving rise to the same feature vector.

Nonetheless, adding features is a good way to deal with gesture sets containing ambiguous

classes. Eventually, the number of features might grow to the point such that the recognizer performs inefficiently; if this happens, one of the algorithms that chooses a good subset of features could be applied [62, 103]. (Though not done in the present work, the contribution of individual features for a given classifier can be found using the statistical techniques of principle components analysis and analysis of variance [74].) However, given the good coverage that can be had with 13 features, 20 features would make it extremely unlikely that grossly different gestures with similar feature vectors would be encountered in practice. Since recognition time is proportional to the number of features, it is clear that a 20 feature recognizer does not entail a significant processing burden on modern hardware, even for large (40 class) gesture sets. There still may be good reason to employ fewer features when possible; for example, to reduce the number of training examples required.

The problem of detecting when a classifier has been trained on ambiguous classes is of great practical significance, since it determines if the classifier will perform poorly. One method is to run the training examples through the classifier, noting how many are classified incorrectly. Unfortunately, this may fail to find ambiguous classes since the classifier is naturally biased toward recognizing its training examples correctly. An alternative is to compute the pairwise Mahalanobis distance between the class means; potentially ambiguous classes will be near each other.

3.7.2 Training considerations

There is a potential problem in the training of classifiers, even when the intended classes are unambiguous. The problem arises when, within a class, the training examples do not have sufficient variability in the features that are irrelevant to the recognition of that class.

For example, consider distinguishing between two classes: (1) a rightward horizontal segment and (2) an upward vertical segment. Suppose all the training examples of the rightward segment class are short, and all those of the upward segment class are long. If the resulting classifier is asked to classify a long rightward segment, there is a significant probability of misclassification.

This is not surprising. Given the training examples, there was no way for classifier to know that being a rightward segment was the important feature of class (1), but that the length of the segment was irrelevant. The same training examples could just as well have been used to indicate that all elements of class (1) are short segments.

The problem is that, by not varying the length of the training examples, the trainer does not give the system significant information to produce the desired classifier. It is not clear what can be done about this problem, except perhaps to impress upon the people doing the training that they need to vary the irrelevant features of a class.

3.7.3 The covariance matrix

An important problem of linear recognition comes from the assumption that the covariance matrices for each class are identical. Consider a classifier meant to distinguish between three gestures classes named **C**, **U**, and **I** (figure 3.5). Examples of class **C** all look like the letter "C", and examples of class **U** all look like the letter "U." Assume that example **C** and **U** gestures are drawn similarly

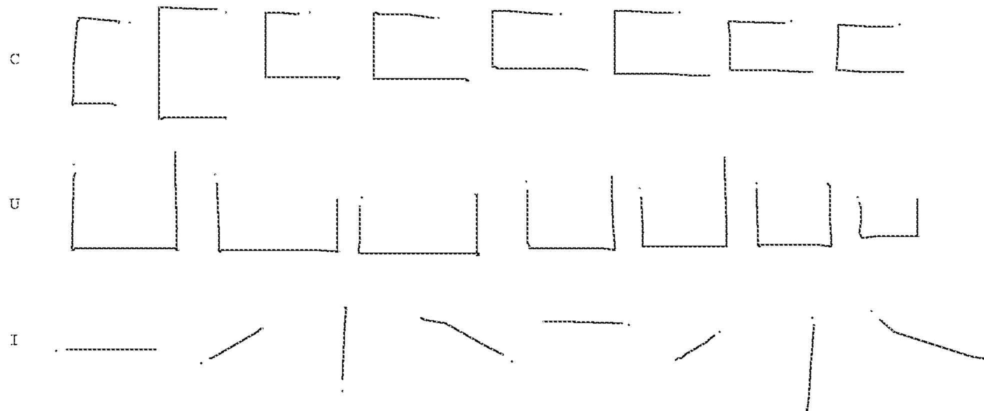


Figure 3.5: A potentially troublesome gesture set

This figure contains examples of three classes: **C**, **U**, *and* **I**. **I** *varies in orientation while* **C** *and* **U** *depend upon orientation to be distinguished. Theoretically, there should be a problem recognizing gestures in this set with the current algorithm, but in practice this has been shown not to be the case.*

except for the initial orientation. Examples of class **I**, however, are strokes which may occur in any initial orientation.

The point of this set of gesture classes is that initial orientation is essential for distinguishing between **C** and **U** gestures, but must be ignored in the case of **I** gestures. This information is contained in the per-class covariance matrices $s_{ij}^{\mathbf{C}}$, $s_{ij}^{\mathbf{U}}$, and $s_{ij}^{\mathbf{I}}$. In particular, consider the variance of the feature f_1, which, for each class c, is proportional to $s_{11}^{\hat{c}}$. Since the initial angle is almost the same for each example C gesture, $s_{11}^{\mathbf{C}}$ will be close to zero. Similarly, $s_{11}^{\mathbf{U}}$ will also be close to zero. However, since the examples of class **I** have different orientations, $s_{11}^{\mathbf{I}}$ will be significantly non-zero.

Unfortunately, the information on the variance of f_1 is lost when the per-class covariance matrix estimates $s_{ij}^{\hat{c}}$ are averaged to give an estimate of the common covariance matrix s_{ij} (equation 3.6). Initially, it was suspected this would cause a problem resulting in significantly lowered recognition rates, but in practice the effect has not been too noticeable. The classifier has no problem distinguishing between the above gestures correctly.

A more extensive test where some gestures vary in size and orientation while others depend on size and orientation to be recognized is presented in Section 9.1.4. The recognition rates achieved show the classifier has no special difficulty handling such gesture sets. Had there been a real problem, the plan was to experiment with improving the linear classifier, say by a few iterations of the perceptron training method [119]. Had this not worked, using a quadratic discriminator (equation 3.4) was another possible area of exploration.

3.8 Conclusion

This chapter discussed how linear statistical pattern recognition techniques can be successfully applied to the problem of classifying single-path gestures. By using these techniques, implementors of gesture-based systems no longer have to write application-specific gesture-recognition code. It is hoped that by making gesture recognizers easier to create and maintain, the promising field of gesture-based systems will be more widely explored in the future.

Chapter 4

Eager Recognition

4.1 Introduction

In Chapter 3, an algorithm for classifying single-path gestures was presented. The algorithm assumes that the entire input gesture is known, *i.e.* that the start and end of the gesture are clearly delineated. For some applications, this restriction is not a problem. For others, however, the need to indicate the end of the gesture makes the user interface more awkward than it need be.

Consider the use of mouse gestures in the GDP drawing editor (Section 1.1). To create a rectangle, the user presses a mouse button at one corner of the rectangle, enters the "L" gesture, stops (while still holding the button), waits for the rectangle to appear, and then positions the other corner. It would be much more natural if the user did not have to stop; *i.e.* if the system recognized the rectangle gesture *while the user was making it*, and then created the rectangle, allowing the user to drag the corner. What began as a gesture changes to a rubberbanding interaction with no explicit signal or timeout.

Another example, mentioned previously, is the manipulation of the image of a knob on the screen. Let us suppose that the knob responds to two gestures: it may be turned or it may be tapped. It would be awkward if the user, in order to turn the knob, needed to first begin to turn the knob (entering the turn gesture), then stop turning it (asking the system to recognize the turn gesture), and then continue turning the knob, now getting feedback from the system (the image of the knob now rotates). It would be better if the system, as soon as enough of the user's gesture has been seen so as to unambiguously indicate her intention of turning the knob, begins to turn the knob.

The author has coined the term *eager recognition* for the recognition of gestures as soon as they are unambiguous. Henry et. al. [52] mention that Artkit, a system similar to GRANDMA, can be used to build applications that perform eager recognition of mouse gestures. There is currently no information published as to how gesture recognition or eager recognition is implemented using Artkit. GloveTalk [34] does something similar in the recognition of DataGlove gestures. GloveTalk attempts to use the deceleration of the hand to indicate that the gesture in progress should be recognized. It utilizes four neural networks: the first recognizes the deceleration, the last three classify the gesture when indicated to do so by the first.

Eager recognition is the automatic recognition of the end of a gesture. For many applications, it

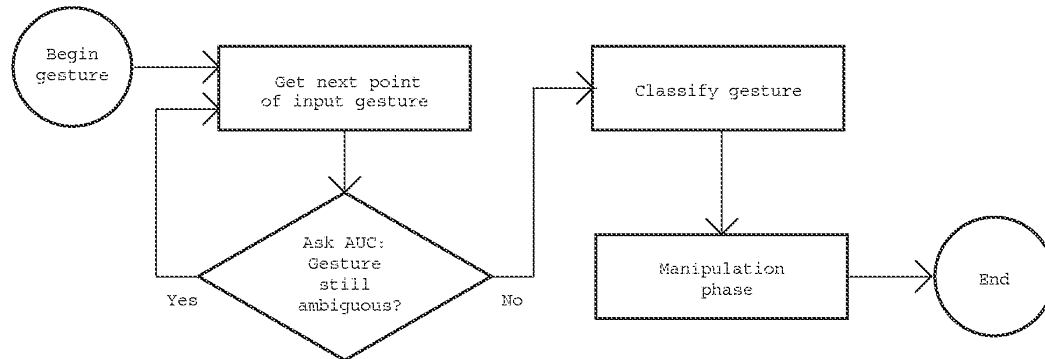


Figure 4.1: Eager recognition overview

Eager recognition works by collecting points until the gesture is unambiguous, at which point the gesture is classified by the techniques of the previous chapter and the manipulation phase is entered. The determination as to whether the gesture seen so far is ambiguous is done by the AUC, i.e. the ambiguous/unambiguous *classifier.*

is not a problem to indicate the start of a gesture explicitly, by pressing a mouse button for example. In the present work, no attempt is made to solve the problem of determining the start of a gesture. Recognizing the start of a gesture automatically is especially important for gesture-based systems that use input devices without any explicit signaling capability (*e.g.* the Polhemus sensor or the DataGlove). For such a device, sudden changes in speed or direction might be used to indicate the start of a gesture. More complex techniques for determining the start of a gesture are outside the scope of this dissertation.

There has been some work on the automatic recognition of the start of gestures. Jackson and Roske-Hofstrand's system [61] recognizes the start of a circling gesture without an explicit indication. In GloveTalk, the user is always gesturing; thus the end of one gesture indicates the start of another. Also related is the automatic segmentation of characters in handwriting systems [125, 13], especially the online recognition of cursive writing [53].

4.2 An Overview of the Algorithm

In order to implement eager recognition, a module is needed that can answer the question "has enough of the gesture being entered been seen so that it may be unambiguously classified?" (figure 4.1). The insight here is to view this as a classification problem: classify a given gesture in progress (called a *subgesture* below) as an ambiguous or unambiguous gesture prefix. This is essentially the approach taken independently in GloveTalk. Here, the recognition techniques developed in the previous chapter are used to build the ambiguous/unambiguous classifier (AUC).

Two main problems need to be solved with this approach. First, training data is needed to train the AUC. Second, the AUC must be powerful enough to accurately discriminate between ambiguous and unambiguous subgestures.

In GloveTalk, the training data problem was solved by explicitly labeling snapshots of a gesture in progress. Each gesture was made up of an average of 47 snapshots (samples of the DataGlove and Polhemus sensors). For each of 638 gestures, the snapshot indicating the time at which the system should recognize the gesture had to be indicated. This is clearly a significant amount of work for the trainer of the system.

In order to avoid such tedious tasks, the present system constructs training examples for the AUC from the gestures used to train the main gesture recognizer. The system considers each subgesture of each example gesture, labels it either ambiguous or not, and uses the labeled subgestures as training data. It seems there is a chicken-and-egg problem here: in order to create the training data, the system needs to perform the very task for which it is trying to create a classifier. However, during the creation of the training data, the system has access to a crucial piece of information that makes the problem tractable: to determine if a given subgesture is ambiguous the system can examine the entire gesture from which the subgesture came.

Once the training data has been created, a classifier must be constructed. In GloveTalk this presented no particular difficulty, for two reasons. There, the classifier was trained to recognize decelerations that, as indicated by the sensor data, were similar between different gesture classes. Also, neural networks with hidden layers are better suited for recognizing classes with non-Gaussian distributions.

In the present system, the training data for the AUC consists of two sets: unambiguous subgestures and ambiguous subgestures. The distribution of feature vectors within the set of unambiguous subgestures will likely be wildly non-Gaussian, since the member subgestures are drawn from many different gesture classes. For example, in GDP the unambiguous delete subgestures are very different from the unambiguous pack gestures, etc., so there will be a multimodal distribution of feature vectors in the unambiguous set. Similarly, the distribution of feature vectors in the ambiguous set will also likely be non-Gaussian. Thus, a linear discriminator of the form developed in the previous chapter will surely not be adequate to discriminate between two classes ambiguous and unambiguous subgestures. What must be done is to turn this two-class problem (ambiguous or unambiguous) into a multi-class problem. This is done by breaking up the ambiguous subgestures into multiple classes, each of which has an approximately normal distribution. The unambiguous subgestures must be similarly partitioned.

The details of the creation of the training data and the construction of the classifier are now presented. First a failed attempt at the algorithm is considered, during which the aforementioned problems were uncovered. Then a working version of the algorithm is presented.

4.3 Incomplete Subgestures

As in the last chapter, we are given a set of C gesture classes, and a number of examples of each class, $g_e^{\hat{c}}$, $0 \leq c < C$, $0 \leq e < E^{\hat{c}}$, where $E^{\hat{c}}$ is the number of examples of class c. The algorithm described in this chapter produces a function $\mathcal{D}$ which when given a subgesture returns a boolean indicating whether the subgesture is unambiguous with respect to the C gesture classes. When the function indicates that the subgesture is unambiguous, the recognition algorithm described in the previous chapter is used to classify the gesture.

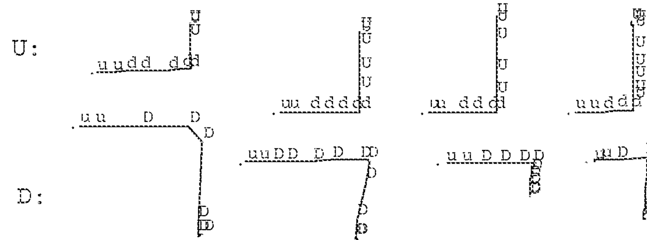


Figure 4.2: Incomplete and complete subgestures of U and D
The character indicates the classification (by the full classifier) of each subgesture. Uppercase characters indicate complete subgestures, meaning that the subgesture and all larger subgestures are correctly classified. Note that along the horizontal segment (where the subgestures are ambiguous) some subgestures are complete while others are not.

The classification algorithm of the previous chapter showed how, given a gesture g, to calculate a feature vector $\mathbf{x}$. A linear discriminator was then used to classify $\mathbf{x}$ as a class c. For much of this chapter, the classifier can be considered to be a function $\mathcal{C}$: $c = \mathcal{C}(g)$. In other words, $\mathcal{C}(g)$ is the class of g as computed by the classifier of Chapter 3.

The function $\mathcal{C}$ was produced from the statistics of the example gestures of each class c, $g_e^{\hat{c}}$. The algorithms described in this chapter work best if only the example gestures that are in fact classified correctly by the computed classifier are used. Thus, in this chapter it is assumed that $\mathcal{C}(g_e^{\hat{c}}) = c$ for all example gestures $g_e^{\hat{c}}$. In practice this is achieved by ignoring those very few training examples that are incorrectly classified by $\mathcal{C}$.

Denote the number of input points in a gesture g as $|g|$, and the particular points as $g_p = (x_p, y_p, t_p)$, $0 \leq p < |g|$. The ith *subgesture* of g, denoted $g[i]$, is defined as a gesture consisting of the first i points of g. Thus, $g[i]_p = g_p$ and $|g[i]| = i$. The subgesture $g[i]$ is simply a prefix of g, and is undefined when $i > |g|$. The term "full gesture" will be used when it is necessary to distinguish the full gesture g from its proper subgestures $g[i]$ for $i < |g|$. The term "full classifier" will be used to refer to $\mathcal{C}$, the classifier for full gestures.

For each example gesture of class c, $g = g_e^{\hat{c}}$, some subgestures $g[i]$ will be classified correctly by the full classifier $\mathcal{C}$, while others likely will not. A subgesture $g[i]$ is termed *complete* with respect to gesture g, if, for all $j, i \leq j < |g|, \mathcal{C}(g[j]) = \mathcal{C}(g)$. The remaining subgestures of g are *incomplete*. A complete subgesture is one which is classified correctly by the full classifier, and all larger subgestures (of the same gesture) are also classified correctly.

Figure 4.2 shows examples of two gestures classes, U and D. Both start with a horizontal segment, but U gestures end with an upward segment, while D gestures end with a downward segment. In this simple example, it is clear that the subgestures which include only the horizontal segment are ambiguous, but subgestures which include the corner are unambiguous. In the figure, each point in the gesture is labeled with a character indicating the classification of the subgesture which ends at the point. An upper case label indicates a complete subgesture, lower case an incomplete subgesture. Notice that incomplete subgestures are all ambiguous, all unambiguous subgestures are complete, but there are complete subgestures that are ambiguous (along the horizontal segment of

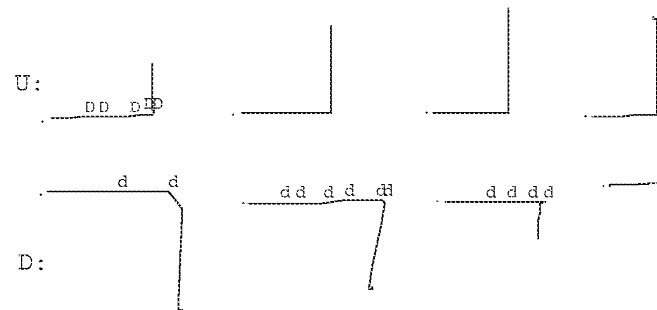


Figure 4.3: A first attempt at determining the ambiguity of subgestures
A two-class classifier was built to distinguish incomplete and complete subgestures, with the hope that those classified as complete are unambiguous and those classified as incomplete are ambiguous. The characters indicate where the resultant classifier differed from its training examples. The horizontal segment of the D *gestures were classified as incomplete (a fortuitous error), but the horizontal segment of the first* U *gesture was classified as complete. The latter is a grave mistake as the gestures are ambiguous along the horizontal segment and it would be premature for the full classifier to attempt to recognize the gesture at such points.*

the D examples).

4.4 A First Attempt

For eager recognition, subgestures that are unambiguous must be recognized as the gesture is being made. As stated above, the approach is to build an AUC, *i.e.* a classifier which distinguishes between ambiguous and unambiguous subgestures. Notice that the set of incomplete and complete subgestures approximate the set of ambiguous and unambiguous subgestures, respectively. The author's first, rather naive attempt at eager recognition was to partition the subgestures of all the example gestures into two classes, incomplete and complete. A linear classifier was then produced using the method described in Chapter 3. This classifier attempts to discriminate between complete and incomplete subgestures. The function $\mathcal{D}(g)$ then simply returns **false** whenever the above classifier reports that g is incomplete, and **true** whenever the classifier claims g is complete.

Figure 4.3 shows the output of the computed classifier for examples of U and D. Points corresponding to subgestures are labeled only when the classifier has made an error, in the sense that the classification does not agree with the training data (shown in figure 4.2). The worst possible error is for the classifier to indicate a complete gesture which happens to still be incomplete, which occurred along the right stroke of the first U gesture.

This approach to eager recognition was not very successful. That it is inadequate was indicated even more strongly by its numerous errors when tried on an example containing six gesture classes. It does however contain the germ of a good idea: that statistical classification may be used to determine if a gesture is ambiguous. A detailed examination of the problems of this attempt is instructive, and leads to a working eager recognition algorithm.

This first attempt at eager recognition has a number of problems:

- The distinction between incomplete and complete subgestures does not exactly correspond with the distinction between ambiguous and unambiguous subgestures. In the U and D example, subgestures consisting only of points along the right stroke are complete for gestures which eventually turn out to be D, and incomplete for gestures that turn out to be U. Yet, these subgestures have essentially identical features. Training a classifier on such conflicting data is bound to give poor results. In the example, as long as the right stroke is in progress the gesture is ambiguous. That it happens to be a complete D gesture is an artifact of the classifier $\mathcal{C}$ (it happens to choose D given only a right stroke).

- All the subgestures of examples were placed in one of only two categories: complete or incomplete. In the case of multiple gesture classes, within each of the two categories the subgestures are likely to form further clusters. For example, the complete U subgestures will cluster together, and be apart from the complete D subgestures. When more gesture classes are used, even more clustering will occur. Thus, the distribution of the complete subgestures is not likely to be normal. Furthermore, it is likely that incomplete subgestures will be more similar to complete gestures of the same class than to incomplete subgestures of other classes. (A similar remark holds for complete subgestures.) It is thus not likely that a linear discriminator will give good results separating complete and incomplete subgestures.

- The classifier, once computed, may make errors. The most severe error is reporting that a gesture is complete when it is in fact still ambiguous. The final classifier must be tuned to avoid such errors, even at the cost of making the recognition process less eager than it otherwise might be.

4.5 Constructing the Recognizer

Based on consideration of the above problems, a four step approach was adopted for the construction of classifiers able to distinguish unambiguous from ambiguous gestures.

Compute complete and incomplete sets.

Partition the example subgestures into $2C$ sets. These sets are named I-c and C-c for each gesture class c. A complete subgesture $g[i]$ is placed in the class C-c, where $c = \mathcal{C}(g[i]) = \mathcal{C}(g)$. An incomplete subgesture $g[i]$ is placed in the class I-c, where $c = \mathcal{C}(g[i])$ (and it is likely that $c \neq \mathcal{C}(g)$). The sets I-c are termed incomplete sets, and the sets C-c, complete sets. Note that the class in each set's name refers to the full classifier's classification of the set's elements. In the case of incomplete subgestures, this is likely not the class of the example gesture of which the subgesture is a prefix.

Figure 4.4 shows pseudocode to perform this step. Figure 4.2, already seen, shows the result of this step, with the subgestures in class I-D labeled d, class I-U labeled u, class C-D labeled D, and class C-U labeled U. The practice of labeling incomplete subgestures with lowercase

```
for c := 0 to C − 1 { /* initialize the 2C sets */
        incomplete_c := ∅ /* This is the set I-c */
        complete_c := ∅ /* This is the set C-c */
}
for c := 0 to C − 1 { /* every class c */
        for e := 0 to E^ĉ − 1 { /* every training example in c */
                p := |g_e^ĉ| /* subgestures, largest to smallest */
                while p > 0 ∧ C(g_e^ĉ[p]) = C(g_e^ĉ) {
                        complete_{C(g_e^ĉ[p])} := complete_{C(g_e^ĉ[p])} ∪ {g_e^ĉ[p]}
                        p := p − 1
                }
                /* Once a subgesture is misrecognized by the full classifier, */
                /* it and its subgestures are all incomplete. */
                while p > 0 {
                        incomplete_{C(g_e^ĉ[p])} := incomplete_{C(g_e^ĉ[p])} ∪ {g_e^ĉ[p]}
                        p := p − 1
                }
        }
}
```

Figure 4.4: Step 1: Computing complete and incomplete sets

letters and complete subgestures with uppercase letters will be continued throughout the chapter.

Move accidentally complete elements.

Measure the distance of each subgesture $g[i]$ in each complete set to the mean of each incomplete set. If $g[i]$ is sufficiently close to one of the incomplete sets, it is removed from its complete set, and placed in the close incomplete set. In this manner, an example subgesture that was accidentally considered complete (such as a right stroke of a D gesture) is grouped together with the other incomplete right strokes (class I-D in this case). Figure 4.5 shows pseudocode to perform this operation.

Quantifying exactly what is meant by "sufficiently close" turned out to be rather difficult. Using the Mahalanobis distance as a metric turns out not to work well if applied naively. The problem is that it depends on the estimated average covariance matrix, which in turn depends upon the covariance matrix of the individual classes. However, some of the classes are malformed, which is why this step of moving accidentally complete elements is necessary in the first place. For example, the C-D class has accidentally complete subgestures in it, so its covariance matrix will indicate large standard deviations in a number of features (total angle, in this case). The effect of using the inverse of this covariance matrix to measure distance is

that large differences between such features will map to small distances. Unfortunately, it is these very features that are needed to decide which subgestures are accidentally complete.

Alternatives exist. The average covariance matrix of the full gesture set (which does not include any subgestures) might be used. It would also be possible to use only the average covariance matrix of the incomplete classes. Or an attempt might be made to scale away the effect of different sized units of the features, and then apply a Euclidean metric. Or, the entire regrouping problem might be approached from a different direction, for example by applying a clustering algorithm to the training data [74]. The first alternative, using the average covariance matrix of the full gesture set (the same one used in the creation of the gesture classifier of Chapter 3) was chosen, since that matrix was easily available, and seems to work.

Once the metric has been chosen (Mahalanobis distance using the covariance matrix of the full gesture set), deciding when to move a subgesture from a complete class to an incomplete class is still difficult. The first method tried was to measure the distance of the subgesture to its current (complete) class, *i.e.* its distance from the mean of its class. The subgesture was moved to the closest incomplete class if that distance was less than the distance to its current class. This resulted in too few moves, as the mean of the complete class was biased since it was computed using some accidentally complete subgestures.

Instead, a threshold is computed, and if the distance of the complete subgesture to an incomplete class is below that threshold, the subgesture is moved. A fixed threshold does not work well, so the threshold is computed as follows: The distance of the mean of each full gesture class to the mean of each incomplete subgesture class is computed, and the minimum found. However, distances less than another threshold, F^2, are not included in the minimum calculation to avoid trouble when an incomplete subgesture looked like a full gesture of a different class. (This is the case if, in addition to U and D, there is a third gesture class consisting simply of a right stroke.) The threshold used is 90% of that minimum.

The complete subgestures of a full gesture were tested for accidental completeness from largest (the full gesture) to smallest. Once a subgesture was determined to be accidentally complete, it, and the remaining (smaller) complete subgestures are moved to the appropriate incomplete classes.

Figure 4.6 shows the classes of the subgestures in the example after the accidentally complete subgestures have been moved. Note that now the incomplete subgestures (lowercase labels) are all ambiguous.

Build the AUC, a classifier which attempts to discriminate between the partition sets.

Now that there is training data containing C complete classes, indicating unambiguous subgestures, and C incomplete classes, indicating ambiguous subgestures, it is a simple matter to run the algorithm in the previous chapter to create a classifier to discriminate between these $2C$ classes. This classifier will be used to compute the function $\mathcal{D}$ as follows: if this classifier places a subgesture s in any incomplete class, $\mathcal{D}(s)$ = **false**, otherwise the s is judged to be

```
for c := 0 to C − 1 {
    ∀ g ∈ complete_c /* each complete subgesture */ {
        m := 0 /* m is the class of the incomplete set closest to g */
        for i := 1 to C − 1 {
            if distance(g, incomplete_i) < distance(g, incomplete_m)
                m := i
        }
        if distance(g, incomplete_m) < threshold {
            complete_c := complete_c − {g}
            incomplete_c := incomplete_c ∪ {g}
        }
    }
}
```

Figure 4.5: Step 2: Moving accidentally complete subgestures

The distance function and threshold value are described in the text. Though not apparent from the above code, the distance function to an incomplete set does not change when elements are added to the set.

in one of the complete classes, in which case $\mathcal{D}(s)$ = **true**. Figure 4.7 shows pseudocode for building this classifier.

Evaluate and tweak the classifier.

It is very important that subgestures not be judged unambiguous wrongly. This is a case where the cost of misclassification is unequal between classes: a subgesture erroneously classified ambiguous will merely cause the recognition not to be as eager as it could be, whereas a subgesture erroneously classified unambiguous will very likely result in the gesture recognizer misclassifying the gesture (since it has not seen enough of it to classify it unambiguously). To avoid this, the constant terms of the evaluation function of the incomplete classes i, w_{i0}, are incremented by a small amount, $\ln(M)$, where M is the relative cost of two kinds of misclassification. A reasonable value is $M = 5$, *i.e.* misclassifications as unambiguous are five times more costly than misclassifications as ambiguous. The effect is to bias the classifier so that it believes that ambiguous gestures are five times more likely than unambiguous gestures, so it is much more likely to choose an ambiguous class when unsure.

Each incomplete subgesture is then tested on the new classifier. Any time such a subgesture is classified as belonging to a complete set (a serious mistake), the constant term of the evaluation function corresponding to the complete set is adjusted automatically (by just enough plus a little more) to keep this from happening.

Figure 4.9 shows the classification by the final classifier of the subgestures in the example. A larger example of eager recognizers is presented in section 9.2.

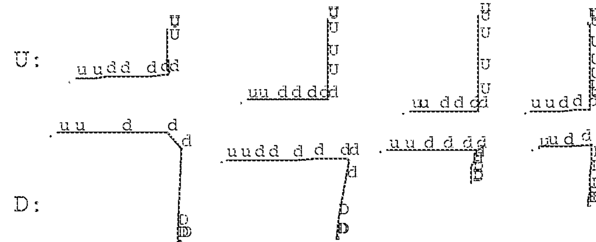


Figure 4.6: Accidentally complete subgestures have been moved
Comparing this to figure 4.2 it can be seen that the subgestures along the horizontal segment of the D *gestures have been made incomplete. Unlike before, after this step all ambiguous subgestures are incomplete.*

```
s := sNewClassifier()
for c := 0 to C − 1 {
        ∀ g ∈ complete_c
                sAddExample(s, g, "C − "c)
        ∀ g ∈ incomplete_c
                sAddExample(s, g, "I − "c)
}
sDoneAdding(s)
```

Figure 4.7: Step 3: Building the AUC
The functions called to build a classifier are `sNewClassifier()`, *which returns a new classifier object,* `sAddExample`, *which adds an example of a class, and* `sDoneAdding`, *called to generate the per-class evaluation functions after all examples have been added. These functions are described in detail in appendix A. The notation* `"C-"`*c indicates the generation a class name by concatenating the string* `"C-"` *with the value of c.*

4.6 Discussion

The algorithm just described will determine whether a given subgesture is ambiguous with respect to a set of full gestures. Presumably, as soon as it is decided that the subgesture is unambiguous it will be passed to the full classifier, which will recognize it, and then up to the application level of the system, which will react accordingly.

How well this eager recognition works depends on a number of things, the most critical being the gesture set itself. It is very easy to design a gesture set that does not lend itself well to eager recognition; for example, there would be no benefit trying to use eager recognition on Buxton's note gestures [21] (figure 2.4). This is because the note gestures for longer notes are subgestures of the note gestures for shorter notes, and thus would always be considered ambiguous by the eager recognizer. Designing a set of gestures for a given application that is both intuitive and amenable to eager recognition is in general a hard problem.

```
/* Add a small constant to the constant term of the evaluation function for */
/* each incomplete class in order to bias the classifier toward erring conservatively. */
for i := 0 to C − 1
      sIncrementConstantTerm(s, "I-"i, ln(M))
/* Make sure that no ambiguous training example is ever classified as complete. */
for i := 0 to C − 1
      ∀ g ∈ incomplete_i
            while ∃ c | sClassify(s, g) = "C − "c
                  sIncrementConstantTerm(s, "C − "c, −ε)
```

Figure 4.8: Step 4: Tweaking the classifier

First, a small constant is added to the constant term of every incomplete class (the ambiguous subgestures), to bias the classifier toward being conservative, rather than eager. Then every ambiguous subgestures is classified, and if any is accidentally classified as complete, the constant term of the evaluation function for that complete class is adjusted to avoid this.

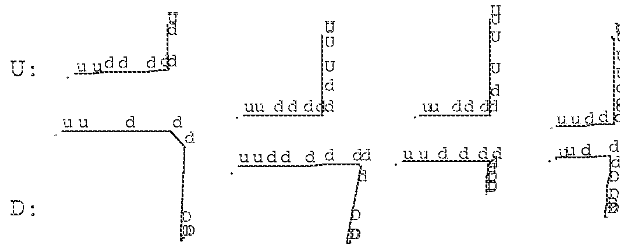


Figure 4.9: Classification of subgestures of U and D

This shows the results of running the AUC on the training examples. As can been seen, the AUC performs conservatively, never indicating that a subgesture is unambiguous when it is not, but sometimes indicating ambiguity of an unambiguous subgesture.

The training of the eager recognizer is between one and two orders of magnitude more costly than the training of the corresponding classifier for full gestures. This is largely due to the number of training examples: each full gesture example typically gives rise to ten or twenty subgestures. The amount of processing per training example is also large. In addition to computing the feature vector of each training example, a number of passes must be made over the training data: first to classify the subgestures as incomplete or complete, then to move the accidentally complete subgestures, again to build the AUC, and again to ensure the AUC is not over-eager. While a full classifier takes less than a second to train, the eager recognizer might take a substantial portion of a minute, making it less satisfying to experiment with interactively. As will be seen (Chapter 7), a full classifier may be trained the first time a user gestures at a display object. One possibility would be to use the full classifier (with no eagerness) while training the AUC in the background, activating eager recognition when ready.

The running time for the eager recognizer is also more costly than the full classifier, though not prohibitively so. A feature vector needs to be calculated for every input point; this eliminates any benefit that using auxiliary features (Section 3.3) might have bought. Of course, the AUC needs to be run at every data point; this takes about $2CF$ multiply-adds (since the AUC has $2C$ classes). Since input points do not usually come faster than one every 30 milliseconds, and $2CF$ is typically at most 1000, this computational load is not usually a problem for today's typical workstation class machine. In the current system, the multiply-adds are done in floating point, though this is probably not necessary for the recognition to work well.

One slight defect of the algorithm used to construct the AUC is that it relies totally upon the full classifier. In particular, a subgesture will never be considered unambiguous unless it is classified correctly by the full classifier. To see where this might be suboptimal, consider a full classifier that recognizes two classes, GDP's single segment line gesture and three segment delete gesture. The full classifier would likely classify any subgesture that is the initial segment of a delete as a line. It *may* also classify some two segment subgestures of delete as line gestures, even though the presence of two segments implies the gesture is unambiguously delete. The resulting eager recognizer will then not be as eager as possible, in that it will not classify the input gesture as unambiguously delete immediately after the second segment of the gesture is begun.

Two classifiers are used for eager recognition: the AUC, which decides when a subgesture is unambiguous, and the full classifier, which classifies the unambiguous subgesture. It may seem odd to use two classifiers given the implementation of the AUC, in which a subgesture is not only classified as unambiguous, but unambiguously in a given class (*i.e.* classified as C-c for some c). Why not just return a classification of c without bothering to query the full classifier? There are two main reasons. First, the full classifier, having only C classes to discriminate between, will perform better than the AUC and its $2C$ classes. Second, the final tweaking step of the AUC adjusts constant terms to assure that ambiguous gestures are never classified as unambiguous, but makes no attempt to assure that when classified as unambiguously c, c is the correct class. The adjustment of the constant terms typically degrades the AUC in the sense that it makes it more likely that c will be incorrect.

It is likely that within a decade it will be practical for neural networks to be used for gesture recognition. When this occurs, the part of this chapter concerned with building a $2C$ class linear classifier will be obsolete, since a two-class neural network could presumably do the same job. However, the part of the chapter which shows how to construct training examples for the classifier from the full gestures will still be useful, since it eliminates the hand labeling that otherwise might be necessary.

4.7 Conclusion

An eager recognizer is able to classify a gesture as soon as enough of the gesture has been seen to conclude that the gesture is unambiguous. This chapter presents an algorithm for the automatic construction of eager recognizers for single-path gestures from examples of the full gestures. It is hoped that such an algorithm will make gesture-based systems more natural to use.

Chapter 5

Multi-Path Gesture Recognition

Chapters 3 and 4 discussed the recognition of single-path gestures such as those made with a mouse or stylus. This chapter addresses the problem of recognizing multi-path gestures, *e.g.* those made using an input device, such as the DataGlove, capable of tracking the paths of multiple fingertips. It is assumed that the start and end of the multi-path gesture are known. Eager recognition of multi-path gestures has been left for future work.

The particular input device used to test the ideas in this chapter is the Sensor Frame. The Sensor Frame, as discussed in Section 2.1, is a frame which is mounted on a CRT display. The particular Sensor Frame used was mounted on the display of a Silicon Graphics IRIS Personal Workstation. The Frame detects the XY positions of up to three fingertips in a plane approximately one half inch in front of the display.

By defining the problem as "multiple-path gesture recognition", it is quite natural to attempt to apply algorithms for single-path gesture recognition (*e.g.* those developed in Chapter 3). Indeed, the recognition algorithm described in this chapter combines information culled from a number of single-path classifiers, and a "global feature" classifier in order to classify a multiple-path gesture. Before the particular algorithm is discussed, the issue of mapping the raw data returned from the particular input sensors into a form suitable for processing by the recognition algorithm must be addressed. For the Sensor Frame, this processing consisted of two stages, path tracking and path sorting.

5.1 Path Tracking

The Sensor Frame, as it currently operates, delivers the X and Y coordinates of all fingers in its plane of view each time it is polled, at a maximum rate of 30 snapshots per second. No other information is supplied; in particular the correspondence between fingers in the current and the previous snapshots is not communicated. For example, when the previous snapshot indicated one finger and the current snapshot two, it is left to the host program to determine which of the two fingers (if any) is the same finger as the previously seen one, and which has just entered the field of view. Similarly, if both the previous and current snapshots indicate two fingers, the host program must determine which finger in the current snapshot is the same as the first finger in the previous snapshot, and so on. This

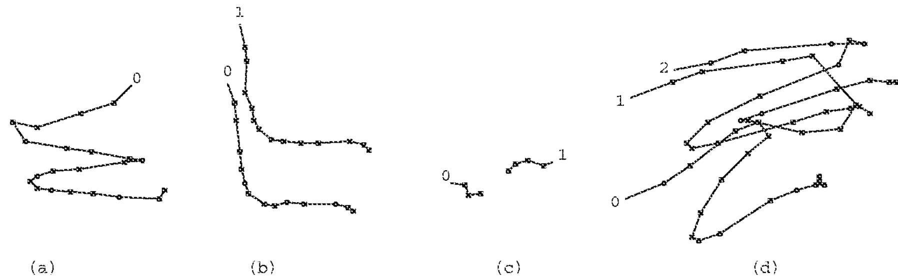


Figure 5.1: Some multi-path gestures

Shown are some MDP gestures made with a Sensor Frame. The start of each path is labeled with a path index, indicating the path's position in a canonical ordering. Gesture (a) is MDP's edit *gesture, an "E" made with a single finger. Gesture (b), parallel "L"s, is two finger* parallelogram *gesture, (c) is MDP's two finger* pinch *gesture (used for moving objects), and (d) is MDP's three finger* undo *gesture, three parallel "Z"s. The finger motions were smooth, and some noise due to the Sensor Frame's position detection can be seen in the examples.*

problem is known as *path tracking*, since it groups the raw input data into a number of paths which exist over time, each path having a definite beginning and end.

The path tracking algorithm used is quite straightforward. When a snapshot is first read, a triangular distance matrix, containing the Euclidean distance squared between each finger in the current snapshot and each in the previous, is computed. Then, for each possible mapping between current and previous fingers, an error metric, consisting of the sum of the squared distances between corresponding fingers, is calculated. The mapping with the smallest error metric is then chosen.

For efficiency, for each possible number of fingers in the previous snapshot and the current snapshot, a list of all the possible mappings are precomputed. Since the Sensor Frame detects from zero to three fingers, only 16 lists are needed. When the symmetry between the previous and current snapshots is considered, only eight lists are needed.

The low level tracking software labels each finger position with a *path identifier*. When there are no fingers in the Sensor Frame's field of view, the `next_path_identifier` variable is set to zero. A finger in the current snapshot which was not in the previous snapshot (as indicated by the chosen mapping) has its path identifier set to the value of `next_path_identifier` which is then incremented. It is thus possible for a single finger to generate multiple paths, since it will be assigned a new path identifier each time it leaves and reenters the field of view of the Sensor Frame, and those identifiers will increase as long as another finger remains in the field of view of the Frame.

The simple tracking algorithm described here was found to work very well. The anticipated problem of mistracking when finger paths crossed did not arrive very often in practice. (This was partly because all gestures were made with the fingers of a single hand, making it awkward for finger paths to cross.) Enhancements, such as using the velocity and the acceleration of each finger in the previous snapshot to predict where it is expected in the current snapshot, were not needed. Examples of the tracking algorithm in operation are shown in figure 5.1. In the figure, the start of

each path is labeled with its path index (as defined in the following section), and the points in the path are connected by line segments. Figure 5.1d shows an uncommon case where the path tracking algorithm failed, causing paths 1 and 2 to be switched.

5.2 Path Sorting

The multi-path recognition algorithm, to be described below, works by classifying the first path in the gesture, then the second, and so on, then combining the results to classify the entire gesture. It would be possible to use a single classifier to classify all the paths; this option is discussed in Section 5.7. However, since classifiers tend to work better with fewer classes, it makes sense to create multiple classifiers, one for the first path of the gesture, one for the second, and so on. This however raises the question of which path in the gesture is the first path, which is the second, etc. This is the *path sorting* problem, and the result of this sorting assigns a number to each path called its *path index*.

The most important feature of a path sorting technique is consistency. Between similar multi-path gestures, it is essential that corresponding paths have the same index. Note that the path identifiers, discussed in the previous section, are not adequate for this purpose, since they are assigned in the order that the paths first appear. Consider, for example, a "pinching" gesture, in which the thumb and forefinger of the right hand are held apart horizontally and then brought together, the thumb moving right while the forefinger moves left. Using the Sensor Frame, the thumb path might be assigned path identifier zero in one pinching gesture, since it entered the view plane of the Frame first, but assigned path identifier one in another pinching gesture since in this case it entered the view plane a fraction of a second after the forefinger. In order for multi-path gesture recognition using of multiple classifiers to give good results, it is necessary that the all thumb motions be sent to the same classifier for training and recognition, thus using path identifiers as path indices would not give good results.

For multi-path input devices which are actually attached to the hand or body, such as the DataGlove, there is no problem determining which path corresponds to which finger. Thus, it would be possible to build one classifier for thumb paths, another for forefinger paths, etc. The characteristics of the device are such that the question of path sorting does not arise.

However, the Sensor Frame (and multifinger tablets) cannot tell which of the fingers is the thumb, which is the forefinger, and so on. Thus there is no *a priori* solution to the path sorting. The solution adopted here was to impose an ordering relation between paths. The consistency property is required of this ordering relation: the ordering of corresponding paths in similar gestures must be the same.

The primary ordering criterion used was the path starting time. However, to avoid the aforementioned timing problem, two paths which start within 200 milliseconds are considered simultaneous, and the secondary ordering criteria is used. A path which starts more than 200 msec before another path will be considered "less than" the other path, and show up before the other path in the sorting.

The secondary ordering criterion is the initial x coordinate. There is a window of 150 Sensor Frame length units (about one inch) within which two paths will be considered to start at the same x coordinate, causing the tertiary ordering criterion to be applied. Outside this window, the path with

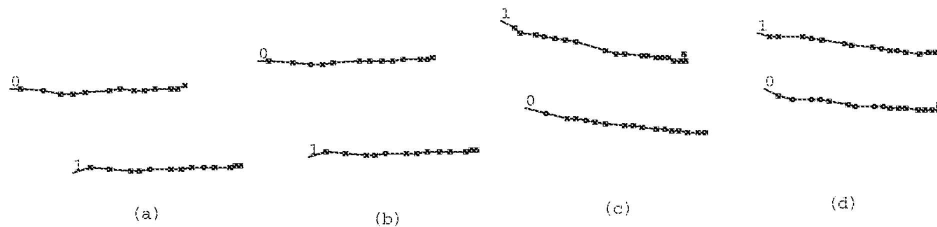


Figure 5.2: Inconsistencies in path sorting

The intention of the path sorting is that corresponding paths in two similar gestures should have the same path index. Here are four similar gestures for which this does not hold: between (b) and (c) the path sorting has changed.

the smaller initial x coordinate will appear before the other path in the sorting (assuming apparent simultaneity).

The tertiary ordering criterion is the initial y coordinate. Again, a window of 150 Sensor Frame length units is applied. Outside this window, the path whose y coordinate is less will appear earlier in the path ordering. Finally, if both the initial x and y coordinate differ by less than 150 units, the coordinate whose difference is the largest is used for ordering, and the path whose coordinate is smaller appears earlier in the path ordering.

Figure 5.2 shows the sorting for some multi-path gestures by labeling the start of each path with its index. Note that the consistency criteria is not maintained between panels (b) and (c), since the "corresponding" paths in the two gestures have different indices. The order of the paths in (b) was determined by the secondary ordering criterion (since the paths began almost simultaneously), while the ordering in (c) was determined by the tertiary ordering criterion (since the paths began simultaneously and had close x coordinates). Generally, *any* set of ordering rules which depend solely on the initial point of each path can be made to generate inconsistent sortings.

In practice, the possibility of inconsistencies has not been much of a problem. The ordering rules are set up so as to be stable for near-vertical and near horizontal finger configurations; they become unstable when the angle between (the initial points of) two fingers causes the 150 unit threshold to be crossed.[1] Knowing this makes it easy to design gesture sets with consistent path orderings. A more robust solution might be to compute a path ordering relation based on the actual gestures used to train the system.

As stated above, some multiple finger sensing devices, such as the DataGlove, do not require any path sorting. To use the DataGlove as input to the multi-path gesture recognizer described below, one approach that could be taken is to compute the paths (in three-space over time) of each fingertip, using the measured angles of the various hand joints. This will result in five sorted paths (one for each finger) which would be suitable as input into the multi-path recognition algorithm. (Of course, the lack of explicit signaling in the DataGlove still leaves the problem of determining the start and

[1] In retrospect, the 150 unit windows make the sorting more complicated then it need be. Using the coordinate whose difference is the largest (for simultaneous paths) makes the algorithm more predictable: it will become inconsistent when the initial points of two paths form an angle close to $-45°$ from the horizontal.

end of the gesture.)

5.3 Multi-path Recognition

Like the single path recognizers described in Chapter 3, the multi-path recognizer is trained by specifying a number of examples for each gesture class. The recognizer consists of a number of single-path classifiers, and a global feature classifier. These classifiers all use the statistical classification algorithm developed in Chapter 3. The differences are mainly in the sets of features used, as described in Section 5.5.

Each single-path classifier discriminates between gestures of a particular sorting index. Thus, there is a classifier for the first path of a gesture, another for the second path, and another for the third path. (The current implementation ignores all paths beyond the third, although it takes the actual number of paths into account.) When a multi-path gesture is presented to the system for classification, the paths are sorted (as described above) and the first path is classified using the first path classifier, and so on, resulting in a sequence of single-path classes.

The sequence of path classes which results is then submitted to a *decision tree*. The root node of the tree has slots pointing to subnodes for each possible class returned by the first path classifier. The subnode corresponding to the class of the first path is chosen. This node has slots pointing to subnodes for each possible class returned by the second path classifier. Some of these slots may be null, indicating that there is no expected gesture whose first and second path classes are the ones computed. In this case the gesture is rejected. Otherwise, the subnode corresponding to the class of the second path is chosen. The process is repeated for the third path class, if any.

Once the entire sequence of path classes is considered there are three possibilities. If the sequence was unexpected, the multi-path gesture is rejected since no node corresponding to this sequence exists in the decision tree. If the node does exist, the multi-path classification may be unambiguous, meaning only one multi-class gesture corresponds to this particular sequence of single-path classes. Or, there may be a number of multi-path gestures which correspond to this sequence of path classes. In this case, a global feature vector (one which encompasses information about all paths) is computed, and then classified by the global feature classifier. This class is used to choose a further subnode in the decision tree, which will result in the multi-path gesture either being classified individually or rejected. The intent is that, if needed, the global feature class is essentially appended to a sequence of path classes; some care is thus necessary to insure that the global feature classes are not confused with path classes.

Figure 5.3 shows an example of the use of a decision tree to classify multi-path gestures. The multi-path classifier recognizes four classes. Each class is composed of two paths. There are only two possible classes for the first path (path 0), since classes P, Q, and S all have similar first paths. Similarly, Q and S have similar second paths, so there are only three distinct possibilities for path 1. Since Q and S have identical path components, the global classifier is used to discriminate between these two, adding another level in the decision tree. The classification of the example input is indicated by dotted lines.