Technical Report

UCAM-CL-TR-330
ISSN 1476-2986

Number 330

UNIVERSITY OF CAMBRIDGE

Computer Laboratory

Interacting with paper on the DigitalDesk

Pierre David Wellner

March 1994

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

http://www.cl.cam.ac.uk/

This technical report is based on a dissertation submitted October 1993 by the author for the degree of Doctor of Philosophy to the University of Cambridge, Clare Hall.

Technical reports published by the University of Cambridge Computer Laboratory are freely available via the Internet:

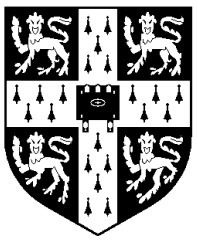
http://www.cl.cam.ac.uk/TechReports/

ISSN 1476-2986

Interacting with Paper on the DigitalDesk

Pierre David Wellner

Clare Hall



University of Cambridge Computer Laboratory

A dissertation submitted for the degree of Doctor of Philosophy

October, 1993

To Linda

Abstract

In the 1970's Xerox PARC developed the "desktop metaphor," which made computers easy to use by making them look and act like ordinary desks and paper. This led visionaries to predict the "paperless office" would dominate within a few years, but the trouble with this prediction is that people like paper too much. It is portable, tactile, universally accepted, and easier to read than a screen. Today, we continue to use paper, and computers produce more of it than they replace.

Instead of trying to use computers to replace paper, the DigitalDesk takes the opposite approach. It keeps the paper, but uses computers to make it more powerful. It provides a Computer Augmented Environment for paper.

The DigitalDesk is built around an ordinary physical desk and can be used as such, but it has extra capabilities. A video camera is mounted above the desk, pointing down at the work surface. This camera's output is fed through a system that can detect where the user is pointing, and it can read documents that are placed on the desk. A computer-driven electronic projector is also mounted above the desk, allowing the system to project electronic objects onto the work surface and onto real paper documents — something that can't be done with flat display panels or rear-projection. The system is called DigitalDesk because it allows pointing with the fingers.

Several applications have been prototyped on the DigitalDesk. The first was a calculator where a sheet of paper such as an annual report can be placed on the desk allowing the user to point at numbers with a finger or pen. The camera reads the numbers off the paper, recognizes them, and enters them into the display for further calculations. Another is a translation system which allows users to point at unfamiliar French words to get their English definitions projected down next to the paper. A third is a paper-based paint program (PaperPaint) that allows users to sketch on paper using traditional tools, but also be able to select and paste these sketches with the camera and projector to create merged paper and electronic documents. A fourth application is the DoubleDigitalDesk, which allows remote colleagues to "share" their desks, look at each other's paper documents and sketch on them remotely.

This dissertation introduces the concept of Computer Augmented Environments, describes the DigitalDesk and applications for it, and discusses some of the key implementation issues that need to be addressed to make this system work. It describes a toolkit for building DigitalDesk applications, and it concludes with some more ideas for future work.

Preface

Except where otherwise stated in the text, this dissertation is the result of my own work and is not the outcome of work done in collaboration.

This dissertation is not substantially the same as any that I have submitted for a degree or diploma or other qualification at any other university.

No part of this dissertation has already been, or is being currently submitted for any such degree, diploma or other qualification.

Acknowledgments

I am grateful for the generous support of Rank Xerox Cambridge EuroPARC. This dissertation is in many ways a product of EuroPARC's excellent and unique work environment. I also thank the University of Cambridge Computer Laboratory for access to its people and facilities.

I thank my supervisor, Peter Robinson for his encouragement, insight, and support, and the many other colleagues and friends who have contributed inspiration, help getting systems to work, and insightful comments. The following deserve special mention: Dan Bloomberg, Stu Card, Kathy Carter, Ian Daniel, Mike Flynn, Steve Freeman, Quentin Stafford-Fraser, Gifford Louie, Bill Gaver, Austin Henderson, Mik Lamming, Wendy Mackay, Allan MacLean, Linda Malgeri, Mike Molloy, Tom Moran, William Newman, Z Smith, Roy Want, Mark Weiser, Marcel Wellner, and Alex Zbyslaw.

Contents

Chapter 1
Introduction

We interact with documents in two separate worlds: the electronic world of the workstation, and the physical world of the desk. Each world has advantages and constraints that lead us to choose one or the other for particular tasks (see Figure 1-1).*

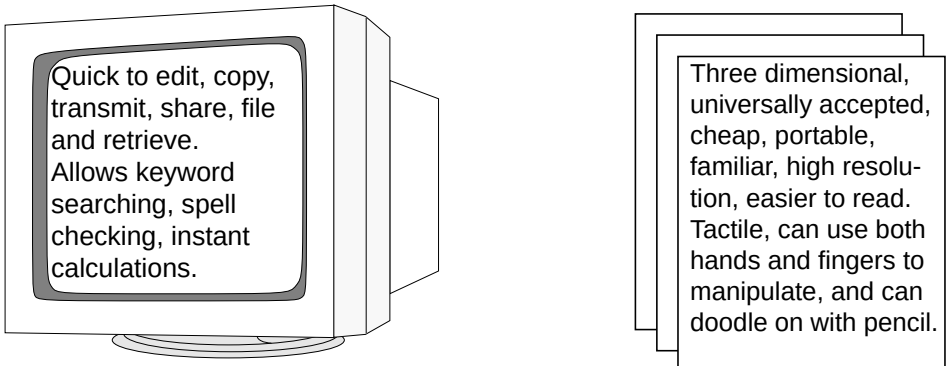


Figure 1-1: Electronic and paper documents.

At one time it seemed that paper might become obsolete, and visionaries predicted the “paperless office” would dominate within a few years. But the trouble is people *like* paper. According to some studies, paper in the office has increased by a factor of six since 1970, and is now growing at 20% annually [Seyb92]. Like electronic documents, paper has properties that people just cannot seem to give up, making it “resilient” in the face of computer-based alternatives [Luff92].

The dual desk

As a result we have two desks: one for paper pushing and the other for pixel pushing. Although activities on the two desks are often related, the two are very separate. They have different interaction styles, different functionality, and they are isolated from each other.

* Many of the figures in this dissertation as well as some of the text have previously been published in [Well93b].

Different interaction styles

The physical desk, paper documents, and the tools we use to manipulate them have evolved over the course of thousands of years to match our motor and perceptual abilities. The ways we physically interact with electronic documents is limited compared with how we interact with paper, pencils, erasers, rulers, and other traditional tools on the desk [Hews90]. When interacting with objects in the physical world, we take advantage of natural skills developed over our lifetimes. We use our fingers, arms, 3D vision, ears and kinesthetic memory to manipulate multiple objects simultaneously, and we hardly think about how we do this because the skills are embedded so deeply into our minds and bodies. Workstations do not take advantage of this, so the skills and habits acquired to master one desk are of very little use for mastering the other.

Different functionality

Paper documents have no direct access to the spellchecking, database queries, and spreadsheet calculations available on electronic workstations. Electronic documents, on the other hand, lack the portability, tangibility, and universal acceptance of paper. Reading from a screen is slower than reading paper [Fren90, Hans88], and this is particularly noticeable with longer documents. If the two desks were better integrated, then this difference in functionality would matter less because users could easily move back and forth between paper and electronic media. As it stands, however, there is often a penalty for choosing the "wrong" medium at the outset.

Lack of integration

Integration between the physical and electronic desks is limited. Printers provide a way out of the electronic desk, and scanners provide a way in. Although desktop printers and hand-held scanners exist today, these devices are not very interactive. Instead, they are primarily used in batch mode to move entire documents in or out of the electronic desktop. This process is inconvenient; for example, Wang Freestyle (a classic paperless office system) was sometimes set up to coexist with partially paper-based processes. A key factor found necessary for successful adoption of the system was to minimize the printing and scanning required. Too much of these tasks would cause the process to revert entirely back to paper [Fran91].

Aiming for the best of both

Choosing to interact with a document in one world generally means forgoing the advantages of the other. A great challenge to office system designers is to provide the best of both, and this has long been a goal of research in human-computer interaction.

The classic approach to this problem is represented by the "desktop metaphor," developed in the 1970's [Smit82, John85, John89]. This approach allows people to use *direct manipulation* [Shne83] of virtual objects on the electronic desktop in order to accomplish their tasks. By making electronic workstations analogous to the physical desk, users can take advantage of their knowledge of the physical

world, the computer becomes more familiar, and less learning is required. This approach has remained successful, and today electronic documents are still gaining more and more properties of physical documents: *e.g.* high resolution colour, portability, pen-based interaction, *etc*. (see for example [Bles88] and [Walt89]).

Where the physical world has an advantageous property over the computer, human computer interaction researchers tend to try and find a way of enhancing the computer with that desired property. Not only are desktops and paper documents put into the computer, but many other things as well. This tendency, taken to an extreme, is *Virtual Reality* (VR), where users abandon the real world to be completely surrounded by the computer [Rhei91] using, for example, 3D head-mounted displays and data gloves. Enthusiasts of this approach believe that all constraints of the real world can be overcome in VR, and physical tools can be made obsolete by more flexible, virtual alternatives. A weakness of data gloves and most other computer input devices, however, is that they provide no tactile feedback when manipulating virtual objects. This "kinesthetic contact" is an important channel through which we experience the world [Kay91]. It is possible to construct input devices that provide tactile feedback for manipulating specific virtual objects [Broo90, Mins90, Buxt85], but this is difficult to do in general, so kinesthetic sensations in virtual reality are usually lacking.

The rest of this chapter introduces an alternative approach that aims for the best of both worlds: *Computer Augmented Environments*. It then describes some examples of work in the area and discusses how this approach could be used to augment paper. The rest of the dissertation describes the DigitalDesk, a computer augmented environment for paper that is built around an ordinary desk with overhead video cameras and a projector.

Computer Augmented Environments[*]

We live in a complex world, filled with myriad objects, tools, toys, and people. Our lives are spent in diverse interaction with this environment. Yet, for the most part, our computing takes place sitting in front of, and staring at, a single glowing screen attached to an array of buttons and a mouse. Our different tasks are assigned to homogeneous overlapping windows. From the isolation of our workstations we try to interact with our surrounding environment, but the two worlds have little in common. How can we escape from the computer screen and bring these two worlds together?

One view of the future that has captured the popular imagination is Virtual Reality. VR allows us to escape from the computer screen by letting us use our whole bodies and a rich variety of (virtual) objects to interact with the computer, replacing the physical world with a computer-generated one. The result is very useful for purposes such as visualization and simulation, but of course, the artificial world is much simpler than the real world; it has lower resolution, leaves out details, and is limited in its behaviour and extent. For certain kinds of enter-

[*] Much of this section is taken from [Well93a] (coauthored with Rich Gold and Wendy Mackay).

tainment, and for tasks like learning to land an aeroplane in a blizzard, the VR approach is invaluable. But for helping us with everyday tasks, VR — even more than the workstation — cuts us off and excludes us from the world in which we live, work, and play.

Another view of the future of computing is emerging that takes the opposite approach from Virtual Reality. Instead of using computers to enclose people in an artificial world, we can use computers to augment objects in the real world. We can make the environment sensitive with infra-red, optical, sound, video, heat, motion and light detectors, and we can make the environment react to people's needs by updating displays, activating motors, storing data, driving actuators, controls and valves. With see-through displays and projectors, we can create spaces in which everyday objects gain electronic properties without losing their familiar physical properties. Computer Augmented Environments (CAEs) merge electronic systems into the physical world instead of attempting to replace it. Our everyday environment is an integral part of these systems; it continues to work as expected, but with new integrated computer functionality.

Computer Augmented Environments emerge from the confluence of a number of disciplines. Recent work has been called names such as "Ubiquitous Computing" and "Augmented Reality." Although the technologies differ, they are united in a common philosophy: the primacy of the physical world and the construction of appropriate tools that enhance our daily activities.

Example work in computer augmented environments

Ken Knowlton

One early system that combined the flexibility of a computer-generated display with the "tactile and kinesthetic feel" of physical buttons was developed by Ken Knowlton at Bell Labs [Know77]. In this system, a semi-transparent mirror and computer monitor were placed above a keyboard in such a way that computer-generated labels appear superimposed on the physical keys (see Figure 1-2).

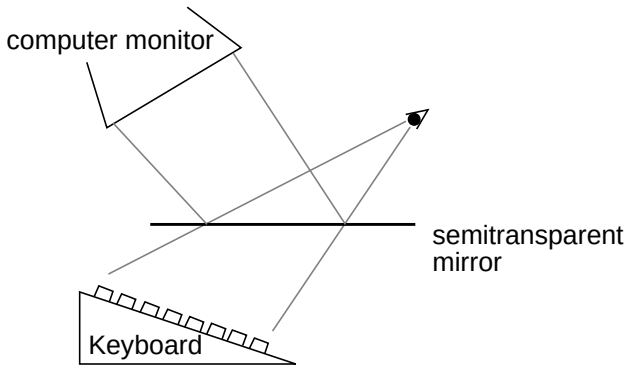


Figure 1-2: Knowlton's system.

The labels would change to show what operations were possible from moment to moment. The system even made it possible to perform operations on text being edited by pressing the keys associated with the text. Although this system

successfully combines the tactile buttons with screen-based graphics, it does not augment anything except the keyboard. In Knowlton's paper he does not mention calibration as an issue, but it probably was not trivial to properly align the graphics with the buttons.

VideoDraw

VideoDraw and VideoWhiteboard [Tang91a, Tang91b] are shared drawing systems that combine the physical and electronic in a way that provides a sort of computer augmented environment. In both these systems, users draw on a video screen with ordinary whiteboard pens while their marks are transmitted to their collaborator's screens (see Figure 1-3 and Figure 1-4). One advantage of these systems over workstation-based shared drawing tools is that writing, erasing, and drawing different colours uses the same "interaction techniques" and "input devices" as an ordinary physical whiteboard. The system adds electronic functionality in a way that is compatible with a real whiteboard.

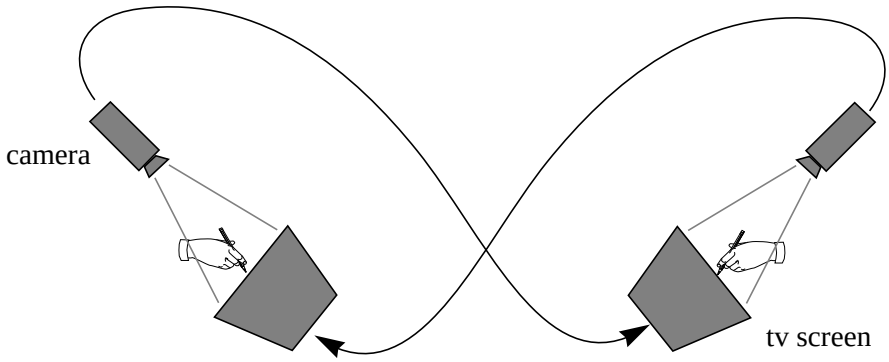


Figure 1-3: VideoDraw

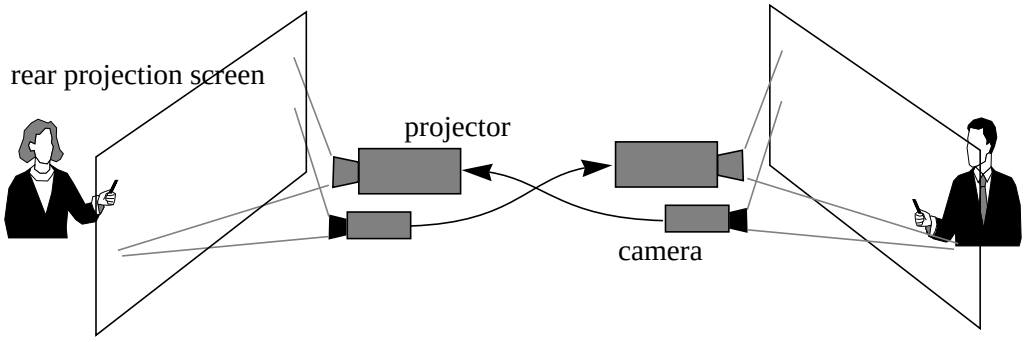


Figure 1-4: VideoWhiteboard

There are two problems with these systems, however. One is the difficulty in setting them up. It takes a lot of fiddling about to get the cameras calibrated with the monitors or projectors in such a way that images are aligned properly and video feedback is tolerable, then once set up they cannot be moved. Another restriction is that although the pens are standard, the writing surface is not. It is

impossible to use of an ordinary whiteboard, and it is impossible to share and draw on ordinary paper documents.

Myron Krueger

Myron Krueger is one of the pioneers of virtual reality, having done his work in the 70's and early 80's before VR became trendy. His philosophy has always been more akin to that of Computer Augmented Environments rather than the completely immersing VR, but his priority is creating works of art more than creating useful tools. He likes to say he hates being "encumbered" by devices. He will not wear a wrist watch, let alone such devices as data gloves and head-mounted displays (HMDs), so his systems rely on sensors such as video cameras that users do not have to wear. He calls his field "Artificial Reality" [Krue83, Krue91], and probably his most important work is VIDEOPLACE [Krue85]. In this environment "participants" (Krueger uses this term instead of "users") can see a real-time coloured outline of their bodies on a video screen, along with other objects with which they can interact. These other objects can be computer generated, or they can be the outlines of other people's bodies or hands that may be located at another site. In some versions of VIDEOPLACE, one participant's body is shrunk to a tiny size relative to the other, and swung from a thread hanging on another participant's finger. This allows the finger to swing and even throw the tiny person. An animated creature called "critter" crawls along the edge of a participant, and climbing to the top where it "does a little jig of celebration." Krueger has also made a desk-based version of this environment (called VIDEODESK) with which a participant can interact with objects on the screen with the silhouettes of his or her hands and fingers.

Although users are unencumbered in VIDEOPLACE, interaction does not truly happen in the "real" world. Users see their silhouettes interact with virtual objects on the screen rather than truly interacting with real world devices and objects such as paper.

Mandalla

The Mandalla system [Vinc90] is essentially a commercial version of VIDEOPLACE, but simpler. A number of active areas can be defined on a video scene, and the user can touch them with his or her silhouette to trigger programmable events of various types. One popular way to use it is to control a MIDI synthesizer. This makes an impressive performance instrument, because it combines a video show with music and a form of dancing.

Team Workstation

The TeamWorkStation [Ishi91] is another example of a system that "fuses" the physical and electronic desks. It combines computer-generated images with video images of the physical desk, and like the DigitalDesk, one of its goals is to remove the "seam" between traditional and electronic media. With this system, a calligraphy instructor can draw using real brushes and paint, and the work appears on a video monitor superimposed with the computer-generated work of a student. This system, therefore merges the real and virtual worlds in the (virtual) screen rather than on the (real) desk.

Clearboard

The Clearboard systems (versions 0 - 3) are all based on the metaphor of two people talking to each other through a clear class board where they can see each other and both draw on the glass with markers [Ishi92]. There is a video-only version of Clearboard that uses ordinary markers and erasers, and there is a digital version of Clearboard that uses digitizing pens and a computer-based shared drawing tool. Like VideoWhiteboard, Clearboard-2 allows the use of traditional drawing tools, but it has the additional advantage of allowing eye contact and "gaze awareness." It does not, however, support the use of real paper.

Ubiquitous computing

Ubiquitous computing (or "ubicomp") makes computers of all size and shapes available to people throughout their physical environment [Weis91, Weis93]. Mark Weiser has decided to pursue this vision through devices of three sizes: inch, foot and yard, so he describes how an office of the future will be filled with "tabs, pads and boards." Tabs are palm-sized computers with a touch-sensitive display that can be brought with you everywhere, and they are connected to the greater computing infrastructure through infra-red communication. Pads are similar devices but notebook-sized, pen-based, X Windows compatible, and connected by radio communication. LiveBoards [Elro92] are like workstations with huge displays the size of whiteboards. They use untethered infra-red pens and run special drawing software for meeting support [Pede93].

This vision of ubiquitous computing has influenced a great number of research projects at Xerox PARC. The work described in this dissertation has also been influenced by this vision, but it is worth mentioning how it differs. Ubicomp is an approach to Computer Augmented Environments that is centred on creating *new* electronic devices of all shapes and sizes. The approach taken by the DigitalDesk, in contrast, is about augmenting *existing* traditional tools — in particular the desk and paper documents. Ubicomp envisages an office full of hundreds of electronic tabs, pads and boards. The approach represented by the DigitalDesk aims to *augment* the existing post-it notes, paper notebooks, and whiteboards instead of replacing them by electronic equivalents.

See-through head-mounted displays

Ivan Sutherland's pioneering research on head-mounted displays (HMDs) inspired current virtual reality systems, but his first HMD was a see-through system [Suth68]. Each eye viewed a miniature vector CRT, whose synthesized graphics were merged with the user's view of the real world by means of a beam splitter. More recently, graphics researchers at University of North Carolina have used non-see-through displays combined with video cameras to create "see through" HMDs [Baju92]. One application of this display is to allow a doctor to view the sonogram of a foetus in context of the woman's abdomen. Another approach, taken by the KARMA [Fein93], system is to use the Private Eye [Refl92] to present 3D objects in alignment with real world objects.

Chameleon

Yet another approach to augmenting the physical world is through a small palmtop computer that can precisely sense its position and orientation in 3D space. George Fitzmaurice at the University of Toronto has developed a prototype

which can act as a window on what he calls a “3D-situated information space” [Fitz93]. The user of such a system can leave electronic objects anywhere in the physical world: along the edge of book cases, by a wall-mounted map or near various pieces of equipment. These objects are always present but normally invisible. To see and access them, one simply needs to move the palm-top display to their locations.

His prototype is implemented with a tiny video screen attached to an Ascension Bird [Asce92] which provides precise six degrees of freedom tracking within a three foot cube. The screen shows the image from a video camera pointed at an SGI workstation monitor, so all the intelligence and 3D computer graphics happens in on the SGI instead of in the palmtop display. In principle, however, the technology necessary to implement a wireless version of this idea is not that far off. Accurate large-area tracking technology, for example is described in [Azum93].

BrightBoard

BrightBoard [Staf93] is a computer augmented whiteboard. Instead of replacing the whiteboard with a computer-based system, BrightBoard is based on an ordinary whiteboard and can be used as such, but a video camera is pointed at the board and allows certain zones on the board to trigger actions on a computer. This work was partially inspired by the DigitalDesk and makes use of the DigitalDesk Toolkit.

Head-up displays

Head-up displays (HUDs) are commonly used in aeroplanes to allow pilots to see computer-generated information without needing to look down at the control panel [Wein92]. Usually the information displayed is not calibrated to be superimposed on the world outside. This is possible, however, and done in some cases (*e.g.* runway approaches).

Consumer products

Although not so well recognized within the computer science research community, there is a strong trend in consumer electronics now to include microprocessors in a wide range of every-day products. As we interact with our cars, watches, stereos, and telephones, for example, we are controlling sophisticated computer systems, but the interaction techniques we use are often exactly the same as those we used with older non-computerized technology. In most cases these devices do not yet communicate with each other except in specially built environments such as cars and aeroplane cockpits. Soon our houses will provide ways for various appliances and products to communicate with each other (descendents of the X-10 protocol for example [Pina93]), and some of today’s home security systems could be considered computer augmented environments. Many toys now use computers — not just video games. Some dolls have microprocessors in them to play recorded speech, and many toys play sampled sounds when you interact with them. Research at MIT with communicating programmable lego-style “bricks” [Resn93] give insight on what to expect toys to be like in a few years.

Entertainment

Also somewhat outside the research community is a long history of using technology to create augmented environments for artistic or entertainment purposes, starting right back with the earliest shadow plays through to current theatre and laser light shows. One very spectacular (truly multi-media) computer augmented show in regular use is at Disneyworld. The “Fantasia” show happens several times a night and includes films projected onto spraying sheets of water, music and speech coming from dozens of sources, animatronic characters of multiple shapes and sizes, fog, boats, explosions, fireworks, sheets of fire on pools of water and live actors just to mention some of the elements that are all synchronized by a sophisticated computer control system.

Augmenting real paper

Trade-offs between electronic and paper documents can make the choice of medium difficult, but imagine if we did not have to choose, and we had a space where documents could be both paper and electronic at the same time. Instead of putting the user in the virtual world of the computer, we could do the opposite: add the computer to the real world of the user and create a Computer Augmented Environment for paper. Instead of replacing paper with computers, we could enhance paper with computation. This has already started to happen in a limited way through the use of barcodes and the Xerox PaperWorks product.

Camera obscura and lucida

Probably the oldest technology for augmenting ordinary paper are the camera obscura and camera lucida [Hamm87]. These devices are optical, and did not make use of computer technology, but they both were used to superimpose images onto paper for the purpose of helping the user sketch this image in proper proportion and perspective (see Figure 1-5).*

”

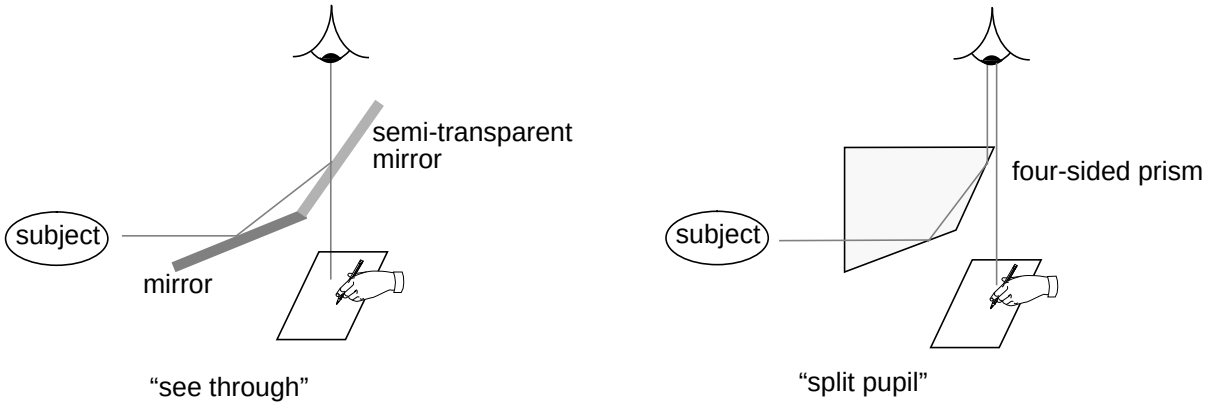


Figure 1-5: Two basic types of camera lucida.

* William Wollaston, who invented the camera lucida in 1807, was a fellow of Caius College at the University of Cambridge. His original wooden model is now in the Whipple Museum of the History of Science, less than a hundred metres away from the Computer Laboratory.

Shadow Parallax

The patent search for DigitalDesk uncovered a technique patented by IBM [Edga84] for projecting an image and detecting a pointing device that uses a double flying spot scanner made up from a projected CRT display and two photocells (see Figure 1-6).Two images from different perspectives taken by video

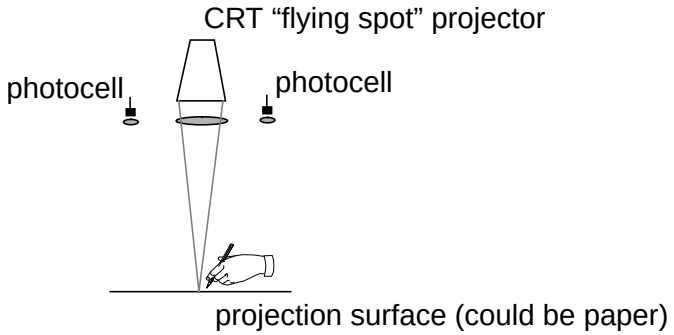


Figure 1-6: Edgar’s shadow parallax system.

cameras are difficult to register for the purpose of calculating parallax, but the pixels of images produced from the photocells are perfectly registered by the flying-spot scanning technique, making it easy to use parallax on the shadow of the pen to detect when it makes contact with the projection surface. Aside from the patent, nothing else seems to have been published about this system, so it is unclear whether it actually worked. The patent does not mention any working prototype applications, but in it, Edgar does mention the fact that this system could be used to augment ordinary paper.

Barcoding textbooks

One way to enhance paper documents with computer functionality is the technique of barcoding text books [Mill91]. This allows a reader to scan a barcode with a laser wand and automatically play a particular track in a videodisk recording.

PaperWorks

The Xerox PaperWorks product [John93, Xero92] also augments ordinary paper with computer functionality, with its fax-based paper user interface to a storage and retrieval system. With this system, ordinary paper forms are enhanced to control a PC through a fax machine. The forms are recognized by means of a “glyph” (a sort of 2-dimensional very dense bar code) [Hech90, Bloo90b, Schr92], and pre-defined areas on the page are examined to see if they have been check marked by the user, and handwritten tables can also be lifted off the page and used as file names. These paper documents gain some important properties of electronic documents, but fax machines are slow compared to computer screens. Response time is limited by the delay it takes to scan and print a page, so this limits the range of interaction techniques possible.

The DigitalDesk

Another way to enhance paper with computation is to do the opposite of the desktop metaphor. Instead of making the workstation more like a desk, we can

make the desk more like a workstation. This is the aim of the DigitalDesk. On this desk, papers gain electronic properties, and electronic objects gain physical properties. Rather than shifting more functions from the desk to the workstation, it shifts them from the workstation back onto the desk.

This Dissertation

The following chapter (Chapter 2) describes the DigitalDesk, its important characteristics, and gives an overview of how it is implemented. Chapter 3 describes a set of novel applications enabled by the DigitalDesk. Many of these have been prototyped and tested, while others were simulated in video. Chapter 4 describes the DigitalDesk Toolkit which was used to build the prototype applications. Most major components of the toolkit are described here except for thresholding, which is the subject of Chapter 5, and calibration, which is the subject of Chapter 6. These two components of the toolkit are discussed in greater depth because their implementation is not obvious, and anyone who may want to build a DigitalDesk will need to solve the same problems. Chapter 7 ends the dissertation with suggestions for future work and a conclusion.

Chapter 2
The DigitalDesk

Description

The DigitalDesk is a real physical desk on which you can stack your papers, lay out your favourite pencils and markers, and leave your coffee cup, but it is enhanced to provide some characteristics of an electronic workstation. A computer display is projected onto the desk, and video cameras pointed down at the desk feed an image processing system that can sense what the user is doing (see Figure 2-1 and Figure 2-2). No desktop metaphor is needed because it is *literally* a desktop.

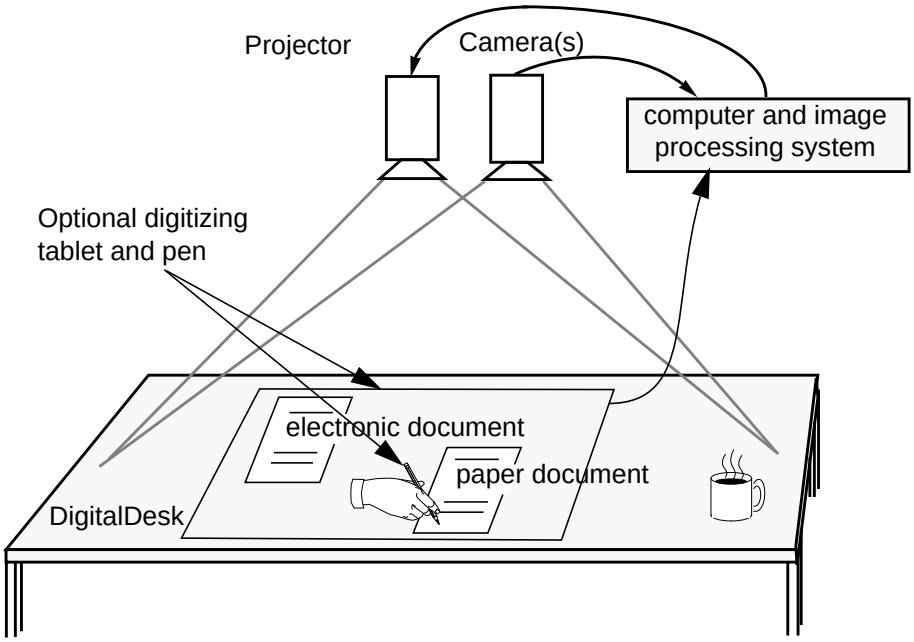


Figure 2-1: Schematic diagram of a DigitalDesk.

The DigitalDesk has the following three important characteristics:

- it responds to interaction with pens or bare fingers (hence *Digit*alDesk),
- it projects electronic images onto the desk and onto paper documents, and
- it can scan and recognize paper documents placed on the desk.

The following sections briefly discuss some of the implementation issues due to these three characteristics along with strategies for addressing them. Further discussion of implementation can be found in Chapter 4: The DigitalDesk Toolkit.

Figure 2-2: The first DigitalDesk prototype was cobbled together from an 1120 x 780 “ScratchPad,” display, an overhead projector, some spare video cameras, a cooling fan, polarizing filter, cardboard, and a *lot* of tape. Slicker versions based on commercial computer projection units have since been built.

Interaction on the desk

One aim of the DigitalDesk is to go beyond so called “direct manipulation” with a mouse (which in fact is not direct at all) and to explore the possibilities of “tactile interaction” with real and electronic objects using the fingers. The DigitalDesk merges paper and electronic documents not only by adding electronic properties to paper documents, but also by adding physical properties to electronic documents. We use pens and fingers to interact directly with paper on our desks, so we should be able to interact with electronic documents on the DigitalDesk in the same way.

One way of interacting with electronic objects with bare fingers is through video-based finger tracking. For some applications, obscuration of fingers by other fingers or parts of the body can be a problem [Stur91], but with desk work this does not seem to be a significant difficulty because the hands have a limited range of motion and they mostly remain in a two-dimensional plane. Pointing out things to the computer is much like pointing them out to another person, so it is easy for users to learn not to cover the object being pointed to. A bare finger is too thick, however, to indicate small objects such as a single letter, so the user must also be able to point precisely with a pen or other thin object.

A wide range of interaction techniques are possible using video-based finger tracking, as demonstrated, for example, by Myron Krueger [Krue91]. His system as well as the Mandalla system [Vinc90], rely on the hands being viewed against a plain background in order to make them easier to distinguish. Unfortunately, this is impossible on a DigitalDesk because there are various paper documents, pens and pencils present in addition to the user’s hands. It is difficult to distinguish these objects from fingers and pointers by shape only. A more effective strategy is to look for motion, assuming that most objects seen on the desk do not move except the user’s hands and the objects they are holding. An effective way to pick out moving objects from the background is to capture successive frames and to examine the image produced by subtracting successive values of each pixel in the two frames. The result, when applied to images of a moving hand, for example, is shown in Figure 2-3. This is a better image to start from than the original cluttered image, but further processing is required to remove noise and to locate the precise position of the fingertips. In the future, more sophisticated techniques will be necessary to track multiple fingers and to recognize gestures (see for example the finger and hand gesture recognition work in [Magg93] and [Sege93]).

Determining when users tap on the desk is difficult when only processing images from the overhead camera. One potential solution to this problem may be to use shadow parallax as described in Chapter 1 and in [Edga84]. Another (used for the calculator application described later in Chapter 3) is to detect finger taps in the same way a person might: by listening. A microphone is attached to the bottom of the desk and the system monitors the signal’s amplitude to determine when the user taps on the desk. This technique works well, but sometimes it confuses other taps, bumps on the desk, or hand claps with a finger tap. Another way to detect tapping is to use a pressure-sensitive mat. Unlike the microphone, it can provide dragging information as well as extra location data. A problem with desk-based touch screens, however (named the “Midas Effect”

Figure 2-3: The difference between two frames in which the finger has moved.

by Steve Freeman), is that users tend to rest their hands on it and everything touched can be interpreted as input.

Projected Display

Having discussed issues on the input side of desk top interaction, we turn now to output: the projected display. Projection from above provides similar capabilities to a large flat display screen and faces the same size *vs.* resolution trade-off, but it has the key advantage that computer-generated images can be superimposed onto paper documents. This is necessary for creating merged paper and electronic documents, and for providing feedback when making selections on paper. Overhead projection, however, does have some problems. One potential problem is shadows; it is not possible, for example, to lean down to look at a projected image too closely. In practice, however, shadows are hardly noticed when the projector is mounted above a horizontal desk (see section on *User Experiences* on page 26), but special measures must be taken to avoid shadow problems on a nearly vertical surface such as the Digital Drawing Board [Cart93].

Another issue with projection is the brightness of the room. The projectors used in these experiments work quite well with normal fluorescent lights, but a bright desk lamp or direct sunlight can make the display unreadable, so this may limit the desk's usability in some settings. One last problem with projection is that not all surfaces make good screens. The projection area should be white in order to see images most clearly, and some executives may be reluctant to hide their polished mahogany under a piece of paper or other screen.

Reading Paper Documents

For the DigitalDesk to read selected portions of paper documents the following steps are necessary: image capture, thresholding, and (in the case of text) character recognition.

Image capture

Document images are captured through an overhead video camera, but a difficulty with standard video cameras is their low resolution compared to scanners. One way to solve this problem, used by the Marcel document recognition system [Newm92], is to pre-scan documents at high resolution and then use the low resolution camera image to look up the corresponding scanned image. Pre-scanning is inconvenient for many interactive applications, however, so most of the prototypes described in this dissertation instead use two cameras, one of which is zoomed in close to the desk to obtain a high resolution image (about 200 spots per inch). This means that only a portion of the desk is used for capturing document images in high resolution, so a window is projected onto the desk to indicate the active area to the user. More cameras could easily be added to cover the whole desk, but this has not yet been necessary, because the applications tried so far only use small parts of a document at a time, and sliding a piece of paper into the camera's window is so easy. In the long run, higher resolution video cameras, high definition television, and advances in low cost, integrated digital cameras will make this approach more economical.

Thresholding

The image produced from a video camera and frame grabber is grey-scale (typically eight bits-per-pixel), even when it represents a white sheet of paper with black ink. This grey-scale image must be thresholded, or converted to a one bit-per-pixel black and white image before it can be used for character recognition or any of the other example applications described above.

Simple global thresholding is not adequate for obtaining an image suitable for character recognition. In normal office lighting, the range of brightness on different parts of the desk varies greatly, so a global threshold creates large patches of black and white with indistinguishable text. In order to make a good one-bit-per-pixel image of a black and white document, the system must use an adaptive thresholding algorithm which varies the threshold value across the image according to its background value at each pixel. Some adaptive thresholding algorithms produce very good results but require more than one pass through the image, and are too slow to support user interaction. It is possible to get nearly as good a result in a single pass, however, by calculating the threshold value at each point from an estimate of the background illumination based on a moving average of local pixel intensities (see *Adaptive Thresholding* section in Chapter 5). This method is fast and can also be combined with a scaling operation if necessary.

Finally, when dealing with text, the thresholded image is skew-corrected and recognized by an optical character recognition (OCR) server (*e.g.* Xerox Imaging System's ScanWorX [XIS93]). If the resolution is high enough relative to the text size, then it returns the associated ASCII string. Because this process is not guaranteed to be accurate, it is important to provide both quick feedback and

a simple way for the user to correct unrecognized characters (as in the calculator example below).

Summary

This chapter has given an overall description of the DigitalDesk and briefly described some of its implementation issues. The next chapter discusses applications that can be built on this desk, then subsequent chapters describe key implementation issues in more detail.

Chapter 3
Example Applications

The DigitalDesk provides a Computer Augmented Environment in which paper gains electronic properties that allow it to overcome some of its physical limitations. This chapter explores the range of applications made possible by the DigitalDesk. It begins by describing a set of working prototypes, implemented to varying levels of robustness. It describes some of the user experience and feedback from the prototypes that were tested on relatively naive users, and finally, it describes some video envisionments that were not fully implemented but simulated up in video. The simulations proved extremely useful to obtain feedback and criticism of the ideas, and to help decide which aspects of the DigitalDesk seemed most promising to pursue in the short term.

Working Prototypes

Many different ways to enhance ordinary paper documents are possible on this desk. The following subsections describe four working prototype applications: a calculator, PaperPaint, a French to English translation system, and the DoubleDigitalDesk. Two more applications (Mosaic and Digital Drafting Table) are also discussed; these were implemented by other researchers using DigitalDesk ideas and the DigitalDesk Toolkit.

Calculator

The calculator is a simple and familiar application that can benefit from the DigitalDesk, and it was the first prototype implemented on the desk to prove feasibility of the concept. People using calculators often enter numbers that are already printed on a piece of paper lying on the desk, and they must copy the numbers manually into the calculator in order to perform arithmetic on them. Transcribing these numbers can constitute a large proportion of the keystrokes when using a calculator, and a large proportion of the errors.

The DigitalDesk Calculator (previously described in [Well91]) addresses this problem by providing another means of entering numbers. It allows people to place ordinary paper documents on the desk and simply point at a printed number to enter it into the calculator. In the working prototype, users can point with a pen or bare finger, and a rectangle is projected in front of the finger to indicate which number is selected. When the user taps, the system reads this number with a camera, recognizes the digits, and treats them as though they had been

typed into the calculator (See Figure 3-1). Users can then do an operation on the number such as add it to others on the paper, or multiply it.

Figure 3-1: (Calculator) This photograph was taken just after the user selected the number 4834 on a piece of paper and the number was recognized and put into the projected calculator as though it had been typed.

This example application shows how a paper document can become more like an electronic document by allowing selection of numbers for calculation. Of course, it is also possible to scan a paper receipt or annual report through an optical character recognition (OCR) program to convert the entire paper document to electronic form, but this is a less interactive, more batch-oriented process, and the resulting electronic document has lost all physical properties of the original. The purpose of the DigitalDesk is not to convert paper into electronic documents; its purpose is to support rapid and direct computer-based interaction with selected regions of paper documents.

In this prototype, numbers are entered into a projected calculator tool, but it would also be possible to use a physical calculator that was connected to the DigitalDesk by wire or infra-red. Another possibility is to leave out the calculator, and project results back onto the paper. A “paper spreadsheet” could be designed on which pen-based interaction techniques would operate in the same

way on both ink and projected numbers (See the section below on video simulations or [Well92] for a video of how this and other possible applications might work).

The system uses image differencing to follow the finger or pen, and it detects taps by listening with a microphone attached under the desk. The smaller window that is visible in Figure 3-1 shows the field of view of the high-resolution camera used for OCR, while the camera used for finger following covers the entire desk surface. The OCR used in this case is not yet the ScanWorX system, but a crude single-fount recognizer that is built into the application. Notice also the projected rectangle that follows the tip of the user's finger to indicate the number selected.

*Desktop Translation**

Another example of an application that can benefit from this desk is foreign language translation. Looking up words in a dictionary can take up a substantial amount of a reader's time when reading documents in a foreign language. The time spent looking up words also makes it more difficult to remember the context of the word or the passage. Some readers find this delay so disruptive to their reading that they prefer to read on despite the presence of many unknown words.

With William Newman, a system was implemented in which French documents can be read at a desk in their paper form and the user can simply point at unknown words. The system extracts the root of the word, looks it up in a French-to-English dictionary and displays the definitions in an electronic window projected onto the desk, allowing the user to point to the location where the translation should be placed on the desk (see Figure 3-2). Any number of words

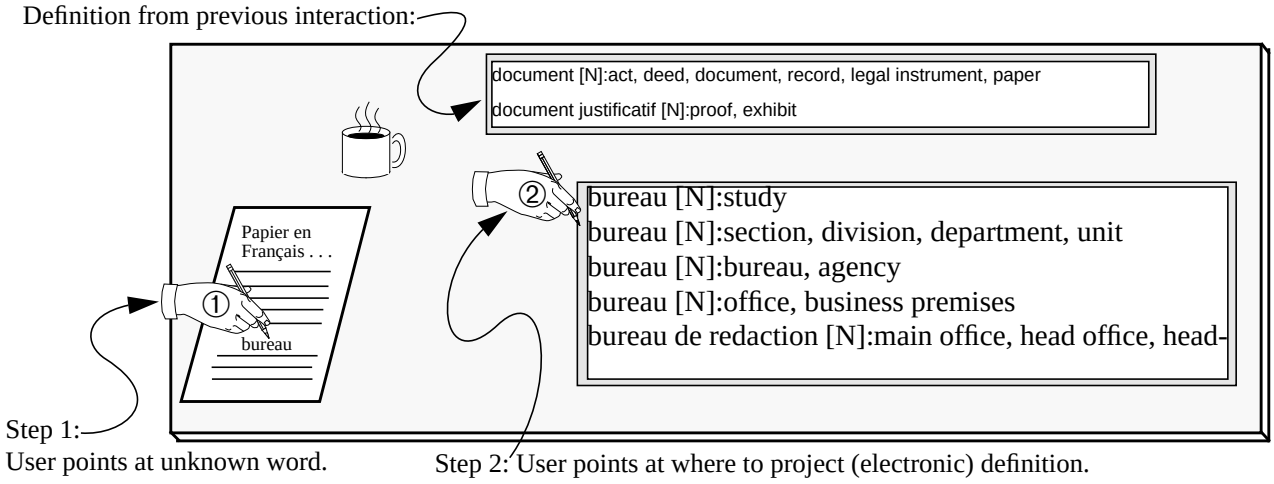


Figure 3-2: Translation on the DigitalDesk.

* This application was featured on the BBC television program *Tomorrow's World* on 29 January, 1992.

can be looked up, and the user can slide the windows around on the desk or remove them when they are no longer needed.

From an interaction point of view, this translation application could be considered very similar to the calculator application. Both support selection of text on a paper document, performing an electronic operation on the recognized text, and getting the result projected back down on the DigitalDesk. From an implementation point of view, however, the two applications use very different strategies. The calculator does the OCR on the actual image that it gets from the zoomed-in camera. The translator, however, uses a pre-scanned image of the paper document which is OCRed ahead of time. William Newman's Marcel system [Newm92] uses the low resolution image it gets from the overhead camera to recognize which of the pre-scanned images it corresponds to. It uses the shape of text margins, gaps between paragraphs and gaps between words to recognize documents in a resolution-independent way. This means the camera does not need to be zoomed in close to the desk, and the single wide-angle view of the desk can be sufficient to access the finest details of recognized pre-scanned documents. In principle Marcel does not have to be limited to pre-scanned images. It could also recognize documents that originated from within the computer system, or any document that has been through a fax machine recently, or a digital copier. The current version of Marcel makes assumptions about how a page looks; it assumes a single column of pure text, so many typical documents cannot be recognized. In principle, however, this is not a fundamental limitation. A more general-purpose resolution-independent representation of document images could be developed which could be very useful to DigitalDesk applications.

There is a fundamental restriction to relying purely on the Marcel approach, however. Any paper document must be scanned or somehow entered into Marcel's database at high resolution before it can be used on the DigitalDesk. In some cases, users might feel that if a document is in the workstation anyway, why not just use it there instead of in its paper form. The interaction necessary to scan a piece of paper into the workstation eliminates many of the advantages of the overhead camera. For small bits of text or numbers, it will often be much more convenient to be able to simply put the paper on the desk and point at the text without first having to scan the page in. Also a sketching application like the one described in the next section would be awkward to use if it relied on this approach. It is much better for users if they can immediately interact with the sketches they make on paper without needing to make an explicit scanning step.

PaperPaint

Although "select and paste" is now a standard feature of electronic documents, the same operation is awkward to perform with real paper, requiring a photocopier, scissors, and some glue or tape. The DigitalDesk, however, makes it possible to select and paste paper documents in the same way that we select and paste electronic documents. A simple paint program has been implemented (PaperPaint) in which a sketch on paper can be electronically selected by

sweeping out an area of the paper with a stylus; the projector displays a rectangle on the paper to indicate what is selected (see Figure 3-3).

Figure 3-3: The user selects the hand-drawn sketch of the window with the digitizing pen, and the projector displays a rectangle for feedback.

When the stylus is raised, the system snaps a picture, and the projected rectangle is replaced by a thresholded electronic copy of the area. This copy can then be moved about and copied to other parts of the paper. Sliding this electronic copy over the drawing to place it somewhere else is very similar to sliding a paper copy (see Figure 3-4). This application allows users to construct a mixed paper

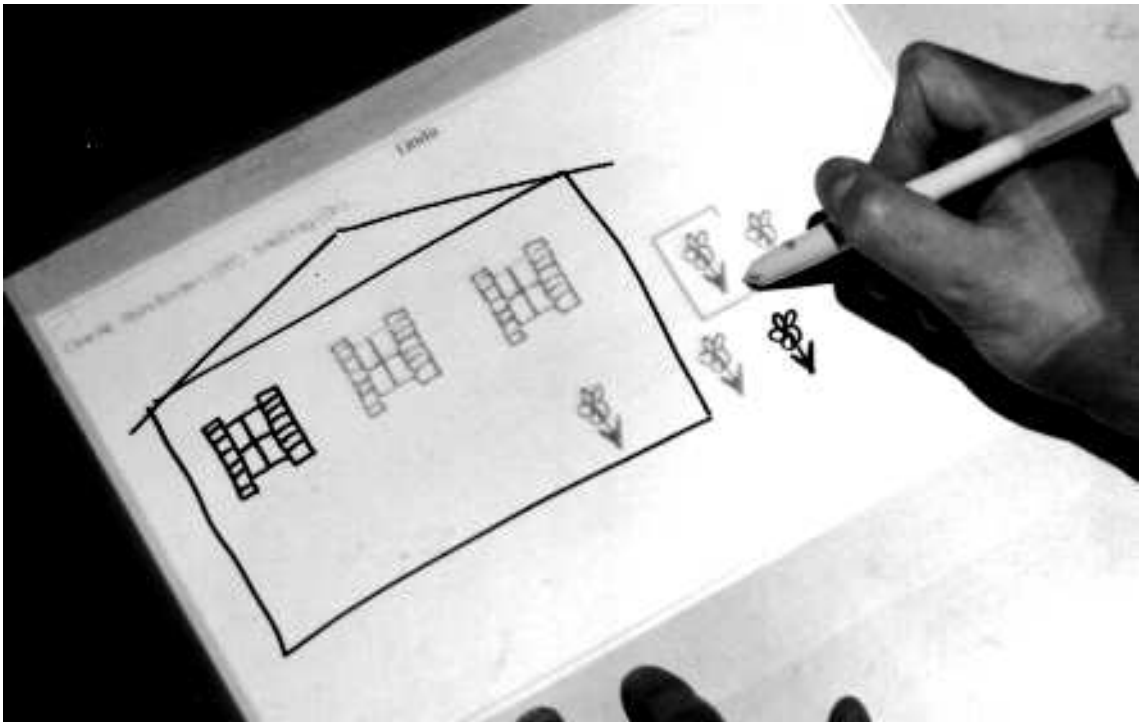


Figure 3-4: (PaperPaint) This user has made two copies of the window to layout the facade. Now he is about to move a copy of the flower that he drew.

and electronic drawing. Currently, the system cannot move the projected image to follow the paper when it moves, so users must keep the paper still. A combined scanner/printer (not yet implemented, but simulated in the video) would be necessary to make the projected marks permanent and allow users to take the merged document away. Even without such a printer, the system could be made to recognize papers whenever they are placed on the desk (using a system such as Marcel) and fill in the appropriate electronic parts.

User testing of PaperPaint revealed another (unexpected) way of using this tool which is also very powerful. Instead of constructing a mixed paper and projected drawing, some users noticed they could construct a purely projected drawing from selected portions of their paper sketches. They can sketch a figure on paper, move it and rotate it to the desired location in the projected drawing, then select it so that it remains "pasted down" in that location after moving the paper away. The effect is like that of dry-transfer lettering or rubber stamping, but from any piece of paper onto projected electronic drawings. This encourages the use of hand-sketched or printed templates of *paper*-based "clip art" that can be naturally placed and rotated into position with the fingertips. This interaction technique is quite different from the standard "select and paste" found on most workstations and takes advantage of unique qualities of the DigitalDesk: using both hands for manipulating and pointing as well as the superimposition of paper and electronic objects.

PaperPaint II

Although PaperPaint has features unavailable anywhere else (*e.g.* the ability to electronically paste down and copy selected areas of paper documents), it lacks most of the features commonly found in traditional paint tools (*e.g.* electronic sketching with lines of various thickness). Lionel Lopez-Welsch has taken the PaperPaint code and enhanced it to provide many of these basic features. His system, PaperPaint II [Lope93], allows sketching with projected ink using the digitizing stylus, selective deleting of copied objects, or-ing selected images into the background instead of always having them overwrite the background as in PaperPaint, inverting bitmaps, flipping them from left to right or top to bottom, etc. These types of features are, of course, important reasons why users would be motivated to use a tool like PaperPaint. A proper version would have all the features of programs such as MacDraw and MacPaint, but provide this seamless access to real paper sketches, potentially giving users the best of both worlds.

DoubleDigitalDesk

People often use documents when working together, and they often need to simultaneously see and modify these documents. Two people on separate continents cannot normally write on, point at, or otherwise manipulate the same paper document, but this is another constraint of physical paper than can be addressed by the DigitalDesk.

Shared editing of documents has been the focus of a number of research projects (see [Olso90], [Ishi92] and [Minn91] for examples). Most of this work has concentrated on screen-based documents, but the DoubleDigitalDesk makes it possible to share real paper documents. It allows users in two separate locations to

"share" their physical desks, both seeing, editing and writing on each other's paper documents.

In this application, each DigitalDesk continuously grabs images from its local desk and projects scaled, thresholded images from the remote desk. The result is that both users see what is on both desks. When a paper document is placed on desk A, its image is projected onto desk B and *vice versa*. The projections are digitally scaled and positioned to provide the same view to each participant. This is possible because each desk is first calibrated using the techniques described in Chapter 6. The contrast and ambient light is adjusted so that feedback (the image of your own desk transmitted back from the remote desk) is minimized. Both users can draw with a real pen on both paper and electronic documents, and the other user will see these marks appear in the corresponding places. Hand motions are also transmitted, enabling users to see each other point to certain areas on the documents (see Figure 3-5). The partner's hands block

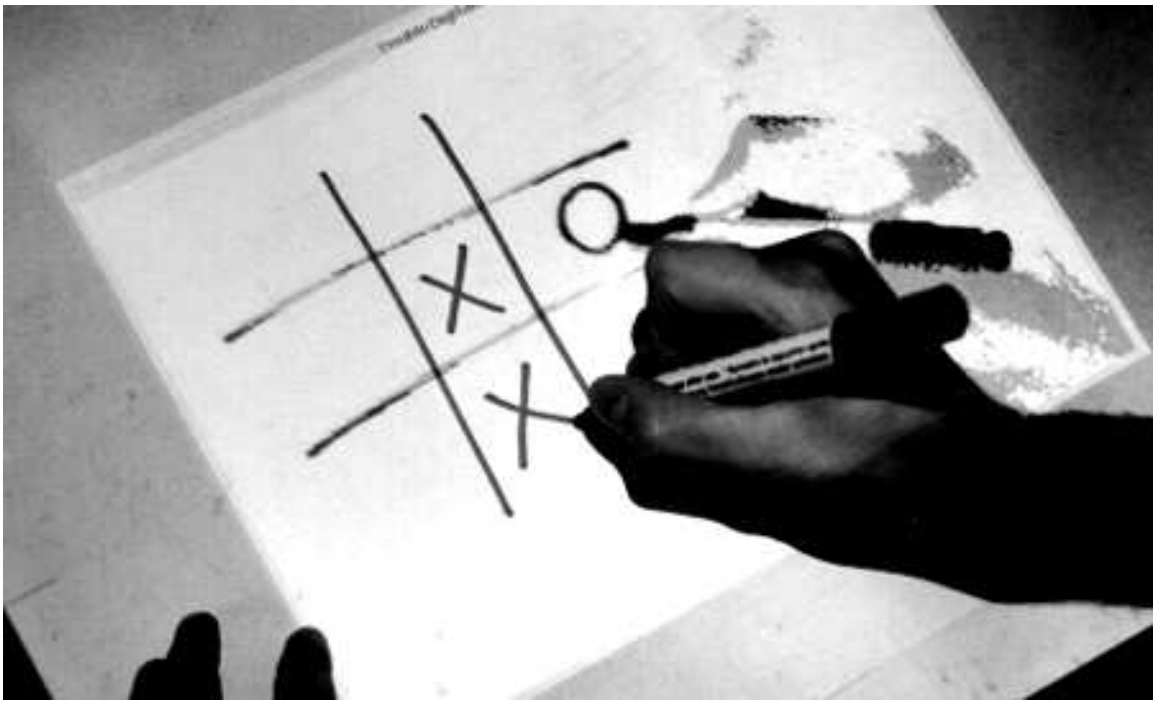


Figure 3-5: (DoubleDigitalDesk) The local user is drawing on paper with real ink, while the remote user's paper and hand can be seen having just finished drawing an "O."

the view of what's underneath them, just as with an ordinary desk, so this must be dealt with through social protocols and speech. Not pictured in the figure is an audio link through telephones or speakerphones. Another useful addition is a face-to-face audio-video link through, for example, the EuroPARC RAVE system [Gave92].

This system could scale up to support three or more shared desks by “or-ing” all the remote thresholded images together, or by displaying each desk’s image in a different colour. Of course, as more people share the space, participants will need to be more careful not to interfere with each other’s documents and writing.

DDD has an important advantage over some of the other DigitalDesk applications in terms of possible commercial value. The extra expense of the DigitalDesk hardware is much easier to justify if it can be used instead of making aeroplane trips. It only takes a few plane tickets, rental cars and hotel rooms before you’ve paid for a couple of DigitalDesks.

The implementation of DDD as described above could be implemented purely with analog hardware, and the performance would be much better. Precise calibration, positioning, and scaling (the trickiest parts of this implementation) could be done with high quality optics, but keystoning could be hard to handle, and it would require a much more time consuming setup that is awkward to change (as in VideoDraw and VideoWhiteboard [Tang91]). Analog hardware would not handle projectors of varying sizes and resolutions so easily, and connecting more than two sites would be more complicated.

Multi-Device DDD

Another way the computers can add value to this system, however, is by allowing users to share the capabilities of PaperPaint. A shared version of PaperPaint would not only allow sharing of paper documents, but it would also provide an electronic shared drawing tool. For this, support for multiple remote input devices is necessary, and Steve Freeman has extended the basic DDD system to support this [Free93]. Another important reason to support multiple pointing devices on the DigitalDesk is so that multiple uses at a *single* desk can easily use it to share applications such as drawing programs or outliners.

Digital Drawing Board

Kathy Carter has applied the ideas of the DigitalDesk to a very large, drafting-table-sized surface. Architects, map-makers, electrical engineers and industrial designers, for example, all need large surfaces on which to sketch their ideas and refer to blueprints and other drawings. The Digital Drawing Board [Cart93] provides this environment with a colour projector. A prototype application grabs images that the designer sketches, and uses them as texture maps for rendering the surfaces on solids of revolution. Designers, such as glass engravers often find it difficult to visualize how their sketches will look when placed on a three-dimensional solid. This tool shows how they could very quickly get from their paper sketches to a three-dimensional image of the final result, all in their preferred working environment.

Mosaic

Another application of the DigitalDesk is being explored by Wendy Mackay. The initial Mosaic application [Mack93] is designed to support video producers who use a combination of video-editing equipment and paper storyboards to manage the development of their videos. A storyboard consists of a set of elements, each containing a sketch or image of the “best frame” of the video, associated text, and notes that describe the action. Storyboards provide an efficient

way to sketch action sequences and develop “what-if” scenarios. They are flexible, portable, and easily annotated. The designer can lay out a large number of storyboard elements and view multiple alternatives at the same time.

Normally there is no easy way to associate the off-line storyboard with the video that has been created on-line, so producers must move to a completely different system to view and edit the moving video. Mosaic, however, uses the DigitalDesk to provide a new interface that combines the benefits of paper story-boards with computer-controlled video. The producer can lay out the paper storyboards in any order, the video camera recognizes them, and the video clips (digitally stored on optical disk) are reorganized accordingly and immediately shown at the desk.

User Experiences

Although no formal experiments have been conducted with the DigitalDesk, a number of people have performed tasks on it using these prototypes, and their reactions were noted. All subjects had used traditional workstations or PCs before, and they all said they found the desk generally comfortable and natural to work with. One unexpected result of the tests was that no one was bothered by the projection system and the shadows cast by hands and other objects. In fact, after using the desk for almost fifteen minutes, one user asked if it was made of glass and looked under it to see where the display was coming from! It may be that people are so used to the shadows cast by overhead lights and desk lamps that these types of shadows are hardly noticed. At the end of each session, subjects were asked how they felt about using the desk compared to a traditional workstation. Their overall reaction was that it was quite similar, but they commented specifically that they had “more space,” it was “more healthy than a screen,” “easier on the eyes,” and “more manual.”

Handedness

Unlike on a traditional workstation, user interfaces on the DigitalDesk must take account of handedness. If feedback is projected to the lower left of the pointer, for example, then a right-handed person has no trouble seeing it, but a left handed person does have trouble because it is projected on the hand. Of course, handedness affects the use of any pen-based system, but with a projected display, shadows strengthen the effect. Not only is feedback affected, but also the general layout of applications. The French to English translation application, for example, inadvertently assumed that users were right-handed, with paper documents on the left, and projected definitions on the right. Left-handed subjects were inconvenienced by this setup because it required them to reach their arm farther than right-handed subjects, and at the same time, their arms hid the paper they were reading. These handedness issues could be addressed by using the overhead camera to automatically detect with which hand the user is pointing, and this information could be used by applications. A pop-up menu, for example, would be projected to the left of the pointer for a right-handed person, and to the right of the pointer for a left-handed person.

Obscuring Selections

Subjects also noticed a difference between selecting pixels with a workstation and selecting marks on paper with a DigitalDesk. On a workstation, we can obscure something with the pointer as we select it, but the system still knows what is underneath. When pointing at paper with the DigitalDesk, however, the system must be able to see the paper, and this means that fingers and other pointing devices must be out of the way. It may be possible to address this issue by storing previously snapped (or scanned) images; the Marcel system, for example, does not face this problem. A solution may be unnecessary, however, because people do not seem to have much difficulty learning how to interact with the system in a way that keeps selections visible. When sweeping out a rectangle in the PaperPaint application, for example, there are four ways of doing this (see Figure 3-6). If right handed people use method ①, or if left-handed people use method ②, they end up with a picture of their pen in the selections. They usually have no difficulty switching to one of the other three methods instead, and they do not seem to repeat the mistake (except sometimes just for the fun of selecting their own finger or hand to watch it get copied). In general, the system cannot see the selection unless the user can see it too, and that is easy to learn.

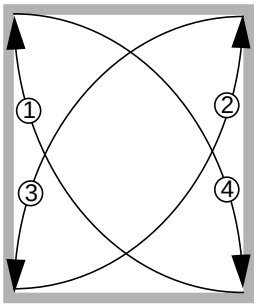


Figure 3-6: Four ways to sweep out a selection rectangle.

Selection feedback can also play an important role in preventing users from obscuring their selections. If a fixed selection rectangle is centred about the pointer, to use an extreme example, then it is impossible to get the pointer out of the way. If the selection rectangle floats slightly ahead of the pointer, however, then it is easy to avoid placing the pointer inside. If the system had handedness detection, then it would even be possible to adapt the selection feedback so that it prevented users from sweeping out a rectangle the wrong way.

Video Simulations[*]

The following pages show storyboards from a video [Well92] that simulates how the DigitalDesk might be used in the future. The video uses minimal software implementation with a lot of “smoke and mirrors” techniques to illustrate how it would work, and the word “Envisionment” appears throughout to remind viewers this is not an implemented system. This video was done after the calculator and translator prototypes, but before any of the other prototypes in order to

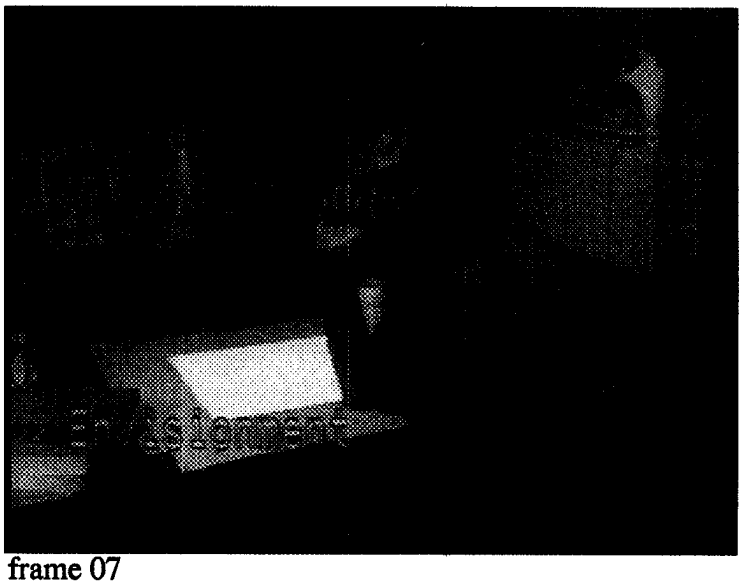
* I am grateful for the acting talents of Steve Freeman, Sian Wicklow, and Linda Malgeri.

help guide further progress. HCI researchers often spend a great deal of effort building partially working prototypes that are useful for “demo-ing” the idea, but not robust enough to test on users. If the purpose is to explore implementation issues and technical feasibility, then a rough prototype (such as the calculator) is more useful than a video tape. But sometimes the purpose is only to illustrate the idea and demonstrate how a novel interaction technique might look before spending the substantial effort necessary to make it work well enough for user testing. In this case, a simulation video is a much more efficient means to accomplish this goal because it is easier to make than a demo, and it can be shown to large numbers of people to solicit comments and discussion. This way a researcher can get useful initial feedback about an idea before spending the effort necessary to implement it.

In the original video, each scenario is shown twice: once in use and the second time with a voice-over that explains what technically is supposed to be happening behind the scenes. In this paper version, however, each scenario is only described once, both from the user’s perspective and with a description of how it would be implemented. The three scenarios illustrated are *taking notes*, a *paper spreadsheet*, and *sketching*.

Taking Notes

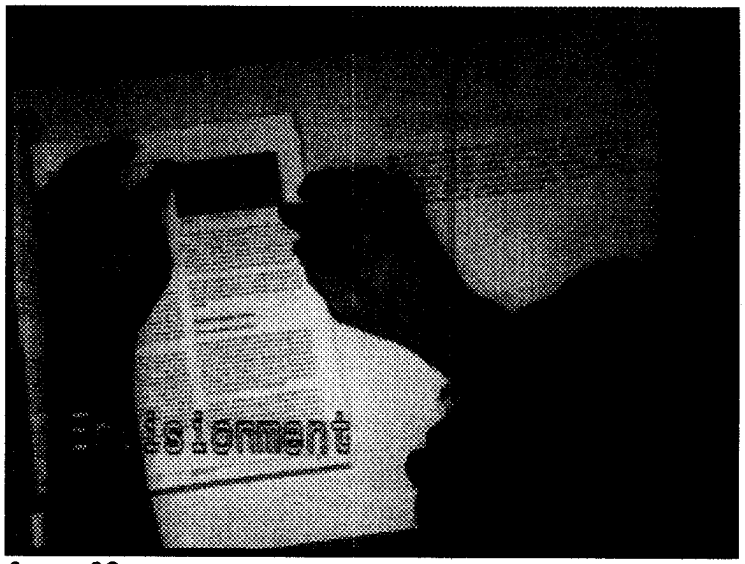
In the scene-setting shot, Steve says "My deadline's tomorrow!"



frame 07

... He leaves through a book, stops, and says "I'll just make a copy of this figure"

Then he selects the figure by touching his two index fingers together and sweeping out a projected black rectangle over the figure. Behind the scenes, the system is tracking his fingers and recognized the gesture for starting a selection. The projector displays the black rectangle for feedback so he can see exactly what he has selected. Other forms of feedback are possible, such as a rectangular outline which would cause less interference with the camera's view of what is being selected.



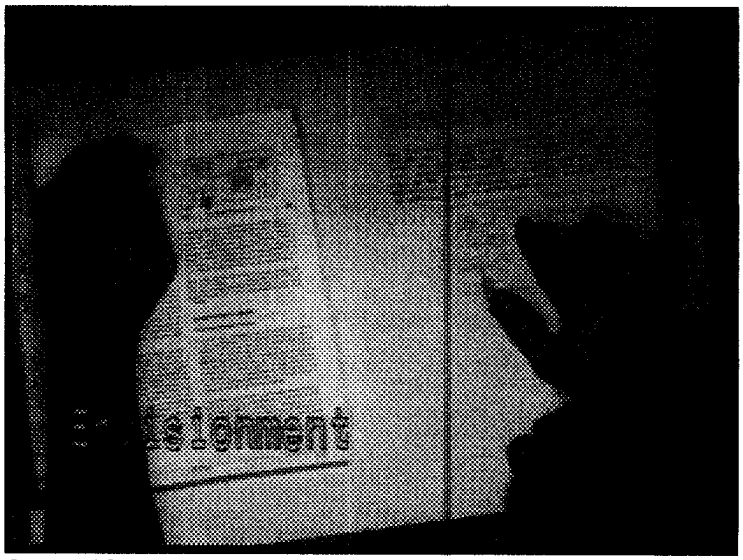
frame 08

Here, Steve is dragging his selection from the paper to where he wants to place it in his electronic document, and the projected rectangle follows his finger.

frame 09

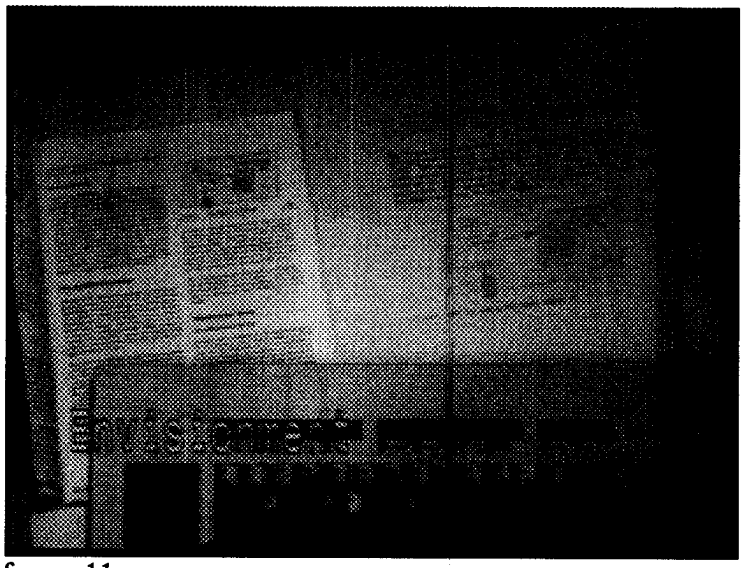
When he taps his finger again, the figure gets pasted into his electronic document at the indicated point.

Behind the scenes, the camera has snapped a picture of his figure, thresholded it and passed it (perhaps through ICCCM [Wide90]) to the electronic document editor.



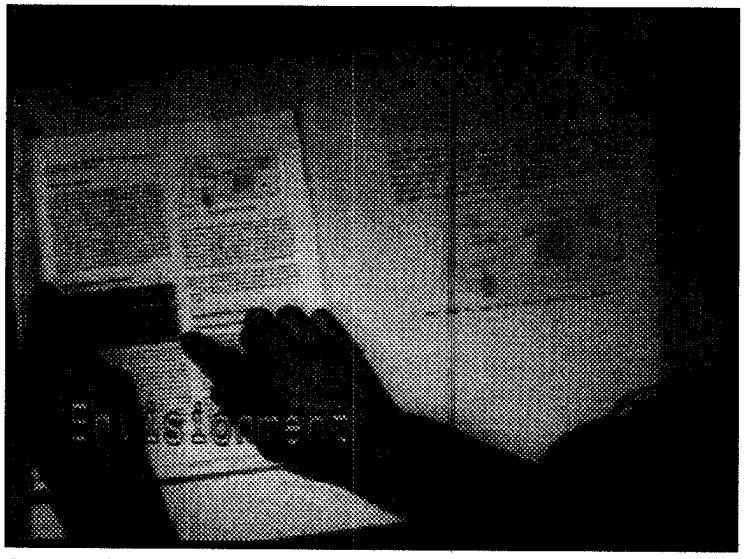
frame 10

Here Steve is typing a note into his electronic document using a cordless keyboard. This scene is to show that using the DigitalDesk does not mean you can't use an ordinary keyboard if you want to.



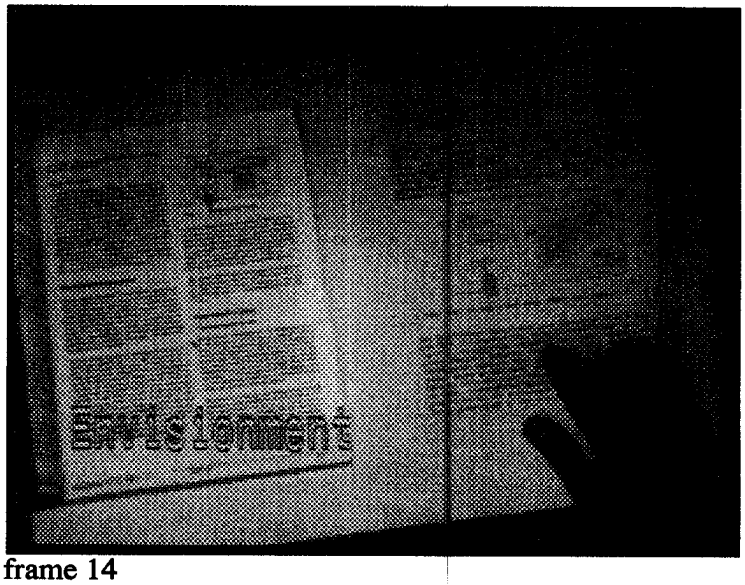
frame 11

Now instead of selecting an image, he is selecting text in the same way as before.



frame 12

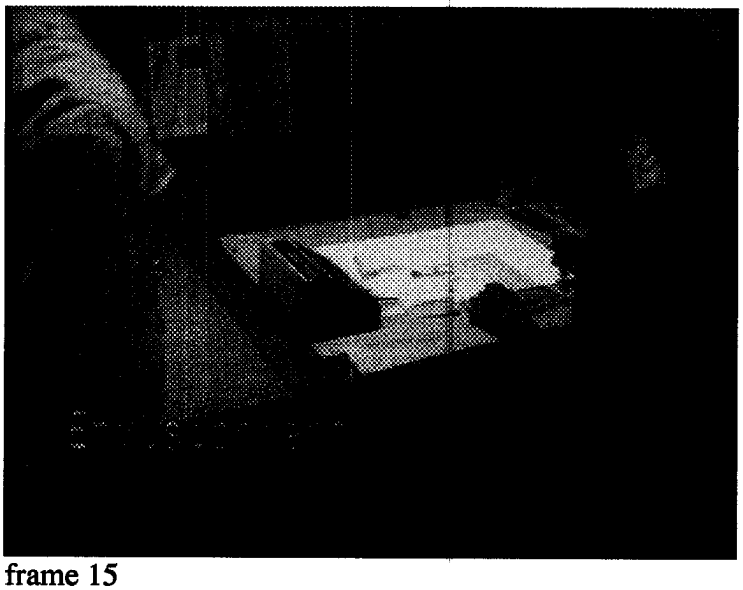
The difference is that when he pastes the text into his document it gets OCRed. The font changes and it gets reformatted to fit into the text flow of the electronic document. Of course, text could be treated in the same way as a figure, but it is often more useful to recognize it.



frame 14

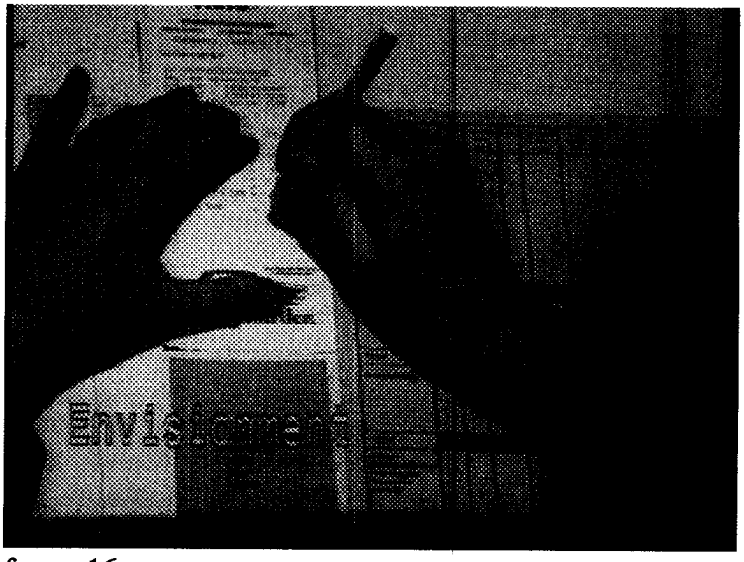
A Paper Spreadsheet

In the scene-setting shot, Pierre comes in to see Sian about filling out the expense claim for his trip, and he dumps a pile of receipts on her desk.



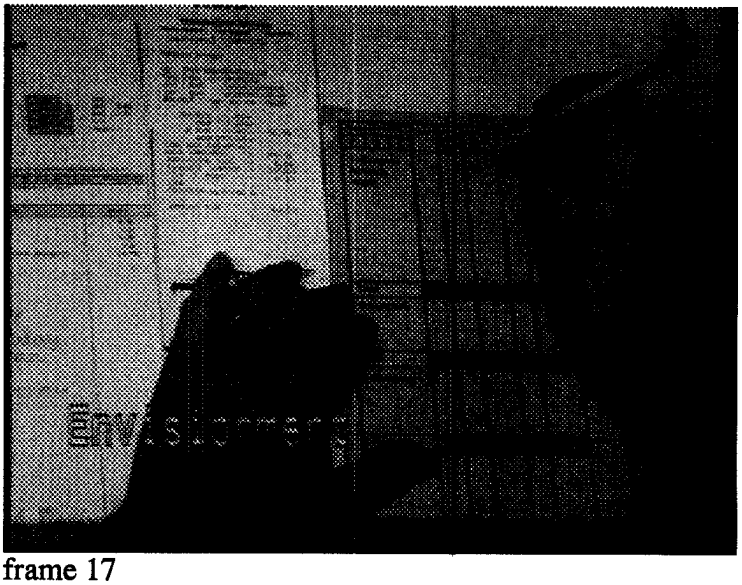
frame 15

Sian selects the total amount from the car rental receipt with her digitizing pen...



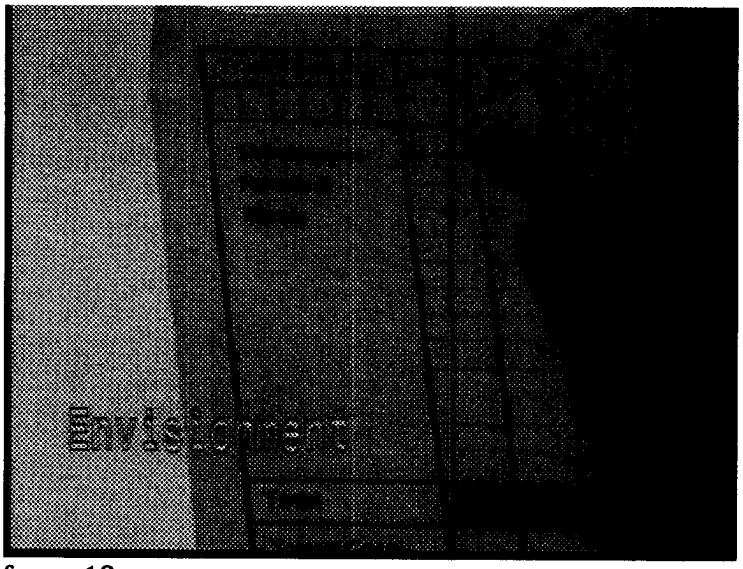
frame 16

then she pastes it onto the standard (pink) paper Rank Xerox expense claim form. This is a paper-to-paper select and paste operation where the camera reads the numbers from the receipts, they are recognized, and then the projector displays them on the claim form.



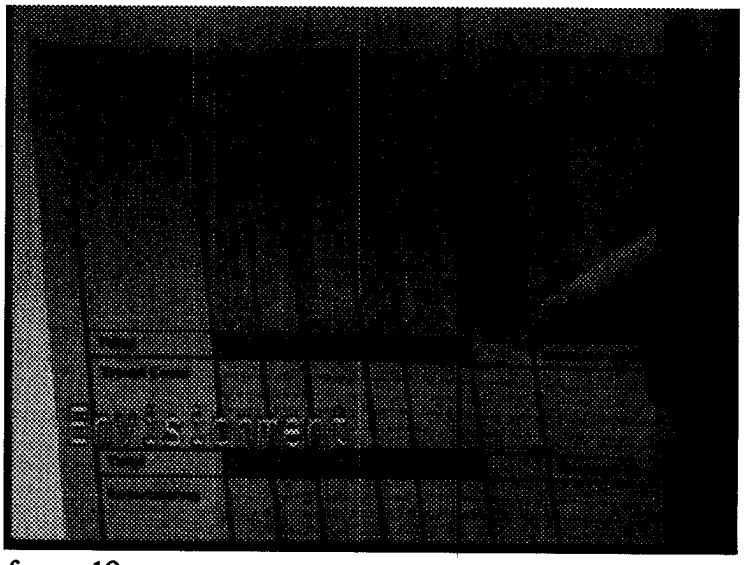
frame 17

Here, Sian writes in the date with her digitizing pen. The "ink" is projected down onto the paper as she draws, then the system recognizes the characters she writes and converts them to the same font as the other numbers on the form. In the next box she draws the symbol for "ditto," and the system projects the same date as in the box above.



frame 18

Here, after having filled the form, she selects a column of projected numbers (she could just as well have selected a column of printed numbers).



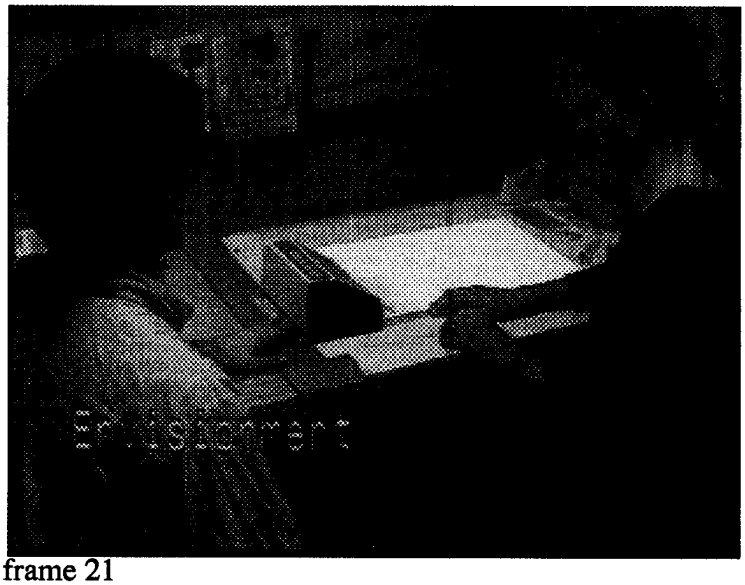
frame 19

Now, she points a yellow paper button (labelled *sum*) at the box below her selected column, and when she presses it, the sum is projected in that place. Behind the scenes, the system recognized the button (perhaps using morph glyphs [Schr92] or a bar code in the drop shadow) and her finger tap to know when and where to perform the summing operation. A more sophisticated version of this system (not mentioned in the video) would recognize the expense form and do the appropriate calculations without the need for such a button.

frame 20

When the form is complete, Sian puts it through a little printer in order to make the projected marks permanent, allowing Pierre to sign the form and take it away.

Note that this printer might need a scanner in it to accurately register the new marks with the old ones.



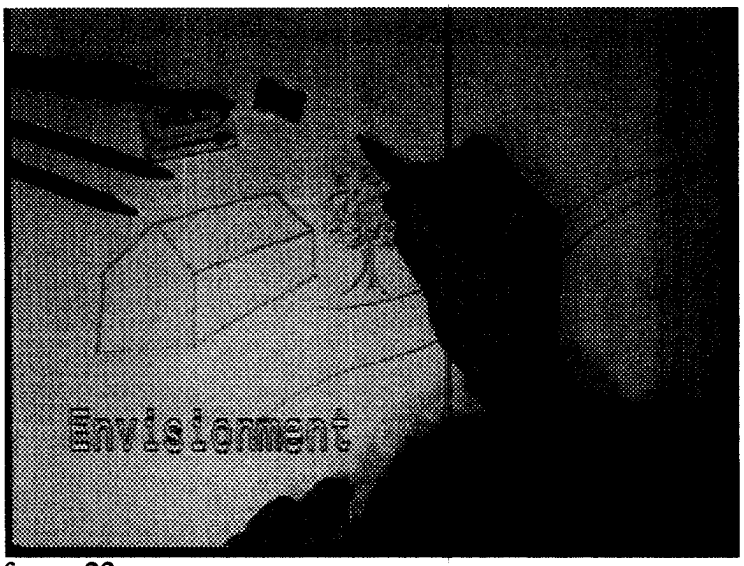
frame 21

Sketching

In the scene-setting shot Linda explains that she is roughing out some sketches for a children's book she's working on.

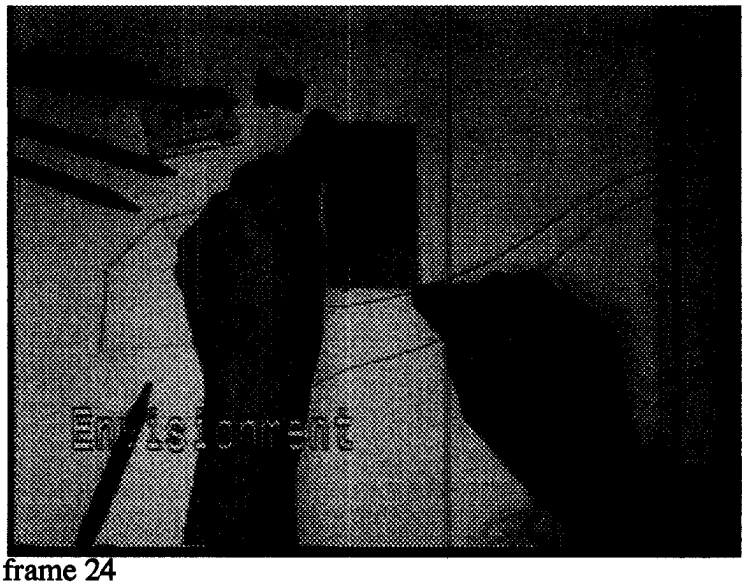
frame 22

First she sketches a tree with an ordinary pencil, and remarks that she wants a row of trees along the lane leading up to the house.



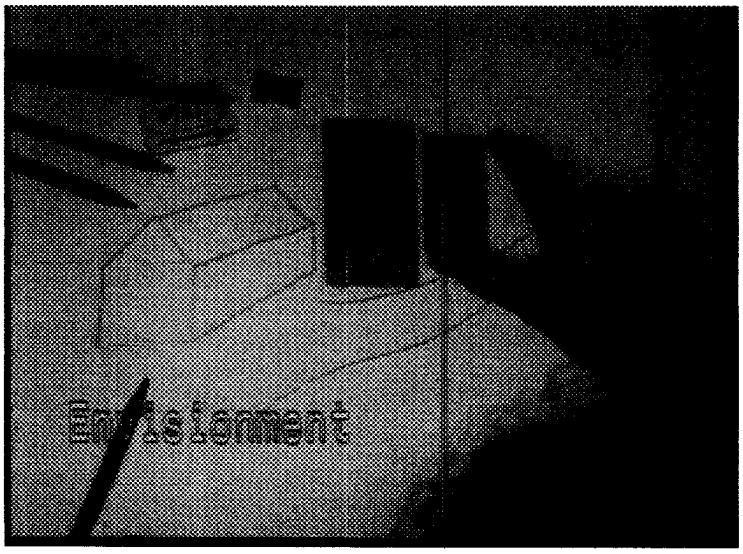
frame 23

She uses her fingers to select the tree, and the projector displays a black rectangle for feedback



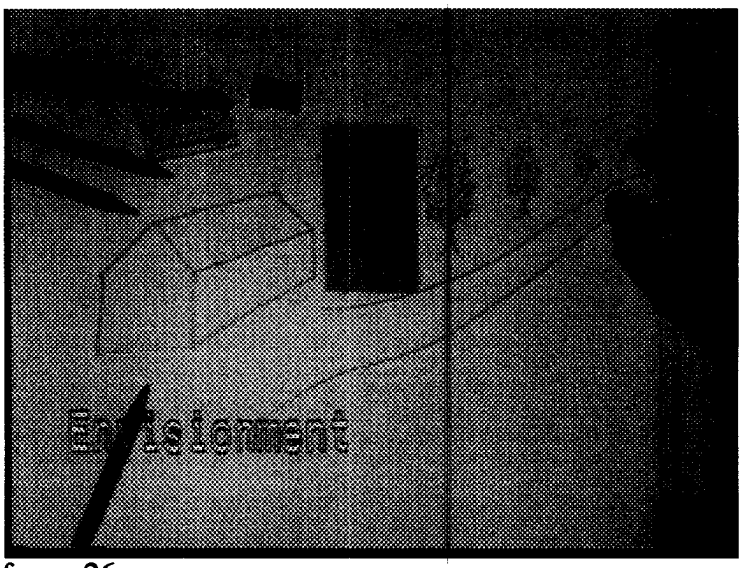
frame 24

Now she copies the tree, using the distance between her fingers to indicate the bitmap scale.



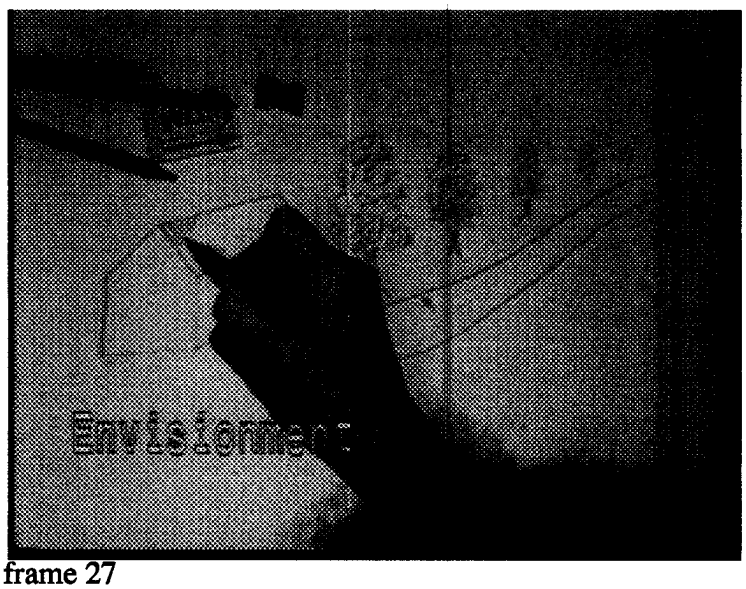
frame 25

The result after tapping her hand three more times with her fingers getting progressively closer together: a perspective row of trees along the lane.



frame 26

Next, she sketches some slates onto the roof, but the process is getting tedious, so...



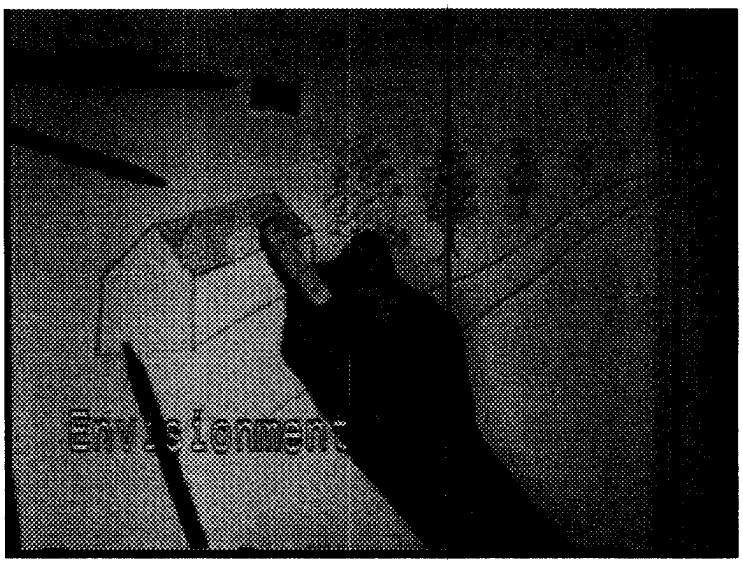
frame 27

She points with the paper "fill" button and taps her finger on it. The system recognizes the button and replicates the slate pattern to fill the area inside the lines drawn for the roof.

frame 28

But the roof needs a dormer window, so some slates need to be erased.

These projected slates can be erased just like pencil marks. The system recognized the eraser she's holding and the system blanks out the area where it rubs.



Sketching in the dormer window where projected slates were erased. The result is a merged physical and electronic sketch, but it would need to go through the little printer of the previous scenario in order to make the projected marks permanent. Another alternative would be to reprint the whole drawing.

Reactions to video

This video turned out to be very useful for demonstrating ways in which the DigitalDesk could augment paper. Many people who did not see much value in the idea when explained verbally, became quite enthusiastic after having seen the tape. The last "sketching" scenario seemed to get the strongest positive reactions, perhaps because it addresses a familiar limitation of current drawing tools. It was because this scenario seemed so appealing that it was chosen to be built first, leading to the implementation of PaperPaint. Many viewers commented that, although these ideas are innovative, they do not seem too far off or impossible to implement. The paper buttons got mixed reactions. Some viewers loved them, but primarily for their novelty. Upon reflection, many people thought they did not have significant advantages over projected buttons.

The video has also proven to be an effective means for publishing the ideas, often more so than the papers. About half a dozen Xerox managers have used it in talks they give. A Japanese magazine reproduced part of it in printed form [Imai92], it was featured as part of the Xerox Corporate Research publicity video at Barcelona during the world's fair, and several faculty members at universities have contacted me for copies to use in courses, including one course that is broadcast through cable television in Florida. I am frequently surprised at how many people I meet who remember having seen it somewhere.

Summary

This chapter has described a number of applications for the DigitalDesk, demonstrating a variety of different ways to augment paper with electronic functionality. The chapter attempts to show that the DigitalDesk enables a set of new human-computer interaction techniques that are not just novel, but also useful. A few of the applications described here were simulated in video, but the majority were implemented prototypes of varying degrees of robustness. The translation system has been used by about a dozen people, including some naive users, and PaperPaint has been used by several dozen. Reactions from users (and researchers) to the desk and its applications are generally quite encouraging, and at least one very promising new interaction technique ("transferring") was discovered during the course of testing the PaperPaint prototype.

Chapter 4
The DigitalDesk Toolkit

Introduction

The DigitalDesk Toolkit (DDT) is designed to support a variety of DigitalDesk applications. Early prototypes (e.g. the calculator) were implemented in such a way that made it difficult for people who might want to implement other applications to benefit from existing code. The variety of applications built and proposed for the DigitalDesk so far (see Chapter 3) have a certain set of common requirements. This toolkit attempts to provide the most important functions required by a variety of applications. Its purpose is to enable other researchers to explore applications on the DigitalDesk without having to reimplement the same functionality themselves, and to enable them to concentrate on aspects that are unique to their own work. This goal has largely been met, and the toolkit has helped several people to explore the design space of interaction techniques and applications made possible by the DigitalDesk.

This chapter gives a brief history of the toolkit, an overview of the architecture, then describes key modules and interfaces provided by the DigitalDesk Toolkit. It leaves out discussion of two very important issues: adaptive thresholding and calibration. Each of these is treated in a separate chapter following this one.

History of the toolkit

The initial implementation of the calculator prototype was done in C and C++ on a pair of workstations: a Sun 4/110 and a SPARCstation. This was because the Itex FG100 image processing board [Imag89] plugs into a VME bus (available only on the Sun 4), while the projected LCD display plugs into a SPARCstation Sbus. Figure 4-1 illustrates how the software modules interfaced to each other and to the hardware (note key in the bottom right corner). Although this implementation served well to demonstrate the feasibility of building a DigitalDesk, it was not suitable for giving to other researchers, or for building more complex facilities such as self-calibration.

At the time, there was a growing interest in Modula-3 [Nels91] among researchers at EuroPARC, Xerox PARC, and the Computer Laboratory, so it seemed that potential users of the toolkit would be familiar with this language. More important, it was clear that Modula-3's lightweight thread facilities would make implementing the toolkit easier than in C, and its clean interfaces would make sharing code with other researchers easier.Version 2 of the toolkit was therefore substantially written in Modula-3 and provided M3 interfaces to necessary C code. This version was immediately put to use by Kathy Carter and William Newman, both of whom found several bugs in it. Without going into the details of each release, the toolkit went through two more versions, gaining new functionality and new users with each release, and keeping up with new versions of

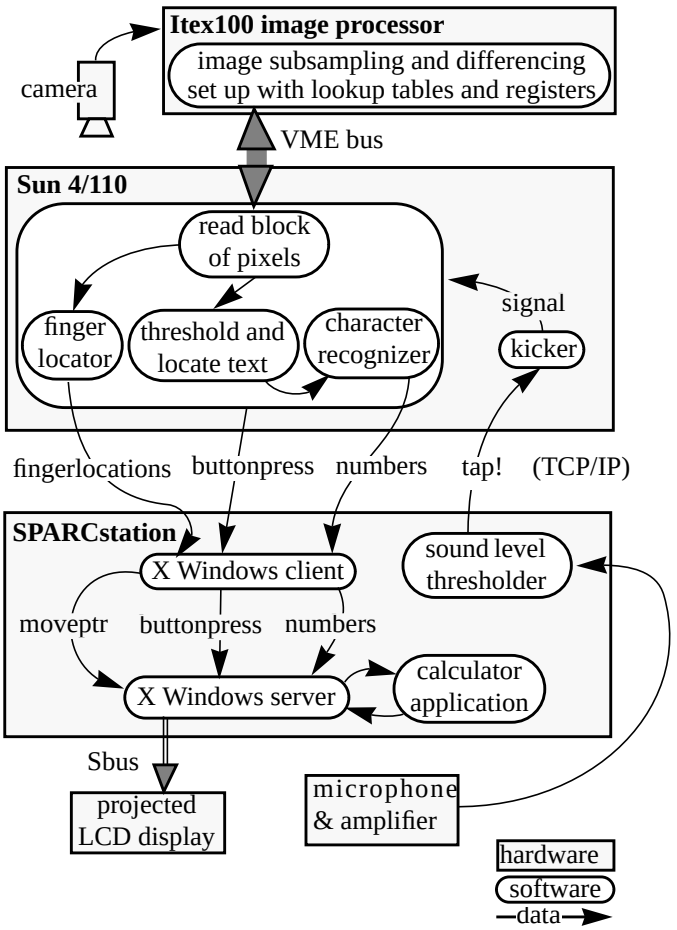


Figure 4-1: Architecture for original DigitalDesk calculator application.

Modula-3. The sections below give an overview of the architecture, then describe key modules and interfaces provided by the current release (Version 4). Thresholding and self calibration are treated in separate chapters.

Architecture

Figure 4-2 shows how some of the major components of the toolkit depend on each other. Modules higher up in the graph import interfaces to modules that are connected lower down.

The `tout` program allows a user to interact with the projected display using a digitizing pen instead of a mouse. The `fout` program is equivalent, but allows the use of a finger. The selection service allows a user to select an area on the desk (with projected feedback), and it returns the grey-scale image taken through the video camera of precisely that area. A client program then has a choice of what to do with this image. It can use it as is, or it can threshold the image by calling the quick adaptive thresholding routine described in Chapter 5. Then it can use the thresholded image as is, or it can use the image processing library to scale it so that when projected, it appears the same size as the original on the desk (this is what PaperPaint does). One last possibility is to send the

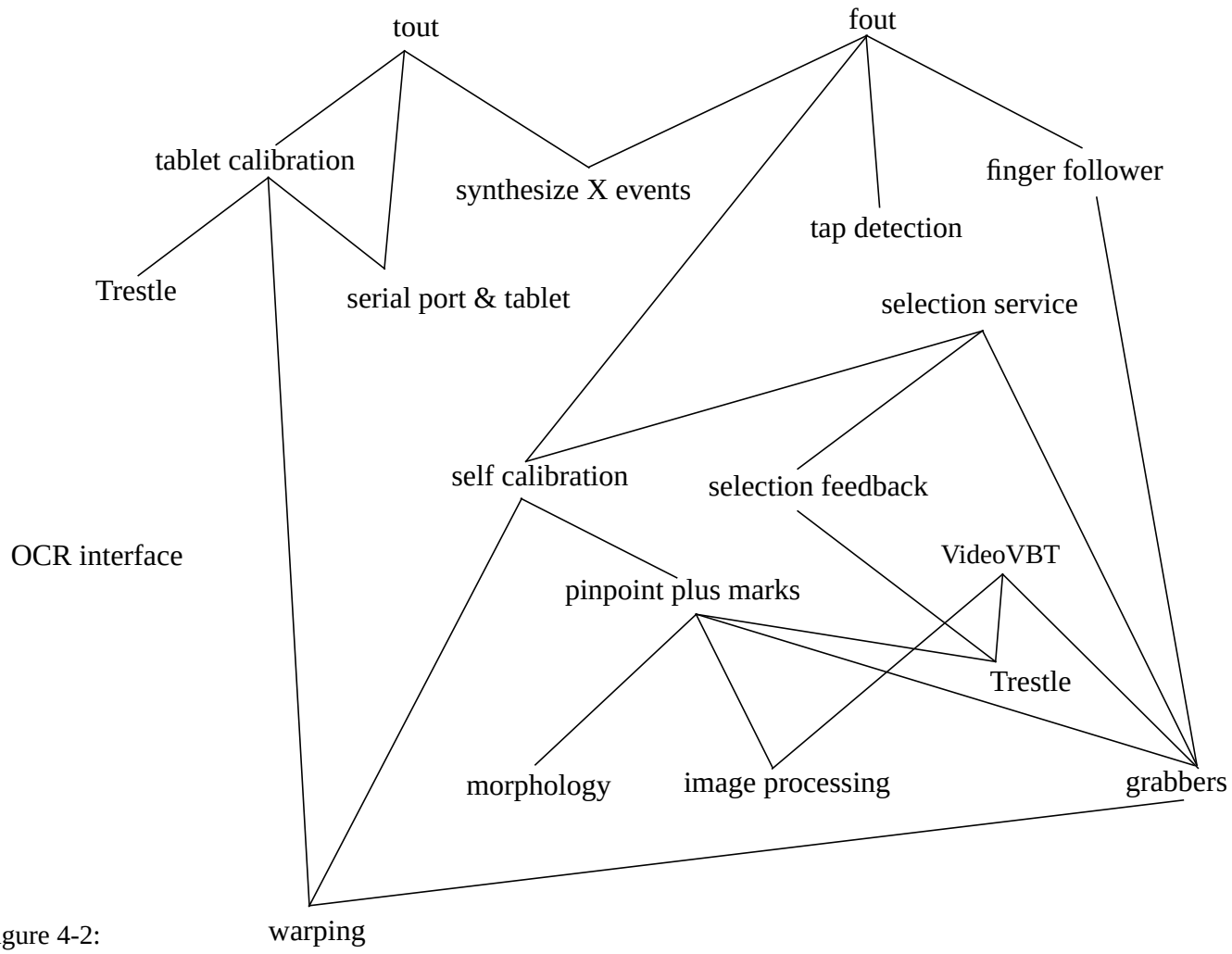


Figure 4-2: Architecture of DDT.

thresholded image to the Xerox Imaging System ScanWorX server [XIS93] through the OCR interface, where it is de-skewed and recognized, and the corresponding text string is returned to the caller (*e.g.* the calculator or translation application).

This basic functionality: converting a selected area on the desk to a grey-scale image, a thresholded image, or text string is the basis for a great number of DigitalDesk interaction techniques and applications. The following sections describe each of the components behind this functionality in more detail.

Frame grabber interface

The frame grabber boards are used for several different purposes, in different configurations, and by different processes. There are programs which do finger-following, optical character recognition, self-calibration and various tracing/ debugging functions. In some cases, multiple concurrent processes must

synchronize their calls to the board in such a way that they don't interfere with each other. In the first (C and C++) implementation, this was accomplished in a very crude way with signals, and it did not work reliably or very fast. The current implementation, however, makes use of Modula-3's lightweight threads and locking mechanisms to support multiple clients with shared memory. This makes it possible, for example, to have one thread looking through camera A for finger following while another thread can periodically grab an image from camera B to do OCR. The multi-threaded ability is also used in the DoubleDigitalDesk, where at each site there are at least two threads that access the frame grabber. One continuously updates the image from the remote desk, and the other occasionally grabs an area from the local desk for copy-and-paste operations. Each thread uses the same frame grabber board, but uses a different port on the board, and a different Warping.T (see Chapter 6 and appendix B). The threads never interfere with each other (except in terms of speed) because each one has its own Grabber.T object through which it access the board. (See Grabber.T interface in appendix A.)

Another requirement of the Grabber interface is that it be hardware-independent. DigitalDesk applications run on two different types of frame grabber boards: Itex FG100 and Sun VideoPix [Sun91]. They are also planned to run on the Itex S2200 board [Data92] in the future. These boards all have very different APIs (application program interfaces), but this is all hidden under the Grabber interface, implemented through the Modula-3 subclassing mechanism. There is an Itex subclass of Grabber.T, and there is also a VideoPix subclass. Client programs can explicitly create an object of either subclass, but they are not required to. There are two libraries that Modula-3 programs can link to: the default Grabber.T constructed in one library is of type Itex, and the default of the other is of type VideoPix. This means that in order to run a program with a different frame grabber, no source code needs to be changed; only a different library needs to be linked. Unfortunately, recent versions of Modula-3 carefully check which library (or Unix archive) programs are compiled with, so usually all source code which imports Grabber interfaces needs to be recompiled in order for the Modula-3 linker to allow linking with a different library. This can be tedious, but can be avoided by including the bulk of this code in with the two libraries. The advantage of this arrangement it is that it is not necessary to maintain different source code when using different frame grabbers. [See appendix A for the Grabber.T interface specification.]

Basic image processing and pixrect support

DigitalDesk applications all require some image processing capabilities. One of the most important is a way to threshold unevenly lit images, and this is dealt with in a separate chapter (Chapter 5). Other functions include simple manipulation of bitmap images, such as calculating histograms, scaling, displaying, copying, trimming, storing and reading from disk. Scaling is needed, for example, because the projector resolution is different from the camera resolutions, and it is necessary in an application like PaperPaint to compensate for this when displaying a grabbed bitmap. The Trestle toolkit has some support for bitmap images, namely a data structure (Pixmap.Raw) for storing them, selecting small

areas of them, and a way to display 1-bit-per-pixel pixmaps on the display. There exist many commonly available applications and libraries for manipulating digital images, and they use various image file formats such as tiff or Sun's rasterfile format [Sun90b]. It is necessary, then, to be able to store and retrieve images in one of these formats. Included in the toolkit is a way to do this with rasterfile images (compressed or uncompressed, one or eight bits per pixel). On Sun workstations, this format is accepted by nearly all applications, and conversion tools exist for any application which does not support it.

Some image processing libraries written in C, such as Dan Bloomberg's morphology libraries (see *mathematical morphology* section in chapter 6), provide functions very useful to DigitalDesk applications. The problem with C libraries is that they do not use Modula-3's garbage collected data structures for storing images, so this can make them tricky to integrate with Modula-3. As a solution, we can declare `UNTRACED` Modula-3 data structures that match the C data structures, and use them to interface with the libraries, but abandoning the use of `TRACED` storage reduces one of the advantages of using Modula-3. It also makes it impossible to use existing Modula-3 libraries that use the Pixmap.Raw structure.

The solution implemented in DDT is to provide both an `UNTRACED` and a `TRACED` version of the C data structures, along with procedures for converting between them and Modula-3's Pixmap.Raw structure. In most cases, when the image originates from within the Modula-3 program, then the `TRACED` version of the C data structure can be used. The cost of converting to it from Modula-3's Pixmap.Raw structure to C's Pixrect structure is very low (only copying a handful of words). It is necessary to make sure to freeze any pointers within this structure before passing it to a C routine, otherwise the Garbage collector will occasionally move these structures while the C routine is working on it. In situations when the image originates from the C program, then it must come into the Modula-3 program as an `UNTRACED` structure. In this case, the programmer has a choice to continue using it this way and accept the inconvenience, or it can convert it to the equivalent `TRACED` data structure before "letting it loose" into the system. This usually makes the program simpler, but the cost is that the entire data structure (including image data) must be copied into freshly allocated (`TRACED`) storage.

Image identification

Some applications, such as Mosaic [Mack93] and electronic annotation of paper documents (See Chapter 7 on *future directions*) need the ability to identify paper documents when they appear under the camera. The Marcel system [Newm92] does this in a resolution-independent way by looking at the shape of margins of text columns on a page. If several pages have identical patterns, then it can look at the spacing between words on the first few lines of the page. Unfortunately, this system only works on pages with a single column of text, but there may be other ways of doing this. One way might be to crop the image, de-skew it, reduce it to a standard size (say 64x64 pixels), then use this canonical resolution-independent form of the image to match against others. A simpler way to identify an image that could be much more robust is to use Xerox glyphs

[Hech90, Bloo90, Schr92] or a bar code [Pavl90] to print an identifier on each page that needs to be recognized. It may not even be necessary to do that if there is some unique text on the page that can be recognized with OCR (see next section). Although not provided in the released DDT, this capability is an important component of a DigitalDesk toolkit. Currently, developers must either use OCR or William Newman's Marcel system.

Optical character recognition

A key ability that some DigitalDesk applications require is to perform optical character recognition (OCR) on selected portions of paper documents. Once an area has been selected, captured, and thresholded, it must be OCRed to convert it to text. The current implementation uses RPC to call the Xerox Imaging Systems ScanWorX server [XIS93], one of the best available OCR programs. It can recognize a wide range of typefaces in a wide range of sizes. It can also learn how to recognize typefaces better as it goes along.

The ScanWorX documentation recommends that for "very accurate" text recognition, the rule of thumb is that

point size x resolution in pixels per inch must be greater than or equal to 2400

This means, for example, that to accurately recognize the 11 point text on this page, the resolution must be at least 218 spots per inch. The VideoPix frame grabber has a resolution of 640 pixels across, so this requirement is met as long as the camera is zoomed into an area no more than three inches wide. The rectangle on this page has the right size to provide "very accurate" recognition on text of this size. A larger field of view for the camera produces less accurate recognition unless a higher resolution grabber/camera combination is used, or larger point size text is to be recognized.

Paper documents may not necessarily be lined up with the video camera, so the grabbed image may need to be de-skewed before getting recognized. Dan Bloomberg's image processing library provides routines for deskewing, and this capability also exists within the ScanWorX server.

Evaluation

The initial Calculator application used a crude OCR program that only worked with a specific numerical font, but the current version of DDT is able to call the ScanWorX server.* None of the applications described in Chapter 3 except for Mosaic uses it, however, so experience with this is limited. Initial tests indicate that even when following the above rule of thumb, recognition is less than "very accurate" for most fonts, though it is quite good for others. ScanWorX does a much better job on scanned images than on video images. To make video based OCR work well requires more work. The new version of ScanWorX has a mode for low resolution images, which may work better, and perhaps the thresholding algorithm (described in the next chapter) could be tuned to produce better output for ScanWorX. Also, see [Cush90] for a discussion of how to evaluate OCR.

* Thanks to Chao Ying Ma and Giles Velay for writing C ScanWorX clients that I could modify.

Window system and user interface widgets

DigitalDesk applications must display a graphical user interface and respond to pointing devices in much the same way as any screen-based application on an ordinary workstation. This means the standard window systems and user interface toolkits on ordinary workstations can be very useful for DigitalDesk applications. There is no reason to build a separate window system or UI toolkit for the DigitalDesk, except perhaps because the standard systems do not have much support for pen-based interaction. The best window system and UI toolkit to use on the DigitalDesk would be one specifically designed to support pen-based interaction. Go PenPoint [Carr91] is probably the best example of this (see appendix discussing it), but Microsoft PenWindows might be good too. The current implementation, however, is based on the X Window system [Gett90] and Trestle (Modula-3's user interface toolkit) [Mana91]. Although it does not support many interaction techniques possible with pens (e.g. handwriting character recognition and gestures), its support of mouse-based interaction covers an adequately large subset of what is possible with a pen that this does not cause much difficulty, and more pen-specific interaction techniques can be implemented on top of it (see, for example, work by David Goldberg [Gold91] and Kathy Carter [Cart93]).

Many user interface "widgets" or interaction techniques on the DigitalDesk can be the same as standard ones on ordinary workstations. Push buttons, menus, reshapable windows, *etc*. are all useful. There are a whole set of interaction widgets better suited to pen-based systems than the ones currently used with mice; this is a separate research area, but one from which the DigitalDesk will benefit greatly. A few widgets were built on DigitalDesk for general use by a variety of applications. One is the VideoVBT (built for the DoubleDigitalDesk), and another is the SelectionVBT

VideoVBT

The VideoVBT is connected to a Grabber.T object, and it continuously grabs an image and displays it. The VideoVBT can display video cropped and scaled to any size. It has a couple of unique features useful for DoubleDigitalDesk applications: absolute position, and the ability to pause and resume it. It is possible to place a VideoVBT in an absolute position relative to the display. This makes it appear in a fixed location on the desk so that even if the VBT is moved or resized, the Video image remains in the same place. It might get cropped by the VBT or it might appear in the middle of it somewhere, but it will not move. This is useful for the DoubleDigitalDesk when the projected video image must correspond exactly to the field of view of the camera. The exact position of the camera relative to the display is determined by the SelfCalibrationVBT, described in Chapter 6.

SelectionVBT

One fundamental interaction techniques supported by the DigitalDesk is the ability to select a specific area on a paper document and get the frame-grabbed image of that area. This interaction technique is supported by using a SelectionVBT in combination with a Grabber.T. The SelectionVBT is a Trestle filter which takes any other VBT as its child, for example a VideoVBT or a

simple blank area. When the user presses the pen down in this VBT it draws a rubber-band rectangle that follows the pen until it is lifted. When the pen is lifted, the SelectionVBT calls a callback function with the area that the user selected. Other interaction techniques can be implemented as subtypes of SelectionVBT. A “lasso” interaction technique, for example, could be used to select non-rectangular regions. In order to grab the selected area on paper, this area is converted to a global coordinate space (usually the display coordinate space instead of the VBT coordinate space), and this area is passed to the `grabRect` method of a Grabber.T object. From this grey-scale image a thresholded image can be generated which can optionally be sent to the OCR server to get text.

Cut and paste between applications

Another capability usually provided by the window system that DigitalDesk applications can benefit from is inter-client communications. This is used to cut and paste data between applications. In X windows there is an ICCCM (Interclient communications conventions manual) [Wide90] which specifies how to do this. These conventions are widely followed for text, and DDesk applications can provide text to other X applications using standard Trestle interfaces. For graphical and bitmap data, however, the conventions are not generally used yet. Different windows in a single DDesk application can cut and paste bitmap images between them, but no source code for an ICCCM-compliant bitmap cut and paste application was found on which to base DDT code, so it currently does not yet support inter-application bitmap cut and paste, except through the use of files.

Pointing device interface

The simplest way for DigitalDesk applications to access pen events is through X windows, in just the same way that they normally access mouse events. The only disadvantage of this approach is that it makes it difficult for applications to access both the pen and the mouse at the same time. It makes development and testing much easier, however, because everything can be done on an ordinary X workstation, and the pen must be used only when it is specifically needed.

Synthesizing X events

Currently, the DigitalDesk uses two different ways to generate synthetic mouse events. The first way uses the journalling facility in versions 2.x of Sun’s XNeWs server [Sun90a]. XNeWS is an X server and NeWS server [Gosl89] all in one; clients can connect to it using either protocol. The journalling facility is implemented only in the NeWS component. If the window system is started up with the `-defeateventsecurity` flag then it allows clients to send PostScript commands that move the mouse and generate other input events. Applications have no way of detecting the difference between real mouse events and synthetic events.

The disadvantage of this approach is that it depends on Sun’s XNeWS 2.x server, which has some bugs and has not kept up with the latest features of X11R5. Most people do not commonly run this server, so it would be much

better if events could be synthesized for applications running on the standard MIT X server. This is possible to a limited extent, using the `XWarpPointer()` and `XSendEvent()` Xlib calls [Nye90]. This can work very well on any X server, but the X server sets a special flag on events generated by `XSendEvent()`. This means that applications can detect the difference, and some applications (such as our version of mwm) choose to ignore these events.

A third, potentially better, way to integrate pointing devices with X is to use the X input extension [Patr91]. This would require building a special version of the X server that includes device-specific code, but it has not yet been necessary for current DDT applications.

Tout

`tout` uses the first two approaches described above to connect the digitizing tablet to X. It first tries to connect to the server through a NeWS connection, but if that fails it uses an ordinary X connection. Before running, however, it must calibrate the tablet coordinate system to the display coordinates. Due to various distortions and rotations, it needs to obtain four tiepoints from the user (see Chapter 6 for more discussion of calibration), which it does by projecting four crosshairs on the desk and waiting for the user to click on each one. From these tiepoints `tout` constructs a warping and sets the resolution of the tablet to be no greater than the size of the displayed pixels; then it continuously reads events from the tablet over the serial port, and synthesizes a corresponding mouse event. Kathy Carter recently improved the original version of `tout` by integrating the calibration step into the main application, and by providing a graphical user interface for changing the mappings between the different possible states of the stylus and the different mouse buttons. The stylus has a tip switch and a side switch, giving it four possible states. The standard mouse has three buttons, giving it eight possible states. Fortunately most applications only use four of those eight states, because chording (holding down more than one mouse button at a time) is rarely used, so it is usually possible to use a stylus with standard applications that expect mouse-based input.

Coordinating multiple pointing devices

The current setup has a finger-following “pointing device,” a mouse, and a digitizing tablet. With the current system, output from the tablet or the finger follower moves the X pointer to absolute locations on the display. At the same time, the relative motions of the mouse (if any) are responded to in the normal way. There is only one pointer, and only one pointing device is used at a time. It is easy to quickly switch back and forth between the mouse and one of the absolute pointing devices. Switching between the tablet and camera is more difficult, but can be done when the digitizing stylus goes in and out of range of the tablet.

Pointing with the finger

Interacting with computers through video-based finger or pen tracking is a very promising area of human-computer interaction, and this technique is especially appealing for the DigitalDesk, because it already has the hardware required to

do this. Computer vision has traditionally been more concerned with robotics applications rather than human-computer interaction, but there are a few good examples of work being done in this area. In addition to the pioneering work done by Myron Krueger, two more examples are [Magg93] and [Sege93]. Because these ideas are being pursued elsewhere, a decision was made not to put much effort into video-based finger tracking for the DDT. An initial prototype was developed for the calculator application, but it was not refined well enough to make it truly useful.

As mentioned in Chapter 2, the finger follower on the DigitalDesk uses a microphone to detect tapping, and it cannot rely on a plain bright background to pick out the fingers because it must work with the cluttered background of a real desk. It therefore makes use of frame differencing (see Figure 2-3 on page 15) to locate moving objects to pick out the finger. The Itex FG100 board has an image loop-back feature that allows real-time frame-differencing with a lookup table. The top-most moving point is used as the pointer. This means that if the hand points down or sideways, then the tip of the finger won't be used as the pointer, but this is not much of a problem, because it's uncomfortable to point this way anyway.

The performance bottleneck is copying bits from the frame grabber into host memory, so the less data can be copied per cycle, the faster the system will track. In order to reduce the amount of copying, the system only reads a small square around the point where the pointer was last seen. If the pointer moves too fast, then it can go out of range and no motion is detected. If this happens the system reads a larger part of the image, but subsamples, so it can still copy over the same amount of data and retain its frame rate.

The accuracy of this method is surprisingly good. When a sharp pen is used instead of a finger, it's possible to point with very high precision. The zooming in and out causes some problems, however, because if the finger stops moving, the system thinks it might have moved out of range, and it zooms out to a coarser resolution. Then small movements can cause the cursor to jump around a little (one pixel at the coarsest resolution represents 64 pixels at the finest resolution) before the system zooms back in and the cursor rejoins the finger. The only way to get really good, accurate pointing with the current system is to continuously move the finger or pen just enough that the system can find it. This is awkward. With a higher performance image processing board this problem might not occur because subsampling would not be necessary. But there is still the problem of noise and shadows, however. These can cause the system to think the finger has jumped somewhere else when in fact it is just resting.

Use of the camera/microphone combination to implement tapping revealed an unexpected problem. In order to tap, it is necessary to lift the finger or pen. The raised finger does not show up in exactly the same spot as after it has been (swiftly) put down. Because of the delay in image processing, when the tap is heard, the system often thinks the finger is in a slightly different spot. At the expense of response-time, this can be solved by giving the image processing an extra cycle or two to catch up after a tap is heard.

There is a general problem with pointing to things that the camera is looking at, and that is simply that the pointer may block or interfere with what is being

pointed to. This applies not only to finger pointing, but also when pointing with a digitizing stylus. One potential solution to this problem is to offset the projected feedback (*i.e.* cursor or rectangular box), but experiments show that different people hold their finger or stylus differently, (the most extreme example is handedness) so one kind of feedback does not work for all people. Feedback positions must be configurable for individuals. A more ambitious solution (not tried in the DDT) is for the image processing system to adjust the feedback based on how it sees users holding their fingers or pen. For example, if it sees that the person is left-handed, it could project feedback to the right and vice versa. One last approach that would be interesting to try is shadow parallax, as described in Chapter 1 and in [Edga84].

Use of the toolkit

Several people have made use of DDT:

Kathy Carter's Digital Drawing Board uses most of the toolkit. Wendy Mackay's Mosaic project [Mack93] and the EuroCODE project [ESPR93] have been using pieces of it (*e.g.* the adaptive thresholding code and `tout`). William Newman and I combined his Marcel system with DDT code to make the French to English translation application, and Marcel uses the Grabber.T interface [Newm92]. `tout` has been used by Simon Anstey to do experiments with online handwriting recognition [Anst92]. The underlying connection to XNews and X for synthesizing events is in regular use by Tabbit, the program for connecting X windows to PARC Tabs [Weis93]. It has proven very reliable and useful not only for DigitalDesk work. Steve Freeman has taken pieces of the DDT and extended them to make remote versions of the Grabber interface, and he has used the warping code in his pointing device servers. He has also taken the entire PaperPaint and DDD application code and used it to make a multi-device version [Free93]. Quentin Stafford-Fraser has been using the Grabber interface, the image processing libraries, and the adaptive thresholding routines to help him with his Brightboard project [Staf93]. Also, Lionel Lopez-Welsch built a more sophisticated version of PaperPaint on top of the original version and DDT [Lope93].

Summary

The DigitalDesk Toolkit provides a set of modules and interfaces to support building of DigitalDesk applications. It has evolved through several versions and been used by its author and several other people to build the variety of applications described in Chapter 3. The current version of the toolkit, including the PaperPaint application, thresholding, calibration, and useful test utilities is made up of 28 Modula-3 interface files (with a total of 7164 lines of code), 38 M3 implementation files (18534 lines), 15 C files (10126 lines), and 10 .h files (2464 lines). The total number of lines of code (including comments and whitespace) is therefore a bit less than 40,000.

Overall, DDT is not mature enough yet to post on the net and expect hundreds of people to easily make use of it, but a significant number of people have used it successfully. It has therefore passed one of the most basic tests for a toolkit worthy of the name, which is that other people besides its author find it useful.

The following chapters describe two more components of DDT: thresholding and calibration.

Chapter 5
Adaptive Thresholding

Introduction to the problem

The image produced by a video camera pointing at black printing on a white sheet of paper is not truly black and white. This image is grey-scale (or colour) no matter where the camera is pointed. Unless lighting on the desk is carefully controlled, video images of paper lying on the desk are poor representations of the originals. Unlike the inside of a scanner or photocopier, it is very difficult to guarantee even lighting across the surface of a desk. This open space may be exposed to desk lamps, overhead lights, windows or moving shadows, all of which can cause varying brightness across the page. Human vision compensates for this, but when the image of paper on the desk is manipulated by machine without taking these variations into account the results can be very bad. In Figure 5-1, for example, the background of the right hand “in case of emergency” box does not match the surrounding background, leaving a distinct edge. The

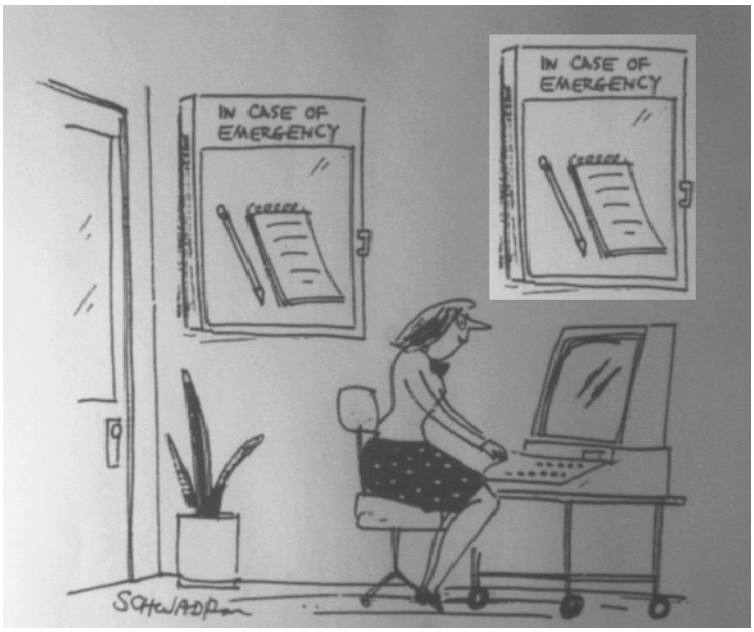


Figure 5-1: Copy & paste with an unevenly lit grey-scale image.

box on the right appears lighter than the box on the left, even though it is an exact copy (fold this page if you want to check). The only difference is in the brightness of the surrounding background. It gets darker from left to right because the light was on the left side of the paper from which this image was grabbed.

This problem is most noticeable when dealing with high contrast line art or text because the original document is genuinely black and white, yet the camera produces an image of varying levels of grey. Many applications prototyped on the DigitalDesk must clearly determine which parts of the image are meant to be black or white in order to project line art back onto the desk, or to pass text images to an optical character recognition (OCR) server. The system cannot use the grey images (typically 8 bits per pixel), so it must convert them to black and white images (one bit per pixel). There are many ways to do this. In some cases, when the resulting black and white image is meant to be viewed by people, it should look as much as possible like a grey-scale image and the image is *dithered* or *half-toned* (using a technique such as [Floy76]) to produce a result like Figure 5-1. Although this image looks like a grey-scale image, it is made up of small black and white spots of varying patterns, densities and sizes. Human vision makes sense of these images almost as well as the grey images and the original black and white images. But for machine processing operations, such as character recognition, select & paste operations, and merging together of many different images, the system cannot use dithered or half-toned images. Although only one bit per pixel, these images are even less suitable for machine processing than the original grey-scale images. The system needs simple patterns of lines, characters, and relatively large patches of black and white. The process by which these types of black and white images are generated from grey images is commonly referred to as *thresholding*, and it is an important requirement for any digital desk system.

There are many ways to threshold an image, but the basic process is to examine each grey pixel and decide whether to make it either black or white. This chapter describes the various techniques that were developed and tested for thresholding on the DigitalDesk, and it ends with the description of an algorithm that was found to be suitable. This algorithm (here called “quick adaptive thresholding”) may not be the best possible, but it works well enough that other problems of the DigitalDesk became more important when the work reached the stage described here.

Global Thresholding

In a way, thresholding can be seen as an extreme form of *contrast enhancement*, or making light pixels lighter and dark pixels darker. The simplest (and most common) way to threshold an image is to set all pixels below a certain grey-level to black, and to clear all others to white. The question then is how to select this grey-level. One possibility is to pick the centre of the range of possible values, so if pixels are eight bits deep (ranging from 0 to 255), for example, then 128 would be selected. This approach works well if all the “dark” pixels of the image really do have values under 128, and light pixels have values above 128, but if the image is over or under exposed, then the result might be all white or black. It is better to look at the range of *actual* values instead of *possible* values to determine the threshold. The maximum and minimum values for each pixel in the image can be found, then the midpoint used as the threshold. An even better way to select the threshold is not just to look at the range of actual values, but also their distribution. If, for example, you expect the image to be of black line

art, or text on a white background then you expect that most pixels will be the intensity of the background and a smaller but significant proportion will be of the dark ink. A histogram of the pixel intensities should look something like Figure 5-2.

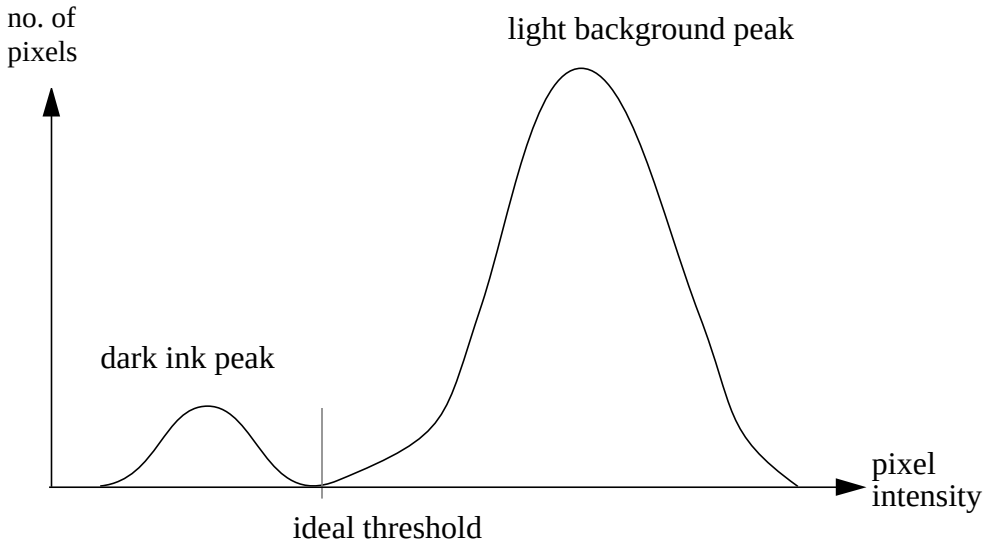


Figure 5-2:
Theoretical histogram.

A large background peak should be visible, as well as a smaller peak for the dark ink. The whole curve might be shifted to the left or right depending on the ambient light level, but in any case, the best value to pick for thresholding is the local minimum between the two peaks. This is fine in theory, but how well does it work in practice? Figure 5-3 shows a frame-grabbed image and its histogram for which the technique works well. The smoothed histogram shows two prominent peaks, and it would not be difficult to calculate a near-ideal threshold by fit-

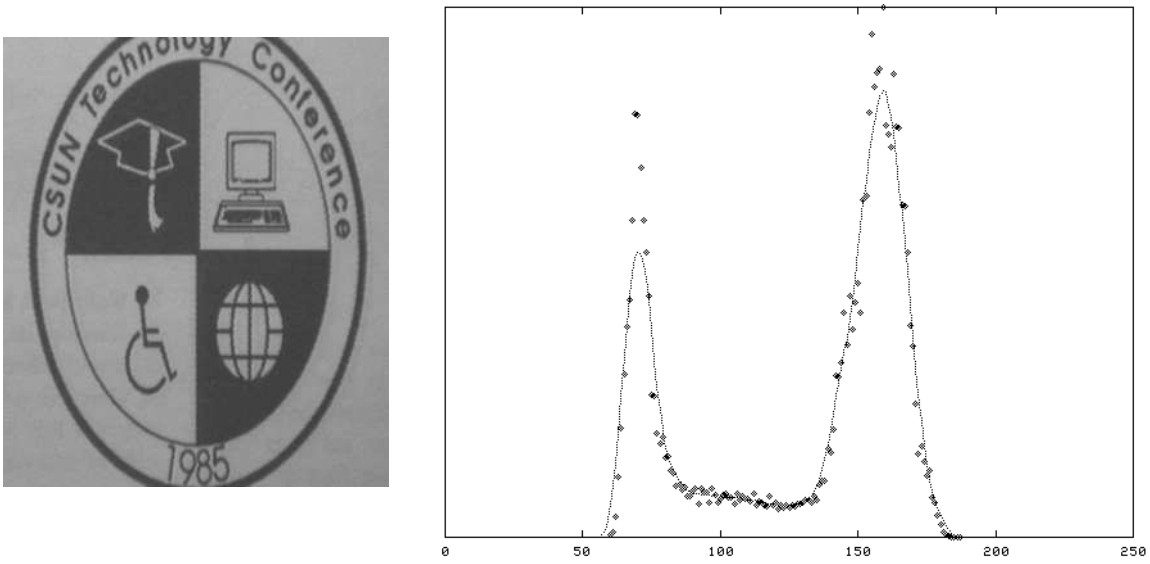


Figure 5-3: Grabbed image with clearly bimodal histogram.

ting a curve to this histogram or even taking the midpoint between the two peaks. This is not a typical image, however, because it has large amounts of both black and white. The DigitalDesk must also be able to threshold images like the one in Figure 5-4. In the histogram of this image, the smaller “dark” peak is almost lost in the noise, so it is impossible to reliably locate a local minimum between peaks

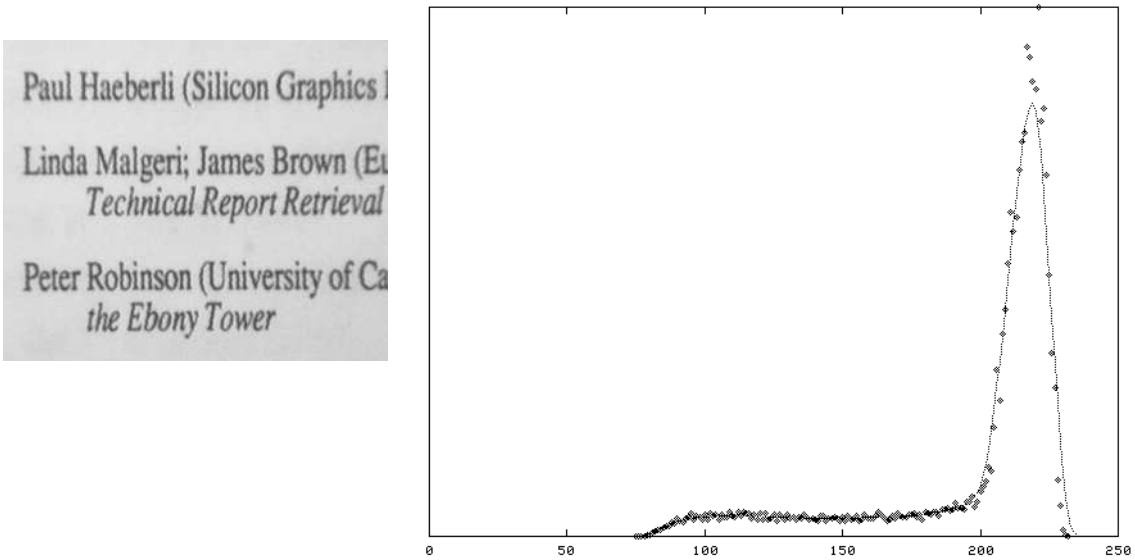


Figure 5-4: Grabbed image with “dark” peak almost lost in the noise.

In any case, a large (background) peak is always present and easy to find, so a useful strategy for thresholding an image is the following:

1) Calculate the histogram (shown by the points in Figure 5-4).
2) Locate the large peak by finding the maximum value of a moving average of the histogram curve. The moving average smooths out the effects of noise on the maximum value, as shown by the curves in Figure 5-3 and Figure 5-4.
3) Select the threshold value at a certain proportion of the distance between this peak and the minimum value (excluding zero counts).

Experimentation seems to indicate that 1/2 of this distance produces quite good results in a broad range of lighting conditions ranging from very bright to almost completely dark. In Figure 5-4, for example the peak is at 215 and the lowest recorded value is 75, so a threshold of 145 would be used. The figures on the next page (Figure 5-5) show an image grabbed under four different lighting conditions and the result from applying this histogram-based global thresholding algorithm to each. Despite the wide range of lighting (as can be seen from the histograms) this algorithm selects an appropriate threshold in each case, and the thresholded result for each is almost identical.

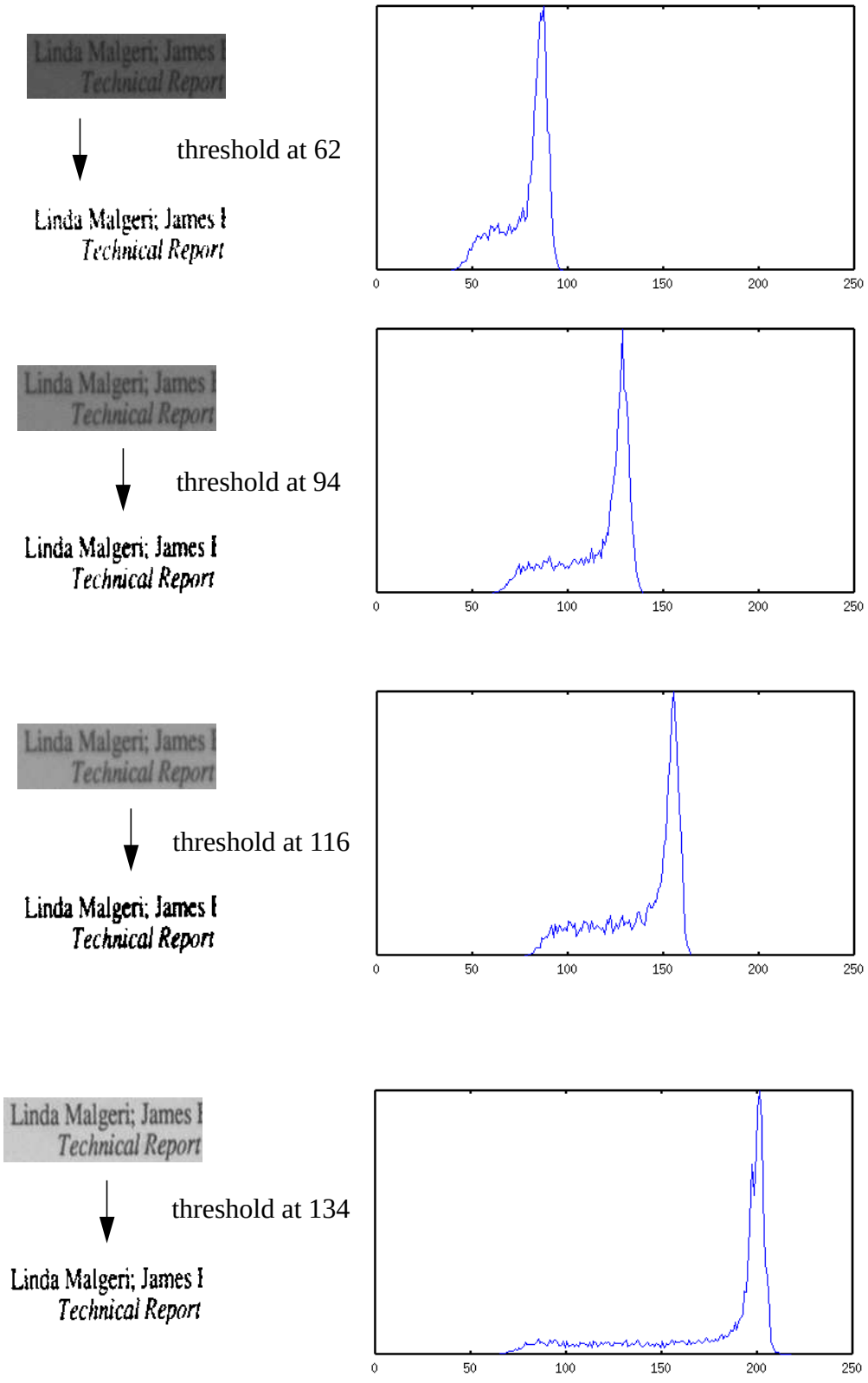


Figure 5-5: Histogram-based global thresholding at various intensities.

This histogram-based global thresholding technique works quite well for images that are evenly lit or, as in the examples above, for images that represent very small parts of paper where the light does not vary much. But it fails to produce good results for larger areas and in normal office lighting where the range of brightness varies greatly across the desk. Because a single global threshold value is used for the entire image, some parts of the image become too white and others too dark. Much of the text then becomes unreadable, as illustrated in Figure 5-6.

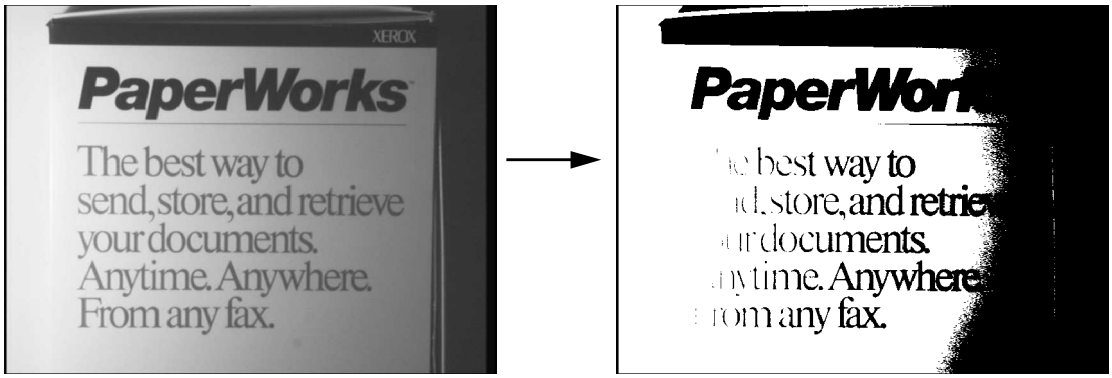


Figure 5-6: Global thresholding of unevenly lit image.

Producing good thresholded images from paper that is unevenly lit requires an *adaptive thresholding* algorithm. This technique varies the threshold value across the image according to the background illumination at each pixel. The discussion below is illustrated with the results of algorithms applied to the image in Figure 5-6. This is a challenging test because the image is lit from the side, and it has black text on a white background (the word “PaperWorks”), white text on a black background (“XEROX”), grey text on a white background (“The best way...”), various shadows, and a thin horizontal black line under the word “PaperWorks.”

Adaptive Thresholding

An ideal adaptive thresholding algorithm would produce the same result when applied to an unevenly lit page as a global thresholding algorithm would produce when applied to a perfectly evenly lit page. The brightness of each pixel is normalized to compensate for being more or less illuminated, and only then is a decision made as to whether the result should be black or white. The question then, is how to determine the background illumination at each point. One simple way to do this is to grab the image of a blank page before grabbing the page to be thresholded. This blank page can then be used as a reference image. For each pixel to be thresholded, the value of the corresponding pixel in the reference image is subtracted before applying the threshold.

This method produces very good results as long as the lighting does not change between the time when the reference image and subject image are each grabbed. On a desk, however, the lighting changes frequently as the user's shadow moves across the desk or doors, lamps and other objects are moved about. If there is a window in the room, then the lighting also changes throughout the day. One solution might be to immediately precede each grab with a reference grab of a blank page at the same location, but this would be almost as inconvenient to use as a scanner, thus defeating an important reason for using an overhead video camera in the first place.

Another solution is to try to estimate the background illumination at each pixel by making some assumptions about what the image should look like. We could assume, for example, that the image is mostly background (e.g. white), and that dark markings make up a smaller part of the image. Another assumption we could make is that the background lighting changes relatively slowly across the page compared to the changes in light and dark due to marks on page. Many algorithms based on such assumptions are possible. There is no mathematical theory of adaptive thresholding, and as a consequence, there is also no standard or optimal method for doing it [Prat91] (p. 597). Instead, there are several ad hoc methods, some of which are more popular than others. Because the methods are ad hoc, it would be useful to be able to measure their performance. For this, Haralick and Shapiro [Hara85] propose the following guidelines: regions should be uniform and homogeneous with respect to grey tone; region interiors should be simple and without many small holes; adjacent regions should have significantly different values; and boundaries of each segment should be simple, not ragged, and must be spatially accurate.

According to Pratt, no quantitative performance metric has been developed for image thresholding. It seems the main way to evaluate an algorithm is simply to look at the result and judge whether it looks right. For images of text, there is a quantitative possibility. The result of various algorithms under various lighting conditions could be fed to an OCR system, and the text produced from this can be compared to the original text. Although potentially useful, this approach was not systematically used for the algorithms described below, because it was judged unlikely to give a different evaluation from the “looks better” criteria.

For interactive applications of the DigitalDesk, users must wait for thresholding to finish when completing interaction techniques such as “copy and paste” of text or graphics. So another important quantitative performance metric is speed. The following sections describe various adaptive thresholding algorithms and show the results they produce.

Adaptive thresholding based on Wall's algorithm

One technique for calculating a threshold that varies across the image according to background illumination was developed by R. J. Wall and is described in [Cast79]. The following algorithm is loosely based on this description. First, it breaks up the image into smaller tiles and calculates a histogram for each tile. Based on the peaks of these histograms, a threshold is chosen for each tile. Then, every point in the image is assigned a threshold by interpolating between

the values chosen for each tile. Figure 5-7 was generated from the same grey image as Figure 5-6, but using this technique. The image was divided into nine

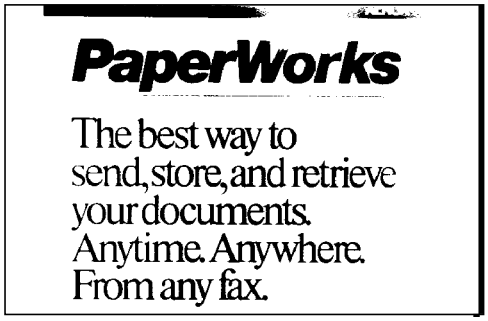


Figure 5-7: Result of adaptive thresholding based on Wall's algorithm.

tiles (3x3) and for each tile a threshold was selected 20% below the peak. From these values, 16 tiles were formed with their corners at the centre of each of the 9 tiles, and the threshold values were interpolated across each of these tiles from the corners. The result is much better than global thresholding, but because it requires more than one pass through the image, it is quite slow. Another problem with this technique is that with some images, the local histograms can be fooled by a large amount of black or white, causing the threshold not to vary smoothly across the image, and the result can be very bad (see in Figure 5-8.)

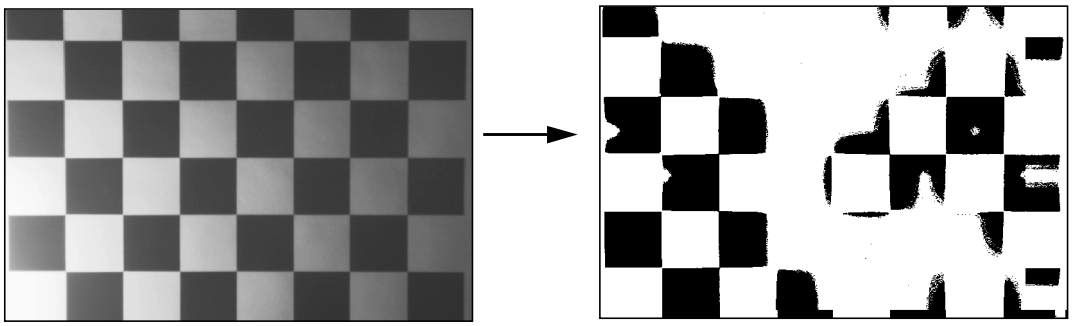


Figure 5-8: Local histograms can be fooled.

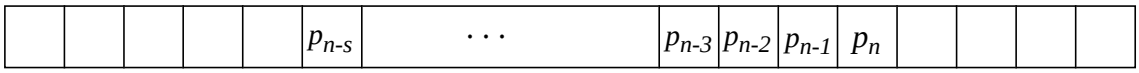
Quick Adaptive Thresholding

Most algorithms in the literature are more complex than Wall's algorithm, require more passes through the image and would take even longer to run *e.g.* [Hara85, Prat91, Wesz78]. It seems possible to implement a much simpler and faster algorithm for adaptive thresholding, so this section describes the steps taken so far in that direction for the DigitalDesk.

The basic idea is to run through the image while calculating a moving average of the last *s* pixels seen.When the value of a pixel is significantly lower than this

average it is set to black, otherwise it is left white. Only one pass through the image is necessary, and the algorithm is simple enough to be implemented in hardware. It is interesting to note the similarities between the following algorithms and the one developed at IBM using analog hardware in 1968 [Bart68].

Let p_n represent the value of a pixel at point n in the image being thresholded. For the moment we treat the image as though it were a single row of pixels com-



posed of all the rows in the image lined up next to each other. This gives rise to some anomalies when starting a new raster line, but less than starting each line from scratch (later we will carry information down as well as across).

Let $f_s(n)$ be the sum of the values of the last s pixels at point n.

$$f_s(n) = \sum_{i=0}^{s-1} p_{n-i}$$

The value of the resulting image T(n) is either 1 (for black) or 0 (for white) depending on whether it is t percent darker than the average value of the previous s pixels.

$$T(n) = \begin{cases} 1 \quad \text{if} \quad p_n < \left(\frac{f_s(n)}{s}\right)\left(\frac{100-t}{100}\right) \\ 0 \quad \textit{otherwise} \end{cases}$$

Using 1/8th of the width of the image for the value of s and 15 for the value of t seems to yield the best results for a variety of images. Figure 5-9 shows the

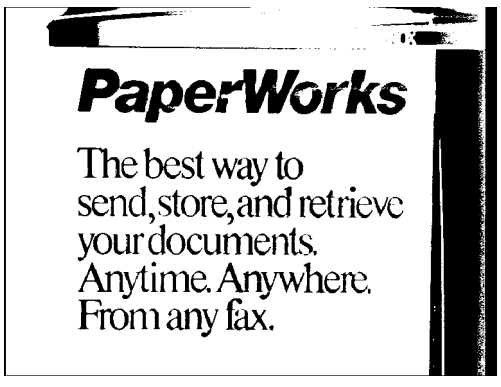


Figure 5-9: Moving average scanning from left to right.

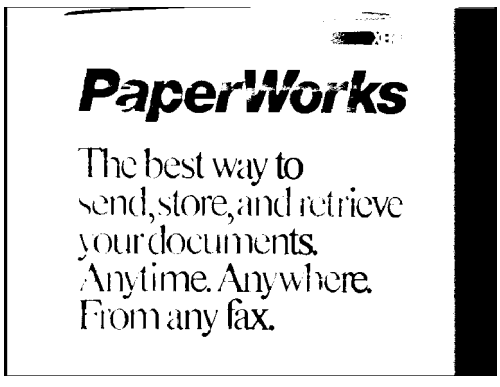


Figure 5-10: Moving average scanning from right to left.

result of using this algorithm scanning the rows of pixels from left to right. Figure 5-10 is exactly the same algorithm except the moving average scans from right to left. Notice in this image, that the left-most letters of the smaller text are incomplete, and there are more holes in the word PaperWorks. Also, the right-most black edge is much narrower. This is all because the background lighting of the image gets darker from left to right.

A fast way to calculate an approximate (weighted) moving average is to subtract 1/*s* part of it and add the value of only the latest pixel instead of using all *s* pixels. Thus *g(n)* is an approximation of *f(n)* where

$$\begin{aligned} g_s(n) &= g_s(n-1) - \frac{g_s(n-1)}{s} + p_n \\ &= p_n + (1-\frac{1}{s}) \cdot g_s(n-1) \\ &= p_n + (1-\frac{1}{s})\, p_{n-1} + (1-\frac{1}{s})^2 p_{n-2} \cdots \\ &= \sum_{i=0}^{n} (1-\frac{1}{s})^i p_{n-i} \end{aligned}$$

The main difference between f*(n)* and *g(n)* is that *g(n)* puts more weight on the pixels that are closest to *n*, which is good (in fact the weighting function is exponential). Notice that if all values of p_n are the same, then *g(n)* = *f(n)*. For example, if *s* = 2, then

$$g_2(n) = p_n + \frac{p_{n-1}}{2} + \frac{g_2(n-2)}{4}$$

and $g_2(n) = f_2(n) = 2p_n$ because $g_2(n-2) = 2p_n$.

A remaining question is how to start the algorithm, or what value to use for *g(0)*. One possibility is to use sp_0, but because of edge effects, p_0 is not always a representative value, so another possibility is 127*s* (based on the midvalue for 8 bit pixels). In either case, this choice only affects the first few values of *g*. The weight of g*(0)* relative to the total of all weights used in calculating $g_s(n)$ is

$$\frac{(1-\frac{1}{s})^n}{\sum_{i=0}^{n} (1-\frac{1}{s})^n}$$

so if $s = 10$, for example, then for any $n > 6$, $g(0)$ contributes less than 10% of $g_{10}(n)$; for any $n > 22$, $g(0)$ contributes less than 1%. For s = 100, the 10% level is passed after 8 pixels, and the 1% level is passed after 68 pixels.

The results from using this approximate moving average are similar (actually better) than the precise moving average, as can be seen in Figure 5-11 and Figure 5-12.

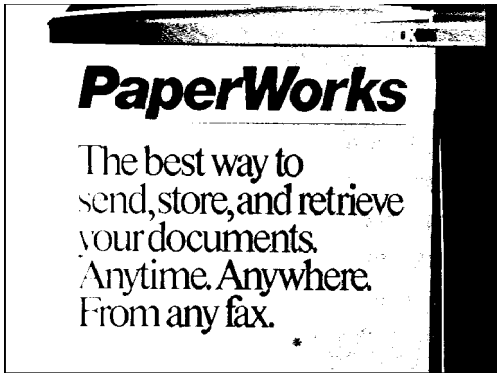


Figure 5-11: Approximate moving average scanning from left to right.

Figure 5-12: Approximate moving average scanning from right to left.

It would be good if the moving average did not work better with lighting from one direction or an other. Figure 5-13 shows the result of using another method to calculate the moving average. Instead of using the trailing line behind each pixel, it uses (an evenly weighted, not approximate) moving average that is centred about the n^{th} pixel. This uses a definition of $f(n)$ where

$$f_s(n) = \sum_{i=0}^{s-1} p_{n+\frac{s}{2}-i}$$

Using this method there are still some significant gaps in the letters, however, and it is also slower to calculate.

Figure 5-13: Centred moving average scanning from left to right.

Figure 5-14: Centred moving average scanning in alternate directions.

Instead of traversing the image from left to right or right to left, another possibility is to traverse it alternatively from the left and from the right as illustrated in Figure 5-15, addressing the line-end problem alluded to earlier.* It makes

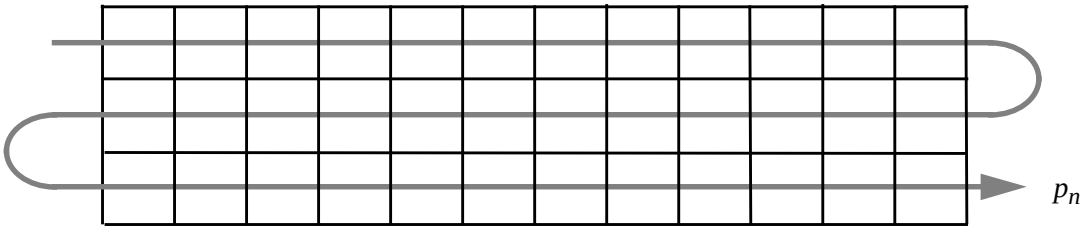


Figure 5-15: Traversing pixels in alternate direction every other line.

very little difference with the centred average (as seen in Figure 5-14), but if we go back to using the approximate moving average defined by *g(n)*, then alternate scanning improves things. The only anomaly is in the "grey" regions of an unevenly lit image, where going in one direction can produce the opposite result from going in the other, producing an every-other-line effect. (See Figure 5-16

* This way of scanning was called by the ancient Greeks *boustrophedon*, or "as the ox ploughs."

which is printed a little larger to show the effect better.) Superficially, this image

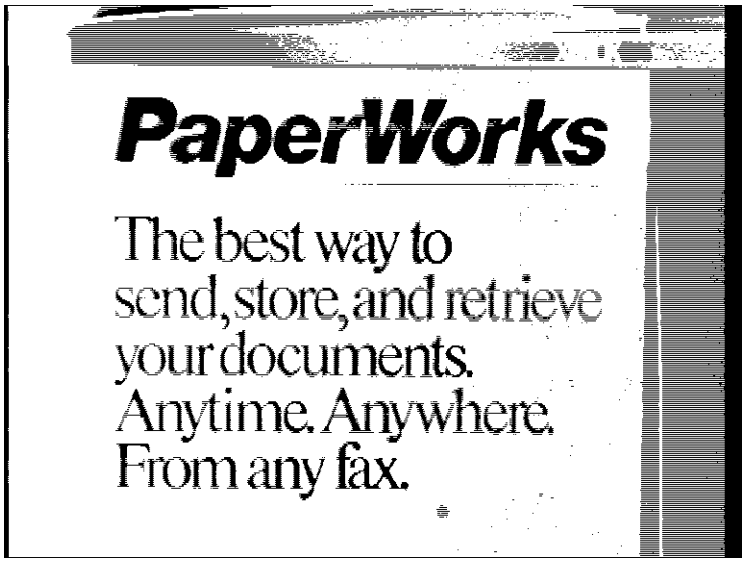


Figure 5-16: Moving average alternatively from left to right and from right to left.

may not look as pretty, but it is better than any of the previous algorithms in important ways because there are no big holes or parts of letters missing. The result is basically a combination of Figure 5-9 and Figure 5-10, but the right-hand shadow has the every-other-line effect because when coming from the left, the background gets darker and the intensity of the shadow falls below the threshold, producing black. When the moving average wraps around the edge and includes the dark black edge (due to over-scanning), the threshold falls below the intensity of the shadow and the result is white as it traverses from right to left.

A small modification of this algorithm gets rid of the every-other-line effect and produces consistently good results across a wide range of images. The modification is to keep the previous line of approximate averages (which were calculated by scanning in the opposite direction) and to average the current line's average with the previous line's average, so we use

$$h(n) = \frac{g(n) + g(n - width)}{2}$$

This lets the threshold take some account of the vertical axis and produces the results in Figure 5-17. Notice how well-formed all the letters are. Also, this is

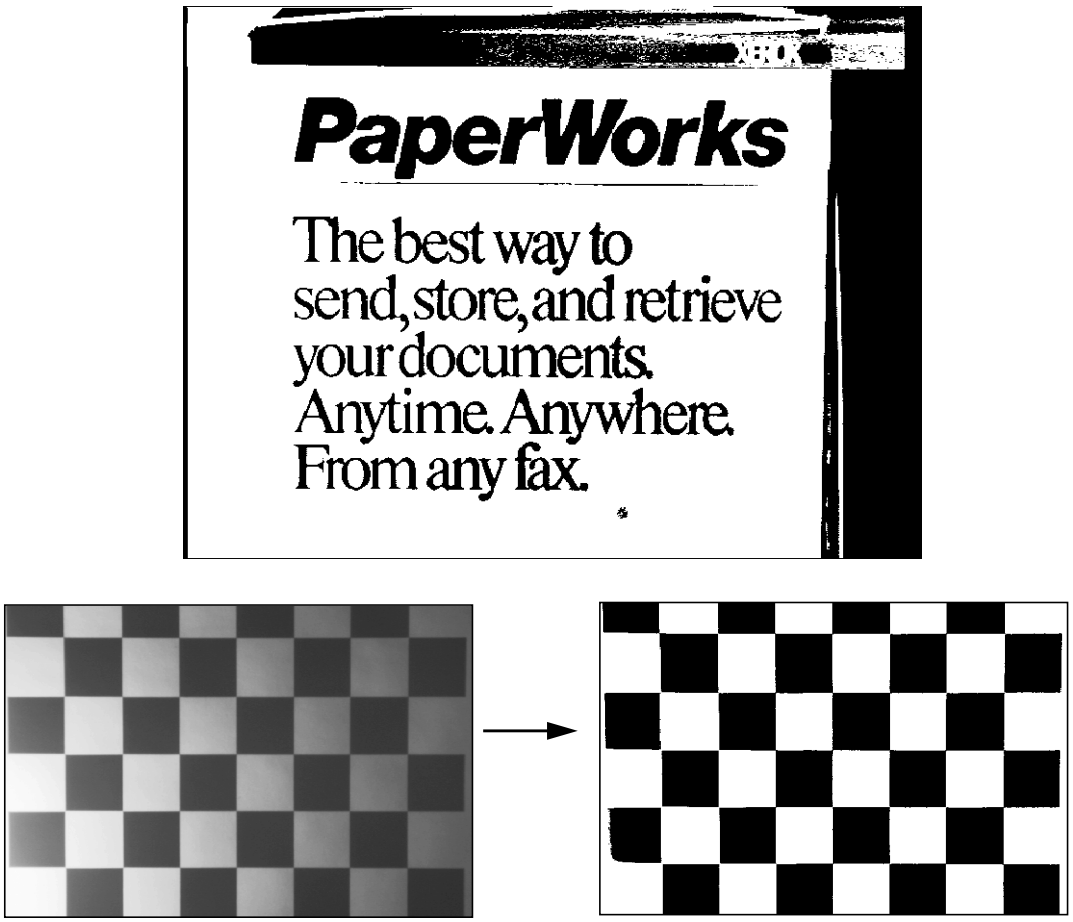


Figure 5-17: Moving average alternatively from left to right and from right to left and averaging the two together.

one of the few algorithms that does not eliminate the thin horizontal line below the word “PaperWorks.” Images whose intensity change mainly in the vertical direction are not as challenging as ones that change horizontally, but for the record, Figure 5-18 shows the result for the rotated image with the illumination

now varying from top to bottom. It seems unlikely that further developments

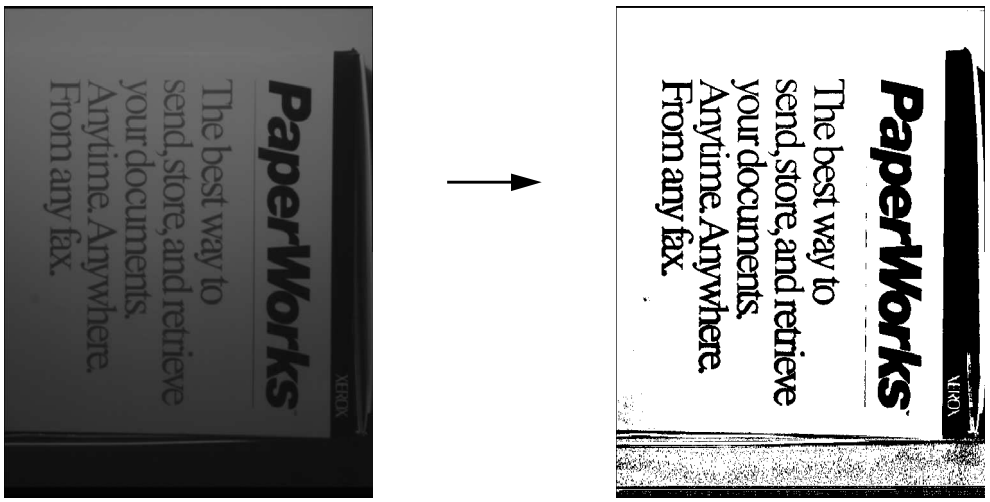


Figure 5-18: Thresholding an image with vertical lighting variation.

would produce a technique that generates significantly better results, so this is currently used for the DigitalDesk. The maximum amount of time it takes to run (for a full 768 x 575 image) is about 2 seconds on a SPARCstation 2 without paying special attention to optimization. Steve Freeman has modified this algorithm to go about 6 times faster by hard-coding multiplications and divisions to use powers of two so that shifting can be used instead. Further optimizations are possible, but will not make much difference for most interactions because they typically only operate on a small part of the image. For thresholded full-frame video as used by the DoubleDigitalDesk, and BrightBoard [Staf93] however, these optimizations will be more important.

Summary

This chapter describes the thresholding problem which must be overcome by DigitalDesk applications. A number of techniques are explored, leading eventually to a quick adaptive thresholding algorithm that has proven to be quite suitable for current purposes and which possibly could be implemented in hardware.

Chapter 6
Calibration

Introduction to the problem

The DigitalDesk has one or more cameras that point down at the desk, and users can select areas in a camera's field of view to perform operations such as "copy and paste" or character recognition on selected areas of paper documents. This chapter describes how the system supports the ability to indicate a specific point or region precisely in the frame-grabber's coordinate system through a pointing device and projected display.

Relative pointing devices such as a mouse or trackball allow the user indirectly to position a cursor and make selections on the screen. The absolute position of the pointing device does not matter because the user dynamically calibrates it with the display by superimposing the cursor on desired screen points. With absolute pointing devices, such as a transparent digitizing tablet or touch screen, the user points directly to the screen, so the pointing device must be calibrated to map the pen or finger position to screen coordinates. The DigitalDesk uses absolute pointing devices because it aims to be as much like an ordinary desk as possible. When we use paper documents on an ordinary desk, we do not need to look at a video monitor to see where we are pointing: we simply point directly at the paper.

The DigitalDesk requires a two step calibration process. First, absolute positions of the digitizing tablet must be mapped to positions on the display (e.g. to provide feedback when the user points at something). Second, positions on the display must be mapped to corresponding positions in the frame grabber (e.g. to support grabbing of selected areas on the desk). If finger tracking is used for pointing instead of a tablet, however, then only the second step is necessary.

There are two ways to calibrate an absolute pointing device to the display: let us call them dynamic and static. A dynamic approach is only possible with certain device-display combinations; it continuously adjusts the pointer to the display, much as a person does when positioning a cursor on screen with a mouse. Although this approach has promise, it was not tried with the DigitalDesk and is only briefly discussed at the end of this chapter. Instead, devices on the DigitalDesk are calibrated using a static method which has two essential steps:

1) Obtain a number of known *tie points* that map from device space to display space.
2) Calculate a *warping* that generalizes the mapping from the tie points to all points.

The following sections describe first how the tablet is calibrated to the display, and then how the display is calibrated to the video cameras.

Calibrating the tablet to the display

The system obtains tie points for calibrating the tablet and display by prompting the user with a cross-hair for each point at which the user points with the stylus. The displayed location of the crosshair is paired with the data returned by the tablet. Because this is a manual process, we would like the user to enter the minimum number of tie points necessary for accurate calibration. This number of points depends on the type of warping that needs to be calculated.

Two point warping

When both the pointing device and the display use rectangular coordinate systems, it is often enough to obtain just two sets of tie points: (x_1, y_1), (x_1', y_1') and (x_2, y_2), (x_2', y_2') as illustrated in Figure 6-1. From these, the system can

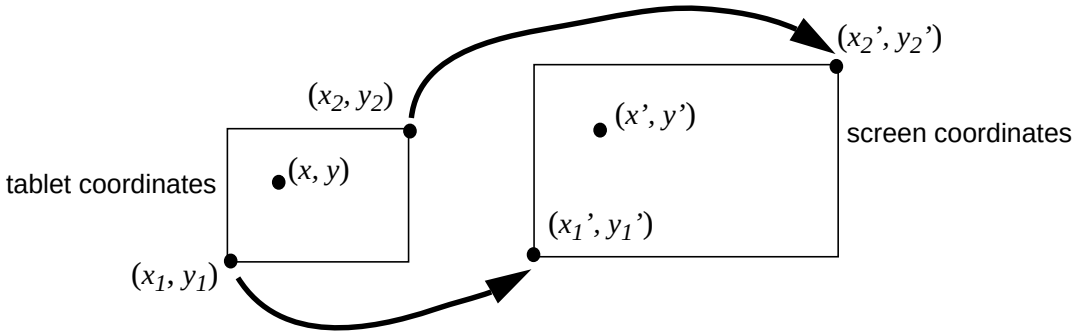


Figure 6-1: Two point warping.

map any device coordinate (x, y) to the corresponding screen coordinate (x', y') by using linear interpolation with the following simple formulae:

$$x' = \frac{(x - x_1)\ (x'_2 - x'_1)}{x_2 - x_1} + x'_1$$

$$y' = \frac{(y - y_1)\ (y'_2 - y'_1)}{y_2 - y_1} + y'_1$$

In practice with the DigitalDesk, this technique is only an approximation at best. Unfortunately, the projected display is not a perfect rectangle. There are optical distortions such as “keystoning” (see Figure 6-2), and it is difficult to rotate the

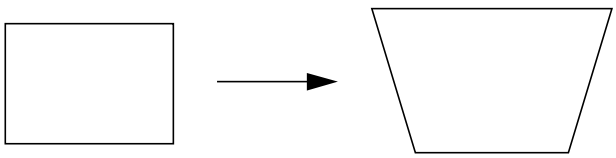


Figure 6-2: “Keystoning” due to projecting a rectangular image at an angle.

tablet and display so they are perfectly aligned. With this two point method, the tie points are mapped correctly, but as the pointer moves farther from a tie point, the cursor displayed at the warped screen coordinate moves farther away from the pen. The error depends on how large the optical and rotational distortions are, but it was not possible to maintain acceptable results with the current equipment, so more than two tie points are necessary. Calculating a warping from more than two points is a bit more complex and is described in the following sections.

Four point warping

If we obtain four pairs of tie points, as illustrated in Figure 6-3, then we can cal-

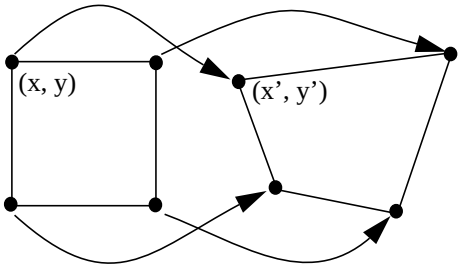


Figure 6-3: Four point warping.

culate a warping by using the following equations:

$$x' = c_1 x + c_2 y + c_3 xy + c_4$$

$$y' = c_5 x + c_6 y + c_7 xy + c_8$$

For each of these two equations, we can find the values of c_n by solving the set of four simultaneous equations obtained from plugging in values of the four tie points. These sets of simultaneous linear equations can be quickly and exactly solved (no need for statistical methods) by using Gauss-Jordan Elimination or Gaussian Elimination [Press88]. The following section shows how this solution not only compensates for translation and scaling of coordinate systems (as the two point warping does) but it also compensates for keystoning and rotation.

Why these particular equations? We need to have at least as many simultaneous equations as unknowns, so if we have only four points, then we must not have more than four unknown terms for *x*' or *y*'. Let us look at the six possible terms in a second order polynomial:

$$x' = c_0 + c_1 x + c_2 y + c_3 xy + c_4 x^2 + c_5 y^2$$

The diagrams below (Figure 6-4) illustrate the effect of increasing each term on a simple square centred about the origin. These only look at terms affecting the *x* coordinates, but effects on *y* coordinates are symmetrical.

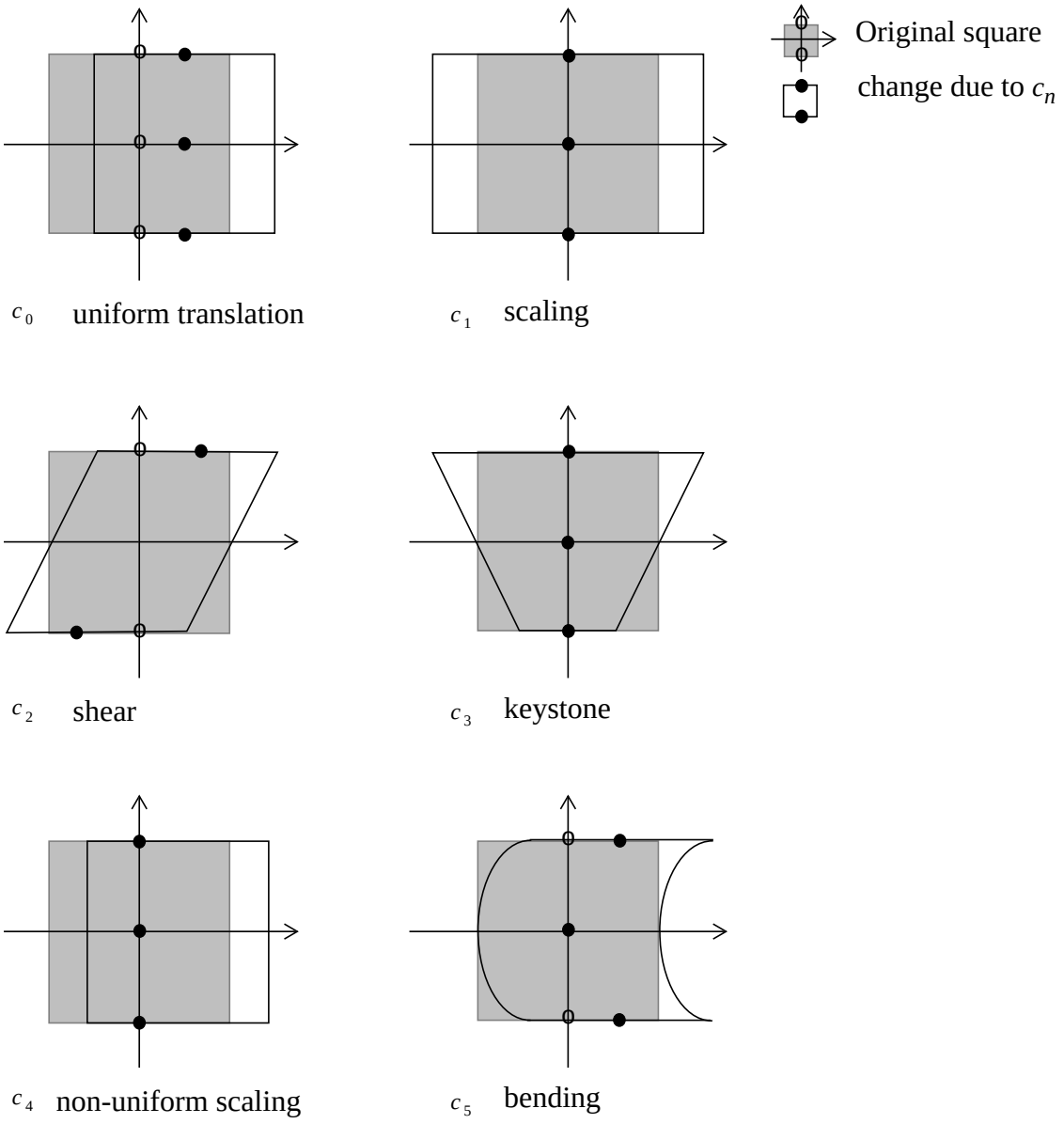


Figure 6-4: Second order polynomial distortions.

Of these terms, c_0, c_1, c_2, and c_3 are responsible for the most important distortions found in the projection system, so they were chosen as the terms to use in the warping equations. A shear in the *x* direction combined with a shear in the *y* direction rotates the coordinate space.

Another advantage of this four point warping is that it can be used to recursively break up a plane of coordinates to use as many points as necessary to make the warping more accurate over a larger area with more complex distortions. If four

points are not enough, then the plane can be broken up into more tiles, each of which has its own four point warping. Although it was the intention to do this for the DigitalDesk, four points worked well enough that more turned out not to be necessary. See appendix B for the interface to the Warping.T object.

Calibrating the frame grabber to the display

The projector displays feedback directly on the page, so users can see exactly what is selected, but how does the system know what pixels in the frame grabber correspond to the area displayed by the projector? A precise warping is necessary to map pixels in the projected display to pixels in the frame grabber.

Unfortunately, the same problems that make calibrating the tablet difficult (e.g. keystoning and rotation) also make it difficult to calibrate the camera. In addition, there are specific factors due to the camera: it may view the display at an angle, it may frequently need to be zoomed or pointed at a new position, and it is sensitive to vibrations caused by air conditioners or slamming doors.

Whereas obtaining tie points for calibrating the tablet is relatively straight forward, obtaining them for calibrating the camera is a little more complex. One obvious way to do this is to adjust the size and shape of a projected rectangle on the desk while looking at its image on a video monitor, and this rectangle can be aligned with the edges of the monitor. This provides only two tie points, and if there is any keystoning, rotation, or under-scanning by the video monitor the result is poor, even after performing this tedious manual procedure. To obtain multiple tie points, users could control a cursor on the digitized video image seen through the camera; then they could point at projected cross hairs in much the same way as users calibrate the tablet. A better approach, however, is for the system to locate these crosshairs without any assistance from the user: for the system to *self* calibrate. The camera is connected to an image processing system, so it precisely locates marks projected by the display, then for each mark the system knows the screen coordinates and its corresponding frame grabber coordinates. These coordinates are the tie points needed to calculate a warping.

Precisely locating a projected cross

The basic steps in locating a projected mark through the video camera are as follows:

1) Prompt the user to place a projected window on the desk to give the system an approximation of where the camera is pointing.
2) Inside each of the four corners and in the centre of this window:
   a) project a thick “plus mark;”
   b) pause to allow the display system (X windows and the LCD panel) to catch up;
   c) grab the image through the camera and threshold it;
   d) perform mathematical morphology operations on the image to locate the mark;
   e) pinpoint the centre of the mark.

3) Use the four corner point pairs located in this way to calculate a warping.
4) Use the warping to predict where the centre point should appear.
5) Test if the centre point is close to its prediction. If it's not then try again, else use the calculated warping.

Now let us go through each of these steps in a bit more detail:

Step 1: This step is necessary to ensure the system can project a mark that can be seen by the camera. The camera may be zoomed into a small part of the display, or it may be zoomed out to cover the entire display. The image processing system expects the apparent size of the marks (in frame grabber pixels) to be within a certain range, so if the camera is zoomed out, then the marks must be projected larger than if the camera is zoomed in. The size of the window the user sweeps out indicates to the system how large to make the projected marks. This manual step is not strictly necessary, because it is possible for the system to discover this approximate rectangle on its own, but this facility has not yet been implemented. A way to do it would be to start by grabbing two frames: one with the entire display white, and one with the entire display black. If there is no difference between the frames, then the camera cannot see the display and an error message would be generated. If there is a difference, then a smaller part of the display would be changed. The goal of this algorithm would be to find the largest rectangular area of the display on which a change is noticed by the camera, and where enlarging this area makes no additional difference.

Step 2: This step finds 5 tie point pairs (display coordinates $\leftrightarrow$ grabber coordinates). The four corner points are projected 1/4 of the distance in from each corner, and the 5^{th} test point is placed in the centre.

Step 2a: this mark is designed to be easily found using morphological techniques, even in the presence of substantial noise. Another approach that was tried was to XOR the mark into the display. The idea here was that no matter what was currently being displayed, the camera would be able to subtract the before and after images and detect a mark, and the hope was that the system could periodically (or continuously) self-calibrate even while it was being used.

This worked well on a background of mostly solid black and white areas, but because of the sometimes large difference in resolution between the camera and display, XORing the mark into a complex or patterned background often did not show up (on a fine gray background, even the human eye can hardly detect an XORed mark). Another problem with the mark was that it did not work well when projected onto paper with black marks on it. There is very little difference between projected black and projected white when projected onto black ink. So to make things simple and reliable, the current system requires the user to clear the area below the camera (e.g. by putting down a blank sheet of paper), and it only projects black marks onto a white background.

Step 2b: If the program does not pause long enough after displaying the mark and before grabbing the image, then the mark does not appear in the frame grabber, or it may appear only very faintly, because the LCDs have not had enough time to finish changing state.

Step 2c: Simply grab the entire field of view of the camera as a gray-scale image and threshold it using the quick adaptive thresholding technique described in the previous chapter.

Step 2d: Mathematical morphology and how it is used to locate calibration marks is described in the next section.

Step 2e: The end result of the morphological analysis is a bitmap with all bits clear except for the central part of the plus mark. This step finds the north-west and south-east corners of this small rectangle. If the corners are farther apart than expected, then something has gone wrong; *e.g.* more than one mark was in the camera's field of view. Otherwise, the midpoint between these two corners is used as the position for the mark.

Step 4: The four pairs of tie points are used to calculate a warping as described above, by solving the simultaneous linear equations. Then it is used to calculate where a mark displayed in the centre of the four displayed tie points should appear in frame grabber coordinates.

Step 5: The predicted location is compared to the measured location. This is done by calculating the sum of the squares of the differences between the x coordinates and the y coordinates. The current system uses 4 as the upper bounds for this. If something goes wrong during the calibration process (*e.g.* the user moves a piece of paper under the camera), and it was not detected before, then the problem is usually discovered at this point because this distance is too great. Even if a little noise, vibration, or changing lighting conditions caused inaccurate readings, then the system will go around and locate the five points again. In normal circumstances, however, the system usually finds the sum of the squares of the errors at this point to be 1 or 2, *i.e.* neither x or y coordinate is more than one pixel off.

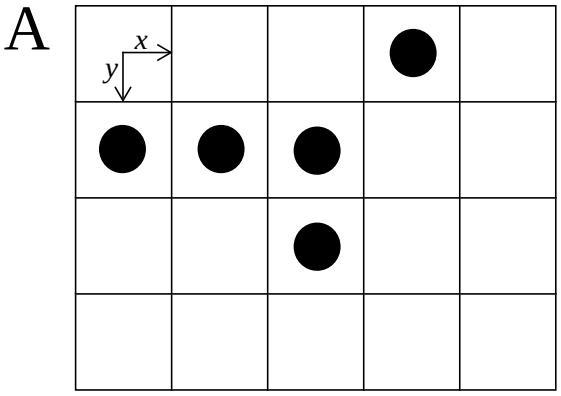
The original intention was for this algorithm to continue recursing down into each of the four quadrants and to continue refining the warping, but a single set of four points turns out to be more than precise enough to allow the user to select areas on the desk and rely on the displayed feedback to precisely indicate what will appear in the grabbed image. The process could be made significantly faster, however by displaying and locating all five points at once.

Mathematical Morphology

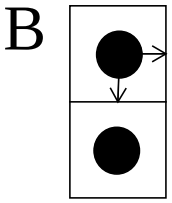
A proper description of mathematical morphology (sometimes called image morphology) is outside the scope of this chapter (for this see [Hara87]), but this section briefly introduces the concepts and walks through how they are applied to the problem of locating the calibration mark.

Mathematical morphology is an approach to image processing that is based on set theory and shape. Two basic operations are *dilation* and *erosion.* In the following discussion, A and B are sets of points that represent binary images; for

each black pixel in the image, the corresponding point is an element of the set (See Figure 6-5 for example).



A = {(0,1), (1,1), (2, 1), (2, 2), (3, 0)}



B = {(0,0), (0, 1)}

Figure 6-5.

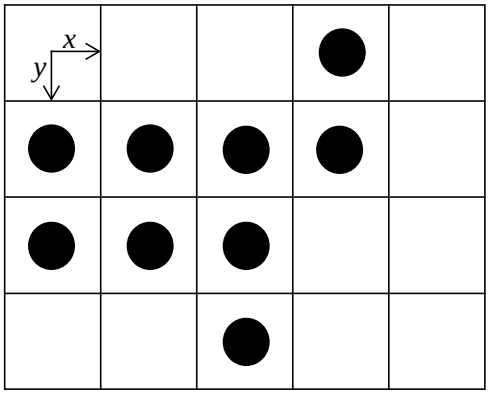
Dilation

The dilation of A by B is denoted by A ⊕ B and is defined below, treating the points *p*, *a*, and *b* as vectors for the purpose of addition.

$$A \oplus B = \{p \mid p = a + b \text{ for some } a \in A \text{ and } b \in B\}.$$

See Figure 6-6 for an example.

A ⊕ B



A ⊕ B = {(0,1), (1,1), (2, 1), (2, 2), (3, 0)
(0,2), (1,2), (2, 2), (2, 3), (3, 1)}

Figure 6-6.

Although A ⊕ B = B ⊕ A, by convention they are handled quite differently. The first operand A represents the image bitmap, and B represents a smaller *structuring element* which is thought of as a single shape parameter for the dilation transformation.

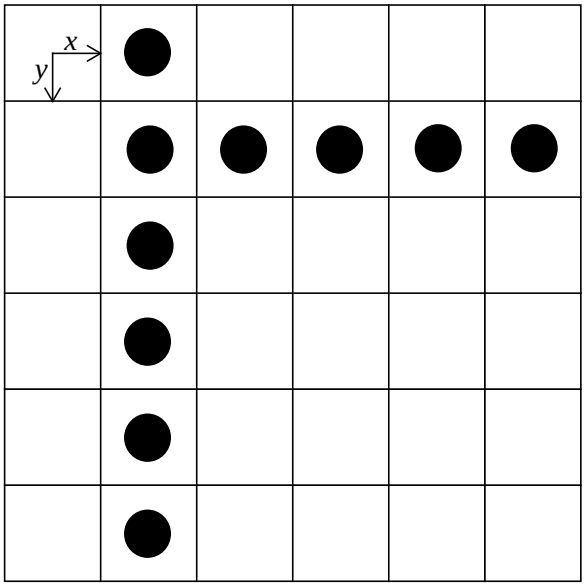
Erosion

The erosion of A by B is denoted by A θ B and is defined by

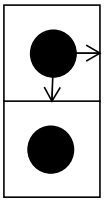
$$A \theta B = \{p \mid p + b \in A \text{ for every } b \in B\}.$$

See Figure 6-7 and Figure 6-8 for an example

A



B



B = {(0,0), (0, 1)}

Figure 6-7:

A = {(1,0), (1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (2, 1), (3, 1), (4, 1), (5, 1)}

A θ B

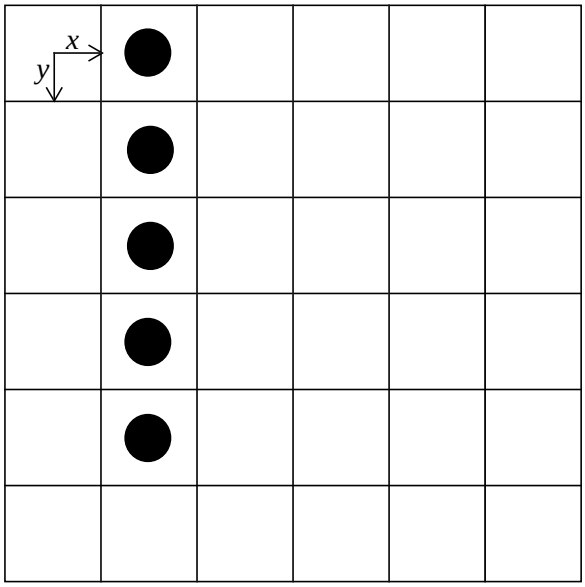


Figure 6-8:

A θ B = {(1,0), (1, 1), (1, 2), (1, 3), (1, 4)}

Opening and Closing

Dilation and erosion are often applied in pairs.

The *opening* of an image A by structuring element B is denoted by $A \circ B$ and is defined by

$$A \circ B = (A \ominus B) \oplus B.$$

The *closing* of an image A by structuring element B is denoted by $A \bullet B$ and is defined by

$$A \bullet B = (A \oplus B) \ominus B.$$

As proven in [Hara87], these two operations are idempotent, *i.e.* applying them a second time does not make further changes. Their effect is to eliminate specific details of the image smaller than the structuring element without distorting larger features. For example, opening an image with a disk structuring element smooths the contours, breaks narrow isthmuses, and eliminates small black islands. Closing an image with a disk structuring element also smooths the contours, fuses narrow breaks, and eliminates small white holes.

Finding the calibration mark

This section describes the use of morphological operations to find the calibration mark in the presence of noise and other bits of black in the image. The basic strategy is to eliminate all pixels in the image other than long horizontal areas and long vertical areas, then to look at the intersection between them. The code calls mathematical morphology routines implemented in C by Dan Bloomberg [Bloo90, Bloo91] through a Modula-3 interface.

Figure 6-9 illustrates the type of image (after thresholding) in which the system must locate the thick “plus” mark. This example is the picture of a final test

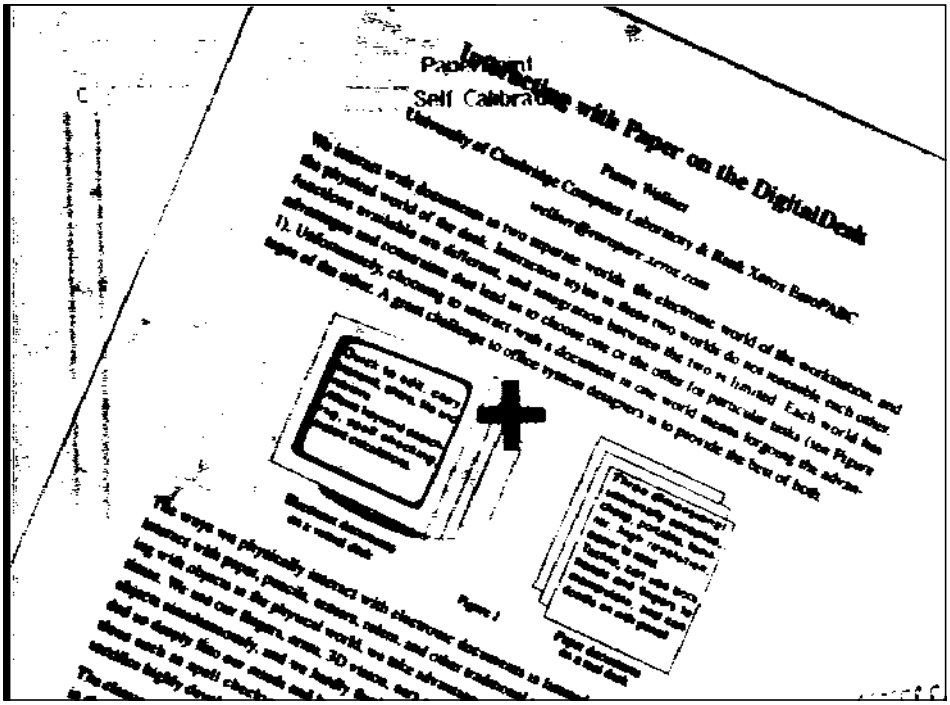


Figure 6-9: Thresholded image of the final test point.

mark projected in the centre of the calibration window to check if it appears in the location predicted. In this example, a paper with text and graphics was left on the desk under the camera to make the location of the cross a bit more challenging — normally it is best for the desk to be completely clear of extraneous marks. Also, the projected self calibration window is a bit too small for the field of view of the camera, so some projected text appears near the top of the image. In addition, the camera and display are slightly rotated relative to each other, as can be seen by the angle of the (fuzzy) projected window borders on the left side. All these features must be filtered out by the morphological operations in order to locate the cross in the centre.

The first step is to erode the thresholded image by a square structuring element 5 pixels by 5 pixels with its origin in the centre. This leaves a mark only in black areas at least this big, but removing 2 pixels off the top, bottom, left and right sides these areas. The size of the projected plus mark should be about 50 frame grabber pixels high and wide and 20 pixels thick to comfortably survive an erosion by this structuring element. Of course this could be any number of projector pixels depending on how far the camera is zoomed, so the user is asked to approximate the camera’s field of view with the self-calibration window, and the projected cross size is based on this window size. In an image less noisy than

this one, most other marks would disappear completely. The result is shown in Figure 6-10.

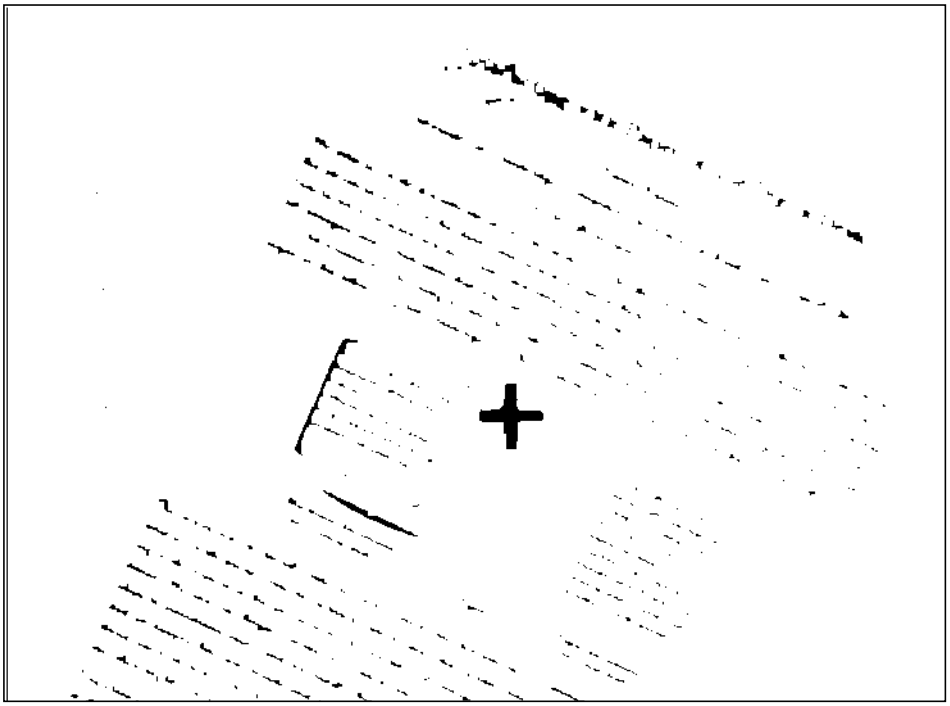


Figure 6-10: Result after encoding with a 5x5 structuring element.

The next step is to open the image with a structuring element 32 pixels tall, 1 pixel wide, and with its origin in the centre. This has the effect of removing all marks except those composed of vertical lines 32 or more pixels tall. The vertical stroke of the plus mark and the over-scanned left edge are the only features which survive this operation (see Figure 6-11). As long as the camera and display are not rotated more than about 30 degrees relative to each other, then this stroke should survive.

The following step is to open the eroded image with a structuring element 32 pixels wide and 1 pixel tall, preserving only marks that are 32 or more pixels wide (see Figure 6-12).

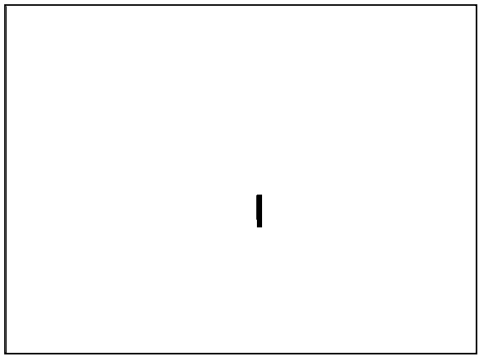


Figure 6-11

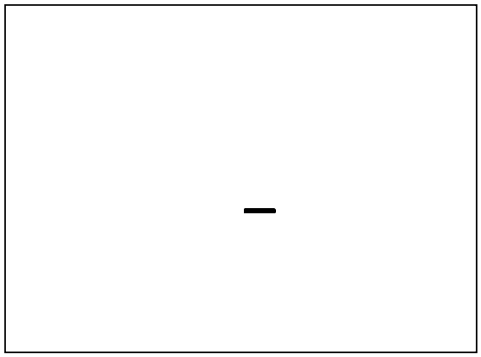


Figure 6-12

Finally, these two images are ANDed together to produce the result in Figure 6-13, where the only pixels remaining are from the centre of the plus mark. The

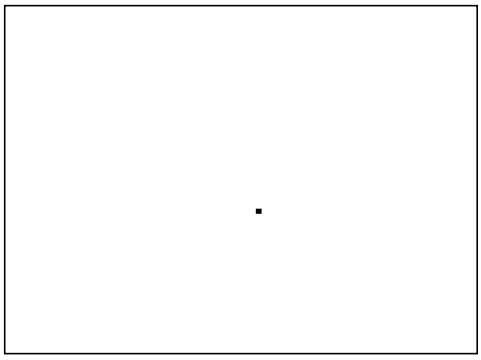


Figure 6-13

system uses the midpoint of this mark to determine the frame grabber coordinates of the displayed cross.

Dynamic Calibration

The DigitalDesk currently uses static calibration technique, with a warping that is initially calculated from a set of tie points and then used, unchanged, until the system is calibrated again. Although this works well with the current equipment and set of applications, the disadvantage of this method is that users must stop everything whenever the system needs to be recalibrated. A future implementation of the DigitalDesk may allow users to adjust the camera and the display frequently depending on the resolution required or on the part of the desk in use. If

users must frequently pause to recalibrate, this can make the system much less usable. It would be much better if the system could continuously calibrate itself without needing to stop working.

In principle, this should be possible with a fast enough image processing system. The key is for the system continuously to recognize and locate what it displays on the desk as part of the user interface. This can be especially useful when displaying dynamic feedback to the user such as a cursor or rubber-band rectangle.

Finger following might also be implemented with dynamic calibration instead of the current static method; the basic process would be simple. When the system locates the user's finger in the image processing system it would guess, based on current information, what display coordinate this position corresponds to, and the system would display a cursor at that point. It then would quickly grab another frame, locate this cursor and see how far it appears from the fingertip. Based on this error, it moves the cursor closer to the finger and checks again. To determine where to move the cursor, the system can use a crude warping (perhaps a two point warping) that is dynamically updated based on recent local cursor positions. The process is essentially the same one a person would use with a mouse to follow someone's finger pointing at a workstation screen. If this loop can be executed at video rates, the cursor might appear to follow the finger very smoothly and precisely. For this, not only a fast frame grabber and image processing system are necessary, but also an LCD panel that can change the state of its pixels at video rates. This method could work even while the camera and display are moving.

Summary

Calibration is necessary both for mapping the tablet to the display and for mapping the display to the frame grabber. In both cases, a variety of distortions must be dealt with, so a two point warping is not adequate. A four point warping based on solving simultaneous linear equations does a good job, however. A second order polynomial without the x^2 and y^2 terms works well, and it was not found necessary to recursively break the warping down into smaller parts. Mathematical morphology allows a projected mark to be accurately located on the desk even in the presence of noise, and these projected marks provide the four tiepoints necessary to automatically calculate the display to grabber warping with minimal user input.

Chapter 7
Future directions and conclusion

Future Directions

This chapter discusses potential directions for continuing work on the DigitalDesk and Augmented Environments. It describes a set of possible projects ranging from untested new ideas to relatively well understood software engineering. Some of these are simply continuation of work that has been started, and others represent relatively new directions. The DigitalDesk has opened up a large new design space, and future work should both continue to explore new regions of this space, and it should refine some of the specific interaction techniques and applications that have been introduced. Future work can focus on getting more out of each device and interaction technique, and it can continue to introduce new ones. The brief descriptions below illustrate some possibilities.

Getting rid of the tablet

A limitation with the current DigitalDesk system is that it requires a digitizing tablet to allow pointing at projected images. This currently makes it harder to use on whiteboards and ordinary desks. Clearly, a better way to point is with unencumbered fingers or pens as observed by the system through the video camera(s). Although early prototypes demonstrated the feasibility of finger following, the current implementation is too crude to be useful for real applications. Usable finger-following should be fast, accurate, and ideally should track multiple fingers at the same time. It may be easier to do this first by wearing something on the fingers (or finger nails) before implementing bare-finger following. It could also be very interesting to experiment with a version of Edgar's shadow parallax [Edga84], but using an infrared laser instead of a CRT. Sound and voice input could also help.

Tivoli

Tivoli [Pede93] has advanced pen-based interaction techniques, and it can run almost without modification on the DigitalDesk. But as it stands, Tivoli has no support for using real paper documents or real pens, markers, and erasers. An interesting project would be to incorporate the basic features of PaperPaint into Tivoli. This would allow users to sketch with real pens on real paper (or bring in preprinted documents), and paste the physical sketches into the electronic background of Tivoli. This may not require much modification of Tivoli or of the DigitalDesk software, and it would provide some of the best features of both in a single application. With finger following in place, this system could also be used to augment an ordinary whiteboard.

Real use studies

Some office tasks involve a lot of cutting and pasting of paper documents before distributing them. A clipping service, for example cuts out articles from various newspapers and magazines, collects them for specific clients, and makes copies for redistribution. A DigitalDesk application might make this task much easier (*e.g.* the "taking notes" scenario in the simulation video). It should be possible to find some people whose jobs could be made much easier by this application and to let them try a real working system, then study the system's impact on its users. Another option for a "real use" application might be a version of the translation system, or a collaborative application making use of the DoubleDigitalDesk.

Pen-based computing & PenPoint

Commercial pen-based computing is maturing, especially for notepad-sized computers, but its also an intrinsic component of the DigitalDesk and the LiveBoard. It may be that the best way of implementing the DigitalDesk (e.g for the application above) is on top of the PenPoint operating system. This would give DD applications all the pen-based interaction techniques and applications available in PenPoint such as character recognition, gesture recognition, and the notebook metaphor. Also, the embedded documents of PenPoint seem appropriate for the DigitalDesk. The aim of this system would be allow computer augmented interaction with physical notebooks in the same ways that PenPoint already supports electronic notebooks.

Electronic Annotation of paper documents

With a document identification system, the DigitalDesk could provide a way to make electronic annotations on paper documents that would be useful for a number of applications. A text book could be read in its paper form on the DigitalDesk, but the projector could add some electronic buttons to the paper pages that when pressed, follow hypertext links or play video clips. Another way of using this is for people who have copies of the same paper document to share their annotations. Each person would show his or her marked up paper document to the DigitalDesk, and it could project the marks that other people made on their copies of the document. EuroCode is beginning to explore some of these ideas now [ESPR93].

Multiple display surfaces of arbitrary size and resolution

Imagine that your drafting table, your walls, your whiteboards and even your ceiling could be a computer display as large and as high-resolution as you want. The LiveBoard and DigitalDesk have started exploring this concept, but it is still impractical to have large displays all over your office. This proposal is for a device and interaction technique that could make it practical.

A fundamental problem with large projected displays is the size *vs*. resolution trade-off: the larger the display, the lower the resolution. But, imagine that on your desk, your whiteboard and your walls there were electronic objects: *e.g.* computer applications, video screens, and electronic documents. Normally you would not see them, but when you point your DigitalDeskLamp at these surfaces, then the objects appear. If the lamp illuminates a large area then you cannot seem them in much detail, but all you need to do in order to see the objects

in more detail is to move the lamp closer. The electronic objects remain the same size and in the same location, but details appear, and tiny text becomes readable.

This display would require a small light-weight projector on an articulated arm, a means for sensing its position and orientation, and software to scale, translate, rotate, and otherwise distort the projected image. Ideally, these distortions should occur in real time response to projector movements.

Summary and conclusion

This dissertation has described the DigitalDesk, a computer augmented environment for paper that enables a set of novel human-computer interaction techniques. A number of working prototypes were developed that demonstrate both the utility and the feasibility of the DigitalDesk. These prototypes and the simulation video have generated significant interest at other research labs (*e.g.* at least five unsolicited invitations to give talks) and interest from the press (*e.g.* [Chev92, BBC92, Metc92, Wise92, Gale93 and Imai92]). The DigitalDesk has inspired other researchers to work in the same area (e.g. [Cart93, Mack93, ESPR93, Staf93 and Lope93]), and the DigitalDesk Toolkit has made it easier for them to do so by providing input device support, an architecture on which to build applications, and implemented solutions to the problems of thresholding and calibration.

Instead of making the electronic workstation more like the physical desk, the DigitalDesk does the opposite: it makes the desk more like the workstation, and it supports computer-based interaction with paper documents. Experience so far with this desk is encouraging. It enables people to do useful tasks with paper documents that are more awkward to do in other ways: tasks such as copying a long number into a calculator, translating a foreign word, replicating part of a sketch, or remote shared editing of paper documents. The interaction style supported by this desk is more tactile than “direct manipulation” with a mouse, and it seems to have a wide variety of potential applications, well beyond those described in this dissertation.

This work can be seen as a step towards better integration of paper documents into the electronic world of personal workstations, making paper-based information more accessible to computers. The motivation for this work, however, is just the opposite. The goal is not to enhance computers by giving them better access to paper; it is to enhance paper by giving it better access to computers. There is a difference between integrating the world into computers and integrating computers into the world. The difference lies in our perspective: do we think of ourselves as working primarily in the computer but with access to physical world functionality, or do we think of ourselves as working primarily in the physical world but with access to computer functionality? Much of the research in human-computer interaction seems to emphasize the former perspective, yet many useful ideas can be gained from the latter. Instead of making *us* work in the computer’s world, let us make *it* work in our world.

Appendix A
Modula-3 interface for Grabber

```
INTERFACE Grabber;

(**************************************************************************
 * Pierre Wellner
 * $Revision$
 *
 * The purpose of this object is to provide an abstraction for frame
 * grabbers that works with several different grabbers with different
 * resolutions and with multiple cameras attached to the same board.
 * Even though there may only be one physcial frame grabber, there can be
 * many Grabber.T objects, representing different frame grabber
 * configurations. These configurations are not actually applied to the
 * grabber until a call such as grabRect() is made (in this way a
 * Grabber.T is a bit like a graphics context). Some frame grabbers have
 * advanced features (such as image loopback) that are not supported by
 * the top-level object.
 *
 * Each Grabber.T object has a Warping. The purpose of this warping is so
 * that many instances of T can exist, corresponding to different cameras
 * and different zoom factors, etc, but clients can specify pixels in a
 * global coordinate space and each Grabber.T instance will warp them to
 * it's own coordinate space. On a digital desk with multiple cameras
 * pointing at it, the global coordinate space could be that of the
 * display. By default this Warping is equal to Warping.Identity.
 *
 * Each Grabber.T object also has a bounds rectangle that clips the points
 * after they have been warped. This clipping rectangle corresponds to the
 * field of view of the camera, and so limits the extent that can be
 * grabbed.
 *
 * If the Grabber's warping is NIL, GrabRect treats its argument as a
 * rectangle in the hardware coordinate space, and simply clips it to the
 * bounds and returns a Pixmap.Raw. This is equivalent to passing a
 * warping equal to Warping.identity
 *
 * If the Grabber's warping is non-NIL, then GrabRect() treats the Rect.T
 * as a parameter in global coordinate space. It Warps each corner of this
 * rectangle to hardware coordinate space and finds the smallest rectangle
 * that contains all four warped points. This new rectangle is then
 * clipped to the bounds actually covered by the grabber.
 *
 * To grab the maximum resolution image supported by the Grabber, you can
 * pass it the argument Rect.Full
 *
 * Normally grabRect should be called with the result argument set to its
 * default of NIL. If the result argument of grabRect is non-NIL, however,
 * then grabRect will fill result up and return it instead of allocating a
 * new Raw, but result must be big enough to hold the resulting image. One
```

```
 * way to create it, for example, could be by calling ScrnPixmap.NewRaw(8,
 * Grabber.getBounds()) or if you grab the same size image over and over in
 * a loop, you can do the following:
 * raw := grabber.grabRect(..., result := raw);
 * If raw starts out as NIL, the right amount of storage will be
 * allocated, and subsequent grabs will save the garbage collector from
 * having to work too hard.
 *
 * getBounds(rect: Rect.T) returns the bounds of the Pixmap.Raw that
 * would be returned if rect were passed to grabRect. The default
 * argument is Rect.Full which will return the size of the grabber in
 * hardware pixels. This method is useful to find out the size of an
 * image (if any) that would be grabbed when you know the rect in global
 * coordinate space. When the sampling factor is half or quarter, this has
 * the same effect as though the hardware was of lower resolution. Note
 * that half and quarter sampling factors to not always produce images
 * that are exactly half or quarter size of full images, so it is neccessary
 * to call getBounds() to find out. To find what the bounds of the grabber
 * correspond to in global coordinate space you would have to do
 * something like the following:
    Warping.FromPointPairs(
          grabber.getSettings().warping).unwarp(grabber.getBounds())
 *)

IMPORT Rect, ScrnPixmap, ExceptionArg;
IMPORT GrabberParms;

EXCEPTION Failed(ExceptionArg.T);

TYPE Settings = GrabberParms.Settings;

CONST
 DefaultSettings = Settings{
 port := 1,
 warping := NIL,
 imageType := GrabberParms.ImageType_Grey,
 sampling := GrabberParms.Sampling_Full,
 instanceName := NIL
 };
TYPE
 T <: Private;
 Private <: Public;
 Public = MUTEX OBJECT
 METHODS
 init(READONLY settings := DefaultSettings): T RAISES {Failed};
 getSettings(): Settings;

 getBounds(READONLY rect := Rect.Full): Rect.T RAISES {Failed};

 grabRect(READONLY rect := Rect.Full;
 result: ScrnPixmap.Raw := NIL
 ): ScrnPixmap.Raw RAISES {Failed};
 END;

END Grabber.
```

Appendix B
Modula-3 interface for Warping

```
INTERFACE Warping;

(*
 * Pierre Wellner
 *
 * A Warping provides a mapping between two coordinate spaces: one
 * “unwarped” and the other “warped.” The mapping is calculated based on
 * a set of tiepoints where each tiepoint is a PointPair that maps a
 * single unwarped point to a single warped point. The init() methods
 * determine how many tiepoints the Warping is based on, and return the
 * appropriate subtype of Warping.T. These tiepoints are then set by
 * the various set methods. The implementation of the warping
 * (or subtype returned by init) depends on the number of points. Only
 * 2 and 4 point warpings are currently supported, although this interface
 * is meant to support warpings based on arbitrary numbers of points.
 *
 * The Types Points and PointPairs should be the same as those in
 * GrabberParms.i3 generated from the .x file. Also see the file warping.x.
 *
 * T is a MUTEX and all methods lock self to prevent thread conflicts.
 *
 * ReadPointPairs() uses the Sx package to parse a lisp-like nested
 * parenthetical expression that specifies a list of PointPairs, e.g.
 *
 * ( ((0 0) (100 100)) ((10 10) (200 200)) ((1 2) (3 4)) ((5 6) (7 8)) )
 *
 * To initialize a Warping.T from a file, you could do something like this:

 TRY
 pointPairs := Warping.ReadPointPairs(warpingFile);
 w4 := NEW(Warping.T).init(NUMBER(pointPairs^));
 EVAL w4.setPairs(pointPairs^); (* if it’s not full then it will fail later,
 or you can check and raise an error here *)
 EXCEPT
 | Warping.Failed (arg) => Utils.PutText(arg.info);
 | Warping.BadFormat =>
 Utils.PutText(
 Fmt.F(“badWarpingformatfile\”%s\”\n”, warpingFilename));
 RTMisc.Exit(1);
 | Rd.Failure =>
 Utils.PutText(Fmt.F(“can’t read file\”%s\”\n”, warpingFilename));
 RTMisc.Exit(1);
 END;

 *)

IMPORT Point, Rect, Rd, Wr, ExceptionArg;
```

```
EXCEPTION
 BadFormat(Failure);
 Failed(Failure);

TYPE
 Failure = ExceptionArg.T BRANDED OBJECT END;
 FullFailure = Failure BRANDED OBJECT size: CARDINAL; END;
 NotReady = Failure BRANDED OBJECT minimum: CARDINAL; END;

TYPE
 PointPair = RECORD
 p : Point.T;
 wp: Point.T; (* warped point *)
 END;



 PointPairs = REF ARRAY OF PointPair;

 T <: Public;

 Public =
 MUTEX OBJECT
 METHODS
 init (READONLY numberOfTiepoints: CARDINAL): T RAISES {Failed};

 set (READONLY p, wp: Point.T): BOOLEAN RAISES {Failed};
 setPairs (READONLY tiepoints: ARRAY OF PointPair): BOOLEAN
 RAISES {Failed};
(*set and setPairs reset existing PointPairs if either the point or warped point
  equal a corresponding tiepoint. This means that it is impossible to have the
  same point tied to two different warped points or vice versa.
  These procedures return TRUE when the Warping is ready, FALSE otherwise.
*)

 getPairs (): PointPairs;
 (* returns a all tiepoints as a value suitable for passing to init *)

 warp (READONLY p: Point.T): Point.T RAISES {Failed};
 unwarp (READONLY pw: Point.T): Point.T RAISES {Failed};
 warpRect (READONLY rect: Rect.T): Rect.T RAISES {Failed};
 unwarpRect (READONLY rect: Rect.T): Rect.T RAISES {Failed};
 END;

PROCEDURE ReadPointPairs (sx: Rd.T): PointPairs
 RAISES {Failed, BadFormat, Rd.Failure};

PROCEDURE WritePointPairs (sx: Wr.T; pps: PointPairs)
 RAISES {Wr.Failure, Failed};

VAR
 Identity: T; (* initialized at start up and should never be changed.
 Treat this like a constant. *)

END Warping.
```

Bibliography

[Anst92] Anstey, Simon *A. Stylus-driven Symbol Recognizer.* Part II CST Project report, University of Cambridge Computer Laboratory, Cambridge UK.

[Akam92] Akamatsu, Motoyuki, and Sigeru, Sato "Mouse Type Interface Device with Tactile and Force Display: Multi-modal Integrative Mouse." *Proceedings of the Second International Conference on Artificial Reality and Tele-Existence* (ICAT'92). July 1-3, 1992, Tokyo, Japan.

[Asce92] Ascension Technology Corp. *The Ascension Bird*, Burlington VT, 1992.

[Azum93] Azuma, Ronald. "Tracking Requirements for Augmented Reality." *Communications of the ACM*. Vol 36 No 7, July 1993.

[Baec87] Baecker, R. and Buxton, W. *Readings in Human-Computer Interaction - A Multidisciplinary Approach*, p. 606. Morgan Kaufmann, Los Altos, California, 1987.

[Baju92] Bajura, M., Fuchs, H., and Ohbuchi, R. "Merging Virtual Objects with the Real World: Seeing Ultrasound Imagery within the Patient." Computer Graphics, 26, 2, July 1992.

[Bart68] Bartz, M.T. "The IBM 1975 Optical Page Reader. Part II: Video Thresholding System." IBM Journal of Research and Development, Sep. 1968; pp. 354-363.

[BBC92] British Broadcasting Corporation. *Tomorrow's World*, 29th January, 1992.

[Beck89] Beck, K., Cunningham, W. "A Laboratory for Teaching Object-Oriented Thinking" *Proceedings of OOPSLA'89*, October 1-6 1989.

[Bewl83] Bewley, W. L., Roberts, T. L., Schroit, D., Verplank, W. L. "Human Factors Testing in the Design of Xerox's 8010 "Star" Office Workstation." In *Proceedings of CHI'83,* ACM, New York 1983.

[Bier93] Bier, Eric A. and Buxton William, "Toolglass" SIGGRAPH '93, Aneheim California.

[Bles88] Bleser, Teresa W., Sibert, John L, and McGee, Patrick "Charcoal Sketching: Returning Control to the Artist" ACM Transactions on Graphics, vol. 7, no. 1, January 1988, pp 76-81.

[Bloo90] Bloomberg, D., and Maragos, P. "Generalized Hit-Miss Operations" In *Proceedings of SPIE Conference 1350: Image Algebra*

and Morphological Image Processing, San Diego, CA, July 1990. (Also available as Xerox PARC Technical Report EDL-90-1.)

[Bloo90b] Bloomberg Dan S. “Self-Clocking Embedded Digital Data.” US Patent 5,168,147.

[Bloo91] Bloomberg, D. “Connectivity-Preserving Morphological Image Transformations.” Xerox PARC Report EDL 91-1 Xerox Palo Alto Research Center, Palo Alto California (1991).

[Brit92] Brittan, David “The Promise of Multimedia Connections,” Technology Review, May/June 1992, MIT.

[Broo88] Brooks, Frederick P. “Grasping Reality Through Illusion - Interactive Graphics Serving Science.” In *Proceedings of the Conference on Computer and Human Interaction* (CHI ’88) 1988

[Broo90] Brooks, F.P., Ouh-Young, M., Batter, J.J., Kilpatrick, P.J. “Project Grope -- Haptic Displays for Scientific Visualization.” *ACM Computer Graphics*, vol. 24, no. 4, August 1990, pp. 177-185.

[Buxt85] Buxton, W., Hill, R., and Rowley, P. “Issues and Techniques in Touch-Sensitive Tablet Input.” SIGGRAPH ’85 Vol, 19, no. 3, San Fransisco, July 22-26 1985.

[Bux86] Buxton, W., and Myers, B. “A Study in Two-Handed Input.” In *Proceedings of the Conference on Computer and Human Interaction* (CHI ‘86) 1986.

[Buxt90] Buxton, W. “Smoke and Mirrors.” Byte vol. 15 no. 4 July 1990.

[Carr91] Carr, Robert, and Shafer, Dan. *The Power of PenPoint*, Addison-Wesley, Reading, MA 1991.

[Cart93] Carter, K. “Computer Aided Design: Back to the drawing board.” In *Proceedings of Creativity and Cognition*, Loughborough, Apr. 1993.

[Cast79] Castleman, K. *Digital Image Processing*. Prentice-Hall Signal Processing Series, 1979.

[Chev92] Chevin, Denise “The art of finger-painting elevated to the science of communication.” *Building* magazine, August 28, 1992, p. 44.

[Chun89] Chung, J.C., Harris, M.R., Brooks, F.P., Fuchs, H., Kelley, M.T., Ouh-young, M., Cheung, C., Holloway, R.L., and Pique, M. “Exploring virtual worlds with head-mounted displays.” In *Proceedings of SPIE* Vol 1083 Three-Dimensioal Visualization and Display Technologies, 18-20 January 1989.

[Cush90] Cushman, W., Ojha, P.S., and Daniels, C.M. “Usable OCR: What are the Minimum Performance Requirements?” In *Proceedings of CHI’90*. April 1-5 1990, Seatle, Washington.

[Data92] Data Cell, *ITEX s2200 Programmer’s Manual*. Data Cell Ltd. Reading, UK 1992.

[Dewi82] DeWitt, T., and Edelstein, P., “Pantomation: A system for position tracking.” In *Proceedings of Second Symposium on Small Computers in the Arts*. Oct 1982.

[Edga84] Edgar, Albert D. “Apparatus and Method for Remote Displaying and Sensing of Information using Shadow Parallax.” United States Patent 4,468,694 Aug. 28, 1984.

[Elro92] Elrod, S. Bruce, R, *et.al*. “Liveboard: A large interactive display supporting group meetings, presentations, and remote collaboration.” In *Proceedings of CHI‘92* (Montrey, Calif. May 3-7, 1992), ACM, New York 1992.

[Elro93] Elrod, S., Hall, G., Costanza, R., Dixon, M., des Rivieres, J. “Responsive Office Environments” *Communications of the ACM*. Vol 36 No 7, July 1993.

[Eric91] Erickson, T., and Salomon, G. “Designing a Desktop Information System: Observations and issues.” In *Proceedings of CHI’91*. ACM, New York 1991.

[ESPR93] ESPRIT Project 6155 “EuroCODE: A CWCW Open Development Environment.” 1993.

[Fein93] Feiner, S., MacIntyre, B., and Seligmann, D. “Knowledge-Based Augmented Reality.” *Communications of the ACM*. Vol 36 No 7, July 1993.

[Fish86] Fisher, S., McGreevy, M., Humphries, J., Robinett, W. “Virtual Environment Display System.” ACM 1986 Workshop on Interactive 3D Graphics, October 23-24 Chapel Hill, North Carolina.

[Fitz93] Fitzmaurice, George W. “Situated Information Spaces and Spatially Aware Palmtop Computers.” *Communications of the ACM*. Vol 36 No 7, July 1993.

[Floy76] Floyd, R. W. and Steinberg, L. “An Adaptive Algorithm for Spatial Greyscale.” In *Proceedings of the S.I.D.* 17,2 (Second Quarter 1976); pp 75-77.

[Fran91] Francik, E., Rudman, S., Cooper, D., and Levine, S. “Putting Innovation To Work: Adoption Strategies for Multimedia Communication Systems.” *Communications of The ACM*. vol 34, no 12, (December 1991).

[Free93] Freeman, Steven M. G. *An Architecture for Distributed User Interfaces* PhD Dissertation, University of Cambridge Computer Laboratory, Pembroke Street, Cambridge, England, 1993.

[Fren90] Frenckner, T. “Legibility of Continuous Text on Computer Screens - A Guide to the Literature.” TRITA-NA-P9010 IPLab-25, June 1990. Department of Numerical Analysis and Computing Science, Royal Institute of Technology, S-100 44 Stockholm, Sweden.

[Gale93] Gale, David. “Meeting in a Virtual World.” *New Scientist*, 13 March 1993.

[Gave92] Gaver, W., Moran, T., MacLean, A., Lövstrand, L., Dourish, P., Carter, K. and Buxton, W. “Realizing a video environment: EuroPARC’s RAVE system.” In *Proceedings of CHI‘92* (Montrey, Calif. May 3-7, 1992), ACM, New York 1992.

[Gave93] Gaver, W. “Synthesizing Auditory Icons” In *Proceedings of INTERCHI‘9*3 (Amsterdam April 24-19, 1993).

[Gett90] Gettys, J., Karlton, P., and McGregor, S. “The X Window System, Version 11.” *Software Practice and Experience* October Vol 20 No 10, 1990.

[Gold91] Goldberg, D. and Goodisman, A. “Stylus User Interfaces for Manipulating Text.” In *Proceedings of the ACM Symposium on User Interface Software and Technology* (UIST ‘91), November 11-13, Hilton Head 1991.

[Good91] Goodisman, Aaron. *A Stylus-Based Interface for Text: Entry and Editing*. Master’s thesis, Massachusetts Institute of Technology Department of Electrical Engineering and Computer Science, May 1991.

[Gosl89] Gosling, J., Rothenthal, D., and Arden, J. *The NeWS Book: an introduction to the Network Extensible Window System*, Springer-Verlag, New York 1989.

[Hala82] Halasz, F. and Moran, T. “Analogy Considered Harmful.” In *Proceedings of Human Factors in Computer Systems*, March 15-17, 1982 Gaithersburg, Maryland.

[Hamm87] Hammond, J.H. and Austin, J. *The Camera Lucida in art and science*. Adam Hilger, Bristol 1987.

[Hans88] Hansen, W. J., and Haas, Christina. “Reading and Writing with Computers: A Framework for Explaining Differences in Performance.” *Communications of the ACM*, vol. 31, no. 9, September 1988.

[Hara85] Haralick, Robert M. and Shapiro, Linda G. “Image Segmentation Techniques.” Computer Vision Graphics and Image Processing, 29 January 1985; pp. 100-132.

[Hara87] Haralick, R., SternBerg, S., Zhuang, X. “Image Analysis Using Mathematical Morphology,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-9, no. 4, July 1987; pp. 532-550.

[Hecht90] Hecht, David L. “Method and means for reducing bit error rates in reading self-clocking glyph codes.” US Patent 5,221,833.

[Hews90] Hewson, R. “Sketching by numbers: is typographic design possible in the electronic paradigm?” Center for Information Technology in Education (CITE) Report no. 108. Open University, Walton Hall, Milton Keynes MK7 6AA UK.

[Imag89] Imaging Technology. *Series 100 Product Brief: Real-Time Digital Image Processor* (47-B10006-08) Imaging Technology Incorporated Corporate Communications Department, Woburn, Massachusetts 1989.

[Imai92] Imai, Takuji. “The Next Generation User Interface: From GUI to Multimodal.” *Nikkei Electronics* July 20, 1992, p. 121. (Japanese version); September 1992, pp. 24-25 (English version).

[Ishi91] Ishii, Hiroshi and Miyake, Naomi. “Toward an Open Shared Workspace: Computer and Video Fusion approach of TeamWorkStation.” *Communications of the ACM*, vol. 34, no. 12 (December 1991).

[Ishi92] Ishii, H., Kobayashi, M, Grudin, J. “Integration of Inter-Personal Space and Shared Workspace: ClearBoard Design and Experiments.” In *Proceedings of the ACM Conf. on Computer Supported Cooperative Work*, (CSCW’92) Toronto, September 1992.

[John85] Johnson, Jeff. “The Desktop Metaphor as an Approach to User Interface Design.” (Panel) In *Proceedings of ACM Annual Conference*, Denver, Colorado, Oct 14-16 1985.

[John87] Johnson, Jeff. “How Faithfully Should the Electronic Office Simulate the Real One?” *SIGCHI Bulletin*, October 1987 vol. 19, no. 2.

[John89] Johnson, J., Roberts, T., Verplank, W., Smith, D., Irby, C., Beard, M., Mackey, K. “The Xerox Star: A Retrospective.” *IEEE Computer* vol. 22, no. 9, September 1989.

[John93] Johnson, W., Jellinek, H., Klotz, L., Rao, R., Card, S. “Bridging the Paper and Electronic Worlds: The Paper User Interface.” In *Proceedings of INTERCHI‘93* (Amsterdam April 24-19, 1993).

[Kay91] Linderholm, Owen “Mind Melding: How far can the human/computer interface go?” (Interview with Alan Kay). Byte Special Edition Vol 16, No 11, 1991.

[Know77] Knowlton, K. “Computer Displays Optically Superimposed on Input Devices.” Bell System Technical Journal, vol. 56, no. 3, March 1977.

[Koba92] Kobayashi, M. and Ishii, H. “DispLayers: Multi-Layer Display Technique to Enhance Selective Looking of Overlaid Images” CHI’92 Poster session, 1992.

[Krue83] Krueger, M. *Artificial Reality.* Addison-Wesley, 1983.

[Krue85] Krueger, M. “VIDEOPLACE -- An Artificial Reality.” In *Proceedings of CHI ’85*.

[Krue91] Krueger, M. *Artificial Reality II*. Addison-Wesley, 1991.

[Krue93] Krueger, M. “Environmental Technology: Making the Real World Virtual.” *Communications of the ACM*. Vol 36 No 7, July 1993.

[Laki89] Lakin, F., Wambaugh, J., Leifer, L., Cannon, D., and Sivard, C. “The electronic design notebook: performing medium and processing medium.” *The Visual Compter,* Springer-Verlag pp. 214-226 1989.

[Lamm92] Lamming, Mik and Newman, W. “Activity-based Information Retrieval: Technology in Support of Personal Memory” *Personal Computer and Intelligent Systems*, Information Processing ’92 Volume III Elsevier, 1992 IFIP.

[Lope93] Lopez-Welsch, Lionel, “A Painting Tool for Real Paper.” University of Cambridge Computer Lab diploma project report, 1993.

[Luff92] Luff, P., Heath, C., and Greatbach, D. “Tasks-in-interaction: Paper and screen based documentation in collaborative activity.” In *Proceedings of the ACM Conf. on Computer Supported Cooperative Work,* (CSCW’92) Toronto, September 1992.

[Mack93] Mackay, W., Velay, G. Carter, K., Ma, C. and Pagani, D. “Augmenting Reality: Adding Computational Dimensions to Paper” *Communications of the ACM*. Vol 36 No 7, July 1993.

[Magg93] Maggioni, Christoph “A Novel Device for Using the Hand as a Human Computer Interface.” In *Proceedings of HCI’93*, Loughborough, England, 1993.

[Malo83] Malone, Thomas W. “How Do People Organize Their Desks? Implications for the Design of Office Information Systems.” *ACM Transactions on Office Information Systems*, Vol 1, No. 1, 1983.

[Mana91] Manasse, Mark S. and Nelson, Greg. *Trestle Reference Manual*. Digital Equipment Corporation Systems Research Center, Palo Alto California. December 1991.

[Metc92] Metcalfe, Bob "WANs, MANs, LANs, and now DANs: Desk Area Networks." *Communications Week* Feb 24, 1992.

[Meye92] Meyer, Kenneth and Applewhite, Hugh "A Survey of Position Trackers." *Presence*, vol. 1, no. 2, Spring 1992.

[Mill91] Miller, R. "Behold the Humble Barcode." *Multimedia and Videodisk Monitor.* Arlington VA, Sept. 1991.

[Minn91] Minneman, S.L. and Bly, S.A. "Menaging a Trois: a study of a multi-user drawing tool in distributed design work." In *Proceedings of the Conference on Computer and Human Interaction* (CHI '91), New Orleans, LA, April 28-May 2, 1991.

[Minn93] Minneman, S.L. and Harrison, S.R. "Where Were We: Making and Using Near-synchronous, Pre-narrative Video." In *Proceedings of ACM Conf. on Multimedia*, Anaheim California, 1993.

[Mins90] Minskey, M., Ouh-Young, M., Steele, O., Brooks, F.P., Behensky, M. "Feeling and Sensing: Issues in Force Display." ACM Computer Graphics, vol. 24, no.2, March 1990, pp. 235-243.

[Monk92] Monkman, G.J. "An Electroheological Tactile Display." *Presence* Vol 1. No 2. Spring 1992.

[Mori91] Morita, H., Hashimoto, S., and Ohteru, S. "A Computer Music System that Follows a Human Conductor." *IEEE Computer*, July 1991.

[Nels91] Nelson, Greg, *Systems Programming with Modula-3*, Prentice-Hall, Englewood Cliffs, NJ 1991.

[Newm92] Newman, W., and Wellner, P. "A Desk Supporting Computer-based Interaction with Paper Documents." In *Proceedings of CHI'92* (Montrey, Calif. May 3-7, 1992), ACM, New York 1992.

[Niel93] Nielsen, Jakob, "Noncommand User Interfaces" *Communication of the ACM* vol. 36, No 4. 1993.

[Nye90] Adrian Nye (ed.) *Xlib Reference Manual* O'Reilly & Associates 1990.

[Olso90] Olson, J.S., Olson, G.M., Mack, L.A. Wellner, P.D. "Concurrent Editing: The Group's Interface." In *Proceedings of the IFIP TC 13 Third International Conference on Human-Computer Interaction* (INTERACT'90) Cambridge, UK 27-31 August, 1990. (North Holland).

[Patr91] Patrick, M. Sachs, G. *X11 Input Extension Library Specification,* MIT X Consortium Standard X Version 11, Release 5, 1991.

[Pavl90] Pavlidis, T., Swartz, J., and Wang, Ynjiun "Fundamentals of Bar Code Information Theory." *IEEE Computer,* April 1990; pp. 74-86.

[Pede93] Pedersen, E.R., McCall, K, Moran, T.P., Halasz, F.G. "Tivoli: An Electronic Whiteboard for Informal Workgroup Meetings." In *Proceedings of INTERCHI'9*3 (Amsterdam April 24-19, 1993).

[Penn92] Pennisi, Elizabeth "Talking Maps: Technologies to give the visually impaired a sense of space." *Science News* vol. 142, no. 23. December 5, 1992.

[Pina93] Pina, Larry "Third Generation Home Controller: HomeBase offers X-10 plus infrared control." *Electronic House,* September-October 1993.

[Prat91] Pratt, W. *Digital Image Processing*. John Wiley & Sons, 1991.

[Pres88] Press, William H., *et.al. Numerical Recipes: The art of Scientific Computing.* Cambridge University Press, 1988.

[Pitt91] Pittman, James A. "Recognizing Handwritten Text." In *Proceedings of the Conference on Computer and Human Interaction* (CHI '91), New Orleans, LA, April 28-May 2, 1991.

[Purc85] Purcell, Patrick "Beyond the Keyboard." *Computer Bulletin*, September 1985.

[Refl92] Private Eye product litterature, Reflection Technologies, Waltham Mass 1992.

[Resn93] Resnick, Mitchel "Behavior Construction Kits" *Communications of the ACM*. Vol 36 No 7, July 1993.

[Rhei91] Rheingold, H. *Virtual Reality.* Secker & Warburg 1991.

[Rhyn87] Rhyne, Jim. "Dialogue Management for Gestural Interfaces." *Computer Graphics*, vol. 21, no. 2 April 1987.

[Schm83] Schmandt, C. "Spatial Input/Display Correspondence in a Stereoscopic Computer Work Station." *Computer Graphics*, vol. 17, no. 3, July 1983.

[Schr92] Schrage, Michael "Xerox's challenge: Intelligently integrating paper into digital world." *San Jose Mercury News*, Monday, August 17, 1992.

[Sege93] Segen, Jakub “Controlling Computers with Gloveless Gestures.” in *Proceedings of Virtual Reality Systems 93*, New York City, March 15-17, 1993; pp. 2-6.

[Seyb92] Seybold, A. “The DOCIT.” *Andrew Seybold’s Outlook on Professional Computing* 11, 2 (Sept 1992).

[Shne83] Shneiderman, B. “Direct Manipulation: A Step Beyond Programming Languages.” *IEEE Computer*, August 1983.

[Smit82] Smith, D.C., Irby, C., Kimbal, R. and Harslem, E. “The Star user interface: an overview.” In *Proceedings of National Computer Conference* 1982 pp. 517-528.

[Staf93] Stafford-Fraser, Quentin. *Controlling Computers by Video*, First Year Report and Project Proposal, University of Cambridge Computer Laboratory, July 30,1993.

[Stur91] Sturman, David J. *Whole-hand Input*. PhD Thesis, Massachusetts Institute of Technology December 20, 1991.

[Sun90a] Sun Microsystems. *NeWS 2.1 Programmer’s Guide*, Sun Microsystems Inc., Mountain View, California, 1990.

[Sun90b] Sun Microsystems. *4.1 Pixrect Reference Manual*, Sun Microsystems Inc., Mountain View, California, 1990.

[Sun91] Sun Microsystems. *Using VideoPix* (Part Number 800-5099-10). Sun Micorsystems Inc, Mountain View, California, 1991.

[Suth65] Sutherland, Ivan E. “The Ultimate Display.” In *Proceedings of IFIP Congress 65*. New York City, May 24-29 1965.

[Suth68] Sutherland, Ivan E. “A head-mounted three dimensional display.” In *Proceedings of Fall Joint Computer Conference*, 1968.

[Tang91a] Tang, J., Minneman, S. “VideoDraw: A Video Interface for Collaborative Drawing.” *ACM Transactions on Information Systems* vol. 9, no. 2, April 1991, pp. 170-184

[Tang91b] Tang, J., Minneman, S. “VideoWhiteboard: Video Shadows to Support Remote Collaboration.” In *Proceedings of the Conference on Computer and Human Interaction* (CHI ‘91), New Orleans, LA, April 28-May 2, 1991; pp. 315-322.

[Tani92] Tani, M., Yamaashi, K., Tanikoshi, K., Futakawa, M., and Tanifuji, S., “Object-Oriented Video: Interaction with Real-World Objects Through Live Video.” In *Proceedings of the Conference on Computer and Human Interaction* (CHI’92), (Montrey, Calif. May 3-7, 1992), ACM, New York 1992.

[Tapp88] Tappert, C.C., Suen, C.Y., Wakahara, T. “On-line Handwriting Recognition - A Survey” *9th International Conference on Pattern Recognition*. Rome, Italy 14-17 November 1988.

[Tyso92] Tyson, Philip J. “The Desk as a Social Institution” EuroPARC Technical report 1992.

[Vinc90] Vincent, J. V., MacDougall, F. “The Mandala System.” CHI ’90 Interactive experience, April 1-5 1990. Seattle, Washington.

[Wein92] Weintraug, D.J. and Ensing, M. *Human Factors Issues in Head-Up Display Design: The Book of HUD*, CSERIAC State-of-the-Art Report, Wrigh-Patterson Air Force Base, Ohio.

[Well91] Wellner, P. “The DigitalDesk Calculator: Tangible Manipulation on a Desk Top Display.” In *Proceedings of the ACM Symposium on User Interface Software and Technology* (UIST ‘91), November 11-13, Hilton Head 1991.

[Well92] Wellner, P. “Tactile manipulation on the DigitalDesk.” (video) In *CHI’92 Special Video Program, ACM SIGGRAPH Video Review* no. 79.

[Well93a] Wellner, P. Mackay, W., and Gold, R. “Computer-Augmented Environments: Back to the Real World.” Guest editors’ introduction to special issue of *Communications of the ACM*. Vol 36 No 7, July 1993.

[Well93b] Wellner, P. “Interacting with Paper on the DigitalDesk” *Communications of the ACM*. Vol 36 No 7, July 1993.

[Weis91] Weiser, M. “The Computer for the 21st Century.” *Scientific American*, September 1991.

[Weis93] Weiser, M. “Some Computer Science Issues in Ubiquitous Computing.” *Communications of the ACM*. Vol 36 No 7, July 1993.

[Wesz78] Weszka, Joan S. “A Survey of Threshold Selection Techniques.” *Computer Graphics and Image Processing*, 7 April, 1978; pp. 259-265.

[Wide90] Widener, G. “The X11 Inter-Client Communication Conventions Manual.” *Software Practice and Experience*. Vol 20(S2) October 1990.

[Wise92] Wise, Deborah “New lease of life for desks and in-trays.” *The Guardian*, Thursday April 23, 1992, p. 33.

[Wolf89] Wolf, C. G., Rhyne, J. R., and Ellozy, H. A., “The paper-like interface.” In *Designing and Using Human-Computer Interfaces and*

Knowledge Based Systems, eds. Salvendy, G. and Smith, M.J., Elsevier, Amsterdam 1989.

[Xero92] Xerox, *Using PaperWorks from a Fax Machine*. Xerox Corporation, Palo Alto, CA, 1992.

[XIS93] Xerox Imaging Systems *ScanWorX API Programmer's Guide* (Part no. 00-07570-00), Peabody, Massachusetts, Feb. 1993.