Support for Computer-Based Observation Tools

Ben Dugan © 1995

1 Introduction

User observation plays a critical role in the design of computer artifacts. Traditionally, researchers relied upon a variety of media such as audiotape and videotape to record their observations. This paper presents a project to unify traditional recording media to support computer based observation of computer users. The project primarily focuses on the technical issues involved in providing the resources to represent and manipulate large quantities of data obtained during long recording sessions. I have developed a system which demonstrates the feasibility of digitally recording large volumes of audio, video, and screen data. It is hoped that this work will provide a solid foundation for building powerful new tools to support the recording and analysis of user observations.

The paper is organized as follows. Section 2 gives background and motivation for performing user observation. Section 3 evaluates current recording tools. Section 4 describes design goals and implementation status. Section 5 gives the technical details of the system, including a system overview, compression methods, and performance numbers. Section 6 concludes with a comparison of my system to traditional recording methods, and suggestions for future work in this area.

2 Background

Historically, systems design begins by constructing a system description, or model, of an activity or work environment. This model forms the basis for the designer’s understanding of the user’s practice. In software design, this model often takes the form of requirements and specifications documents. Design is seen to be successful if the requirements are fulfilled. [Simon] is a general example of this approach.

In the last decade, the traditional approach has come under attack from various directions, such as [WF], [Ehn], [GK], [SN], and [AW]. Although the criticisms differ markedly, many share an emphasis on early and extensive involvement with the intended users of the new system. With this new, user-centric focus has come a desire to gain a deep understanding of the user’s practices. As systems descriptions have not been shown to be capable of adequately capturing the rich and complex interactions which form the basis for human practice, researchers have looked to other techniques for gaining insight into this problem. While today it is generally accepted that users should be viewed as valuable resources for guidance and information when developing and validating user interfaces, the approaches to user involvement and observation vary widely. On the traditional end of the spectrum, users are viewed primarily as a objects of scientific study, to be understood via controlled experimental techniques in laboratory settings [CMN]. Users are not actively involved in the design process. At the other end of the spectrum, researchers such as [Blom] advocate ethnography as an approach to learning about the practices of a group of skilled workers. Ethnography advocates direct observation, participation, and interview as means to build an understanding of a cultural group and its practices. Ethnography aims to paint a holistic, rather than reductionist picture of the area of study. A central tenet of ethnography is that cultural practices cannot be understood separate from the context in which they occur. Therefore, ethnographic approaches insist that the observation and participation take place within the setting of the activity (the workplace, the control room) rather than in laboratory settings.

While the debate continues over the applicability and appropriateness of the various means of gaining knowledge about people and their practices, common to all of the techniques is their reliance on a variety of media for recording the observations of the researcher. The purpose of my project is building tools to support common recording media in a digital manner, thereby facilitating computer support for observing, and analyzing the practices of computer users.

Historically, researchers relied upon three different media to record their observations: audiotape, videotape, and field notes. With the proliferation of computer systems, researchers employed one further technique -- input logs of keystrokes and other input events captured by the computer. Each of these four media has distinct advantages and disadvantages, which will be discussed below. This discussion will motivate the introduction of a computer-based recording tool which integrates the best characteristics of some of the above media, to support the observation of computer users.

3 Current recording media

3.1 Audiotape

Audiotape is generally inexpensive, relatively unobtrusive, and very portable. It has a number of serious drawbacks, though. First, randomly accessing portions of the recording is difficult. Second, if recording is done during task observations, the audio record alone may not be very revealing, because it may be difficult or impossible to connect comments (if any) made during the task to the actual activity. Furthermore, as many tasks do not involve a great deal of verbalization, participants must continually be urged to “say aloud” what they are doing, which can be distracting and annoying to the participant. Generally speaking, the audio record provides the researcher with a limited bandwidth of information about the user and their activities. Finally, by its very nature, an audio record can only capture verbalized (explicit) knowledge, which is in many cases not the same as the tacit knowledge the users may be relying on in their everyday activity.

3.2 Videotape

Videotape is necessarily a high-bandwidth information source about the user and their activities. It provides the researcher with a reasonably objective record of the user, their activity, and the surrounding environment, and therefore supports a somewhat holistic view of the user’s practices. It also includes the benefits of audio (provided that it has an audio channel, which is usually the case), without many of its disadvantages. In other words, verbal protocol is grounded in the physical activity recorded on the tape. Furthermore, video gives the researcher access to the user’s non-verbal communication.

On the other hand, video recording can be problematic. Video cameras can be obstrusive and distracting. Their recordings exhibit the same random access problems of audio tapes. Often, the setting can make recording difficult (too dark, too cramped, etc.). Another common problem with video is that a single camera often cannot adequately capture all important aspects of the user and their practice. For instance, when observing a computer user, the researcher might be forced to choose between keeping the camera pointed at the user or at the screen, which inevitably decontextualizes the information gathered. The solution is either to use multiple cameras, which can compound the intrusive effect of the cameras, or to compromise with a half-and-half view of the user and their work, leaving the researcher with a less than perfect view of both aspects of the setting. Multiple camera sources incur greater cost and can be difficult to synchronize for playback. Finally, once the video data has been gathered, analyzing it and generating content logs can be a tedious, time consuming process.

As a side note, it is interesting to observe that many of the problems with video have been overcome by the development of usability labs. In these labs, office environments are “simulated” and physical constraints to filming can be easily controlled (lighting, space requirements, noise levels). Further, researchers can employ multiple cameras in these environments without the usual burden of setting up additional hardware at a remote site. Some labs (e.g. Microsoft) are high-tech, big money investments. While it is nice to see companies showing an increasing interest in usability, it is my position that these labs are of limited utility for researchers carrying out true ethnographic studies -- ethnography cannot take place in a laboratory setting. Furthermore, high tech usability labs appear to have contributed to the development of an “aura” surrounding usability. Many people hold the opinion that usability cannot be achieved or tested without a special laboratory designed for the purpose (observation due to Borning).

3.3 Field Notes

Taking notes can be a very inexpensive and unobstrusive means of gathering data. Also, notes provide a good general overview of the observed situations and environments. Generally speaking, notes represent a valuable record of the researcher’s on-the-spot observations, which may differ markedly from post-hoc reconstructions of the events.

The major problem with field notes is that they are very low bandwidth and therefore make it difficult to record large quantities of information. Furthermore, because the notetaker cannot record a great deal of raw information, the data recorded is often very condensed or filtered. This condensation and filtering of information is clearly a double-edged sword in that it provides a powerful, on-the-spot analysis, but can also cause the researcher to overlook, or filter out important events or actions. Finally, reconstructing events based on notes alone can be difficult; researchers commonly interpret their notes days, weeks, or even months after the fact, by which time much of the original context is lost or forgotten.

3.4 Input logs

The proliferation of electronic devices has enabled the logging of input events to technological artifacts. Commonly, both the device which does the recording and the device being monitored are computers, but this need not be the case. One could imagine rigging telephones, electric typewriters, photocopy machines, etc. to enable monitoring and recording of their input events. The advantages of this kind of recording are that it can be unobtrusive and it provides fine-grained data about the user's actions. Furthermore, if the stream of data is in a convenient form (i.e. a computer file) it can be easily manipulated, indexed, searched, and replayed.

The greatest drawback to input logs is that they encourage a keystroke-level analysis of the user and their task. This reductionist tendency can be problematic, for it encourages the researcher to view the user as a mere stream of input events to the application, rather than a skilled, practicing human engaged in a meaningful activity. Input logs decontextualize the recorded activities and are anti-holistic. Furthermore, analyses of input logs can be difficult precisely because there is no context to the keystrokes -- it is difficult to determine the "big picture" because the researcher is left only with a stream of keystrokes. Technically speaking, the code to generate input logs may not be application independent (although it certainly could be). Depending on the level of operating system support, it may need to be compiled into the application(s) the user is using. This can be problematic when researchers are evaluating multiple applications simultaneously or if they do not have access to an application's source code. A final problem with input logs is that to effect a realistic playback, the user's initial environment must be carefully reconstructed. This can be difficult if the user has undertaken a series of destructive actions (e.g. removing or restructuring parts of their file system), which cannot be recovered.

4 Computer Support for the Observation of Computer Users

Clearly, we want an application to support the observation of computer-users that will combine the best qualities of the above techniques. I built a prototype system which supports several of the above recording media in a unified manner. These media are unified in a single, digital data stream. Currently, the stream supports three "channels": one for audio and two for images. The stream is created by merging data from various sources, such as audio input, video frame grabbers, and screen captures. The audio channel allows the user's conversation to be recorded. The first image channel records data from a display mounted video camera (hereafter known as the video channel), providing a frontal view of the user. The other image channel contains successive screen captures (hereafter known as the screen channel), providing a "movie" of the user's activities on the screen. I have built a variety of tools for capturing, displaying, and manipulating the data stream.

4.1 Design Goals and Philosophy

In developing this system, I tried to achieve several design goals while following a number of design principles. They are presented here, in no real order of importance. It should be remembered that these goals are not necessarily absolute ideals, but rather reasonable targets given current hardware.

1. A powerful technical-economic influence on the design was that disk space is generally considered a "better buy" than specialized hardware. At street prices of well under $.50/MB, disk drives have a high utility/price ratio when compared to specialized hardware, such as MPEG encoding/decoding chips. This concern challenged me to focus on software-based solutions to problems such as data compression.

2. Low cost. I wanted to move away from the mythology of the usability labs and encourage anyone to conduct their own studies, with a minimum of economic overhead. The system should perform reasonably well using today's high-end, stock hardware equipped with video and audio support.
3. The quality of the video and audio channels is more important than the quality of the screen channel. Psychological studies have shown that 100ms (10 frames per second) is the minimum acceptable interframe delay to provide the illusion of smooth motion [CMN]. Television and movies usually use very high frame rates, such as 24 or 30 frames per second (hereafter, fps). While such high frame rates are appealing, they result in unreasonably high data rates. Even a small framed (160x120 pixel), silent, uncompressed, 256 color movie at standard (24 fps) frame rates results in a data rate of over 25 MB/minute. While sophisticated techniques such as MPEG can support high frame rates at reasonable bit rates, it is currently impossible to perform software MPEG encoding on a uniprocessor in real time. To provide quality video and audio at more reasonable data rates, the system would clearly require some form of software compression.
4. One frame per second is a reasonable frame rate for the screen channel. This is a controversial position, subject to empirical validation, but I believe that one frame per second gives a reasonable, "impressionistic" portrayal of the user's screen activity for applications such as reading news, text editing, and browsing the Web. This said, there are admittedly many "sub-second" actions that would still be missed by this frame rate. Furthermore, this rate would not be sufficient for highly interactive applications such as games. It is technically infeasible to support frame rates over 3 fps for large, 1000x1000, 8-bit pixel screen sizes without compression, because the data rate exceeds the bandwidth of most file systems.
5. Given a reasonable (200 MB) amount of disk space, the tool should be able to capture long (around an hour) sessions. This goal gave me a target data rate of around 3 MB/minute.
6. The tool should be able to provide high quality video and audio with minimal loss, if desired. In other words, if the user chooses to *only* record video and audio, the quality of the resulting stream should be comparable to that achieved by other formats such as MPEG, the SGI MV format, or Apple's QuickTime. This requires frame rates in excess of 20 fps, frame sizes of 320x240, and minimal loss.
7. The stream should be editable, and support full VCR style playback (step forward, fast forward, and so on).

The reader will note that there is little emphasis on usability and user interface issues in the above list. This is not an omission. The project did not set out to create a fully capable set of observation tools, but rather develop strong technical foundation upon which such tools could be built.

4.2 Current Implementation Status

The stream and its supporting applications run on an SGI Indy, equipped with a small video camera, audio support, video frame grabber and less than 200 MB free disk space. The processor is a 134 MHz MIPS 4600. The system performs quite well: I integrated all three media into the stream and built a variety of tools (totalling around 6000 lines of C code). Currently, the recording tool provides a combined 10 fps 320x240 8-bit color video channel, a 16-bit, 8 kHz audio channel, and a 1 fps 1280x960 screen channel, at a data rate of 10 +/- 5 MB/minute. The data rate is achieved through the application of up to four compression techniques to the data stream. This performance meets or comes close to meeting many of the basic design goals laid out above. In fact, the tool can capture at rates higher than this, but the data rates quickly become excessive. More performance details are given below.

In terms of striking a balance between quality, frame rates, disk space usage, the system does reasonably well. The major problem in this area is that the burden of striking this balance is forced upon the user. The user must currently have some understanding of how the various system parameters (number of colors used, frame size, frame rate, lossiness, etc.) will affect the overall data rate. A possible solution to this problem will be discussed further below. Currently, the greatest bottleneck to providing long term, high quality, high frame rate image channels is the amount of free disk space. The solution is either larger disks or better compression schemes. Below, I will show that there is more room for on-line compression, although I have yet to implement further methods.

5 Technical Details

5.1 System Overview

The stream consists of interleaved blocks of data from three possible sources: audio, video, and screen capture. The blocks of data in the stream are temporally interleaved, trivializing the problem of synchronization during playback. I have built a variety of tools around this stream. The two most important applications are responsible for capturing and displaying the stream.

A non-preemptive threads package which I built forms the foundation for the capture tool. There are three primary threads, each of which handles capturing, processing, and outputting the audio, video, and screen data, respectively. The audio thread is the most simple: it wakes up at a regular interval, reads audio samples by making an SGI Audio Library call, and writes the data. Currently, 8000, 16-bit samples are taken per second. The video and screen threads are identical at an abstract level: they grab an image frame, decide which parts of the frame need to be written and write these portions, possibly compressing them on their way out. The screen thread obtains its images through an Xlib call. The screen images are 8-bit colormapped images, whose pixel values are indices into a table of 256 24-bit colors which can be painted on the display. The video thread obtains its images via an SGI Video Library call. The video card provides 8-bit RGB images. For each pixel, three bits, three bits and two bits are devoted to the colors red, green, and blue, respectively. The decision to write data is made by a technique called *delta block compression*, which is described below. The system-dependent calls made by the various threads to get data are abstracted through a standard interface, which should make it possible to port the system to other platforms, given the system dependent modules.

The display tool is conceptually very simple. In the stream, each block of data is preceded by a type flag which informs the displayer of the type of data (video, audio, or screen) it will be consuming. With this knowledge the displayer can read, interpret, and direct the data from the file to its destination (an audio buffer or screen image).

Additionally, I have developed a variety of other tools to demonstrate the applicability of the stream and its supporting components. I have implemented a simple, multicasting video conferencing tool. Because I was successful in achieving moderate data rates, the conferencing tool gets reasonable results over a local area, 10 Mbit/second ethernet. Additionally, I have built an MPEG converter, which will extract the video blocks and assemble them into an MPEG stream, using the publicly available Berkeley MPEG Encoder. Finally, I have implemented graphical interfaces to the capture and display tools, using Tcl/Tk. While these tools leave a lot to be desired in terms of usability, they at least make an effort to present the bewildering variety of configuration options in a principled manner (as compared to a command line tool). Also, the Tcl/Tk interface to the display tool nearly gives the user full functioned VCR type controls over playing back the movie stream, which was an important design goal. While this project did not set out to be a user interface project, there are a many problems with the current interfaces, as well as interesting possibilities presented by this somewhat new medium. Both of these issues will be discussed in more detail in the conclusion. The reader will find screen snapshots of the display and capture tools in the appendix.

5.2 Compression methods

The overriding concern of this project was to make reasonably long recordings of large quantities of data feasible on today's machines, which meant that substantial compression would be necessary. The challenge was to find software compression schemes which were computationally cheap but provided reasonable results. Individually, many of the schemes employed do not provide impressive compression ratios, but linking several of them together has yielded substantial results. The current system employs four compression techniques to obtain a workable data rate: delta block compression, color compression, standard data compression, and size compression, each of which are discussed in more detail below.

5.2.1 Delta Block Compression

Delta block compression (or just block compression) is a simple, lossy compression scheme, which exploits the fact that any two successive video frames often differ by very little. MPEG takes advantage of this fact in a sophisticated manner using motion vectors, in which it can determine if (and in what direction) certain portions of a given video frame have shifted with respect to nearby frames. Making this determination turns out to be computationally expensive, but can result in very impressive compression results. I employ a simplified scheme, (which is similar to one employed by the MBone NetVideo tool). Each frame is broken into blocks (usually in an 8x8 grid, meaning that a 640x480 screen results in 64, 80x60 pixel blocks). I maintain a master frame, which serves as a source of comparison to incoming frames. The master frame is initialized with the first frame. When a new frame arrives, each of its blocks are compared to the corresponding blocks in the master frame. If a block is sufficiently different (if more than a threshold number of pixels are different) from its corresponding block in the master frame, it is written to the stream *and* copied into the master frame. A threshhold is necessary because of the randomness inherent in the video data. For "clean" data such as screen data, the threshold is set to zero, which effects lossless compression.

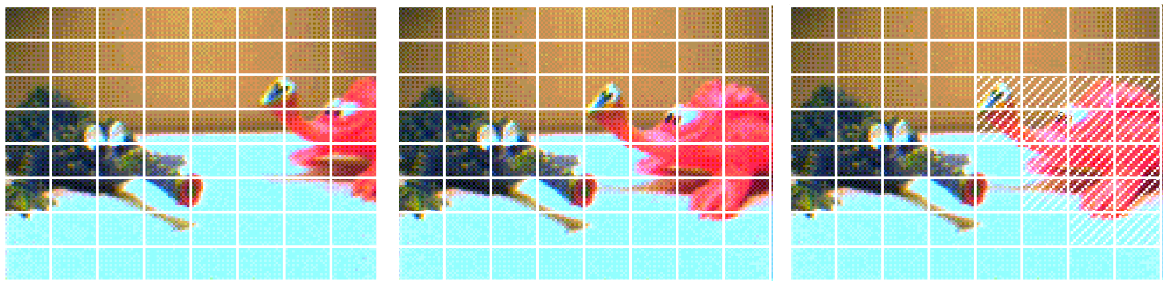


FIGURE 1: Delta Block Compression

Figure 1 shows a typical situation. The only difference between the master frame (1) and the incoming frame (2) is the dinosaur entering on the right of the frame. Therefore, only the blocks which change between these two frames need be updated. The stippled blocks in image (3) are the only blocks of the incoming frame that are written to the stream and copied to the master image.

The strengths of this technique are that it is simple to implement and cheap to compute (no search needs to be performed, as in MPEG). While it does not have the generality of the block motion vector technique, this method can still give reasonable results. The reason is that unlike regular movies, the point of view (the camera) rarely moves during a recording session. The camera is more or less stationary, and while the person using the machine may move about and change facial expressions, the background rarely changes, if at all. These qualities are even more true of screen capture. While users may open, close, and move windows, the basic arrangement of windows, their contents and the screen background is usually quite static. With screen capture, compression ratios of as high as 10:1 have been observed, while 8:1 is typical. With video capture, ratios of 4:1 have been observed but ratios between 1.5:1and 2:1 are typical.

5.2.2 Color compression

Another simple compression scheme is to reduce the number of colors used in an image. The 256 incoming colors can be mapped onto some smaller number of colors. For instance, 16 shades of gray provide reasonable image quality, at just 4 bits per pixel. The stream supports two color compression schemes for the video channel: 16 and 4 shades of gray, which give us 2:1 and 4:1 compression respectively. Sixteen shades of gray is quite adequate in many cases, while just four shades of gray leaves a lot to be desired, in terms of quality. Color compression has yet to be implemented for the screen channel. Although I did not pursue this option, it would be possible to choose the 16 most common colors in the channel and only use those. This scheme would especially make sense for the screen frames, considering that many applications do not use very many of the available colors.

5.2.3 Size compression

At times, it is acceptable to capture small frames and expand them on playback. If the zoom factor is not higher than 2, the image usually looks quite presentable. This can result in very cheap compression (the cost is paid during zooming on playback rather than during capture). The compression ratio is (zoomfactor^2):1 (e.g. 4:1, when zoomfactor=2). For zoom factors greater than 2, smoothing needs to be performed to make the image presentable. Because smoothing is computationally expensive, it has yet to be implemented.

5.2.4 Standard data compression

Finally, I take take advantage of conventional compression technologies (like those employed by the UNIX tools `compress`). I have observed compression ratios ranging from 2:1 to 10:1 on sample files, with 4:1 being typical. Standard compression is employed as a final step by piping the output of the capture program through `compress`. `Compress` uses Lempel-Ziv encoding [Welch]. Using a separate, external compression process does incur some pipeline overhead and takes cycles away from the capture program, reducing its maximum possible framerates.

5.3 Compression results

The video and screen data is always compressed using delta block compression. Currently, color compression is only applied to the RGB data of the video signal. Similarly, size compression only makes sense for the video data. Audio data is not subjected to any compression besides possibly the final UNIX compression along with the rest of the capture data data stream.

An important question is, what is the overall compression ratio? This is difficult to determine because it depends partially on the dynamics of the video and screen data. Given a static scene, we can achieve very high overall compression ratios. My experiments have shown typical, overall ratios ranging from 10:1 to 35:1, with the average at 15:1. An uncompressed stream consisting of 10 fps, 320x240 video, audio, and one fps 1280x640 screen produces a data rate of over 2 MB/s which pushes on the maximum bandwidth of typical file systems. 15:1 compression reduces this to a manageable 130 KB/s, or around 8 MB/minute. If the user is willing to sacrifice video screen size or reduce the screen frame rate to .5 frames per second, even greater savings can be had. It should be evident that while I did not achieved the goal of an hour of material in 200 MB of space, this set of techniques brings us at least half of the way to meeting that goal.

5.4 Performance

In this section I detail a variety of performance numbers for the system, to give the reader an idea of maximum frame rates, the effectiveness (in many cases) of the compression schemes at increasing frame rates, and the nature of the barriers to higher performance.

5.4.1 Time for primitive operations:

Source	Grab Time (ms)
Full Screen (any size)	30
Quarter Video (160x120)	37
Half Video (320x240)	33
Full Video (640x480)	32

TABLE 1. Time for Primitive Operations

The screen grab numbers reflect the use of the X window system shared memory extension. Without the extension, 1280x960 screen grabs take over 300 ms. Shared memory screen grabbing is independent of frame size because no copying takes place. I hypothesize that the smaller video frames take longer to grab because scaling is taking place.

5.4.2 Maximum Frame Rates

In evaluating the performance and effectiveness of the tool for recording just video and audio, I compared its maximum framerates to maximum framerates achieved by the standard SGI capture tool for recording MV files.

Video Size	1. SGI movie capture	2. No Compr.	3. DB only	4. DB and UC	5. DB, CC, and UC
Quarter (160x120)	30	28	28	28	28
Half (320x240)	24	29	29	14	22
Full (640x480)	5.1	7	15	4	7

TABLE 2. Maximum frame rates for video and audio (fps)

Column 1 is included as a baseline for comparison; it contains results from the SGI capture tool recording uncompressed video and audio. Columns 2 through 5 show maximum frame rates for my capture tool with various compression options. Column 2 uses no compression, column 3 uses just delta block compression (DB), column 4 uses DB followed by UNIX `compress` (UC), and column 5 uses DB, color compression (CC), and finally UC.

Comparing columns 2 and 3 shows the performance impact of delta block compression. Delta block compression does not affect the frame rates for quarter and half frame captures. It increases frame rates for full frame captures because it reduces file system output substantially. Columns 3 and 4 demonstrate the impact of UNIX compression. Even though the frame rates drop, they are still adequate for quarter- and half-sized frames. The addition of UNIX compression causes the CPU, rather than the disk to become the bottleneck. Column 5 versus column 4 shows the positive impact of color compression on frame rates. Frame rates increase because color compression reduces the amount of data piped to `compress`. Perhaps the most telling comparison is between columns 1 and 3; it shows superior frame rates achieved by my capture tool at minor cost in image quality but great benefit in disk space savings. In this case, delta block compression results in a 2:1 compression ratio, while the SGI tool records its 8-bit video data in a 24-bit format. In effect, my files are a factor of 6 smaller than the SGI MV files.

The maximum frame rate from capturing, processing, and recording just screen data is between 6 and 8 frames per second, depending on how dynamic the screen activity is. When the screen data is piped through `compress`, the frame rate drops to around 3 fps. Although it is difficult to measure the maximum frame rate of video, audio, and screen capture combined, because there are so many variables, the capture program *is* able to record a 20 fps, 320x240 screen channel, standard audio, and a 3 fps 1280x960 screen channel, using delta block and color compression. This performance is better than double what I hoped to achieve. If `compress` is employed, it is possible to record a 15 fps, 160x120 screen channed, standard audio, and a 1.5 fps 1280x960 screen channel.

5.4.3 Bottlenecks

The delta block compression routines can process incoming frame data (i.e. decide whether or not a given block needs to be written) at around 10 MB/s. Furthermore, the maximum bandwidth of the file system is around 2 MB/s. Given the frame grab times above, data can arrive at a rate of 10 MB/s for full video, 2.5 MB/s for half video, .5 MB/s for quarter video, and 41 MB/s for screen captures.

There turns out to be no single bottleneck that I can point to, because the data flow is so dependent upon frame size, frame rate, and the dynamics of the scene. For large video frames at high frame rates, the disk is the bottleneck, because the delta block compression ratios for video are rarely better than 3:1, meaning that a lot of data is still written to disk. For small frames at high frame rates, the bottleneck is the frame grab time. For screen capture, where the scene is usually pretty static, the CPU becomes the bottleneck, because we cannot examine incoming screen image data at rates faster than 10 MB/s (this is about 9 screen fps). If the screen activity is very dynamic and a lot of data needs to be written, then the disk becomes the bottleneck. While these answers may not be very specific, they do paint a general picture of where the bottlenecks are.

5.4.4 Data Rates for Typical Scenarios

From the above results, it is apparent that frame rates are not a major issue; it is clearly possible to achieve high frame rates, if required. But what about data rates? Given a few hundred MB of disk space, how long of a session can we record at our objective rates of 10 video fps and 1 screen fps with audio? The following table demonstrates the efficiency of the various compression schemes employed. These numbers are *not* empirical, but the compression ratios used in the calculations *are* based upon actual observations from many test runs. The ratios used to calculate the rates for column 2 are 2:1 delta block video frame compression (VC) , 6:1 delta block screen frame compression (SC), and 4:1 UNIX compression (UC).

Scenario	1. “Worst case” no compression	2. VC = 2:1; SC = 6:1; UC = 4:1	3. Ratio between 1 and 2
160x120 video at 10 fps 1280x960 screen at 1 fps 16-bit, 8kHz audio	1.4 (84)	.08 (4.8)	18:1
320x240 video at 10 fps 1280x960 screen at 1 fps 16-bit, 8kHz audio	2.0 (120)	.15 (9.0)	13:1

TABLE 3. Data rates MB/s (MB/minute)

Obviously, the numbers can be improved at the expense of colors or sharpness by the use of color or size compression.The upshot of these results is that it is entirely possible to record half hour sessions at reasonable quality given 200 MB of free disk space. Longer sessions are possible at the expense of image quality.

6 Discussion and Conclusions

6.1 Comparison to Traditional Recording Media

How does the unification of audio and multiple “video” sources into a single, digital data stream compare to the traditional media used in observational studies? The fact that multiple sources of image data have been unified and synchronized is clearly an advantage over the traditional approach of using multiple video cameras with synchronization hardware and/or mixing boards. Less hardware is involved, and there is only one “tape” to deal with. Having multiple image sources also solves the problem of where to point the video camera, when only one camera is available. The researcher must no longer settle for the imperfect compromise of a camera capturing half user and half screen. Further, the camera is very small (smaller than a soda can) and has no moving parts, making it less obtrusive than traditional video cameras. Also, the act of grabbing screen images is independent of the user’s application, because capture takes place through the window system (in this case, X). No hooks for capturing keyboard events need to be compiled into any application. Furthermore, this method of recording emphasizes a holistic view of the user and their environment. Rather than merely viewing the user as a source of input events, this method should give the researcher a clear, environmental perspective on the user’s activities. I believe that this is the “right” - both from a political and a technical perspective - level for learning about the user’s practices. This is not to say that there is no benefit to be gained from a keystroke level analysis. The addition of a keyboard/mouse event channel to the stream will be discussed below. Finally, the user and their activities are recorded into a digital format, giving the researcher distinct advantages during the analysis phase. Building tools for aiding in the analysis of this data is an interesting research direction and will be discussed below.

On the down side, the technical success of this tool depends largely on the number of free cycles and disk space on the user’s machine. Even with compression, the recordings can still grow to be prohibitively large. Clearly, traditional techniques will beat this tool in terms of sheer information recorded. However, the economic situation is currently working in favor of this format because as disk space becomes cheaper, this tool will be able to perform longer and more detailed recordings. Furthermore, as CPUs become more powerful, more real time compression will be possible, counteracting the need for larger disks. In terms of quality, video cameras, with their large frame size and fast frame rates still maintain the edge over this system, although this too should change as better cameras, more disk space, and more CPU cycles become available.

6.2 Operating System Support

Building the capture program upon a lightweight threads package improved both the quality of the stream and decreased the complexity of the code. Quality was improved because audio and video threads could get their frames at regular intervals, giving smooth, regularly spaced playback, with few dropouts (important for audio). The per-frame computation for the screen thread, which is typically an order of magnitude greater than that of the video thread, could also be distributed over the periods of inactivity on the part of the audio and video threads. The code became more readable because the complexity of scheduling these various activities was hidden in the threads package. This said, it was a great surprise to me that I could not find a widely available, standard threads package for the IRIX operating system (such as P-threads for OSF or the LWP library for SunOS). In the end, I had to implement my own package.

Another area for improvement would be decreasing the overhead for video frame grabbing. Getting a video frame takes around 30 ms, which means that in some cases over 90% of the time per frame is spent getting the video frame. If this number could be improved, much time would be made available for doing more real time compression.

6.3 Future Challenges

6.3.1 Model Mismatch

Currently, the biggest problem the recording tool presents the user is due to a mismatch in models. The user approaches the tool with a high level goal like, “I want to make an hour long recording of video, audio, and screen.” They do not know and probably do not want to know about how the recording tool’s data rate is affected by the multitude of options such as frame rate, image loss, color scheme, and frame size. They should not be forced to find out how much free disk space they have available, and then calculate which parameters will result in sufficiently low data rates to achieve the length of recording required.

Ideally, the user would specify a set of constraints, and the system would adjust its parameters to satisfy these constraints to the best of its ability. For instance, a user might specify that they want to make a screen, video, and audio of 45 minutes in length, with a minimum screen frame rate of 2 fps. Given these constraints (and others, based on system resources such as available disk space, processor speed, and disk throughput) the system would adjusts parameters such as lossiness, frame size, video frame rate, and so on, to achieve a data rate which could meet the user’s target recording length. Further, the user should be able to prioritize their constraints. For instance, the user might specify that in general, video image quality is more important than video frame size. In this case, the system would maintain video quality (by keeping the loss level low) at the expense of frame size (perhaps by choosing a smaller frame size to reduce the data rate). While a high-tech solution like this would be an interesting challenge, a more immediate solution would be to discover a set of two or three reasonable default settings based on user requirements and use.

6.3.2 User Interface Challenges

Earlier, I mentioned that the stream should be editable and support full VCR style playback. This turned out to be a difficult goal to support and the current tools do not do so adequately. First, because many of the resulting files are compressed, it is not possible to conveniently “seek” (i.e. move backwards) in those files, making it difficult to provide functions like reverse and “step-reverse” playback. A more troubling difficulty arises due to the delta block compression scheme. Because this scheme only causes blocks which have changed between successive frames to be written to the file, the information required to assemble a full frame can be distributed over a wide region of the file. In the worst case, the information would be distributed over the entire range between the beginning of the file and the current file position. This makes a seemingly trivial function such as “step-reverse” difficult to implement. Given a format in which entire frames are written, such as the SGI MV format, stepping to the previous frame is trivial -- find the previous frame and draw it to the screen. However, given my data stream, to correctly step to the previous frame means finding and drawing the most recent occurrence of each block of the last frame drawn. For example, if the last frame drew just two blocks (say, from coordinates (0,0) and (0,4)), one would need to search backwards for the most recent occurrence of blocks from (0,0) and (0,4). In the best case, they would have been part of the previous frame, and would be quickly found; in the worst case, these blocks would not be found until the initial frame of the stream. This problem could be solved (as in MPEG) by writing full frames (all blocks of a frame) at regular intervals (say, every 50th frame), which would bound the distance one would need to search to reassemble the state of the previous frame. The current implementation does not support “true” reverse functionality in its VCR interface. It simply redraws the blocks of the previous frame, which occassionaly results in ugly artifacts.

Currently, there is little support for the analysis of a recorded session. A session can be played back with (somewhat crippled) VCR style playback, but I have not developed any tools for editing, annotating, or logging sessions. Editing tools should allow the researcher to extract important or revealing segments and possibly reassemble them into another file. Annotation tools would allow the researcher to insert or attach commentary or notes to the stream while they played it back. Logging tools would support the development of content logs (fine-grained textual descriptions of the activity observed). Ideally these content logs would by hypertextual, meaning that links to media clips from the session could be inserted into the content log, providing a rich record of the users activities. Finally, speech indexing technologies could aid researchers in processing long sessions. I believe that there is a great deal of interesting and important work to be done in the area of analysis support before this technology becomes a viable, value-added alternative to the current methods and media.

6.3.3 Improving the Stream

To begin work on the interesting issues described above, some modifications must be made to the stream. First, a mechanism for implementing true VCR style playback such as the one discussed above would need to be added. Second, the stream definition should be modified to allow for more dynamic sessions. By this I am referring to the fact that currently, the parameters of the stream (how many tracks, frame sizes, color modes, etc.) are specified in the header of the stream and cannot be changed once data is written to the stream. Ideally it should allow for new sources to enter and exit at arbitrary points in its lifetime. The restriction to two image tracks and one audio track should be removed. One could also imagine the addition of new types of stream data blocks, such as text blocks, which could be used for annotations. Another powerful new type of data block would be input events. This would enable the integration of the fourth recording media - input events - discussed at the beginning of the paper.

6.4 Conclusion

A final, important question remains: Is the system usable by people who actually perform usability studies or contextual inquiries? Several students from a Human Computer Interaction course used the record and display tool during an observation exercise, and gave me valuable feedback. The capture tool appeared to be very unobtrusive; it was started at the beginning of the session and operated flawlessly through the course of several 30-40 minute sessions. It did not unduly increase the load on the machine, so as to make it difficult to get work accomplished. The greatest complaints were directed towards the display tool. Because the files were saved in compressed form, the playback could not be reversed. Also, the sessions were very long, and it appeared that the tool's fast forward functionality was not fast enough to rapidly scan through the recording. Furthermore, the users commented that they would have liked visible time signatures and random access abilities during playback.

Perhaps the most profound criticism, however, comes from my own realization of the irony of my own design process. In attempting to build tools which would foster improved practices in human-computer interaction design, I violated many of the basic principles of those practices. While I did extensive research on the state of the art in observation techniques and media, I did not involve users from the beginning of the process. I did not perform ethnographic studies of ethnographers. I did not invite ethnographers to participate in the design of the system. Personally, my experience was an example of how *not* to do design. By the end of the project I realized that my work was exemplar of the style of design I hoped to change.

Not all is lost, however. I believe that the absence of user involvement and the flaws in my design process, do not make the project a loss, because a great deal of low-level, technical issues needed to be explored and resolved to provide a basis for the development of powerful tools. In other words, I do not feel that my criticisms demolish the technical accomplishments I have made. The stream and abstractions I have implemented provide a robust, efficient, and flexible foundation for computer supported observation of computer users.

Appendix

This page shows screen snapshots of some of the tools. FIGURE 1 shows the Tcl/Tk interface to the capture program. FIGURE 2 and FIGURE 3 show examples of the Video Display window. FIGURE 4 shows the Tcl/Tk interface to the playback tool.

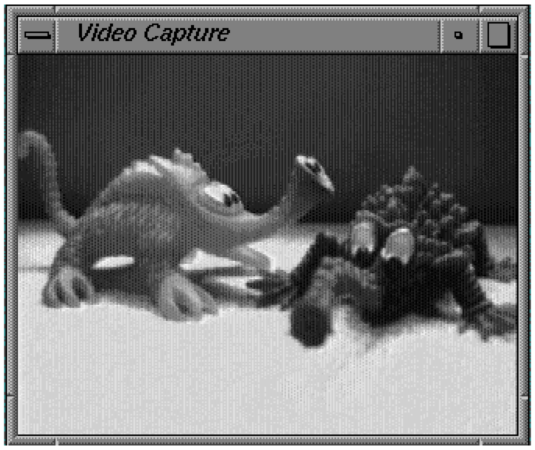


FIGURE 2

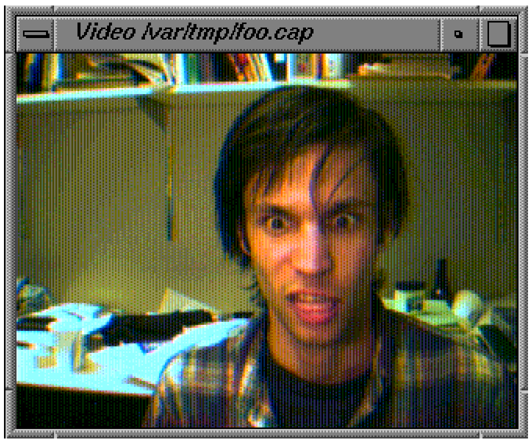
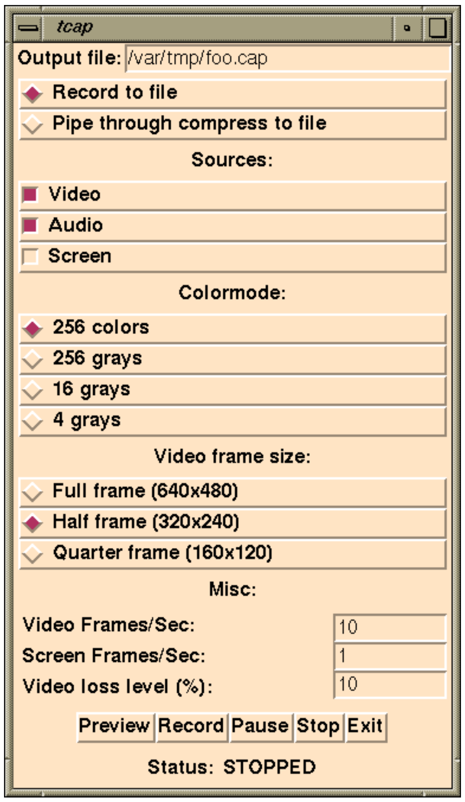


FIGURE 3



Output file: /var/tmp/foo.cap

[✓ Record to file]
Pipe through compress to file

Sources:

[✓ Video]
[✓ Audio]
Screen

Colormode:

[✓ 256 colors]
256 grays
16 grays
4 grays

Video frame size:

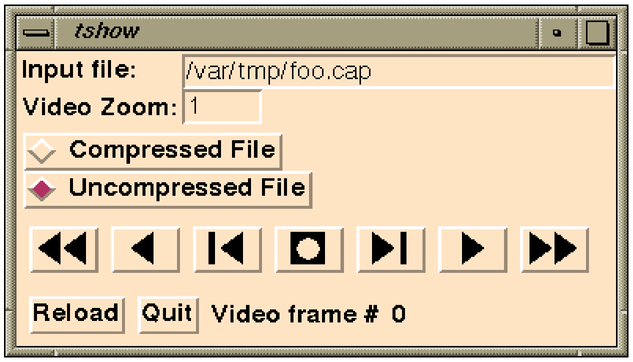
Full frame (640x480)
[✓ Half frame (320x240)]
Quarter frame (160x120)

Misc:

Video Frames/Sec: 10
Screen Frames/Sec: 1
Video loss level (%): 10

Status: STOPPED

FIGURE 1



Input file: /var/tmp/foo.cap
Video Zoom: 1

Compressed File
[✓ Uncompressed File]

Video frame # 0

FIGURE 4

References

[AW] Paul Adler and Terry Winograd. *Usability: Turning Technologies into Tools.* Oxford University Press, 1992.

[Blom] Jean Blomberg. Ethnographic Field Methods and their Relation to Design. *Participatory Design: Principles and Practices.* L. Erlbaum Associates, 1993.

[CMN] Stuart Card, Thomas Moran, and Alan Newell. *The Psychology of Human Computer Interaction.* L. Erlbaum Associates, 1983.

[Ehn] Pelle Ehn. *Work-oriented Design of Computer Artifacts*. Stockholm, Arbetslivcentrum, 1989.

[GK] Joan Greenbaum and Morten Kyng, eds. *Design at Work: Cooperative Design of Computer Systems.* L. Erlbaum Associates, 1991.

[Simon] Herbert Simon. *Sciences of the Artificial.* MIT Press, 1981.

[SN] Douglas Shuler and Aki Namioka, eds. *Participatory Design: Principles and Practices.* L. Erlbaum Associates, 1993.

[Welch] Terry Welch. A Technique for High Performance Data Compression. *IEEE Computer 17 (6)*. pp. 8-19, June, 1984.

[WF] Terry Winograd and Fernando Flores. *Understanding Computers and Cognition*. Addison-Wesley, 1987.