LEGIBILITY NOTICE

A major purpose of the Technical Information Center is to provide the broadest dissemination possible of information contained in DOE's Research and Development Reports to business, industry, the academic community, and federal, state and local governments.

Although a small portion of this report is not reproducible, it is being made available to expedite the availability of information on the research discussed herein.

CONF-881118--1

LA-UR-88-890

Los Alamos National Laboratory is operated by the University of California for the United States Department of Energy under contract W-7405-ENG-36

LA-UR--88-890

DE88 007834

TITLE **DESIGN AND IMPLEMENTATION OF A SUPERCOMPUTER FRAME BUFFER SYSTEM**

AUTHOR(S) John D. Fowler, Jr., X-7
Michael McGowen, C-5

SUBMITTED TO Supercomputing '88 Conference, Orlando, Florida

DISCLAIMER

Los Alamos Los Alamos National Laboratory
Los Alamos, New Mexico 87545

Design and Implementation of a Supercomputer Frame Buffer System

by

John D. Fowler, Jr., Computational Physics Group

and

Michael McGowen, Computer Network Engineering Group

Los Alamos National Laboratory

ABSTRACT

A 512 by 512 pixel by 8 bits per pixel frame buffer has been designed, constructed, and installed on a 48 Mbit/s I/O channel of a Cray X-MP 4/16 supercomputer. This project was undertaken to test whether such a system would be useful and, if so, how it would be used. Supporting software provides the ability to convert vector graphics description files into raster format, to show raster movies interactively, and to show vector files by real-time conversion from vector to raster formats. We have shown that real-time animations in an interactive supercomputer environment are feasible and useful with this system.

Keywords: Supercomputer, Graphics, Frame buffer, Rasterization, Animation, User Interface, Scientific Visualization

Equipment needed: Overhead projector, VHS video player w/color monitor

I. Introduction

Frame buffers came into use in the mid-seventies as an integral part of raster display systems.[1][2] The frame buffer, also called refresh buffer, is the link between the raster processor and video hardware. It is a memory array that holds a digital representation of the current picture to be displayed on the viewing device. Typically, this display memory corresponds directly to pixels on the screen, with the requisite number of bits per pixel stored sequentially in display memory or in separate physical planes, one plane for each bit per pixel. The key to frame buffer viability is fast, cheap memory, and a relatively fast bus for accessing it.

In practice, frame buffers, being near the end of the graphics chain, have been located in the end-user graphics terminal or workstation, far away from the main computer which was the source of the graphical data. With the introduction and increasing popularity of local area networks and graphics workstations, this distance from the source has both decreased and lengthened: the former for calculations for which the workstation has sufficient power, and the latter for applications which still require supercomputer power, but the results of which are viewed on the user's workstation.

For supercomputer applications we have attempted the approach of putting the frame buffer as close as possible to the application itself, for a number of reasons:

1. We are interested in real-time user control of data while his program is running on the supercomputer. The closer the graphics gets to the main application, the easier it is to build a

system to do this in a highly interactive fashion. The data are already there, so putting the graphics on the supercomputer eliminates the burden of shipping huge amounts of data through the network.

2. Suitable bandwidths between supercomputers and frame buffers are becoming available. All Cray supercomputers have a 48 Mbit/s channel, and X-MP models allow the use of an 800 Mbit/s HSX channel. Updating a 512 by 512 pixel by eight bits per pixel frame buffer at 24 frames per second requires a channel bandwidth of 50 Mbit/s. 1000 by 1000 by 24 bits at the same rate requires 525 Mbit/s.

3. Given the existence of a supercomputer with an available channel and interface, such a system is cost-effective. The only hardware expense is for the frame buffer itself.

4. Our users show a certain reluctance to learn a new system of interaction, as would be required with a workstation. They are already familiar with the operating system on the supercomputer, Cray Time Sharing System (CTSS), and appear willing to learn a few simple utilities on this system in order to get the graphics to the frame buffer.

5. As higher speed channels and switchers become available, along with more and faster CPUs per mainframe, the simple concept of frame buffers becomes supportable for many users simultaneously at many locations from a given supercomputer.

In order to accomplish our objective, we had to answer satisfactorily several questions:

1. Can a time-shared machine in a demanding, multi-user environment support real-time graphics? There are at least two

issues here: a) Will the operating system allow enough sequential time-slots to show continuous motion? b) Can graphics algorithms, which are mainly integer and sequential in nature, be programmed to run efficiently on a supercomputer optimized for vectorized floating-point operations?

2. Will the computing cost of such graphics swamp the system and/or bankrupt the user?

3. Will users have a use for such a system if it can be developed?

4. What are the required spatial and color resolutions for typical applications? In particular, what are the tradeoffs between resolution and frame rate? How well does temporal antialiasing overcome the effects of jaggies in animations?

5. What sort of real-time control do users need? Is interaction over a serial network using a dumb terminal sufficient?

In order to answer these questions, we built and wrote supporting software for a simple 512 by 512 by eight bits per pixel frame buffer and attached it to a 48 Mbit/s Cray channel.

II. The User Environment

The user environment at Los Alamos National Laboratory includes about 30 Cray-1 equivalents, in the form of Cray-1s and Cray X-MPs, tied to a common file system and a common user interface through keyboard control concentrators (kcc's). The network allows serial communications at up to 38 kbaud from the supercomputer to the user, and a few hundred baud in the other direction. In addition, we have a number of Tektronix 4125 terminals supported through the Tektronix DMA interface at effective rates of about 225 kbit/s. Most of our serious users have

this capability, and this is the standard by which we are judged. Additionally, a growing number of local area networks and workstations (mostly Sun 3's) is supported by Ethernet communication links at peak sustained rates of about 100 kbit/s.

The supercomputers run the Cray Time-Sharing System (CTSS) in a highly interactive user environment. Most of our users' graphics are created by programs running on the Crays. Some of the graphical output is intended to be viewed interactively as the data are being created, but most of the graphics is written to files or drawn on film for later viewing.

Most of the graphics consists of vector descriptions containing frames with 5,000 to 50,000 vectors per frame, but we see an increasing number of raster images, mainly solid modeler-based frames and animations from calculations that lend themselves to raster display, such as fluid flow instabilities.[3] The advent of a significant amount of 3-dimensional calculations is starting to have an impact on our graphics systems, due to the increased amounts of data in 3-D descriptions and the need to manipulate (rotate) and illuminate images in order to perceive them properly. Nevertheless, our current environment is predominantly 2-D vector in nature.

III. Hardware

In order to demonstrate that our approach was viable in a larger context than direct Cray connections, the frame buffer was designed to connect to a High Speed Parallel Interface (HSPI) device, which is our standard network connection. Each host in the Central Computing Facility has such a device, which converts its unique I/O interface to a common format. The connection to the

HSPI was done to demonstrate two things: first, that full access to all the bandwidth available in existing equipment would enable extreme simplification of the hardware architecture, which in turn allows enormous gains over current solution methods; and second, that the hardware architecture used in image capture systems could be employed in network processor design, enabling the devices to sustain full bandwidth transfers of real-time digital video across a network. This second issue is detailed in another publication.[4]

The HSPI definition was developed when Los Alamos installed its first Cray supercomputer and is similar to the I/O channel definitions of the Cray (the DN/DO and DA/DB channels).

These channels share a fundamental feature: They transmit exactly that data which is of interest, with no command structure or format specification imbedded in the data stream. We use this feature to make the frame buffer system simple and efficient. The host maintains an array in memory which represents the image to be displayed. When a frame is complete, the whole array is output to the frame buffer. By continuously outputting data, the display achieves animation.

The frame buffer display is organized as an array of 512 by 512 pixels. This corresponds to the memory organization of the device, which is an array of 512 by 512 bytes. Each byte is an index to one of 256 colors from a palette of 262,144. The Cray channels run at 48 Mbits per second, which allows the transmission of 24 complete image frames per second.

The frame buffer consists of four main sections: The HSPI front-end, video refresh memory, a micro-program engine, and a

video interface. Figure III-1 contains a block diagram of the frame buffer. Its architecture addresses the primary function, which is the movement of data into and out of memory.

HSPI Front End

The HSPI front end manages the flow control on input from the HSPI channel and notifies the micro-program engine when the input shift register buffers are full. The front end is also responsible for delineating the data in a fashion that is acceptable by the memory array. The memory buffers provide a depth of 256 words that can be strobed in without any other action being taken. The front end counts 256 strobes and asserts a signal to the micro-controller notifying it that the buffer is full. This memory is double buffered so that input can immediately proceed with data being directed to a second buffer while the first is being processed. The HSPI channel also contains a RESET line, asserted by the transmitter, which is used to signal an abnormal termination or to provide a resynchronizing event. On RESET, the frame buffer reads the color table, as described in Part IV.

Video Refresh Memory

Video refresh memory consists of 64k video DRAMs which provide an independent shift register for the input and output of sequential data. The shift registers are 256 locations deep. After they are full, the 256 data values can be moved in parallel into the memory array with one random access memory cycle. High-speed output is accomplished by loading the shift register with the 256 memory locations in parallel and then sequentially shifting the data out. Display memory is single-buffered. Two separate and independent shift registers with separate clocks allow

reading and writing without sparkles of random data being displayed on the screen.

Video Interface

The video interface converts the sequential stream of digital data from refresh memory into analog red, green and blue (RGB) signals which drive the display monitor. The display is treated as a 1024 by 1024 image that is updated at 60 frames per second. Each pixel and each row are displayed twice to convert the 512 by 512 memory organization into display format. The video signal is RS-343A compatible.

Micro-Program Engine

The micro-program engine is responsible for maintaining the proper sequencing of events such as video timing and generation of addresses for the next destination of input or the source of the video output. All timing parameters are programmable so that adjustments can be made for different display formats. Input is a completely asynchronous event, yet the output timing must be synchronous and the same every time. The micro-program engine code coordinates all activities and in fact generates the control signals of every function. It is this micro-code that actually defines the behavior of the frame buffer.

Connection to Supercomputer

The frame buffer at Los Alamos is connected to a Cray X-MP 4/16 supercomputer having four processors, 16 million 64-bit words of memory, and a 512 million word solid-state storage device (SSD). A switch on the frame buffer output directs the video signals to either a nearby monitor or to a fiber-optic modem for transmission to a monitor in a user's office about 150 meters away. The picture

quality in the user's office is actually better than that from the local monitor because of the amount of radio-frequency interference in the central computing facility. Normally output from the frame buffer is directed to the user's office.

Users access the frame buffer software by logging onto the Cray X-MP 4/16 in the usual manner through our standard network, using any terminal.

IV. Software

The purpose for writing the software was dual: to find answers to the questions posed in Part I, and, if the answers were sufficiently encouraging, to provide a basic set of programs for our users. Therefore, we tried to make the programs useful both for testing the system and for showing graphics under realistic situations.

We wrote three programs and one utility. The software was written using the FORTRAN 77 compiler provided by Cray Research, Inc., CFT77. Although we have multitasking support on our machines, we have not used it. We have also used traditional graphics algorithms (See below for descriptions). We are confident that by programming in Cray Assembly Language, using multitasking, and/or using more advanced algorithms, we could substantially better the performance reported here.

Software support for the frame buffer consists of three programs and one library of subroutines. These programs are concerned with converting vector descriptions to raster and showing the results. The Common Graphics System (CGS), supported at Los Alamos, uses a vector graphics description embedded in a system-independent graphics metafile.[5] Consequently, our raster

programs were written to read graphics files created with this system.

Brief descriptions follow of the programs we have written for this project.

MR (Metafile-to-Raster)

The program MR accepts as input a CGS metafile and creates a raster movie file, with a minimum of user interaction. The program prompts the user first for the name of the metafile, then for the name of the raster file. The program then processes the metafile, frame-by-frame, into raster frames, placing each frame sequentially in the raster movie file. At the end of the program, the first frame of the movie file is created, containing the color lookup table definition and the number of frames in the file. This information is needed by VCR, the program that reads and displays the raster movie file.

The number of different colors in the CGS metafile determines the color lookup table, which is not known until the metafile has been processed. Each time a unique color is requested, it is added to the lookup table. If the maximum of 255 colors is reached, subsequent colors are drawn in white. The first color index, index zero, is reserved for the background color, usually black, and the next index is always defined as white. The color lookup table is packed into the rightmost word of each of the first 256 scan lines. The contents of these words are not displayed.

MR supports the following CGS metafile codes:[6] move, draw, draw marker, set rgb color, hardware text string, set aspect ratio, new page, and end of data.

VCR (Video Cassette Recorder)

This program reads raster movie files generated by MR or any other program that creates a raster file with the proper format, and sends the output to the frame buffer. Its name comes from its ability to emulate many of the functions of a video cassette recorder.

VCR lets the user interactively show a movie in forward or reverse directions, at normal, fast, or slow speeds, and pause on any given frame. The movie can also be shown as an endless loop.

PZIP

PZIP reads a CGS metafile, converting it to raster and showing it under interactive user control. It converts the same CGS opcodes and transforms at the same vector rates as the program MR, above. Interactively, it acts much the same as VCR, except for the fast and reverse commands, with additional commands described below.

PZIP lets the user zoom and/or pan while in single-frame (pause) mode or while in slow-motion or forward mode. At any time, the user can write raster frames from PZIP into a file for subsequent showing with VCR. Whatever the user sees on the screen goes into the raster file. One reason for writing a movie file from PZIP is that VCR is less CPU-intensive than PZIP and has additional commands.

FBLIB

FBLIB is a library that lets the CGS user create a CGS view surface which is the frame buffer. Any CGS program can initialize this view surface at graphics initialization time with a single subroutine call. Subsequently, CGS commands can go to any

CGS-supported device and/or to the frame buffer. This is accomplished by using the CGS GON and GOFF view-surface commands and by writing the frame-buffer driver as a so-called CGS "XX" driver.

FBLIB uses run-time memory management to run-time allocate the 262,144-word raster array. Thus, this library adds very little to the executing program's field length until FBINIT is actually called.

Data Structures

The important data structures used by the software reported here are:

the raster array

the packed raster array

the color index array

The Raster Array

The raster for the frame buffer contains 512 lines of 512 pixels, of which 475 lines of 495 pixels are viewable on the screen. Each pixel is defined by a six-bit color index.

A time-efficient but space-consuming method of retaining the raster data is to dedicate a 64-bit Cray word for each pixel, using a 512-by-512 word array (262,144 words) to hold the raster. This affords quick access by the program to any pixel, since each pixel is just a pair of indices into the array.

A more space-efficient approach would involve packing the raster into a smaller-sized array. A natural size would be a 32,768-word array containing the pixels packed into bytes, since this is the form of the packed raster array required by the hardware. This approach would require expending shifts, masking,

and logical operations to add a pixel to the array. This was tried and found to be too time consuming. The temporal overhead to such an approach varies with the complexity of the picture, but typically a 20 percent run-time overhead was imposed by this approach. Thus it was dropped in favor of the more space-consuming method, which requires vectorizable packing only at the end of each frame.

The Packed Raster Array

This array contains 32,768 64-bit words for a 512-by-512 pixel raster, allowing eight bits per pixel.

At the end of each frame, the raster array is packed into the packed raster array, an eight-into-one packing. Swapping of adjacent even and odd bytes is necessary to undo what the network does to the data between leaving the Cray and arriving at the frame buffer.

The Color Index Array

This array contains the color index to red, green, and blue intensity definitions. Each color has a range of 64 shades and takes six bits of data. Each index, starting with zero and going to 255, is packed into word 64*(index + 1) of the packed raster array. This information is read by the frame buffer only when a channel reset occurs, which is sent by VCR on the first frame, and by PZIP whenever a new index has been added since the previous frame.

Algorithms

MR, PZIP, and the library FBLIB use Bresenham's line algorithm[1] for vector-to-raster conversions and the Cohen-Sutherland clipping algorithm[1] for clipping.

Rasterization

Converting vector information to raster typically takes about half of the time in MR and PZIP. Because most of the vectors are short in the CGS metafiles we have used (typically, 90 to 99 percent of the vectors in a given file will be fewer than 10 pixels long), vectorizing approaches to rasterization such as the incremental slope method[1] have not been very successful. We have found that Bresenham's algorithm, although it is not vectorizable, converts typical frames faster.

Clipping

The Cohen-Sutherland clipping algorithm was chosen because it provides for trivial acceptance or rejection of most vectors without further testing, when vectors are short. The required testing and further clipping when necessary uses about ten percent of the time in PZIP.

Hardware Characters

We provide hardware character emulation by inserting five-by-seven bit-map representations of characters directly into the raster array. Characters at this size are larger than users expect, based on experience with their terminals, but they are legible. Most CGS metafiles contain at most only a few dozen hardware characters per frame. We have not attempted to evaluate the efficiency of this technique.

V. Results and Discussion

Because the showing of raster movies consists of simply reading a frame, writing a frame, and checking for user input, it places very little demand on the CPU resources of the supercomputer. Showing a movie from a vector description file, on

the other hand, can be quite CPU intensive. The operations for showing a vector frame include reading from the CGS metafile, unpacking and decoding op codes and data, performing the indicated operations (such as setting color index, hardware text, accumulating vectors and rasterizing), packing the output array, and sending it. Because of these differences, we consider the two processes separately.

Two other timing issues arise. We found using the SSD for file storage to be much more satisfactory than using a physical disk unit. In the timings below, we give results for files which were on a "real" disk and for files on the SSD. The SSD on our machine is large enough to hold about 15 minutes of raster information at one time. The other timing issue is the number of users competing for the four CPUs on a Cray X-MP 4/16. We considered three environments: Single-user, where we were the only ones on the machine, Moderate interactivity, involving about 15 users in the execution queue (ready for a CPU), and Heavy, with about 30 users in the execution queue.

In the tests that follow, our timings indicate throughput as measured by timing actions on the display with a stopwatch. Data obtained in this manner are more indicative of usefulness than are data obtained by calculating theoretical performance, by using system timings, or by timing individual loops in the code. The data we measured indicate throughput from reading a file to showing the image on a monitor.

Raster Movies

Showing raster movies with VCR uses about four percent of a single CPU, or about two and a half minutes of CPU time per

wallclock hour of movie. Because of this low rate of CPU usage and the presence of four CPUs on an X-MP 4/16, fairly continuous movie rates are perceived in all but the Heavy environment, where pauses of about a second occasionally occur. Table V-1 shows the frame rates we observed in our tests.

Environment	Disk	
	Real	SSD
Single-User	19.3	23.3
Moderate (15 Users)	13.1	20.2
Ratio (Single/Moderate)	1.47	1.15

Table V-1. Frame rates (frames/second) for raster movies shown with VCR under various circumstances.

We have found that in a heavy environment, the ratios shown above would approximately double.

For SSD files, the performance in a moderate environment is perceived to be quite satisfactory. Even in a heavy environment, useful animations can occur. The performance using a real disk is much more sporadic, depending on who else is using the disk unit on which the file resides. The average frame rates quoted above are far from uniform, and long pauses can occur in an interactive

environment using a real disk. Our measurements in the single-user environment indicate that the disks themselves are capable of supplying data fast enough for animations. We believe that if we had a dedicated physical disk unit, performance from it would be satisfactory even in an interactive environment. Operating system constraints prevent us from testing this hypothesis.

Vector Movies

Performance issues concerning vector movies are more complicated because of nonuniformity in frame sizes and in the information to be converted and displayed. Considerations such as the number and length distribution of vectors, number of color changes, and hardware characters complicate the issue.

We have tested the system for a range of vector files covering most of the spectrum our users typically encounter. The results for Single-user tests are shown in Fig. V-1. For a moderately loaded system, the data in Fig. V-1 will shift to the left by about 30 percent. For a heavily loaded system, the shift will be about 60 percent.

One would expect that the simplest predictor of performance might use an inverse relationship between number of vectors per frame and frame rate. The constant of proportionality is the vector conversion rate (vectors per second). The solid line in Fig. V-1 is such a curve with the constant equal to 170,000 vectors per second. This appears to fit our observations fairly well over a wide range of files. It is reasonable that, if the curve is normalized near the middle of the range, frames with few vectors would fall to the left of the curve due to the frame-constant overhead of packing arrays, sending data, and waiting for I/O

completion. For the same reason, large frames, which have low frame rates, should exceed the curve and have higher than average vector conversion rates.

All other considerations being equal, the average vector throughput should increase with frame size. This is true for the data in Fig. V-1, except for the file with the most data per frame. This file contained unusually large vectors and also a larger than average number of color changes.

Figure V-2 shows a typical frame from the file that had a frame rate of 17 frames per second in Fig. V-1. This file has about 7,500 vectors and a few dozen hardware characters per frame.

Qualitatively, users find the current performance in a moderate environment to be adequate over the range of frame sizes considered here. Obviously, performance improvements for frames with more than about 20,000 vectors would be welcomed. Many of the large-data frames are zoomed on, to examine details. When most of the vectors are removed by clipping, some performance is restored.

Resolution and Color Capability

Our observations on resolution are that 512 by 512 pixels is satisfactory in animations because of temporal antialiasing. Jaggies can be seen but are not obtrusive. For viewing still frames, this resolution is marginal, especially for users who are accustomed to seeing images at about twice this linear resolution. The ability to zoom and pan interactively to magnify fine detail in vector images partially alleviates this problem.

Most of our users' vector description files contain just a few colors, so limitations on color selection are not an issue. For purely raster images, however, 256 colors is often not enough.

Sometimes useful information can be conveyed within these color limitations, but realistic image rendering is difficult or impossible.

Cost of Use

Our user accounts are sufficient that no one has trouble paying for time to show raster movies with VCR. Running PZIP, on the other hand, can be costly. A wallclock minute with PZIP typically is almost a full CPU minute. Nevertheless, we do show frames from CGS metafiles about fifteen times more efficiently (in CPU utilization) than does CGS itself when sending the same frames to a Tektronix 4115 terminal. Users with limited accounts can limit their charges by running MR or PZIP only once to create a raster file for subsequent extended viewing with VCR.

VI. Conclusions

Based on over six months' experience using the system described here, we offer the following conclusions:

1. A Cray X-MP 4/16 running CTSS in moderate to heavy interactive environment can support real-time animation from raster graphics files. Animations using interactive zoom and pan from vector description files are useful, but the graphics algorithms and coding we currently use for rasterization and clipping need to be improved in order to make rendering of large vector frames more interactive and less costly.

2. The computing costs of such graphics are not beyond the means of most of our users, nor does the CPU load in driving a single frame buffer overtax the supercomputer. We believe that future supercomputers, with more processors and power, will be able to support many such users simultaneously.

3. Users will be quick to take advantage of the animation capabilities of the system described here, if it is located sufficiently nearby. We have found that most of our users work on the same hallway where the frame buffer's output monitor is located. Some users who generate raster animations and have no viewing alternative except film come from large distances (up to several miles) to use the frame buffer. The key to user support is getting the capability into their offices.

4. Spatial and color limitations in the current system restrict its use to animations of wire-frame images, line drawings, and raster images with fewer than 256 colors. Almost all of our users' graphics fall within these limits.

5. User interaction with the frame buffer system over a 9600 baud line is sufficient. Although network limitations prevent use of a mouse or other locator device, we find that interactive features such as zoom and pan can be accomplished by having the user initiate the action with a single keystroke, and having the action continue until another keystroke is issued. A more natural interface should be developed when the network allows it.

6. Although enthusiasm has been high for the current system, users would like higher resolution, support for eight bits of red, green, and blue, higher speed, and more interactivity. We also believe that three-dimensional polygon rendering support would be found useful by those involved in three-dimensional calculations.

We view this system as a step closer to the goal of scientific visualization, which is to provide the user with real-time visualization and control of his calculations.

Acknowledgements

The authors wish to thank Joseph Rieken, of Los Alamos National Laboratory, for providing the CTSS channel driver, Steve Poole, of the Center for Scientific Computer Visualization, Houston, TX, for supplying an efficient packing subroutine, and Digital Equipment Corporation for providing hardware support for the prototype frame buffer board.

References

1. Foley, J. D. and A. Van Dam, Fundamentals of Interactive Computer Graphics, Addison-Wesley, Reading, Mass., 1983.
2. Newman, W. M. and R. F. Sproull, Principles of Interactive Computer Graphics, McGraw-Hill, New York, 1973.
3. Winkler, K-H. A., J. W. Chalmers, S. W. Hodson, P. R. Woodward, and N. J. Zabusky, "A Numerical Laboratory," Physics Today 40(10), October 1987, pp. 28-37.
4. Winkler, K-H. A., S. W. Hodson, J. W. Chalmers, M. McGowen, and D. E. Tolmie, "Ultra-Speed Graphics for Computational Science," Cray Channels 9(2), Summer 1987, pp. 4-9.
5. Reed, T. N., "Experiences in the Design and Support of a Graphics Device Driver Interface," Eurographics'81 Proceedings, September 1981, pp. 281-289.
6. Reed, T. N., "A Metafile for Efficient Sequential and Random Display of Graphics," Computer Graphics 16(3), July 1982, pp. 39-43.

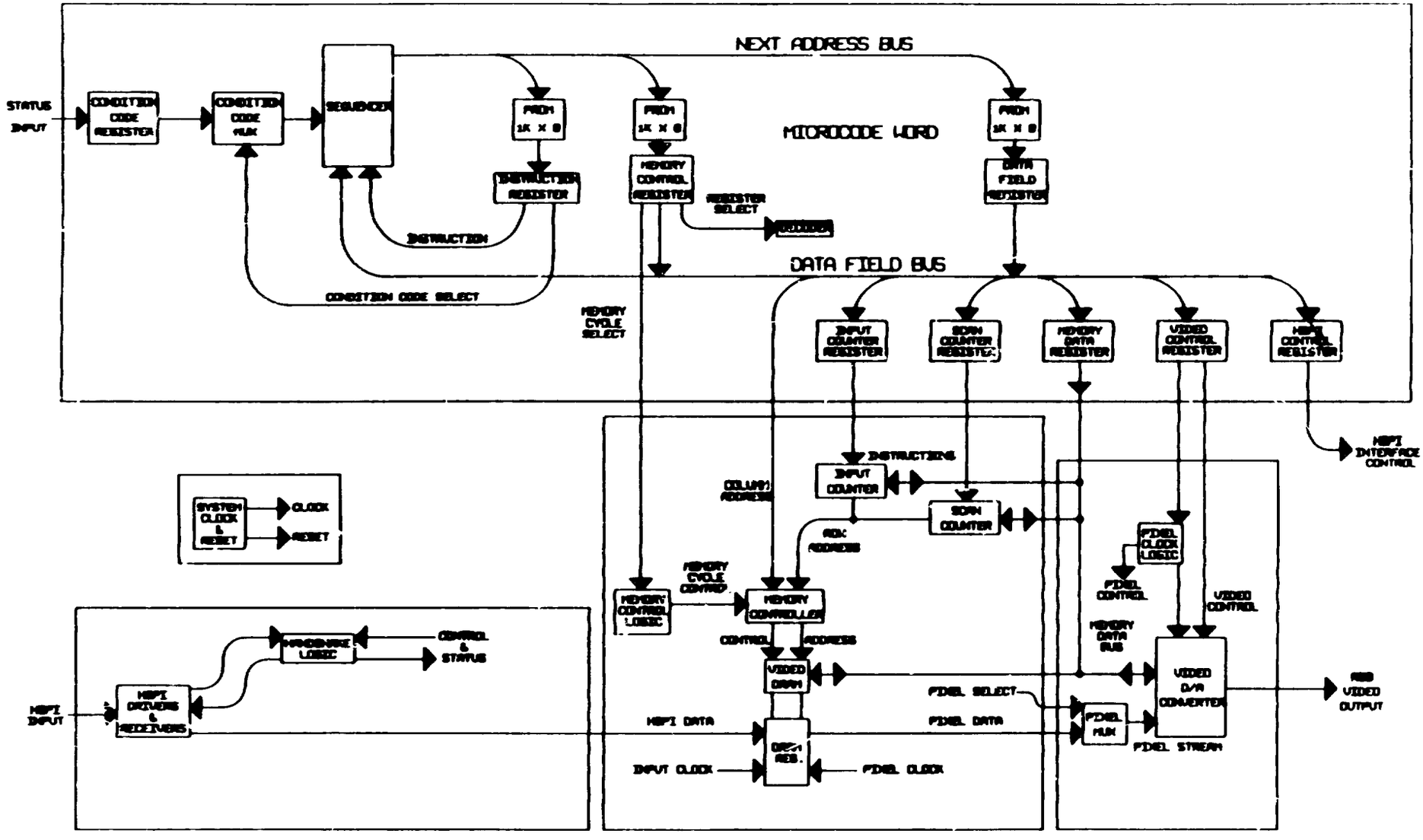

Fig. III-1. Block Diagram of Frame Buffer

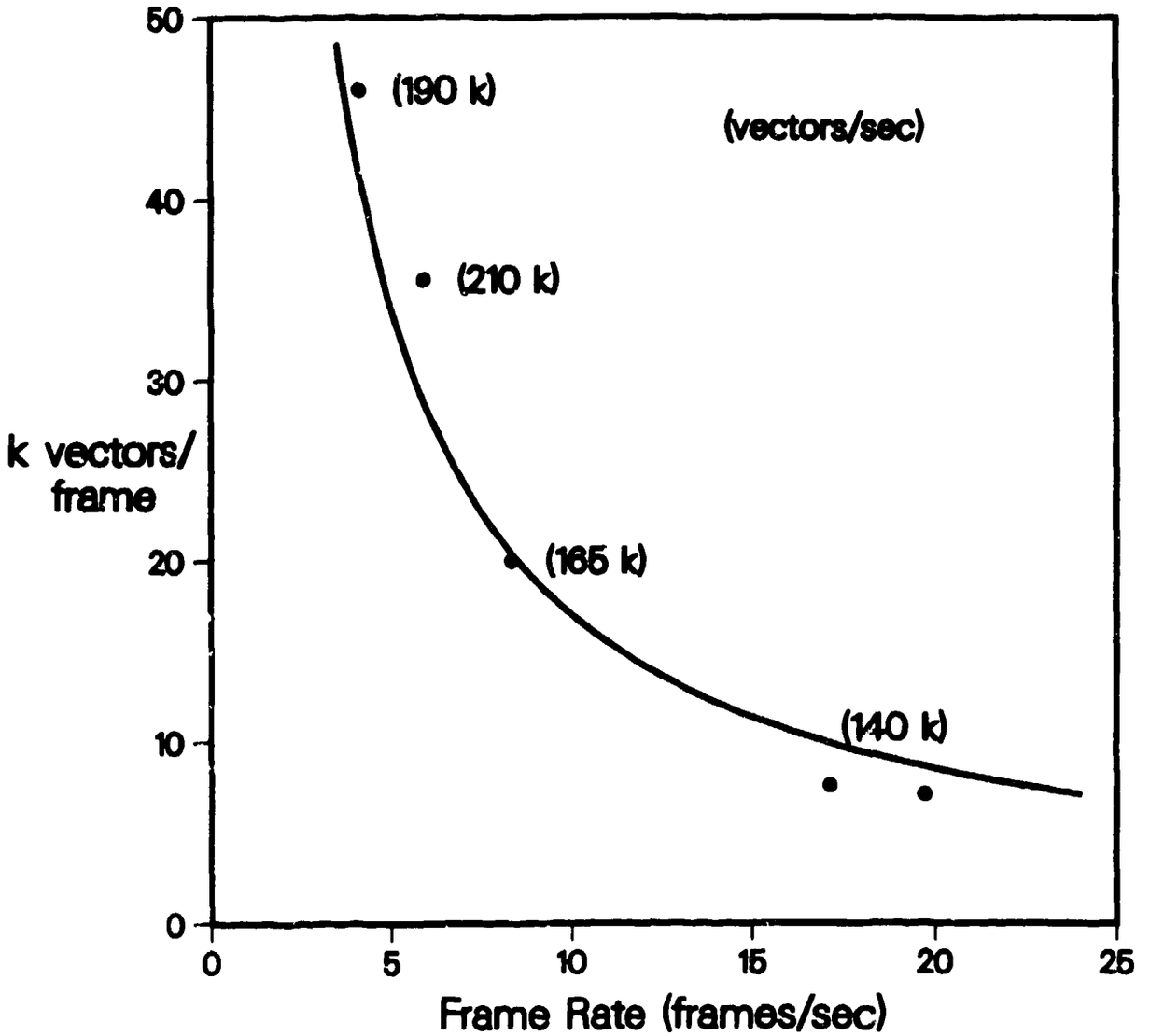

Fig. V-1. Frame rates for five CGS metafiles using PZIP in a single-user environment. Numbers in parentheses are average vector rates per frame for the indicated metafiles. Solid line represents the equation: vectors/frame = 170,000/(frames/sec). In a moderately interactive environment, frame rates would be reduced by about 30 percent.

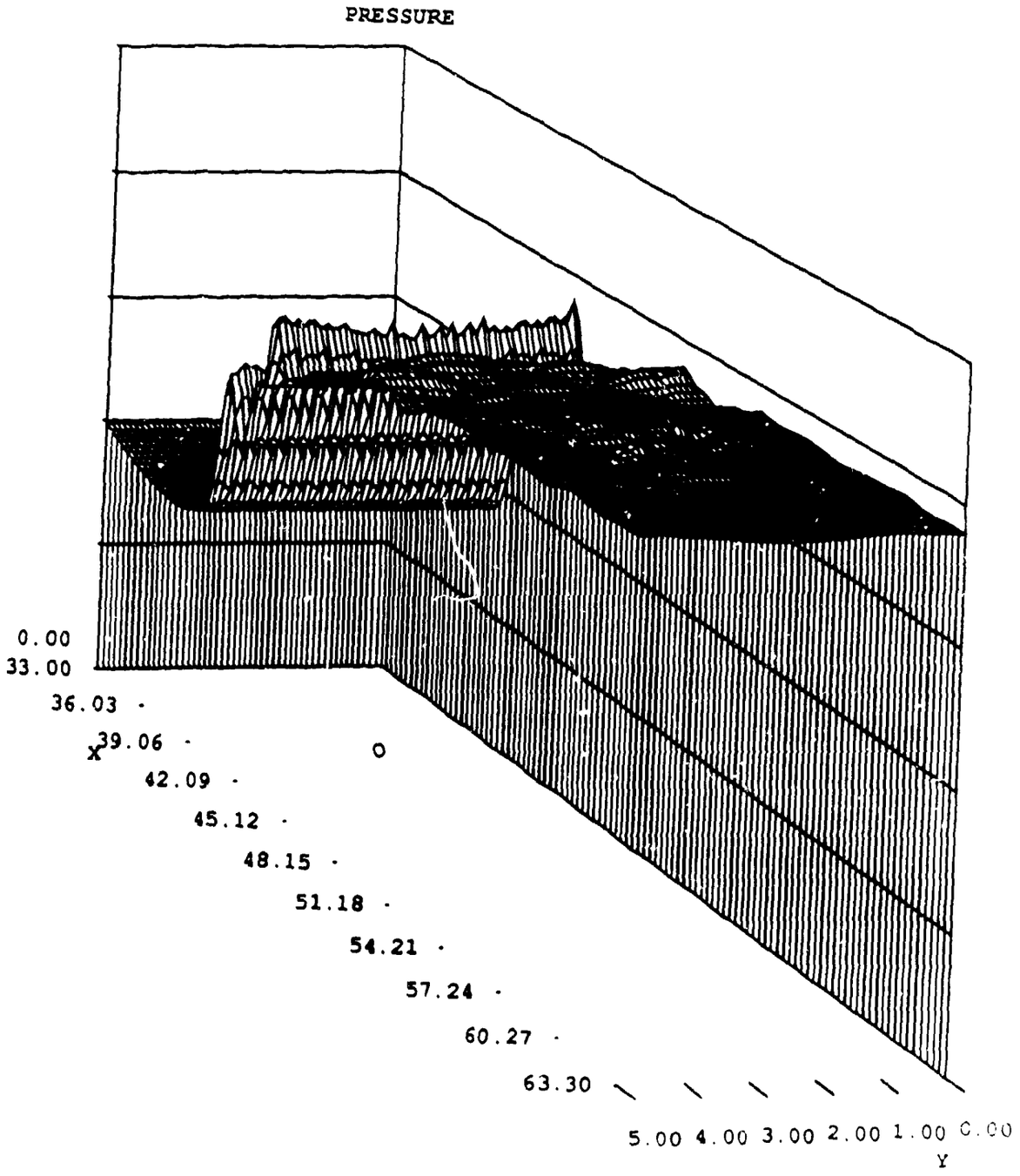

Fig. V-2. A frame from an animation sequence containing about 7,500 vectors and 80 hardware characters.