Pixel Data Access for End-User Programming and Graphical Macros

Richard Potter, Ben Shneiderman, and Ben Bederson
Human-Computer Interaction Laboratory
UMIACS and Department of Computer Science
University of Maryland
College Park, MD 20742.
E-mail: {potter, ben, bederson}@cs.umd.edu

ABSTRACT
Pixel Data Access is an interprocess communication technique that enables users of graphical user interfaces to automate certain tasks. By accessing the contents of the display buffer, users can search for pixel representations of interface elements, and then initiate actions such as mouse clicks and keyboard entries. While this technique has limitations it offers users of current systems some unusually powerful features that are especially appealing in the area of end-user programming.

KEYWORDS: End-User Programming, Programming by Example, Pixel Data Access, Interprocess Communication, Graphical Macros.

INTRODUCTION

Users of many systems appreciate the power they have to create novel automations that extend the power of these systems. Macros in word processors such as Emacs, WordPerfect, and Word enable users to record frequent sequences of actions for repeated replay and save them for future use. Macros can automate tasks such as making back-up files, merging complex lists, or converting formats. Macros are widely used in spreadsheets such as Excel and can be useful in presentation tools such as PowerPoint or operating systems such as Windows.

However, macros, scripting languages and other end-user programming tools are often limited because they put a burden on the application developer. End-user customization is often considered an "extra" and is left out, or only supported in minimal ways.

A good solution typically requires all or most of the application's internal functions to be wrapped and made available to the macro program. This can be a significant amount of work, and has the potential to create significant maintenance problems when internal features change. One of our goals is to create a powerful end-user customization system that does not require this effort of the application builder.

Pixel Data Access directly accesses pixels in the display buffer. This paper reports on how pixel data access can be combined with exact pixel pattern matching, programming by demonstration and the simulation of mouse and keyboard actions to make an end-user programming system that can be used with any application. Application developers do not have to go to any effort to make these techniques work because they take advantage of the already existing graphical user interface.

These ideas have been validated with Triggers-III, a prototype system that runs on the Macintosh OS, and by a user study of 6 computer science students. In addition, we have learned from the current implementation, and designed a new version that addresses several questions of visibility. The new design (Triggers-IV) is described in detail in [20].

Although pixel data access could be combined with higher level techniques, Triggers-III has shown it can do a surprising amount when combined with only keyboard and mouse actions. We have used it to do graphical search and replace in drawing and paint programs, circle negative number in a spreadsheet program, and use a word processor to convert a folder full of files from one format to another. The ability for pixel data access to periodically look at the screen makes it possible to write programs that trigger on a variety of conditions, such as the appearance of a particular dialog box or user interface widget. We have used this ability to simulate floating menus and snap-dragging alignment techniques[4]. Although the techniques require significant computer sophistication to use, we anticipate that they would appeal to users who appreciate being able to learn a relatively simple system that can automate tasks in any application.

PREVIOUS WORK

Pixel data access provides additional interprocess communication for end-user programming, a need that operating system developers have long recognized. Apple Computer provides Apple Events [1] for the Macintosh OS, which can be used with the AppleScript and other programming systems. Microsoft supports OLE Automation [5] for the Windows OS, which can be used with Visual Basic and other programming systems. Formal solutions such as these are developed especially for end-user programming. They support not only the communication of information between user programs and existing applications, but also define standard ways for applications, such as word processors, to define their data objects, such as characters, words, and paragraphs. Formal solutions also define ways for applications to document what functionality they expose to interprocess communication, so that users can browse the functionality in a standardized way when writing programs.

Apple Events also defines a way for applications to communicate information about a users manual actions. This allows programming systems to record user actions. There have been research proposals to extend this technique to communicate the information as a hierarchy of events so that the programming system can have information to more accurately infer generalized programs [8]. Making an application "recordable" by using these techniques is powerful but requires even more effort from application developers. Other researchers are investigating ways to simulate recording capabilities by polling applications that support interprocess communication [12].

Interprocess communication only requires that the receiving process be able to monitor some entity that the sending process can affect. To be dependable, the entity must also be stable. Formal solutions have the advantage that the entity stays stable by design. Thus, when they are available and the applications support them, they usually provide the most dependable solution. The disadvantage is that developers of applications must go to extra effort to support the formal solutions. Many applications do not support formal interprocess communication, and few application support it completely

When formal protocols are not sufficient, developers and researchers have found other entities that can fill the role, even those not originally designed for interprocess communication. The keyboard device driver has been used by many automation utilities including Tempo II and QuicKeys to simulate keystrokes and thus communicate with applications. A search for "macro" on any shareware Web site will reveal dozens of similar utilities. Many of these also connect to the mouse driver to simulate mouse actions. These solutions have proved useful for providing communication in the direction from user programs to applications. However, they are unable to provide communication in the direction from applications to user programs. Some of the utilities also connect to standard graphical user interface widgets to simulate selections at a higher level. Widget-level solutions achieve a limited amount of communication from applications to user programs. While they can get the contents of text boxes and read items in menus, they cannot process custom-rendered content such as a drawing or even a browser page.

Pixel data access is related to these solutions because it connects to the display buffer, an entity originally not meant for interprocess communication. Because it is a low-level solution, it can access information that is not available to widget-level solutions. Other researchers are exploring pixel data access. VisMap [21] is a system that has been successful using pixel data access to recognize common Windows user interface widgets. Although it provides higher-level functionality to the user program, it does provide programming by demonstration techniques to help the user map the pixel data to new uses. AutoMouse [22] is similar to our research in that its programs work at the device level, but it does not allow programmatic control of a search area, a technique that will be shown to be useful later in the paper. Another related system is Expect [11], which performs analogous data access reading stdout for interactive Unix commands such as telnet and ftp. However, Expect can not work with graphical user interfaces.

Our research shows that pixel data access can be used in a programming by demonstration system. Previous work in programming by demonstration [7] has required that a system be integrated with applications at a higher level, which has limited the application of programming by demonstration to research systems. Our system, in contrast, can be used with any application. However, our system does not provide all the benefits that these other systems can achieve with higher-level data. For example, Eager [6] allows even non-programmers to be successful in creating programs. Chimera [9,10] provides many useful techniques for automating graphical editing tasks such as graphical search and replace and a comic strip style graphical history that can be used to compose and generalize graphical macros. Many of these systems also provide some form of inference to automatically make programs more general [6,14,15,16]

LESSONS FROM DEVICE-LEVEL KEYBOARD MACROS

The main design issue for pixel data access is how to process information in pixel form. The strategy used in Triggers-III is a generalization of the strategy successfully used in device-level keyboard macros. Keyboard macros allow a user to record keystrokes into a simple program called a macro. When the macro is run or "played back", it simulates the keystrokes. From an interprocess communication perspective, keyboard macros are programs that can communicate with any application that responds to keystrokes, which are low-level input into a user interface. Analogously, pixels are the low-level output from a graphical user interface. However, recording and playing back, which works so well with keystrokes, is difficult to apply directly to pixels. Therefore a more general understanding of keyboard macros is necessary.

A keyboard macro system is a simple programming system with one control structure (sequential), one data type (keystroke), and one operator (send keystroke). Although its programs contain only device-level data and operators, they can perform tasks in many domains, because the user interfaces of various applications can map the keystrokes to higher-level roles. Therefore one lesson that can be learned from keyboard macros is that a small amount of device-level functionality can result in programs that can perform a surprising range of useful tasks.

A second lesson is that with the right user interface, a user can create these programs easily. Keyboard macros require that the user be aware of the mapping from device-level operators to higher-level roles. The user must also verify that the assumptions under which the mappings are valid are reasonable for the specific task situation. Keyboard macro systems help the user handle these responsibilities with programming by demonstration [7]. Since programs are created by recording, the user can see many relevant details of the application while choosing each keystroke of the macro. Therefore lesson two is that with programming by demonstration it is possible to provide context that can help the user choose the device-level operators that map to the higher-level task.

A third lesson from keyboard macros is that programming by demonstration can bring other benefits. A keyboard macro system can be extremely simple and still be useful, because a keyboard macro system essentially reuses the user interfaces of applications for creating. The classic keyboard macro system can be useful with only three commands: start recording, stop recording, and play macro. For special cases, it is possible to simplify the interface to one repeat command [13]. Interfaces this simple can be mastered such that it is possible for an experienced user to write macros without ever seeing any extra user interface. The user can focus almost all attention on aspects of the application and task. Although a keyboard macro system can provide these benefits of programming by demonstration, it does not automatically generalize programs, which has been a popular goal for programming by demonstration research [6,14,15,16].

DESIGN OF TRIGGERS-III

The three lessons from keyboard macros provide the design guidelines for using pixel data access in Triggers-III. First, include generic device-level data types and operators. Second, allow the user to choose device-level operators in meaningful contexts. Finally, maximize the programming by demonstration benefits. Figure 1 summarizes the features in Triggers-III as influenced by the guidelines.

I. Device-Level Functionality
- a. Data Types
    - • Pixel Images
        - - to store patterns and screen images
    - • Points
        - - to store locations that mark instances of patterns
    - • Rectangles
        - - to store areas in which to search for patterns
- b. Operators
    - • Vector Arithmetic
        - - to shift locations or areas
    - • Pixel Pattern Match
        - - to find a pattern within an area of the screen
    - • Keyboard and mouse macros
        - - to manipulate and send information to applications

II. User Interface for Mapping from Device-Level to Task Domain
- a. Data Types in pixel-image context
    - • Pixel Images
    - • Points
    - • Rectangles
- b. Operators in pixel-image context
    - • Vector Arithmetic parameters
    - • Pixel Pattern Match parameters
    - • Macro parameters

III. Programming by Demonstration
- • Desktop Blanket
    - - to toggle quickly between applications and programming system
    - - to provide screen images for context
- • Pixel Pattern Match shortcut
    - - to specify patterns and search areas by direct manipulation
- • Keyboard and mouse macro recording
    - - to specify keyboard and mouse actions by demonstration
- • Trigger control structure
    - - to specify an ordered set of rules

Figure 1: The Features in Triggers-III.

The main piece of device-level functionality in Triggers-III for processing pixel data is the *pixel pattern match operator*. It searches for a rectangular pixel pattern in a rectangular area of the computer display buffer. The pixel pattern match operator returns a value indicating whether a match was found and its location. Matches must be identical to the pattern. Pixel pattern matching is a device-level operator because it operates on the device-level *form* of the information rather than the higher-level *content*. For example, to match the word "Cancel" in a button on the screen requires matching the actual pixels on the screen that make up that word, *not* a text string of those six characters. As will be shown in the example programs that follow, the pixel pattern match operator can map to many task-domain roles, similar to how the sending of keystrokes can map to many roles.

The *pixel image*, *rectangle* and *point* data types support the input and output parameters of the pixel pattern match operator. Pixel images are used to store pixel patterns. Rectangles are used to designate which area of the display buffer to search. Points are used to store the locations of the found instances of the patterns. To provide complementary interprocess communication, Triggers-III includes operators to simulate keyboard and mouse actions. *Vector arithmetic* is used to shift rectangles based on the location of another pixel pattern and to shift point locations for mouse actions.

Following the second guideline, Triggers-III allows a user to edit device-level details in context by way of the *Desktop Blanket*. The Desktop Blanket allows widgets to be displayed above the graphical user interfaces of other applications. The user raises the Desktop Blanket by pressing the control and options keys together. The Desktop Blanket displays as a light gray grid across the entire display and above all other applications, which show through the grid as in Figure 2. The front application is not deactivated, which is important since deactivating it would change its pixel-level appearance. The control and option key combination acts as a toggle, so pressing it again restores the screen as if the Desktop Blanket were never raised.

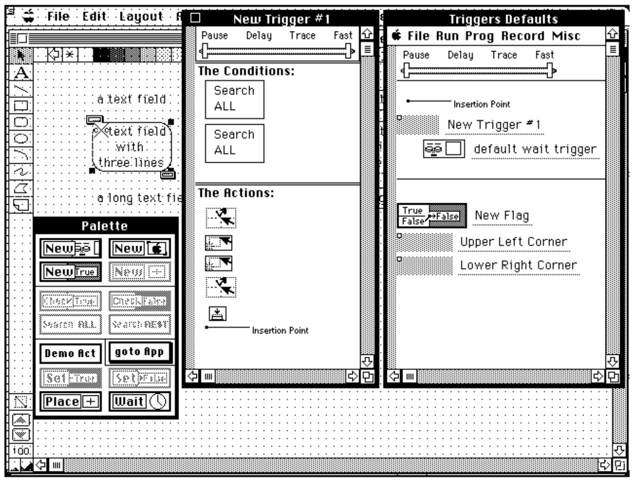

Figure 2: Triggers-III's interface floats above other applications on a light gray grid called the Desktop Blanket.

The *pixel pattern match specification widget* allows all parameters of a pixel pattern match operator to be specified on the Desktop Blanket and thus in the context of any application's GUI. For example, if

```
        PixelPatternMatch(□,theScreen,
                {120,120,500,150},Location1);
```

is an example of a pixel pattern match operator written in pseudo code, it can be difficult to interpret what the various low-level parameters would mean. Figure 3 shows the same information displayed on top of a drawing application. In context, one can see that the pattern is the top of a line and that the search area bounds the tops of lines that represent suspension cables in the drawing of a bridge. Triggers-III supports similar widgets for specifying the parameters of mouse actions in context.

For the third guideline of maximizing programming by demonstration benefits, Triggers-III uses a trigger control structure that allows conditionals to be recorded in a way similar to how keyboard macros are recorded. Each trigger consists of a conditions part and an actions part. The user first records steps for the conditions part followed by steps for the actions part. When run, a trigger acts like an *if* statement. If all the condition steps succeed, then the action steps are performed. The pixel pattern match operator can be included as a condition step that succeeds if a pattern is found. Mouse and keyboard actions are used for action steps.

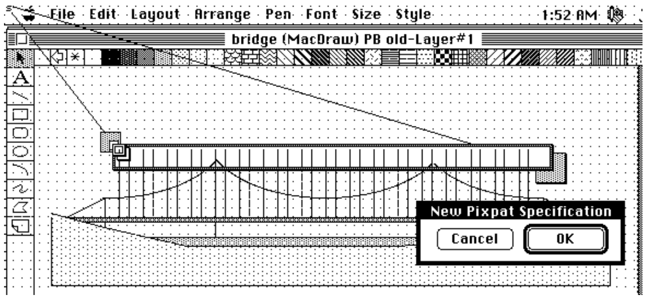
File Edit Layout Arrange Pen Font Size Style
1:52 AM
bridge (MacDraw) PB old-Layer#1
New Pixpat Specification
Cancel
OK

Figure 3: Pixel pattern match shown in context.

Triggers-III has shortcuts for creating triggers and the various program steps. If the user has mastered these, it is possible to create complete programs with loops and conditionals without looking at the programs' static representations, analogously to how keyboard macros can created. Some of these shortcuts will be used in the examples that follow.

CHANGE WIDGET DEFAULT EXAMPLE

Pixel data access is useful for programs that simulate a user doing a task manually. In general, pixel data access supplies functionality that simulates times when the user must look at the computer screen. When used with pixel pattern matching, pixel data access is useful for cases where the user must look at the screen to find the *existence* or *location* of something in the graphical user interface.

For example, consider a Web-based search engine that lets the user choose how many matches to return, such as in Figure 4. It defaults to 25 matches, but the user always changes the widget to 100 matches. A simple program could simulate the user's action of noticing the undesired default and changing the default to 100. Pixel data access can be used to simulate the user looking at the screen to notice the widget at the undesired default.

Triggers-III can be used to show how this program can be created quickly using pixel pattern match, a trigger, and a few shortcuts. The basic form of the program is "if a widget with the label 'Number of matches:' is set to 25, then change its setting to 100." This form of program fits well with the trigger control structure.

To create the program with Triggers-III, the user first displays the web page with the widget set to 25. Then the user raises the Desktop Blanket by pressing the control and options keys simultaneously. Next, pressing the command-I shortcut creates a new trigger. The shortcut of pressing the command key and dragging the mouse across the appearance of the widget creates a new pixel pattern match specification widget as shown in Figure 5. The user clicks on the widget's OK button (not shown) to accept the selected pixel pattern. Then, the user presses command-A to record a search using the new specification into the trigger. The conditions part of the trigger has been completed.

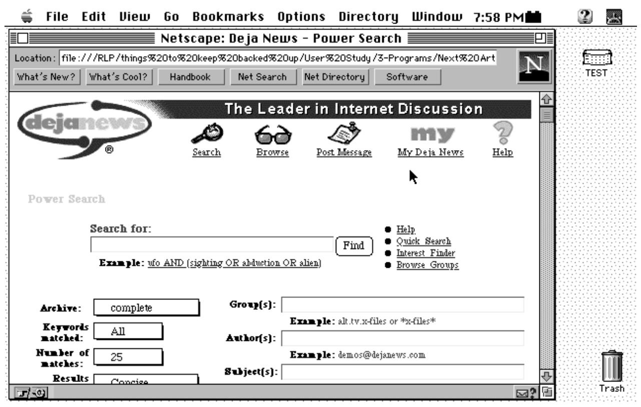
File Edit View Go Bookmarks Options Directory Window 7:58 PM
Netscape: Deja News - Power Search
Location: file:///RLP/things%20to%20keep%20backed%20up/User%20Study/3-Programs/Next%20Art
What's New? What's Cool? Handbook Net Search Net Directory Software
TEST
dejanews
The Leader in Internet Discussion
Search Browse Post Message My Deja News Help
Power Search
Search for:
Find
Example: ufo AND (sighting OR abduction OR alien)
Help
Quick Search
Interest Finder
Browse Groups
Archive: complete
Keywords matched: All
Number of matches: 25
Results Concise
Group(s):
Example: alt.tv.x-files or *x-files*
Author(s):
Example: demos@dejanews.com
Subject(s):
Trash

Figure 4: A Web page to customize.

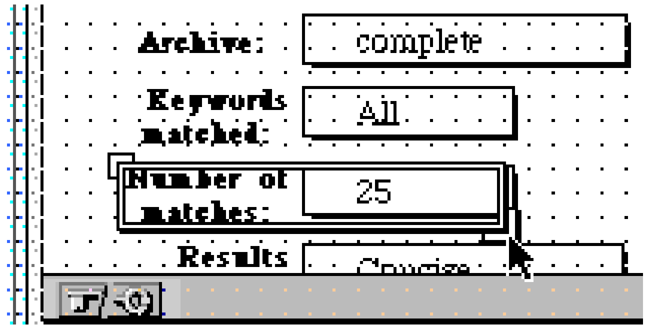

Figure 5: Selecting a pixel pattern on the Desktop Blanket.

To create the actions part of the trigger, the user presses the command-D shortcut, which puts Triggers-III into a macro recording mode. The user can then demonstrate the mouse drag that will change the default from 25 to 100, and then conclude the recording mode by clicking on the OK button on a special dialog. This completes the program, which takes approximately 30 seconds to create with Triggers-III. Pressing command-G starts the program running.

This example shows how pixel data access has the potential to be applied to any application. Triggers-III could be used to easily create a similar program for changing almost any widget from one default to another. The main requirement is that the widget must be rendered with the same pixel pattern in its original default state. It does not matter how the application developer creates the widget or whether the application developer has gone to the effort to tie the widget to any interprocess communication. The widget could be built with standard widgets from the operating system or built specially for an application. The widget could be part of a web page, as in this case, or a custom Java widget running in a Web page. The widget could even be on another machine being used by remote control software.

SHORTEN LINES EXAMPLE

The main advantage of pixel data access is that it provides interprocess communication for applications that do not explicitly support it. Another advantage is that it can sometimes provide a new perspective for automating a task that would not likely be supplied by other interprocess communication techniques. For example, consider the task of shortening the 35 lines representing suspension cables in the drawing in Figure 3 so that the lines just reach the arcs that represent the support cable. A program could automate this task by simulating the user actions for doing this task manually. The user would first find a line needing shortening. Then the user would click the line to select it, and then drag the top handle of the line downwards until the line just reaches the arc that represents the support cable. The user must visually find each line and visually decide how far to shorten it. It is for the visual steps that pixel data access can be useful.

The user's manual actions can be simulated with a device-level program. The basic ideas are that each line needing shortening can be found by searching for the top of the line, and the intersection point can be found by searching down the edge of the line for the first black pixel. The program can be recorded as four steps into one trigger. The first step searches for the top of a line needing shortening. The second step searches for the intersection point. The third step selects the line with a mouse click. The fourth step shortens the line with a mouse drag.

Recording the first step is similar to the pixel pattern match of the previous program. However, in this case the search area must be set. Therefore after the user command-clicks on the Desktop Blanket to select the pattern of the top of the line, the user defines the search area by dragging two new handles that Triggers-III has placed around the new pattern. For this search, the user drags these outwards until the search area encloses every line needing to be shortened, which has just been done in Figure 3. Then the user can click on the pixel pattern match specification widget's OK button and record it into a trigger with command-A. The search area is removed, but the pattern remains on the Desktop Blanket.

Recording the second step is similar to the first step since both a pattern and search area must be specified. However, for the second step the search area is different for each line. In other words, the rectangle is defined relative to the location of the match, using the match location as origin.

With Triggers-III, this is specified by designating a pattern that is displayed on the Desktop Blanket as an origin marker. Holding down the command key and clicking on the top-of-the-line pixel pattern raises a popup menu that can be used to make the pattern's location an origin for any search areas subsequently defined. Now the user can use the same technique to define a pixel pattern match specification that searches for a pattern of one pixel in a search area one-pixel wide that is located down the right edge of the line. Since the pattern and the search area are small, Triggers-III provides a magnifying view (Figure 6) that pops up whenever the control key is pressed, similar to the magic lens technique.[3]

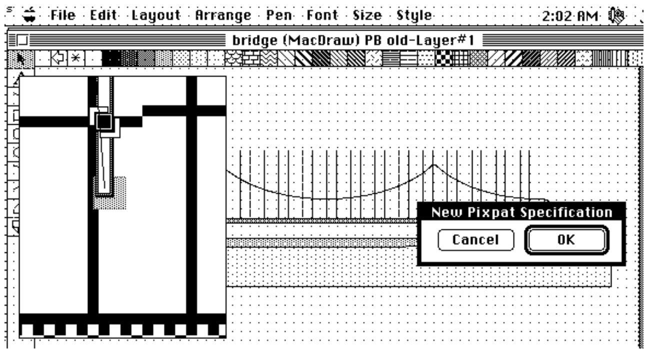
File Edit Layout Arrange Pen Font Size Style
2:02 AM
bridge (MacDraw) PB old-Layer#1
New Pixpat Specification
Cancel
OK

Figure 6: Creating a one-pixel-wide search area after having created the one pixel search pattern by using the magnifying view.

Recording the third step requires a mouse click that is located a certain distance from the top-of-the-line pattern. This is done by first setting the pattern to be the Mouse Down Origin by way of a popup menu. Then the mouse click can be demonstrated by pressing command-D and clicking on the line and then clicking the OK button of a special dialog to conclude the demonstration.

The final step is a mouse drag that starts close to the top of the line and ends close to the intersection point. The top-of-the-line pattern is already the Mouse Down Origin. The intersection-point pattern can be set to be the Mouse Up Origin by way of its pop-up menu. Then the mouse drag can be demonstrated by pressing command-D, shortening the line manually with a mouse drag, and then clicking the OK button of the "Demo Actions Control" dialog.

It is interesting to compare the solution that uses pixel data access with one that uses formal interprocess communication, because the solutions are so different. For example, the formal interprocess communication in Microsoft Office 97 does provide programs with information about lines, arcs, and other objects drawn in Word, Excel, or PowerPoint. For lines, it provides the location of each endpoint. For arcs, it provides the bounding box of the portion of the arc shown, the angle at which the endpoints of the arc are located, and other properties that seem to matter, such as whether the arc has been flipped vertically or horizontally. If this scenario were transferred to a drawing of a bridge in PowerPoint, a program to automate shortening the lines would have to compute the intersection point from these properties. The program would also have to handle the fact that the arc that intersects each line is one of three different arcs that make up the support cables.

One could argue that the solution made possible by pixel data access is easier to create, because these complexities are avoided. However, the formal interprocess communication solution has several advantages. The line lengths can be computed more accurately than with pixel data access, which is limited to within-a-pixel precision. Speed would be greater because the commands to change the line lengths would not be slowed by the limited speed of the mouse actions. For a general solution to be archived and used many times, formal interprocess communication has compelling advantages. For a quick solution that will only be used once, the pixel data access solution could offer superior benefits. For an expert user of Trigger-III, this program takes about 90 seconds to implement. It took us several minutes to manually edit the bridge, and so in this case, building and running the macro saved us time.

PERFORMANCE ISSUES

The use of pixel pattern matching in programs raises performance issues because computer display buffers are multi-megabyte structures; searching them will require significant processing time. While performance has not been the focus of our research, we have reasons to believe that using pattern matching with pixel data access can be efficient enough to be practical.

For many of the useful programs we have created, the area searched is small, such as with the second pixel pattern match in the shorten lines example. For this program, the time bottleneck is the time for the drawing program to respond to mouse actions, not pixel data access. It is likely that the bottleneck will be other factors for many programs, because screen updates too must process significant amounts of display data. For today's computers, interactive speeds can be obtained even if every pixel has to be touched. Switching between two full-screen applications can give an idea of the performance possible on today's computer, since there too, every pixel must be processed. Furthermore, pattern matching is a sublinear algorithm and can be significantly faster than algorithms that must touch every pixel.

Programs such as the first program that poll the screen periodically raise a performance issue because many such programs could be running in the background, potentially making the computer run sluggishly. However, it is still possible for performance to be very fast. In the case above, only one location on the screen is checked for the pattern. When testing this location, the system could dynamically determine which part of the pattern is most likely not to match and check it first. Therefore, the pixel pattern match operator could need only to check just one word of memory from the display buffer.

This discussion assumes that the pattern matching is done in software. If display hardware ever supports pixel pattern matching, the case for pixel data access would be even stronger.

MAPPINGS AND ASSUMPTIONS

The previous programs show how the pixel pattern match operator can map to different roles. In the first it tests for the existence of a widget and its specific state. In the second program, the pixel pattern match operator searches for graphic objects within a collection and computes a relationship between two graphic objects. The ability to map from general purpose functionality to various roles is in many ways an advantage of using a device-level algorithm strategy with pixel data access. Generic functionality allows the programming system to be simple and still be useful. A simple system is more likely to be mastered by users, who are then more likely to use it. Also, generic functionality has a chance to map to unanticipated future use with no need to update the system. For example, the techniques used above in the program for changing the widget default would likely work for new widgets developed in the future.

However, these advantages come with a necessary tradeoff. While the system is simpler and requires less to be learned, it requires more thought and skill to use. The primary difficulty comes from the mapping of generic operators to various roles, which introduces assumptions necessary to make the mappings valid. The user must understand the assumptions when writing and using the program. The user must understand that introducing some assumptions into the program is inevitable. The user's goal is therefore not to eliminate assumptions, but rather to verify that the *assumptions are reasonable for the specific task situation*.

For example, the program to change the widget's default setting has several assumptions. Since the search area was not adjusted after defining the pixel pattern, the pixel pattern match will only look in one location for the pattern. Therefore, the program assumes that the Web browser window is not moved and the widget is displayed at the same location on the page. The pixel pattern matching in Triggers-III is exact, so the Web browser must always display the widget using the exact same pixels. In addition, since the pattern matching runs periodically, it must not take away so much processing time that the computer runs sluggishly.

In our experience, these assumptions are reasonable. The Web browser does in fact display the widget with the same pattern. We typically run the Web browser in full screen mode, so the widget always appears in the same place. Since the pattern is only tested for one location, the pattern matching runs very quickly. When the assumptions are reasonable for the specific intended use, pixel data access can offer the ideal solution.

When the assumptions are not reasonable, a skillful user can often make simple adjustments to a program to make them reasonable. For example, the requirement that the Web browser not move is clearly too restrictive for many users. Changing the program so that it places the search area relative to the front window would solve this problem. (Triggers-III does not support this directly. However it is currently possible to simulate using only pixel pattern matching.) Enlarging the search area would also remove this assumption, but brings in the tradeoff of increased processing time. How to make full screen searches efficient enough for programs such as this is a productive topic for future research.

Another important skill is to recognize when use of pixel data access is not appropriate. Some assumptions that are reasonable in one situation may not be in another. One important factor that can introduce unrealistic assumptions is fractional width fonts that can make text appear with different pixel patterns depending on where it is displayed. Zoomable interfaces [2], animations, and configurable interface styles are examples of other interface situations that may make pixel data access less appropriate.

To put the issue of mappings and assumptions in a larger perspective, all computer programs make assumptions about the domain they will be used within. When programs are used outside a well-defined domain, they may cease to work. For example, if a formal interprocess communication technique were used for the change widget default example, it might assume that the company sponsoring the search engine did not revise their page (which they have done several times since the screen shot above was taken). In this specific case, the ability to quickly redefine the program with pixel data access could be more valuable than robustness that turned out not to be the weak link.

USER STUDY

Use of pixel data access with the device-level algorithm strategy requires that users have certain skills. Although a programming system can provide tools to make implementing the program easier and faster, ultimately users must understand that the program is indeed processing pixels. Users must have enough skill to choose device-level operators that can process the pixels in a way that maps to the higher-level task.

How difficult is it for users to acquire these skills? On one hand, since users are able to use device-level keystrokes, mouse actions, and pixels when doing tasks manually, it is plausible that users will to a large extent already have the necessary skills. On the other hand, users do not normally reduce their actions to algorithms, nor are they restricted to a few device-level operators. When shown the shorten lines task, many people have remarked that they would have never thought of the program that searches down the edge of a line. Thus it is plausible that skills for using the device-level algorithm strategy must largely be learned.

To better understand where the truth lies between these extremes, we conducted a user study in which subjects were given an hour-long tutorial, followed by four tasks to automate. The subjects wrote programs that could automate the tasks on paper, working unaided as if taking an exam. We decided not to have the students implement the programs with Triggers-III because we wished to concentrate on the more fundamental conceptual issues, not the specific interface choices made for the prototype. Device-level functionality including pixel pattern match was presented in a C-language syntax. The students were then free to express their program using the functionality using any syntax that expressed an unambiguous algorithm.

The five male subjects and one female subject were all graduate students in computer science at the University of Maryland. They reported a median of 8.5 years of academic and 1 year of professional programming experience. The median number of programming languages in which they claimed to be fluent was 3 and claimed to have used 3 different programming languages in the previous year. All subjects had used the C programming language in the previous year, so all were comfortable with the C syntax used to explain pixel data access in the tutorial. Five of the six had used a Macintosh in the last year. A scripting language was used by three of the subjects (Unix shell, TCL, or Perl). The other three reported that they did not make use of any scripting or macro languages. In total, the user study took approximately three hours for each subject, for which they received $30 compensation.

Tutorial

The tutorial was given in two parts. The first part presented device-level data types and operators, which were summarized on one sheet of paper. A separate 9-page document illustrated each data type and operator and gave an overview of the few parts of C's syntax that would be used in the tutorial. The document had no text for self guidance, so the experimenter led the subjects through each page and explained each data type and operator. The subjects were told that they did not have to worry about memorizing all the details because the details would be revisited in the soon-to-come example programs. This first part of the tutorial was completed in about 10 minutes.

The second part of the tutorial was structured around four example tasks and the programs to automate them. Each task was presented on paper with a paragraph description of the task situation, the task description, and a screenshot of the application with which the program would need to communicate to automate the task.

The program for automating the task was presented in three passes in order to emphasize the underlying simplicity of the programs. The first pass showed the simplicity of the program in English phrases. The second pass represented the amount of detail the subject would be striving for when working out programs in the exam part of the study. The third pass represented a truly realistic program with details that might be difficult to specify without tools to measure pixel distances.

After presenting each task, the experimenter reviewed general techniques that were summarized on another sheet of paper. Those that applied to the task were emphasized. In this way, the summary sheet returned focus to the general issues after the example tasks provided realistic details. All the materials used in the study are in [20].

Exam

For each task, the subjects received a task description, a full-page screen shot, and several sheets of blank paper. The experimenter set up a Macintosh PowerBook computer to reflect the task situation so that subjects could explore details of the task situation. The subjects read explanations of the task situation and the task, which were written in the same style as the tasks used for the tutorial. At this point, timing was started for the subjects' maximum of 15 minutes for each of the first three tasks and 20 minutes for the final task, which required a longer program.

The first task was the "Change Widget Default" task discussed above. The second task was the "Next Article" task, which was to provide a keyboard shortcut for a Web page link. The third task was the "Shorten Lines" task discussed above. The fourth task was to convert a text in a document of the form B<arbitrary text>" to "**arbitrary text**".

The subjects wrote the answers on paper. For each task the subjects produced two programs and a list of assumptions. The first programs were quick sketches whose purpose was to capture the overall strategy in case the subjects ran out of time. The second programs gave all the detail necessary for the program to be unambiguous, except for pixel distances that would be difficult to determine without special tools. The assumptions were those upon which the programs depended for the mapping between device-level data and operators and the higher-level roles to be valid.

Results

The programs were classified into 4 categories according to how well the subjects demonstrated the skills necessary to automate the tasks. Category 1 programs were error free and showed the subject had the necessary skills to automate the task using pixel data access. Category 2 programs had some errors, but still clearly showed the subjects had the necessary skills. Category 3 programs did not clearly show the subjects had the necessary skills, but showed no major misunderstandings. Category 4 programs showed some misunderstandings. Table 1 summarizes how the 24 programs were categorized. A total of six programs were in category 1, eight programs were in category 2, nine programs were in category 3, and one was in category 4.

Subject	Change Widget Default	Next Article	Shorten Lines	Bold Text
1	2	1	3	2
2	2	1	1	3
3	2	2	3	3
4	3	3	3	4
5	2	1	1	1
6	2	2	3	3

Table 1: The 24 programs created by the six subjects were put in four categories. Only one program was rated as category 4, showing misunderstanding.

Discussion

The most encouraging result of the user study was that the subjects showed little or no confusion about pixel data access, its power or limitations. They considered various alternatives and accepted only those possible with the few data types and operators provided. The subjects also were able to identify the important assumptions necessary for their programs to work. This indicates that with time and patience, a significant number of users could be successful with pixel data access even if support is minimal from the programming environment.

While the subjects showed little confusion, many of their programs did have errors. At least some of the errors can be attributed to the design of the study. There was little or no time for proofreading. The subjects had limited time and some were unfamiliar with the applications. One subject used up 10 of his 15 minutes struggling with MacDraw to learn how to shorten lines manually. He asked if he could ask questions about MacPaint, but the study format was that all subjects would work unaided so the experimenter could not clear up the fact that he was working with MacDraw, not MacPaint. Another subject produced programs that were too sketchy, and based on the fact that he completed every program in less than 10 minutes, he probably took the instruction "not to get too distracted on details" too generously.

One way to explain many of the errors is the fact that the subjects had no programming tools. Few people write out programs on paper and expect them to be perfect. Testing is part of most program development. The programs classified in Category 2 were those that were judged to contain only the sort of errors that could be removed during testing. One mistake made by several subjects was not to include the "25" in the pixel pattern for the "Change Widget Defaults" task. When asked to explain the program the subjects correctly explained how it would change the setting to 100. When asked what the program would do next, they quickly saw that the program would repeatedly click on the popup menu, even though 25 was no longer showing. Then they quickly realized what the correct pattern should be. This is the sort of error that would be obvious upon running the program, and that would have an easy solution. With this in mind, 14 of the 24 programs were essentially correct.

Errors in the other 10 programs suggest that pixel data access and the programs in which it will be used have an inherent complexity that must be addressed. For programs that offer great benefit, extra effort from users may be an acceptable solution. For other programs, better tools may allow users to comfortably deal with the complexity with less effort.

The subjects showed a variety of solutions indicating that they could use pixel data access in creative ways. Programming systems that support an exploratory approach to programming could give users more freedom to apply their creativity by allowing them to quickly try out various solutions. Of the 10 programs that had significant errors, 9 were pursuing solutions that were using pixel data access in a practical way. Better tools could have allowed the subjects to complete the programs and work around the errors quickly.

The subjects' comments during the study indicated that they clearly understood pixel data access and at times enjoyed working out solutions that used it. During the tutorial, subjects were quick to say that they understood each point. During the tutorial tasks, the subjects often anticipated points about the programs. While doing the exam part of the study, the subjects' comments and actions showed they were actively engaged in the programming process. They considered different alternatives and understood the goals well enough to confidently judge their progress. For some, the experience seemed like solving a fun puzzle. One subject said the techniques were becoming "second nature."

SUMMARY AND CONCLUSIONS

Triggers-III shows that useful programs that use pixel data access can be created quickly. The user study contributes to understanding the skills necessary to use the device-level algorithm strategy and verifies that it is possible to learn these skills quickly.

Triggers-III give evidence that pixel data access can be useful in a simple programming system that could be added to any graphical user interface. Though the low-level techniques are useful independently, we expect they will be more useful when combined with other interprocess communication such as OLE Automation and Apple Events, and higher-level pixel data access techniques.

ACKNOWLEDGMENTS
This research was partially funded by Apple Computer, Inc.

REFERENCES

1. Apple Computer, Inc. *Inside Macintosh: Interapplication Communication*. Addison Wesley, Reading, MA, 1993.

2. Bederson, B., Hollan, J., Perlin, K., Meyer, J., Bacon, D., and Furnas, G. Pad++: A zoomable graphical sketchpad for exploring alternate interface physics *Journal of Visual Languages and Computing*, 7, 3-31, 1996.

3. Bier, E., Stone, M., Fishkin, K., Buxton, W., and Baudel, T. A taxonomy of see-through tools. In *Proceedings of ACM CHI'94 Human Factors in Computing Systems (1994)*, pp. 358-364.

4. Bier, E., Stone, M. Snap-dragging. In *Proceedings of ACM SIGGRAPH '86* (1986), pp. 233-240.

5. Brockschmidt, K. *Inside OLE*. Microsoft Press, Redmond, WA, 1995.

6. Cypher, A. Eager: programming repetitive tasks by example. In *Proceedings of ACM CHI'91 Human Factors in Computing Systems (1991)*, pp. 33-39.

7. Cypher, A., ed. *Watch What I Do: Programming by Demonstration*, MIT Press, Cambridge, MA, 1993.

8. Kosbie, D, Myers, B. A system-wide macro facility based on aggregate events: a proposal. In *Watch What I Do: Programming by Demonstration.* MIT Press, Cambridge, MA. 1993.

9. Kurlander, D., Bier, E. Graphical search and replace. In *Proceedings of ACM SIGGRAPH '88* (1988), pages 113-120.

10. Kurlander, D., Feiner, S. A history-based macro by example system. In *Proceedings of the ACM Symposium on User Interface Software and Technology* (1992), pages 99-106.

11. Libes, D. *Exploring Expect: A Tcl-Based Toolkit for Automating Interactive Programs*. O'Reilly & Associates, Cambridge, MA, 1994.

12. Lieberman, H. Integrating user interface agents with conventional applications. In *Proceedings of ACM Intelligent User Interfaces '98.* (1998), pp. 39-46.

13. Masui, T., Nakayama, K. Repeat and predict-two keys to efficient text editing. In *Proceedings of ACM CHI'94 Human Factors in Computing Systems* (1994), pp. 118-123.

14. Maulsby, D., Witten, I. Inducing programs in a direct-manipulation environment. In *Proceedings of ACM CHI'89 Human Factors in Computing Systems* (1989), pp. 57-62.

15. Modugno, F., Corbett, A., Myers, B. Graphical representation of programs in a demonstrational visual shell-an empirical evaluation. *ACM Transactions on Computer-Human Interaction*, 4(3):276-308, September 1997.

16. Myers, B. Creating user interfaces using programming-by-example, visual programming, and constraints. *ACM Transactions on Programming Languages and Systems*, 12(2):143-177, April 1990.

17. Potter, R. Guiding automation with pixels: a technique for programming in the user interface. Technical Report *HCIL 1992 Video Reports*, Department of Computer Science, University of Maryland, College Park, MD, 1992. Also appears in the *INTERCHI '93 Video Program*.

18. Potter, R. Triggers: guiding automation with pixels to achieve data access. In *Watch What I Do: Programming by Demonstration.* MIT Press, Cambridge, MA. 1993.

19. Potter, R. Graphical macros: a technique for customizing any application using pixel pattern matching. Technical *Report HCIL 1994 Video Reports*, Department of Computer Science, University of Maryland, College Park, MD, 1994.

20. Potter, R. *Pixel Data Access: Interprocess Communication in the User Interface for End-User Programming and Graphical Macros*. Ph.D. thesis. University of Maryland Department of Computer Science, May 1999.

21. Zettlemoyer, L and St. Amant, R., A visual medium for programmatic control of interactive applications. In *Proceedings of ACM CHI'99 Human Factors in Computing Systems (1999)*, pp. tbd.

22. Yamamoto, K. A programming method of using GUI as API. *Transactions of Information Processing Society of Japan: Programming*, pp. 26-33, December 1998.