Transmitting Graphics Images Over a Local Area Network

Timothy A. Davis, W. Timothy Strayer,
Alfred C. Weaver

Computer Science Report No. TR-90-31
November 14, 1990

Transmitting Graphics Images Over a Local Area Network

Timothy A. Davis
W. Timothy Strayer
Alfred C. Weaver

Computer Networks Laboratory
Department of Computer Science
Thornton Hall
University of Virginia
Charlottesville, Virginia

Abstract

Due to the proliferation of computer networks in the professional world and the subsequent pursuit of new uses for them, the task of implementing applications requiring graphics transmission naturally arises. This task is unique among other networking problems due to the nature of graphics processing itself. Graphics applications require manipulation of a large amount of pixel data which must be displayed in a timely manner and be free from errors, as any slight deficiency is immediately recognizable.

Several application programs were written in C to demonstrate the usefulness of graphics transmission in the medical field. Examples include a generalized full-screen image transmitter for real digitized images (such as X-rays) and an animated EEG display. Though specific in terms of application, these demonstration programs illustrate some general principles in graphics transmission which demand consideration.

1. Introduction

Computer networks have revolutionized many different areas of the professional world due to their high data transfer rate and overall versatility. For example, commercial banking has undergone substantial changes in the last decade due to the networking technology employed in automatic teller machines. Similarly, graphics processing is also experiencing widespread use due to the large number of applications, such as microscopic imaging. As graphics applications grow in acceptance and use, the demand naturally arises for the ability to transport these images quickly and effectively between the end users.

In the business world the ability to transfer images such as charts, graphs, CAD displays, and floor plans instantly from one personal computer to another is quite valuable, especially for presentations and collaborative efforts. Graphics transmission in numerous medical fields also hold potential as medical practices utilize computers on a larger scale. Through the transmission of static images (e.g., X-rays) and animated sequences (e.g, a beating heart, EEG readings, etc.), graphics transmission free doctors and other staff from hardcopy delivery. The best technique for achieving these and other tasks of graphics transmission, however, requires further investigation.

This paper discusses solutions and suggestions for various image transmission problems. First, we introduce some special characteristics of graphics processing which generate complexities for image transmission. We then suggest some general strategies and techniques for transmitting images and discuss the advantages and disadvantages of each. Finally, we present two implemented medical applications, a single-frame X-ray transmitter and an animated EEG graph, which illustrate several general concerns associated with the transmission of graphics images over a network.

afford to have probabilistic errors since the erroneous data will be replaced within a short amount of time. This can reduce the amount of protocol processing time spent ensuring reliability. However, if the screen is transmitted without locality information, where a piece of data goes in the screen buffer may be dependent on in-order delivery. Distortion may occur if a MAC frame is lost and not recovered. Finally, there are some graphics transmission, particularly in medical applications, where absolute reliability if unquestionability needed.

Providing generality has long been recognized as an anthesis of providing performance. If screen formats and buffer methods are different between communicating workstations, there must be some translation from one format to the other. Typically this is not a concern as installations would opt for similar or identical equipment. Yet, as the need for performance increases, the amount of standardized, "off-the-shelf" software that one can use to achieve that performance decreases. Algorithms are developed which are optimized for the application, and implementation-specific techniques for manipulating the data are employed. As long as the system is self-contained, this application-specific techniques work well. Expandability, portability, flexibility, and adaptability all suffer.

3. Strategies

We implemented a variety of strategies for the sending and receiving graphics transmissions using reasonably powerful 80386-based personal computers over a 10 Mbit/sec token ring LAN. Since each strategy involved a tradeoff between generality and performance, multiple techniques were developed using varying degrees of each. Strategies for transmitting static images were evaluated separately from those involving animated sequences since each differ in characteristics and constraints.

areas of the screen in programmer-defined buffers (whose format is unknown) and redisplay these images in other screen areas. The transmission algorithms are straightforward: the sender copies part of the screen in a buffer and transmits the buffer to the receiver, which displays that buffer on its screen.

Although this method may be the easiest to implement, it suffers from two basic limitations. First, the built-in `getimage()` function manipulates the image data in an unknown format, and thus the matching `putimage()` call is required. This implies that all transmitter/receiver pairs must use these built-in image functions. Second, though the graphics functions in Turbo C are powerful, the programs using these functions are not particularly fast. A medium resolution image required almost 1.6 seconds for transfer (see Table 1).

Table 1 also shows that most of the performance time in our experiments is consumed by reading from and writing to the video RAM. The network can process data at a rate much faster than can be handled.

		Time (in seconds)	
Resolution	Total Delay	RAM Access	Network
640x200	0.93	0.80	0.13
640x350	1.59	1.37	0.22
640x480	2.14	1.84	0.30

Table 1
End-to-end Screen Tranfer Time for Turbo C Routines

In an effort to increase speed another method was developed involving direct access to the data in the video memory. The sending algorithm simply addresses the binary video data in the video memory and sends it across the network, where the receiver loads the bits directly into its video memory. Various complexities surface, however, when the programmer uses a standard version of C and is forced to deal directly with the registers on the VGA card to perform graphics functions. For example, Turbo C provides a useful function, `initgraph()`, which with it. DOS, however, is not inherently suited for multitasking; therefore, we had to write our own multitasking code. When the transmitting program begins, it creates a new handler for the PC clock interrupt and then invokes a system call which begins the digitizing software. Once an image in graphics mode appears on the screen, the new interrupt allows the graphics sending procedure to run every clock tick. During this procedure, the sender's screen display is temporarily halted while VGA information (such as mode, color table, etc.) and portions of the graphics image are transferred. No recognizable delay results in the sender's display since the switching between display and transmitting functions occurs quickly and frequently. Once the image has been fully displayed at the transmitter, the receiver's image is almost complete as well since the time required to transmit and redisplay is effectively shared between the sender's original image and the receiver's copy. Additionally, note that any image, X-ray or other, can be captured on the screen, transmitted, and redisplayed due to the generality of the technique.

Examples of X-rays captured and digitized with an overhead projector as back lighting and resolution of 320x200 are shown in Figures 1 and 2. A lower resolution than those previously mentioned was used to increase the number of shadings available in the gray scale, thereby providing a more realistic final image. Because of this lower resolution, the amount of time to transmit the image data and display it on both the sender's and receiver's screens requires less than 1 second.

5. Animated Sequences

Unlike the transmission of single frame images, animated sequences must be transmitted in real time to produce the desired effect. Two basic approaches to animation transmission naturally arise: transmitting full-screen images for each frame and transmitting localized areas of change for each frame. While the former method is more general, it does not perform adequately. Conversely, the latter reduces performance time, but does not allow for highly overall transmission time.

A compromise can be reached depending on the size and resolution of the image. Additionally, the decision to make the shape of the divisions vertical bars, horizontal bars, rectangles or square boxes depends on the nature of the animated sequence. For example, we found square boxes containing 400 to 500 pixels each to be a reasonable selection in experiments we performed on a small ball moving across many areas of the screen. Once we have chosen size and shape, how do we choose which pieces to send?

Perhaps the simplest and most time-efficient approach is to require the process controlling the screen image to notify the sending algorithm of the locations of recent updates. On the low level, the sender could get this information from the graphics package controlling the current working environment in which the animation is running. Each time part of the video RAM is updated, the location of the changed region would be reported by the graphics package to the sending program. Such a scheme, however, requires a common interface between the graphics package and application program, which is not readily available in most graphics packages. On the higher level, the sending program could require the program code producing the animation to relay to it locational data for image changes each frame. These "hints" would greatly aid the sender in isolating and minimizing the area of the image which has been updated; however, another burden would be placed on the programmer for each animation program he writes.

Another approach which is less efficient but more general requires storage of the entire image each frame. At given time intervals the sections comprising the current image are compared with those stored from the last image. Only sections or pieces which have been changed within the last frame are transmitted and updated on the receiver's display. The obvious drawback here is the amount of time such a method requires. On the other hand, this consideration should be diminished over time as newer and faster graphics processors are

6. Conclusions

Throughout our experiments in graphics transmissions we have focused on the tradeoffs between image reliability, algorithm complexity, software portability, and overall transmission and display speed. Of these, speed is the crucial factor and is countered by the other three. A high degree of reliability serves to slow transmission due to acknowledgment processing involved, which requires timeouts and the transmission of additional packets. Generalized algorithms also inhibit speed due to their intrinsic nature of including extra processing to ensure coverage of all possible cases. Furthermore, highly portable software may also include additional processing and adversely affect speed since it can make no assumptions about existing hardware and must therefore make provisions for many types. Depending on the application, however, we may be willing to make sacrifices in any of these areas to achieve gains in others.

For cases of single-frame image transmissions, speed may not be a primary issue since no real-time constraints are involved. In our experiments using low-level graphics BIOS and assembly language programs, we achieved relatively fast transmission and display times, but we paid the cost in tedious programming and a loss in generality. Conversely, our Turbo C programs were slow but much more general and easier to write. Turbo C, therefore, may be more practical to use in some cases of single-frame transmission. As for portability, both low-level and Turbo C programming are average: low-level programming is portable only across hardware of the same type, while Turbo C programs are portable only across Turbo C software. Choice of programming method, therefore, may depend on existing hardware and software for the given application. With regard to reliability, any degree of it will probably adversely affect speed due to the processing of acknowledgments associated with the large amount of pixel data which must be transmitted for each image. Because of these characteristics, general-purpose protocols, such as TCP-IP, are most practical for single image transmission applications.

7. References

BORL88 Borland International. *Turbo C Reference Guide, Turbo C User's Manual*, 1989.

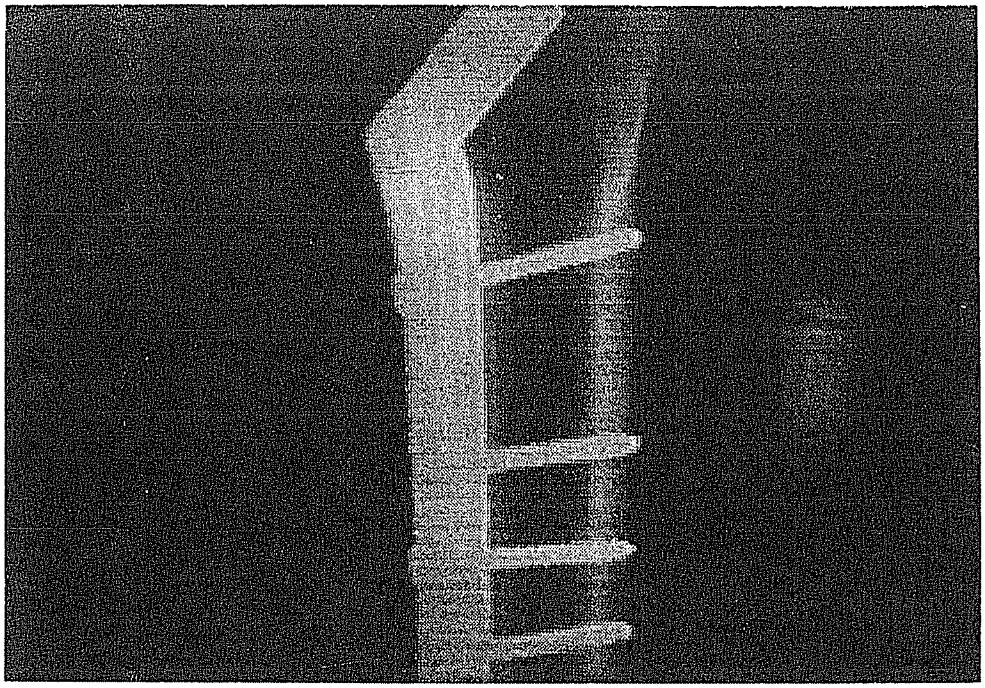
CORE89 CORECO. Image Processing Product and Support, VGTIZE 2.04, 1989.

LEWE85 Lewell, John. *Computer Graphics*, Orbis Publishing Limited, London, p. 91, 1985.

STRA88a Strayer, W. and A. Weaver. "Performance Measurements of Data Transfer Services in MAP", IEEE Network, Vol. 2, No. 3, pp. 75-81, May, 1988.

STRA88b Strayer, W., M. Mitchell, and A. Weaver. "ISO Protocol Performance Measurements", Proceedings of the ISMM International Symposium on Mini and Microcomputers, Miami Beach, FL, pp. 263-265, December 14-16, 1988.

WEAV87 Weaver, A. and M. Colvin. "A Real-time Messaging System for Token Ring Networks", *Software — Practice and Experience* 17 (12), December, 1987.



Figurc 1
A digitized X-ray showing a steel pin in the femur.

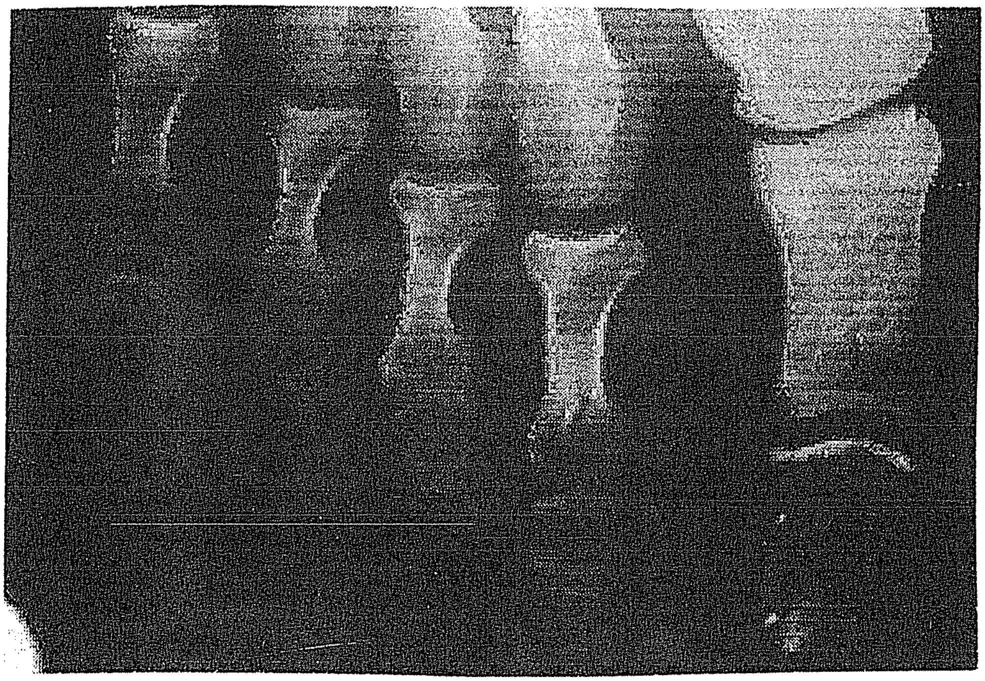


Figure 2
A digitized X-ray showing part of the foot.

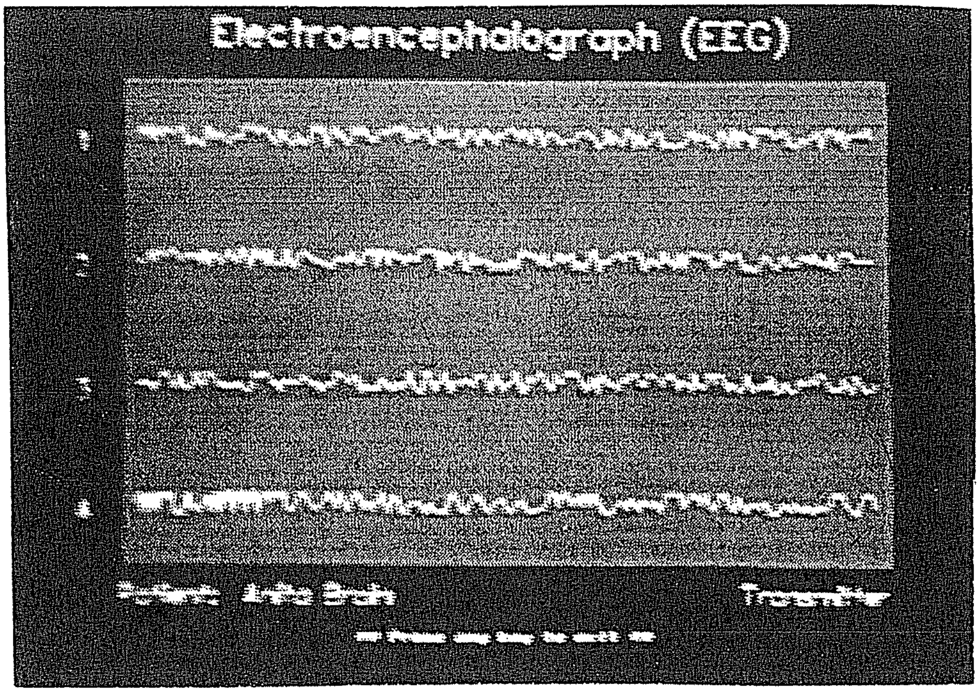


Figure 3
A simulated electroencephalograph.